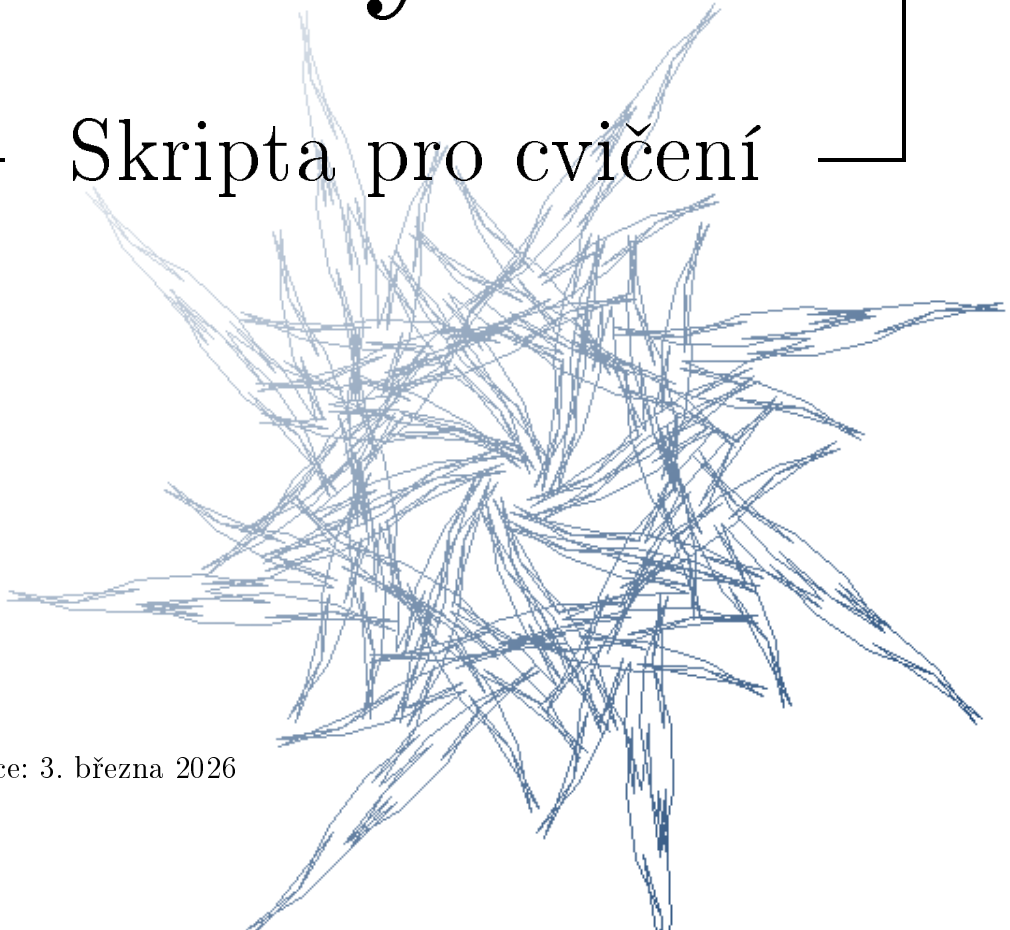
SLEZSKÁ
UNIVERZITA
FILOZOFICKO-
PŘÍRODOVĚDECKÁ
FAKULTA V OPAVĚ
ÚSTAV INFORMATIKY

Šárka Vavrečková

Operační systémy II

Skripta pro cvičení



Opava
Poslední aktualizace: 3. března 2026

Anotace: Tento dokument je určen pro studenty předmětu týkajícího se operačních systémů na Slezské univerzitě v Opavě (předmět Operační systémy II), v letním semestru. Navazujeme na předmět Operační systémy I, a tedy se předpokládá, že se čtenář orientuje v souborové struktuře systémů Windows a Linux, principech správy uživatelů, oprávnění, procesů, zvládá základy konfigurace v textovém režimu v těchto systémech.

I zde se zabýváme postupně operačními systémy Windows a Linux, konkrétně tématy:

- správa procesů, služeb a uživatelů,
- správa zařízení a sítě,
- nasazení systému, instalace, řešení problémů.

Na rozdíl od předchozího předmětu se budeme pohybovat především v textovém režimu.

Operační systémy II

Skripta pro cvičení

RNDr. Šárka Vavrečková, Ph.D.

Dostupné na: https://vavreckova.zam.slu.cz/opsys.html

Ústav informatiky
Filozoficko-přírodovědecká fakulta v Opavě
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Sázeno v systému LaTeX

Předmluva

Co najdeme v těchto skriptech

 *Rychlý náhled:* Tato skripta jsou určena pro studenty Ústavu informatiky Slezské univerzity v Opavě. Ve cvičeních předmětu Operační systémy II se v první části semestru probírají operační systémy z rodiny Windows a v druhé části semestru UNIXové systémy se zaměřením na Linux.

Navazujeme na obdobná skripta z předmětu Operační systémy I, tedy předpokládá orientace v souborové struktuře systémů Windows a Linux, principech správy uživatelů, oprávnění, procesů, u Windows i orientace v registru, základy konfigurace v textovém režimu v těchto systémech.

Skripta se mohou zdát značně rozsáhlá. Důvodem je zařazení mnoha ukázkových (řešených) příkladů a motivačních (neřešených) úkolů, které mají pomoci při pochopení a osvojení si učiva. U některých příkazů jsou uvedeny také jejich obvyklé výstupy, také z důvodu často nedostatečných přístupových oprávnění studentů v učebnách.

Značení

Ve skriptech se používají následující barevné ikony:

-  *Rychlý náhled* (skript, kapitoly), ve kterém se dozvíme, o čem to bude.
-  *Klíčová slova* kapitoly.
-  *Cíle studia* pro kapitolu nám řeknou, co nového se v dané kapitole naučíme.
-  Nové *pojmy*, značení apod. jsou označeny modrým symbolem, který vidíme zde vlevo. Tuto ikonu (stejně jako následující) najdeme na začátku odstavce, ve kterém je nový pojem zaváděn.
-  Konkrétní *postupy a nástroje* (příkazy, programy, soubory, skripty), způsoby řešení různých situací, do kterých se může administrátor dostat, syntaxe příkazů atd. jsou značeny také modrou ikonou.
-  Některé části textu jsou označeny fialovou ikonou, což znamená, že jde o *nepovinné úseky*, které nejsou probírány (většinou; studenti si je mohou podle zájmu vyžádat nebo sami prostudovat). Jejich účelem je dobrovolné rozšíření znalostí studentů o pokročilá témata, na která obvykle při výuce nezbývá moc času.

-  Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Nejčastěji u této ikony najdeme webové odkazy na stránky, kde se dané tématice jejich autoři věnují podrobněji.
-  Červená je ikona pro *upozornění* a poznámky.

Pokud je množství textu patřícího k určité ikoně větší, je celý blok ohraničen prostředím s ikonami na začátku i konci, například pro definování nového pojmu:

 **Definice**

V takovém prostředí definujeme pojem či vysvětlujeme sice relativně známý, ale komplexní pojem s více významy či vlastnostmi.

Podobně může vypadat prostředí pro delší postup nebo delší poznámku či více odkazů na další informace. Mohou být použita také jiná prostředí:

 **Příklad**

Takto vypadá prostředí s příkladem, obvykle nějakého postupu. Příklady jsou obvykle komentovány, aby byl jasný postup jejich řešení.

 **Úkol**

Otázky a úkoly, náměty na vyzkoušení, které se doporučuje při procvičování učiva provádět, jsou uzavřeny v tomto prostředí. Pokud je v prostředí více úkolů, jsou číslovány.

 Na konci každé kapitoly najdete odstavec se souhrnem kapitoly.

Jak probíhají testy

Zápočet z předmětu Operační systémy II lze získat absolvováním dvou testů – první se týká Windows, druhý Linuxu. Nejméně jeden z nich (podle vlastní volby) je nutno absolvovat osobně v učebně, zbývající může být online.

Online varianta testu obsahuje otázky typů umožněných informačním systémem (výběr možností, zatrhnutí odpovědí, doplnění slova apod.).

Varianta v učebně: dostanete klasické zadání na papíru, ale lze používat počítač, a to nápovědu a nástroje běžně dostupné ve standardní instalaci daného operačního systému, ale bez přístupu na Internet. Nejsou dovoleny dokumenty vlastní ani cizí výroby, které nejsou součástí standardní instalace, nelze používat internetový prohlížeč ani jiný způsob přístupu na externí zdroje informací.

Na stránkách předmětu je k dispozici orientační seznam otázek a úkolů, které se mohou objevit na testu, ovšem v testu se mohou objevit mírné odlišnosti (například v názvech zpracovávaných souborů či adresářů, jiné přepínače příkazů, apod.).

Obsah

Část I

Windows

Kapitola 1

Příkazový řádek a PowerShell

Rychlý náhled: V této kapitole si zopakujeme některá témata o textovém režimu ve Windows a navážeme dalšími tématy: naučíme se v textovém režimu vytvořit nový soubor, používat číselné proměnné, podíváme se na psaní skriptů, naučíme se používat složené příkazy. Zaměříme se také na moduly v PowerShellu a další pokročilá témata.

Klíčová slova: Příkazový řádek, PowerShell, regulární výraz, skript, dávkový soubor, složený příkaz, větvení, cyklus, hromadné zpracování dat, CMDLET, objekt, zprostředkovatel.

Cíle studia: Po prostudování této kapitoly si zopakujete práci v textovém režimu ve Windows a dále se naučíte vytvářet skripty, filtrovat výstup a používat další typy příkazů: především podmíněný příkaz a různé typy cyklů. Také se naučíte postupy použitelné pro hromadné zpracování souborů.

1.1 Návrat k Příkazovému řádku

1.1.1 Příkazy a parametry

Víme, že Příkazový řádek lze spustit příkazem `cmd.exe` nebo přes nabídku *Start*. Příkazy, které lze spustit v *Příkazovém řádku*, můžeme rozdělit do dvou skupin:

- *vnitřní příkazy* – jsou implementovány přímo v souboru `cmd.exe`, většinou se jedná o základní a jednoduché příkazy jako například kopírování souborů nebo procházení mezi adresáři (složkami),
- *vnější příkazy* – jedná se o spustitelné soubory (s příponou COM, EXE, MSC, apod.), tuto skupinu příkazů můžeme libovolně rozšiřovat instalací nových programů.

Je obvykle jedno, jestli příkaz napíšeme malými nebo velkými písmeny (až na výjimky). V tomto dokumentu jsou při prvním použití většinou použita velká písmena.

Jen málokdo si pamatuje název a přesnou syntaxi jednotlivých příkazů, už proto, že verzi od verze těchto příkazů neustále přibývá. Proto bychom měli zvládnout vyhledání nápovědy k příkazu. To lze několika způsoby:

- program `help`, lze použít také ve tvaru `help | more` (výpis nápovědy po stránkách, výpis přerušíme klávesou q),
- nápověda k určitému příkazu ve formátu `příkaz /?` nebo `help příkaz`,

- systém nápovědy Windows přes tlačítko *Start*, vhodná klíčová slova se liší v různých verzích Windows (většinou lze použít klíčové slovo příkazy nebo příkazový řádek).

Program `help` (ať už s parametrem nebo bez něj) funguje obvykle pouze pro vnitřní příkazy. U většiny příkazů (především těch pro práci se sítí) proto volíme spíše zobrazení nápovědy použitím `/?`.

Práci s *Příkazovým řádkem* může zjednodušit program `doskey` (ve Windows je automaticky spuštěn). Nabízí kromě jiného možnost používat *historii příkazů*:

- ↑ , ↓ : zobrazíme (příp. i upravíme) předchozí/následující příkaz z historie,
- ← , → : posun o znak doleva/doprava v zadávaném příkazu,
- F7 zobrazí celou historii příkazů, šipkami je možné si vybrat ze seznamu,
- Alt+F7 smaže historii příkazů.

Příkazový řádek je možné spouštět i s parametry, tak lze určovat jeho vlastnosti nebo stanovit příkazy, které se provedou hned po spuštění příkazového řádku, viz tabulku 1.1. Pokud je použito více přepínačů, pak `/k` a `/c` musí být poslední.

Tabulka 1.1: Parametry příkazového řádku

Příklad	Význam
`cmd /k příkaz`	hned po spuštění příkazového řádku se provede zadaný příkaz
`cmd /k CD "C:\program files"`	pracovním adresářem se stane adresář uvedený v příkazu `CD`
`cmd /k C:\nastav.bat`	pokud existuje tento dávkový soubor, pak se provede hned po startu Příkazového řádku
`cmd /c příkaz`	totéž jako `/k`, ale po provedení příkazu bude Příkazový řádek ukončen
`cmd /t:xy`	nastaví barvy pozadí (`x`) a textu (`y`) obrazovky, `x` a `y` jsou hexadecimální číslice, barvy lze měnit také příkazem `color`, proto možnosti barevných kombinací lze získat příkazem `color /?`

Přepínače můžeme používat při spuštění Příkazového řádku z menu *Start* ⇨ *Spustit* nebo využít možnosti Windows při definování zástupců (do příkazové řádky zástupce napíšeme celý příkaz včetně parametrů).

Úkoly

1. Vzpomeňte si na aplikaci *Process Explorer* od firmy Sysinternals, se kterou jsme se seznámili v předchozím semestru. Pokud ho nemáte k dispozici, stáhněte z internetu.

   Spusťte Process Explorer, potom spusťte *Příkazový řádek* a obě okna umístěte tak, aby byla obě zároveň viditelná. V Process Exploreru najděte ve stromové struktuře záznam *Příkazového řádku* (`cmd.exe`). V Příkazovém řádku spusťte postupně následující příkazy a sledujte v okně Process Exploreru, co se stane s řádkem `cmd.exe`:

   - `netsh`

     program `netsh` budeme probírat později, jde o prostředí s rozsáhlými možnostmi konfigurace především síťových nastavení; práci v prostředí ukončíme příkazem `exit`,

- `copy con d:\pokusny.txt`
  opět budeme probírat později, příkaz `copy` slouží ke kopírování souborů, ale varianta s prvním parametrem `con` a druhým parametrem názvem souboru pracuje takto:
  - vytvoří soubor s daným názvem (zde na disku `D:`),
  - text, který od této chvíle budeme psát na klávesnici, se objeví na obrazovce (tj. zařízení `con` – console) a pak se z obrazovky bude kopírovat do vytvořeného souboru, každý stisk klávesy Enter bude interpretován jako konec řádku,
  - až nás přestane bavit psaní, stiskneme klávesu F6 a Enter (nebo klávesovou zkratku Ctrl+Z) a interaktivní režim bude ukončen.

  Obě možnosti srovnejte. Poznáte podle zobrazovaných informací v Process Exploreru, který z těchto příkazů je vnitřní a který vnější?

2. Vytvořte na pracovní ploše zástupce s příkazem `cmd /t:1e /k Z:` (po spuštění poklepáním se stane pracovním diskem `Z:`, barvy jsou nastaveny na modré pozadí, žluté písmo).

   Barvu textu a pozadí upravte podle svého vkusu (vkusu!), pokud nemáte přístup na disk `Z:`, použijte jiné umístění.
3. Vzpomeňte si (nebo najděte v nápovědě), jakým způsobem lze příkaz `dir` spustit rekurzívně – tedy zobrazit veškerý obsah uvnitř zadaného či pracovního adresáře včetně obsahu podadresářů, jejich podadresářů, atd., s potlačením nadbytečných informací (třeba záhlaví a zápatí). Vyzkoušejte na kořenový adresář disku `C:`, a zároveň vyzkoušejte klávesovou zkratku pro zastavení činnosti pracujícího programu – Ctrl+C – tuto zkratku si pamatujte, je velmi užitečná).
4. Vypište všechny soubory s příponou .MSC nacházející se na disku `C:` (tj. rekurzívně). Některý z těchto souborů spusťte (Pamatujete si, o jaké soubory se jedná?).
5. Proveďte totéž co v předchozím úkolu, ale pro soubory s příponou .CPL, výstup přesměrujte do souboru `panely.txt` (umístěte do adresáře, ve kterém máte právo zápisu).
6. Vypište obsah adresáře `neexistujici` (případně jiného neexistujícího adresáře), výstup přesměrujte do souboru `prazdny.txt`, chybový výstup směrujte tamtéž (přičemž název souboru uvedete v celém příkazu pouze jednou).
7. Proveďte totéž co v předchozím příkazu (využijte klávesu pro přesun v historii příkazů), ale chybový výstup přesměrujte tak, aby se nikam nezobrazil ani neuložil.
8. Výstup příkazu `time /t` směrujte tak, aby se uložil na konec souboru `cas.txt` (tj. přidávejte na jeho konec k původnímu obsahu, který zůstane zachován). Příkaz včetně směrování spusťte několikrát za sebou (využijte historii příkazů), pak zkontrolujte soubor s výslednými časovými údaji.

1.1.2 Adresáře a soubory

 Nový soubor můžeme vytvořit několika způsoby. Předně se může jednat o výstup některého programu, což můžeme zajistit i ručně – přesměrováním výstupu některého příkazu do souboru. Existuje však také způsob, jak vytvořit nový soubor s „originálním“ obsahem.

Příklad

V pracovním adresáři vytvoříme nový soubor s názvem `soubor.txt`:

```
COPY CON soubor.txt  Enter
... (nějaký obsah souboru)
...
Ctrl+Z  nebo  F6 , pak  Enter
```

Klávesová kombinace Ctrl+Z nebo F6 vytvoří textový znak znamenající konec souboru.

Podobný postup můžeme využít, pokud chceme k existujícímu souboru připojit daší řádky:

```
COPY soubor.txt+CON soubor.txt  Enter
... (řádky, které chceme připojit)
...
Ctrl+Z  nebo  F6 , pak  Enter
```

Výsledek můžeme samozřejmě uložit i do jiného souboru než původního. Nové řádky se přidají na konec souboru. Kdybychom obrátili pořadí operandů kolem symbolu „+“, přidaly by se nové řádky na začátek souboru.

Úkoly

1. V adresáři, kde máte právo zápisu, vytvořte pomocí `COPY CON` textový soubor, dovnitř vepište několik různých řádků. Pak ho přejmenujte pomocí `REN`, setřiďte a výsledek třídění uložte do jiného souboru, ten vypište pomocí `TYPE`.
2. Vytvořte v pracovním adresáři soubor `pokus1.txt`, napište do něj jakýkoliv text (několik řádků) a uložte. Potom ho zkopírujte do téhož adresáře na soubor s názvem `pokus2.txt`, otevřete, na některém řádku pozměňte jeden znak (tak aby se délka souboru nezměnila) a uložte.

   Dále porovnejte oba soubory postupně pomocí příkazů `FC` a `COMP` a porovnejte jejich výstupy. U příkazu `COMP` vyzkoušejte také uvedené přepínače.
3. Na začátek souboru `pokus1.txt`, který jste vytvořili v předchozím úkolu, přidejte řádek jakéhokoliv textu, a podobně také na jeho konec – použijte metodu příkazu `copy con` uvedenou výše v druhé části příkladu.

Zopakujeme si dva z nejdůležitějších příkazů pro zpracování textu – příkazy pro vyhledávání. Víme, že ve Windows existují dva – jednodušší `FIND` a pokročilejší `FINDSTR`. Používáme je nejen tehdy, když potřebujeme v existujícím textovém souboru najít řetězec odpovídající zadanému klíčovému slovu nebo regulárnímu výrazu, ale především (a nejčastěji) tehdy, když chceme zpracovat a profiltrovat výstup některého jiného příkazu. Typické použití je tedy ve formě filtru zřetězeně v „rouře“.

Syntaxi příkazu `findstr` známe, případně si ji můžeme najít v nápovědě `findstr /?`

Pro základní orientaci: prvky, které lze použít v regulárním výrazu, jsou v tabulce 1.2.

Příklad

V předchozím semestru jsme se již s vyhledáváním trochu seznámili, teď si pro připomenutí dáme pár příkladů.

Tabulka 1.2: Regulární výrazy v příkazu `findstr`

Prvek	Význam
`.`	Libovolný znak
`*`	Nula nebo více výskytů předcházejícího znaku nebo třídy
`^`	Začátek řádku (znak „stříška“)
`$`	Konec řádku
`[znaky]`	Jakýkoli (jeden) znak z množiny v závorkách
`[^znaky]`	Jakýkoli znak mimo prvky množiny
`[x-y]`	Jakýkoli znak v rozsahu daném dvěma znaky (jeden znak)
`\<text`	Začátek slova
`text\>`	Konec slova

V adresáři `windows\system32` (což je právě náš pracovní adresář) a případně jeho podadresářích se nachází hodně programů, které lze spustit na příkazovém řádku. Chceme najít příkazy související s diskem, zřejmě budou mít v názvu podřetězec „disk“, přípona může být libovolná:

```
dir /S /B | findstr "disk"
```

V pracovním adresáři si vytvoříme testovací soubor `pokus.txt`:

```
P:\> copy con pokus.txt
slovo první řádek znakový soubor
druhý znak slovo
třetí řádek znak, čárka
čtvrtý řádek příznak slovo
```

Stiskneme postupně klávesy F6 a Enter. Dále budeme vyhledávat v tomto souboru:

`findstr "první řádek" pokus.txt`
: očekávali bychom, že bude nalezen a vypsán pouze první řádek souboru, ale ve skutečnosti se vypíšou všechny řádky – mezeru ve výrazu totiž tento příkaz chápe jako operátor OR, tedy jsou vypsány řádky obsahující jeden nebo druhý řetězec (nepomůže ani escape sekvence – opačné lomítko před mezerou)

`findstr /c:"první řádek" pokus.txt`
: vypíše se opravdu jen první řádek, řetězec mezi uvozovkami je brán jako celek (pozor, je omezen význam metaznaků)

`findstr "znak" pokus.txt`
: vyhledá všechny výskyty řetězce `znak` v zadaném souboru, tedy vypíše všechny řádky (nachází se na všech)

`findstr "znak\>" pokus.txt`
: najde slova končící řetězcem `znak`, tj. taková, za nimiž je některý oddělovač (mezera, čárka, tečka, středník apod.), předpona je „dovolena“ – vypíše druhý, třetí a čtvrtý řádek

`findstr "\<znak\>" pokus.txt`
: vyhledá všechny výskyty slova `znak` obklopené některými oddělovači, v našem případě vypíše pouze druhý a třetí řádek (na ostatních řádcích má toto slovo ještě předponu nebo příponu)

`findstr "znak[o,]" pokus.txt`

vypíšou se řádky obsahující některý z řetězců „`znako`“ nebo „`znak,`“, tedy první a třetí

`findstr "znak[^o,]" pokus.txt`

výsledek je opačný vzhledem k předchozímu, vypíšou se řádky obsahující řetězec `znak`, za kterým *nenásleduje* písmeno `o` ani čárka (tj. druhý a čtvrtý řádek)

`findstr "^dru" pokus.txt`

hledáme řádek začínající řetězcem `dru`, to je druhý řádek (všimněte si duality významu metaznaku „`^`“)

`findstr "soubor$" pokus.txt`

vypíše se řádek, který končí řetězcem `soubor`

`findstr /c:"\<znak\>" pokus.txt`

chyba, takto je výraz brán jako literál a hledá se řetězec `\<znak\>`, nikoliv řetězec `znak` obklopený oddělovači (tj. nevypíše se nic, takový řetězec v souboru nemáme)

`findstr "\<znak\> slovo" pokus.txt`

vypíšou se všechny řádky, na každém je buď samostatný řetězec `znak` nebo řetězec `slovo`

`findstr /c:"znak slovo" pokus.txt`

získáme druhý a třetí řádek, tam jsou tyto řetězce přímo za sebou (i když první není samostatné slovo obklopené oddělovači)

`findstr /c:"znak slovo" pokus.txt | findstr "\<znak\>"`

takto jsme určili řetězec, ve kterém je `znak` samostatné slovo a je následován řetězcem `slovo` – krkolomné, ale funguje to

`findstr "\<znak\>" *.txt`

projde všechny soubory s příponou TXT nacházející se v pracovním adresáři a hledá tam zadaný řetězec, vždy napíše název souboru a pak nalezený řádek, v našem případě (předpokládejme, že se řetězec nachází ještě v dalším souboru)

```
pokus.txt:druhý znak slovo
pokus.txt:třetí řádek znak, čárka
pokus2.txt:což je typický znak těchto typů programů
```

`findstr /N "\<znak\>" *.txt`

přidá se číslo řádku, výstup:

```
pokus.txt:2:druhý znak slovo
pokus.txt:3:třetí řádek znak, čárka
pokus2.txt:51:což je typický znak těchto typů programů
```

`findstr /M "\<znak\>" *.txt`

nevypíšou se řádky s nalezeným řetězcem, ale pouze soubory, ve kterých se hledaný řetězec nachází, výstup může vypadat například takto:

```
pokus.txt
pokus2.txt
```

`findstr /S /M /I "\<znak\>" *.txt`

parametr `/S` způsobí rekurzivní chování programu – prohledávají se všechny soubory vyhovující masce `*.txt`, a to rekurzívně v celém podstromě podadresářů pracovního adresáře, vypisují se opět pouze názvy souborů (protože jsme použili parametr `/M`), navíc se porovnává bez rozlišování malých a velkých písmen (parametr `/I`)

`findstr /V "slovo" pokus.txt`

vypísou se řádky, na kterých se nenachází řetězec `slovo`, tedy pouze třetí řádek

Úkoly

1. Vyzkoušejte si příkazy z příkladu.
2. Zjistěte, zda v souboru `C:\Windows\WindowsUpdate.log` je řádek začínající řetězcem obsahujícím záznam ve tvaru „rok–měsíc–den“ pro současné datum (například 2010–01–25).
3. Pro přemýšlivé: jak by vypadal příkaz, který by dokázal totéž, co v předchozím úkolu, provést souhrnně přes všechny LOG soubory rekurzívně v pracovním adresáři?

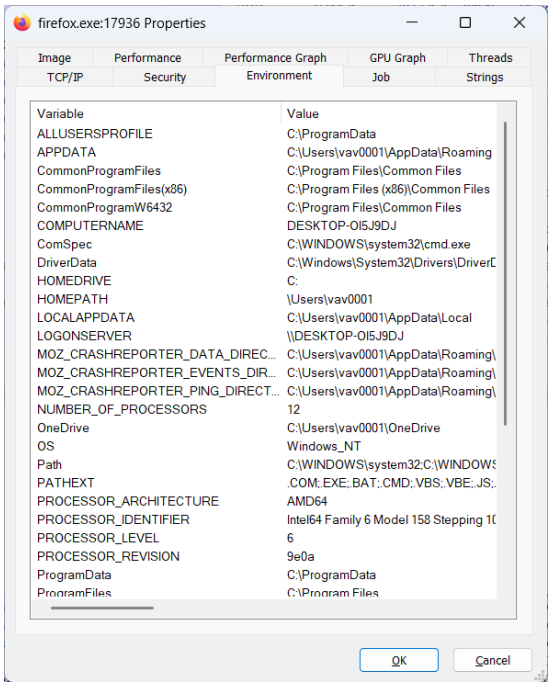
1.1.3 Proměnné a jejich využití pro čísla

Z předchozího semestru již víme, že na Příkazovém řádku lze používat proměnné, které obvykle chápeme jako řetězcové. Víme také, že existují

- *systémové proměnné* – definované pro celý systém, v registru je najdeme v klíči *HKLM/System/CurrentControlSet/Control/Session Manager/Environment*
- *uživatelské proměnné* – definované uživatelem a platné jen pro něj, v registru je najdeme v klíčích *HKCU/Environment* *HKCU/Volatile Environment*
- *dynamické proměnné* – jejich obsah je při každém volání generován dynamicky (nejsou nikde fyzicky uloženy), nelze do nich ukládat ani vytvářet vlastní nové, jsou to např. `TIME`, `RANDOM`, `ERRORLEVEL` a další, nejsou uloženy v registru.

Proces má své vlastní *prostředí*, ve kterém běží a kde má uloženo vše, co potřebuje a k čemu má přístup. Odtud máme název *proměnné prostředí* – jsou to prostě proměnné, které má proces namapovány ve svém prostředí. Jde buď o proměnné, které si proces „natáhl“ z registru (systémové nebo uživatelské), ale také o proměnné, které si proces vytvořil pro svou vlastní potřebu. Po ukončení běhu procesu se obsah prostředí vyčistí, tedy vše, co proces s proměnnými dělal, se „ztratí“. Dynamické proměnné nejsou mapovány do prostředí procesů, nemohou se z nich stát proměnné prostředí.

V Příkazovém řádku pracujeme s proměnnými (kromě dynamických) pomocí příkazu `set`, který slouží k výpisu seznamu proměnných (případně lze vypsat jen některé proměnné – zadáme řetězec, který se má vyskytovat v jejich názvu), a dále k vytváření nových proměnných či změně jejich obsahu.



Obrázek 1.1: Proměnné prostředí procesu firefox.exe v Process Exploreru

Dále využíváme příkaz `echo`, který vypisuje jakékoliv řetězce včetně obsahu proměnných. Pro výpis dynamických proměnných nelze použít příkaz `set`, proto například píšeme `echo %random%`, pokud chceme vypsat

náhodné číslo. Obecně platí, že v „univerzálnějších příkazech“, které nejsou určeny výhradně pro práci s proměnnými, musíme název proměnné uzavřít do znaků „%“.

Příklad

Několik ukázek pro připomenutí:

`set` vypíše seznam definovaných proměnných (kromě dynamických) i s jejich obsahy

`set computername` vypíše obsah proměnné `computername` (z názvu proměnné je zřejmé, že jde o název počítače)

`set mojeprom = abc` vytvoří novou proměnnou s názvem `mojeprom` a iniciuje ji řetězcem `abc`, pokud již existuje, tak pouze změní její hodnotu

`echo Název počítače: %computername%` vypíše zadaný řetězec, místo názvu proměnné dosadí hodnotu zadané proměnné (obklopení symboly % určuje řetězec k interpretaci, vyhodnocení)

`set prom=Právě přihlášen: %username%` do proměnné můžeme uložit i momentální obsah jiné proměnné, také v kombinaci s čímkoliv dalším

`set pomocna = %random%` do proměnné `pomocna` jsme načetli náhodné číslo (všimněte si, že proměnná `random`, která je dynamická, je sice na řádku se „zakázaným“ příkazem `set`, ale tímto příkazem vlastně není vypisována, příkaz pouze využívá její předem získanou hodnotu)

`dir %WINDIR%` přesun do adresáře s instalací Windows

`%systemdrive%` přesun na systémový disk

`cd %USERPROFILE%` přesun do vlastního profilu

`SET PATH=%PATH%;X:\MUJ_PROG` přidáváme další cestu do proměnné `PATH`

Počítáme. U příkazu `set` lze použít parametr `/a`. Následující řetězec pak bude považován za matematický výraz. Můžeme používat běžné aritmetické operátory a také některé bitové operace, seznam najdeme ve výpisu `set /?`.

Pokud chceme uvnitř výrazu (za přiřazovacím operátorem) použít proměnnou, umístíme výraz do uvozovek (potom všechny písmenné řetězce vpravo od přiřazovacího operátoru budou považovány za číselné proměnné a nahrazeny svou hodnotou).

Příklad

Ukážeme si několik příkladů:

`set pom=2` vytvoříme novou proměnnou a inicializujeme ji na 2

`set /a pom=2*(4-3+2)` na pravé straně přiřazovacího příkazu je výraz, proto musíme použít parametr `/a`, díky kterému se s výrazem nezachází jako s řetězcem, ale před přiřazením se vypočte

`set /a "abc=30-pom"` na pravé straně máme proměnnou, proto celý řetězec umístíme do uvozovek

`set /a "abc=30 % pom"` použili jsme operátor pro zbytek po celočíselném dělení (modulo)

`set /a pom+=2` další z forem přiřazení (k proměnné se přičte číslo 2)

`set /a 152*4` vypíše výsledek výpočtu (608), použijeme pro rychlé výpočty

`set /a "%random%+50"` dynamické proměnné musíme uzavřít mezi symboly procenta (ostatní proměnné můžeme, ale nemusíme), jinak je za ně dosazeno číslo 0; v tomto příkazu jsme určili spodní hranici pro generovaná náhodná čísla

`set /a 0x2af7` převede zadané hexadecimální číslo na desítkové (zde 10999)

Úkoly

1. Vyzkoušejte si příkazy z příkladu 1.1.3 (vytvořené proměnné budou existovat jen do vypnutí či restartu počítače, takže jich můžete vytvořit kolik chcete).
2. V nápovědě příkazu `set` najděte seznam podporovaných operátorů (je třeba použít nápovědu v textovém režimu `set /?`).
3. V příkladu o počítání s proměnnými je uveden příkaz, kterým lze stanovit spodní hranici pro generování náhodných čísel proměnnou `%random%`. Zamyslete se nad tím, jak lze generovaná čísla omezit shora. Nápověda: můžete použít zbytek po celočíselném dělení, neztraťte se v symbolech procent.

1.1.4 Skriptování v Příkazovém řádku: dávkové soubory

Skript je textový spustitelný soubor, tedy textový soubor obsahující příkazy určené k provedení nějakým interpretačním programem.

Dávkové soubory jsou soubory s příponou BAT (z anglického *batch file*). V dávkovém souboru můžeme vytvořit tzv. dávku příkazů, tedy posloupnost příkazů, která se má provést. Jde o textové soubory, které můžeme editovat například pomocí programu `notepad` (nebo jakéhokoliv jiného programu pracujícího s ASCII soubory), ale přitom to jsou spustitelné soubory jako třeba EXE (ve skutečnosti jsou interně spouštěny jako parametr příkazu `cmd`). Lze je spustit z Příkazové řádky nebo přímo ve Windows poklepáním jako kterékoliv jiné spustitelné soubory.

Dávkový soubor může mít také *parametry* (při spuštění se píšou za název BAT souboru, oddělují se mezerou), uvnitř souboru se k nim dostaneme přes proměnné `%0`, `%1`, ..., `%9`, kde `%0` je samotný název dávkového souboru, `%1` je první parametr, atd. Pokud nebyl parametr zadán, je v příslušné proměnné prázdný řetězec.

Příkazy píšeme každý na nový řádek. Kromě dříve uvedených příkazů používáme:

`REM`
komentář (remark); všechno od tohoto příkazu do konce řádku je považováno za komentář (v některých konfiguračních souborech se místo příkazu `rem` používá středník)

`ECHO`
tento příkaz už trochu známe, slouží k řízení výpisu hlášení na obrazovku:

`echo text` vypíše na obrazovku text (není třeba ho ohraničit uvozovkami)
`echo.` zařádkuje na obrazovce (vypíše prázdný řádek)
`echo ON` zapne výpis výstupů příkazů na obrazovku (implicitní nastavení), například pokud v následující posloupnosti příkazů je příkaz `COPY` ..., na obrazovku se vypíše řádek s tímto příkazem a dále vše, co samotný příkaz posílá na výstup,
`echo OFF` vypne výpis výstupů příkazů na obrazovku (tichý režim)
`echo Název tohoto souboru je %0, první parametr je %1` vypíše název a první parametr dávkového souboru, ve kterém je tento příkaz

`@příkaz`
vypne výpis na obrazovku u toho příkazu, jehož řádek tímto znakem začíná obvyklé použití příkazu `@ECHO OFF` (příkaz sice vypíná výstup na obrazovku, ale až od následujícího řádku, proto na tento řádek musíme použít znak `@`, aby se nezobrazil), tímto řádkem začíná většina dávkových souborů

`CALL soubor.bat`

spuštění jiného dávkového souboru; programy EXE a COM lze spustit zadáním jejich jména, ale kdybychom totéž provedli s BAT souborem, tak po jeho provedení by původní dávkový soubor (ve kterém byl tento BAT soubor volán) byl násilně ukončen, proto je nutné použít příkaz `CALL`

`PAUSE`

pozastaví provádění dávkového souboru, používáme, když chceme, aby si uživatel přečetl text vypsaný na obrazovce (vykonávání dalších příkazů pokračuje po stisknutí kterékoliv klávesy)

`GOTO :návěští`

příkaz skoku (narozdíl od vyšších programovacích jazyků je v dávkových souborech nezbytný), lze odskočit na kterýkoliv řádek souboru, který označíme návěštím

```
REM ukázka odskoku na řádek s návěštím :konec
:zacatek
...
goto :zacatek
...
:konec
```

Návěští musí vždy začínat dvojtečkou (a neobsahuje mezery, ani za tou dvojtečkou), v příkazu goto je mezera před celým návěštím (tj. před dvojtečkou).

`SET /p proměnná=výzva`

vypíše na obrazovku výzvu, počká až uživatel něco napíše a stiskne Enter, pak to, co uživatel zadal, uloží do proměnné:

```
set /p soub=Zadejte soubor, se kterým budeme dále pracovat:
```

`SHIFT`

posun obsahu parametrů souboru – to, co bylo uloženo v `%1`, se přesune do `%0`, obsah `%2` se přesune do `%1` atd., do `%9` se místo původního obsahu načte následující parametr, používá se, pokud má dávkový soubor více než 9 parametrů (dostaneme se tak i k desátému a následujícím).

`SETLOCAL, ENDLOCAL`

při použití v dávkovém souboru jsou všechny proměnné vytvořené mezi těmito dvěma příkazy pouze lokální (přesněji lokální jsou všechny změny související s příkazovým prostředím, obvykle jde o proměnné), platí jen v tomto bloku, a po ukončení interpretace tohoto dávkového souboru (nebo uvedení příkazu `endlocal`) přestanou existovat.

Příklad

Ukážeme si použití příkazů `setlocal` a `endlocal`. Uvnitř prostředí vytvořeného touto dvojicí příkazů vytvoříme proměnné, které přestanou existovat zároveň s koncem prostředí.

```
@echo off
...
set testovaci=xxx
setlocal
set path=%ProgramFiles%\mujprogram;%path%
set mojeprom="Moje proměnná"
set /p retezec=Zadejte řetězec, který má být později vypsán:
rem (teď uživatel něco napsal a stiskl Enter)
echo Zadali jste: %retezec%
```

```
...
rem používáme vlastní nastavení proměnné path a proměnnou mojeprom
...
set testovaci=yyy
endlocal
rem teď už má proměnná path původní hodnotu (bez našeho přídavku),
rem    proměnné mojeprom, retezec neexistují
```

Všimněte si proměnné `testovaci`. Jak to bude s ní? Vznikla ještě před vznikem vnitřního bloku definovaného příkazy `setlocal` a `endlocal`, takže zjevně bude existovat i po jeho ukončení, jenže uvnitř bloku jsme měnili její obsah. Bude změna viditelná i po odchodu z prostředí, nebo se proměnná vrátí k původní hodnotě?

Postup

Příkazy `setlocal` a `endlocal` mají také parametry, například lze vypnout či znovu zapnout tzv. *rozšíření příkazů*.[1] Podrobnosti o rozšíření příkazů můžeme zjistit například v nápovědě příkazu `cmd`.

Nejen binární spustitelné soubory, ale také dávkové soubory mohou do dynamické proměnné `errorlevel` uložit svůj ukončovací kód. Provádí se to příkazem

```
EXIT /B číslo
```

Například když chceme v dávkovém souboru ukončit jeho běh, můžeme na daný řádek umístit příkaz

```
exit /B 0
```

(to znamená, že jde o regulérní ukončení, k žádné chybě nedošlo). Pokud se rozhodneme pomocí proměnné `errorlevel` rozlišit různé typy chyb, tak například při problému s přístupem k souboru vypíšeme hlášení a provedeme `exit /B 1`, při nesprávně ukončeném kopírování `exit /B 2`, atd. podle vlastního rozhodnutí.

Další informace

Na stránce http://vavreckova.zam.slu.cz/obsahy/os/skripty.html najdete krátké ukázky dávkových souborů. Dále je zajímavá stránka s vysvětlením parametrů dávkových souborů http://www.robvanderwoude.com/parameters.php.

Úkoly

1. Vytvořte dávkový soubor, který na disku, kde máte právo zápisu, vytvoří adresář a do něho zkopíruje všechny soubory s příponou TXT z adresáře `C:\`.
2. Vytvořte dávkový soubor, ve kterém
   - zajistěte, aby nadbytečné výstupy příkazů nebyly vypisovány,
   - prompt nastavte na název právě přihlášeného uživatele (použijte příslušnou proměnnou),
   - vypište právě přihlášeného uživatele (dopište ještě nějaký vysvětlující řetězec) – pokud si nemůžete vzpomenout, jak se jmenuje proměnná obsahující název přihlášeného uživatele,

[1] Rozšíření příkazů ovlivňuje poměrně mnoho příkazů (například `for`, `assoc`, `cd`, `md`, `prompt`, `set`, atd.). Obvykle rozšiřuje množství parametrů těchto příkazů nebo obohacuje jejich chování. Standardně jsou rozšíření příkazů zapnuta, což lze ovlivnit buď ručním zadáním příslušného přepínače v příkazu `cmd` při spuštění Příkazového řádku, anebo změnou proměnné `comspec`.

využijte faktu, že příkaz `set` vypisuje seznam všech proměnných obsahujících zadaný řetězec (tedy stačí zadat `set user` a zjistíme názvy proměnných, které mají co dělat s uživateli),
- požádejte o zadání řetězce s cestou k některému adresáři (pomocí proměnné),
- předpokládejte, že uživatel zadal platnou (skutečnou, správnou) cestu k některému adresáři a tento údaj využijte:
  - vypište hlášení o tom, že následuje strom s adresářovou strukturou v zadaném adresáři,
  - pak vypište stromovou strukturu v zadaném adresáři (použijte obsah proměnné, kterou jste vytvořili při načtení řetězce od uživatele).

3. Zjistěte, jak lze spustit Příkazový řádek bez funkce rozšíření příkazů. Zjistěte (třeba v nápovědě), jak se zapnutí či vypnutí rozšíření příkazů projeví na funkcích příkazu `start`.

1.2 Návrat k PowerShellu

1.2.1 CMDLETy a objekty

Základy PowerShellu už bychom měli mít, tedy víme, že je to novější kolega Příkazového řádku postavený na technologii .NET. Kromě vnitřních příkazů PowerShellu, které se nazývají CMDLET, můžeme používat naprostou většinu příkazů běžně používaných v Příkazovém řádku. Výjimkou mohou být některé vnitřní příkazy Příkazového řádku, ale i ty jsou obvykle v PowerShellu implementovány jako aliasy. Pro připomenutí je několik nejběžnějších CMDLETů uvedených v tabulce 1.3.

Tabulka 1.3: Některé příkazy PowerShellu

Příkaz	Význam
`GET-PROCESS`	seznam spuštěných procesů včetně základních informací (jako parametr můžeme použít regulární výraz pro masku názvu procesů)
`GET-SERVICE`	seznam služeb, používá se podobně jako předchozí
`GET-COMMAND`	bez parametrů vypíše seznam příkazů, taktéž můžeme filtrovat regulárním výrazem
`GET-HELP`	jako parametr obvykle dodáváme příkaz, ke kterému chceme zobrazit nápovědu (může se nám také hodit parametr `-examples`)
`GET-ALIAS`	bez dalších parametrů zobrazí seznam aliasů
`NEW-ALIAS`	vytvoření nového aliasu: `NEW-ALIAS -name název -value hodnota`
`STOP-PROCESS`	zastaví spuštěný proces (název dodáme jako parametr)
`SET-VARIABLE`	nastaví proměnnou na danou hodnotu

Na rozdíl od příkazů Příkazového řádku (a spustitelných souborů) CMDLETy vracejí na svůj výstup *objekt nebo seznam objektů*. Pokud ten výstup jednoduše necháme vypsat, získáme samozřejmě text, ale jestliže je tento výstup přesměrován jinému CMDLETu (například přes rouru), pak ten následující CMDLET nemusí řešit rozpoznávání textu a může přímo pracovat se získanými objekty. To má kladný vliv na rychlost zpracování předávaných dat.

Výstup CMDLETu bývá formátován buď jako tabulka nebo jako seznam. U tabulky, která je výchozím typem výstupu většiny CMDLETů, jsou data jednotlivých vypisovných objektů uspořádaná

do sloupců, kdežto v zobrazení seznamu jsou data každého objektu zvlášť vypsána na jednotlivých řádcích. Pokud chceme určit výstupní formát, používáme například tyto CMDLETy:

- `FORMAT-TABLE`, zkráceně `FT`,
- `FORMAT-LIST`, zkráceně `FL`,
- `FORMAT-WIDE`, zkráceně `FW`.

Úkol

Vyzkoušejte v PowerShellu tyto tři varianty výpisu procesů:
`GET-PROCESS | FT` `GET-PROCESS | FL` `GET-PROCESS | FW`
Srovnejte se základní variantou bez určení formátování. Který typ formátování je pro tento CMDLET výchozí? Použijte příkaz `GET-COMMAND format*` a zjistěte, jaké další možnosti existují.

1.2.2 Proměnné

Oproti Příkazovému řádku se liší syntaxe používání proměnných. Když k proměnné přistupujeme, neobklopujeme její název symboly `%`, ale před název dáme symbol `$` (což syntaxe obvyklá v UNIXových systémech). Například:

```
$PSHOME
```

Jde o proměnnou obsahující adresář, v němž je PowerShell nainstalován. Ovšem pozor, takto přistupujeme k proměnným, které jsou vytvořeny přímo PowerShellem nebo si je vytvoříme sami. Seznam těchto proměnných zobrazíme příkazem

```
GET-VARIABLE
```

Mnohé proměnné však v tomto seznamu nenajdeme, především proměnné prostředí jsou přístupné jen tak, že přidáme etězec `env:`. – například pro proměnnou `PATH`

```
$env:path
```

Takže když do proměnné PATH potřebujeme přidat další umístění pro spustitelné soubory, uděláme to takto:

```
$env:path += ";novy_adresar"
```
(jak vidíme, adresáře v této proměnné jsou odděleny středníkem)

PowerShell si tedy některé proměnné vytváří sám, další si pak do svého prostředí natáhne z registru. Seznam (jeho) proměnných prostředí získáme takto:

```
get-childitem env:
```

Použili jsme příkaz, který slouží i k výpisu obsahu adresáře, položky registru a dalších typů objektů. Typ položek, které se mají vypsat, se (v případě, že nejde o soubory) určuje modifikátorem před dvojtečkou. Takže v předchozím příkazu šlo o prostředí, a kdybychom chtěli vypsat určitou část registru, například některý klíč v HKLM, použili bychom modifikátor `hklm:\` následovaný cestou k danému klíči.

Poznámka

Místo dynamických proměnných se v PowerShellu spíše používají příslušné CMDLETy. Například pro výpis data existuje CMDLET `GET-DATE` vracející objekt typu datum/čas, pomocí parametrů se pak dá určovat formát výpisu. Pro náhodná čísla máme `GET-RANDOM`.

Novou vlastní proměnnou můžeme vytvořit jednoduše takto:

```
$pom = hodnota
```

Nebo můžeme použít CMDLET `NEW-VARIABLE`, který dodává další možnosti – například určení viditelnosti proměnné (proměnná může být tzv. soukromá), nastavení vlastnosti read-only, atd.

Pokud chceme do proměnné načíst vstup od uživatele (třeba v nějakém skriptu), provádí se to stejně kostrbatě jako v Příkazovém řádku. Používá se cmdlet `read-host`, a to takto:

Příklad

Vytvoříme proměnnou `Jmeno` a požádáme uživatele o zadání jeho jména.

```
PS C:\Users\uzivatel> $Jmeno = read-host "Zadejte prosím své jméno"
Zadejte prosím své jméno:
```

Uživatel zadá řetězec, klepne na Enter a výsledek se uloží do námi určené proměnné.

Úkoly

1. Vypište seznam proměnných, vyzkoušejte také výpis proměnných prostředí.
2. Vytvořte novou proměnnou `rok`, do ní uložte momentální rok. Pak se do této proměnné pokuste načíst něco od uživatele (sice nejsme ve skriptu, ale jde o i přímo v PowerShellu, tím uživatelem budeme prostě my sami).
3. V nápovědě zjistěte, jak se používá příkaz `NEW-VARIABLE`.

1.2.3 Funkce

Vlastní příkaz můžeme vytvořit jako alias nebo jako funkci. Aliasy jsou určeny pro jednodušší případy, ale pokud potřebujeme příkaz zastupující víc „vnořených“ příkazů nebo obecně něco složitějšího, vytvoříme funkci.

Příklad

Ukážeme si vytvoření funkce, jejímž účelem bude vypsat seznam všech dynamických knihoven (s příponou .DLL).

```
PS C:\Users\uzivatel> function knihovny
```

Prompt se změní na `>>` a můžeme psát tělo funkce (jako v jazyce C, v lomených závorkách):

```
{
echo "Seznam knihoven v systemu:"
get-childitem c:\windows\system32\*.dll -recurse -force
}
```

Místo příkazu `echo` bychom mohli použít CMDLET `WRITE-HOST`, ale kdo by se s tím psal...

Naši vytvořenou funkci bychom měli najít v seznamu `GET-COMMAND` (případně můžeme filtrovat podle jména). Kód funkce pak zobrazíme takto:

```
(get-command knihovny).definition
```

Další informace

https://learn.microsoft.com/cs-cz/powershell/scripting/learn/ps101/09-functions

Úkol

Vytvořte si vlastní funkci podle svého uvážení, například takovou, která vypíše postupně různé druhy proměnných (vlastní proměnné PowerShellu, vaše vytvořené, a pak proměnné prostředí).

1.2.4 Objekty a filtrování výstupu

Každý objekt má určité vlastnosti a metody. V PowerShellu se pro ně používá poněkud nepřípadný název „member“ a pro jejich zjištění používáme CMDLET `GET-MEMBER`.

Příklad

Chceme zjistit, jaké vlastnosti a metody mají objekty typu služba. To můžeme udělat takto:

```
GET-SERVICE | GET-MEMBER
```

Získáme tabulku všech typů údajů, které jsou o službách evidovány.

Úkoly

1. Zjistěte v nápovědě, k čemu se používá CMDLET `GET-MEMBER`. Co je tedy míněno tím „member“?

   Vyzkoušejte na příkazu `GET-SERVICE | GET-MEMBER`. Výstupem nebude seznam služeb, ale seznam vlastností a metod objektu typu služba.
2. Zjistěte, jaké vlastnosti a metody existují pro objekt typu proces (tj. budete potřebovat CMDLET vypisující procesy).

Výstup příkazů může být poměrně dlouhý, tedy je dobré si ho filtrovat, abychom získali ideálně jen to, co nás opravdu zajímá. Pokud se soustředíme na výstup typu tabulka, můžeme chtít ovlivnit vypsané řádky (to jsou jednotlivé zjištěné objekty) a/nebo uvedené sloupce (to jsou vlastnosti těchto objektů). Nejdřív se soustředíme na sloupce.

Existují dvě možnosti: buď použijeme formátovací CMDLETy `FT` a `FL` a jejich parametr `-property`, nebo zvolíme CMDLET `SELECT-OBJECT`.

Příklad

Pomocí `GET-MEMBER` jsme si zjistili, že procesy mají kromě jiných vlastnosti `name`, `id`, `ProductVersion`. Chceme vypsat hodnoty těchto vlastností pro všechny procesy s názvem firefox a thunderbird (dosaďte si klidně jiné procesy).

```
get-process -name firefox,thunderbird | ft -property name,id,productversion
```

Pokud chceme uvést jen jeden proces, nemusíme použít atribut `-name`, stačí rovnou napsat název procesu.

Teď si ukážeme totéž s využitím CMDLETu `SELECT-OBJECT`:

```
get-process -name firefox,thunderbird | select-object name,id,productversion
```

Výstup by v obou případech měl být stejný, takže je na nás, co budeme preferovat.

A teď řádky. Už víme, že výstupem CMDLETu bývá objekt nebo (většinou) seznam objektů. Filtrovat můžeme buď tradičně pomocí jednoduchých regulárních výrazů (hvězdička apod.), nebo můžeme použít CMDLET `WHERE-OBJECT`. Pomocí tohoto CMDLETu můžeme filtrovat podle hodnot konkrétních vlastností objektů (ty dokážeme zjistit pomocí `GET-MEMBER`).

Příklad

Chceme seznam všech procesů, které na procesoru strávily více než 10 sekund, tedy to nebudou žádné procesy čekající na pozadí bez toho, aby něco dělaly.

```
get-process | where-object {$_.cpu -ge 10}
```

Příkaz `WHERE-OBJECT` postupně prochází seznam objektů, které dostal přes rouru. Dvojznak `$_` nám zastupuje právě zpracovávaný objekt, k tomu přidáme přes tečku název vlastnosti, která nás zajímá. Nemůžeme používat běžné relační operátory, místo nich máme slovní operátory, zde jsme použili `-ge` znamenající $\geq$.

Seznam procesů, jejichž základní priorita je větší než 9, získáme takto:

```
get-process | where-object {$_.basepriority -gt 9}
```

Pro CMDLET `WHERE-OBJECT` existuje alias `WHERE`, což nám ušetří napsání několika písmen.

Úkoly

1. Vypište seznam procesů jako tabulku, zvolte sloupce s názvem procesu, počtem vláken (threads) a dobou startu (StartTime). Tentýž výstup si nechte vypsat i jako seznam a široký.
2. Vypište seznam procesů s názvem svchost jako tabulku, ke každému vlastnost StartTime a PSConfiguration. Pokud máte PowerShell spuštěný s běžnými oprávněními, některé řádky budou částečně prázdné. Porovnejte obsah jednotlivých sloupců.
3. Zjistěte, jaké vlastnosti lze vypsat u služeb a souborů (připomeňme si, že seznam souborů lze zobrazit CMDLETem `get-childitem`). Některé z vlastností vypište podobně, jak jsou na konci posledního příkladu u procesů.

1.2.5 Verze a moduly

Powershell existuje v různých verzích. Sem tam se objeví nová verze, a proto je dobré občas zkontrolovat a nainstalovat novou verzi. Proč? Protože v nových verzích přibývají další příkazy (cmdlety) a odstraňují se různé problémy s funkčností příkazů již zahrnutých.

Příklad

Ke zjištění, kterou verzi máme, můžeme použít příkaz `host` (nebo `get-host`):

```
PS C:\Users\uzivatel> host

Name             : ConsoleHost
Version          : 5.1.14409.1018
.....
```

Další možnost je vypsat si obsah proměnné `PSVersionTable`:

```
PS C:\Users\uzivatel> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.14409.1018
PSEdition                      Desktop
PSCompatibleVersions           1.0, 2.0, 3.0, 4.0...
BuildVersion                   10.0.14409.1018
...
```

Pravděpodobně budete mít PowerShell verze 5.1 (i když máte nainstalované Windows 11). Tato verze se už nevyvíjí, není aktualizována (až na bezpečnostní záplaty). Existuje verze 7, která je dále vyvíjena, ale je nutno ji nainstalovat, do systému se nedostane automaticky.

K instalaci najdeme informace například zde:
https://learn.microsoft.com/cs-cz/powershell/scripting/install/install-powershell-on-windows?view=powershell-7.5

Původně byla tabulka verzí na webu projektu https://github.com/PowerShell/PowerShell/. Teď tam najdeme jen odkaz na web, ze kterého se dá stáhnout instalační obraz, a také tam jsou dostupné zdrojové kódy.

Pozor – veškeré zásahy do instalace PowerShellu vyžadují vyšší přístupová oprávnění, tj. například pokud upgrade provádíme přes Příkazový řádek nebo PowerShell, spustíme okno jako správce.

Kromě samotného PowerShellu je vhodné mít kompletní a aktuální také nápovědu. To se provádí přímo v PowerShellu cmdletem

```
update-help
```

spuštěným samozřejmě v okně s oprávněním správce (tj. už když spouštíte okno PowerShellu, použijte pravé tlačítko myši a zvolte *Spustit jako správce*). Kdyby aktualizace nevypadala dokončeně, zkuste restartovat Windows.

Modul v PowerShellu je balíček, který obsahuje různé prvky použitelné v PowerShellu, například další cmdlety, funkce, proměnné, aliasy a další. Některé moduly také umožňují napojit se na jiné nástroje, například Office365, Active Directory, různé virtualizační nástroje (hypervizory) či bezpečnostní nástroje. Některé moduly jsou standardní součástí PowerShellu, jiné je třeba doinstalovat, pokud chceme jejich obsah používat.

Příklad

Jak zjistit, které moduly máme stažené či dokonce nainstalovány? Cestu k modulům zjistíme výpisem proměnné `$ENV:PSMODULEPATH`.

```
PS C:\Users\uzivatel> $ENV:PSMODULEPATH
C:\Users\uzivatel\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\
Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
```

Proměnná může obsahovat více než jednu cestu, z nich jedna je „systémová“. Seznam modulů získáme výpisem obsahu dané složky, například pro tu systémovou (v cestě použijte vaši verzi PowerShellu):

```
PS C:\Users\uzivatel> dir c:/windows/system32/windowspowershell/v1.0/modules

    Directory: C:\windows\system32\windowspowershell\v1.0\modules

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d---s-         12.4.2011     10:45                AppLocker
d---s-         12.4.2011     10:34                BitsTransfer
d-----        12.10.2020     16:24                CimCmdlets
d-----        12.10.2020     16:24                ISE
d-----        13.10.2020      3:20                Microsoft.PowerShell.Archive
d-----        12.10.2020     16:12                Microsoft.PowerShell.Diagnostics
d-----        12.10.2020     16:12                Microsoft.PowerShell.Host
d-----        12.10.2020     16:12                Microsoft.PowerShell.LocalAccounts
.... (zkráceno)
d-----         12.4.2011     10:34                TroubleshootingPack
```

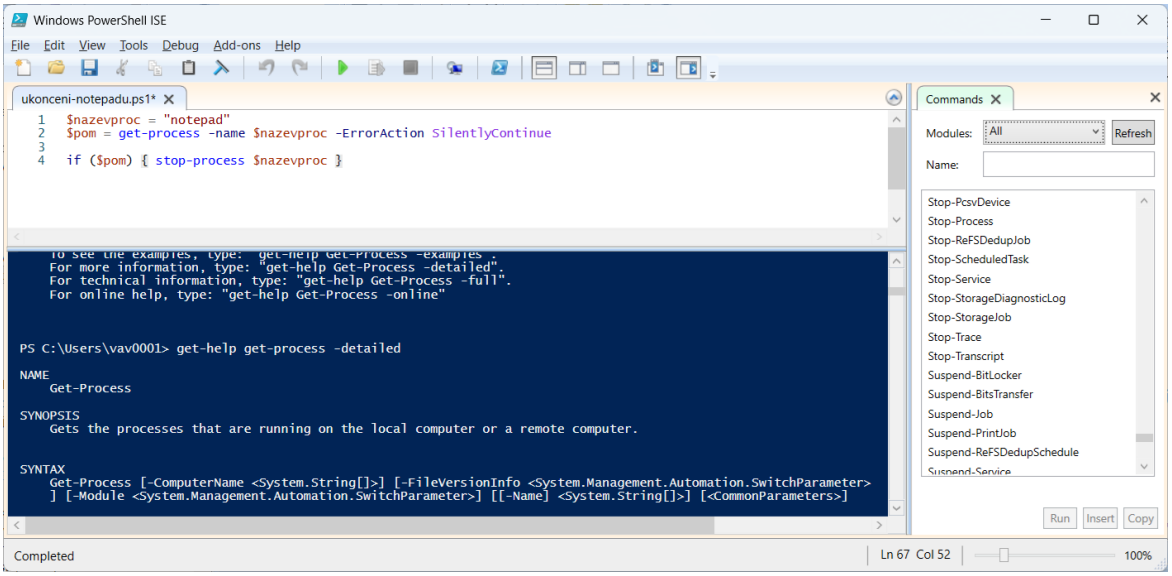
V příkladu je výpis systémového úložiště modulů, nicméně pokud některý modul chceme sami přidat, měli bychom zvolit jinou cestu: tu v Program Files (modul bude přístupný všem uživatelům) nebo v Users (bude přístupný jen tomu, komu patří příslušný profil).

Instalace nového modulu spočívá v těchto krocích:

- stáhneme si modul, oblíbené úložiště je například PowerShell Gallery (https://www.powershellgallery.com/, v menu nahoře *Packages*),
- uložíme modul do úložiště, které jsme zjistili podle předchozího příkladu,
- importujeme modul (tj. řekneme PowerShellu, že ho má používat), a to příkazem `import-module -name název`

1.2.6 Skriptování v PowerShellu

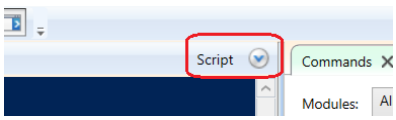
Také v PowerShellu lze psát skripty (zde to budou soubory s příponou .ps1). Můžeme používat jakýkoliv textový editor, který neukládá formátování, ale ve Windows existuje také nástroj přímo pro tento účel: PowerShell ISE. Jeho náhled je na obrázku 1.2.



Obrázek 1.2: Prostředí PowerShell ISE pro psaní skriptů

Poznámka

PowerShell ISE spustíme běžně přes nabídku Start. Možná někoho zarazí, že po spuštění není vidět horní část okna s bílým pozadím, ve které máme na obrázku 1.2 kód skriptu. Je třeba použít „šipku“, pak se ta část okna rozbalí.



Skripty .ps1 není možné spouštět tak jednoduše jako dávkové soubory Příkazového řádku. Předně musí být povoleno spouštění těchto skriptů. To se dá provést buď pomocí zásad skupiny, nebo přímo v PowerShellu CMDLETem `set-ExecutionPolicy`. Pro toto nastavení existuje celá řada hodnot, nejběžnější jsou:

- restricted – výchozí hodnota, žádný skript nelze spustit (a ani se nanačítají konfigurační soubory),
- unrestricted – všechny skripty lze spustit, také se načítají konfigurační soubory PowerShellu,

- AllSigned – lze spustit pouze skripty podepsané důvěryhodným digitálním podpisem,
- RemoteSigned – místní skript lze spustit bez problémů, u skriptu staženého ze sítě bude vyžadován důvěryhodný digitální podpis.

Příklad

Zjistíme, jestli můžeme spouštět skripty PowerShellu:

```
PS C:\Users\uzivatel> get-executionPolicy
Restricted
```

Aha, takže nelze. Nicméně to můžeme změnit:

```
PS C:\Users\uzivatel> set-executionPolicy unrestricted
```

Ovšem to bude fungovat jen tehdy, když budeme mít PowerShell spuštěný s oprávněním správce, jinak se nám zobrazí chybové hlášení.

Pokud vůbec potřebujeme spouštět skripty PowerShellu, doporučuje se použít nastavení RemoteSigned.

Předpokládejme, že máme napsaný skript a vyřešeno oprávnění ke spouštění skriptů. Přímo v PowerShellu se skript spouští tak, že musíme uvést jeho celý název včetně cesty k němu, také včetně přípony souboru. Navíc musí být správně nastaveno oprávnění ke spuštění přímo k danému souboru.

Další informace

Nejdřív několik základních zdrojů o PowerShellu:

- https://www.tutorialspoint.com/powershell/index.htm
- https://www.johndcook.com/blog/powershellcookbook/
- https://www.powershellgallery.com/

PowerShell nabízí zajímavé funkce pro zpracování řetězců a také se zdá být velmi silným nástrojem pro psaní skriptů. Zde se těmto vlastnostem již nebudeme věnovat, další informace najdeme například v těchto zdrojích:

- https://docs.microsoft.com/cs-cz/powershell/scripting/samples/sample-scripts-for-administration?view=powershell-7.1
- http://en.wikipedia.org/wiki/Windows_PowerShell
- https://www.techrepublic.com/blog/10-things/10-fundamental-concepts-for-powershell-scripting/

1.3 Složené příkazy v Příkazovém řádku

1.3.1 Propojení příkazů a podmíněné vyhodnocování

Pokud chceme na jeden řádek umístit více příkazů, musíme je spojit znakem `&`. Jednou z možností použití tohoto zřetězení příkazů je v *Zástupcích*, také se používá ve složených příkazech (seznámíme se s nimi v následující podsekci).

Příklad

```
REM vyčistíme obrazovku, vypíšeme hlášku, prázdný řádek, pak obsah pracovního adresáře:
CLS & ECHO Výpis pracovního adresáře: & ECHO. & DIR
```

Další způsob propojení příkazů už známe – rouru:

Příklad

```
REM chceme zjistit údaje týkající se hesla uživatele novak:
net user novak | find /i "heslo"
```

(vypíše se, kdy bylo heslo naposledy změněno, kdy vyprší, zda uživatel smí měnit heslo, apod.)

Příklad

Vytřídíme soubory a adresáře podle atributů. Příkaz vypisuje postupně všechny tři skupiny souborů a adresářů.

```
ECHO Skryté: & DIR /a:h & ECHO Systémové: & DIR /a:s & ECHO Pro čtení: & DIR /a:r
```

Z UNIXových systémů byla do Windows řady NT převzata možnost využití výše uvedených symbolů pro podmíněné vyhodnocování. Pokud symboly `&` a `|` zdvojíme, budou se chovat jinak.

- dvojsymbol `&&` znamená konjunkci:
  - pokud předchozí příkaz skončil s úspěchem (true), bude vyhodnocen i následující příkaz,
  - pokud předchozí příkaz skončil s neúspěchem (false), nebude následující příkaz vyhodnocován (protože false v konjunkci s čímkoliv znamená vždy false v celém výrazu),
- dvojsymbol `||` znamená disjunkci (mezi znaky nesmí být mezera):
  - pokud předchozí příkaz skončil s úspěchem (true), další příkaz již nebude vyhodnocován (protože jedno true v disjunkci znamená true v celém výrazu),
  - pokud předchozí příkaz skončil s neúspěchem (false), bude vyhodnocen i další příkaz.

Typické použití je vypisování hlášení při správném nebo nesprávném provedení prvního příkazu v pořadí.

Příklad

Příkaz `net start` slouží ke spouštění služeb. Chceme spustit službu `fax` a vypsat hlášení při chybě:

```
net start fax || ECHO Službu Fax Service se nepodařilo spustit.
```

Chceme vypsat hlášení při bezchybném provedení příkazu:

```
net start fax && ECHO Služba Fax Service byla spuštěna.
```

Příklad

Použijeme příkaz `net user` k práci s uživateli, kterému se budeme podrobněji věnovat až v pozdějších sekcích. Porovnáme tyto dva příkazy:

```
net user neexistujici | ECHO Neexistující uživatel
net user neexistujici || ECHO Neexistující uživatel
```

První příkaz je samozřejmě nesmyslný, hláška se vypíše i při úspěšném vypsání údajů o uživateli, kdežto druhý příkaz vypíše hlášku pouze při chybě (uživatel neexistuje). Zajímavé je však to, v jakém pořadí se vypisují výstupy příkazů. V prvním případě se nejdřív objeví výstup `ECHO` a pak až chybové hlášení `NET`, v druhém případě právě naopak. Je to z toho důvodu, že v prvním případě se chybový kód vytváří pro řádek jako celek.

Pokud chceme tuto chybovou hlášku použít jako výchozí, můžeme příkaz poupravit:

```
net user neexistujici 2>nul || ECHO Neexistující uživatel
```

Příklad

Chceme ukončit dávkový soubor při výskytu chyby a zároveň do proměnné `errorlevel` uložit číslo, ze kterého by bylo možné poznat, jaká chyba nastala.

```
@echo off
...
dir /b 2>nul || exit /B 1
copy *.txt || exit /B 2
```

Pokud došlo k chybě u příkazu `dir`, v proměnné `errorlevel` bude hodnota 1, jestliže chyba nastala v dalším příkazu, bude v `errorlevel` hodnota 2, atd. Když nedojde k žádné chybě, hodnota `errorlevel` bude obsahovat 0.

Pokud budeme chtít i něco vypsat (nebo provést jiný příkaz), můžeme příkazy zřetězit:

```
dir /b 2>nul || echo Nastala chyba & exit /B 1
```

Úkoly

1. Příkaz `dir` je jedním z příkazů používajících proměnnou `errorlevel` pro uložení typu chyby. Sestavte příkaz, který v případě chyby vypíše hodnotu proměnné `errorlevel` (použijte podmíněné propojení příkazů, výpis proměnné `errorlevel` proveďte pomocí `echo`). Vyzkoušejte například tak, že příkazu `dir` zadáte neexistující adresář.
2. Sestavte příkaz, který buď vypíše všechny skryté soubory a podadresáře v pracovním adresáři, anebo (když tam žádné nejsou) hlášení, že v zadaném adresáři žádné skryté soubory nejsou.
3. Zjistěte, jaká hodnota se objeví v proměnné `errorlevel`, když se pokusíte vypsat sdílené prostředky na (jiném) počítači v síti, který neexistuje (pro název počítače použijte sice správnou syntaxi, ale nesmyslný název počítače, třeba `\\ABC`).

   Příkaz vypisující sdílené prostředky na jiném počítači je `net view \\nazev-pc`.

1.3.2 Podmíněný příkaz

Příkaz `IF` používáme v dávkových souborech k větvení programu. Existuje ve více variantách: můžeme testovat existenci souboru (či složky), existenci proměnné (zda je definována), chybový výstup předchozího programu, a pak můžeme porovnávat hodnoty.

`IF EXIST`

testujeme, zda existuje soubor

`IF EXIST soubor příkaz` pokud existuje uvedený soubor, provede zadaný příkaz

`IF EXIST dopis.txt TYPE dopis.txt | MORE` pokud existuje `dopis.txt`, bude vypsán, a to po stránkách

`IF NOT EXIST dopis.txt ECHO soubor dopis.txt neexistuje` pokud zadaný soubor neexistuje, vypíše se chybové hlášení

`IF EXIST dopis.txt (TYPE dopis.txt) ELSE ECHO soubor neexistuje`
pokud existuje soubor `dopis.txt`, bude vypsán, jinak se vypíše chybové hlášení; všimněte si, že příkaz před `else` je uzavřen do závorek, protože obsahuje mezeru (jinak by `else` bylo považováno za parametr příkazu z první větve, nebylo by nalezeno)

Příklad

Chceme v dávkovém souboru se souborem provést postupně několik operací, ale jen pokud existuje.

```
IF EXIST ab.txt (
    ECHO Soubor ab.txt: & TYPE ab.txt & DEL ab.txt & GOTO :hotovo )
ECHO Soubor ab.txt neexistuje
:hotovo
```

Pokud zadaný soubor existuje, vypíše se o tom hláška, soubor je vypsán a následně smazán, pokud neexistuje, pouze se o tom vypíše hláška na obrazovku)

`IF DEFINED`

zjistí, jestli je definovaná zadaná proměnná

`IF DEFINED userprofile CD /D %userprofile%` pokud je definována proměnná `userprofile`, přesuneme se pomocí ní do uživatelského profilu (parametr `/D` změní nejen pracovní adresář, ale i pracovní disk, viz nápovědu příkazu `CD`)

`IF DEFINED date ECHO %date%` pokud je definována proměnná `date`, vypíše její hodnotu (příkaz `if defined` je jediným příkazem, který dokáže pracovat s dynamickými proměnnými i bez jejich obklopení symboly %)

`IF DEFINED vlastni_cesta CD %vlastni_cesta%` příkaz přesunu do adresáře uloženého v proměnné bude proveden jen tehdy, pokud proměnná existuje

`IF NOT DEFINED cesta set cesta=%cd%` pokud není definována proměnná `cesta`, vytvoříme ji a inicializujeme na cestu k pracovnímu adresáři

`IF ERRORLEVEL`

testuje, zda se proměnná `errorlevel` rovná danému číslu; do proměnné `errorlevel` ukládají programy číslo určující způsob jejich ukončení (0 znamená OK, vyšší číslo některý druh ukončení s chybou), tento příkaz se často kombinuje s příkazem skoku a při testování různých hodnot naskládáme víc příkazů `if errorlevel` pod sebou

`IF ERRORLEVEL číslo příkaz` pokud je v `errorlevel` uložena hodnota *větší nebo rovna* zadanému číslu, provede se příkaz a zároveň se proměnná vynuluje

`IF ERRORLEVEL 2 GOTO :NAV2` pokud poslední spuštěný program vrátil hodnotu 2 nebo vyšší, vyhodnotí podmínku jako *true* (provede odskok na dané návěští a vynuluje proměnnou `errorlevel`), jinak *false* (odskok neprovede a pokračuje následujícím příkazem)

`IF ERRORLEVEL 1 exit /B 1` předáme chybový kód o úroveň výše (dávkový soubor, ve kterém pracujeme, bude ukončen s chybovým kódem 1)

Příklad

Následující úseky kódu z dávkového souboru provedou totéž – načtou od uživatele vstup, a pokud je to prázdný řetězec, vypíšou hodnotu proměnné `errorlevel`.

1. `set /p vypis="Zadej řetězec: " || ECHO Errorlevel = %errorlevel%`
2. ```
   set /p vypis="Zadej řetězec: "
   if errorlevel 1 ECHO Errorlevel = %errorlevel%
   ```
3. ```
   set /p vypis="Zadej řetězec: "
   if %vypis%=="" ECHO Errorlevel = %errorlevel%
   ```

Nesmíme zapomenout, že pokud je v proměnné `errorlevel` číslo větší nebo rovno uvedenému číslu (testování dopadne s hodnotou *true*), proměnná se vynuluje, proto tyto řádky *nesmíme* vyměnit, testování začíná od nejvyššího čísla, které nás zajímá.

`IF op1 OPERÁTOR op2`
porovná řetězce nebo čísla (řetězce by měly být v uvozovkách) podle zadaného operátoru. Možné operátory jsou v tabulce 1.4 (pokud jsou oba operandy čísla, je s nimi také vnitřně zacházeno jako s čísly)

Tabulka 1.4: Operátory pro příkaz `IF`

Operátor	Význam
`==`	rovná se (pro řetězce)
`EQU`	rovná se (pro čísla)
`NEQ`	nerovná se

Operátor	Význam
`LSS`	menší než
`LEQ`	menší nebo rovno
`GTR`	větší než
`GEQ`	větší nebo rovno

`IF "%1"=="" MD novy_adr` pokud první parametr dávkového souboru obsahujícího tento příkaz je prázdný řetězec (tj. nebyl zadán, v tomto případě nebyl zadán žádný parametr kromě samotného názvu souboru), pak vytvoříme nový adresář

`IF "%1"=="" (MD novy_adr) ELSE MD %1` podobně jako předchozí, ale pokud soubor neexistuje, provede se větev `else`

`IF NOT "%prom%"=="%1" ECHO nerovná se` pro porovnávání řetězců používáme jen zdvojené rovnítko, jestliže chceme testovat „nerovná se“, použijeme klíčové slovo `not`

`IF %prom% GEQ 0 (ECHO nezáporné) ELSE ECHO záporné` zjistí, jestli je hodnota proměnné větší nebo rovna nule

Úkoly

1. Vyzkoušejte výše uvedené příklady (pokud to jde). Vytvořte si dvě vlastní proměnné, uložte do nich různá čísla a vyzkoušejte také porovnávání číselných proměnných.
2. Na stránce http://ss64.com/nt/robocopy-exit.html si všimněte využití testování chybových kódů (proměnná `errorlevel` u příkazu `robocopy`. Jsou zde uvedeny přímo úseky z dávkových souborů využívajících k zálohování příkaz `robocopy`.
3. Vytvořte dávkový soubor, který bude zkoušet sčítání. Proměnná `random` vrací hodnotu v rozmezí 0–32 767, což je poměrně hodně, tedy v příkazech generujících operandy upravte maximální hodnotu například pomocí operace dělení (aby sčítání bylo proveditelné rychle zpaměti).

   Cyklus implementujte pomocí příkazu `goto` (zeptejte se, zda chce uživatel pokračovat, když ne, proveďte skok na konec souboru). V cyklu do proměnných vygenerujte dvě náhodná čísla z vhodného intervalu, vypište „první číslo + druhé číslo=“ a očekávejte výsledek. Ten pak otestujte a vypište hodnocení.

1.3.3 Cyklus přes množinu

V Příkazovém řádku a v dávkovém souboru můžeme použít příkaz `FOR` s množinami řetězců. Zadáváme vždy proměnnou, přes kterou probíhá cyklus, množinu, ze které jsou brány řetězce (lze použít i zástupné znaky) a příkaz, který se má opakovaně provést. Do proměnné jsou postupně dosazovány všechny prvky množiny.

Před proměnnou píšeme

- jeden symbol procenta %, pokud příkaz zadáváme přímo na Příkazovém řádku,
- dva symboly procenta %%, pokud příkaz píšeme do dávkového souboru (aby nedošlo k záměně s jinými vnitřními proměnnými, které předchází jeden symbol %),
- další symbol procenta % přidáváme ve vnořeném cyklu (tj. pokud máme v dávkovém souboru cyklus `for` vnořený v jiném cyklu for, vnitřní proměnná vnitřního cyklu `for` bude mít tři symboly procenta, atd.).

`FOR %C IN (*.BAT *.MSC *.EXE) DO DEL %C` smažeme všechny soubory s příponou BAT, COM a EXE v pracovním adresáři (použito přímo na Příkazovém řádku)

`FOR %%C IN (*.BAT *.MSC *.EXE) DO DEL %%C` totéž (použito v dávkovém souboru)

`FOR %%a IN (*.TXT) DO ECHO %%a & COPY "%%a" "Kopie\%%a"` postupně vypisuje názvy všech souborů s příponou TXT v pracovním adresáři a v podadresáři `Kopie` od každého vytvoří kopii. Můžeme pracovat samozřejmě také s adresáři, nejen se soubory (použito v dávkovém souboru)

Příklad

Množina nemusí nutně obsahovat jen názvy souborů. Pokud uvádíme jmenovitě více položek, oddělíme je čárkou nebo jen mezerou. Sestavíme příkaz, který postupně vypíše čísla od 1 do 4:

```
for %i in (1 2 3 4) do @echo %i
```

U příkazu `for` bychom si měli dávat pozor především na to, že pokud příkaz v těle ovlivňuje množinu, nad kterou celý příkaz pracuje, může se stát, že výpočet půjde do nekonečna. Kromě znalosti důležité klávesové zkratky Ctrl+C bychom proto měli vždy dbát o to, aby v takovém případě byl výstup vnořeného příkazu směrován jinam (třeba do podadresáře).

Příklad

Porovnáme funkčnost těchto dvou příkazů:

```
for %i in (*.docx *.jpg *.png *.txt) do copy %i Zaloha_%i
for %i in (*.docx *.jpg *.png *.txt) do copy %i Zaloha\%i
```

První příkaz je chybný, způsobí zacyklení. Je sice užitečné, že všechny soubory se zadanými příponami budou zálohovány, ale protože jsou vytvořené zálohy uloženy v témže adresáři, začnou se postupně vytvářet i zálohy záloh, atd., například pro soubor `soub.txt` vzniknou postupně tyto soubory:

```
Zaloha_soub.txt
Zaloha_Zaloha_soub.txt
Zaloha_Zaloha_Zaloha_soub.txt
```

atd.

Oproti tomu druhý uvedený příkaz funguje dobře, protože vytváří nové soubory v podadresáři.

Příklad

Budeme počítat zkopírované soubory, následující kód je v dávkovém souboru:

```
@echo off
set pocet=0
echo Kopírujeme všechny soubory s příponou ini:
set /p adresar="Zadejte cílový adresář s celou cestou: "
if not exist "%adresar%" goto :neexistujeadr

for %%i in (*.ini) do (
  xcopy /s /q %%i "%adresar%"
  set /a pocet+=1
)
goto :ok

:neexistujeadr
echo Zadaný adresář %adresar% neexistuje.
goto :konec
:ok
Echo Kopírování dokončeno, bylo zkopírováno %pocet% souborů.
:konec
```

Další informace

Příkaz `FOR` má ve skutečnosti mnohem širší možnosti použití (zjistíme pomocí `for /?` nebo v grafické nápovědě). Najdeme tady několik různých přepínačů, a také modifikátory, kterými se dostaneme k částem řetězce dosazeného za proměnnou.

Postup

Například u některých konverzních programů je třeba zadat tentýž název souboru, jako původní, ale s jinou příponou, zde vidíme použití přímo na příkazovém řádku:

```
for %i in (*.png) do convert %i %~ni.eps
```

(program `convert` je součástí balíku programů *Image Magick*) – uvedený příkaz provede konverzi všech souborů s příponou PNG v pracovním adresáři na stejně pojmenované soubory, ale s příponou (a také typem) EPS, modifikátor `~n` způsobí u proměnné odtržení přípony včetně tečky, a tedy řetězec `%~ni` vrátí tentýž název souboru jako `%i`, ale bez přípony.

Úkoly

1. Vytvořte dávkový soubor, který postupně všechny své parametry vypíše na samostatné řádky, pokud žádný parametr není zadán, vypíše pouze řetězec `"Žádný parametr nebyl zadán"` (využijte příkazy `IF`, `GOTO`, `FOR`, `SHIFT`, `ECHO`, neexistující parametr zjistíte tak, že příslušná proměnná je prázdný řetězec).
2. Vytvořte dávkový soubor, který vypíše obsah zadaného adresáře (tento adresář bude zadán jako první parametr souboru). Dále se zeptá, které soubory má vypsat, načte řetězec zadaný uživatelem (příkaz `SET /p`) a použije v příkazu `FOR` s příkazem `DIR`.
3. V příkazu uvedeném v příkladu se seznamem čísel na straně 25 odstraňte symbol `@` před `echo`. Jaký vliv má uvedení či neuvedení tohoto symbolu?

1.3.4 Cyklus přes interval

Pokud chceme použít cyklus `for` tak, jak je obvykle používán v programovacích jazycích, máme k dispozici přepínač `/L`. Základní syntaxe je:

```
for /L %prom in (začátek,krok,konec) do příkaz
```

Je to podobné, jako když například v jazyce C napíšeme příkaz

```
for (prom=začátek; prom<konec; prom+=krok)
```

Také máme možnost určit délku kroku jinou než 1, to znamená, že proměnná `prom` bude v každém dalším volání zvýšena o zadanou hodnotu.

Příklad

Vypíšeme lichá čísla z intervalu od 1 do 20:

```
for /L %i in (1,2,20) do @echo %i
```

Vypočteme faktoriál zadaného čísla:

```
set /p konec="Zadej číslo: "
set vypocet=1
for /L %i in (1,1,%konec%) do @set /a "vypocet*=%i" & echo.
```

Úkoly

1. Na Příkazovém řádku vypočtěte aritmetický průměr ze zadaného počtu zadaných čísel (využijte postup z výše uvedeného příkladu).
2. Podobně vypište řadu Fibbonacciho čísel o délce zadané uživatelem.

1.3.5 Hromadné zpracování dat

Přepínač `/F` příkazu `for` můžeme použít ke zpracování textového souboru. To využijeme například tehdy, když do souboru uložíme posloupnost záznamů, na které chceme postupně uplatnit některý příkaz.

Příklad

Předpokládejme, že máme k dispozici CSV soubor (tj. textový soubor, ve kterém je uložena tabulka, buňky na řádku odděleny středníkem, lze otevřít v jednoduchém textovém editoru nebo v tabulkových editorech), ve kterém jsou uloženy údaje o objednávkách různých zařízení – předpokládejme, že jde o notebooky. Sloupce mají tento význam (nejsou v souboru přímo popsány, najdeme tam jen samotná data o objednávkách):

1. identifikátor, každý řádek má jiný (můžeme brát jako „číslování řádků“)
2. číslo objednávky; řádky vztahující se ke stejné objednávce zde mají stejnou hodnotu
3. název zařízení
4. počet objednaných kusů tohoto zařízení
5. dodavatel
6. datum objednávky
7. komentář k zařízení

Chceme zjistit *seznam zařízení, která jsou součástí objednávky číslo 152734*, vypíšeme název zařízení a počet objednaných kusů:

```
for /F "delims=; tokens=2-4" %a in (objednavky.csv) do (
    if "%a"=="152734" (echo %b: %c kusů ))
```

Vidíme, že hned za přepínačem `/F` je upřesnění formátu načítaného souboru (zadáváme, které části souboru má příkaz brát v úvazu a jakým způsobem). Zadali jsme oddělovač (středník) a pak sloupce, které má příkaz „vidět“ (tj. od druhého do čtvrtého sloupce, první nás nezajímá).

Následuje proměnná `%a`, která bude přiřazena prvnímu z „viditelných“ sloupců, tedy vlastně druhému sloupci ze souboru. Následující sloupec zpracovávaného řádku (třetí ze souboru) bude přístupný v proměnné `%b`, další v proměnné `%c`. Pozor, to funguje jen tehdy, pokud předtím použijeme klíčové slovo `tokens` pro určení sloupců.

Konstrukce `in (objednavky.csv)` určuje soubor, který má být zpracován, musí následovat klíčové slovo `do` a pak příkaz, který se má provést pro každý nalezený řádek zvlášť. Vnitřním příkazem je příkaz `if`, který otestuje, zda daný řádek chceme (patří k objednávce, která nás zajímá), a když ano, vypíše požadované údaje z tohoto řádku (název zařízení a počet objednaných kusů).

Všimněte si, že příkaz zabírá dva řádky. To není problém, jen je třeba dbát na to, abychom na konec řádku, která má mít pokračování, napsali levou závorku (pak Příkazový řádek automaticky počítá s tím, že příkaz není kompletní a pokračuje na dalším řádku).

Příklad

Opět budeme prohledávat soubor, který je popsán v předchozím příkladu. Zajímají nás *dodavatelé dodávající notebooky Lenovo řady IdeaPad*. Víme, že součástí názvu výrobku bude řetězec „IdeaPad“, tedy pro filtrování použijeme příkaz `find` (v podmínce příkazu `if` se zjišťuje pouze rovnost či nerovnost).

```
for /F "delims=; tokens=2-5" %a in (objednavky.csv) do (
    echo objednavka: %a, dodavatel: %d, zarizeni: %b | find /i "ideapad" )
```

Vypíšou se záznamy obsahující hledaný řetězec, ke každému nejdřív číslo objednávky, název dodavatele a pak název zařízení.

Úlohu bychom také mohli řešit příkazem `findstr` použitým přímo na soubor `objednavky.csv`, ale výše uvedeným způsobem si vybereme, které části řádku budou vypsány a které mají být ignorovány.

Při použití příkazu `for` v dávkovém souboru je třeba k proměnným přidat druhý symbol `%`, abychom odlišili tyto „vnitřní“ proměnné od jiných typů proměnných, především parametrů dávkového souboru.

Příklad

Chceme *přidat několik desítek (nebo stovek, to je jedno) nových uživatelů* (studentů, popř. zaměstnanců, brigádníků), ale nechce se nám to dělat ručně. Všichni mají být zařazeni do nové skupiny `2024`.

Seznam jsme dostali v souboru (sice ne textovém, ale text bez problémů vytáhneme). Předpokládejme, že jsme podle původního souboru vytvořili soubor `uzivatele2024.txt`, kde na každém řádku souboru je jeden student a pro něj vždy tři údaje – jako první je identifikační číslo, pak příjmení, a potom jméno. Všechny tři údaje na řádku jsou odděleny *mezerou*. Takže jeden řádek může vypadat například takto:

```
f123456 Novák Jan
```

Ve stejném adresáři, ve kterém máme soubor `uzivatele2024.txt`, vytvoříme dávkový soubor, který chceme volat (spouštět) se dvěma parametry takto:

```
pridejuziv.bat studenti2024 uzivatele2024.txt
```

První parametr je název nové skupiny, druhý parametr je název souboru se seznamem uživatelů.

V následujícím dávkovém souboru používáme příkazy `net user` (pro vytvoření nového uživatele) a `net localgroup` (pro vytvoření lokální skupiny na počítači a dále přidání uživatelů do této skupiny). S oběma těmito příkazy se podrobněji seznámíme v dalších kapitolách těchto skript.

Vytvoříme tedy dávkový soubor `pridejuziv.bat` s tímto obsahem:

```
@echo off
REM vytvorime skupinu uzivatelu s nazvem, ktery vezmeme z prvniho parametru:
net localgroup %1 /add
if errorlevel 1 goto :chyba

REM bereme v uvahu existenci tri "sloupcu",
REM na zpracovavanem radku je vzdy prvek z prvniho sloupce %%a, z druheho %%b, z tretiho %%c:
for /F "tokens=1-3" %%a in (%2) do (
  REM prvni prvek je prihlasovaci jmeno - id. cislo, druhe prijmeni, treti krestni jmeno:
  net user %%a %%a /add /fullname:"%%c %%b"
  net localgroup %1 %%a /add
)
goto :konec
:chyba
Echo Chyba při zpracování
:konec
```

Ve skutečnosti bychom nové uživatele přidávali spíše na serveru do domény (příkaz `net group` apod.), zde pro zjednodušení používáme lokální skupinu.

Vytvořený dávkový soubor pak můžeme používat každoročně, stačí jen zaměnit soubor s uživateli za nový a v parametru napsat název jiné skupiny. Studenti mají nastaveno heslo totožné se svým přihlašovacím jménem (to je identifikační číslo). Dále bychom měli správně přiřadit vhodná přístupová oprávnění (nové skupině, není třeba je přidělovat zvlášť všem novým uživatelům). To by také mohlo být součástí skriptu.

Pokud budeme chtít pro kontrolu vypsat všechna jména a příjmení (jen na Příkazovém řádku, ne v dávkovém souboru), zadáme:

```
for /F "tokens=2,3" %a in (uzivatele2024.txt) do @echo %b %a
```

Příklad

Textové soubory s uživateli přidanými v jednotlivých letech si můžeme archivovat a po uplynutí maximální doby studia pomocí obsahu souboru všechny uživatele z daného roku odstranit:

```
for /F "tokens=1-3" %a in (uzivatele2010.txt) do net user %a /delete
net localgroup studenti2024 /delete
```

Je zřejmé, že tyto příkazy jsme zadávali přímo na Příkazovém řádku, není nutné k tomu tvořit dávkový soubor (i když i to by šlo).

Na příkladech vidíme syntaxi – za parametrem `/F` je *řídicí řetězec*, který říká, co konkrétně ze souboru máme použít. Následuje *vnitřní proměnná*, do které bude dosazen první „oznámený“ prvek

na každém řádku (většinou chápán jako prvek v prvním sloupci, resp. v tom, který zadáme v rozsahu `tokens`). Dále zadáme název souboru, který se má zpracovávat, a zbytek už známe (příkaz).

Podle počtu deklarovaných sloupců (v části `tokens`) jsou vytvořeny další vnitřní proměnné (následující podle abecedy za tou první), přes ně pak přistupujeme k dalším prvkům na řádku. První je dostupný přes `%a` nebo `%%a`, druhý přes `%b` nebo `%%b`, atd.

Velmi důležitý je řídicí řetězec. V příkladech je `"tokens=1-3"`, to znamená, že zpracováváme první až třetí sloupec. Sloupce mohou být zadány také výčtem, některý můžeme i přeskočit (to má vliv na „přidělování“ položek řádku k vnitřním proměnným příkazu). Řídicí řetězec může vypadat i takto:

```
for /F "delims=: tokens=1,3-5" %%a ...
```

(oddělovačem položek na řádku je dvojtečka, zpracováváme pouze první a třetí až pátou položku na řádku, druhou a případně šestou přeskočíme). Všimněte si, že celý řídicí řetězec musí být v uvozovkách.

Příklad

Předpokládejme, že nakupujeme a prodáváme třeba automobily (pro vlastní účely si doplňte zboží podle svých zájmů – mobily, notebooky, květiny, knihy, atd.). Vytvoříme si soubor `auta.csv` (resp. jinak pojmenovaný podle toho, co máte uvnitř) s tímto obsahem:

```
1;154;Citroen;4
2;154;Renault;3
3;87;Wolkswagen;5
4;87;Audi;2
5;87;Skoda;7
6;121;Dacia;3
```

První sloupec je identifikátor, druhý číslo objednávky, třetí značka, čtvrtý sloupec je počet kusů v objednávce. Opět si skutečný obsah souboru upravte dle svých zájmů, případně jiný počet řádků. Soubor můžete vytvářet v kterémkoliv textovém editoru, třeba v Poznámkovém bloku ve Windows. Jak vidíme, oddělovačem buněk na řádku (tedy sloupců) je středník.

Naším úkolem bude vypsat seznam položek z objednávky č. 87, a to číslo řádku (ID), značku a počet kusů. Budeme pracovat přímo na příkazovém řádku, tedy u řídící proměnné cyklu použijeme jen jeden symbol procenta.

Příkaz můžeme zapsat třeba takto:

```
for /f "delims=; tokens=1-4" %a in (auta.csv) do if %b==87 echo znacka: %c, kusu: %d
```

Vpodstatě to funguje, ale problém je s echem (tedy odezvou) příkazů. Každý (pod)příkaz se nejdřív vypíše (i pro řádky, které ve skutečnosti nechceme vypsat), a teprve pak se provede. Lepší by bylo toto:

```
@for /f "delims=; tokens=1-4" %a in (auta.csv) do if %b==87 echo znacka: %c, kusu: %d
```

Ještě lepší bude přidat symbol zavináče i před příkaz `if`:

```
@for /f "delims=; tokens=1-4" %a in (auta.csv) do @if %b==87 echo znacka: %c, kusu: %d
```

Teď už by se nám opravdu mělo vypsat následující:

```
znacka: Wolkswagen, kusu: 5
znacka: Audi, kusu: 2
znacka: Skoda, kusu: 7
```

A co když chceme vypsat číslo řádku, na kterém je značka Dacia?

```
@for /f "delims=; tokens=1-4" %a in (auta.csv) do @if %c==Dacia echo radek: %a
```

V našem ukázkovém souboru nejsou mezery, tedy ani při porovnávání na ně nemusíme brát ohled, ale co kdyby tam byly? Mohou být, protože jako oddělovač používáme středník. Pak bychom museli hledaný řetězec uzavřít do uvozovek, ale pozor – i proměnnou, do které má být přiřazen:

```
@for /f "delims=; tokens=1-4" %a in (auta.csv) do @if "%c"=="Dacia" echo radek: %a
```

Možností tohoto typu cyklu je více, podrobnosti získáme zadáním `for /?` na Příkazovém řádku.

Úkoly

1. Projděte si pořádně příklady v této sekci – pro hromadné zpracování dat, podle okolností vyzkoušejte.
2. Podle prvního z těchto příkladů sestavte postup, kterým vytvoříte obdobné soubory pro celou sérii objednávek z rozsahu 140000–160000 (číslo objednávky bude součástí názvu souboru). Použijte jednu z forem příkazu `for`, která je blízká pojetí příkazu v klasickém programování (najdete v předchozím textu).

1.4 Ještě pár témat k PowerShellu

1.4.1 Zprostředkovatelé

V PowerShellu existují tzv. zprostředkovatelé, které si můžeme představit jako pracovní prostory, do kterých jsou směřovány určité příkazy.

Příklad

Chceme vypsat seznam zprostředkovatelů v PowerShellu:

```
PS C:\Users\uzivatel> $GET-PSProvider

Name            Capabilities                                    Drives
----            ------------                                    ------
Registry        ShouldProcess, Transactions                     {HKLM, HKCU}
Alias           ShouldProcess                                   {Alias}
Environment     ShouldProcess                                   {Env}
FileSystem      Filter, ShouldProcess, Credentials              {C, D}
Function        ShouldProcess                                   {Function}
Variable        ShouldProcess                                   {Variable}
```

Z názvů je patrné, o jaké pracovní prostory se jedná. Může být přítomen také prostor Certificate pro práci s certifikáty. Výchozím poskytovatelem je FileSystem, ale můžeme se pohybovat v registru Windows, v úložišti aliasů či funkcí nebo (vlastních) proměnných, případně v prostředí (kde najdeme seznam proměnných natahaných z registru). Do některých prostorů se lze přímo přesunout:

```
PS C:\Users\uzivatel> set-location env:
PS Env:\> set-location alias:
PS Alias:\> set-location c:
PS C:\Users\uzivatel>
```

Na promptu vidíme, kde zrovna jsme. Přesunuli jsme se nejdřív do prostoru Environment, pak Alias, a potom zpět do souborového systému. Všimněte si, že jsme použili řetězce z posledního sloupce předchozího výpisu (Drives).

V jednotlivých pracovních prostorech můžeme používat například příkaz `get-childitem`, popřípadě přidávat nové položky pomocí `new-item` nebo likvidovat pomocí `remove-item`.

S pracovním prostorem Function pracujeme nepřímo, například:

```
PS C:\Users\uzivatel> get-childitem function:

CommandType     Name                                Version    Source
-----------     ----                                -------    ------
Function        A:
Function        B:
Function        C:
Function        cd..
Function        cd\
Function        Clear-Host
Function        ConvertFrom-SddlString              3.1.0.0    Microsoft.PowerShell.Utility
Function        D:
Function        E:
Function        F:
Function        Format-Hex                          3.1.0.0    Microsoft.PowerShell.Utility
...

PS C:\Users\uzivatel> get-content function:\clear-host

$RawUI = $Host.UI.RawUI
$RawUI.CursorPosition = @{X=0;Y=0}
...
```

1.4.2 Větvení kódu a cykly

Pokud chceme uvést několik příkazů na jeden řádek a nechat je provést sekvenčně, oddělíme je středníkem, stejně jako v UNIXových systémech.

Také v PowerShellu fungují roury, jak už jsme ostatně zjistili. Můžeme taktéž používat směrování výstupu (operátory `>` a `>>`), včetně deskriptorů, kterých je dokonce k dispozici více než v Příkazovém řádku, tedy například chybový výstup směrujeme pomocí `2>` a `2>>`. Jen místo `nul` používáme „UNIXové“ `null`.

V PowerShellu máme k dispozici možnosti pro podmíněné vyhodnocování kódu včetně příkazů `if` a `switch`.

Příklad

Nejdřív zjistíme, kolik procesů má spuštěn Firefox (každý prohlížeč, i jiný než tento, má spuštěno poměrně dost procesů, takže to bude pravděpodobně číslo větší než 10). Výsledek uložíme do proměnné. Pak si pro orientaci vypíšeme, co v té proměnné vlastně je, a následně použijeme příkaz `if`.

```
PS C:\Users\uzivatel> $pocetProhlizecu=get-process firefox | measure
PS C:\Users\uzivatel> $pocetProhlizecu

Count    : 15
Average  :
Sum      :
Maximum  :
Minimum  :
Property :
```

```
PS C:\Users\uzivatel> if ($pocetProhlizecu.count -gt 10)
{ write-host -nonewline "Firefox má "
   write-host $pocetProhlizecu.count -nonewline ; write-host " procesů!"}

Firefox má 15 procesů!
```

Poznámka

Podmínky u podmíněných příkazů a také u cyklů zásadně uzavíráme do (kulatých) závorek. Tím dáváme najevo, že jde o výraz k vyhodnocení. Závorka u výrazů není nutná jen tehdy, když je výraz v roli samotného příkazu a píšeme ho přímo za prompt.

Tělo složeného příkazu (podmínky nebo cyklu) musí být uzavřeno do složených závorek.

V PowerShellu existuje několik typů cyklů, zde si ukážeme alespoň cyklus `while`.

Příklad

Necháme PowerShell vypočíst druhé mocniny čísel od 1 do 5:

```
PS C:\Users\uzivatel> $i=1

PS C:\Users\uzivatel> while ($i -le 5)
{ write-host ($i*$i) ; $i++ }

1
4
9
16
25
```

Všimněte si, že výraz s násobením je uzavřen v závorkách. To je nutné, jinak by byl tento výraz jako parametr příkazu `write-host` považován za řetězec, nebyl by vypočten.

Souhrn kapitoly 1

V této kapitole jsme navázali na témata probíraná v první části zimního semestru a rozšířili jsme si znalosti o práci v textovém režimu ve Windows. Věnovali jsme se jak Příkazovému řádku, tak i PowerShellu.

V Příkazovém řádku jsme se naučili vytvořit nový soubor s určitým obsahem, dále jsme se věnovali vyhledávání a využívání proměnných obsahujících číslo včetně výpočtů, a naučili jsme se psát skripty, zde tedy dávkové soubory.

V PowerShellu jsme si rozšířili znalosti o proměnných, naučili se psát funkce a využívat vlastnosti objektů pro ovlivnění výstupu příkazů, vysvětlili jsme si problematiku verzí a modulů a seznámili se s možnostmi skriptování v PowerShellu.

V další sekci jsme si podrobně vysvětlili využívání složených příkazů v Příkazovém řádku – propojování příkazů, podmíněné vyhodnocování, podmíněný příkaz a různé typy cyklů, také jsme si ukázali, jak využít cyklus `for /f` pro hromadné zpracování textových dat.

V poslední sekci jsme se zaměřili na pokročilejší témata ze světa PowerShellu: používání zprostředkovatelů (providers), větvení kódu a cykly.

Kapitola 2

Řízení přístupu a správa uživatelů a objekty ve Windows

Rychlý náhled: V této kapitole se budeme zabývat uživateli, objekty a řízením přístupu uživatelů k objektům. Nejdřív si projdeme celou sérií pojmů z oblasti řízení přístupu, pak se zaměříme na uživatele a skupiny a práci s nimi v systému, dále se budeme věnovat oprávněním. V následující sekci si ujasníme, co to vlastně jsou objekty ve Windows a kde se k nim dostaneme, nakonec po krátkém úvodu do Active Directory se zaměříme na bezpečnostní zásady a šablony ve Windows.

Klíčová slova: Důvěrnost, dostupnost, autenticita, nepopiratelnost, integrita, autentizace, SID, ACE, ACL, DACL, SACL, deskriptor zabezpečení, LSA, SAM, uživatelský účet a uživatelský profil, skupina, cestovní profil, účet Microsoft (MSA), objekt jádra, objekt exekutivy, manipulátor (handle) objektu, WinObj, GDI objekt, stanice oken, WinSta0, COM, komponenta, CLSID, GUID, IID, LIBID, COM+, DCOM, OLE, .NET, Active Directory, LDAP, adresářová databáze, AD schéma, doménový řadič, Globální katalog, zásady skupiny, šablony pro správu, šablony zabezpečení.

Cíle studia: Po prostudování této kapitoly byste se měli orientovat v problematice správy uživatelů a skupin ve Windows, chápat systém objektů ve Windows včetně řízení přístupu k nim, měli byste zvládnout problematiku bezpečnostních zásad ve Windows.

2.1 Základní pojmy související s oprávněními

Nejdřív několik pojmů:

Důvěrnost (Confidentality) některá data jsou označena jako „důvěrná“, tj. určená pouze pro vymezený okruh uživatelů (příp. procesů). Nikomu jinému by neměla být dostupná.

Dostupnost (Availability) uživatelům je zajištěn (bezproblémový) přístup ke všem datům, ke kterým jsou oprávněni přistupovat (tj. pokud má uživatel právo přistupovat k datům, je mu tento přístup umožněn).

Autenticita (Authenticity) je možnost ověření původu dat.

Nepopiratelnost (Non-repudiation) k možnosti ověření původu dat přidává vlastnost nemožnosti popření původu dat (původce nemůže popřít, že data vytvořil a poskytl).

Integrita (Integrity) data jsou správná a úplná, nebyla pozměněna (například při transportu přes síť), integrita dat může být narušena buď úmyslně (při transportu, malwarem apod.) nebo neúmyslně (hardwarovou nebo softwarovou chybou).

Autentizace je proces ověřování přístupových oprávnění uživatele k datům (obecně k objektu).

Autentizační protokol je protokol (tedy popis postupu) autentizace – jakým způsobem má probíhat komunikace během autentizace, typické příklady autentizačních protokolů jsou Kerberos nebo RADIUS.

SID (Security ID) je identifikátor zabezpečení, který jednoznačně (v lokální síti) identifikuje uživatele, skupinu, službu nebo počítač v síti, systém tento identifikátor přiřazuje při vytvoření uživatele (skupiny, služby, připojení počítače). Některé identifikátory SID jsou předdefinované.[1]

Příklad

Asi nejjednodušší způsob, jak zjistit SID některého uživatele nebo skupiny, je použití programu `PsGetSID` z balíku programů *PSTools* od firmy Sysinternals.

`psgetsid` vypíše SID *počítače*, například `S-1-5-21-000111222-3334445555-666777888` (místo sekvence `000...888` velmi pravděpodobně budou jiné číslice)

`psgetsid` *`uživatel`* vypíše SID zadaného uživatele, například `S-1-5-21-000111222-3334445555-666777888-1005` (místo sekvence `000...888` taktéž budou jiné číslice, ale stejné, jaké bychom dostali ve výstupu předchozího příkazu)

`psgetsid administrator` vypíše SID administrátora (lokálního počítače), například `S-1-5-21-000111222-3334445555-666777888-500`

`psgetsid guest` vypíše SID účtu *guest* (host), například `S-1-5-21-000111222-3334445555-666777888-501` (je zajímavé, že účty *Administrator* a *Guest* mají téměř stejné SID, liší se jen v poslední číslici)

`psgetsid everyone` SID skupiny *everyone*, tj. `S-1-1-0`

`psgetsid "local service"` SID účtu *Local Service*, tj. `S-1-5-19`, jedná se o účet pro služby pracující na počítači lokálně bez nutnosti autorizovaného přístupu na síť

`psgetsid "network service"` SID účtu *Network Service*, tj. `S-1-5-20`, pod tímto účtem pracují síťové služby

`psgetsid "nt authority\system"` SID účtu *System* (oficiálně NT Authority\System), tj. `S-1-5-18`,

Příkaz `psgetsid` může být použit také naopak, pro zjištění účtu určeného číslem SID, lze ho také využívat v síti (pracujeme s účty na vzdáleném počítači v síti).

Pokračujeme v pojmech:

ACE (Access Control Entry) je položka řízení přístupu, je to vlastnost, kterou lze uživateli (procesu) povolit nebo zakázat (například právo číst, zapisovat, otevřít složku, atd.).

ACL (Access Control List) seznam řízení přístupu, je to seznam položek ACE (přidělená oprávnění) pro jednotlivé uživatele či skupiny (představte si seznam uživatelů/skupin, a ke každému zvlášť seznam položek ACE). Jednotliví uživatelé a skupiny jsou stanoveni pomocí svých SID.

[1] Seznam obvyklých SID pro uživatele a skupiny najdeme na http://support.microsoft.com/kb/163846

Existují dva druhy seznamů ACL – DACL, u kterého jednotlivé položky ACE znamenají povolení daného typu přístupu k objektu, a SACL, u kterého jednotlivé položky ACE znamenají, že každý pokus o daný typ přístupu má být zaznamenán (auditován). Oba typy LSA jsou stejné co se týče datové reprezentace, ale každý má trochu jiný význam. Takže:

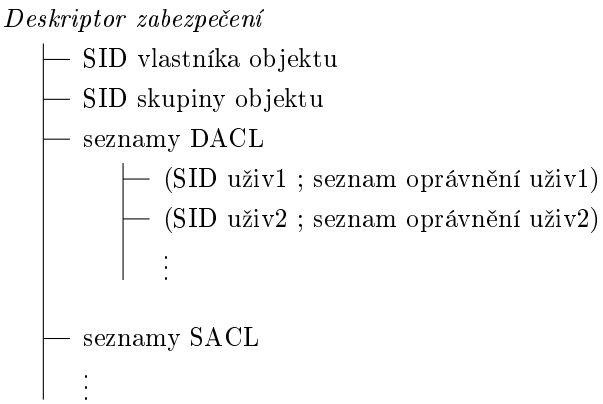
DACL (Discretionary ACL) volitelný seznam řízení přístupu, který určuje pro jednotlivé uživatele či skupiny jednotlivá přístupová oprávnění (položky ACE) k danému objektu. Při pokusu o přístup daného uživatele (či člena skupiny) určitého typu je důsledkem povolení nebo zakázání tohoto přístupu.

SACL (System Access Control List) auditovací seznam řízení přístupu, který taktéž u daného objektu obsahuje pro uživatele a skupiny seznam položek ACE. Při pokusu o přístup daného uživatele (či člena skupiny) určitého typu je důsledkem zaznamenání do bezpečnostního protokolu (ať už byl přístup povolen nebo ne).

Popisovač zabezpečení (Security Descriptor, bezpečnostní deskriptor) tuto strukturu má každý objekt, obsahuje všechny údaje důležité při přidělování oprávnění (SID vlastníka, SID skupiny, seznamy DACL, seznamy SACL, atd.), používá se při kontrole každého pokusu o přístup k danému objektu.

LSA (Local Security Authority, Místní úřad zabezpečení) je komponenta Windows (jedná se o soubor `lsass.exe`), která zajišťuje autorizaci (například ověřuje heslo při přihlášení uživatele nebo ověřuje přístupová oprávnění při přístupu k některému objektu).

Ve Windows (obvykle na diskovém oddílu se souborovým systémem NTFS) má každý objekt svůj popisovač zabezpečení, ve kterém je určen vlastník objektu, jeho skupina a přidružená přístupová oprávnění (DACL seznamy) pro různé uživatele a skupiny. Zjednodušenou strukturu deskriptoru zabezpečení vidíme na obrázku 2.1.



Obrázek 2.1: Zjednodušená struktura deskriptoru zabezpečení některého objektu

Příklad

Předpokládejme, že se uživatel *novak* pokouší otevřít soubor `xyz.txt` pro zápis.

- LSA nejdřív zjistí, zda v deskriptoru zabezpečení tohoto souboru existuje DACL přímo pro uživatele *novak*.
  - Pokud ano (existuje DACL pro uživatele *novak*) a je v něm položka ACE „w“, je otevření povoleno.
  - Pokud ano, ale položka „w“ v něm není, je otevření zamítnuto.
  - Pokud v deskriptoru vůbec neexistuje DACL pro uživatele *novak*, přecházíme k dalšímu bodu.

- LSA dále zjistí, jestli v deskriptoru zabezpečení existuje DACL některé skupiny, do které uživatel *novak* patří.
  - Pokud ano (existuje DACL pro některou takovou skupinu) a je v něm položka ACE „w“, je otevření povoleno.
  - Pokud ano, ale položka „w“ v něm není, je otevření zamítnuto.
  - Pokud v deskriptoru vůbec neexistuje DACL pro žádnou skupinu uživatele *novak*, je otevření definitivně zamítnuto.

Jinými slovy – LSA zkouší nejdřív uživatele a pak jeho skupiny, hledá položku ACE „w“ (právo zápisu). Pokud najde, už nehledá dál a přístup povolí (takže nastavení pro uživatele má přednost před nastavením pro jeho skupinu). Pokud nenajde, uživatel má smůlu.

Se seznamy DACL lze manipulovat také v grafickém rozhraní, a to ve vlastnostech objektu (souboru nebo jakéhokoliv jiného objektu), karta *Zabezpečení*.

Přihlašovací údaje jsou ověřovány

- moduly *SAM* (Security Accounts Manager) a LSASS, pokud jde o lokální účet,
- službou *Active Directory* na řadiči domény, pokud jde o doménový účet v síti.

SAM udržuje na počítači místní databázi zabezpečení obsahující údaje o zásadách pro účty, a to v souboru `secedit.sdb` ve složce `...\Windows\security\Database`.

Úkoly

1. Procvičte si to, co znáte z předchozího semestru – najděte v grafickém rozhraní (některém správci souborů, případně v Průzkumníku) ACL složky `Program Files`.
2. Pokuste se zjistit své SID. V registru zjistěte, čí SID jsou v klíči *HKU*.

2.2 Uživatelské profily, účty a skupiny

Uživatelský profil (User Profile) je souhrn prostředků přidělených uživateli a nastavení systému (většinou prostředí) typických pro tohoto uživatele. Obvykle zde bývá řazen datový prostor přidělený uživateli (standardně složka `Dokumenty`) a dále nastavení plochy, nabídky Start, individuálních položek kontextového menu Nový a Odeslat, Cookies, apod.

Celý uživatelský profil je obvykle načten ve složce `Users`, v podsložce každého uživatele.

Uživatelský účet (User Account) je jednoznačně určen svým SID a fyzicky implementován v registru (na rozdíl od profilu, který je vlastně adresářem/složkou), je to záznam obsahující převážně bezpečnostní informace určující konkrétního uživatele v operačním systému. Tyto informace jsou obvykle uživatelské jméno, heslo, členství ve skupinách, oprávnění k přístupu ke zdrojům systému (přístupová práva) a cesta k profilu uživatele.

Uživatelský účet může být buď lokální, tedy platný pouze na jednom počítači (pak jeho SID vychází z SID tohoto počítače), nebo cestovní definovaný obvykle v systému Active Directory (ale ne nutně). Na cestovní profily se podíváme o něco dále.

Služby se nemusí přihlašovat s přístupovými právy přihlášeného uživatele, většinou využívají účty *System*, *LocalService* a *NetworkService*. Službám se budeme věnovat později.

Skupina je definována svým názvem, seznamem členů a také oprávněními k přístupu ke zdrojům pro členy skupiny. Jeden uživatel může být členem více skupin. Také skupina je jednoznačně určena svým SID a může být definována buď lokálně na jednom počítači nebo v síti na serveru.

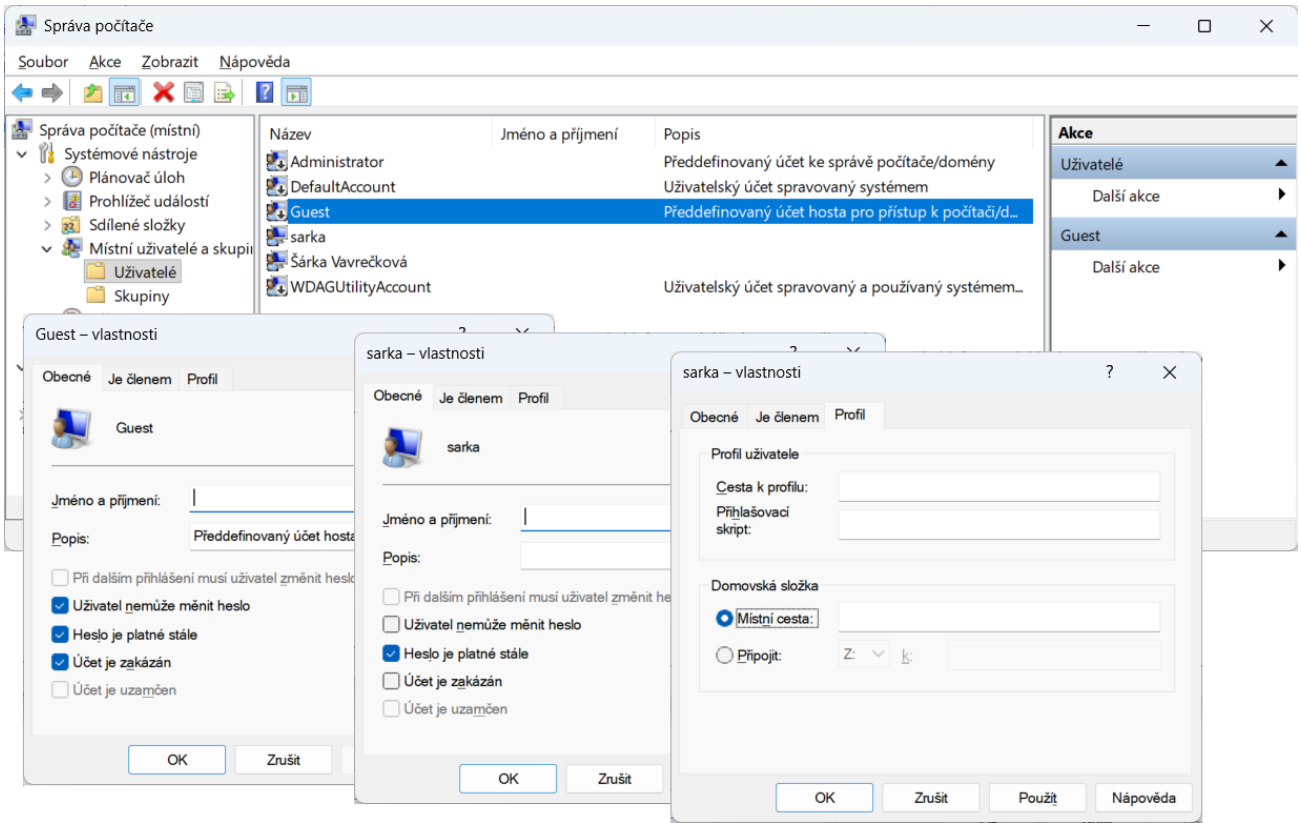
Cestovní profily (Roaming Profiles) umožňují uživatelům používat tentýž profil na více různých počítačích v téže síti. Veškerá data profilu jsou uložena na serveru. Po přihlášení k tomuto serveru (tj. přihlášení do sítě) je celý profil načten do počítače podobně, jakoby byl profil nadefinován na samotném počítači, po odhlášení se profil opět uloží na server.

Cestovní profily se využívají například ve školství, aby žáci/studenti nemuseli být vázáni na konkrétní počítač, a samozřejmě ve firmách (obvykle v rámci domény Active Directory), kde z bezpečnostních důvodů není vhodné nechávat obsah profilů zaměstnanců na serverech Microsoftu.

Nevýhodou je rozsáhlost dat, která při každém přihlášení a odhlášení putují po síti, může zpomalovat přihlašování a odhlašování a za určitých okolností může dojít k problémům při přenášení dat na server. Ve firemním prostředí se bezpečnost připojení (zejména pro cesty mimo firemní síť) řeší použitím VPN. Navíc bývá problém s kompatibilitou při přechodu na novou verzi systému.

Novou cestu k profilu lze určit v nástroji Místní uživatelé a skupiny (je dostupný také v konzole Správa počítače), jak vidíme v malém okně vpravo na obrázku 2.2. Kopírování profilu provádíme v nástroji *Nastavení* ⇨ *Systém* ⇨ *O systému* ⇨ *Upřesnit nastavení systému*, karta *Upřesnit*, část *Profily uživatelů*), tlačítko *Nastavení*, vybereme profil a klepneme na tlačítko *Kopírovat do*. Zobrazí se okno, ve kterém zadáme cíl (USB flash disk, na kterém profil přeneseme, nebo síť).

Profil můžeme kopírovat jen tehdy, pokud zrovna není používán, tedy nelze zkopírovat profil patřící „kopírujícímu“. Volba může být bohužel neaktivní i z jiných důvodů.



Obrázek 2.2: Vlastnosti uživatele v nástroji *Místní uživatelé a skupiny*

Účet Microsoft (MSA, Microsoft Account) je cloudový typ účtu, který je ve většině edic Windows od verze 10 povinný, třebaže existují cesty, jak to obejít. Může připomínat cestovní profil, hlavně tím, že data profilu jsou v cloudu (zde na serverech Microsoftu), a tím podobnost končí. U MSA dochází pouze k průběžné synchronizaci dat mezi zařízeními téhož uživatele a cloudem – není to tak, že by celý obsah profilu neustále cestoval. Uživatel se obvykle přihlašuje lokálně (přičemž se provede synchronizace případných změn provedených na jiném zařízení), jen výjimečně je nutno přihlásit se do cloudu.

Nastavení související s uživateli a skupinami provádíme v těchto nástrojích:

- v nástroji *Nastavení* ⇨ *Účty*,
- v nástroji *Uživatelské účty* (v *Ovládacích panelech*),
- v konzole *Správa počítače* ⇨ *Místní uživatelé a skupiny* (konzola `lusrmgr.msc`) – vytváření a rušení skupin, přidávání a odstraňování uživatelů ze skupin, zakázání či uzamčení účtu, vynucení změny hesla při dalším přihlášení, atd.,
- v *Místních zásadách zabezpečení* (`secpol.msc`), v *Zásadách skupiny* (`gpedit.msc`),
- v Příkazovém řádku pomocí `cacls.exe` a dalších.

Ne všechny tyto možnosti jsou dostupné ve všech verzích a edicích Windows, mimořádně osekané jsou například edice Home, kde nenajdeme konzolu *Místní uživatelé a skupiny* ani *Zásady skupiny*.

Úkoly

1. Najděte svůj profil a zjistěte, jak jsou k němu nastavena přístupová oprávnění pro vás a pro administrátora. Porovnejte s nastavením oprávnění u profilu `All Users` a `Default`. Ověřte si, ve které systémové proměnné je uložen název profilu právě přihlášeného uživatele.
2. Najděte svůj uživatelský účet v registru. Dále si vzpomeňte, kde v grafickém rozhraní lze konfigurovat uživatelské účty a kde může uživatel nastavit své heslo.

2.3 Správa uživatelů a skupin v Příkazovém řádku

Příkaz `NET` je komplexní nástroj pro práci především se sítí, ale taky je určen pro některé úkoly související se správou počítače a uživatelů (Windows řady NT).

Jeho první parametr obvykle určuje oblast, které se budou další parametry týkat, můžeme ho chápat jako vnořený příkaz. Zde použijeme klíčové slovo související s uživateli. Pro správu uživatelů a skupin slouží klíčová slova (podpříkazy) `user`, `localgroup`, `group` a `accounts` příkazu `net`.

Další informace

K jednotlivým variantám příkazu `NET` získáme krátkou nápovědu v textovém režimu (např. po zadání příkazu `net user /?` získáme stručný popis syntaxe).

2.3.1 Správa uživatelů

Nejdřív se tedy podíváme na příkaz `NET USER`. Tento příkaz nám umožňuje vypsat seznam uživatelů, zjistit informace o existujících uživatelích, přidávat nové uživatele, měnit jejich různá nastavení (včetně těch, ke kterým se v oknech nedostaneme nebo je to komplikované), aktivovat či deaktivovat účty, atd.

Uživatel si pomocí tohoto příkazu může změnit heslo (pokud je mu tato operace povolena, což se taky nastavuje pomocí tohoto příkazu).

Ukázky využití příkazu `NET USER`:

`net user` vypíše seznam uživatelů (v několika sloupcích)

`net user novak` vypíše podrobné informace o uživateli `novak`, včetně skupin, do kterých patří, údajů o hesle (jestli smí uživatel měnit heslo, kdy bylo nastaveno, kdy vyprší, apod.), kdy se uživatel naposledy přihlásil, zda je účet aktivní, atd.

`net user novak` *heslo* nastavíme uživateli `novak` zadané heslo

`net user novak *` jako předchozí, ale na heslo jsme dotázáni, při jeho zadávání se nezobrazují znaky (pro případ, že kolega vidí na naši obrazovku)

`net user administrator` *heslo* nastaví heslo administrátora na zadaný řetězec; takto zprovozníme účet administrátora pro nouzový režim i v edicích Home (také pro funkce, kdy je vyžadováno zadání hesla administrátora – pokud není definováno, k těmto funkcím se nedostaneme – samotné zaktivnění je probíráno o pár řádků níže); případně můžeme místo hesla napsat hvězdičku

`net user novak /active:no` zneaktivníme účet uživatele

`net user novak /active:yes` zaktivníme neaktivní účet uživatele, funguje taky na účet administrator (v novějších Windows bývá z bezpečnostních důvodů deaktivován a nelze se takto přihlásit)

`net user novak /passwordchg:no` od této chvíle zadaný uživatel nemá možnost změnit své heslo (toto nastavení je typické například pro účet *host/guest*)

`net user novak` *heslo* `/add /fullname:"Jan Novák"`
vytvoří nového uživatele se zadaným přihlašovacím jménem, heslem a zobrazovaným jménem

`net user novak * /add /fullname:"Jan Novák"`
totéž, ale na heslo jsme interaktivně dotázáni, při zadávání se nezobrazují znaky

`net user novak /scriptpath:`*souborskriptu* zadáme skript, který se má provést při každém přihlášení zadaného uživatele (cesta ke skriptu nesmí být absolutní, musí být relativní vzhledem k adresáři `...\System32\Repl\Import\Scripts`, ale může obsahovat také přechody do nadřízeného adresáře)

`net user novak /delete` odstraní uživatele

`net user novak /times:Po-Pá,6-16` umožní danému uživateli být přihlášen pouze v zadaných dnech a hodinách (zde v pracovních dnech mezi 6. a 16. hodinou, další možnosti najdeme v nápovědě).

Lze také použít parametr `/domain`, pak pracujeme s účtem na primárním řadiči domény (když tento parametr nezadáme, pracujeme s lokálním účtem).

Příkaz `net user` se dá využít také pro automatizaci přidávání nových uživatelů (údaje předem uložíme do souboru a pak je v cyklu načítáme – to jsme viděli v sekci 1.3.5 o hromadném zpracování dat na straně 27).

Poznámka

Tady vidíme hlavní výhodu nástrojů textového režimu – představte si, že máte v systému (třeba na serveru, doménovém řadiči apod.) vytvořit několik stovek nových uživatelů. Jak dlouho by to trvalo v nástroji *Uživatelské účty*? A co na to váš karpální tunel?

Další výhodou je, že tyto příkazy můžeme používat i vzdáleně a provádět konfiguraci systému, u kterého přímo nesedíme. K tomu ovšem potřebujeme přístupová oprávnění dostačující pro konfigurovaný počítač či síť.

Úkol

V Příkazovém řádku vypište seznam uživatelů. Jednoho z nich vyberte (třeba ten účet, pod kterým právě pracujete) a zobrazte si jeho vlastnosti. Pokud máte dostatečná přístupová oprávnění, vyzkoušejte si vytvoření nového uživatele a jeho odstranění, případně změnu parametrů tohoto účtu tak, jak je ukázáno výše (před jeho odstraněním).

Příklad

A jak je to s uživateli v PowerShellu? Seznam souvisejících příkazů a cmdletů získáme snadno:

```
PS C:\Users\uzivatel> get-command *user*

CommandType     Name                               Version    Source
-----------     ----                               -------    ------
Function        Set-PcsvDeviceUserPassword         1.0.0.0    PcsvDevice
Cmdlet          Disable-LocalUser                  1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          Enable-LocalUser                   1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          Get-LocalUser                      1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          Get-WinUserLanguageList            2.0.0.0    International
Cmdlet          New-LocalUser                      1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          New-WinUserLanguageList            2.0.0.0    International
Cmdlet          Remove-LocalUser                   1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          Rename-LocalUser                   1.0.0.0    Microsoft.PowerShell.LocalAccounts
Cmdlet          Restore-UevUserSetting             2.1.639.0  UEV
Cmdlet          Set-LocalUser                      1.0.0.0    Microsoft.PowerShell.LocalAccounts
...
Application     quser.exe                          10.0.19... C:\WINDOWS\system32\quser.exe
...
```

Zjevně jde o cmdlety obsahující podstatné jméno `LocalUser`, a pak nějaké další.

Poznámka

Pokud v novějších verzích Windows vytvoříme nového uživatele, jeho profil se vytvoří až po prvním přihlášení tohoto uživatele.

Úkoly

1. V Powershellu si zjistěte syntaxi cmdletů pro výpis seznamu uživatelů (tj. cmdlet `get-`), použijte ho a srovnejte výstup s příkazem `net user`. Dále zjistěte syntaxi cmdletu pro vytvoření nového uživatele.
2. Ve výstupu jste si určitě všimli, že existuje příkaz `quser`. Zjistěte, k čemu slouží (zobrazte si jeho nápovědu). Pozor, není to cmdlet, takže postupujte jako na Příkazovém řádku.

2.3.2 Správa uživatelských skupin

Pro správu uživatelských skupin máme příkaz `NET LOCALGROUP` (pro lokální uživatelské skupiny platné jen na počítači, na kterém jsou vytvořeny) a `NET GROUP` (pro skupiny existující na serveru, který je doménovým řadičem v Active Directory). Zde si ukážeme použití prvního uvedeného příkazu. Ovšem

oba tyto příkazy mají tytéž parametry, používají se stejně. Můžeme vytvořit či odstranit skupinu, a dále přidat do skupiny uživatele či ho odebrat.

V grafickém režimu bychom pro podobné úlohy použili nástroj *Místní uživatelé a skupiny*, v částech *Skupiny* (vytvoření či zrušení skupiny) nebo *Uživatelé* (členství uživatelů ve skupinách).

Ukázky práce s příkazem `NET LOCALGROUP`:

`net localgroup` vypíše seznam uživatelských skupin

`net localgroup users` vypíše informaci o skupině Users – komentář a seznam členů skupiny

`net localgroup "remote desktop users"` vypíše informaci o skupině Remote Desktop Users, což je skupina pro uživatele, kteří mohou v systému pracovat vzdáleně přes terminál (název skupiny obsahuje mezeru, proto musí být uzavřen do uvozovek)

`net localgroup uctarna /add` vytvoří novou skupinu se zadaným názvem (na lokálním počítači)

`net localgroup uctarna novak koutova /add` do skupiny *uctarna* přidá (předem vytvořené) uživatele *novak* a *koutova*, všimněte si, že parametr `/add` má u tohoto příkazu dvojí význam (vytvoření skupiny vs. přidání člena do skupiny)

`net localgroup uctarna marketing\zlatnik /add` do skupiny *uctarna* bude přidán uživatel *zlatnik* z domény *marketing*

`net localgroup` *`uctarna novak`* `/delete` odstraníme uživatele ze skupiny

`net localgroup` *`uctarna`* `/delete` zrušíme skupinu (dvojí význam parametru `/delete` – buď odstranění uživatele ze skupiny nebo likvidace skupiny)

Opět používáme parametr `/domain`, pokud chceme pracovat se skupinami na úrovni domény.

Příklad

Zobrazíme seznam uživatelů, zobrazilo se jich 6:

```
C:\> net user

Uživatelské účty pro \\DOMACIPC
-------------------------------------------------------------------------------
Administrator            DefaultAccount           Guest
Prvniuzivatel            Tretiuzivatel            WDAGUtilityAccount
Příkaz byl úspěšně dokončen.
```

Za účtem WDAGUtilityAccount stojí Windows Defender, najdeme ho ve Windows od verze 10. Vytvoříme novou skupinu a ověříme si, zda je v seznamu skupin, jsou podle abecedy:

```
C:\> net localgroup tiskpovolen /add
C:\> net localgroup

Aliasy pro \\DOMACIPC
-------------------------------------------------------------------------------
*Administrators
*Backup Operators
*Cryptographic Operators
...
*tiskpovolen
*Users
*Vlastníci zařízení
Příkaz byl úspěšně dokončen.
```

(výstup je zkrácen).

Do nové skupiny zařadíme dva uživatele a vypíšeme seznam členů skupiny:

```
C:\> net localgroup tiskpovolen Prvniuzivatel Tretiuzivatel /add
C:\> net localgroup tiskpovolen

Název aliasu     tiskpovolen
Komentář

Členové
-------------------------------------------------------------------------------
Prvniuzivatel
Tretiuzivatel
Příkaz byl úspěšně dokončen.
```

Tak teď jsme si to rozmysleli, novou skupinu odstraníme:

```
C:\> net localgroup tiskpovolen /delete
```

Příklad

Také v Powershellu existují cmdlety pro práci se skupinami. Jak je zjistíme?

```
PS C:\Users\uzivatel> get-command *group*
```

Úkoly

1. Vypište seznam uživatelů, kteří se mohou na počítači přihlásit. Najděte v seznamu své přihlašovací jméno a vypište informace o sobě (s využitím svého přihlašovacího jména).
2. Sestavte příkaz, kterým přidáte nového uživatele se zadáním jména, hesla, vypisovaného jména a parametru, který zajistí, že po uživateli nebude heslo vyžadováno (tento parametr najdete v nápovědě).
3. Vypište seznam skupin, které jsou vytvořeny na vašem počítači.
4. Pokud máte příslušná přístupová oprávnění, vytvořte nového uživatele s názvem *TestovaciUzivatel*, dále novou skupinu s názvem *PokusnaSkupina* a nového uživatele do ní zařaďte. Potom tohoto uživatele ze skupiny odstraňte a skupinu zrušte.
5. V PowerShellu si vypište seznam cmdletů pro práci s místními skupinami a u některých si ověřte syntaxi.

2.3.3 Konfigurace zásad souvisejících s účty

V minulém semestru jsme se dozvěděli, co to jsou zásady (policies, politiky) – předpisy v systému obvykle související se zabezpečením. Zde se podíváme, jak v textovém režimu pracujeme se zásadami dostupnými v grafickém režimu v *Místních zásadách zabezpečení*, položka *Zásady účtů*.

Používáme příkaz `NET ACCOUNTS`. Jedná se o bezpečnostní zásady platné pro všechny účty, například stanovíme minimální délku hesla, nutnost změnit heslo po určité době, můžeme zde zabránit tomu, aby při změně hesla uživatel jako nové heslo zvolil to, co už někdy dříve jako heslo používal – určujeme, kolik změn hesla v minulosti má být tako sledováno, atd.

Ukázky využití příkazu `NET ACCOUNTS`:

`net accounts` vypíše momentálně nadefinované vlastnosti uživatelských účtů

`net accounts /minpwlen:8` nastaví minimální požadovanou délku hesla uživatele na 8 znaků (ve více parametrech se vyskytuje zkratka „pw“, to znamená, že se vztahují k nastavení hesla – password)

`net accounts /maxpwage:120` heslo platí vždy nejvýše 120 dnů, po této době si uživatel musí zvolit nové heslo

`net accounts /maxpwage:unlimited` uživatel není nucen pravidelně měnit své heslo, jeho časová platnost není prakticky omezena

`net accounts /uniquepw:4` uživatel smí zvolit heslo, které už měl někdy v minulosti, ale až po nejméně 4 změnách hesla

`net accounts /maxpwage:30 /uniquepw:6` heslo platí vždy maximálně 30 dnů, uživatel (při této poměrně restriktivní době) smí zvolit i takové heslo, které už měl, ale až po nejméně 6 změnách hesla

V Powershellu neexistují přímo cmdlety pro lokální politiky hesel (tj. lokálně na daném počítači), existují až cmdlety na síťové úrovni (na serveru, v Active Directory).

Úkol

Zjistěte, jaké zásady jsou nastaveny pro uživatelské účty na vašem počítači. Především zkontrolujte, jaká musí být minimální délka hesla uživatele.

2.4 Přístupová oprávnění

V souborovém systému NTFS je u každého objektu (také složky a souboru) evidován seznam DACL s přístupovými oprávněními (angl. *credentials*). Je zde stanoveno, kdo (třeba člen konkrétní skupiny nebo uživatel) může provádět kterou akci s tímto objektem. O přístupových oprávněních už něco víme z předchozího semestru. Základní symboly pro přístupová oprávnění jsou

- f (full) – všechna práva,
- r (read) – právo čtení,
- w (write) – právo zápisu,
- c (change) – právo modifikace (změny),
- n (none) – žádná práva.

U složek navíc můžeme určit, kterých vnořených objektů se nastavená práva budou týkat (aktuální složky, podsložek, souborů). Dědičnost práv se určuje jako kombinace voleb

- (OI) – pro tuto složku a soubory v ní
- (CI) – pro tuto složku a její podsložky
- (IO) – neplatí pro tuto složku (tato volba „odebírá“ oblast platnosti)

Možné kombinace voleb pro ACL jsou v tabulce 2.1.

2.4.1 Nastavení oprávnění

V grafickém rozhraní je nastavujeme ve vlastnostech souboru na kartě *Zabezpečení*. Po klepnutí na tlačítko *Upřesnit* můžeme nastavovat speciální oprávnění specifická pro určitý typ objektu.

Příkaz `CACLS` slouží k nastavení přístupových práv souborového systému NTFS (pracuje pouze na logickém disku formátovaném jako NTFS), je součástí Windows od verze 2000.

Tabulka 2.1: Dědění přístupových oprávnění v ACL

Řetězec	Význam
prázdný řetězec	jen pro tuto složku, nedědí se
(OI)	pro tuto složku a soubory v ní
(CI)	pro tuto složku a její podsložky
(OI)(CI)	platí pro tuto složku, soubory v ní i její podsložky (plné dědění)
(OI)(CI)(IO)	platí pro soubory ve složce a její podsložky, ale ne pro samotnou složku
(OI)(IO)	platí jen pro soubory ve složce, ne pro složku ani podsložky
(CI)(IO)	platí jen pro podsložky, ne pro složku ani soubory v ní

`cacls soubor` výpis nadefinovaných přístupových práv souboru (ACL listy)

`cacls *` totéž, ale postupně pro všechny adresáře a soubory v pracovním adresáři (můžeme zadat také složitější výraz)

`cacls soubor /g patrik:r` k zadanému souboru přidělí uživateli s názvem patrik práva pro čtení (jakákoliv původní oprávnění budou přepsána)

`cacls * /e /g patrik:w` ke všem souborům v pracovní složce přidá uživateli patrik právo zápisu (ostatní práva zůstanou zachována)

`cacls *.doc /t /e /g patrik:w` ke všem DOC souborům z pracovní složky i z jejích podsložek (parametr `/t`, rekurze) přidá uživateli patrik práva pro zápis (ostatní práva zůstanou zachována)

`cacls abc.doc /r patrik` odebere uživateli patrik všechna přístupová práva k souboru `abc.doc`, která měl navíc oproti právům skupiny apod.

`cacls abc.doc /d patrik` zakáže uživateli patrik přístup k danému souboru

Ve výpisu přístupových práv (příkaz bez parametrů, například v kořenovém adresáři systémového disku) vidíme řetězce, které určují způsob rekurzivního přenášení práv na vnořené objekty (třeba podsložky).

Příklad

Pohrajeme si s přístupovými oprávněními. Přesuneme se na disk, na kterém máme právo zápisu a vytvoříme dva vnořené adresáře. Také vytvoříme nového uživatele a budeme mu přidávat nebo odebírat přístupová oprávnění.

`md pokus` vytvoříme první adresář

`cd pokus` přesuneme se do něj

`md vnitrni` vytvoříme podadresář

`cd..` přesuneme se o úroveň výše (všechny předchozí příkazy včetně tohoto lze provést jediným příkazem `md pokus\vnoreny`)

`net user novy novy /add` vytvoříme nového testovacího uživatele (jeho heslo je stejné jako jméno), po vytvoření by měl mít přístupová oprávnění spíše nižší (pravděpodobně skupina Users, záleží na verzi Windows)

`net user novy` pro jistotu si ověříme vlastnosti nově vytvořeného uživatele, všimněte si, do kterých skupin patří (tato informace je ve výpisu)

`cacls pokus` zobrazíme přístupová oprávnění nového adresáře, měl by tam být záznam pro skupinu everyone s právy `f` (full, vše)

`cacls pokus /e /d novy` novému uživateli zakážeme přístup k adresáři (pozor, nesmíme zapomenout parametr `/e`, aby se nepřepsal celý ACL – seznam řízení přístupu)

`cacls pokus /e /g novy:r` ale přidělíme mu právo čtení

`cacls pokus` ověříme ve výpisu, měl by tam být řádek pro nového uživatele s oprávněním „n“

`cacls pokus\vnoreny` podíváme se, jestli to nastavení mělo nějaký vliv na oprávnění u vnořeného adresáře – zjistíme, že ne, protože jsme nepoužili parametr pro rekurzi

`cacls pokus /e /r novy` teď pozor – mohlo by se stát, že práva ještě ubíráme, ale ve skutečnosti jsme ze seznamu oprávnění ACL zadaného adresáře vymazali údaje o novém uživateli, tedy v našem případě jsme vlastně odstranili záznam o zákazu přístupu

`cacls pokus` ověříme si výsledek předchozího příkazu

`cacls pokus /t /e /d novy` tento příkaz je podobný jako ten, který jsme použili pro zákaz přístupu novému uživateli, ale díky parametru `/t` je rekurzivní

`cacls pokus /t /e /g novy:r` rekurzivně mu povolíme čtení

`cacls pokus` ověříme si, teď by měl mít stejná oprávnění i v podadresáři

`cacls pokus /d novy` aha, zapomněli jsme parametr `/e`, takže tímto příkazem jsme přepsali ACL, můžeme si to ověřit zadáním `cacls pokus`

`cd pokus` teď by se mělo objevit hlášení „Přístup byl odepřen“, protože pro nás nejsou definována žádná oprávnění (je definováno pouze jedno, a to „Nepustit dovnitř uživatele *novy*“), do adresáře se nedostaneme

`cacls pokus /e /g everyone:f` napravíme chybu, kterou jsme provedli v předchozím příkazu, jinak bychom tento adresář nemohli ani smazat[2]

`rd /s pokus` uklidíme v adresářích – rekurzivně smažeme adresář i jeho podadresář

`net user novy /delete` uklidíme po sobě také v seznamu uživatelů

Příkaz `ICACLS` se využívá jak v novějších desktopových Windows, tak i na serverech. Některé parametry jsou podobné jako u `cacls` (například `/t` používáme pro rekurzi), ale obecně má odlišnou syntaxi. Například

```
icacls adresar /grant:r novy:r /t
```

přidá uživateli `novy` právo ke čtení (pozor – „r“ za `/grant` znamená přidání práv bez přepsání již definovaných, až druhé „r“ za uživatelem je právo čtení), poslední parametr znamená rekurzivní dědění.

Příklad

Nápověda příkazu `icacls` je celkem dlouhá, proto je dobré při jejím procházení použít stránkovací příkaz: `icacls /? | more`

Uživatele (vlastně i skupiny) je možné zadat nejen názvem, ale také řetězcem SID. Tento případ odlišujeme použitím hvězdičky před SID, například

```
icacls adresar /grant:r *S-1-5-27-697:r /t
```

Podobně se používá parametr `/deny`, kterým se práva odebírají.

[2] Uvědomte si, že se sice do adresáře nedostaneme, ale můžeme k němu definovat přístupová oprávnění.

Kromě běžných oprávnění lze pracovat také se speciálními oprávněními:

```
icacls soubor /grant *S-1-1-0:(d,wdac)
```

Skupině *Everyone* (ve výchozím nastavení má SID S-1-1-0) přiřazuje speciální práva pro mazání (právo D, delete) a dále možnost zapisovat do seznamů určujících přístupová oprávnění k zadanému objektu (souboru) – WDAC.[3]

Také můžeme podrobně určovat dědičnost:

```
icacls adresář /grant:r uzivatel:(OI)(CI)r /T
```

(určili jsme plnou dědičnost, můžeme používat jakoukoliv kombinaci z tabulky 2.1. Řetězec `(NP)` znamená, že se nemá použít žádné dědění (do not propagate inherit).

Tento příkaz má ještě další důležitou funkci – můžeme seznamy přístupových práv ukládat do souboru (textového) anebo ze souboru načítat:

```
icacls * /save souborACL.txt /t
icacls * /restore souborACL.txt /t
```

(první příkaz uloží nastavená oprávnění z pracovního adresáře a rekurzívně z celého jeho obsahu do zadaného souboru, druhý příkaz je opětovně načte).

Příklad

Pokud chceme použít Powershell, nabízí se cmdlet `get-acl`:

```
PS C:\Users\uzivatel> get-acl p:\objednavka_cartridge.pdf | fl

Path   : Microsoft.PowerShell.Core\FileSystem::P:\objednavka_cartridge.pdf
Owner  : O:S-1-5-21-1939040898-431487083-1465283063-1000
Group  : G:S-1-5-21-1939040898-431487083-1465283063-513
Access : BUILTIN\Administrators Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  FullControl
         NT AUTHORITY\Authenticated Users Allow  Modify, Synchronize
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
Audit  :
Sddl   : O:S-1-5-21-1939040898-431487083..... (zkráceno)
```

Pro změnu ACL k nějakému souboru bychom použili cmdlet `set-acl`, případně také v Powershellu jsou dostupné programy `cacls.exe` a `icacls.exe`.

Úkoly

1. Pokud na to budou stačit vaše přístupová oprávnění, vyzkoušejte si postupně všechny zde uvedené příkazy. Dbejte na bezpečnost – pokus provádějte na adresáři, který není důležitý (nejlépe nově vytvořený), a to s jiným uživatelským jménem, než pod kterým pracujete (nejlépe nově vytvořeným, stejně jako v příkladu). Nezapomeňte po sobě uklidit.
2. Vypište si nápovědu příkazu `cacls`. Projděte si všechny možné přepínače tohoto příkazu. Totéž udělejte pro příkaz `icacls`.
3. Sestavte příkaz (s použitím `cacls`), kterým uživateli *honza* přidáte právo pro zápis k adresáři `d:\faktury`, a to rekurzívně i pro podadresáře. Potom tentýž příkaz sestavte s použitím `icacls`.

[3] Seznam všech zkratek pro speciální oprávnění včetně WDAC najdeme na https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-R2-and-2012/cc753525(v=ws.11) .

4. Zobrazte si (stránkovanou) nápovědu příkazu `icacls`. Všimněte si, že obsahuje i příklady použití.
5. Vyzkoušejte cmdlet `get-acl` v PowerShellu na některý soubor ve svém profilu.

2.4.2 Navyšování přístupových oprávnění

Příkaz `RUNAS` slouží ke spuštění procesu s právy jiného uživatele, většinou administrátora (to znamená, že příkaz umožňuje navýšení – eskalaci – přístupových oprávnění). Základní syntaxe příkazu je

```
runas [/profile | /noprofile] /user:uživatel program
```

Příklad

Základní využití příkazu může být následující:

```
runas /user:administrator "net user novy heslo /add"
```
nebo
```
runas /user:pocitac\administrator "net user novy heslo /add"
```
nebo
```
runas /user:administrator@pocitac "net user novy heslo /add"
```

Lze zadat pouze takového uživatele, který má definováno heslo, tedy například pokud chceme použít účet `administrator` a nemáme u něj definováno heslo, máme smůlu. A samozřejmě by dotyčný účet měl být aktivní.

Přepínač `/profile` nebo `/noprofile` použijeme, pokud chceme či nechceme načíst profil přihlašovaného uživatele (*ne*načtení profilu sice zrychlí start programu, ale některé aplikace nemusí správně fungovat).

Mechanismus „runas“ je přizpůsoben i mechanismu UAC, tedy využívání úrovní integrity (z pohledu procesu), resp. důvěryhodnosti (z pohledu uživatele). Není přímo nutné zadávat uživatele, můžeme zadat úroveň důvěryhodnosti, na kterou se chceme posunout, a následovně budeme požádáni o heslo s touto úrovní související.

Příklad

Nejdřív si zobrazíme seznam dostupných úrovní důvěryhodnosti (na které se můžeme posunout):

```
runas /showtrustlevels
```

Zobrazí se seznam úrovní, například se tam může objevit `0x20000 (Standardní uživatel)`. Pokud bychom pracovali jako uživatel s nižšími oprávněními než standardní uživatel a chtěli spustit program na této úrovni, napsali bychom třeba:

```
runas /trustlevel:0x20000 "mmc c:\windows\system32\diskmgmt.msc"
```

Všimněte si, že v předchozím příkladu jsme nezadali přímo `diskmgmt.msc`, ale přidali jsme celou cestu k souboru a navíc spuštění procesu `mmc.exe`. Je to proto, že mechanismus navyšování oprávnění umí spouštět jen spustitelné soubory ve formátu PE (tedy například EXE soubory), nic jiného. Takže tomuto mechanismu musíme sdělit, pomocí čeho má dotyčnou konzolu spustit.

Ekvivalentem v grafickém rozhraní je zobrazení položky *Spustit jako* (v anglické verzi *Run as*) v kontextovém menu objektu (spustitelného souboru, to může být i skript), resp. *Spustit jako správce*. Pokud tato položka v kontextovém menu není, zobrazíme ji takto:

- stiskneme a dále držíme klávesu Shift ,
- zároveň obvyklým způsobem vyvoláme kontextové menu spustitelného souboru.

Nebo může být tato možnost dostupná přes položku *Zobrazit další možnosti*.

Pokud nám chybí původní možnost spouštět programy s přístupovými oprávněními různých uživatelů, můžeme si nainstalovat program *ShellRunAs*, který najdeme na stránkách http://sysinternals.com.

Poznámka

Pokud chceme delší dobu pracovat na Příkazovém řádku s oprávněními správce, spustíme s těmito oprávněními samotný Příkazový řádek a následovně vše, co na něm spustíme, bude mít k dispozici správcovská oprávnění.

Postup je jednoduchý – ve vyhledávání v nabídce Start napíšeme `cmd` (neklepeme na Enter), zobrazí se položka `cmd.exe`, na ni klepneme pravým tlačítkem myši a vybereme *Spustit jako správce*.

Ve Windows 11 24H2 se objevila podpora příkazu `sudo`, který známe z UNIXových operačních systémů. Jen je třeba zapnout podporu tohoto příkazu. To se dělá přes *Nastavení* ⇨ *Systém* ⇨ *Upřesnit* ⇨ *Povolit sudo*. Tam se dá sudo povolit, a taktéž je dobré určit, zda se má při použití otevřít nové okno nebo jestli má takto spuštěný příkaz převzít původní okno.

Další informace

https://learn.microsoft.com/en-us/windows/sudo/

2.5 Objektový model ve Windows řady NT

2.5.1 Objekty

Windows NT nejsou čistě objektovým operačním systémem, ale používají objekty pro reprezentaci systémových zdrojů (pozor, není to totéž co objekty v objektovém programování, zde jsou vlastnosti objektů pevně stanoveny). Systému to umožňuje pracovat se všemi objekty jednotným způsobem včetně zajištění bezpečnosti.

Z pohledu jádra systému se objekty dělí do dvou kategorií:

- *Objekty jádra* jsou jednoduché nízkoúrovnové objekty viditelné pouze v privilegovaném režimu (jenom pro jádro), tj. procesy běžící v uživatelském režimu s těmito objekty nemohou přímo pracovat.
- *Objekty exekutivy* (řídicího programu jádra) jsou tvořeny obvykle jedním nebo více objekty jádra, dalšími daty a rozhraním (přístupovými metodami), např.:
  - proces a podproces (vlákno procesu),
  - událost, párová událost (párová pro meziprocesovou komunikaci),
  - job (úloha) = skupina procesů sdílejících společné kvóty (stanovení maximálního využívání určitých prostředků), lze s nimi pracovat jako s celkem (například skupina úzce spolupracujících procesů),
  - sekce (úsek) = oblast sdílené paměti,
  - soubor (přesněji struktura určující otevřený soubor a umožňující k němu přístup),
  - port (přístupový bod pro volání procedur jiných procesů, umožňuje informovat jiný proces o dokončení I/O operace),
  - semafor (obecný semesfor: počitadlo pro regulaci počtu procesů sdílejících tentýž prostředek),

- mutex (pro realizaci vzájemného vyloučení procesů v kritických sekcích, vnitřně se nazývá mutant),
- adresář objektů (jakýsi kontejner),
- klíč registru,
- ukazatel na jiný objekt (symbolický odkaz, Symbolic Link),
- typ objektu, atd.

Všimněte si, že v seznamu objektů není uživatel ani skupina.

Procesy a systém ke svým vlastním objektům obvykle přistupují přes *manipulátor* (handle), který získají při vytvoření objektu (obvykle se jedná o návratovou hodnotu příslušné API funkce). Jde o nepřímý ukazatel, který (zjednodušeně) znamená index v tabulce manipulátorů pro daný proces. Tento manipulátor (zachycený do vhodné proměnné) procesy mohou používat jako parametr API funkcí objekt zpracovávajících.

K objektu lze také přistupovat přes jeho *název* (pokud ho známe), ale je to pomalejší než přes manipulátor (systém pokaždé překládá název na manipulátor). Název může být zpřístupněn i jinému procesu než je původní vlastník, používá se proto obvykle u sdílených objektů.

Jádro (včetně exekutivy) může k objektům přistupovat přímo (bez manipulátoru či názvu).

Hlavička (záhlaví) objektu obsahuje všechny důležité informace týkající se objektu, jako je

- jméno objektu (pokud ho má přiděleno),
- adresář objektu v hierarchické struktuře objektů,
- deskriptor zabezpečení s uvedením vlastníka, skupiny a přístupových oprávnění,
- režim, ve kterém je objekt používán (privilegovaný nebo uživatelský),
- ukazatel na objekt typu (objekty určitého typu mají některé vlastnosti společné, nemusí být ukládány u každého objektu zvlášť, tedy máme speciální objekty, které shrnují společné vlastnosti, které dědí jejich potomci), atd.

Každý objekt má svůj *bázový typ* a dále své atributy a metody. Například atributem objektu typu proces může být PID procesu, jeho priorita, ukazatel na bezpečnostní deskriptor, atd., k jeho metodám patří například metody `open` a `close`.

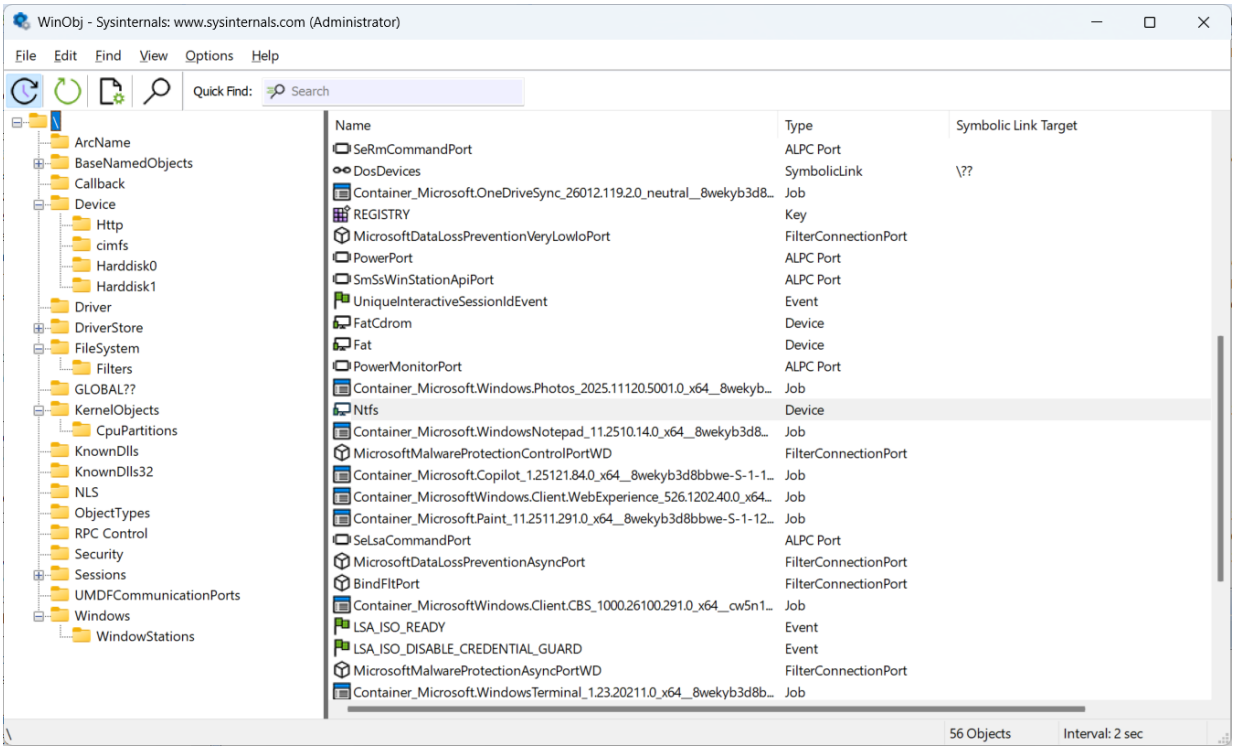
Poznámka

Pokud alespoň občas používáte API funkce Windows pracující s objekty (včetně oken, tlačítek, ikon, atd.), velice pravděpodobně jste se už s handly setkali. Handle se často používá jako parametr nebo návratová hodnota těchto funkcí, resp. používají se jeho definované podtypy: například HWND (handle na okno nebo jiný složitější objekt), HBITMAP (handle bitmapy, která je někde v prostředí vykreslená), HICON (handle ikony), HMENU (handle na menu okna), atd. – typicky začíná písmenem „H“.

Další informace

V následujícím textu budeme místy využívat nástroje, které nejsou součástí standardní instalace Windows. Většinou se jedná o nástroje firmy Sysinternals, které najdete na adrese https://sysinternals.com, vlevo odkaz *Utilities Index*. Z tohoto zdroje již dobře známe například *Process Explorer*.

Po zadání adresy bychom měli být přesměrováni na web Microsoftu – Sysinternals je součástí společnosti Microsoft.



Obrázek 2.3: Objekty v aplikaci *WinObj*

K objektům se lze dostat několika způsoby. Můžeme použít aplikaci *WinObj* od společnosti Sysinternals (obrázek 2.3), je třeba ji spustit jako správce. Dostaneme se pouze k obecným základním vlastnostem objektů (ne všech, ani správce se nedostane všude), například počtu manipulátorů (handle) na tento objekt a přístupovým oprávněním.

Dále mnohé aplikace pracující s procesy dokážou zobrazit seznam manipulátorů vlastněných procesem. Například v Process Exploreru se k těmto údajům dostaneme tak, že ve spodním podokně zobrazíme objekty (*View* ⇨ *Lower Pane View* ⇨ *Handles*). Od společnosti Sysinternals také pochází program `handle.exe`, který je přímo určen pro různé úlohy související s manipulátory.

Mnohem sofistikovanější a úplnější přístup k objektům nabízejí programy pro ladění jádra, například aplikace *WinDbg* (tyto postupy najdeme v příloze).

Správa objektů především spravuje *prostor jmen objektů* (také obor názvů) s názvy všech pojmenovaných objektů, přes který jsou objekty přístupné v různých procesech, tento prostor je globální a viditelný pro všechny procesy. Objekty, které nemají být viditelné pro jiné procesy, zde nesmí být, a proto ani nemají jméno.

Objekty jsou organizovány v hierarchické struktuře v paměti systému, v tzv. *adresářové struktuře objektů* (adresář objektů je také objekt). Tuto strukturu vidíme (alespoň částečně) na obrázku 2.3 v aplikaci *WinObj*. Strukturu objektů udržuje komponenta nazvaná *Správce objektů* (Object Manager).

Pouze objekty z adresářů `\BaseNamedObjects` a `\GLOBAL??` jsou viditelné procesům v uživatelském režimu. První z nich obsahuje objekty událostí, objekty synchronizačních mechanismů (mutexy a semafory), časovače a objekty sdílené paměti, tedy vše potřebné ke komunikaci procesů navzájem nebo procesů s jádrem. Druhý obsahuje objekty odkazů na zařízení (většina z nich odkazuje na objekty zařízení v adresáři `\Devices`) s poněkud srozumitelnějšími názvy než objekty, na které odkazují (také se označují jako Dos Devices).

Každý pojmenovaný (viditelný) objekt je zařazen ve stromové struktuře objektů a je adresován cestou od kořene této struktury (kořen je označen \). Například

`\KnownDLLs\gdi32.dll`
: je objekt typu „section“ (to je obecně oblast v paměti), zde konkrétně to je odkaz na místo v paměti, do které je namapovaná knihovna `gdi32.dll`,

`\Driver\pci`
: je objekt typu „Driver“ (ovladač), konkrétně ovladač sběrnice PCI a PCIe,

`\Driver\kbdclass, \Driver\kbdhid`
: jsou taky objekty typu „Driver“, konkrétně ovladače pro klávesnici (dílčí, spolupracují spolu),

`\Driver\Null`
: je ovladač nulového portu, v uživatelském režimu ho známe jako zařízení `NUL` na Příkazovém řádku (to konkrétně najdeme jako objekt typu „Symbolic Link“ – symbolický odkaz – ve větvi `GLOBAL??`),

`\ObjectTypes\Process`
: je objekt typu „typ objektu“ (určuje typ objektu proces),

`\Registry\System\CurrentControlSet`
: je objekt typu „klíč registru“, a to *`HKLM/System/CurrentControlSet`*,

`\Device\HardDiskVolume1`
: je objekt reprezentující první oddíl na prvním pevném disku (je typu Device, tedy zařízení),

`\Device\HardDisk0\Partition1`
: je odkaz (symlink) na předchozí, tedy nepřímo odkazuje na první oddíl na prvním pevném disku,

`\Device\HardDisk0\Partition0`
: je odkaz na objekt `\Device\HardDisk0\DR0`, který reprezentuje MBR sektor na prvním pevném disku.

Zatím víme, že se k objektům dostaneme buď ve WinObj nebo v Process Exploreru (jeho spodní podokno). Od Sysinternals existuje i další nástroj pro práci s objekty, tentokrát pro Příkazový řádek – `handle.exe`. Dá se taktéž stáhnout na stránce od Sysinternals.

Příklad

Stáhneme `handle.exe` a tento program budeme používat v Příkazovém řádku spuštěném s oprávněním správce (to většinou nebude nutné, ale např. přístup k vlastnostem procesu `winlogon` by jinak nebyl možný).

```
D:\programy\sysinternals>handle -p winlogon.exe

Nthandle v5.0 - Handle viewer
Copyright (C) 1997-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

------------------------------------------------------------------------------
winlogon.exe pid: 13160 NT AUTHORITY\SYSTEM
   54: File  (RW-)   C:\Windows\System32
  2A4: File  (R-D)   C:\Program Files\WindowsApps\Microsoft.LanguageExperiencePackcs-CZ_2....\user32.dll.mui
  2D4: Section       \Sessions\7\Windows\ThemeSection
  404: Section       \Windows\Theme34052611
...
```

Získáme seznam manipulátorů vlastněných daným procesem (přepínač `-p` umožňuje zadat název nebo PID procesu), výstup (zkrácený).
Získali jsme seznam objektů používaných procesem `winlogon.exe`. Jak vidíme, u objektu typu soubor (typ `File`) bývá souhrn požadavků na typ přístupu (většinou pro čtení a zápis). Každý řádek začíná číslem manipulátoru (hexadecimálně) a jeho typem, následuje jméno. Záhlaví s názvem a autorem programu `handle` již nebudeme do dalších výpisů zařazovat, i když ve skutečnosti se v nich objevuje.

Zadáme `handle -p 24288` (místo názvu procesu zadáme jeho PID, toto PID je právě přiřazeno procesu notepad.exe), výstup (opět zkrácený):

```
D:\programy\sysinternals>handle -p 24288
....
   58: File  (RW-)   C:\Windows\System32
   A0: File  (RW-)   C:\Windows\WinSxS\amd64_microsoft.windows.gdiplus_6595b64144ccf1df_1.1....
   C0: File  (RW-)   C:\Windows\WinSxS\amd64_microsoft.windows.common-controls_6595b64144ccf1df_....
  2E8: Section       \BaseNamedObjects\__ComCatalogCache__
....
  6C4: File  (R-D)   C:\Program Files\WindowsApps\Microsoft.WindowsNotepad_11.2510.14.0_x64__....
....
```

Pokud bychom zadávali proces názvem, a zároveň by existovalo víc procesů se stejným názvem, vypsaly by se údaje o všech takových procesech.

V každém případě – zatím se nám vypisovaly pouze některé objekty, většinou typu File, Section nebo objekty z kontejneru `\BaseNamedObjects`. Kdybychom chtěli vypsat opravdu všechny objekty vlastněné procesem s PID 24288, musíme použít také přepínač `-a`:

```
D:\programy\sysinternals>handle -p 24288 -a
....
    4: Event
    8: Event
    C: Key           HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
   10: Event
   14: WaitCompletionPacket
....
   58: File  (RW-)   C:\Windows\System32
   5C: SchedulerSharedData
....
   70: Directory     \Sessions\7\BaseNamedObjects
   74: Semaphore     \Sessions\7\BaseNamedObjects\SM0:24288:304:WilStaging_02_p0
....
```

Výstup by byl velmi dlouhý, běžné aplikace s rozhraním vlastní velmi mnoho objektů. V seznamu vidíme nějaké soubory, události, adresáře, synchronizační objekt typu semafor a další.

Pokud nás zajímá jen počet objektů jednotlivých typů (a máme spuštěný Poznámkový blok), potřebujeme přepínač `-s`.

```
D:\programy\sysinternals>handle -p notepad.exe -s
....

Handle type summary:
  ALPC Port       : 37
  Composition     : 25
  Desktop         : 2
  Directory       : 4
  DxgkSharedResource: 4
  EtwRegistration : 354
  Event           : 330
  File            : 39
  IoCompletion    : 18
  IoCompletionReserve: 9
  IRTimer         : 8
  Key             : 60
  Mutant          : 22
  SchedulerSharedData: 2
  Section         : 60
  Semaphore       : 40
  Thread          : 33
  Timer           : 3
  TpWorkerFactory : 4
  WaitCompletionPacket: 73
  WindowStation   : 4
Total handles: 1131
```

Ovšem nesmíme zapomenout na to, že takto se počítají pouze manipulátory, ale dva manipulátory mohou odkazovat na tentýž objekt (například u tohoto výpisu dva manipulátory typu *WindowStation* ve skutečnosti odkazují na tutéž stanici oken).

Pokud chceme vědět, který proces ve své tabulce obsahuje handle konkrétního pojmenovaného objektu, třeba souboru (a tedy chceme vědět, který proces má otevřený tento soubor), zadáme například

```
D:\programy\sysinternals>handle -a 06_ExtendedACL.docx
....
WINWORD.EXE        pid: 3440   type: File     1150: D:\vyuka\sitemgr\cviceni2025\06_ExtendedACL.docx
```

Na tentýž objekt může odkazovat více handlů v různých procesech, například pokud chceme vědět, které procesy pracují s naším podregistrem (větví registru začínající *HKCU*, zadáme `handle -a hkcu`. Výpis bývá většinou velmi dlouhý, je vhodné ho přesměrovat do souboru (mezipaměť příkazového řádku na delší výpisy nestačí).

Program `Handle` (nebo jiný program umožňující pracovat s manipulátory) se může hodit také například tehdy, když chcete pracovat s některým souborem (či jiným objektem), ale tento soubor je uzamknut (vlastněn) jiným procesem.

Postup

Ukážeme si způsob zjištění vlastníka souboru a jeho „uvolnění“. V předchozím příkladu jsme zjistili, že program WINWORD.EXE s PID 3440 vlastní objekt typu soubor `06_ExtendedACL.docx`, jehož handle je 1150. Kdybychom chtěli s tímto souborem cokoliv provést (například ho smazat nebo třeba vypnout počítač), objeví se chybové hlášení.

Pokud budeme chtít tento objekt uvolnit, potřebujeme výše zmíněné identifikátory (PID procesu a handle objektu). Obojí zadáme do následujícího příkazu, potřebujeme přepínač `-c`:

```
D:\programy\sysinternals>handle -c 1150 -p 3440
....
 1150: File  (R-)   D:\vyuka\sitemgr\cviceni2025\06_ExtendedACL.docx
Close handle 1150 in WINWORD.EXE (PID 3440)? (y/n) y

Handle closed.
```

Byli jsme dotázáni, zda opravdu chceme handle uvolnit. Souhlasili jsme a handle (objekt) byl uvolněn. Z hlediska uživatele pracujícího se souborem ve Wordu se prakticky nic neprojeví, ale soubor bude zpřístupněn. Jen může dojít k synchronizačnímu problému při vícenásobném přístupu k objektu.

Předchozí postup je užitečný například tehdy, když aplikace z nějakého důvodu při svém ukončení neuvolní své manipulátory (například při nekorektním ukončení nebo „zamrzne“).

Úkoly

1. Stáhněte si a vyzkoušejte aplikaci *WinObj*. Projděte si názvy objektů související s pevnými disky a souborovými systémy, podívejte se na objekty jádra. Projděte si adresáře pojmenovaných objektů, které jsou viditelné i v uživatelském režimu.
2. V aplikaci Process Explorer zapněte zobrazení spodního podokna a nastavte v něm zobrazování manipulátorů (handlů). V podokně pro manipulátory zobrazte sloupec *File Share Flags*. Vyzkoušejte na některém z procesů, u kterých máte dostačující přístupová oprávnění.

3. Pokud máte dostatečná oprávnění, podívejte se, jaké objekty vlastní procesy `system`, `smss.exe` a některé další. U objektů typu File (soubor) si všimněte příznaků (sloupec *Share Flags*). Zjistěte, který z těchto procesů je vlastníkem hlavního adresáře objektů pro registr (`\REGISTRY`).
4. Pokud máte možnost, vyzkoušejte program `handle` (podle výše uvedených příkladů).

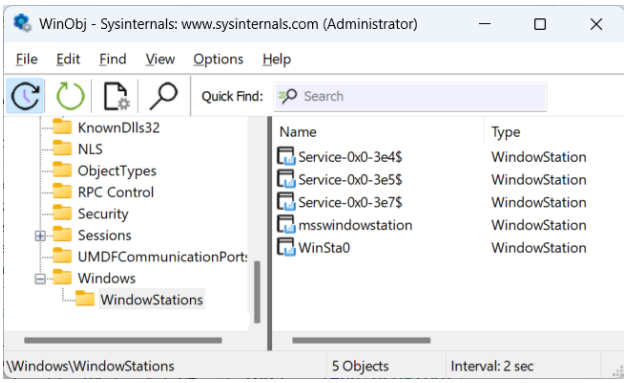
2.5.2 GDI objekty a stanice oken

Z hlediska určení můžeme objekty rozdělit do tří skupin – objekty jádra, GDI objekty a uživatelské objekty. GDI objekty souvisejí s grafickým rozhraním. Typickými zástupci jsou font, bitmapa, štětec, pero, paleta, oblast na obrazovce, grafická metadata.

GDI objekty nejsou nativně chráněny tak jako objekty jádra. Jsou vždy soukromé pro konkrétní proces (procesy nemohou používat GDI objekty jiných procesů) a navíc k jednomu GDI objektu může existovat vždy jen jeden manipulátor.

Protože GDI objekty (a další objekty na nich založené) obecně zabírají hodně systémových zdrojů (hlavně paměti), je stanovena horní hranice pro jejich počet. Tato hranice je odlišná u různých verzí Windows a navíc může být změněna, najdeme ji v registru v klíči *`HKLM/SOFTWARE/Microsoft/Windows NT/CurrentVersion/WindowsGDIProcessHandleQuota`*.

Stanice oken (Window Stations) jsou objekty jádra určené k dodatečné ochraně uživatelských objektů (GDI objekty tuto ochranu nepotřebují, díky výše uvedeným omezením). Každá stanice oken má kromě jiného přiřazenu jednu *schránku* a jeden nebo více objektů typu *plocha* (desktop). Stanice oken najdeme v adresáři objektů `\Windows\WindowStations`, jehož obsah vidíme na obrázku 2.4.



Obrázek 2.4: Objekty stanic oken

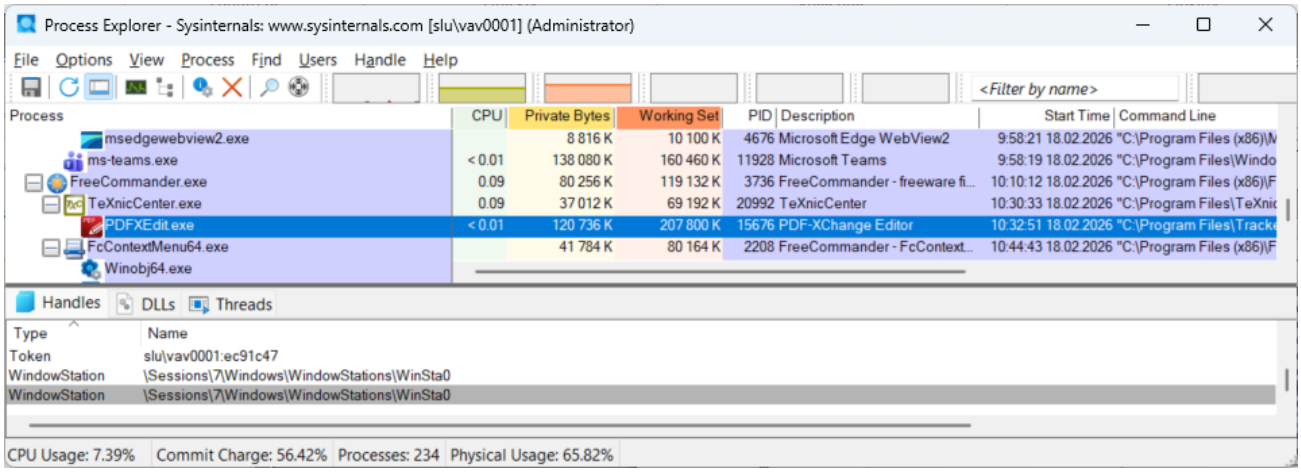
Z hlediska uživatele je nejdůležitější stanicí oken objekt `WinSta0`, která jako jediná dokáže zobrazit uživatelské prostředí a zajišťovat komunikaci s uživatelem, je určena pro interaktivní procesy a služby (většina služeb je neinteraktivní, ale některé vyžadují možnost interakce s uživatelem a tedy zařazení do `WinSta0`). Při vytvoření interaktivního procesu je tomuto procesu automaticky vytvořeno okno na výchozí ploše stanice `WinSta0` a proces může komunikovat s jinými procesy přes schránku stanice.

Dále existuje ještě několik stanic oken pojmenovaných `Service-0x0-`⟨*hex číslo*⟩`$`, které jsou určeny pro systémové neinteraktivní procesy nepoužívající okno. Hexadecimální číslo v názvu je identifikátor přihlašovací relace. Jsou určeny pro neinteraktivní služby a jiné procesy se speciálními přístupovými oprávněními, například `Service-0x0-3e7$` je stanice oken pro neinteraktivní systémové procesy běžící pod účtem s vysokými přístupovými oprávněními na lokálním počítači (účet *Local System*).

Pokud služba běží pod konkrétním uživatelským účtem (tj. žádným ze speciálních účtů pro služby), její proces je připojen do stanice oken, jejíž název je částečně odvozen z SID uživatelského účtu, pod kterým běží (horní část SID se použije jako hexadecimální číslo v názvu, například `Service-0x0-63a8f$`). Je zřejmé, že taková služba nemůže být interaktivní (protože není napojena na stanici oken `WinSta0`).

Počet GDI objektů a stanic oken je možné v Process Exploreru zobrazit jako další sloupec (klepneme pravým tlačítkem myši na záhlaví kteréhokoliv sloupce, zvolíme *Select Columns* a na záložce

Process Memory zaškrtneme volby pro tyto objekty). Seznam těchto objektů je ve spodním podokně, pokud máme nastaveno zobrazování manipulátorů (handle). Také v tomto podokně můžeme vybírat zobrazené sloupce (v kontextovém menu některého sloupce zvolíme *Select Columns*.



Obrázek 2.5: Objekty *Window Station* procesu PDFXEdit.exe

Na obrázku 2.5 vidíme seznam objektů vlastněných procesem PDFXEdit.exe, tedy PDF-XChange Editor.

Příklad

Výstup příkazu `handle -a windowstations\winsta0` je následující (zkrácený):

```
....
spoolsv.exe       pid: 11740  type: WindowStation   C8: \Windows\WindowStations\WinSta0
spoolsv.exe       pid: 11740  type: WindowStation  398: \Windows\WindowStations\WinSta0
winlogon.exe      pid: 13160  type: WindowStation  1E0: \Sessions\7\Windows\WindowStations\WinSta0
winlogon.exe      pid: 13160  type: WindowStation  1F4: \Sessions\7\Windows\WindowStations\WinSta0
dwm.exe           pid: 2536   type: WindowStation  150: \Sessions\7\Windows\WindowStations\WinSta0
dwm.exe           pid: 2536   type: WindowStation  158: \Sessions\7\Windows\WindowStations\WinSta0
....
PDFXEdit.exe      pid: 15676  type: WindowStation   C0: \Sessions\7\Windows\WindowStations\WinSta0
PDFXEdit.exe      pid: 15676  type: WindowStation   C8: \Sessions\7\Windows\WindowStations\WinSta0
....
procexp64.exe     pid: 1396   type: WindowStation  1B8: \Sessions\7\Windows\WindowStations\WinSta0
procexp64.exe     pid: 1396   type: WindowStation  1C0: \Sessions\7\Windows\WindowStations\WinSta0
....
handle64.exe      pid: 23940  type: WindowStation   D4: \Sessions\7\Windows\WindowStations\WinSta0
handle64.exe      pid: 23940  type: WindowStation   DC: \Sessions\7\Windows\WindowStations\WinSta0
```

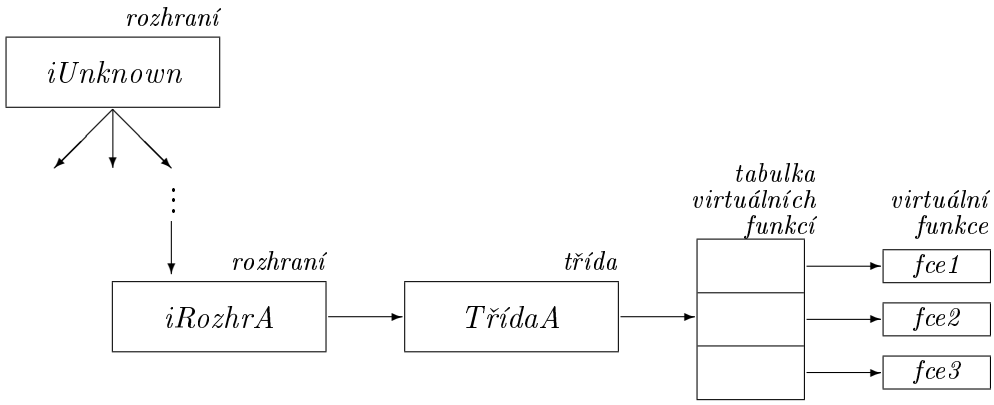
Získali jsme seznam procesů, které jsou napojeny do stanice oken `WinSta0` (mohli jsme zadat jen `winsta0`, ale tento podřetězec se vyskytuje v názvech některých dalších objektů). Podobně bychom zjistili seznam procesů napojených do kterékoliv jiné stanice oken.

2.5.3 Model COM

COM (Component Object Model) je specifikace určující formu, vlastnosti a způsoby přístupu k objektům Windows při programování (není to programovací jazyk). Objekty tohoto modelu se nazývají *komponenty*, odtud název modelu.

Tento model byl navržen, aby bylo možné vytváření binárních (tj. s přeloženým kódem) *komponent* (to jsou ty cílové objekty) napsaných v libovolném jazyce pro využití opět v kterémkoliv programovacím jazyce. Můžeme říci, že jde o specifikaci binárního rozhraní objektových komponent přístupného nejen v rámci jednoho procesu, ale i v jiných procesech, než ve kterém běží. Účelem je také umožnění relativně snadné komunikace mezi procesy.

Základem celého modelu je strom definovaných *rozhraní* (interface), všechna rozhraní jsou potomky (v rámci dědičnosti) rozhraní *iUnknown* (názvy rozhraní tradičně začínají písmenem „i“). Struktura rozhraní je naznačena v levé části obrázku 2.6. Rozhraní určuje činnosti – metody, funkce, nikoliv data.



Obrázek 2.6: Vztah tříd a rozhraní v modelu COM

Samotné objekty (tedy komponenty, resp. třídy) jsou pak definovány na základě těchto rozhraní. Jedna komponenta může být implementací jednoho nebo více rozhraní, tedy k metodám daného (daných) rozhraní přidává konkrétní data. Vnitřní datové struktury komponenty nejsou přístupné, proto nedochází k problémům běžným u klasických dynamických knihoven.

Poznámka

Nenechte se zmást – Microsoft poněkud převrací terminologii, kterou známe z běžných programovacích jazyků. Zatímco jinde máme hierarchii tříd, tady máme hierarchii rozhraní. Zatímco jinde vytváříme objekt na základě třídy, zde vytváříme objekt (komponentu, třídu) na základě rozhraní.

Protože na totéž rozhraní se mohou napojit různé komponenty s různými daty, musí být nějakým způsobem zajištěna přístupnost (kompatibilita) metod rozhraní pro daný případ (danou komponentu). Proto je přístupnost metod definovaných rozhraním fyzicky řešena jako ukazatel na *tabulku virtuálních funkcí*, která obsahuje ukazatele na konkrétní implementace metod dané komponenty.

Komponenty (třídy) jsou identifikovány podle čísla CLSID (Class ID), které jsme často viděli například v registru. CLSID je vlastně typ označení odpovídající označením, která souhrnně nazýváme GUID (Globally Unique ID).

Existuje tedy několik typů GUID:

- CLSID (Class ID) identifikuje třídu,
- IID (Interface ID) identifikuje rozhraní,
- LIBID (Library ID) identifikuje knihovnu, atd.

Aby GUID bylo opravdu unikátní (jednoznačné), musí být vygenerováno programem pro tento účel určeným (můžeme stanovit nějaké GUID i ručně, ale nemáme zaručeno, že například u zákazníka takové GUID neexistuje. Programátoři obvykle používají program `Guidgen.exe` (nebo jiný podobný).

Poznámka

Hodně systémových funkcí je implementováno jako rozhraní COM, na této technologii jsou založeny technologie OLE, OCX, ActiveX a částečně .NET Framework. Také z ní vychází rozhraní WMI, kterému se budeme věnovat v jedné z dalších kapitol.

Velmi důležitou výhodou technologie COM je univerzálnost (COM komponenty lze použít prakticky v jakémkoliv moderním jazyce i po jejich přeložení, je definováno jejich binární rozhraní) a dále vlastní správa paměti (obvykle se nemusíme starat o uvolnění alokované paměti). Tento model ale má také nevýhody, a to

- komplikovanost návrhu a používání,
- problémy s bezpečností při programování internetových aplikací (komponenty mají sice svá data skrytá, ale samy mohou volně přistupovat všude tam, kam má přístup proces, v jehož kontextu běží), atd.

Nejen z důvodu těchto nevýhod bylo navrženo (a používá se) několik rozšíření:

- *COM+* je rozšíření původního COM o nové vlastnosti, například byl vylepšen systém přístupových oprávnění (oprávnění založená na rolích) a byla přidána podpora COM+ Events (událostí, které lze využívat mezi procesy),
- *DCOM* (Distributed COM) je rozšíření možnosti komunikace mezi objekty i na různých počítačích (podpora distribuovaných aplikací), toto rozšíření bylo původně součástí COM+.

Komponenty najdeme i v registru. Třídy jsou v klíči *HKCR/CLSID* (v podklíčích nazvaných podle jejich CLSID) a rozhraní komponent v *HKCR/interface* (v podklíčích nazvaných podle jejich IID). V obou případech vždy najdeme v položce *(Výchozí)* název třídy (resp. rozhraní). V podklíčích jsou pak další informace. Ovšem víme, že klíč *HKCR* je pouze odkazem jinam, tedy ve skutečnosti se nacházejí v klíči *HKLM*.

Definice tříd (typů) se vytváří v jazyce MIDL (Microsoft Interface Definition Language) a dál se překládá do konkrétního jazyka, ve kterém programujeme aplikace. Programujeme obvykle v jazycích C++, Delphi, Visual Basic, C# (a dalších .NET jazycích), asi nejobvyklejší programovací prostředí pro COM je MS Visual Studio (resp. Visual Studio.NET). V každém případě pro zvolený jazyk musí existovat kompilátor jazyka MIDL.

Speciálním typem COM objektů jsou *Automation Objects*, implementace potomků rozhraní *iDispatch* (to znamená, že Automation Server je COM komponenta, která implementuje rozhraní iDispatch). Původně byly nazývány OLE Automation, ale nejsou používány pouze v technologii OLE. Účelem používání Automation je zjednodušení přístupu ke komponentě. Rozhraní iDispatch umožňuje zjistit metody nabízené komponentou bez nutnosti vyhledávání přes tabulku virtuálních metod.

Jsou určeny k použití ve skriptovacích jazycích pod Windows (zvláště založených na Visual Basicu). Aplikace (Automation Server) může nabízet (exportovat) svůj Automation Object ke sdílení, a jiná aplikace nazvaná *Automation Controller* (klient), s tímto objektem může manipulovat.

Automation Objects s vizuálním rozhraním, které vždy běží uvnitř některé aplikace, nazýváme *ActiveX Controls* (ovládací prvky ActiveX) a manipulující aplikace je *ActiveX Client*. Typické použití

je při distribuovaném spouštění drobných aplikací či prvků grafického rozhraní (to jsou právě ovládací prvky ActiveX) ze serveru na klientských počítačích v síti. Prvky ActiveX jsou tedy Automation objects, které kromě rozhraní iDispatch implementují ještě některá další rozhraní související právě s grafickým rozhraním komponenty.

Zkratka *ADO* znamená ActiveX Data Objects, technologie ADO byla původně založena na prvcích ActiveX (podobně *ASP* – Active Server Pages).

Další známý typ objektů jsou OLE objekty, implementující rozhraní *iOleObject*. Konkrétní rozhraní implementované některým OLE objektem se nazývá *OLE kontejner*.

Úkoly

1. Najděte v registru rozhraní *IUnknown* a zjistěte jeho IID.
2. V klíči `HKCR/icmfile/shellex/ContextMenuHandlers` zkopírujte název některého podklíče obsahující CLSID (na klíči zobrazte kontextové menu a vyberte *Zkopírovat název klíče*). Pak se pokuste v klíči `HKCR/CLSID` najít podklíč se stejným názvem a zjistěte, jaké jsou v jeho podklíči položky. Podobně můžete vyhledávat i další klíče nazvané podle CLSID souvisejících tříd komponent (nebo IID rozhraní).

2.5.4 .NET

Technologie .NET (čteme [dotnet]) vznikla roku 2000, používá se od roku 2001/02. Bývá považována za nástupce technologie COM, ale obě technologie existují „vedle sebe“.

Jedná se o standard pro běhové prostředí (ekvivalent podsystému ve Windows) podobně jako například Java, s vlastní správou paměti. Je definováno rozhraní, které (výlučně) slouží ke komunikaci .NET aplikace se systémem. Podobně jako v Javě, .NET aplikace jsou překládány do *bytecode* (mezikódu), který je nezávislý na softwarové platformě (operačním systému) a teprve při spuštění aplikace dochází k překladu bytecode do kódu specifického pro daný systém (aplikace zcela přeložená pro běh v konkrétním systému se nazývá *assembly*).

S touto technologií se běžně setkáváme od Windows XP SP2 a Server 2003 pod názvem .NET Framework, od verze Vista je dokonce velmi úzce spojena s grafickým rozhraním systému (podsystém *WinFX* pro .NET aplikace včetně grafického rozhraní samotných Windows, v tomto podsystému běží například také *PowerShell*).

.NET technologie je prezentována jako multiplatformní, tedy dostupná pro kterýkoliv operační systém, i když ze začátku tím „kterýkoliv“ byl míněn kterýkoliv systém typu Windows. Dnes již existují porty pro další operační systémy. Přímo na MSDN[4] jsou už nějakou dobu dostupné porty pro FreeBSD a MacOS, na internetu je několik řešení pro Linux (MONO, dotGNU – Portable.NET).

Programujeme v tzv. .NET jazycích, což je především C#, ale také Visual Basic.NET, Delphi.NET, J#, F#, IronPython, IronRuby, atd.

Od .NET jsou odvozeny technologie ADO.NET (potomek starší technologie ADO, pro přístup k databázím) a ASP.NET (pro programování webových aplikací).

[4] MSDN (Microsoft Developer Network) je část serveru Microsoftu dostupná na jeho stránkách, obsahuje nástroje a dokumenty pro podporu programátorů.

Další informace

Seznam všech nástrojů od Microsoftu pro práci s .NET je na
https://learn.microsoft.com/en-us/dotnet/framework/tools/.

2.6 Objekty, zásady a šablony

2.6.1 Krátký úvod do Active Directory

Adresářové služby slouží ke správě a zabezpečení prostředků (objektů) řazených v určité struktuře (která může připomínat strukturu adresářů v souborových systémech). Jedná se o služby běžící v lokální síti, přičemž existuje databáze objektů, ke které taková služba poskytuje zabezpečený přístup. Na adresářové službě často staví i možnost autentizace na zařízeních v síti.

Co se týče evidovaných objektů, typicky se jedná o zařízení (počítače, tiskárny, servery, atd.) a uživatele. Ke každému objektu najdeme v *adresářové databázi* související informace, včetně určení přístupových oprávnění k danému objektu.

Většina adresářových služeb je implementací protokolu *LDAP* (Lightweight Directory Access Protocol), který právě popisuje zacházení s objekty na adresářovém serveru. Je to odlehčená varianta staršího (a velmi komplexního, těžko implementovatelného) protokolu DAP (jednoho z protokolů rodiny X.500).

Existuje víc různých adresářových služeb, z nejznámějších:

- Microsoft Active Directory je jednou z nejrozšířenějších implementací protokolu LDAP, používá se především v sítích s Windows servery,
- OpenLDAP je volně šiřitelná implementace protokolu LDAP, do určité míry kompatibilní s Active Directory,
- NDS/eDirectory je adresářová služba v sítích Novell.

Active Directory je implementace protokolu LDAP (Lightweight Directory Access Protocol) pro Windows od verze 2000. Jedná se o systém pro evidenci, správu a zabezpečení objektů ve firemní síti. Používáme tyto pojmy:

- *adresářová databáze* (adresář) je databáze objektů, které jsou v systému spravovány, je hierarchicky uspořádaná (takže adresář je vlastně o objektech a vztazích mezi nimi),
- *objekty* mohou být například uživatelé, skupiny, počítače, domény, apod., každý objekt má své vlastnosti (například přístupová oprávnění), tyto vlastnosti se v hierarchické struktuře mohou dědit,
- *kontejner* je objekt, který může obsahovat další objekty (obdoba složek na disku),
- *AD schéma* popisuje objekty, které mohou být uloženy v adresáři Active Directory (jaké mohou mít atributy, co v nich může být uloženo), můžeme si ho představit jako definici typu objektu nebo třeba jako záhlaví tabulky (kde máme informace o tom, jaký typ informací najdeme v jednotlivých sloupcích tabulky),
- *doména* je skupina počítačů sdílejících společnou adresářovou databázi,
- *organizační jednotka* (OU) je podskupina domény (ale ne jakékoliv) oddělená za určitým účelem (například firma může mít jedinou doménu a tu rozčlení na organizační jednotky podle svých oddělení). OU mohou být i vnořené.

V síti je Active Directory provozován na *doménových řadičích* (domain controller, v podstatě jde o doménové servery), musí existovat alespoň jeden (primární řadič domény) a případně další (doporučuje se mít alespoň dva).

V každé síti je *Globální katalog* (obvykle na primárním řadiči domény). V Globálním katalogu jsou především souhrny informací obsažených v dalších doménových serverech sítě, hovoříme také o *replikaci* (dynamickém vytváření kopií).

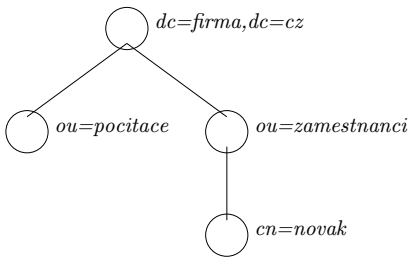
Globální katalogy slouží při vyhledávání informací v síti a také k autentizaci (uživatel se při přihlašování k počítači vlastně z technického hlediska přihlašuje ke globálnímu katalogu, ve kterém je daný počítač zařazen) a autorizaci (jsou mu udělena oprávnění, podle záznamu o něm jako o uživateli). Takže přes doménové řadiče s globálními katalogy přistupujeme k objektům a také se na nich provádí autentizace (kontrola při přihlašování), autorizace a obecně kontrola oprávnění při přístupu k objektům v síti.

V Active Directory se používá několik druhů názvů podle typu zanoření v doménách. Jsou to především *Domain Component* (DC, uzel domény), *Organization Unit* (OU, organizační jednotka, to je Active Directory Container, obdoba složky) a *Common Name* (CN, objekt). Adresace (popis cesty k objektu) podle struktury na obrázku 2.7 je

```
cn=novak,ou=zamestnanci,dc=firma,dc=cz
```

Tento způsob adresace objektu se označuje *DN* (Distinguished Name). Další způsob adresace, UNC, známe z DNS názvů:

```
firma.cz/zamestnanci/novak
```



Obrázek 2.7: Názvy v doménách

V serverových verzích Windows máme k dispozici nástroje pro práci s Active Directory několik nástrojů. Nebudeme je zde probírat, je to spíše záležitost počítačových sítí. Tyto nástroje mají obvykle ve svém názvu podřetězec „Active Directory“ či zkratku „AD“, například pro správu uživatelů, skupin, počítačů a organizačních jednotek používáme nástroj *Uživatelé a počítače služby AD* (Active Directory Users and Computers). Další nástroje jsou dostupné na instalačním CD Windows Server a také na internetu. Zajímavý a užitečný nástroj je například *Active Directory Explorer* od Sysinternals.

Na desktopu obvykle služba Active Directory není nainstalována, pracujeme zde pouze se *zásadami* (politikami), a to v nástrojích *Místní zásady zabezpečení* a *Místní zásady skupiny* (Group Policies Editor `gpedit.msc`).

Dnes jsou běžné heterogenní sítě (tj. na počítačích v síti jsou různé typy operačních systémů. Může se zdát, že používání mechanismu Active Directory v heterogenní síti je problém, ale řešení existuje, spočívá v použití jakýchsi „překladatelů“ – protokolů, které zprostředkují komunikaci mezi počítači s různými operačními systémy. V případě použití Active Directory jde především o protokol OpenLDAP implementovaný také na jiných operačních systémech včetně Linuxu, dále pro přístup k datům se používá protokol SMB.

Další informace

Informace o základních principech a postupech v Active Directory najdeme na

- https://www.samuraj-cz.com/clanek/active-directory-komponenty-domain-tree-forest-site/
- https://www.mcmcse.com/microsoft/guides/ad.shtml

2.6.2 Zásady skupiny

Skupinové zásady (zásady skupiny) ve Windows (včetně serverů) slouží k podrobnější konfiguraci systému a pracovního prostředí uživatele, pro své fungování v síti potřebuje službu Active Directory. Se zásadami jsme se seznámili v minulém semestru, zde si je jen připomeneme.

Se zásadami skupiny lze pracovat lokálně pomocí konzoly *Editor místních zásad skupiny* (spustíme v menu *Start* ⇨ *Spustit*, zadáme `gpedit.msc`, obrázek 2.8), ale i v lokální síti. Obvyklé použití je v kombinaci se službou Active Directory, kdy k jednotlivým objektům této služby (zařízením, doménám, uživatelům, apod.) přidružujeme *objekty Zásad skupiny* s určením způsobu zacházení s objektem, na který jsou navázány.

Nástroj je rozdělen do dvou částí – *Konfigurace počítače* a *Konfigurace uživatele*. Nastavení provedená v první části platí obecně pro počítač a všechny uživatele, v druhé již konkrétně pro určitého uživatele. Obrazem těchto dvou částí jsou některé podklíče klíčů *HKLM* a *HKCU* v registru. Pokud jsou stejné typy položek v obou, pak přednost mají nastavení v druhé části. Pokud jsou stejné položky v *Místních zásadách zabezpečení* (`secpol.msc`), mají přednost nastavení v *Zásadách skupiny*.

Poznámka

Hierarchie platnosti na konkrétním počítači je tedy následující:

- zásady v Active Directory,
- zásady v nástroji *Editor místních zásad skupiny* (`gpedit.msc`), větev *Konfigurace počítače*,
- zásady v nástroji *Editor místních zásad skupiny* (`gpedit.msc`) větev *Konfigurace uživatele*,
- zásady v nástroji *Místní zásady zabezpečení* (`secpol.msc`).

Nepleťte si poslední dva nástroje – každý z nich je určen k něčemu trochu jinému, třebaže to, co lze nastavit v *Místních zásadách zabezpečení*, je obsaženo i v *Editoru místních zásad skupiny* (část *Konfigurace počítače* ⇨ *Nastavení Windows* ⇨ *Nastavení zabezpečení*).

K práci se *Zásadami skupiny* slouží také programy `gpupdate.exe` (změny provedené v `gpedit.msc` se někdy neprojeví okamžitě; pokud chceme, aby se změny provedly hned, spustíme `gpupdate.exe`) a `gpresult.exe` (tento program prověří, zda nastavení provedená v *Zásadách skupiny* platí).

Úkol

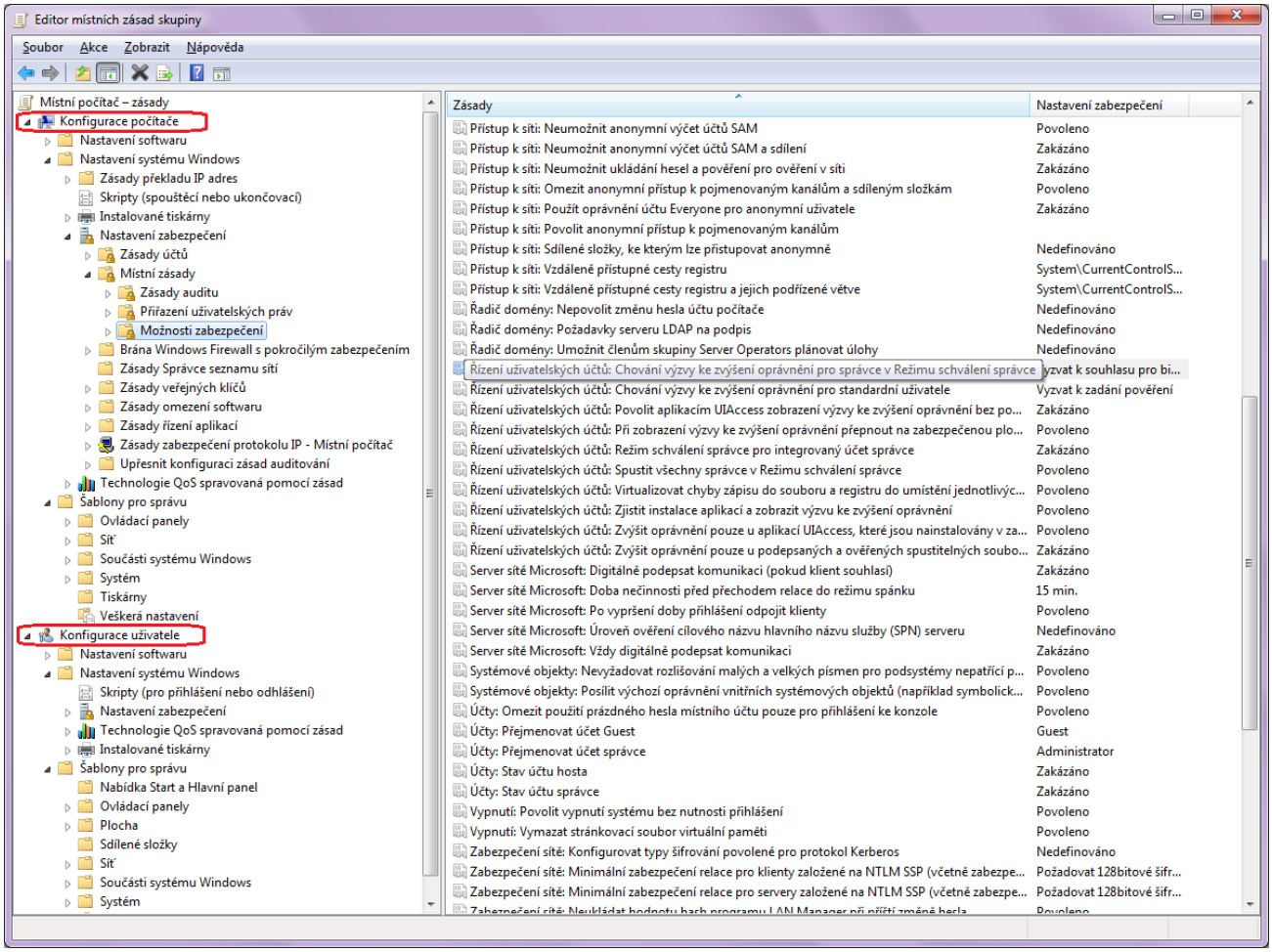
Pokud máte možnost, projděte si pro zopakování konzolu *Editor místních zásad skupiny*.

2.6.3 Šablony pro správu

Šablony pro správu jsou mechanismus pro komplexní přístup k zásadám uloženým v registru. Vlastně jsme se s nimi už letmo setkali v minulém semestru, když jsme probírali registr. Přistupujeme k nim v nástroji *Editor místních zásad skupiny* (`gpedit.msc`), kde v obou základních větvích najdeme položku *Šablony pro správu*. Šablony ve větvi Konfigurace počítače se vztahují k části registru uložené v klíči *HKLM*, šablony ve větvi Konfigurace uživatele se vztahují k části registru pro konkrétního uživatele (*HKCU*).

Z toho, že používáme množné číslo, vyplývá, že těchto šablon je víc. Například máme šablony

- *Ovládací panely* (Control Panel) pro konfiguraci různých nastavení obsažených v Ovládacích panelech (například můžeme znepřístupnit konkrétní nástroj v Ovládacích panelech nebo dokonce



Obrázek 2.8: Nástroj *Editor místních zásad skupiny* `gpedit.msc`

celé Ovládací panely, zakázat některé konkrétní nastavení, třeba změnu motivu, zakázat změnu jazykového nastavení apod.),

- *Plocha* (Desktop) pro konfiguraci možností souvisejících s plochou Windows (například můžeme zakázat změnu tapety na ploše nebo třeba odebrat položku *Vlastnosti* z kontextového menu ikony *Počítač*),
- *Tiskárny* (Printers) pro konfiguraci přístupu k tiskárnám v rámci Active Directory,
- Systém (System) pro nastavení různých parametrů systému, přičemž rozsáhlejší je šablona v části Konfigurace počítače (například zde pracujeme s diskovými kvótami, můžeme zakázat používání výměnných zařízení typu USB flash disk nebo určovat, jak se lze do systému přihlásit),...

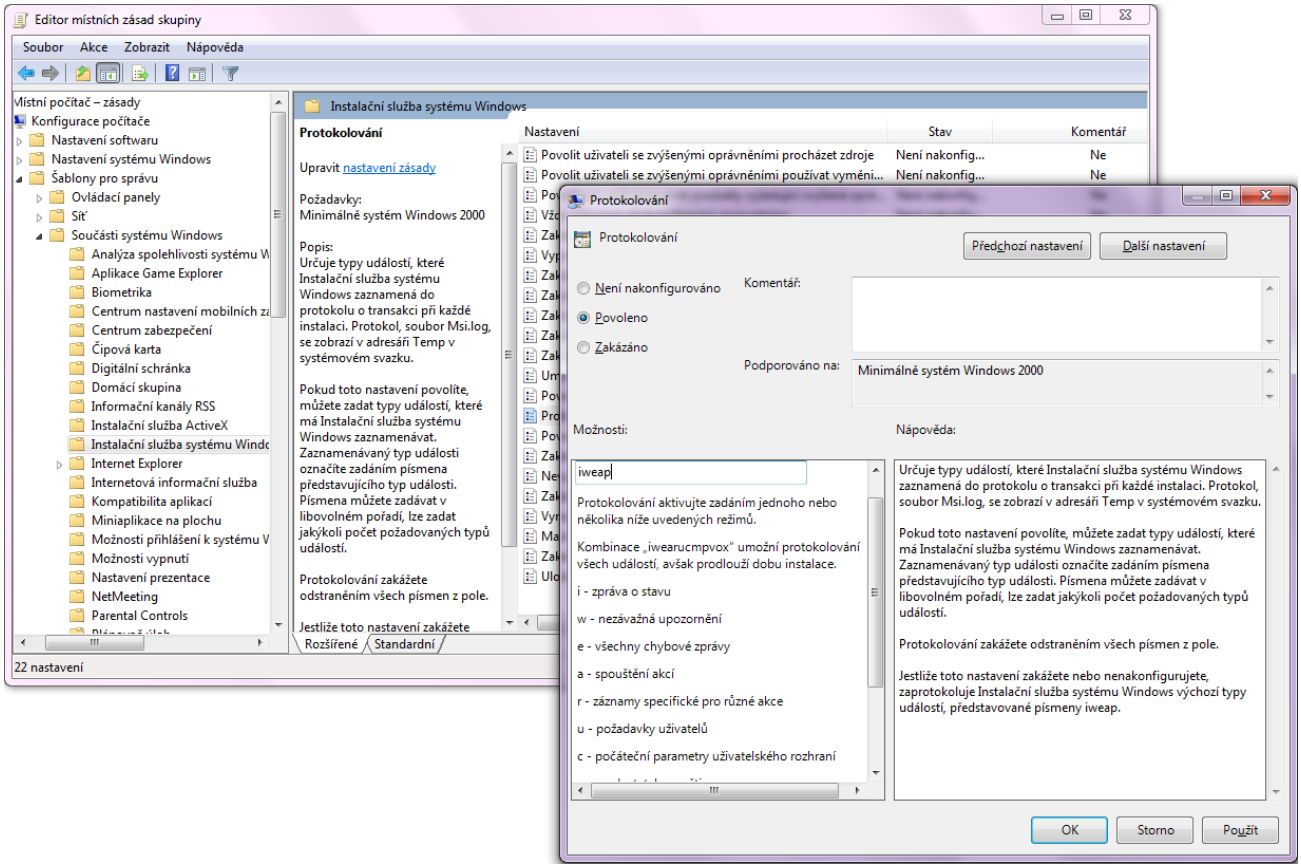
Každé nastavení v šablonách se nachází v jednom ze tří stavů:

- *Není nakonfigurováno* – tato zásada není nikde implementována, v registru se nenachází,
- *Povoleno* – nastavení uvedené v zásadě je uloženo v registru a aplikováno,
- *Zakázáno* – nastavení uvedené v zásadě není aplikováno a v registru je uložena informace o tom, že dané nastavení není použito (na rozdíl od první možnosti tam příslušný klíč existuje).

Když určitou zásadu povolíme, zapíše se do příslušné větve registru (pro každou šablonu Microsoft striktně stanoví, kam konkrétně se má zapsat). Když už nechceme, aby platila, navolíme u ní *Není nakonfigurováno*.

Příklad

Podívejme se na obrázek 2.9 (případně – pokud máte možnost, vyzkoušejte v reálu). Na obrázku je otevřena zásada Protokolování v šabloně Součásti systému Windows, Instalační služba systému Windows. Po poklepání na příslušnou zásadu se zobrazilo okno, ve kterém můžeme zásadu konfigurovat. Konkrétně u této zásady můžeme stanovit, co se má protokolovat v souvislosti s instalací aplikací.



Obrázek 2.9: Šablony pro správu

V konfiguračním okně předně určujeme, jestli má být zásada použita (klepneme na *Povoleno*). Následně se zpřístupní zbytek okna, kde už můžeme zásadu konfigurovat. V tomto případě můžeme dopsáním či smazáním písmen do řetězce určit, co konkrétně má být během instalace aplikací zaznamenáno. Dokonce si k zásadě můžeme napsat vlastní komentář (například dopíšeme informaci o tom, kdo a kdy nastavení provedl, za jakým účelem apod.).

Jednou z typických vlastností šablon je možnost snadné distribuce. Distribuují se v souborech s příponou ADM, u novějších Windows přibývají nové šablony ADMX (formát je odvozen z XML). Oba formáty existují zároveň, není to tak, že by ten novější měl nahradit starší. Práce s těmito soubory souvisí spíše s konfigurací v Active Directory ve firemních sítích.

Úkol

Projděte si nastavení šablon pro správu na počítači, kde máte oprávnění ke spuštění nástroje *Editor místních zásad skupiny*.

2.6.4 Šablony zabezpečení

Šablony zabezpečení jsou některé soubory s příponou INF obsahující konfiguraci zabezpečení (například zásady definování hesel, zabezpečení registru, systému souborů, služeb), jsou v adresáři `%WINDIR%\security\templates`.

Pracujeme s nimi buď lokálně na jednom počítači nebo v síti. Můžeme je importovat do objektu *Zásad skupiny* přidruženého k objektu služby Active Directory (po importu se přenastaví přístupová práva objektu služby Active Directory). S některými šablonami zabezpečení můžeme pracovat na lokálním počítači v grafickém rozhraní (například zásady definování hesel jsou v *Místních zásadách zabezpečení* (v *Nástrojích pro správu*).

Lepší nástroj (zvláště z hlediska automatizace a plného přístupu k možnostem) je *Konfigurace a analýza zabezpečení* – `secedit.exe`. Pro jeho spuštění bohužel potřebujeme práva administrátora. Po spuštění bez parametrů se spustí aplikace nápovědy (v grafickém rozhraní), samotný příkaz se vždy používá s parametry, například

`secedit /export /cfg d:\vysledek.inf` (s dalšími případnými parametry) exportuje nastavení zabezpečení z objektu do souboru šablony zabezpečení (toho, jehož název jsme zadali, měl by to být nejlépe dosud neexistující soubor),

`secedit /analyze` provede analýzu zabezpečení systému, je také nutné zadat název souboru, do kterého se uloží výsledek analýzy,

`secedit /configure` (s dalšími parametry) provede konfiguraci pro určitý objekt, tedy načte soubor INF šablony zabezpečení.

Úkoly

1. Pokud máte možnost spouštět příkazy s vyššími oprávněními, exportujte celou databázi zabezpečení do souboru `seced.inf`, umístěte soubor na disku, ke kterému máte přístup k zápisu. Tento soubor si prohlédněte.

   Všimněte si množství SID. Proč je před nimi vždy hvězdička? Vzpomeňte si na příkazy, ve kterých se SID zadává jako parametr.
2. Zjistěte, co vše umí příkaz `secedit`. Prohlédněte si syntaxi tohoto příkazu.
3. Projděte si složku `...\Windows\security` včetně jejích podsložek. V podsložce `Logs` otevřete soubor `winlogon.log` a najděte záznamy z současného data (na konci souboru). Všimněte si pořadí záznamů pro daný den (jde o posloupnost zavádění a konfigurace zásad při startu procesu `winlogon`). Projděte i další logy v této složce.

Souhrn kapitoly 2

V této kapitole jsme se zabývali tématy souvisejícími s uživateli, řízením přístupu a objekty ve Windows. Naučili jsme se pracovat s uživatelskými účty a skupinami, pracovat s přístupovými oprávněními, vysvětlili jsme si princip, na jakém pracují objekty ve Windows, a kde se k nim dostaneme. Také jsme si zopakovali znalosti o zásadách skupiny, šablonách pro správu a šablonách zabezpečení.

Kapitola 3

Správa procesů a služeb ve Windows

Rychlý náhled: V této kapitole se zaměříme na správu procesů a služeb ve Windows, zejména v textovém režimu. U procesů půjde o zjišťování informací, řízení jejich startu, plánování jejich spouštění a komunikaci mezi nimi, podíváme se také na programové rozhraní a kompatibilitu. U služeb si ujasníme, jak vlastně fungují, a také se podíváme na programy pro práci se službami. Posledním tématem kapitoly je WBEM, konkrétně mechanismus WMI ve Windows.

Klíčová slova: Proces, vlákno (thread), TID, přístupový token, SMSS, LSASS, SAM, LSAISO.exe, CSRSS.exe, explorer.exe, úloha, tasklist, taskkill, start, schtasks, shutdown, dde, ole, dynamicky linkované knihovny, sfc, sigverif, regsvr32, rundll32, DLL Hell, lokální verze knihoven, WinSxS, Win API, WoW64, SCM, svchost.exe, sc.exe, net config, net statistics, net start, net stop, restart-service, WBEM, CIM, WMI, wmic, gwmi.

Cíle studia: Po prostudování této kapitoly lépe porozumíte správě procesů ve Windows, budete mít přehled o nejdůležitějších systémových procesech, dokážete získat informace o spuštěných procesech jak v grafickém, tak i v textovém režimu, dokážete spustit proces s určitou prioritou či s ovlivněním jiných vlastností, budete umět naplánovat spuštění vybraného procesu, porozumíte vztahům mezi procesy včetně možností komunikace mezi nimi a případně systémem, naučíte se řešit problémy s kompatibilitou aplikací. Dále si ujasníte, co to vlastně je služba, jak funguje, naučíte se služby ovládat. V poslední části kapitoly se naučíte využívat WMI pro správu zařízení se systémem Windows.

3.1 Správa procesů

Z předchozího semestru víme, že k úlohám můžeme přistupovat pomocí *Správce úloh* anebo jiného nástroje, který si doinstalujeme (například *Process Explorer*). Nástrojů s grafickým rozhraním pro práci s procesy existuje poměrně hodně.

3.1.1 Procesy ve Windows NT

Proces je instance programu, aplikace nebo služby systému, která má přiděleny určité zdroje (paměť, čas procesoru apod.). V 32bitových a 64bitových verzích Windows jádro nepracuje přímo s procesy, ale s jejich *výpočetními vlákny* (thread), taktéž jádra procesoru jsou přidělována vláknům,

nikoliv celým procesům. Každý proces má minimálně jedno vlákno, vlákna samotná provádějí výpočty. Každý proces má

- tabulku deskriptorů objektů, což je vlastně odkaz na zdroje, které má k dispozici,
- vlastní adresový prostor, který byl přidělen správcem paměti (správcem virtuálních počítačů),
- spustitelný program (kód a globální data),
- *přístupový token* pro určení jeho přístupových oprávnění, atd.

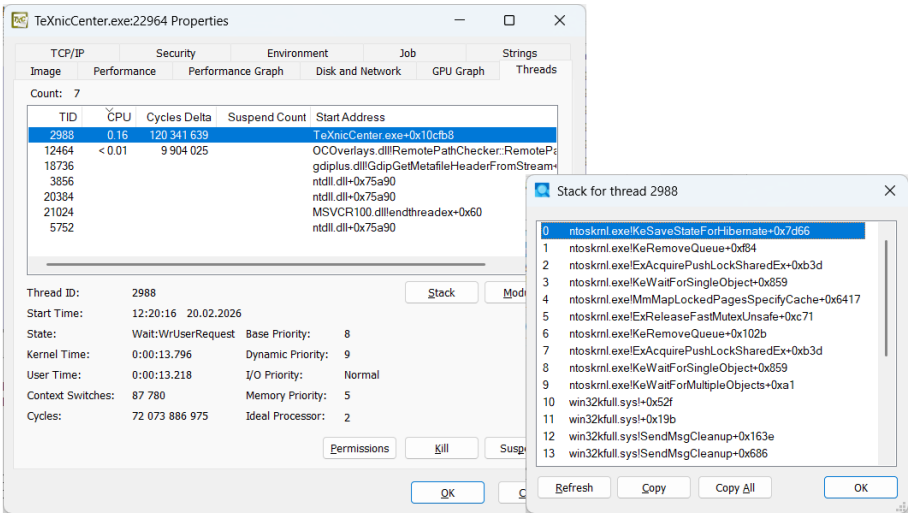
Každé *vlákno* má

- dva zásobníky (pro uživatelský a privilegovaný mód),
- sadu registrů, které obsahují stav procesoru,
- soukromou ukládací oblast v paměti používanou podsystémy a knihovnami DLL.

Hlavní vlákno procesu je spuštěno podle hlavního kódu procesu hned po vytvoření procesu, ale všechna ostatní vlákna jsou spuštěna API funkcí `CreateThread()` s kódem některé k tomu účelu naprogramované funkce. Může to být funkce uvnitř spustitelného souboru procesu anebo funkce z některé dynamicky linkované knihovny.

Příklad

Adresu funkce spuštěného vlákna zjistíme například v Process Exploreru – poklepeme na vybraný proces, čímž zobrazíme jeho vlastnosti, přejdeme na kartu *Threads* (Vlákna). Na této kartě je kromě jiného pro každé vlákno číslo *TID* (Thread ID, číslo vlákna) a adresa funkce vlákna (obvykle název souboru s číslem označujícím vzdálenost funkce v kódu od začátku souboru).



Obrázek 3.1: Vlastnosti vlákna procesu v Process Exploreru a jeho zásobník

Na obrázku 3.1 je karta *Threads* ve vlastnostech procesu TeXnicCenter.exe. Je zde seznam vláken, přičemž pro každé z nich je uvedeno číslo TID (jednoznačné identifikační číslo vlákna, bez ohledu na PID „rodičovského“ procesu). V malém okně vpravo je obsah zásobníku vybraného vlákna (získáme klepnutím na tlačítko Stack).

Ve sloupci CPU vidíme, jak moc je které vlákno zaneprázdněno (používá procesor).

Sloupec Start Adress udává, kde konkrétně je kód prováděný vláknem. Obvykle zde najdeme název spustitelného souboru procesu nebo název některé knihovny (ať už vlastní nebo systémové) následovaný adresou funkce uvnitř souboru – hlavní vlákno procesu vykonává hlavní funkci `main()`, ostatní vlákna některé jiné funkce.

Tlačítko *Permissions* zobrazuje oprávnění související s vláknem (když se proklikáme až ke speciálním oprávněním přes *Upřesnit*, *Upravit*, vidíme tam například oprávnění ukončit či uspat vlákno). Tlačítko *Kill* umožňuje násilně ukončit vlákno či proces a tlačítko *Suspend* ho uspí.

Dále se podíváme na několik důležitých procesů, zejména na podsystémy. Podsystém je proces, který poskytuje buď běhové prostředí dalším procesům nebo jiným způsobem slouží jako zprostředkovatel.

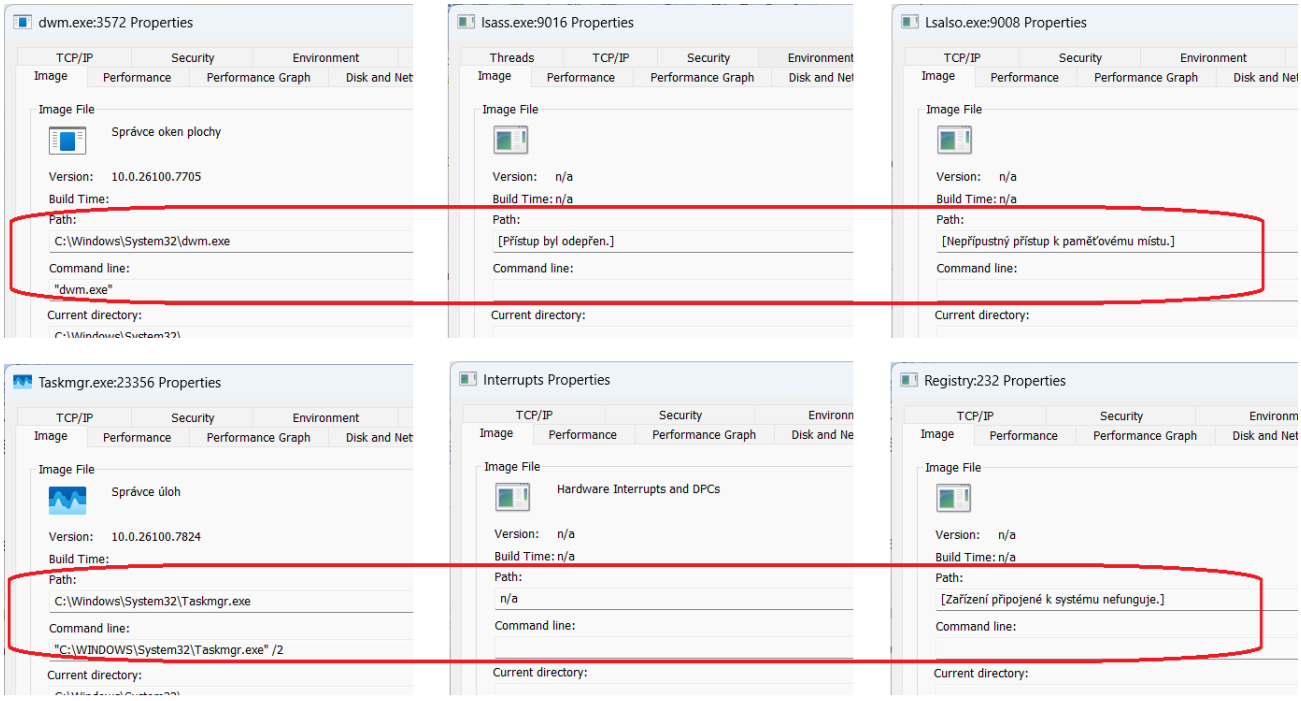
Podsystém *SMSS* (Session Manager Subsystem) je správce relací uživatelů. Jeho souborem je `smss.exe`. Jako jeden z mála procesů není napojen na žádnou stanici oken (tudíž nemá okno a nepoužívá schránku). Tento proces řídí všechny relace probíhající na daném počítači (takže pokud se přihlašujeme, ať už lokálně nebo přes síť, nebo když využíváme nasdílený prostředek z dotyčného počítače, zajišťuje to tento podsystém).

LSASS (Local Security Authority Subsystem) je podsystém, který ověřuje udržuje informace o všem, co se týká zabezpečení systému a právě zde běží všechny procesy ověřování přístupu, také spravuje zásady související se zabezpečením.

Podsystém LSASS pracuje například při přihlašování uživatelů (ověřuje jejich přihlašovací údaje), při změnách hesla, vytváření nových uživatelů (je třeba pro nového uživatele zřídit přístupový token s informacemi o jeho oprávněních), taky jako jediný proces zapisuje do *Protokolu zabezpečení* (který si můžeme prohlédnout v *Prohlížeči událostí*).

Tento podsystém používá databáze SAM (Security Account Manager), ve kterých jsou uloženy informace o uživatelských účtech a účtech skupin, a to buď databázi SAM na lokálním počítači (součást registru, v samostatném souboru) nebo v doméně.

Ve Windows 10 a vyšších najdeme také proces *LSAISO.exe* (LSA Isolated), jehož úkolem je zvyšovat zabezpečení procesu LSASS při jeho autentizační roli. Běží izolovaně od systému ve virtuálním prostředí (Virtual Secure Mode, Windows k tomu využívají Hyper-V).



Obrázek 3.2: Srovnání obsahu polí Path a Command line u různých procesů v Process Exploreru

Úkol

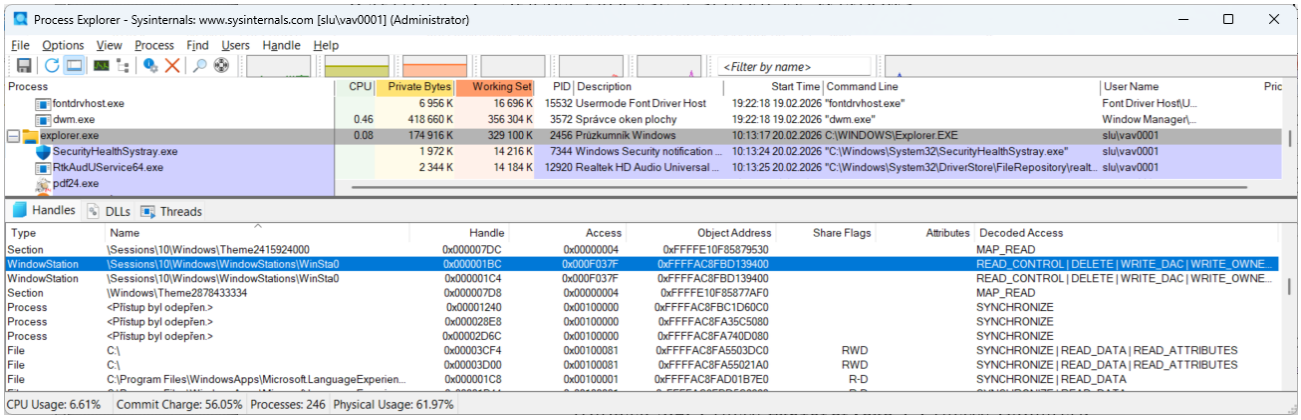
Srovnejte obsah polí Path a Command line u procesů na obrázku 3.2. Process Explorer byl spuštěn s oprávněním správce, přesto je obsah polí v některých případech nedostupný, ovšem z různých důvodů. Spusťte Process Explorer (pokud můžete, tak s oprávněním správce) a vyzkoušejte u jiných procesů.

Hlavní běhový podsystém se nazývá *Win32* (v 32bitových Windows) nebo *Windows* (v 64bitovém systému). Zajišťuje běh většiny ostatních procesů včetně mnoha systémových (rozhodně všech, které jsou interaktivní a využívají stanici oken WinSta0). Je fyzicky uložen především v souboru `csrss.exe`.

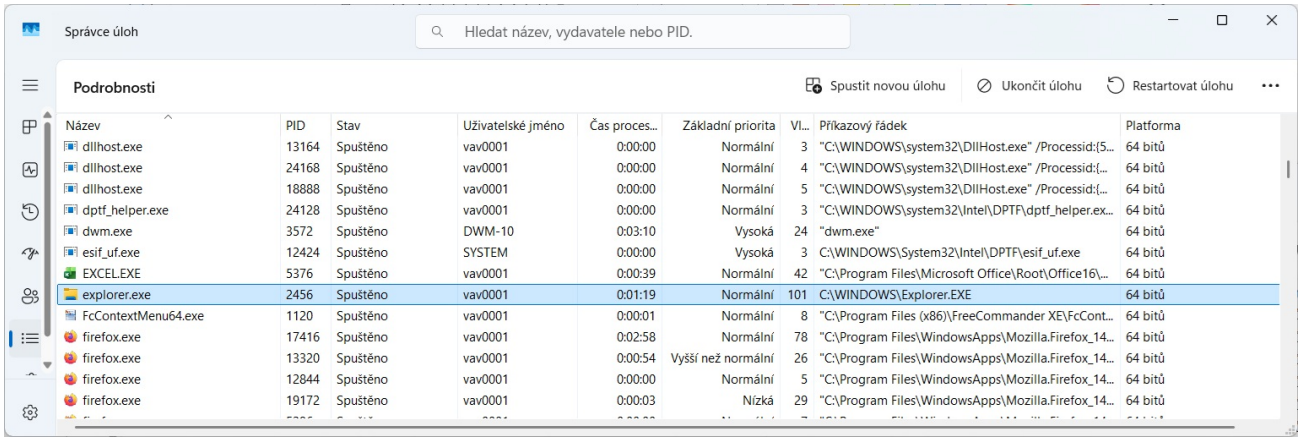
Proces, který se ve většině nástrojů zobrazuje jako *System*, nás v zobrazení procesů může překvapit. V Process Exploreru vidíme, že v hierarchické struktuře je předkem prakticky všech systémových procesů. Pokud si zobrazíme vlastnosti tohoto procesu, zjistíme, že nemá žádný spustitelný soubor, ale přesto v něm běží hodně vláken. Vlákna mají svůj kód buď v souboru `ntoskrnl.exe` (tj. hlavním souboru jádra) nebo v různých souborech s příponou SYS (ovladače). Tento proces je *hostitelem* vláken mnoha ovladačů – ovladače běžící v režimu jádra totiž většinou (ne vždy) nemají žádný vlastní proces.

Proces `Explorer.exe` je především procesem zajišťujícím grafické rozhraní. Ale část jeho vláken používáme ve formě aplikace *Průzkumník souborů*.

Na obrázku 3.3 vidíme proces `explorer.exe` v Process Exploreru a na obrázku 3.4 ve Správci úloh (karta *Podrobnosti*).



Obrázek 3.3: Proces `Explorer.exe` v Process Exploreru



Obrázek 3.4: Proces `Explorer.exe` ve Správci úloh ve Windows 11

Další „záhadný“ proces, který vidíme v Process Exploreru, je *Interrupts*. Ve skutečnosti vůbec nejde o proces (nejen že nemá žádný spustitelný soubor, ale dokonce ani vlákna ani přidělené prostředky), slouží k informování o čase, který systém stráví obsluhou hardwarových přerušení. Pokud zde najdeme vyšší hodnoty než obvykle, znamená to obvykle nějaký problém s hardwarem nebo ovladačem.

Úkoly

1. Pokud v Process Exploreru nemáte zobrazen sloupec s počtem vláken procesu, zobrazte ho (*View* ⇨ *Select Columns*, záložka *Process Performance*).

   U některého procesu s více vlákny ve vlastnostech na kartě *Threads* si vyberte některé vlákno a zobrazte jeho zásobník.
2. Spusťte některý jednoduchý program, třeba Poznámkový blok. Pak najděte jeho záznam v Process Exploreru a uspěte ho (uspání je zde myšleno jako suspendování). Pokuste se proces ukončit standardním způsobem, když to nepůjde, tak ho v Process Exploreru „zabijte“ (násilně ukončete).
3. Najděte v Process Exploreru ten proces `explorer.exe`. Prozkoumejte vlastnosti tohoto procesu, včetně vláken a času startu procesu (můžete také pro porovnání zobrazit sloupec *Start Time*, je u vybírání sloupců na záložce *Process Performance*).

3.1.2 Úlohy a procesy

Úloha je abstraktní zadání, které je třeba provést, ale tento pojem se v praxi používá i pro sadu procesů, které definovaným způsobem spolupracují na řešení zadání. Úlohu můžeme brát jako zobecnění činnosti a reprezentace procesů. Vztah mezi procesy patřícími do jedné úlohy bývá často typu rodič–potomek (ale vztah rodič–potomek neznamená automaticky, že patří do stejné úlohy).

V angličtině se používají dva pojmy – *job* a *task*. Obojí znamená úloha, jen v trochu jiném kontextu. Pojem task souvisí většinou s plánováním, zatímco job používáme v souvislosti s tiskovými úlohami a také s úlohami coby abstrakcí procesů (nebo skupin procesů).
S procesy a úlohami pracujeme ve Windows v nástroji *Správce úloh* (klávesová zkratka Ctrl+Shift+Esc), přičemž platí, že čím vyšší verze Windows, tím lépe je tento nástroj vybaven.

Alternativ ke Správci úloh je mnoho. Kromě nám již známého *Process Exploreru* můžeme používat například tyto *správce procesů*:

- *System Informer* (https://systeminformer.sourceforge.io/, dříve známý jako Process Hacker),
- *Process Lasso* (https://bitsum.com/),
- *Task Manager DeLuxe* (TMX, https://www.mitec.cz/wp/tmx/),
- *Task Patrol* (https://www.asmdev.net/products/taskpatrol/index.html),
- *Security Process Explorer* (https://www.glarysoft.com/security-process-explorer/),
- *DTaskManager* (spolu s dalšími aplikacemi na http://www.dimio.altervista.org/eng/).

Tyto aplikace nejsou všechny stejné. Vždy nabízejí alespoň takovou funkčnost jako správce úloh, mohou mít vylepšené bezpečnostní funkce, rozšířenou podporu sítě, množství informací o procesech, některé se nemusejí instalovat (portable), atd. Všechny jsou buď volně šiřitelné nebo mají volně dostupnou verzi.

Úkol

Projděte si nejméně dva správce procesů, se kterými jste zatím nepracovali (stačí projít jejich WWW stránky). Zaměřte se na odlišnosti v nabízených funkcích oproti Správci úloh a Process Exploreru.

S úlohami můžeme pracovat pomocí nástroje `tasklist`. Slouží k výpisu úloh (také podle vybraných kritérií), a to na místním nebo vzdáleném počítači.

`tasklist` zobrazí seznam běžících procesů (úloh) – název, PID, relaci, přidělená paměť

`tasklist /S počítač` totéž, ale pracujeme s procesy na zadaném počítači

`tasklist /V` zobrazené informace jsou podrobnější – přidají se sloupce pro stav procesu, jeho vlastníka (uživatelský účet), čas procesoru a titulek

`tasklist /SVC` k procesům se vypíšou i služby, které v nich běží

`tasklist /M` ke každému procesu zjistíme dynamicky linkované knihovny, které používá

`tasklist /M icmp.dll` takto zjistíme, které procesy používají zadanou knihovnu

`tasklist /fi "imagename eq svchost.exe"` používáme filtr – chceme pouze procesy, u nichž je název spustitelného souboru `svchost.exe` (spustitelný soubor se nazývá *image*)

`tasklist /fi "username eq NT AUTHORITY\SYSTEM"` chceme procesy běžící pod systémovým účtem

`tasklist /fi "memusage le 98000"` chceme procesy, které zabírají maximálně 98 000 kB paměti (`le` znamená less or equal, menší nebo rovno), kromě `eq` a `le` máme k dispozici operátory `ne` (není rovno), `gt` (větší), `lt` (menší), `ge` (větší nebo rovno)

`tasklist /FO list` výstup nebude míst formu tabulky, ale seznamu

`tasklist /FO csv >> proc.csv` výstup je ve formátu CSV, který lze importovat do běžných tabulkových editorů včetně Excelu (výstup v tomto formátu obvykle směrujeme do souboru)

Procesy a úlohy nemusíme pouze zobrazovat, můžeme je i ukončovat. K tomu slouží příkaz `taskkill` („zabíjení“ úloh).

`taskkill /pid 5810` ukončí proces se zadaným PID, jde o řádné ukončení podobně jako bychom klepli myší na křížek v pravém horním rohu okna

`taskkill /F /pid 5810` *zabije* proces se zadaným PID (F jako Force, silný), použijeme, pokud proces nereaguje a odmítá se nechat ukončit běžným způsobem

`taskkill /T /pid 5810` ukončíme nejen zadaný proces, ale i všechny jeho potomky (rekurzívně)

`taskkill /IM firefox.exe` proces můžeme zadat pomocí tohoto přepínače i jeho názvem

`taskkill /fi "username eq uživatel"` můžeme používat stejné filtry jako u příkazu `tasklist`, zde ukončíme všechny procesy pracující pod zadaným uživatelským účtem

Poznámka

Pozor – to, že úloha neodpovídá (nereaguje), ještě neznamená, že zamrzla. Někdy prostě stačí počkat. Může například zrovna něco náročnějšího zpracovávat nebo čeká na přidělení prostředku, třeba otevření souboru nebo přidělení dalšího úseku paměti.

Příklad

Výstup příkazu vypisujícího základní informace o procesech (zkrácený):

```
C:\> tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          8 K
System                           4 Services                   0        152 K
```

```
Secure System                  188 Services                   0     56 104 K
Registry                       232 Services                   0     55 380 K
smss.exe                       688 Services                   0        712 K
csrss.exe                     6580 Services                   0      4 120 K
wininit.exe                   8908 Services                   0      4 484 K
services.exe                  8988 Services                   0     13 100 K
LsaIso.exe                    9008 Services                   0      3 476 K
lsass.exe                     9016 Services                   0     31 196 K
svchost.exe                   8900 Services                   0     41 592 K
...
powershell.exe               23268 Console                   10     71 532 K
OpenConsole.exe              11768 Console                   10     12 092 K
cmd.exe                      13836 Console                   10      6 388 K
tasklist.exe                 11288 Console                   10     12 264 K
WmiPrvSE.exe                 10928 Services                   0     15 336 K
```

Když budeme chtít podrobnější výpis, použijeme přepínač `/v`. Záhlaví výpisu je následující (samotný výpis je dlouhý, a navíc dost do šířky):

```
C:\> tasklist /v

Image Name      PID   Session Na Session# Mem Usage  Status  User Name  CPU Time  Window Title
=============== ===== ========== ======== ========== ======= ========= =========  =============
```

Nechceme všechno, ale zajímají nás pouze procesy `svchost.exe`, tedy budeme filtrovat:

```
C:\> tasklist /v | findstr /i "svchost.exe"

svchost.exe     8900  Services          0   41 428 K Unknown NT AUTHORITY\SYSTEM          0:03:17   N/A
svchost.exe     9348  Services          0   18 980 K Unknown NT AUTHORITY\NETWORK SERVICE 0:22:21   N/A
...
svchost.exe     6600  Console          10   38 400 K Unknown slu\vav0001                  0:00:02   N/A
svchost.exe    23312  Console          10   12 168 K Unknown slu\vav0001                  0:00:00   N/A
svchost.exe    12480  Console          10   38 196 K Running slu\vav0001                  0:00:01   Windows Push Not..
...
```

Oproti předchozímu výpisu máme navíc především účet, pod kterým služba pracuje (zobrazí se jen tehdy, pokud máme příkazový řádek spuštěný jako správce), což už může být cenná informace. Pokud chceme zjistit, které služby běží v různých procesech, zadáme (výstup je zkrácený):

```
C:\> tasklist /svc

Image Name                     PID Services
========================= ======== ============================================================================
System Idle Process              0 N/A
System                           4 N/A
Secure System                  188 N/A
Registry                       232 N/A
smss.exe                       688 N/A
csrss.exe                     6580 N/A
wininit.exe                   8908 N/A
services.exe                  8988 N/A
LsaIso.exe                    9008 N/A
lsass.exe                     9016 KeyIso, SamSs, VaultSvc
svchost.exe                   8900 BrokerInfrastructure, DcomLaunch, PlugPlay, Power, SystemEventsBroker
WUDFHost.exe                  9028 N/A
fontdrvhost.exe               9228 N/A
svchost.exe                   9348 RpcEptMapper, RpcSs
svchost.exe                   9388 LSM
WUDFHost.exe                  9460 N/A
svchost.exe                   9684 HvHost
svchost.exe                   9692 BDESVC
svchost.exe                   9908 NcbService
svchost.exe                   9924 TimeBrokerSvc
svchost.exe                   9932 nsi
svchost.exe                  10052 Dnscache
...
```

Jednotlivé procesy `svchost.exe` lze rozlišit jen podle jejich PID. Zjištěné PID můžeme využít také k výpisu všech modulů (především dynamicky linkovaných knihoven, ale i dalších namapovaných souborů) daného procesu:

```
C:\> tasklist /m /fi "pid eq 10052"

Image Name                     PID Modules
========================= ======== ============================================
svchost.exe                  10052 ntdll.dll, KERNEL32.DLL, KERNELBASE.dll,
                                   ucrtbase.dll, combase.dll, RPCRT4.dll,
                                   kernel.appcore.dll, msvcrt.dll,
                                   bcryptPrimitives.dll, user32.dll,
                                   win32u.dll, GDI32.dll, gdi32full.dll,
                                   msvcp_win.dll, sechost.dll, dnsrslvr.dll,
                                   DNSAPI.dll, ncrypt.dll, IPHLPAPI.DLL,
                                   NSI.dll, NTASN1.dll, Fwpuclnt.dll,
                                   WS2_32.dll, WINNSI.DLL, gpapi.dll,
                                   shcore.dll, firewallapi.dll, fwbase.dll,
                                   mswsock.dll, dhcpcsvc6.DLL, dhcpcsvc.DLL,
                                   WINHTTP.dll, CRYPTBASE.dll, wshbth.dll,
                                   rasadhlp.dll, DEVOBJ.dll, CFGMGR32.dll,
                                   nlansp_c.dll, winrnr.dll, napinsp.dll
```

Jde o seznam modulů procesu `svchost.exe`, ve kterém běží služba *DNSCache*. Uživatelský proces by měl modulů ještě více.

V Powershellu máme k dispozici cmdlet `get-process`. Pomocí parametrů pak určujeme, co konkrétně o procesech chceme zobrazit. V první kapitole jsou na straně 16 příklady jak různých parametrů a filtrování, tak i formátování výstupu.

Úkoly

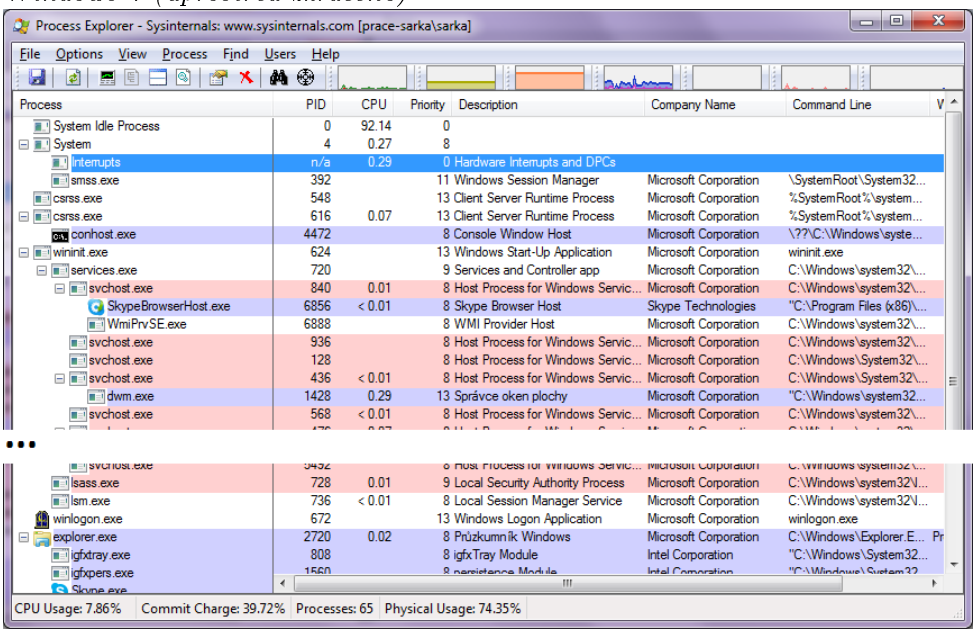
1. Vypište seznam procesů pomocí příkazu `tasklist` a porovnejte výstup s tím, co zobrazuje Správce úloh. Pak vyzkoušejte různé možnosti formátování výstupu, výstup ve formě CSV (směrujte do souboru) pak otevřete (importujte) v některém tabulkovém editoru.
2. Vyzkoušejte u příkazu `tasklist` nejrůznější filtry, které je možné použít (celý seznam najdete v nápovědě příkazu nebo na internetu). Bohužel je možné, že v české verzi Windows se u filtrů setkáte s problémy.
3. Spusťte některý jednoduchý program (Poznámkový blok, Kalkulačku apod.). Pomocí příkazu `tasklist` si prohlédněte vlastnosti této úlohy a pak ji ukončete příkazem `taskkill`.
4. V Powershellu si zobrazte detailní nápovědu cmdletu `get-process`, a pak nápovědu s příklady: `get-help get-process -examples`. Vyzkoušejte výpis vlastností konkrétního procesu s určitým názvem (například pokud máte spuštěný webový prohlížeč), dále pro tento proces si vypište jeho načtené moduly (parametr `-module`). Upozornění: u webového prohlížeče to bude velmi dlouhý výpis, případně vyzkoušejte notepad nebo nějaký jiný (spuštěný) proces.

3.1.3 Vztahy mezi procesy

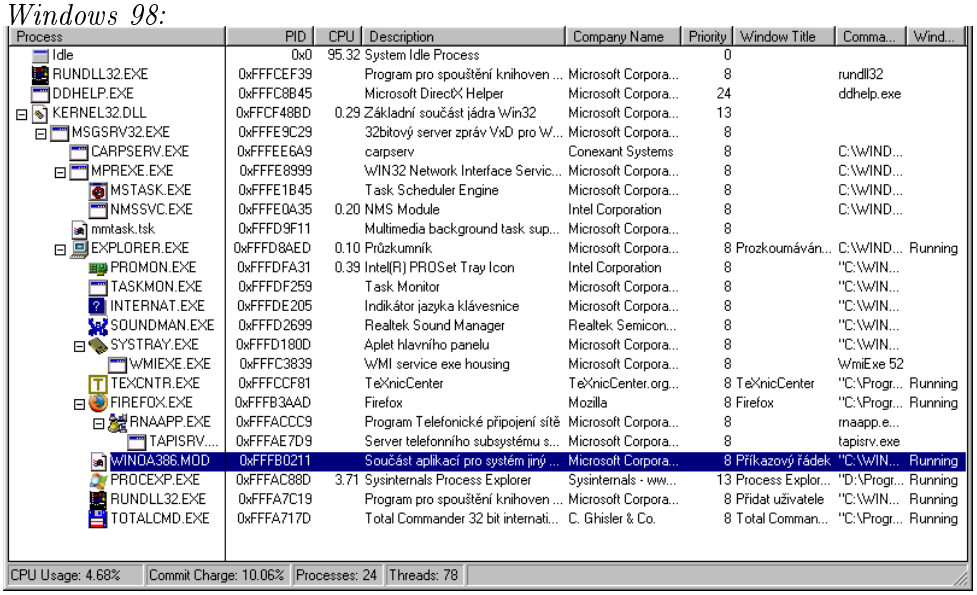
Procesy bývají ve většině operačních systémů zařazeny ve stromové struktuře podle vztahu rodič–potomek (potomek je proces, který byl spuštěn rodičovským procesem). Každý proces má přiděleno své identifikační číslo PID (Process ID) a zná také PID svého rodiče (PPID, Parent PID). Ve Windows s DOS jádrem žádná PID neexistovala, procesy se identifikovaly pouze podle handlu svého hlavního

okna (ano, to známe z kapitoly o objektech). Na obrázku 3.5 vidíme srovnání okna Process Exploreru spuštěného ve Windows 7 (NT jádro) a Windows 98 (DOS jádro). Všimněte si obsahu sloupce PID.

Windows 7 (uprostřed zkráceno):



Windows 98:



Obrázek 3.5: Process Explorer ve Windows 7 a 98

Ve Windows je struktura procesů pojata volněji než v UNIXových systémech, mohou zde existovat i procesy bez rodiče (rodič byl ukončen). Pokud už rodičovský proces neexistuje (ukončil svou činnost) a potomci pořád ještě existují, jsou tito potomci „zarovnáni“ doleva, tedy už nemají žádného rodiče.

Příklad

V Process Exploreru zjistíme PPID procesu jednoduše tak, že poklepeme na proces a na záložce *Image* najdeme položku *Parent* (je ve spodní části okna). Pokud se jedná o proces mající rodiče, vidíme tam název a PID rodičovského procesu, což je vlastně PPID našeho procesu.

Na obrázku 3.5 (v části pro Windows 7) je hned několik procesů bez rodiče, například `explorer.exe` (vlastně všechny, které jsou zarovnány zcela vlevo). Pokud se podíváme ve vlastnostech tohoto procesu na záložku *Image*, zjistíme, že údaj *Parent* je nastaven na „*<Non-existent process>(2249)*“ (neexistující proces, číslo v závorce znamená PPID před ukončením rodiče, zřejmě budete mít jiné).

3.1.4 Řízení startu procesů

Abychom spustili nějaký program v Příkazovém řádku, stačí napsat jeho jméno (příp. s cestou k tomuto programu) a potvrdit klávesou Enter. Příkaz `START` však při spouštění programu rozšiřuje naše možnosti především tak, že umožňuje spuštění programu řídit pomocí parametrů.

Používání tohoto příkazu si ukážeme na příkladu.

Příklad

Program `notepad.exe` je „okenní“ aplikace *Poznámkový blok*. V Příkazovém řádku tuto aplikaci můžeme spustit několika způsoby:

`notepad.exe` spustí Poznámkový blok, jakobychom použili grafické prostředí, Příkazový řádek nečeká na ukončení aplikace a hned zobrazí prompt

`START notepad.exe` provede totéž

`START /wait notepad.exe` spustí aplikaci a počká na její ukončení, teprve pak zobrazí prompt a očekává příkazy

`START /min notepad.exe` aplikace se spustí minimalizovaná (přepínač může mít v některých Windows tvar `/m`)

`START /max notepad.exe` aplikace se spustí maximalizovaná

`START /low notepad.exe` aplikace se spustí s nízkou prioritou (vysoká priorita se určuje parametrem `/high`, pro ostatní stupně priority existují další parametry)

`START notepad.exe soub.txt` spustí aplikaci a předá jí zbytek řádku jako parametry

`START soub.txt` spustí aplikaci ve Windows asociovanou s příponou TXT (obvykle Poznámkový blok) a jako parametr jí předá soubor `soub.txt`

`START /i cmd` spustí nové okno Příkazového řádku (`cmd.exe`) jako by bylo spuštěno přímo z Windows (nové okno apod.)

Příkaz `start` nám tedy může pomoci i s nastavením základní priority procesu při jeho spuštění. Priorita se označuje slovně nebo číslem (v parametru příkazu tedy slovně). V grafickém rozhraní vidíme priority spuštěných procesů v těchto nástrojích (pokud tedy máme zobrazen příslušný sloupec):

- *Správce úloh* (na kartě *Procesy*, resp. ve Windows 10 volíme Více informací, karta Podrobnosti), sloupec *Základní priorita* – je zde pouze slovní určení priority,
- *Process Explorer* – zobrazuje také číselnou hodnotu priority, na obrázku 3.5 ji vidíme ve sloupci *Priority*.

V obou těchto nástrojích můžeme změnit základní prioritu procesu (kontextové menu), ale pouze mezi „slovními“ úrovněmi, například „Normální“ (Normal), „Vysoká“ (High), „Nízká“ (Low) apod. (v kontextovém menu procesu).

Dynamickou prioritu určuje systém, pohybuje se kolem (námi určené) základní priority.

Příklad

V Powershellu spustíme nový proces cmdletem `start-process`, pro ukončení vybraného procesu (nebo procesů) použijeme cmdlet `stop-process`. Při startu procesu můžeme určit styl okna (například minimalizované):

```
start-process notepad.exe -WindowStyle minimized
```

Zamrznutý proces ukončíme takto (zde pomocí názvu, ukončí se všechny procesy s tímto názvem):

```
stop-process -force -name notepad.exe
```

`ASSOC` a `FTYPE` jsou příkazy pro definování asociací přípon souborů a programů. První příkaz slouží k přiřazení typu souboru dané příponě, druhý pak k přiřazení aplikace danému typu souboru. Syntaxe:

```
ASSOC přípona=typ
FTYPE typ=aplikace
```

Například:

`assoc` vypíše seznam asociací přípon a typů souborů

`ftype` vypíše seznam přiřazení typů souborů a aplikací

`assoc .htm=textfile` souborům s příponou HTM je přiřazen typ „textfile“, to znamená, že soubor s touto příponou se po poklepání otevře v programu určeném pro tento typ (obvykle Poznámkový blok)

`assoc .html=htmlsoubor` pro soubory s příponou HTML jsme vytvořili nový souborový typ

`assoc .abc=` odstranili jsme asociaci definovanou pro příponu ABC

`ftype htmlsoubor=pspad.exe %1` souborovému typu vytvořenému v předchozím příkazu jsme přiřadili aplikaci PSPad, má se vždy spustit s jedním parametrem, kterým je název otevíraného souboru (tj. když v grafickém rozhraní poklepeme na soubor s danou příponou, otevře se v této aplikaci)

`ftype htmlsoubor=` odstranili jsme nadefinovaný souborový typ

Úkoly

1. Spusťte některou menší aplikaci s grafickým rozhraním (`notepad.exe`, `calc.exe`, apod.) s nižší a potom naopak vyšší prioritou, a to s minimalizovaným oknem. Výsledek zkontrolujte ve Správci úloh nebo Process Exploreru.

   V Process Exploreru nebo Správci úloh snižte prioritu vámi spuštěného procesu o jeden „slovní“ stupeň. V Process Exploreru se pak podívejte na číselné vyjádření výchozí priority (místo 8 by mělo být 6).
2. V Process Exploreru seřaďte procesy podle priorit (stačí klepnout na záhlaví sloupce s prioritami) a zkontrolujte, které procesy mají nejvyšší prioritu.
3. V nápovědě příkazu `start` zjistěte, jaké další přepínače lze použít pro nastavení priority spouštěné aplikace.
4. V Powershellu spusťte některý nový proces a následně ho ukončete (také v Powershellu). Pro ukončení použijte PID procesu (parametr `id`), které zjistíte ve výpisu cmdletu `get-process`.

3.1.5 Plánování

Spouštění procesů můžeme plánovat v grafickém režimu v nástroji *Plánovač úloh*, což je konzola `taskschd.msc`. V prostředí Příkazového řádku se původně používal příkaz `AT`, ale v novějších Windows je podporován už jen příkaz `schtasks`. V každém případě musí být spuštěna služba plánování procesů, aby tyto nástroje fungovaly.

Ukážeme si, jak se používá příkaz `schtasks` (Scheduled Tasks, naplánované úlohy). Pomocí tohoto nástroje můžeme (nejen) plánovat spouštění zadaných úloh, procesů a příkazů buď jednorázově nebo pravidelně v zadaný čas a jakkoliv řídit jejich naplánování, a to i na vzdáleném počítači. Příkaz je rozhraním k mechanismu, který používá i samotný operační systém, podle toho také vypadá délka a obsah výpisů naplánovaných úloh.

`schtasks /CREATE /SC daily /ST 16:00 /TN Název /TR soubor.exe` naplánujeme spouštění úlohy denně vždy v 16 hodin, úloha se jmenuje *Název* a je spuštěna ze zadaného souboru

`schtasks /CREATE /?` zobrazíme nápovědu k podpříkazu `CREATE` (to můžeme provést s kterýmkoliv podpříkazem)

`schtasks /QUERY` získáme tabulku všech naplánovaných úloh na našem počítači

`schtasks /QUERY /S počítač` totéž, ale na zadaném počítači

`schtasks /QUERY /FO list` výpis nebude ve formě tabulky, ale jako seznam (oproti tabulce získáme některé informace navíc – způsob spouštění a název hostitelského počítače)

`schtasks /QUERY /TN Název /FO list` vypíše se informace o zadané úloze

`schtasks /CHANGE /TN "Antivirová kontrola" /TR C:\antivir.exe` zadaná úlo- ha už existuje, ale chceme ji pozměnit – změníme program, který se má v dané úloze spustit

`schtasks /RUN /TN "Antivirová kontrola"` zadaná úloha bude spuštěna okamžitě, bez ohledu na nastavení plánování úloh (mohli bychom spustit přímo spustitelný soubor, ale zde můžeme využít vytvořené definice úloh)

`schtasks /END /TN "Antivirová kontrola"` okamžité ukončení zadané běžící úlohy (ale naplánovaná zůstane)

`schtasks /DELETE /TN Úloha /S počítač` odstraní naplánování úlohy na zadaném počítači

Příklad

Vytvoříme novou úlohu nejdřív tak, aby při jejím spouštěním musel uživatel zadávat heslo (pokud se jedná o úlohu s potřebou vyšších oprávnění), a pak podobnou úlohu tak, aby uživatel heslo zadávat nemusel (úloha poběží na pozadí).

`schtasks /CREATE /S ucetni /SC daily /ST 16:00`
`/TN Monitorovani /TR c:\prog\monitoring.exe` vytvoříme novou naplánovanou úlohu s názvem *Monitorovani*, která se má provést na počítači s názvem *ucetni*, a to denně vždy v 16 hodin, uživatel bude pokaždé dotázán na heslo.

Získáme hlášení:

```
Úspěch: Naplánovaná úloha Monitorovana byla úspěšně vytvořena.
```

`schtasks /QUERY /S ucetni` zobrazí se tabulka naplánovaných úloh na zadaném počítači, něco podobného:

```
...
Složka: \
Název úkolu                             Čas příštího spuštění Stav
=======================================================================
Monitorovani                            10.2.2010 16:00:00    Připraveno
...

Složka: \Microsoft\Windows\Defrag
Název úkolu                             Čas příštího spuštění Stav
=======================================================================
ScheduledDefrag                         10.2.2010 1:00:00     Připraveno

Složka: \Microsoft\Windows\DiskDiagnostic
Název úkolu                             Čas příštího spuštění Stav
=======================================================================
Microsoft-Windows-DiskDiagnosticDataColl 21.2.2010 1:00:00     Připraveno
Microsoft-Windows-DiskDiagnosticResolver                       Zakázáno
...
```

`schtasks /query /S ucetni /FO list /tn Monitorovani` zobrazí informaci o zadané úloze, a to na počítači účetního a navíc ve formátu „seznam“ (výchozí formát je tabulka, kterou jsme viděli v předchozím výpisu). Výsledek:

```
Složka: \
Název hostitele:       UCETNI
Název úkolu:           \Monitorovani
Čas příštího spuštění: 10.2.2010 16:00:00
Stav:                  Připraveno
Režim přihlášení:      Pouze interaktivně
```

`schtasks /CREATE /SC daily /ST 16:00 /RU uživatel /RP /TN Monitorovani2 /TR c:\prog\monitoring.exe` vytvoříme úlohu *Monitorovani2* tentokrát na lokálním počítači spouštěnou ve stejném čase jako předchozí (také má stejný spustitelný soubor), ale úloha bude spouštěna s přístupovými oprávněními zadaného uživatele, heslo zadáváme pouze nyní (uživatel není pravidelně obtěžován žádostí o heslo, které ani nemusí znát)

`schtasks /query /FO list /tn Monitorovani2` zobrazíme informaci o zadané úloze ve formátu „seznam“. Výsledek:

```
Složka: \
Název hostitele:       NOTEBOOK
Název úkolu:           \Monitorovani2
Čas příštího spuštění: 10.2.2010 16:00:00
Stav:                  Připraveno
Režim přihlášení:      Interaktivně nebo na pozadí
```

Další informace

Podrobné informace o příkazu `schtasks` získáme buď z nápovědy příkazu (běžným způsobem, který známe od jiných příkazů) anebo na adresách

- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/schtasks
- https://learn.microsoft.com/en-us/windows/win32/taskschd/schtasks

Naplánovat se dá také vypnutí, restart či uspání počítače (také vzdáleně) anebo odhlášení uživatele. K tomu účelu používáme příkaz `shutdown` (v některých edicích Windows není zahrnut).

`shutdown /L` odhlášení právě přihlášeného uživatele

`shutdown /s /m \\počítač` vypne zadaný počítač

`shutdown /r` restart

`shutdown /h` hibernace (pouze ve verzích, které hibernaci umožňují)

`shutdown /s /t 5` je nastaven časový limit 5 sekund, po něm se počítač automaticky vypne

`shutdown /fw /r` zařízení se restartuje a přejde do UEFI (viz nápovědu)

V Powershellu můžeme použít cmdlety `stop-computer` a `restart-computer`. Takto lze vypnout nebo restartovat počítač, ale hibernovat či uspat ne.

Úkoly

1. Naplánujte spuštění programu `kontrola.exe` tak, aby se spouštěl každý pátek ve 4 hodiny odpoledne. Úlohu nazvěte *Kontrola*.
2. Proveďte totéž co v předchozím úkolu, ale vzdáleně na počítači `\\sekretarka`.

   Pokud nemáte možnost plánovat spouštění na jiných počítačích než na tom, u kterého sedíte, použijte název svého počítače – pamatujete si, jak ho zjistit?
3. Zjistěte, jaké údaje jsou uloženy o naplánování provedeném v předchozích dvou bodech (tj. zobrazte informace o naplánovaných příkazech/úlohách). Vyzkoušejte tabulkové i seznamové zobrazení naplánované úlohy.
4. Odstraňte naplánování všech příkazů/úloh, které jste vytvořili v předchozích úkolech.
5. Vyzkoušejte své odhlášení pomocí příkazu, ke konci práce s počítačem také restart nebo vypnutí pomocí příkazu.

3.2 Komunikace mezi procesy

Procesy mohou mezi sebou komunikovat různě. Pokud navzájem znají své PID (což je typické třeba mezi rodičem a potomkem), pak si mohou posílat zprávy. Další možnost je použití schránky, pokud ovšem vlastní objekt WinSta0. Další možnosti jsou COM, DDE, OLE, roury, sockety, sdílená paměť,... Některým z těchto možností se budeme věnovat v třetím ročníku v předmětu Architektura operačních systémů, jiné už více méně známe (schránku, roury), zde se podíváme na DDE a OLE.

3.2.1 DDE

DDE (Dynamic Data Exchange, Dynamická výměna dat) je technologie umožňující komunikaci mezi procesy (je ve Windows už od verze 2). Dnes už není příliš aktuální, ve Windows ji máme spíše kvůli zpětné kompatibilitě, ale občas se objeví chybové hlášení s DDE související, tedy je dobré alespoň vědět, co to je.

Jeden program může žádat data od druhého, aniž by to vyžadovalo zásah uživatele, jde tedy o dynamickou formu *sdílení dat* (technicky jde o nástavbu schránky). Jeden proces (*Server*) poskytuje data druhému procesu (*Klientovi*). Oba komunikující procesy (Server i Klient) musí být spuštěny. Aplikace takto mohou sdílet určitý objekt a komunikovat na jednom nebo více kanálech.

Můžeme se setkat s chybovou zprávou „DDE server error“, typicky DDE server přestal pracovat. Pokud se nám podaří vysledovat, o který proces ve skutečnosti jde (může to být třeba `explorer.exe` nebo proces kancelářského balíku), obvykle stačí příslušný proces restartovat (resp. ukončit a spustit).

3.2.2 OLE

OLE (Object Linking and Embedding) je technologie, která umožňuje vnořit do objektu vytvářeného jednou aplikací objekt vytvořený jinou aplikací nebo vytvořit propojení k takovému objektu. OLE objekty byly zmíněny na straně 59 v sekci o modelu COM.

Aplikace, které mohou být zdrojem, se nazývají *Servery*, cíle jsou *Klienty*. Některé aplikace dokážou být obojí. Na rozdíl od DDE můžeme sdílet nejen data, ale obecně různé objekty (všechny OLE objekty jsou implementací rozhraní *IOleObject* z modelu COM), a pokud je nutné v „přijímající“ aplikaci objekt upravit či s ním jakkoliv pracovat, nechá tuto činnost na „poskytující“ aplikaci.

Při *vnoření* objektu se získaný objekt stává součástí vytvářeného dokumentu (obecně souhrnu objektů), vazba je jen nepřímá (je to pouze vazba na aplikaci, která je OLE serverem).

Příklad

Typický příklad použití je spolupráce internetového prohlížeče (takového, který OLE podporuje) a Adobe Readeru. Pokud v internetovém prohlížeči, který nemá integrovaný prohlížeč PDF souborů, klepneme myší na odkaz na PDF soubor, prohlížeč vytvoří OLE vazbu k Adobe Readeru či jinému programu pracujícímu s PDF a ve svém vlastním okně poskytne prostor pro zobrazení PDF souboru.

Jiná možnost pro OLE je *propojení*. Na rozdíl od vložení nejsou data ve skutečnosti součástí objektu klienta, ale zůstávají uložena u serveru, u klienta je vytvořena pouze vazba na tato data. V případě provádění změn objektu na straně serveru se změny projeví i na straně klienta.

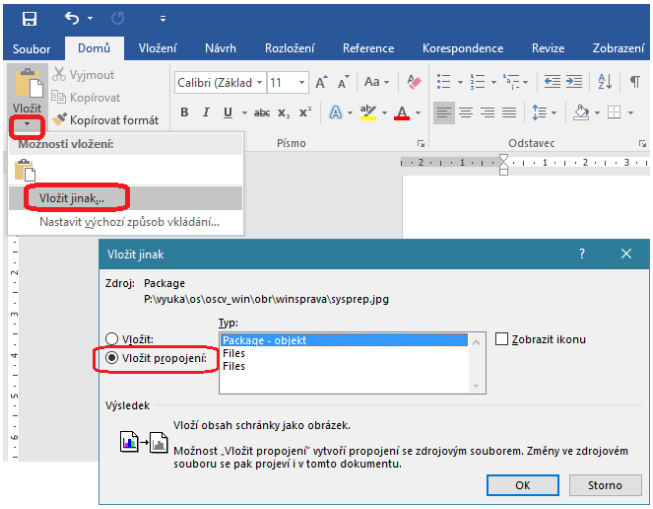
Pokud dokument vytvořený klientem přemístíme, propojený objekt se „ztratí“, protože zůstal u serverové aplikace. Pokud se ale vazba „ztratí“ za běhu klienta, nemusí dojít ke ztrátě dat, pouze se ztratí vazba na server, a data objektu si klient uloží v tom formátu, kterému „rozumí“ (třeba obrázek).

Typicky se propojení používá třeba při spolupráci mezi různými částmi kancelářského balíku (týká se to jak MS Office, tak i například OpenOffice.org a jeho odvozenin).

Příklad

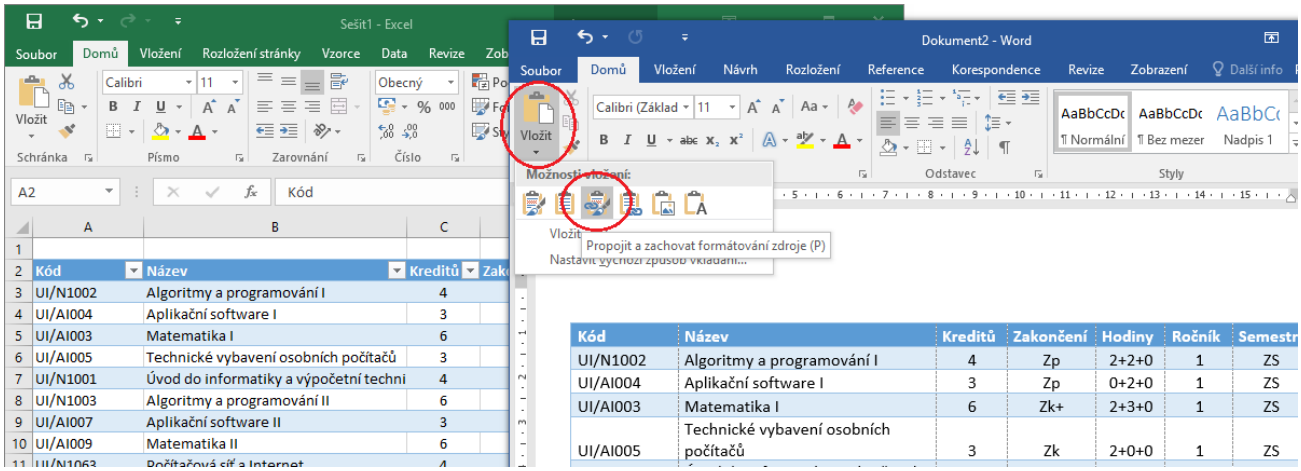
Na obrázku 3.6 je ukázka, jak můžeme do dokumentu Wordu vložit soubor s obrázkem. Na soubor použijeme zkratku Ctrl+C, pak ve Wordu *Vložit jinak...*, zobrazí se dialogové okno. V něm určíme, že místo běžného vložení chceme „Vložit propojení“, taky můžeme zaškrtnout, že místo náhledu objektu vložíme jen ikonu.

Jiný příklad: na obrázku 3.7 vidíme tabulku původně vytvořenou v Excelu (včetně vzorců pro součet a počet polí), s funkcí *Formátovat jako tabulku*. Tabulka byla potom vložena do dokumentu Wordu (běžně – Ctrl+C, Ctrl+V, včetně záhlaví



Obrázek 3.6: OLE vazba propojením – obrázek ve Wordu

a zápatí tabulky). Pokud jsou obě aplikace (serverová/Excel i klientská/Word) spuštěny, můžeme provádět změny v Excelu, projeví se i ve Wordu.



Obrázek 3.7: Využití OLE vazby propojením v MS Word

Rozdíl mezi vnořením a propojením je tedy především v umístění objektu, se kterým pracujeme. Při vnoření je objekt součástí klienta, který žádá server o pomoc s jeho zpracováním, kdežto při propojení je samotný objekt na straně serveru a klient dostává jen jeho zjednodušenou podobu, se kterou si umí poradit (třeba ji umí zobrazit v dokumentu).

Programátoři používají *OCX* (OLE Control eXtension), což jsou binárně přenosné komponenty obsahující prvky uživatelského rozhraní (lze je používat v různých programovacích jazycích), jsou uloženy v souborech s příponou OCX (což jsou dynamicky linkované knihovny).

Úkol

Vyzkoušejte OLE vazbu mezi aplikacemi patřícími do některého kancelářského balíku (MS Office nebo OpenOffice.org), třeba podle výše uvedeného příkladu.

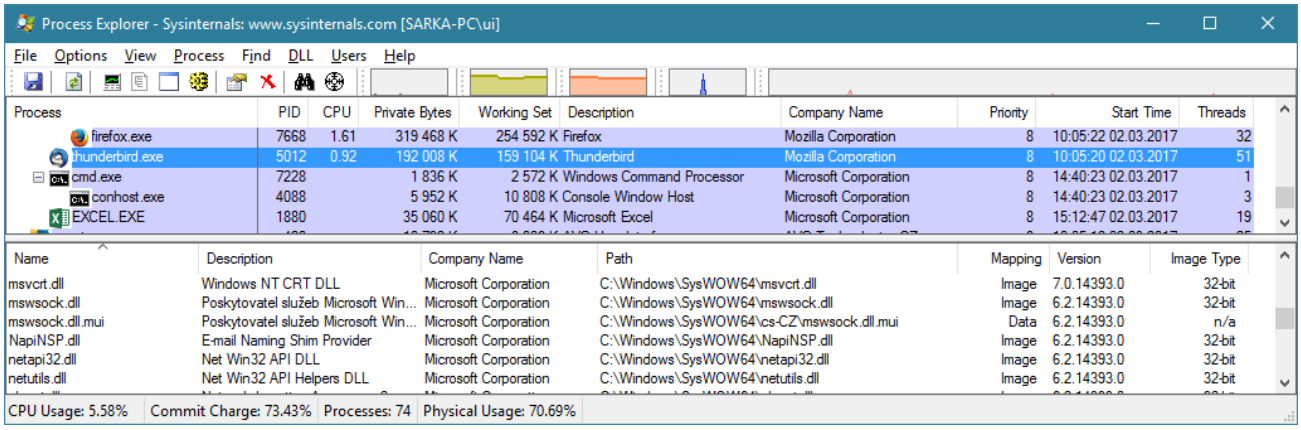
3.3 Programové rozhraní

3.3.1 Dynamicky linkované knihovny

Dynamicky linkované knihovny (DLL) používají především programátoři. Aby bylo možné DLL knihovnu používat v aplikaci, je nutné ji nejdřív načíst (zavést), pak je knihovna namapována do adresového prostoru aplikace a její funkce a objekty můžeme volně používat (jen musí být deklarovány jako externí).

Typické přípony souborů dynamicky linkovaných knihoven jsou DLL, TTF (True Type Font), FON (staré rastrové fonty), NLS (National Language Services, ovladač pro jazyk), OCX (ActiveX Server). Jako dynamickou knihovnu lze použít také EXE soubor nebo ovladač.

Seznam DLL knihoven, které jsou nalinkovány v konkrétním procesu, získáme například v programu *Process Explorer* (viz obrázek 3.8), když v menu zvolíme *View* ⇨ *Show Lower Pane*, hned následující položka určuje, zda se zobrazí DLL knihovny nebo manipulátory (handle) procesu.



Obrázek 3.8: Seznam dynamických knihoven používaných procesem `thunderbird.exe`

3.3.2 Operace s knihovnami a jinými soubory

Kontrola systémových souborů včetně systémových dynamicky linkovaných knihoven se provádí příkazem `SFC`. Zkontroluje, zda nedošlo k pozměnění či poškození systémových souborů (s příponou SYS, DLL, EXE, OCX, FON, TTF), používá kontrolu digitálního podpisu. Tento program je třeba spouštět s oprávněním správce.

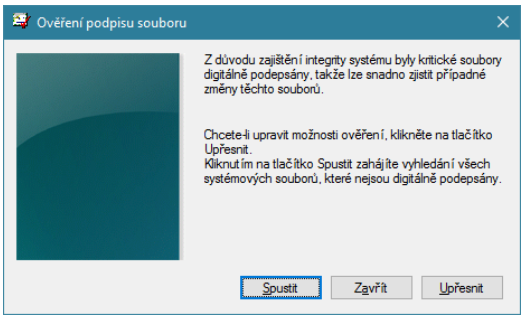
`sfc /scannow` okamžitě zkontroluje všechny systémové soubory, bude to vypadat nějak takto:

```
Beginning system scan.  This process will take some time.

Beginning verification phase of system scan.
Verification 100% complete.

Windows Resource Protection found corrupt files and successfully repaired them.
For online repairs, details are included in the CBS log file located at
windir\Logs\CBS\CBS.log. For example C:\Windows\Logs\CBS\CBS.log. For offline
repairs, details are included in the log file provided by the /OFFLOGFILE flag.
```

`sfc /scanboot /quiet` pravidelně při každém spuštění systému zkontroluje systémové soubory, navíc nebude uživatele informovat o případném nahrazování (tichý režim, quiet)

`sfc /scanonce` provede okamžitou rychlou jednorázovou kontrolu; použijeme, pokud máme podezření, že některý systémový soubor není zcela v pořádku

Kontrola digitálních podpisů se provádí, pokud chceme zjistit, zda jsou všechny ovladače a systémové soubory správně digitálně podepsány. Slouží k tomu program `sigverif` (spustíme jednoduše zadáním tohoto řetězce), přičemž se jedná o nástroj s grafickým rozhraním – viz obrázek 3.9.



Obrázek 3.9: Nástroj sigverif pro ověření digitálních podpisů

Klepnutím na tlačítko *Upřesnit* můžeme určit, do kterého souboru se má uložit výsledek testování, tlačítkem *Spustit* spustíme testování. Ve starších verzích bylo také možné určit konkrétní cíl testování (třeba typy souborů nebo složku, jejíž obsah má být otestován), v novějších verzích už tato možnost není.

Registrace dynamicky linkované knihovny: aby mohla být dynamicky linkovaná knihovna obsahující COM komponenty (také ActiveX prvky) používána, musí být v systému registrována. K registraci dochází obvykle během instalace aplikace, která knihovnu do systému přináší, a to příkazem `regsvr32`. Podobně při odinstalaci bývá odregistrována.

Postup registrace:

```
regsvr32 %ProgramFiles%\aplikace\knihovna.dll
```

(doplníme cestu k dynamické knihovně). Deregistrace (uvolnění knihovny):

```
regsvr32 /U %ProgramFiles%\aplikace\knihovna.dll
```

Poznámka

Opětovná registrace knihovny někdy může vyřešit nefunkčnost služby nebo programu. Někdy se stává, že služba Windows Update přestane fungovat a řešením je opětovná registrace knihoven, které služba využívá, popřípadě naopak po aktualizaci systému či některé aplikace může být nutné provést opětovnou registraci některých knihoven.

Rundll32: Program `Rundll32.exe` známe už z předchozího semestru. Víme, že pomocí tohoto programu lze z Příkazového řádku (nebo ze skriptu) volat procedury uložené v některé dynamicky linkované knihovně (ostatně, zamyslete se nad názvem programu). Nelze volat jakoukoliv proceduru, volané procedury musí být k tomu účelu naprogramovány.[1]

Tento program můžeme používat i vzdáleně. Například:

`rundll32 printui.dll,PrintUIEntry /?` získáme nápovědu k proceduře s názvem `PrintUIEntry`, která je uložena v knihovně `printUI.dll` (pozor, v názvu procedury je nutné zachovávat velká a malá písmena, názvy procedur jsou case-sensitive); výstupem tohoto příkazu je okno (ano, opravdu okno) s obsahem (zde zkráceným):

```
Použití: rundll32 printui.dll,PrintUIEntry [parametry] [@soubor_příkazů]
   /a[soubor] Název binárního souboru
   /b[název] Název základní tiskárny
   /c[název] Název UNC počítače, jde-li o příkaz pro vzdálený počítač
   /dl Odstranit místní tiskárnu
   /dn Odstranit připojení místní tiskárny
   /dd Odstranit ovladač tiskárny
  ...
```

(součástí výpisu jsou také příklady). Tímto příkazem (a vhodnými parametry, kterých je opravdu hodně) můžeme dělat se vzdálenou tiskárnou téměř cokoliv, včetně nastavení popisovače zabezpečení, instalace a odinstalace ovladače, atd.

`rundll32 printui.dll,PrintUIEntry /p /n\\počítač\tiskárna` vypíše informace o zadané tiskárně,

`rundll32 PowrProf.dll,SetSuspendState` přechod do úsporného režimu (tento příkaz můžeme například uložit do zástupce na pracovní plochu, pak stačí poklepat na ikonu zástupce a okamžitě přejdeme do úsporného režimu),

`rundll32 iedkcs32.dll,CloseRASConnections` způsobí odpojení počítače od internetu (to je ideální pro vytvoření zástupce na ploše nebo naplánování spuštění úlohy na zadanou dobu),

[1] Programování procedur v knihovnách tak, aby mohly být volány přes Rundll32, je stručně popsáno v dokumentu https://learn.microsoft.com/cs-cz/windows-server/administration/windows-commands/rundll32.

`rundll32 advapi32.dll,ProcessIdleTasks` způsobí okamžité provedení procesů, které čekají na událost *OnIdle*, tedy na dobu, kdy žádný jiný proces nemá přiřazen procesor (týká se to především procesů, které provádějí pravidelnou údržbu Windows).

Další informace

Možnost volat procedury systémových knihoven není moc dobře zdokumentována. Především je problém najít seznamy procedur, které lze takto volat v jednotlivých knihovnách. Jeden ze způsobů, jak tyto informace najít, je zadat do vyhledávacího řádku Googlu `rundll32 knihovna site:microsoft.com` (doplníme název knihovny), například `rundll32 shell32.dll site:microsoft.com`.

Úkoly

1. Pokud máte k dispozici *Process Explorer* (pokud nemáte, stáhněte ho z webu), spusťte ho, nastavte zobrazení spodního podokna se seznamem knihoven procesu a prohlédněte si knihovny (i jejich vlastnosti) pro spuštěné aplikace. K vlastnostem knihovny se dostanete poklepáním na název knihovny.

   Všimněte si řádku *Mapping* (na první záložce vlastností dynamické knihovny) – u běžné dynamické knihovny je zde *Image*, u datové knihovny (třeba TTF nebo NLS) tu najdeme *Data*. Oba typy knihoven najdete například na řádku `csrss.exe` nebo pokud máte spuštěný Poznámkový blok, u `notepad.exe`.
2. Zjistěte (příkazem pro okamžitou kontrolu), zda všechny systémové soubory včetně dynamicky linkovaných knihoven jsou v pořádku.

3.3.3 Lokální verze knihoven

DLL Hell (Peklo DLL) je stav, kdy některá aplikace při své instalaci pozmění (nahradí) některou DLL knihovnu, ale tato změna začne vadit jiným aplikacím, které ke svému běhu vyžadují dřívější verzi pozměněné knihovny. Tyto případy se stávaly dokonce i při aktualizaci systémových knihoven přes službu Windows Update.

Tento problém je v současné době řešen (u systémových knihoven) dvěma opatřeními:

1. verze knihoven jsou pravidelně kontrolovány a při zjištěné modifikaci nahrazovány originálními verzemi (program SFC a složka DLLCache), toto opatření je určeno pro systémové knihovny,
2. zavedení tzv. *lokálních verzí* (soukromých kopií) *knihoven*, toto opatření je určeno pro jiné než systémové knihovny.

Lokální verze knihoven si může soukromě vytvářet každá aplikace ve svém instalačním adresáři, ale je možné vytvářet lokální verze určené ke sdílení. Lokální verze ke sdílení jsou ve složce `...\Windows\WinSxs`. V této složce najdeme

- podsložky obsahující soubory s knihovnami, názvy podsložek obsahují identifikaci názvu knihovny, hardwarovou architekturu, verzi a další informace (v jedné podsložce může být i víc než jedna knihovna),
- podsložka `Manifest` obsahuje pro každou podsložku z předchozího bodu dva soubory (`xxx` představuje název podsložky z předchozího bodu): `xxx.manifest` (XML soubor s podrobnou informací

o uložených knihovnách) a `xxx.cab` (katalogový soubor s bezpečnostními informace, především digitálním podpisem),

- `Policies` – pro některé položky z prvního bodu tohoto seznamu je zde podsložka (ne nutně stejně pojmenovaná) obsahující jednu nebo více dvojic souborů – ve dvojici jeden soubor katalogu CAT a soubor `xxx.policy`, kde `xxx` je označení verze knihovny (jedná se o XML soubor s nastavením zásad používání knihovny).

Ve výše uvedených souborech se setkáme s pojmem *assembly*. Assembly je *sestava* zahrnující sdílené prostředky (jednu nebo více knihoven), manifest a případně katalogový soubor.

3.3.4 Win API

Programové rozhraní Windows je realizováno přes Win API (Applications Programming Interface), což je souhrn prostředků a procedur (funkcí), které mohou programy používat. Win API je uloženo v dynamicky linkovaných knihovnách – DLL, některé EXE, TIFF, ..., například funkce pro vykreslení určitého dialogového okna se zadaným textem a tlačítky.

API je vlastně přeložený, binární kód, tentýž jako u spustitelných (EXE) souborů, dynamicky linkované knihovny mají strukturu hodně podobnou struktuře EXE souborů. Existuje víc verzí API. V novějších Windows najdeme zejména:

- *Win32 API* je pro 32bitové aplikace (většina kódu je v knihovnách `kernel32.dll`, `advapi32.dll`, `user32.dll`, `gdi32.dll`, `comctl32.dll`, `comdlg32.dll`, `shell32.dll` a dalších), přímo jsou používány modulem `csrss.exe` (část podsystému Win32 pro uživatelský režim) a `win32k.sys` (pro režim jádra).
- *Win64 API* plní podobnou úlohu jako Win32, ale na 64bitových systémech, je uloženo ve stejných knihovnách.
- *.NET API*

Nízkoúrovňové API pro komunikaci s jádrem je `ntdll.dll`.

Nenativní aplikace využívají API ze svého podsystému či virtuálního stroje. K překladu API dochází také při běhu 32bitové aplikace na 64bitovém systému, používá se podsystém *WoW64* (což znamená Windows 32bit on Windows 64bit).

Poznámka

Implementací Win API je také například podsystém Wine používaný v UNIXových systémech pro běh Win aplikací, další zajímavá implementace je například ReactOS.

Jinak jde o pojem poměrně hodně používaný, API jednoduše znamená programové rozhraní (k čemukoliv). Své API mají také grafické a zvukové karty. Můžeme se například setkat s pojmem DirectX API nebo OpenGL API, obojí je rozhraní k multimediálním instrukcím implementovaným například v grafických čipech.

Další informace

Podrobné informace o Win32 API můžeme získat například na

- https://learn.microsoft.com/en-us/windows/win32/api/
- https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list
- https://stevedonovan.github.io/winapi/api.html

Zajímavým projektem Microsoftu je .NET API Browser (https://learn.microsoft.com/en-us/dotnet/api/, kde najdeme nejen referenční příručku samotného .NET API, ale i spousty dalších. Stačí si vybrat v rozbalovacím seznamu.

Úkoly

1. Najděte domovskou stránku projektu ReactOS a zjistěte, o jaký systém se vlastně jedná.
2. Zjistěte, ve které složce se nacházejí soubory, které byly v této sekci zmíněny.

3.4 Kompatibilita verzí Windows

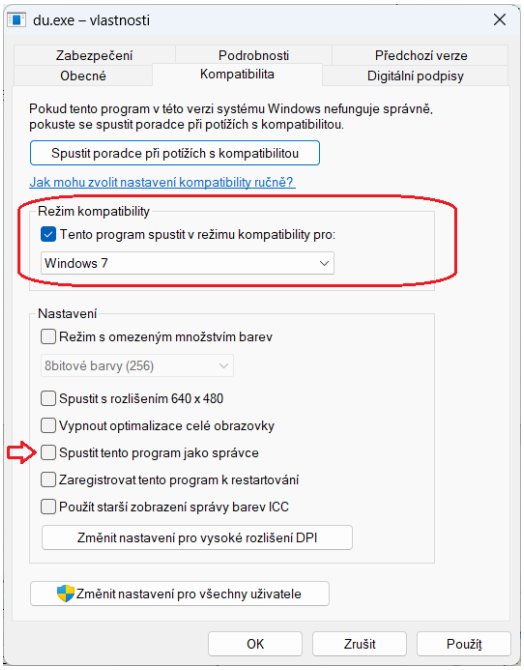
V souvislosti s provozováním aplikací pro různé verze Windows v jednom systému se setkáváme se zkratkou *WoW* (Windows on Windows) – jde o emulační platformu ve Windows pro starší Windows aplikace. Například WoW64 je emulační platforma na 64bitových Windows pro 32bitové aplikace. Tato zkratka se vyskytuje také v registru.

32bitové aplikace pro Windows je obvykle možné spouštět na všech 32bitových Windows včetně řady NT. Výjimkou jsou programy, které potřebují přímý přístup k prostředkům nebo „trvají“ na umístění systémových souborů charakteristickém pouze pro určité verze systému, případně hodně starých verzích knihoven z API. Řešením, které nezabírá vždy, je vhodné nastavení parametrů spouštění aplikace ve *Vlastnostech* v kontextovém menu.

Ve Windows můžeme nastavit *režim kompatibility*. Nastavuje se ve vlastnostech takovéhoto spustitelného souboru – v kontextovém menu *Vlastnosti* ⇨ karta *Kompatibilita*, viz obrázek 3.10 – tam máme možnost si v rozbalovacím seznamu vybrat „simulovaný“ systém, aplikace pak reálně poběží v jakémsi virtuálním počítači.

Některé starší programy dokážou běžet jen s oprávněním správce, ale tomuto nastavení je dobré se z bezpečnostních důvodů spíše vyhnout.

Další možností je provozovat „cizí“ aplikaci ve virtuálním počítači. Ideální situace by pak byla v případě, že vybraná virtualizační platforma umí tzv. „bezešvý mód“, tedy okna virtuálně běžící aplikace jsou zařazena na stejné úrovni jako okna běžných aplikací a z pohledu uživatele se s nimi dá zacházet zcela stejně.



Obrázek 3.10: Nastavení režimu kompatibility

Úkol

Pokud máte možnost, vyzkoušejte výše popsané postupy s kompatibilitou.

3.5 Správa služeb

3.5.1 Jak služby fungují

Služba je proces, který slouží k podpoře funkcí jiných procesů a obvykle pracuje na pozadí systému (tj. nejde o interaktivní procesy, uživatel do jejich činnosti většinou nezasahuje). Od běžných procesů se liší v několika aspektech:

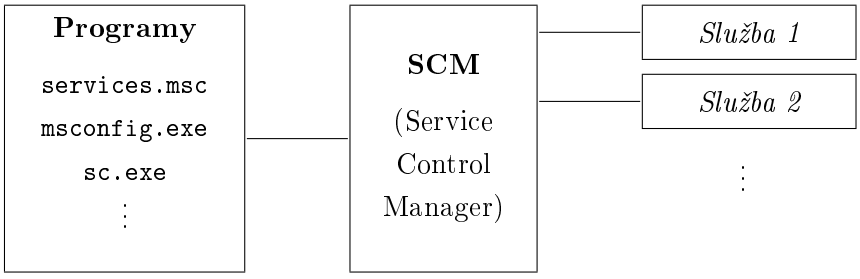
- běží na pozadí systému,
- přístupová oprávnění služeb bývají odlišná od přístupových oprávnění běžných procesů,
- běh služeb je nezávislý na přihlašování a odhlašování uživatelů (služby mohou běžet i tehdy, když žádný uživatel není přihlášen),
- existují nástroje určené speciálně pro místní a vzdálenou správu služeb, také v registu najdeme služby na specifických místech, nepoužíváme na ně nástroje určené pro správu procesů.

Ke službám se ve skutečnosti řadí také *ovladače*, tedy ovladač je vlastně speciální druh služby sloužící jako rozhraní k zařízení (včetně virtuálních zařízení). K ovladačům můžeme přistupovat stejným způsobem jako ke službám.

Služby používají tyto účty pro stanovení svých přístupových oprávnění:

- *Local System* – účet pro systémové procesy, služba má plný přístup k čemukoliv, v seznamech se také může zobrazovat jako *System*, resp. *NT AUTHORITY/SYSTEM*,
- *Local Service* – pro služby, které nepotřebují zvýšená přístupová oprávnění, k místním prostředkům mají stejný typ přístupu jako běžný uživatel, k síti používají jen anonymní přístup,
- *Network Service* – pro služby, které potřebují autorizovaně pracovat na síti,
- *Doménový nebo lokální účet* – podobně jako u kteréhokoliv jiného procesu, lze nastavit oprávnění.

Služby (a všechny další procesy) pracující pod účtem Local System mají velmi rozsáhlá oprávnění, s určitým omezením: používají uživatelský profil uložený ve větvi registru `HKU/.Default`, a nemohou přistupovat do uživatelských profilů jiných uživatelů v dalších podklíčích klíče `HKU`.



Obrázek 3.11: Architektura služeb

Na obrázku 3.11 je znázorněna architektura služeb. Veškerá komunikace se službami (včetně jejich konfigurace) probíhá přes modul *SCM* (Service Control Manager – správce služeb), který je fyzicky představován souborem `services.exe`.

SCM tedy spravuje databázi služeb a ovladačů a zprostředkovává k nim přístup. Databáze je zpřístupněna také přes registr, najdeme ji v klíči
`HKLM/System/CurrentControlSet/Services`.
V tomto klíči jsou podklíče nazvané podle krátkých jmen služeb a ovladačů (každá služba má dlouhé a krátké jméno). V podklíčích pak jsou všechny údaje související s danou službou či ovladačem.

Najdeme zde obvykle tyto položky:

- *DisplayName* – dlouhé jméno (krátké jméno známe z názvu klíče, dlouhé může být v jiném jazyce než v angličtině),
- *Description* – popis služby či ovladače,
- *ImagePath* – cesta k souboru (spustitelný soubor, soubor ovladače, dynamická knihovna apod.),
- *ObjectName* – název účtu, pod kterým se má služba spustit (u ovladačů tuto položku nenajdeme, protože jsou vždy spouštěny s vysokými přístupovými oprávněními, pracují v režimu jádra), pokud u služby není tento klíč uveden, jde o účet LocalSystem,
- *Tag* – pořadové číslo, které určuje pořadí spouštění služeb (je jen u těch služeb, které závisejí na jiných službách a je tedy důležité, v jakém pořadí jsou spouštěny),
- *Type* – typ služby nebo ovladače (je to číselná hodnota), například
  - 1 (Service Kernel Driver) je ovladač zařízení pracující v režimu jádra,
  - 2 (Service File System Driver) je ovladač souborového systému (například NTFS),
  - 16 (Service Win32 Own Process) je služba, která má vlastní proces (žádná jiná služba v tomto procesu neběží), může být spuštěna z vlastního spustitelného souboru nebo může jít o modul spouštěný procesem `svchost.exe`,
  - 32 (Service Win32 Share Process) je služba, která je součástí procesu hostujícího více služeb,
  - 272 (Service Win32 Own Process Interactive) podobně jako 16, ale může pracovat interaktivně (například má konfigurační nástroj s GUI – oknem),
  - 288 (Service Win32 Share Process Interactive) podobně jako 32, ale může pracovat interaktivně,

  o řešení uživatelského rozhraní pro interaktivní služby jsme se učili v sekci o objektech na straně 55, všechny interaktivní služby musí být spuštěny pod účtem Local System,
- *Start* – kdy má být služba načtena, například
  - 0: služba či ovladač se načítá při zavádění systému (se zavaděčem systému),
  - 1: při inicializaci jádra (tj. po všech službách s hodnotou 0),
  - 2: spouští se automaticky hned po startu systému,
  - 3: lze ji spustit, ale jen ručně (služba není spuštěna automaticky),
  - 4: služba je zakázána (nelze ji spustit ani ručně),
- *Group* – služba může patřit do některé skupiny služeb,
- *DependOnService*, *DependOnGroup* – služba může být závislá na službě nebo skupině služeb,
- atd.

Většina služeb a ovladačů má ve svém klíči podklíče s dalšími, specifickými informacemi vztahujícími se k zabezpečení nebo konfiguraci.

Poznámka

K zápočtu není nutné umět odříkat všechny parametry včetně hodnot položek Start a Type, jen je důležité vědět, kde tyto informace najít.

Další informace

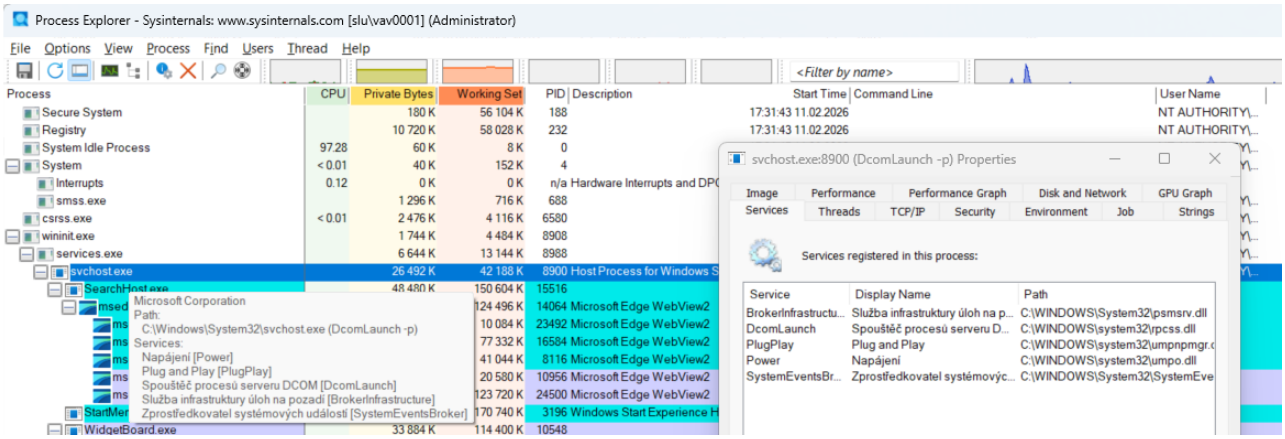
Podrobnosti o dalších možných položkách zjistíme například na adrese https://learn.microsoft.com/en-us/windows-hardware/drivers/install/hklm-system-currentcontrolset-services-registry-tree

Úkol

Spusťte konzolu *Služby*. Vyberte si kteroukoliv službu (například *Klient DHCP*) a zobrazte její vlastnosti. Zjistěte, jestli běží, zda se spouští automaticky, pod jakým účtem se přihlašuje, na kterých službách závisí a které služby závisejí na ní pak tyto údaje najděte v registru.

3.5.2 Sdílené procesy služeb, *Service Host*

Jak bylo výše uvedeno, několik služeb může běžet v jediném procesu. Je to především z důvodu šetření systémovými prostředky (každý proces zabere poměrně hodně paměti a času procesoru, služba jako taková však moc prostředků nepotřebuje, zvláště když není interaktivní).



Obrázek 3.12: Zobrazení služeb v procesu

Seznam služeb, které hostí zadaný proces, zjistíme například v Process Exploreru, a to hned dvěma možnými způsoby – buď podržíme myš nad názvem procesu (žluté okno na obrázku 3.12) anebo zobrazíme vlastnosti procesu (poklepáním), služby najdeme na kartě *Services* (podokno na tomtéž obrázku).

Procesy služeb s vlastním procesem (obvykle typu 16 podle výše uvedeného výčtu) jsou obvykle potomky (ve stromové struktuře) procesu `services.exe`, například proces `spoolsv.exe` pro službu *Zařazování tisku*. Stejně je tomu i u procesů, které hostí více služeb.

Služby mohou běžet v některém systémovém procesu, například některé služby najdeme v procesu SCM (`services.exe`) nebo LSASS (`lsass.exe`). Nejběžnějším „kontejnerem“ pro služby je však proces *Service Host* (`svchost.exe`). Služby takto spouštěné jsou uloženy v DLL knihovnách.

V Process Exploreru (také na obrázku 3.12) vidíme, že je spuštěno více instancí procesu Service Host. Liší se především ve dvou vlastnostech:

- účet, pod kterým proces běží (sloupec *User Name*), obvykle System (Local System), Network Service nebo Local Service,
- Command Line (příkaz včetně parametrů, kterým byl proces spuštěn), žádné dvě instance by neměly mít tuto položku stejnou.

Příklad

To, ve které instanci programu Service Host bude služba spuštěna, záleží na řetězci uvedeném v registru v klíči *HKLM/SYSTEM/CurrentControlSet/Services/⟨služba⟩* v položce *ImagePath* (tedy cesta ke spustitelnému souboru pro službu). Například služby *AppMgmt* (Správa aplikací), *CryptSvc* (Šifrování), *Dhcp* (Klient DHCP) a další mají tento řetězec

%SystemRoot%\system32\svchost.exe -k netsvcs

to znamená, že všechny budou běžet ve stejném procesu s touto příkazovou řádkou (můžeme si ověřit v Process Explorerovi nebo Správci úloh, když necháme zobrazit sloupec *Command Line*).

Úkoly

1. V registru najděte klíč obsahující podklíče služeb (a ovladačů). Najděte následující podklíče a příslušné služby roztřiďte podle jejich typu. U každé zjistěte, jaké je její dlouhé jméno, kdy a jak se spouští, ve kterém souboru je uložena, pod jakým účtem se spouští (tj. jaká má přístupová oprávnění).

appmgmt	EventSystem	null	spooler
cdfs	ipsec	PlugPlay	tcpip
cdrom	MountMgr	RemoteAccess	wmi
cisvc	netlogon	SamSs	wuauserv

2. Ze seznamu v předchozím bodu si vyberte některou interaktivní službu, která je spuštěna. Spusťte Process Explorer a zobrazte ve spodním podokně manipulátory (handle). Najděte v něm proces vybrané služby a zjistěte, v jaké stanici oken je napojen (záznam bude zřejmě někde ke konci seznamu handlů procesu).
3. V Process Exploreru zjistěte, které služby jsou spouštěny řetězci

   `svchost -k rpcss`

   `svchost -k netsvcs`

   Prohlédněte si vlastnosti těchto procesů, ve vlastnostech projděte všechny karty.

3.5.3 Program `sc.exe`

Program `sc.exe` je součástí Windows od verze XP (Server 2003). Jeho účelem je pokročilá konfigurace služeb a dostaneme se prakticky k jakýmkoliv nastavením, která pro služby (obecně) platí. Podobně jako příkaz `net`, i zde máme sadu podpříkazů, které upřesňují chování příkazu samotného.

`sc query`
: zobrazí informace (podáváme dotaz)

  `sc query` zobrazí informace o všech službách (hodně dlouhý výpis)

  `sc query webclient` zobrazí informace o službě WebClient (zadáváme vždy jen krátký název služby)

  `sc query type= interact` vypíšeme všechny interaktivní služby (pozor, před rovnítkem není mezera, za ním naopak musí být)

  `sc query type= filesys` vypíšeme všechny ovladače souborových systémů (je jich poměrně hodně)

`sc \\počítač query lmhosts` vypíše informaci o zadané službě běžící na zadaném počítači

`sc qc, sc queryex`

zobrazí podrobnější informace zaměřené spíše na umístění a přístupová oprávnění či na proces

`sc qc rpcss` zobrazí tento typ informací o službě Vzdálené volání procedur

`sc queryex rpcss` z tohoto výpisu zjistíme například PID procesu, ve kterém služba běží

`sc create, sc delete`

vytvoří pro zadanou (novou) službu položku v registru, aby mohla být spouštěna; tento příkaz slouží ke zprovoznění služby a ovladače, je součástí instalace

`sc create novaSluzba binpath= c:\windows\system32\soubor.exe`
`type= own start= auto` vytváříme novou službu, zadali jsme její spustitelný soubor, typ služby (služba s vlastním spustitelným souborem) a požadavek na automatické spouštění při startu systému; můžeme zadat také další položky včetně závislostí na jiných službách či jejich skupinách, uživatelský účet, zobrazované jméno, atd., prakticky cokoliv bývá v příslušných položkách registru

`sc delete novaSluzba` odinstalujeme službu (ve smyslu odstranění údajů z registru)

`sc config`

upraví konfiguraci služby či ovladače (konkrétně hodnoty v registru)

`sc config spooler start= auto` změní konfiguraci spuštění zadané služby Zařazování tisku na automatické spouštění při startu systému

`sc enumdepend`

zjišťujeme, které další služby závisejí na zadané službě (chceme vědět, které služby by nefungovaly, kdybychom zadanou službu ukončili)

`sc enumdepend samss` zjišťujeme závislosti na službě Správce zabezpečení účtů

`sc enumdepend rpcss` Zjišťujeme totéž pro službu Vzdálené volání procedur

Možných podpříkazů je více (zjistíme je v nápovědě). Můžeme například službu pozastavit a pak ji kdykoliv odblokovat, nastavit akce při selhání služby, přinutit správu služeb k aktualizaci stavu služby, pokud byly provedeny změny a neprojevují se. Lze také pracovat s popisovači zabezpečení a také zamykat nebo odemykat databázi služeb na zadaném počítači.[2]

Příklad

Budeme pracovat se službou *Klient DHCP*. V konzole `services.msc` lze zjistit, že krátký název služby je `dhcp`, spouští se příkazem `svchost.exe -k netsvcs`, tedy běží v hostitelském procesu služeb a nemá svůj vlastní proces, spouští se automaticky, je spuštěna, proces má přístupová oprávnění „místní systémový účet“ (tj. Local System), závisí na třech dalších službách (AFD, Ovladač protokolu TCP/IP, Rozhraní NetBios nad protokolem TCP/IP). Údaje jsou z Windows XP, v jiných verzích mohou být odlišné.

Podíváme se na výstupy příkazu `sc`:

```
C:\> sc query dhcp

SERVICE_NAME: dhcp
        TYPE               : 20  WIN32_SHARE_PROCESS
```

[2] Důsledkem zamknutí databáze služeb je například to, že SCM nemůže od té chvíle na daném počítači spustit žádnou službu, tedy můžeme tento postup použít při synchronizaci – pokud chceme provést konfiguraci služeb (třeba i vzdáleně) na daném počítači, zamkneme databázi služeb, provedeme konfiguraci a pak databázi odemkneme.

```
        STATE              : 4  RUNNING
                                (STOPPABLE,NOT_PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\> sc queryex dhcp

SERVICE_NAME: dhcp
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE,NOT_PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 1260
        FLAGS              :

C:\> sc qc dhcp

[SC] GetServiceConfig SUCCESS

SERVICE_NAME: dhcp
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\WINDOWS\system32\svchost.exe -k netsvcs
        LOAD_ORDER_GROUP   : TDI
        TAG                : 0
        DISPLAY_NAME       : Klient DHCP
        DEPENDENCIES       : Tcpip
                           : Afd
                           : NetBT
        SERVICE_START_NAME : LocalSystem
```

Jak vidíme, mezi prvními dvěma výpisy není velký rozdíl, v druhém máme navíc dva poslední řádky (PID procesu, ve kterém služba běží, a pak příznaky, ale tato služba žádné nemá). Oproti tomu třetí výpis již odlišný je. Dozvíme se v něm tak důležité věci jako je třeba typ spuštění služby (automaticky), příkaz, kterým je spuštěna, skupina, pořadí při spouštění, zobrazované jméno (česky), závislosti a účet, pod kterým je služba přihlášena.

Závislosti „opačným směrem“ – tedy které služby závisejí na této službě – zjistíme takto:

```
C:\> sc enumdepend dhcp

Enum: entriesRead  = 0
```

Na této službě tedy žádná jiná služba nezávisí. Jinak je tomu například u služby *Správce zabezpečení účtů* (samss):

```
C:\> sc enumdepend samss

Enum: entriesRead  = 1

SERVICE_NAME: MSDTC
DISPLAY_NAME: Koordinátor DTC
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
                                (NOT_STOPPABLE,NOT_PAUSABLE,IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 1077       (0x435)
```

```
SERVICE_EXIT_CODE  : 0  (0x0)
CHECKPOINT         : 0x0
WAIT_HINT          : 0x0
```

Úkoly

1. Vyzkoušejte dotazovou variantu příkazu `sc`, podle výše uvedených ukázek použití.
2. Najděte v nápovědě celkovou syntaxi příkazu `sc`. Zjistěte, jak lze některou běžící službu pozastavit a pak znovu odblokovat.
3. Pro práci se službami v Powershellu slouží cmdlety, které mají ve svém názvu řetězec „service“. Vypište si jejich seznam pomocí `get-command *service*`.

   Vypište všechny právě běžící služby: `get-service | where-object {$_.status -eq "running"}`

3.5.4 Příkaz `NET` – práce se službami a další úlohy

Příkaz `net` má speciální podpříkazy pro práci se službami *server* a *workstation*, ale také další podpříkazy pro práci se službami obecně (především spuštění a zastavení služby z příkazového řádku).

`NET CONFIG`

umožňuje konfigurovat službu *Server* nebo *Workstation* (podle dalšího parametru, ta druhá se v české variantě systému může jmenovat *Pracovní stanice*)

`net config workstation` zobrazí momentální konfiguraci služby *Workstation* (například název počítače, doména, a také parametry pro posílání dat na síť)

`net config workstation /charcount:32` zajistí, aby při shromažďování dat posílaných na síť systém vyčkal, dokud data nejsou v uvedeném množství (zde 32 B)

`net config server` zobrazí momentální konfiguraci služby *Server* (na rozdíl od *Workstation* se většinou týkají relací přihlášených uživatelů – kolik uživatelů maximálně může být přihlášeno, kolik souborů může být otevřeno v rámci jedné relace, zda je server skrytý, apod.)

`net config server /autodisconnect:10 /hidden:yes` nečinná relace je po 10 minutách nečinnosti ukončena, a také zajišťujeme, že nás server „není vidět“ v seznamu počítačů v síti (ale dá se k němu dostat, podle přístupových oprávnění uživatelů, když víme, jak se jmenuje)

`net config server /srvcomment:"Počítač v garáži" /hidden:no` nastavíme řetězec, který se zobrazuje vedle identifikace serveru (například také při použití příkazu `net view`), chceme, aby byl počítač běžně viditelný v síti (ovšem to je výchozí hodnota, obvykle není nutné ji měnit)

Poznámka

Při použití příkazu `net config server` potřebujeme vyšší přístupová oprávnění. Takže je nutné spustit příkazový řádek s oprávněními správce a v jeho okně pak můžeme zadat tento příkaz.

Příklad

Ukážeme si, jak vypadá výpis příkazu `net config server` na desktopu (předpokládejme, že služba Server je spuštěna, jinak by nebylo možné konfiguraci vypsat).

```
C:\> net config server

Název serveru                              \\PC152
Komentář serveru                           PC152

Verze softwaru                             Windows 10 Enterprise
Server je aktivní na
Server je aktivní na
        NetbiosSmb (SARKA-PC)
        NetBT_Tcpip_C53723A8-0890-4FA4-9E29-F078C907B74E (PC152)
        NetBT_Tcpip_88B670E2-53BB-4F8F-A3F3-5A32C48D044C (PC152)

Skrytý server                              Ne
Maximum přihlášených uživatelů             20
Maximum otevřených souborů na relaci       16384
Trvání nečinné relace (minuty)             15
Příkaz byl úspěšně dokončen.
```

Z výpisu je zřejmé, jak se počítač jmenuje, jaký je na něm operační systém, základní údaje o protokolech pro provoz místní sítě (NetBIOS, NetBT), u NetBT vidíme v lomené závorce ID síťové karty. Dále zjistíme, že server je v síti viditelný, maximálně 20 uživatelů na něm může být najednou přihlášeno (tj. je schopen vést max. 20 relací) a existuje také limit na množství otevřených souborů na jednu relaci. Po 15 minutách je neaktivní relace automaticky ukončena.

Tyto hodnoty jsou typické pro desktopovou instalaci a vpodstatě nám nic nebrání je změnit. Na skutečném serveru by stanovené limity byly značně kontraproduktivní, serverové varianty Windows je mají stanoveny značně odlišně.

`NET STATISTICS`

vypíše veškeré protokolované informace (tj. z protokolů) služby *Server* nebo *Workstation* (další parametr je workstation nebo server) o počtu přijatých nebo odeslaných dat, navázaná připojení, spuštěné relace, úspěšné a neúspěšné operace, chybná zadání hesla, různé typy chyb, atd.

`net statistic workstation` vypíše se statistika služby *workstation* (kolik dat bylo přeneseno, chyby přenosu, počet navázání spojení, relací, atd.)

`NET START`

spustí zadanou službu

`net start` zobrazí seznam všech služeb, které jsou právě spuštěné

`net start "terminálová služba"` spustí službu *Terminálová služba* (její spuštění musí být nastaveno na „Ručně“ nebo „Automaticky“; kdyby byl typ spouštění služby nastaven na „Zakázáno“, spuštění selže)

`net start "indexing service"` spustí službu *Indexing Service* – pokud název obsahuje mezery, musí být uzavřen do uvozovek

`net start cisvc` provede totéž (každá služba má kromě „lidského“ dlouhého názvu v češtině nebo angličtině také krátký univerzální název bez mezer a diakritiky, tedy se nemusíme obtěžovat s uvozovkami)

`NET STOP`

zastaví zadanou službu

`net stop "indexing service"` zastaví službu *Indexing Service*

`net stop cisvc` provede totéž (zastaví službu *Indexing Service*), zadali jsme zkrácený název služby, který je stejný ve všech jazykových variantách Windows

Úkoly

1. Zjistěte momentální konfiguraci služeb *Workstation* a *Server*.
2. Vypište statistiky související se službami *Workstation* a *Server*. Zjistěte, zda někdo nezadal chybné heslo.
3. Vypište seznam spuštěných služeb. Dále si (například v nástroji `services.msc`) vyberte některou službu, která není spuštěná, ale zároveň její spuštění není zakázáno, a pomocí příslušného příkazu ji spusťte (nezapomeňte – krátký název). Potom ji ukončete.

3.5.5 Služby v PowerShellu

Už v první kapitole jsme se v sekcích o PowerShellu zaměřili kromě procesů také na služby. Víme tedy, že si lze snadno zjistit seznam příkazů pro práci se službami:

```
get-command *service*
```

V seznamu najdeme příkazy pro výpis seznamu služeb (`GET-SERVICE`), spuštění služby (`START-SERVICE`), restartování služby `RESTART-SERVICE` a další. Také už víme, jak si pomocí různých technik ve výstupu vyfiltrovat konkrétní služby, případně ke službám konkrétní vlastnosti.

Úkol

Zjistěte, jakým příkazem lze restartovat některou konkrétní službu.

3.6 WBEM

3.6.1 Princip a implementace WBEM

WBEM (Web-Based Enterprise Management) je skupina technologií a postupů vytvořená za účelem sjednotit správu distribuovaných počítačových prostředí (tj. včetně vzdálené správy, ale také lokálně na jednom počítači). Správa se provádí buď přes webové rozhraní nebo pomocí specializovaných shellů.

Data jsou organizována v modelu *CIM* (Common Information Model), což je vlastně objektově orientovaná databáze, kde jednotlivé objekty jsou zapouzdřeny a přistupuje se k nim přes unifikované rozhraní. CIM je otevřený standard, tedy je možné ho jakkoliv využívat a dále upravovat.

Rozlišujeme *WBEM server* (ukládá a poskytuje informace, provádí příkazy) a *WBEM klienta* (reprezentován především rozhraním, se kterým pracuje administrátor, a příslušným API). Klient odesílá žádosti, server konfrontuje se skutečným stavem a provádí je (případně zajišťuje proces autentifikace a autorizace). Spolu komunikují přes protokol HTTP nebo HTTPS. Právě podle těchto protokolů je v názvu „web-based“.

Architektura je postavena na vzoru model–obraz. Klient čte data z „obrazu“, server přistupuje k „modelu“ reprezentovanému skutečným stavem hardwaru a softwaru v síti a při jakékoliv změně modelu aktualizuje obraz.

WBEM má více různých implementací, například:

WMI (Windows Management Instrumentation) je implementace Microsoftu pro Windows. Umožňuje (i vzdáleně) spravovat počítače s Windows v síti, je předinstalována na všech verzích od

Win 2000 (resp. Win ME). Je postavena na VBScriptu a PowerShellu, WMI těmto skriptovacím jazykům poskytuje přístup prakticky k čemukoliv ve Windows.

OpenWBEM od Novellu je implementace určená pro Novell Netware, Linux a některé jiné UNIXové systémy (včetně Solarisu a MacOSX). Je používána v komerční i nekomerční sféře a stejně jako WMI nabízí rozsáhlé možnosti zásahů do konfigurace sítě a monitorování. Je ke stažení na adrese http://www.openwbem.org/.

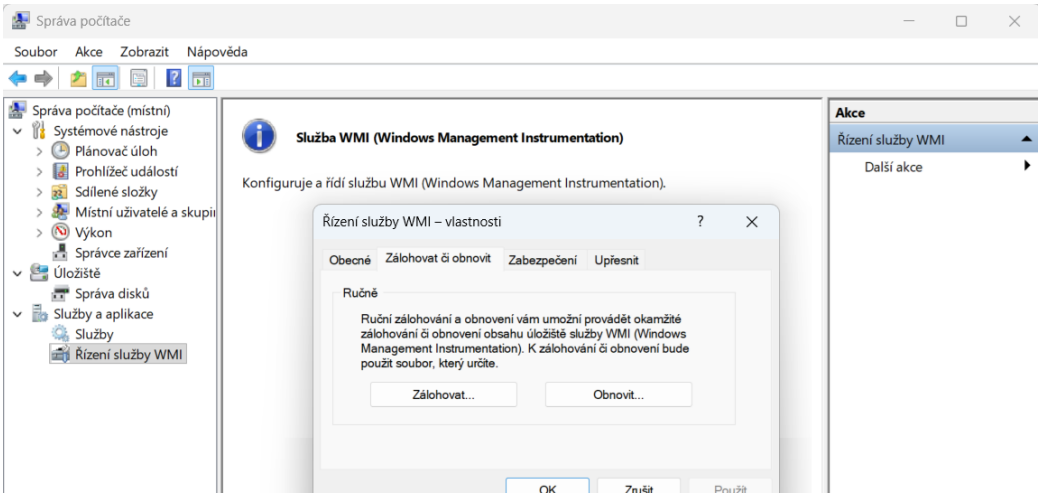
OpenPegasus je další otevřená implementace pro různé UNIXové systémy včetně Linuxu, a Windows. Je ke stažení na https://collaboration.opengroup.org/pegasus/.

3.6.2 WMI

Jak bylo výše uvedeno, WMI je implementace modelu WBEM pro správu Windows. Databáze CIM je taktéž objektová, oproti poměrně primitivním objektům, o kterých jsme se učili dříve, jde o objekty plnohodnotné s implementací tříd, vlastností, metod, událostí, využívající dědičnost a kompozici.

Třídy jsou psány v objektovém jazyce *MOF* (Managed Object Format), který je interpretovaný (dá se říct skriptovací). Většina komponent WMI (včetně tříd) je v adresáři `...\System32\Wbem`. Najdeme tam hodně souborů s příponou MOF.

Ke službě WMI lze přistupovat přes konzolu *Řízení služby WMI*. Spouštíme ji souborem `wmimgmt.msc`, ale je dostupná také v konzole *Správa počítače*. V kontextovém menu položky *Řízení služby WMI* zvolíme *Vlastnosti* a získáme okno, které vidíme na obrázku 3.13. Nastavujeme obecné vlastnosti řízení WMI jako je způsob protokolování a umístění protokolu, zálohování (máme možnost obnovit databázi WMI ze zálohy) a určujeme zabezpečení prvků databáze.



Obrázek 3.13: Řízení služby WMI

Rozhraní WMI se velmi často používá ve skriptech, včetně skriptů PowerShellu.

Poznámka

V novějších verzích Windows se WMI spouští jako služba v rámci procesu `svchost.exe`. Kdykoliv se spouští něco souvisejícího s WMI (například požadavek na data z databáze CIM), kód je spouštěn v procesu `wmiprvse.exe`, který je potomkem procesu hostícího službu RPC.

3.6.3 Program `wmic`

Program `wmic` (WMI Console) umožňuje využívat rozhraní WMI na Příkazovém řádku. pracuje ve dvou režimech – klasickém (externím) a interaktivním. Při použití v klasickém režimu zadáváme příkaz začínající názvem programu (`wmic`) následovaný parametry, do interaktivního režimu se dostaneme zadáním příkazu `wmic` bez dalších parametrů (prompt se změní na `wmic:root\cli>` a zadáváme interní příkazy).

Další informace

Nápovědu získáme buď v grafickém režimu, nebo příkazem `wmic /?`, anebo v interaktivním režimu této konzoly příkazem `/?`. Ještě podrobnější nápovědu zobrazíme příkazem `/?:FULL`.

Příkaz `wmic` je velmi komplexní. Má tuto syntaxi:
`WMIC přepínače předmět sloveso parametry`, kde

- *přepínače* nastavují obecné chování příkazu, mohou být například

  `/node:`*`počítač`* – příkaz bude proveden na zadaném počítači (před názvem počítače nebudeme dávat opačná lomítka)

  `/user:`*`uživatel`* – při vyhodnocování bude použit zadaný uživatel s jeho přístupovými oprávněními (budeme dotázáni na heslo)

  `/output:`*`soubor`* – výpis bude směrován do zadaného souboru
- *předmět* (také alias) určuje to, s čím chceme manipulovat nebo na co se dotazujeme, následuje *zkrácený* seznam (všechny možnosti zjistíme v nápovědě):

bios	diskdrive	memPhysical	onBoardDevice	registry
bootconfig	fsdir	netuse	OS	service
cpu	irq	NIC	pageFile	share
dcomApp	job	NICConfig	printer	temperature
desktop	memLogical	NTEvent	process	useraccount

- *sloveso* určuje požadovanou akci, která má být provedena s předmětem:
  - LIST – toto sloveso lze použít na všechny předměty, zobrazí obecnou informaci o předmětu, můžeme upřesnit, jak podrobné informace chceme (full – všechny, brief – základní v tabulce, writeable – které lze měnit, atd.),
  - GET – získání podrobnějších informací o všech nebo vybraných vlastnostech předmětu,
  - SET – změna vlastností předmětu,
  - ASSOC – vrátí instance zadaného objektu (tedy s ním asociované),
  - CREATE, DELETE – vytvoření nové instance, odstranění instance nebo třídy,
  - CALL – spuštění metody zadané třídy WMI.

Příklad

Příkaz *v neinteraktivním režimu* používáme takto:

`wmic bios list` vypíše se informace o BIOSu (úplná)

`wmic os list brief` stručná informace o operačním systému ve formě tabulky

`wmic os list full` úplná informace o operačním systému ve formě seznamu, výstup (zkrácený) bude vypadat nějak takto:

```
BootDevice=\ Device\ HarddiskVolume1
BuildNumber=2600
BuildType=Multiprocessor Free
CodeSet=1250
CountryCode=420
CSDVersion=Service Pack 3
CSName=nazevpocitace
CurrentTimeZone=60
...
FreePhysicalMemory=49996
FreeSpaceInPagingFiles=490728
FreeVirtualMemory=2054164
...
MaxNumberOfProcesses=-1
MaxProcessMemorySize=2097024
...
SystemDevice=\ Device\ HarddiskVolume1
SystemDirectory=C:\ WINDOWS\ system32
SystemDrive=C:
TotalSwapSpaceSize=
TotalVirtualMemorySize=2097024
TotalVisibleMemorySize=515516
Version=5.1.2600
WindowsDirectory=C:\ WINDOWS
```

Z výstupu se dá vyčíst poměrně hodně o nastaveních operačního systému (ve skutečnosti by tam bylo mnohem více řádků, z bezpečnostních a kapacitních důvodů je výstup značně zkrácen). Všimněte si řádku `MaxNumberOfProcesses=-1`. To znamená, že není omezen počet spouštěných procesů (například u Vista Starter by tady bylo jiné číslo).

`wmic diskdrive get model,size,interfacetype,mediatype` příkaz zobrazí seznam všech paměťových zařízení (i výměnných), a to vlastnosti z výčtu oddělené čárkou, výstup:

```
InterfaceType   MediaType                Model        Size
IDE             Fixed   hard disk media  ST3160812AS  160039272960
```

`wmic /output:D:\procinfo.txt cpu get` informace o procesoru (v tabulce) budou uloženy do zadaného souboru

`wmic cpu get > D:\procinfo.txt` totéž

`wmic service where state="running" get caption, name` vypíše seznam běžících služeb (pouze ty vlastnosti, které byly specifikovány), všimněte si syntaxe podobné SQL (ostatně pracujeme s databází)

`wmic process call create notepad.exe` spustí zadaný proces (zavolá proceduru create třídy `process` pro vytvoření procesu)

`wmic /node:`*`počítač`* `process call create notepad.exe` totéž, ale na jiném počítači (to příkaz `start` neumí), název počítače se zadává bez úvodních opačných lomítek

`wmic os call reboot` restart systému

`wmic /node:`*`počítač`* `os call shutdown` vzdálené vypnutí systému

`wmic service "spooler" call startservice` spustí zadanou službu (Zařazování tisku)

`wmic /node:ucetni process where name="explorer.exe" call terminate`
zadaný proces bude ukončen, a to na uvedeném počítači (vypadá to, že účetnímu bude ukončeno grafické prostředí, ale tento proces se obvykle po ukončení znovu spustí, tedy jde vlastně o restart grafického prostředí)

`wmic /node:ucetni /user:dadmin process where name="explorer.exe" call terminate` totéž jako předchozí, ale v příkazu na zadaném počítači vystupujeme s přístupovými oprávněními zadaného uživatele

Vidíme, že můžeme používat i dotazy filtrované podle vlastností (where) podobně jako v SQL. Pokud se jedná o řetězcovou hodnotu, musíme ji uzavřít do uvozovek (například `name="explorer.exe"`, ale čísla nebo hodnoty `true`/`false` do uvozovek neuzavíráme.

Příklad

Ukážeme si práci v interaktivním módu.

`wmic` spustíme interaktivní režim, prompt je `wmic:root\cli>`

`os /?` dotážeme se, co lze použít na předmět `os` (operační systém)

`os list /?` chceme upřesnit parametry pro sloveso `list`

`os list brief` získáme tabulku s několika základními informacemi

`os list full` získáme seznam (už ne tabulku) s dvojicemi vlastnost=hodnota

`os list free` tabulka s hodnotami volného místa (ve fyzické paměti, v stránkovacích souborech, ve virtuální paměti)

`/output:D:\procinfo.txt cpu get` do souboru se uloží hodně široká tabulka vlastností procesoru

`/output:D:\procinfo.txt cpu list full` totéž, ale místo široké tabulky máme v souboru seznam položek vlastnost=hodnota (při takovém množství je to poněkud přehlednější)

`cpu get /?` zeptáme se, co se dá zjistit o procesoru

`cpu get NumberOfLogicalProcessors` získáme údaj o počtu logických procesorů (počet jader nebo jeho dvojnásobek, pokud procesor podporuje hyperthreading)

`cpu get AddressWidth,Caption,CurrentClockSpeed,DataWidth,Description,ExtClock` vypíše se tabulka s vybranými sloupci

`process get` vypíše se seznam běžících procesů, u každého je příkaz, kterým byl spuštěn (všimněte si, že v seznamu je i `wmiprvse.exe`, který zajišťuje vyhodnocování dotazů na WMI)

`process list brief` získáme tabulku procesů s některými informacemi

`process where ThreadCount>8 list brief` podobný výstup, ale vypíšou se pouze procesy, které mají více než 8 vláken

`service get name,state,serviceType` vypíše se tabulka s názvy, stavy a typy služeb

`service where desktopInteract=true get name,state` získáme seznam interaktivních služeb (všimněte si, že u neběžících služeb je PID=0)

`service where (desktopInteract=true and startMode="auto") get name,processid` výběrová kritéria můžeme kombinovat (jako v SQL), ale v tom případě je uzavřeme do závorky

`service where desktopinteract=true get name,processid,status /every:5`
takto zajistíme, že dotaz bude automaticky opakován v intervalu 5 sekund (stisknutím některé klávesy opakování přerušíme)

`quit` konec, ukončíme interaktivní režim, změní se prompt a přesuneme se do Příkazového řádku (funguje také příkaz `exit`)

Příklad

K WMI se dostaneme taky v Powershellu. Základním cmdletem je `get-WMIObject`, pro který existuje alias `gwmi`. Obvykle zadáváme typ objektu (parametr `-class`), případně vyfiltrujeme podle toho, co chceme. Například pokud budeme chtít vypsat údaje o BIOSu/UEFI, použijeme cmdlet:

```
gwmi -classs win32_bios
```

Hlavní problém je, že člověk těžko pojme všechny „třídy“, které existují. Naprosto všechny bychom si mohli vypsat cmdletem

```
gwmi -list
```

Nicméně to je velmi rozsáhlý výstup. Proto má smysl si ho profiltrovat, podle toho, co nás zajímá. Například pokud nás zajímají pouze třídy související s pamětí, napíšeme:

```
gwmi -list win32_*memory*
```

Takže pokud chceme vypsat parametry operační paměti, použijeme:

```
gwmi -class win32_physicalmemory
```

Úkoly

1. Vyzkoušejte práci v interaktivním módu příkazu podle příkladu na straně 99.
2. Najděte seznam všech předmětů (aliasů) pro příkaz `wmic`. V nápovědě v grafickém rozhraní jsou témata pro `wmic` značně rozházená, můžete použít například `wmic /?`.
3. Zjistěte, co vše lze zjistit v předmětech (aliasech) csproduct, datafile, memphysical, netprotocol, recoveros, registry, voltage.
4. Pomocí `wmic` zjistěte veškeré informace o řadičích síťového rozhraní (plný výpis), přesměrujte do souboru a ten potom prostudujte.
5. Vypište všechny běžící procesy, jejichž PID je větší než 1000.

Souhrn kapitoly 3

V této kapitole jsme si prohloubili své znalosti o procesech a službách. Naučili jsme se získávat informace o procesech a službách také v textovém režimu, ovlivňovat jejich spuštění a chod, zjistili jsme, jak mohou procesy mezi sebou komunikovat, naučili jsme se řešit problémy s kompatibilitou aplikací v různých verzích systému. Po sekcích o procesech a službách jsme se zaměřili na mechanismus WBEM a především jeho implementaci WMI a ukázali jsme si, jak tento mechanismus využívat při správě zařízení se systémem Windows, a to i v síti.

Kapitola 4

Správa zařízení a sítě ve Windows

Rychlý náhled: V této kapitole se budeme zabývat správou zařízení a počítačových sítí ve Windows. Nejdřív se zaměříme na ovladače, pak na paměťová média a operační paměť, a ke konci kapitoly se budeme věnovat postupům a příkazům pro správu sítě ve Windows.

Klíčová slova: Ovladač, certifikát WHQL, model ovladače, WDM, WDDM, DirectX, dxdiag, synchronizace, zálohování, disková kvóta, dirty bit, diskpart, virtuální disk, bod připojení, fsutil, hardlink, stream, ADS (alternativní datový stream), stránkovací soubor, soubor hosts, IP adresa, MAC adresa, směrovací tabulka, sdílené prostředky, NetShell.

Cíle studia: V této kapitole se naučíte ověřovat vlastnosti ovladačů, kontrolovat jejich digitální podpisy, porozumíte modelům ovladačů. Dále se naučíte používat příkazy pro synchronizaci a zálohování paměťových médií, pracovat s kvótami, kontrolovat stav disků, připojovat do systému oddíly, složky a sdílené složky na síti. Zjistíte, co to jsou alternativní datové streamy a jak se s nimi pracuje. Také se naučíte zjišťovat a konfigurovat parametry síťového připojení.

4.1 Ovladače

4.1.1 Co je to ovladač

Ovladač zařízení je rozhraní mezi zařízením a systémem, které zajišťuje komunikaci procesů se zařízením (například u tiskárny proces při požadavku na tisk předá hodnoty předem stanovených parametrů, jako je adresa dat, která se mají tisknout, formát tisku apod.). Obvykle jde o modul v jádře Windows, i když existují i ovladače běžící v uživatelském režimu.

Existují také jiné ovladače než ty, které souvisejí s konkrétními zařízeními – struktura ovladačů je ve Windows poměrně košatá, mnohé například filtrují, transformují či přesměrovávají data. Souborové systémy jsou také naprogramovány jako ovladače, například souborový systém NTFS je ovladač načítaný ze souboru `ntfs.sys`.

Poznámka

Určitě je každému jasné, že bezpečnostní programy (například antiviry) potřebují mít některou svou komponentu v jádře systému, protože jinak by byly snadno napadnutelné, a napadnutelný software

nemůže plnit bezpečnostní roli. Obvykle se to dělá tak, že součástí bezpečnostního programu je modul, který se při startu systému zavádí do jádra ve formě ovladače, přes který pak jdou veškeré požadavky, které je třeba filtrovat (od paměťových zařízení, ze sítě, přes IPC – komunikaci procesů, atd.).

Podobně jako pro služby existuje modul SCM (Service Control Manager), který zajišťuje spouštění a běh služeb a taky komunikaci s nimi, pro ovladače existuje *Driver Manager* s podobným účelem.

Informace o nainstalovaných ovladačích jsou (zároveň se službami) uloženy v registru ve větvi `HKLM\SYSTEM\CurrentControlSet\Services`, kde (jak víme) jsou jednotlivé podklíče pojmenovány zkrácenými názvy služeb a ovladačů.

K ovladači konkrétního zařízení se dostaneme ve *Správci zařízení* (spustíme ho příkazem `devmgmt.msc` nebo ho najdeme v konzole *Správa počítače*) – poklepeme na dané zařízení a přejdeme na záložku *Ovladač*. Tam taky můžeme ovladač odinstalovat či aktualizovat.

Odinstalace ovladače se může hodit ve dvou případech:

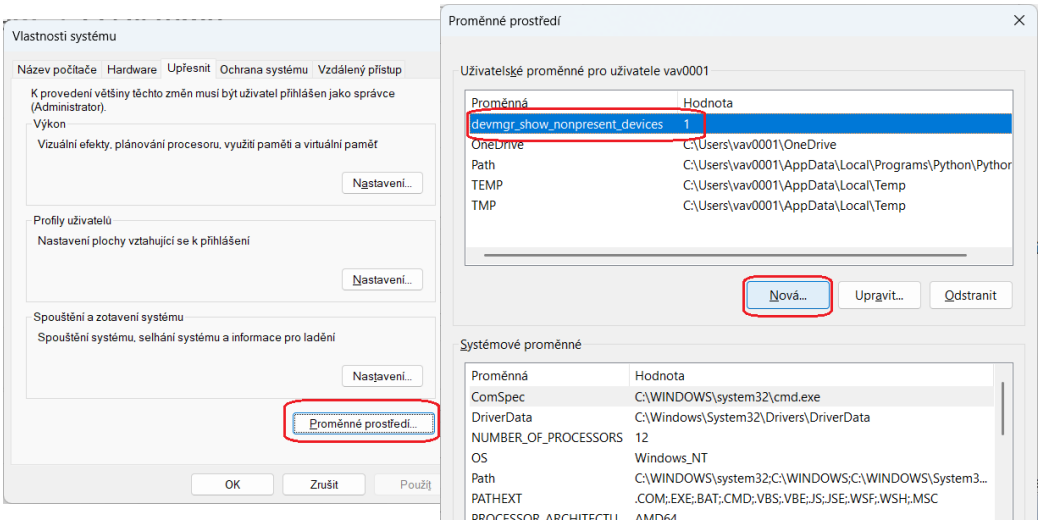
- pokud některá zařízení (například USB flash disky) připojujeme jenom jednorázově nebo jednou za dlouhý čas a takových zařízení je hodně, nabývá počet ovladačů, které se načítají při každém startu systému, což zbytečně start zpomaluje (a taky: čím víc ovladačů, tím víc problémů),
- pokud je zařízení špatně rozpoznáno nebo se chová chybně (například USB flash disk není rozpoznán jako paměťové médium a nepřipojí se pod písmenem).

Postup

V takových případech nechtěný či poškozený ovladač odinstalujeme, přičemž postupujeme takto:

- spustíme *Správce zařízení* a pokusíme se najít ovladač, který chceme odinstalovat,
- pokud to nejde, necháme zobrazit i skrytá zařízení (v menu *Zobrazit*),
- jestliže ani pak není k nalezení (což je typické třeba pro poškozené ovladače), přejdeme do nástroje *Systém* (třeba přes *Tento počítač* ⇨ *Vlastnosti*), *Upřesnit nastavení systému*, karta *Upřesnit*, tlačítko *Proměnné prostředí*, v uživatelských proměnných tlačítko *Nová*, vytvoříme proměnnou `devmgr_show_nonpresent_devices`, nastavit na 1 (jak je naznačeno na obrázku 4.1), potvrdíme a opět přejdeme do *Správce zařízení*,
- nalezený ovladač odstraníme (stačí klávesou Del, nebo v kontextovém menu či vlastnostech),
- ovladač znovu instalujeme – u jednodušších zařízení Plug&Play typu USB klávesnice nebo USB flash disku stačí toto zařízení prostě fyzicky připojit, u jiných (třeba tiskárna) budeme potřebovat instalační soubor ovladače.

Může se stát, že zařízení nepracuje správně. Na vině může být poškozený ovladač (to může nastat i hned po instalaci, nebo po změnách v konfiguraci systému) nebo ovladač nekompatibilní s momentální konfigurací systému (s některou částí systému si „nerozumí“, to může nastat při instalaci takových opravných balíčků, se kterými programátor ovladače nepočítal). Když je na vině ovladač, často je možné najít na internetu jeho novější verzi (na stránkách výrobce zařízení). Pokud jde o zařízení integrované na základní desce, bývá ovladač součástí většího balíku, který se dá stáhnout právě na webu výrobce základní desky, ale v některých speciálních případech i přesto může být lepší hledat přímo u výrobce daného zařízení, například kvůli verzím (výrobce základní desky někdy ve svém „balíku“ nemá nejnovější verze). Ale pozor, někdy je právě problém s nejnovější verzí.



Obrázek 4.1: Vynucení zobrazení všech ovladačů včetně poškozených

Úkol

Spusťte *Správce zařízení* a rozbalte větev *Diskové jednotky*. Nechte zobrazit skrytá zařízení a zjistěte, jestli v této větvi nejsou skryté položky. Pokud máte oprávnění, pročistěte seznam (nemažte nic, co je aktuálně používáno, včetně pevného disku).

4.1.2 Instalace ovladačů a certifikáty

Windows jdou dodávány s ovladači pro mnoho typů zařízení od různých výrobců, většinou tyto ovladače stačí alespoň pro základní používání zařízení a používají se při automatické detekci službou Plug&Play.

Pokud služba Plug&Play najde nové zařízení, především se pokouší najít pro zjištěné zařízení příslušný soubor INF v adresáři `Windows\Inf`. V něm jsou údaje potřebné pro instalaci a především cesta k souboru ovladače. Pokud INF soubor není nalezen, je uživatel požádán o jeho dodání.

Od Windows 98 Microsoft testuje ovladače a jejich použitelnost pod Windows. Pokud v laboratořích Microsoftu není nalezen žádný problém s kompatibilitou, přidělí ovladači *certifikát WHQL* (Windows Hardware Quality Lab) a ovladač je tzv. *digitálně podepsaný*. Testy se zaměřují pouze na kompatibilitu s Windows, nikoliv na kvalitu ovladače směrem k samotnému zařízení, tedy to, že je ovladač podepsaný, nic neříká o samotné kvalitě ovladače. V 64bitových Windows lze používat pouze podepsané ovladače.

Ne každý výrobce však „ztrácí čas“ čekáním, než Microsoft provede testy, a proto existují ovladače, které sice jsou kompatibilní s Windows, ale nejsou podepsané.

Někteří výrobci podepisují své ovladače vlastním certifikátem (důvěryhodným), který získali od společnosti Microsoft (tj. ne každý může své ovladače podepisovat – jen ten, komu je to „dovoleno“). Takto podepsané ovladače lze instalovat i v 64bitových verzích Windows.

Pokud chceme nalézt nepodepsané ovladače (někdy mohou způsobovat problémy, i když to není pravidlem), můžeme použít program *Ověření podpisu souboru* (SigVerif, spustíme ho přes *Start* ⇨ *Spustit*, napíšeme `sigverif`). O tomto nástroji najdeme podrobnější informace na straně 82.

Úkol

Propátrejte adresář `Windows\Inf`, najděte informační soubor základního ovladače pro monitor. Otevřete soubor a zjistěte, jak konkrétně se jmenuje soubor se samotným ovladačem (bude mít příponu .sys).

4.1.3 Druhy a modely ovladačů

Model ovladače souvisí především s tím, pro jaké API je ovladač programován, v jakém frameworku (běhovém prostředí) pracuje (tj. které knihovny vyžaduje a jak komunikuje), kde konkrétně ukládá své konfigurační údaje. V současné době se setkáváme s těmito modely:

- WDM (Windows Driver Model) – abstraktní model pro (všechny) ovladače používaný už od Windows 2000, popisuje architekturu ovladačů (co je to ovladač, jak a s čím komunikuje, API funkce, atd.),
- WDDM (Windows Display Driver Model) – abstraktní model pro přídavné multimediální ovladače (používaný například ke grafickým kartám, resp. grafickým čipům/jádrům), existuje od Visty.

Zatímco WDM zajišťuje funkcionalitu všech typů ovladačů, model WDDM přidává ovladačům multimediálních rozhraní další funkce především pro 3D zobrazování – využívání API Direct3D. WDDM umožňuje aplikacím s GUI plně využívat funkce kompozitního správce oken *Desktop Window Manager* (DWM) přenášejícího část výpočetní zátěže z procesoru na grafickou kartu (resp. grafické jádro, GPU).

Podle modelu WDM rozlišujeme dva druhy ovladačů podle toho, v jakém prostoru se nacházejí:

- ovladače běžící v režimu jádra (kernel drivers) – klasické ovladače ve formě modulů, které se buď při startu systému nebo později na vyžádání načítají do jádra, obvykle se načítají ze souborů s příponou .sys,
- ovladače běžící v uživatelském prostoru (user-mode drivers) – celý ovladač nebo jeho část pracuje v uživatelském prostoru, používají se už od Windows verze XP, obvykle bývají v dynamicky linkovaných knihovnách.

Ovladače běžící v uživatelském prostoru mají hlavní výhodu v tom, že když se „něco pokazí“, systém není zasažen (žádná modrá obrazovka). Tyto ovladače nemají přímý přístup k hardwaru (ani tomu svému), kumunikují s ním přes speciální API. Často jde právě o nejrůznější filtrační a transformační moduly, taky to, co je výpočetně náročnější (například ovladače podle modelu WDDM).

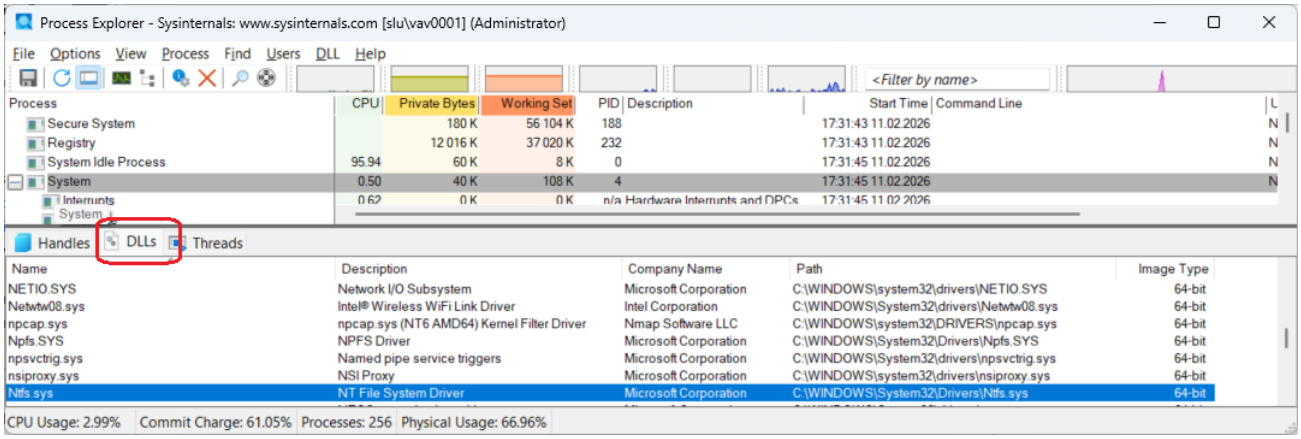
Každý ovladač potřebuje ke svému fungování určitou infrastrukturu (běhové prostředí) – *Windows Driver Frameworks* (WDF), přičemž existují dva používané frameworky:

- Kernel-Mode Driver Framework (KMDF) pro ovladače běžící v režimu jádra,
- User-Mode Driver Framework (UMDF) pro ovladače běžící v uživatelském režimu.

WDF je ve skutečnosti něco jako abstraktní vrstva nad oběma dílčími frameworky, která je důležitá zejména pro programátory ovladačů a hodně jim usnadňuje práci, včetně toho, že ovladače pro oba základní frameworky se mohou ve WDF programovat dost podobně. Framework totiž není jen běhové prostředí, je to i sada knihoven a šablon, které se používají při programování toho, co v tom prostředí má běžet.

Postup

Načtené (fungující) ovladače běží jako součást některého procesu, a právě v těchto procesech je najdeme jako moduly. Ovladače běžící v režimu jádra (tedy přes KMDF) jsou moduly procesu *System*, kdežto



Obrázek 4.2: Zobrazení modulů načtených procesem System v Process Exploreru

ovladače běžící v uživatelském režimu (přes UMDF) jsou moduly buď v některém hostitelském procesu služeb (`svchost.exe`) běžícím pod účtem Local Service, nebo mají vlastní proces běžící pod vlastním účtem (to je typické pro ovladače, které potřebují GUI).

Takže když si spustíme Process Explorer, zapneme zobrazování spodního podokna (*Show Lower Pane*) a v něm moduly (*Lower Pane View* ⇨ *DLLs*), stačí klepnout na příslušný proces (třeba System) a zjistíme, které moduly má proces načteny (mezi nimi mohou být i ovladače).

Poznámka

Jaký je rozdíl mezi UMDF ovladačem (tj. běžícím v uživatelském prostoru) a běžnou aplikací (která sice taky běží v uživatelském prostoru, ale není ovladačem)?

- jinak se spouštějí – UMDF ovladač se spouští přes modul *Driver Manager* a komunikují se svým okolím přes speciální API (zajišťované Driver Managerem),
- ovladače běží spíše podobně jako služba – většinou v hostitelském procesu služeb pod účtem Local Service, kdežto aplikace mají svůj vlastní proces a běží s oprávněními účtu uživatele (nicméně některé ovladače mohou mít taky vlastní proces),
- UMDF ovladače mohou být napojeny na mechanismus Plug&Play a správu napájení.

Jistě víte, že s kompatibilitou ovladačů bývají často větší problémy než s kompatibilitou aplikací. Například mechanismus UMDF existuje v různých verzích. Od Windows 8.1 se používají verze 2.x, kde se ovladače programují čistě v jazyce C a je snadnější je přepsat na ovladače typu KMDF (do jádra). Ovladače pro starší verzi UMDF 1.x se programovaly pro model COM, obvykle v C++. Rozdíly jsou jak v obsahu API funkcí, tak i v jejich názvech, takže přechod mezi verzemi znamená zásah do kódu.

Další informace

- https://learn.microsoft.com/en-us/windows-hardware/drivers/wdf/
- https://learn.microsoft.com/cs-cz/windows-hardware/drivers/wdf/overview-of-the-umdf
- https://learn.microsoft.com/cs-cz/windows-hardware/drivers/wdf/user-mode-driver-framework-frequently-asked-questions

Můžeme se setkat se staršími modely ovladačů. Před modelem WDM se používal model PMD (Protected-Mode Drivers, také VXD), ještě před ním model RMD (Real-Mode Drivers), jehož ovladače si ukládaly konfigurační údaje do souboru `config.sys`.

Úkol

Spusťte Process Explorer jako správce (pokud můžete) a projděte si moduly procesu System. Ve spodním podokně zvolte zobrazení modulů (DLLs) a případně si pohrajte s tím, které sloupce budou zobrazeny (pravé tlačítko na záhlaví tabulky s moduly, *Select Columns*). Především byste měli mít zobrazeny sloupce Name, Description, Path, Image Type. Vyberte si některý ovladač (třeba monitor.sys) a zjistěte, jaké informace se o něm zobrazují.

4.1.4 Informace o ovladačích

V předchozích sekcích bylo popsáno, jak získat informace o ovladači v grafickém prostředí: ve Správci zařízení a v Process Exploreru. Další možnosti nabízí textový režim.

Příklad

Jednou z možností je použití příkazu `driverquery`. V nápovědě se sice tvrdí, že je určen pro použití s administrátorskými oprávněními, ale ve skutečnosti se dá spustit i bez oprávnění správce. Základní možnosti jsou:

```
driverquery
driverquery /v
```

Přepínač `/v` znamená „verbous“ mód, tedy vypíše se o dost více informací. V nápovědě bychom zjistili, že existují i další možné parametry, včetně vzdáleného přístupu na jiný počítač v síti.

V PowerShellu máme příkaz `get-WindowsDriver`, který však opravdu vyžaduje oprávnění správce. Pokud takové oprávnění máme, seznam všech ovladačů získáme takto:

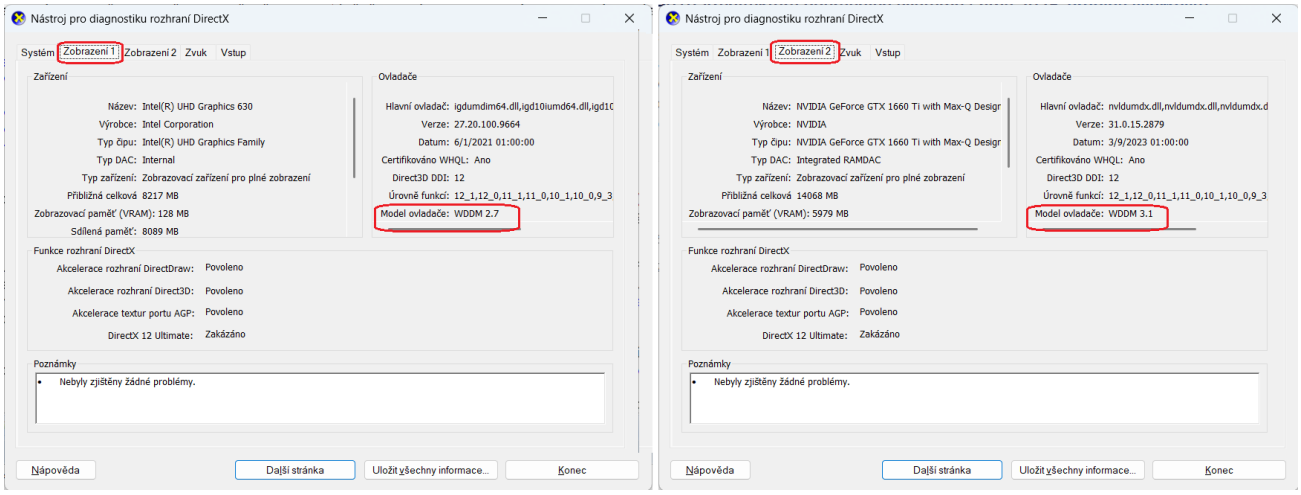
```
get-WindowsDriver -online -all
```

Výstup je velmi dlouhý, protože tam jsou opravdu všechny ovladače včetně virtuálních a různých filtrů v jádře. Položky jsou vypisovány ve formě seznamu, například pro základní diskový ovladač bychom dostali něco podobného:

```
Driver           : disk.inf
OriginalFileName : C:\Windows\System32\DriverStore\FileRepository\disk.inf_amd64_43f40a98bca24156\disk.inf
Inbox            : True
ClassName        : DiskDrive
BootCritical     : True
ProviderName     : Microsoft
Date             : 21.06.2006 0:00:00
Version          : 10.0.26100.7705
```

Multimédia se ve Windows řeší často přes API DirectX a jeho rozšíření (například Direct3D). Vlastnosti související s DirectX se u ovladačů také dají ověřovat.

Nástroj pro diagnostiku rozhraní DirectX (`dxdiag.exe`) slouží právě k tomuto účelu. Nejrychleji ho spustíme napsáním názvu programu `dxdiag` ve Startu.



Obrázek 4.3: Nástroj pro diagnostiku rozhraní DirectX na stroji se dvěma síťovými kartami

Nicméně samotný nástroj má grafické rozhraní. Po spuštění jsou nejdřív ověřovány ovladače (včetně digitálních podpisů) a následně se zobrazí okno nástroje se čtyřmi záložkami, z nichž dvě vidíme na obrázku 4.3. Na první záložce jsou informace o systému (název počítače, verze Windows a číslo sestavení (build), procesor, verze DirectX apod.

Druhá a třetí záložka jsou z celého nástroje nejdůležitější – najdeme zde informace o grafické a zvukové kartě. V případě grafické karty vidíme na obrázku 4.3 kromě parametrů samotného grafického čipu, že se používá model ovladače WDDM verze 1.2 (a taky jak se nazývají soubory ovladače; všimněte si, že nejde o soubory .SYS, ale o knihovny, takže zřejmě budou v uživatelském prostoru).

Na třetí záložce je informace o zvukové kartě nebo jejím integrovaném ekvivalentu. Jak vidíme, tento ovladač (který není až tak výpočetně náročný jako WDDM ovladače pro grafiku) je podle modelu WDM a jedná se o soubor `HdAudio.sys`.

Na čtvrté záložce jsou informace o dalších souvisejících zařízeních pro vstupy/výstupy, typicky o klávesnici a myši. Zjistili bychom zde například, jak se jmenují ovladače pro tato zařízení a související ovladače (například pro USB rozbočovače).

Tlačítko *Uložit všechny informace* slouží k exportu do textového souboru, ve kterém pak budeme mít to, co se nám zobrazilo na jednotlivých záložkách.

Úkol

Spusťte *Nástroj pro diagnostiku rozhraní DirectX* a projděte si jeho jednotlivé záložky. Exportujte výstup do textového souboru a tento soubor si prohlédněte. Berte v úvahu, že okno nástroje nelze libovolně rozšiřovat, tedy k části informací se vlastně přímo v okně nemusíme dostat, ale pro export to neplatí – tam máme všechno.

4.2 Paměťová média

4.2.1 Synchronizace a zálohování

Poměrně častým úkonem při práci s daty je synchronizace a zálohování dat. Jde o to, abychom s minimálním úsilím zajistili zjištění rozdílů mezi původním a novým umístěním a abychom měli na více

místech aktuální synchronizovaná data. Příkaz `copy` není pro tento účel optimální – chybí nám u něj možnost podrobněji konfigurovat, co a jak se má synchronizovat.

Pro účely synchronizace máme ve Windows dva příkazy – `XCOPY` a `ROBOCOPY`. První z nich má výhodu rozšířenosti (je na prakticky všech verzích Windows, se kterými se můžeme setkat, takže je použitelný ve skriptu, který musí fungovat opravdu všude), druhý zase má víc pokročilejších funkcí.

Nejdřív se podíváme na příkaz `XCOPY`. Podrobnou syntaxi bychom získali v nápovědě (`xcopy /?`), takže jen pár ukázek použití:

Příklad

Ukážeme si, jak se používá příkaz `XCOPY`.

`xcopy d:\dopisy g:\ /s` rekurzívně (parametr `/s`) zkopíruje celý adresář z prvního parametru do adresáře v druhém parametru (zde na disk G:, to může být třeba USB flash disk); pokud je některý podadresář ve struktuře prázdný, nezkopíruje ho

`xcopy d:\dopisy g:\ /s /e` totéž, ale navíc kopíruje i prázdné adresáře (empty)

`xcopy d:\dopisy g:\ /s /m` kopíruje pouze soubory s nastaveným atributem *archivovat*, tento atribut po kopírování na zdroji vždy odstraní (je pak nastaven při následovné změně souboru) – při opakovaném používání příkazu zálohujeme pouze ty soubory ze zdrojového umístění, které se od posledního zálohování změnily

`xcopy d:\dopisy g:\ /s /d` porovnává datum poslední změny každého souboru ve zdrojovém a cílovém adresáři (také rekurzívně v podadresářích) a kopíruje jen ty soubory, které byly změněny od posledního zálohování

`xcopy d:\dopisy g:\ /s /d:12-25-16` budou kopírovány pouze soubory, které byly změněny po zadaném datu (25. 12. 2016, formát je měsíc–den–rok), a to rekurzívně i v podadresářích

`xcopy d:\dopisy g:\ /s /L > seznam.txt` parametr `/L` znamená, že ve skutečnosti na kopírování nedojde, jen bude vypsán seznam souborů, které by byly kopírovány (zde je přesměrován do souboru) – simulujeme zálohování, abychom měli představu, kterých souborů se kopírování bude týkat

`xcopy d:\dopisy g:\ /s /h /k` kopíruje také skryté a systémové soubory (parametr `/h`) a navíc zachová atributy (například jen pro čtení, systémový, apod., parametr `/k`)

`xcopy d:\dopisy g:\ /s /u` provede pouze aktualizaci (update), kopíruje ze zdroje jen ty soubory, které v cíli již existují

`xcopy d:\dopisy \\pocitac\evidence /s /z` parametr určený pro kopírování v síti; pokud dojde k přerušení kopírování, dokáže po obnovení spojení navázat a pokračovat v kopírování (také ukazuje procento průběhu kopírování)

Teď se zaměříme na příkaz `ROBOCOPY` („robustní“ kopírování). Je přítomen ve Windows od verze Vista, na starších systémech nebude fungovat. Byl navržen přímo za účelem zálohování a rozsáhlých přesunů datových struktur.

Jak bylo výše uvedeno, tento příkaz má víc synchronizačních funkcí a je to poznat i na struktuře nápovědy. Jednotlivé přepínače jsou rozděleny podle typu (volby pro kopírování, volby pro určení souborů, se kterými se má pracovat, volby pro opakování, volby pro logování průběhu, volby pro hromadné zpracování v rámci úlohy). Podrobnou syntaxi opět získáme v nápovědě (`robocopy /?`). Základní syntaxe je

`ROBOCOPY` *`zdroj cíl filtr_souborů parametry`*

Příklad

Ukážeme si, jak se používá příkaz `ROBOCOPY`.

`robocopy D:\mojedata G:\zaloha` jednoduše zálohuje všechny soubory z prvního umístění do druhého umístění (bez podadresářů), určení souborů je výchozí (`*.*`, tedy všechny)

`robocopy D:\mojedata \\server\D$\zaloha` podobně, zálohujeme na server, jehož adresu jsme zadali ve formě UNC, všimněte si, jakým způsobem jsme zadali adresář na disku `D:` na serveru (vzpomínáte na skrývání názvů sdílených položek?)

`robocopy D:\mojedata G:\zaloha *.xlsx` pouze excelovské soubory (se zadanou příponou)

`robocopy D:\mojedata G:\zaloha /e /np` první parametr určuje, že se má kopírovat i obsah podadresářů, včetně prázdných, druhý parametr způsobí, že se během kopírování nezobrazuje průběh, což šetří čas

`robocopy D:\mojedata G:\zaloha /s /np /mir` první parametr znamená, že budou kopírovány i podadresáře, ale narozdíl od `/e` se případný prázdný adresář ze zdroje v cíli neobjeví, druhý parametr již známe, třetí určuje, že se má provést zrcadlení (mirroring) – celá struktura ze zdroje se zohlední v cíli (berou se v úvahu změny, souborů beze změn si příkaz nebude všímat), a pokud byl od poslední zálohy některý prvek ve zdroji smazán, bude smazán i v cíli (pozor, to může být za určitých okolností nebezpečené)

`robocopy D:\mojedata G:\zaloha /s /np /mir /log:zaloha.log` totéž jako předchozí, navíc je proces zálohování logován do zadaného souboru (taktéž – výstup příkazu nebude vypisován na obrazovku, ale pouze do log souboru)

`robocopy D:\mojedata G:\test /s /create` jakási poloviční simulace kopírování – v cílovém adresáři pouze vytvoří adresářovou strukturu stejnou jako je ve zdroji, soubory v cíli sice vytvoří, ale s nulovou délkou (tj. vpodstatě zkopíruje pouze obsah adresářů, obsah souborů nikoliv)

Veškeré možnosti `robocopy` zde nebudeme probírat – je možné například pracovat s atributy souborů a adresářů (včetně atributu archivace), vylučovat určité soubory a adresáře z kopírování, filtrovat podle data změny souborů a pracovat s časovými razítky, odložit kopírování na restart systému (aby bylo možné kopírovat i ty soubory, které jsou obvykle uzamknuté systémem), přesouvat místo kopírování, optimalizovat kopírování po síti pro nižší propustnost, atd.

Další informace

- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy
- https://www.sevenforums.com/tutorials/187346-robocopy-create-backup-script.html
- https://ss64.com/nt/robocopy.html

Úkoly

1. Zjistěte v nápovědě příkazu `robocopy`, jak lze zálohovat celý oddíl disku `C:` na server tak, aby se zálohování netýkalo souboru `pagefile.sys` ani adresáře `"System Volume Information"`, aby se zálohovala–zrcadlila celá adresářová struktura a logovalo se do souboru `vysledek.log`.
   Poznámka: v reálu by vyloučených souborů a adresářů bylo samozřejmě více.

2. Sestavte příkaz, kterým zálohujete složku `Dokumenty` ze svého profilu (celou včetně obsahu – rekurzívně) na disk `F:` (to může být třeba USB flash disk), a to tak, že jsou zálohovány pouze soubory s nastaveným atributem pro archivaci (během zálohování tento atribut odstraňujte). Použijte jeden z výše uvedených příkazů.
3. Sestavte příkaz, kterým zálohujete celou složku se všemi profily (rekurzívně) se stejným chováním k atributu pro archivaci jako v předchozím úkolu, ale cílem bude adresář nazvaný podle názvu tohoto počítače na serveru `\\fileserver`, a to tak, aby příkaz dokázal v případě poruchy sítě po obnovení spojení navázat na předchozí zálohování.
4. Pro příkaz `robocopy` existuje grafické uživatelské rozhraní (GUI) přímo od Microsoftu. Pokuste se o tomto GUI zjistit více.

4.2.2 Diskové kvóty

Diskové kvóty určují, kolik místa na disku (obvykle síťovém) může který uživatel zabrat, znemožňují zabrat více než je stanovené množství nebo dovolují protokolovat překročení stanoveného místa. Je sledováno množství místa zabraného soubory, jejichž vlastníkem je daný uživatel (místo zabrané uživatelem, který patří mezi správce systému, není omezováno, protože administrátoři je vlastníkem všech systémových souborů, kterých není zrovna málo).

Rozlišujeme dva pojmy:

- *Maximální disková kvóta* – množství místa, které bude mít uživatel k dispozici.
- *Úroveň pro upozornění* – hodnota o něco nižší než předchozí, slouží k upozornění, že se uživatel blíží horní povolené hranici.

Obě hranice (maximální hranici a hranici pro upozornění) lze protokolovat, při protokolování můžeme nastavit příslušnou reakci (například upozornit uživatele nebo upozornit správce).

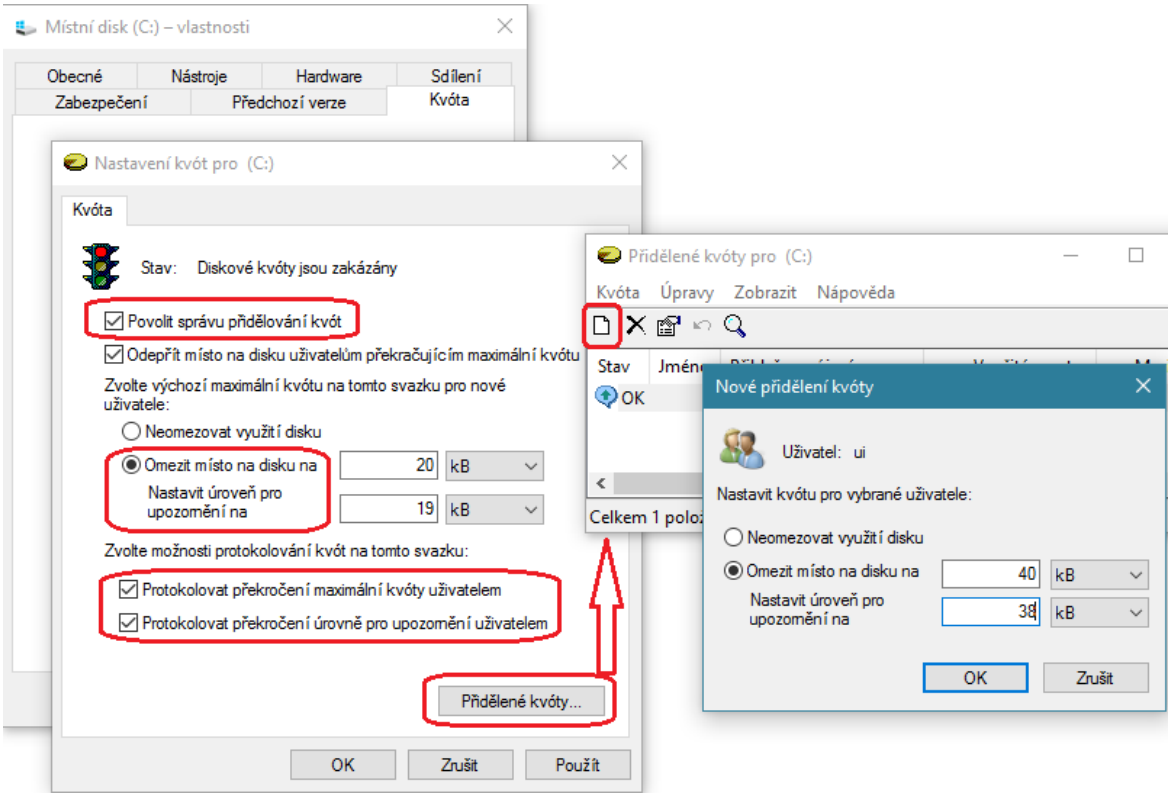
K protokolům se dostaneme například v nástroji *Prohlížeč událostí*, a to v *Systémovém protokolu*.

Obecně je možné přidělovat diskové kvóty na discích (složkách), které jsou sdílené a naformátované souborovým systémem NTFS. Možnost používání kvót je jednou z vlastností tohoto souborového systému; pokud máme disk naformátován souborovým systémem FAT32 nebo starším, nemůžeme na něm kvóty používat.

Kvóty jsou také záležitostí lokální sítě. Můžeme přidělovat kvóty na kterémkoliv počítači v síti, ke kterému má daný uživatel jakýkoliv typ přístupu. To ovšem vyžaduje, aby byli uživatelé v záznamech kvót identifikováni jednoznačně v rámci celé lokální sítě, tedy se používají SID uživatelů místo jejich jmen.

Na lokálním počítači se v grafickém režimu dostaneme ke kvótám takto: zobrazíme vlastnosti disku (např. v Průzkumníkovi v kontextovém menu jednotky oddílu na disku), a pokud je typu NTFS, je zde záložka *Přidělená kvóta* (musíme mít dostatečné oprávnění k přístupu, aby se tato záložka zobrazila). Povolíme přidělování kvót (na obrázku 4.4 vlevo) a dále je přidělíme jednotlivým uživatelům (k podrobnějšímu nastavení pro jednotlivé uživatele se dostaneme klepnutím na tlačítko *Přidělené kvóty*).

Pokud chceme pracovat s kvótami nastavenými na jiném počítači, než u kterého sedíme, můžeme jednoduše připojit disk z toho počítače jako síťový disk (přidělíme písmeno jednotky) a stejným způsobem jako u lokálního disku zobrazíme jeho vlastnosti.



Obrázek 4.4: Nastavení kvót

Pro práci s kvótami existuje také příkaz `fsutil quota`, se kterým se seznámíme dále.

Úkol

Zjistěte, zda jsou na discích vašeho počítače přiděleny kvóty (pokud máte dostatečná přístupová oprávnění).

4.2.3 Kontrola stavu disků

Na kontrole stavu disků spolupracuje několik programů, z nichž každý má svůj speciální účel.

`CHKDSK`

(v adresáři `system32`, název je zkratkou z „Check Disk“) při použití bez parametrů provede rychlou (asi tak minutovou) kontrolu a vypíše informaci o stavu systému, pomocí přepínačů můžeme spustit podrobnou kontrolu s možností oprav (ta je však prováděna při následujícím startu systému programem `autochk.exe`), data zachráněná z poškozených sektorů najdeme v souborech s příponou CHK.[1]

`chkdsk` provede rychlou kontrolu a vypíše informace o disku

`chkdsk d:` provede rychlou kontrolu zadané jednotky a vypíše o ní informace

`chkdsk d: /F` důkladná softwarová kontrola, u zjištěných chyb se pokouší provést opravu (fix errors), oddíl musí být odpojen (jen softwarově!)

[1] Někdy bývá trochu problém s obnovou dat z těchto souborů, už proto, že jde vlastně o fragmenty původních souborů z disku. Existuje několik možností, jak se s tímto problémem vypořádat, zajímavá stránka o tomto tématu je například http://www.ericphelps.com/uncheck/.

`chkdsk d: /R` důkladná hardwarová kontrola (zahrnuje v sobě i přepínač `/F`), hledá chybné sektory a pokouší se z nich obnovit data

Existují dvě možnosti, jak zajistit, aby oddíl na disku (svazek nebo celý disk) byl důkladně zkontrolován po startu systému:

1. *dirty bit* – pokud kterýkoliv program s vyššími oprávněními zjistí problémy s oddílem na disku (při čtení, zápisu nebo jakékoliv jiné manipulaci s oddílem), je tento oddíl označen jako „špinavý“ (má nastaven dirty bit),
2. *položka* `BootExecute` v registru v klíči *`HKLM/SYSTEM/CurrentControlSet/Control/Session Manager`*, obvykle za běhu systému obsahuje řetězec ve tvaru „`autocheck autochk *`“; tato položka určuje oddíly, které mají být zkontrolovány při startu systému, i kdyby neměly nastaven dirty bit. Uvedený řetězec vlastně znamená „všechno“ a takto je zajištěno, že při nesprávném vypnutí počítače budou při následném startu zkontrolovány všechny jednotky, tento řetězec je změněn při každém správném vypnutí počítače a opětovně nastaven při startu (takže kontrola je vyvolána, jen když nesprávně vypneme počítač).

S dirty bitem pracují další nástroje:

`CHKNTFS` (v adresáři `system32`)
slouží k řízení naplánování spuštění kontroly disků těsně před startem systému. Název příkazu je sice zkratkou z anglického „Check NTFS“, ale ve skutečnosti lze takto naplánovat také kontrolu jiných souborových systémů než NTFS (pracuje také s FAT a FAT32).

Tento příkaz ve skutečnosti nedokáže měnit samotný dirty bit (ten nuluje pouze program `chkdsk`), ale dokáže zajistit, aby kontrola nebyla vyžadována při následujících startech systému. Také dokáže naopak vynutit kontrolu u oddílu, který nemá nastavený dirty bit.

`chkntfs d:` zobrazí informaci o tom, zda je či není zadaný oddíl na disku označen jako „špinavý“ (a také informaci o typu souborového systému na oddílu)

`chkntfs /x e: f:` zadané disky nebudou kontrolovány při startu systému, i kdyby byly označeny jako „špinavé“ (jsou vyloučeny z kontroly – excluded)

`chkntfs /c d:` zadaný disk bude zkontrolován po startu počítače, i když není označen jako „špinavý“.

Nastavení dirty bitu lze zjistit ještě jiným způsobem – příkazem `fsutil dirty query` (bude probíráno později).

Program `chkdsk` není moc silný, má problém s opravou poškozeného sektoru a případným zachráněním dat, ale také se mu některé chyby nedaří detekovat. V horších případech se doporučuje použít jiný nástroj – buď komerční, anebo některý z volně šiřitelných, které jsou také velmi kvalitní.

Pokud je poškozena systémová část disku, volíme nástroje, které jsou bootovatelné (obvykle jde o upravený Linux, WinPE nebo FreeDOS).

Úkoly

1. Prověřte, zda některý oddíl na disku nemá nastaven dirty bit.
2. Podívejte se do registru, jak je nastavena výše popisovaná položka *`BootExecute`*.
3. Spusťte zběžnou (rychlou) kontrolu sektorů na disku (bez nutnosti restartu počítače).
4. Zjistěte, k čemu slouží a jak se používá program `diskperf.exe`.

4.2.4 Oddíly na disku

Ve Windows s NT jádrem používáme pro práci s oddíly na disku grafickou konzolu *Správa disků* (`diskmgmt.msc`) lokálně nebo ke správě vzdáleného počítače, kterou však nelze ovládat programově (ve skriptu). Můžeme zde vytvořit či zrušit oddíl a taky formátovat (vytvořit souborový systém na oddílu), vše přes kontextové menu příslušného objektu. Nicméně – někdy se hodí nástroje z Příkazového řádku.

Pokud chceme oddíl na disku jen formátovat, můžeme použít program `FORMAT`.

`format X:` zformátuje zadaný disk `X:` (připraví k použití, vytvoří souborový systém, tedy umožní na disk ukládat soubory), obsah disku samozřejmě vymaže; výchozí nastavení (chování) tohoto příkazu je odlišné v systémech s DOS a NT jádrem

`format X: /U` (Windows s DOS jádrem; v NT je tato volba výchozí) hluboké formátování; u běžného formátování je pouze přepsána tabulka obsazení disku, tedy soubory je možné ještě obnovit, u hlubokého formátování je celý disk přepsán symboly s ASCII kódem #0

`format X: /Q` (Windows s NT jádrem, v DOS je výchozí) rychlé formátování, nemažou se data

`format X: /FS:`*`souborový_systém`* (Windows s NT jádrem) formátuje na zadaný souborový systém, můžeme použít `FAT`, `FAT32` nebo `NTFS`

`DISKPART` je program pro práci s diskovými oddíly ve Windows od verze Vista a Server 2008 (lze jej stáhnout z webu Microsoftu i pro starší systémy), je také součástí *Konzoly pro zotavení* a Windows PE. Ve skutečnosti se jedná o interaktivní textovou konzolu a lze ho používat jak v interaktivním, tak i neinteraktivním módu.

Tento příkaz umožňuje na velmi pokročilé úrovni pracovat s disky a jejich jednotlivými oddíly včetně dynamických svazků a dynamických oddílů. Typické úlohy jsou zjistění informací o disku či oddílu, vytvoření či zrušení oddílu, vytvoření nebo rozšíření dynamického svazku, práce s RAID (zrcadlení), připojení nebo odpojení oddílu.

Některé příkazy jsou určeny k práci s konkrétním diskem nebo oddílem. Před použitím takovýchto příkazů musíme nejdřív daný disk či oddíl vybrat příkazem `select`, vybraný disk (oddíl) se nazývá *zaměřený* (focused). Neplést si s aktivním diskem, to je něco jiného!

Příklad

V tomto souhrnném příkladu si ukážeme základní práci s programem `diskpart`.

`diskpart` spustili jsme program, jsme v interaktivním režimu, dále zadáváme příkazy určené tomuto programu (prompt je teď ve tvaru `diskpart>`)

`list disk` vypíšeme seznam disků s informacemi (pracuje také s externími disky včetně USB flash disků), v seznamu má každý disk své číslo (od 0)

`select disk 1` vybrali jsme disk číslo 1 (tj. druhý v pořadí), tento disk je teď *zaměřený*

`list partition` vypíšeme seznam oddílů na zaměřeném disku

`select partition 3` zaměříme vybraný oddíl (č. 3)

`delete partition` odstraníme zaměřený oddíl (pozor, opravdu se odstraní!), při práci s dynamickými svazky tento příkaz nepoužíváme (místo něj používáme `delete volume`)

`create partition primary size=18000` vytvoří nový *primární* oddíl na disku se zaměřením o velikosti 18 000 MB, na disku musí být dostatek nepřiřazeného místa (nepřiřadí žádné písmeno jednotky), jiné příkazy jsou pro vytvoření rozšířeného (extended) a logického (logical) oddílu (budeme se učit na přednáškách)

`assign letter=e` přiřadí vytvořenému oddílu písmeno jednotky (bude dostupný pod písmenem E:), tímto příkazem (`assign mount=...`) můžeme vytvořený oddíl připojit také do adresáře (bod připojení)

`detail disk` zobrazí detailní informaci a disku se zaměřením

`exit` odchod z prostředí programu

Další informace

Popis parametrů příkazu najdeme na http://support.microsoft.com/kb/300415 (a také v nápovědě).

Pokud nepracujeme v interaktivním režimu, používáme trochu jinou syntaxi. Interní příkazy zadáváme pomocí parametrů programu, například

`diskpart /add \Device\HardDisk0 15000` na pevném disku, který je první v pořadí, vytvoří nový oddíl o velikosti 15 000 MB (na disku musí být tolik volného místa nepřiřazeného žádnému oddílu)

`diskpart /delete \Device\HardDisk0\Partition1` na disku 0 odstraní první oddíl (pozor, oddíly, které běžně vidíme, jsou číslovány až od 1)

`diskpart /delete F:` odstraní oddíl, který má přiřazeno písmeno F

Úkol

Zjistěte, do jaké míry program `diskpart` umožňuje pracovat s dynamickými svazky (používá se pojem volume). Dále zjistěte, jakým způsobem lze pomocí tohoto nástroje vytvořit RAID (diskpart umožňuje pracovat se zrcadlením disků).

4.2.5 Virtuální disky

Pokud je cesta do adresáře příliš dlouhá, můžeme si ji jednoduše zkrátit – podobně jako oddílu na disku lze písmeno přidělit také adresáři.

Příkaz `SUBST` vytvoří virtuální disk s označením `X:` směrovaný na zadaný adresář, označení disku volíme tak, aby nekolidovalo s existujícími disky v systému.

Používáme například tehdy, když chceme tentýž adresář (s dlouhou specifikací) používat často, ale nechce se nám jeho celý název vypisovat. Použitelné také ve Windows, disk s tímto označením se objeví ve všech běžných správcích souborů.

Příklad

Následující dva příkazy můžeme použít přímo za sebou (žádný disk s přiřazeným písmenem `G:` do této chvíle ještě neexistoval).

`subst G: Z:\adr1\adr2\adr3` vytvoří virtuální disk s písmenem G:, který ukazuje na zadaný adresář

`copy soubor.xxx G:` do adresáře kopírujeme soubor, můžeme používat jakkoliv běžným způsobem

`subst G: /d` zruší přiřazení virtuálního disku, uvolní písmeno, které jsme pro něj použili (zde `G`)

Úkoly

1. Zjistěte, které písmeno pro jednotky je ještě volné (zvolte některé spíše několik písmen před koncem abecedy). To lze například v některém souborovém manažerovi či Průzkumníkovi, v úvahu obvykle připadají písmena W, T, Y nebo podobná.

   Toto písmeno přiřaďte virtuálnímu disku, který vytvoříte jako přístupovou jednotku pro adresář `C:\Windows\system32`.
2. Přiřazení písmene virtuálnímu disku vytvořené v předchozím úkolu využijte při kopírování souboru `charmap.exe`, který je v uvedeném adresáři (a tedy i na virtuálním disku), na disk `D:`.
3. Zrušte přiřazení písmene virtuálního disku, které jste vytvořili v předchozích úkolech, a soubor, který jste nakopírovali na disk `D:`, smažte.

4.2.6 Body připojení

Ve Windows (v souborovém systému NTFS) lze připojit oddíly buď pod písmenem (jak je většina uživatelů zvyklá), nebo do adresáře (jako v UNIXových systémech), obecně prostě do *bodu připojení* (přípojného bodu). Pro práci s body připojení (ať už „písmenkovými“ nebo adresáři) slouží příkaz `MOUNTVOL` (podle „mount volume“).

Příklad

Můžeme vytvořit nebo zrušit přípojný bod, anebo jen zobrazit diskové oddíly, které lze připojit.

`mountvol` vypíše krátkou nápovědu a seznam diskových oddílů, které jsou k dispozici (ať už připojené nebo nepřipojené), u připojených vidíme také přidělené písmeno jednotky nebo adresář bodu připojení

`mountvol c:\disky\prvni /L` zjistíme, co konkrétně je v zadaném adresáři připojeno (pokud se jedná o bod připojení)

`mountvol c: /L` zjistíme, kterému oddílu je zadané písmeno jednotky přiřazeno (je to vpodstatě varianta předchozího příkazu)

`mountvol c:\disky\prvni \\označenísvazku` připojíme svazek (jeho označení zjistíme předchozími vypisovacími příkazy) do adresáře `prvni` (tento adresář musí už existovat), v tomto adresáři pak najdeme v adresářové struktuře obsah připojeného svazku

`mountvol c:\disky\prvni /D` zrušíme přípojný bod (odpojíme)

Diskové oddíly, které lze připojit, jsou označeny takto:
`\\?\Volume{označení GUID}\`
kde označení GUID (Global Unique Identifier) je jednoznačné určení zařízení nebo jeho části (zde oddílu na disku), například
`\\?\Volume{865f1ff0-a863-11d9-9f02-806d6172696f}\`

Poznámka

Pozor, `mountvol` pracuje na nižší úrovni (na úrovni jádra) než `subst` pro virtuální disky. Důsledkem je, že potřebujeme vyšší přístupová oprávnění, označení disků není zrovna intuitivní a také `mountvol` „nevidí“ virtuální disky vytvořené příkazem `subst`.

Úkoly

1. Zjistěte, které oddíly disku jsou připojeny a pod jakými písmeny. Existují nějaké nepřipojené oddíly?
2. Vyberte si jakýkoliv oddíl z výpisu předchozího příkladu a zjistěte, kde v registru se jeho GUID nachází.

4.2.7 Program `fsutil`

Program `FSUTIL` slouží ke správě diskových oddílů se souborovým systémem NTFS (ale rozumí také FAT a FAT32), na rozdíl od příkazu `diskpart` se nezajímá ani tak o samotný oddíl, ale spíše o souborový systém na něm. Podobně jako u jiných silných příkazů je třeba mít administrátorská přístupová oprávnění (alespoň k provádění změn).

Syntaxe je podobná příkazu `net` – za názvem `fsutil` následuje upřesňující příkaz.

`fsutil fsinfo`
vypíše informace o diskových jednotkách a souborovém systému, například

`fsutil fsinfo drives` vypíše seznam připojených jednotek (včetně externích)

`fsutil fsinfo drivetype d:` chceme zjistit typ jednotky d: (například pevný disk, výměnné médium, síťový disk, CD mechanika, apod.)

`fsinfo volumeinfo d:` k zadané jednotce vypíše o něco podrobnější informaci (název a sériové číslo, souborový systém, atd.)

`fsutil fsinfo ntfsinfo d:` pokud je na jednotce souborový systém NTFS, vypíšou se podrobné informace týkající se nastavení tohoto souborového systému (počet sektorů, počet clusterů, adresy některých metadat, atd.)

`fsutil fsinfo statistics d:` vypíše statistické informace o jednotce (předpokládá se souborový sytém NTFS) jako je počet čtení, počet zápisů apod.

Příklad

Zjistíme údaje o souborovém systému na oddílu C: pevného disku.

```
C:\> fsutil fsinfo ntfsinfo c:

NTFS Volume Serial Number :        0x081490701490630c
NTFS Version      :                3.1
LFS Version       :                2.0
Total Sectors     :                1 951 262 024  (930,4 GB)
Total Clusters    :                  243 907 753  (930,4 GB)
Free Clusters     :                  178 667 972  (681,6 GB)
Total Reserved Clusters :                7 095  ( 27,7 MB)
Reserved For Storage Reserve :               0  (  0,0 KB)
Bytes Per Sector  :                512
Bytes Per Physical Sector :        512
Bytes Per Cluster :                4096  (4 KB)
Bytes Per FileRecord Segment    :  1024
Clusters Per FileRecord Segment :  0
Mft Valid Data Length :            1,94 GB
Mft Start Lcn  :                   0x00000000000c0000
Mft2 Start Lcn :                   0x0000000000000002
Mft Zone Start :                   0x000000000338b640
Mft Zone End   :                   0x0000000003397e60
```

```
MFT Zone Size  :                  200,13 MB
Max Device Trim Extent Count :    256
Max Device Trim Byte Count :      0xffffffff
Max Volume Trim Extent Count :    62
Max Volume Trim Byte Count :      0x40000000
Resource Manager Identifier :     872EDD36-EA0B-11EF-B199-E44146BA15E6
```

Z údajů můžeme vyčíst, jak velký je oddíl (můžeme například vynásobit počet sektorů velikostí sektoru, pozor, hexadecimální číslo je třeba nejdřív převést na dekadické), sektory mají velikost 512 B (půl KiB) jak je dnes pořád ještě obvyklé (některé nejnovější disky 4 KiB), jeden cluster zabírá 8 sektorů, atd.

Předchozí výpis se vztahuje k SSD. Klasický HDD nepotřebuje funkci TRIM, tedy například hodnoty `Max Device Trim Extent Count` a `Max Device Trim Byte Count` by byly 0.

Podobně zjistíme statistiku o daném oddílu disku:

```
C:\> fsutil fsinfo statistics c:

File System Type :     NTFS

UserFileReads :        3375303
UserFileReadBytes :    27031251456
UserDiskReads :        3288696
UserFileWrites :       2821934
UserFileWriteBytes :   24083798528
UserDiskWrites :       3046392
MetaDataReads :        514620
MetaDataReadBytes :    2147512320
MetaDataDiskReads :    726394
MetaDataWrites :       1174863
MetaDataWriteBytes :   6868901888
MetaDataDiskWrites :   1624811

MftReads :                     450846
MftReadBytes :                 1846693888
MftWrites :                    975888
MftWriteBytes :                5392273408
Mft2Writes :                   0
Mft2WriteBytes :               0
RootIndexReads :               0
RootIndexReadBytes :           0
RootIndexWrites :              0
RootIndexWriteBytes :          0
BitmapReads :                  13740
BitmapReadBytes :              86646784
BitmapWrites :                 159742
BitmapWriteBytes :             1111298048
MftBitmapReads :               28
MftBitmapReadBytes :           552960
MftBitmapWrites :              19630
MftBitmapWriteBytes :          105222144
UserIndexReads :               261827
UserIndexReadBytes :           1172381696
UserIndexWrites :              445962
UserIndexWriteBytes :          2470039552
LogFileReads :                 8
LogFileReadBytes :             32768
LogFileWrites :                401604
LogFileWriteBytes :            8656019456
LogFileFull :                  128
DiskResourceFailure :          0
```

Zde už nezáleží na tom, jestli jde o SSD nebo HDD, jde o přístupové statistiky.

`fsutil file`

práce s konkrétními soubory (vytváření, vyhledávání, atd.)

`fsutil file createnew d:\soubor.txt 10000` vytvoří nový soubor se zadaným názvem o délce 10 000 B (necelých 10 KB)

`fsutil file findbysid novak d:\pisemnosti` bude hledat soubor, jehož vlastníkem je `novak`, v adresáři `d:\pisemnosti` (funguje, pokud máme nastaveny diskové kvóty pro uživatele)

Tento příkaz také dovoluje pracovat také s tzv. „řídkými soubory“.

`fsutil volume`

základní správa svazku – možnost zjištění volného místa a možnost jeho odpojení

`fsutil volume diskfree d:` zjistí, kolik volného místa je na disku d: (kolik lze ještě použít pro soubory

`fsutil volume dismount d:` odpojí zadanou diskovou jednotku

`fsutil behavior`

zobrazí nebo změní chování souborového systému (NTFS)

`fsutil behavior query disablelastaccess` zjistíme, zda je povoleno nebo zakázáno časové razítko při každém přístupu (zda se položka času posledního přístupu mění při každém přístupu k souboru včetně přístupu pouze pro čtení)

- pokud tento příznak není nastaven, zobrazí se hlášení
  `disablelastaccess není aktuálne nastaven`
- pokud naopak nastaven je, zobrazí se
  `disablelastaccess = 1`

`fsutil behavior set disablelastaccess 1` nastavíme zákaz časového razítka při každém přístupu k souborům (to je praktické zvláště u SSD disků, kde každý nadbytečný zápis čehokoliv zbytečně snižuje životnost disku)

Položek, které lze takto zjišťovat nebo nastavovat, je více. Příkaz obsahuje vždy buď výraz `query` (zjišťování) nebo `set` (nastavování).

`fsutil dirty`

pracuje s příznakem *dirty* („špinavý“), se kterým jsme se setkali už u příkazu `chkntfs` (na straně 112)

`fsutil dirty query d:` zjistí, zda je u oddílu D: nastaven příznak dirty

`fsutil dirty set d:` jednotka D: je označena jako „špinavá“ (pak by měla být tato jednotka zkontrolována při příštím startu systému)

`fsutil hardlink`

práce s pevnými odkazy (u novějších verzí NTFS jsou podporovány pevné odkazy s podobnými vlastnostmi jako pevné odkazy v unixových souborových systémech, také původní cesta k souboru je chápána jako pevný odkaz a tedy na každý soubor vede vždy nejméně jeden)

`fsutil hardlink create d:\zkratky\novy.dat c:\cesta\puvodni.dat` vytvoří se pevný odkaz s názvem v prvním parametru za klíčovým slovem `create`, na existující soubor, který je posledním parametrem příkazu

Pevné odkazy lze používat stejně jako původní cesty k souborům, pevný odkaz lze vytvořit pouze na běžný soubor (pevný odkaz na adresář by mohl způsobit zacyklení při některých operacích v souborovém systému)

`fsutil quota`
práce s kvótami (o kvótách jsme se učili v kapitole 4.2.2 na straně 110). Kvóta stanovuje každému uživateli limit pro místo na jednotce, které může využívat pro své vlastní soubory. Obvykle je stanovena určitá tolerance (threshold), která sice může vyvolat „poplach“, ale uživatel ještě může do této stanovené míry kvótu překročit

`fsutil quota query d:` zobrazí nastavení diskových kvót
`fsutil quota violations` zobrazí porušení stanovených kvót
`fsutil quota track d:` povolí sledování kvót
`fsutil quota enforce d:` povolí vynucování kvót
`fsutil quota disable d:` zakáže používání kvót
`fsutil quota modify d: 1024 10000000 novak` vytvoří novou kvótu nebo pozmění existující pro zadaného uživatele; první číselný parametr určuje *toleranci* (threshold), druhý pak místo na jednotce vymezené zadanému uživateli (oba údaje v B)

Poznámka

Pro úplnost – souborový systém NTFS podporuje nejen pevné odkazy (které můžeme vytvářet například pomocí `fsutil`), ale i symbolické odkazy (to není úplně totéž jako zástupci). Lze je vytvářet (a také prohlížet a rušit) programem `junction`, který je ke stažení na stránkách https://www.sysinternals.com

Úkoly

1. Zobrazte nápovědu příkazu `fsutil` a zjistěte, jaké další podpříkazy podporuje.
2. Pokuste se pomocí tohoto příkazu vytvořit nový soubor o nulové délce a soubor o délce 1 KB (dejte pozor na to, v jakých jednotkách se délka souboru zadává). U obou souborů použijte příponu TXT a pak je otevřete v některém editoru (třeba Poznámkovém bloku).
3. Na některý ze souborů vytvořených v předchozím úkolu vytvořte pevný odkaz v tomtéž adresáři, ale jinak pojmenovaný. Projděte si pak vlastnosti tohoto pevného odkazu (v kontextovém menu).
4. Zjistěte, zda jsou nastaveny kvóty na jednotkách pevného disku ve vašem počítači.
5. Zjistěte, jestli jsou nastavována časová razítka při každém přístupu k souboru na jednotkách vašeho disku.

4.2.8 Streamy v NTFS

V souborovém systému NTFS můžeme ke každému souboru (i složce) přidružit stream – proud dat (nebo více streamů). Fyzicky každý soubor obsahuje alespoň jeden (hlavní) stream, který je pojmenovaný názvem souboru. Ostatní streamy jsou pojmenované pouze doplňkovým řetězcem za názvem souboru a jsou přístupné přes „dvojtečku“, nazývají se *alternativní datové streamy* (ADS, Alternative Data Streams).

Vytvořit stream můžeme buď programově (v procesu), anebo „ručně“ přesměrováním (například u krátkého řetězce stačí přesměrovat výstup příkazu `echo`, ale lze přesměrovat i obsah souboru).

Příklad

Vytvoříme textový soubor, jehož hlavní stream zůstane prázdný a ve dvou dalších streamech uložíme řetězce písmen.

`echo Ahoj > d:\soubor.txt:cesky` první stream obsahuje český pozdrav,

`echo Hello > d:\soubor.txt:anglicky` druhý stream obsahuje anglický pozdrav,

`more < soubor.txt:cesky` vypíšeme stream s českým obsahem,

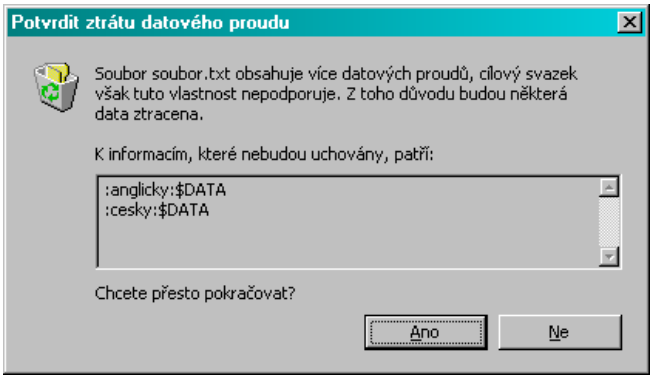
`more < soubor.txt:anglicky` vypíšeme další stream,

`type test.txt` vypíšeme obsah souboru – všimněte si dvou věcí:

- tímto příkazem se nevypíše nic, soubor je sám o sobě prázdný (streamy jsou „navíc“, nejsou obsaženy v souboru),
- k výpisu souboru můžeme použít příkaz `type`, ale k výpisu streamů ho použít nelze (musíme je vypsat příkazem `more` a také přesměrování je nutné), můžete vyzkoušet výpis streamu příkazem `type` nebo `more` bez směrování (nebo nemusíte, když nemáte rádi chybová hlášení),

`dir test.txt` vypsali jsme údaje o našem souboru – velikost je 0.

Je těžké poznat, jestli je u souboru ukrytý stream. Rychlé a spolehlivé (pokud máme podezření na jeden konkrétní soubor) je pokusit se tento soubor v grafickém režimu zkopírovat na diskový oddíl, jehož souborový systém nepodporuje streamy (třeba FAT32, což je obvyklé například na USB flash discích). Při pokusu o vložení „podezřelého“ souboru na takový diskový oddíl se objeví hlášení, které vidíme na obrázku 4.5. Takto zjistíme názvy streamů a můžeme se podívat i na jejich obsah (třeba postupem z příkladu 4.2.8).



Obrázek 4.5: Pokus o zkopírování souboru se streamy na FAT

Při práci se streamy můžeme buď používat postup z předchozího příkladu, anebo lze využít program `Streams` od firmy Sysinternals (je ke stažení na webu firmy). Tímto programem můžeme streamy také odstraňovat.

Postup

Ukážeme si použití programu `streams`.

`streams soubor.txt` vypíše streamy v zadaném souboru včetně jejich délky, výstup:

```
Streams v1.56 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2007 Mark Russinovich
Sysinternals - www.sysinternals.com

soubor.txt:
       :anglicky:$DATA 8
          :cesky:$DATA 7
```

`streams -d soubor.txt` odstraní všechny streamy u zadaného souboru.

`streams -s c:\` vypíše streamy všech souborů a adresářů na disku C: rekurzívně, zkrácený výstup:

```
Streams v1.56 - Enumerate alternate NTFS data streams
Copyright (C) 1999-2007 Mark Russinovich
Sysinternals - www.sysinternals.com
```

```
Error opening c:\pagefile.sys:
Proces nemá přístup k˜souboru, neboť jej právě využívá jiný proces.

c:\soubor.txt:
        :anglicky:$DATA 8
           :cesky:$DATA 7
c:\Documents and Settings\Uzivatel\Dokumenty\dokument.zip:
   :Zone.Identifier:$DATA       26
c:\WINDOWS\Thumbs.db:
     :encryptable:$DATA 0
...
```

 Ve Windows od verze Vista je příkaz `dir` obohacen o přepínač `/r`, který dokáže zobrazit i alternativní streamy:

```
dir /s /b /r c:\ | find "$DATA"
```

Poznámka

Stream *Zone.identifier* bývá obvykle přidáván Internet Explorerem při stažení souboru z internetu. Ne všechny streamy jsou bohužel takto nevinné. Protože běžný uživatel prakticky nemá šanci náhodně zjistit, že k souboru patří alternativní stream (dokonce se v běžných výpisech včetně Průzkumníka neprojeví ani jeho délka), mohou být streamy zneužívány malwarem k transportu nežádoucích dat či kódu. Pokud výše popsanými technikami najdeme alternativní stream s délkou větší než obvyklou (ve výpisu `streams` tří- nebo víceciferné číslo), měli bychom zpozornět a spustit antispyware.

Úkoly

1. Podobně jako v příkladu na straně 119, vytvořte soubor se dvěma streamy. Zjistěte délku souboru a kolik místa zabírá na disku.
2. Pokud máte možnost, spusťte rekurzivní vyhledávání streamů na vašem počítači.

4.3 Operační paměť

 Stránkovací soubor se obvykle jmenuje `pagefile.sys` a je na systémovém disku, ale ve skutečnosti můžeme mít více stránkovacích souborů (v tom případě by však každý měl být na jiném pevném disku). Názvy stránkovacích souborů najdeme v klíči `HKLM/SYSTEM/CurrentControlSet/Control/Session Manager/Memory Management`, a to v položce typu pole řetězců `PagingFiles`.

Poznámka

Pokud máme více stránkovacích souborů, měl by být každý na jiném *fyzickém* disku (pokud je jich víc na stejném fyzickém disku, třeba i na jiných oddílech, může to systém dokonce zpomalit).

 V tomtéž klíči je ještě jedna zajímavá položka – `ClearPageFileAtShutdown`. Pokud existuje a je nastavená na 1, před každým (regulérním) vypnutím systému se smaže stránkovací soubor. To je důležité z hlediska bezpečnosti, protože v stránkovacím souboru se mohou nacházet důležité informace, které by se neměly dostat do nepovolaných rukou (jistá verze Windows takto „zveřejnila“ nezašifrované heslo administrátora).

Úkol

Zjistěte, kolik máte stránkovacích souborů a jak se jmenují. Také zkontrolujte, zda máte nastaveno mazání stránkovacího souboru při každém vypnutí systému.

4.4 Správa sítě

Upozornění: podrobnější informace k následujícím příkazům a samozřejmě další příkazy a příklady najdeme ve skriptech do předmětu Počítačová síť a internet (cvičení).

4.4.1 Soubory související se správou sítě

Část parametrů souvisejících se správou sítě se nachází v registru, ale z důvodu kompatibility s jinými operačními systémy komunikujícími přes síť máme i ve Windows několik důležitých konfiguračních souborů souvisejících se sítí. Najdeme je v adresáři `...\system32\drivers\etc` (jsou to textové soubory, třebaže nemají příponu):

- `hosts` je soubor urychlující mapování IP adres na známé doménové adresy,
- `networks` obsahuje doménové a IP adresy lokálních sítí,
- `services` obsahuje informace o známých síťových službách,
- `protocol` totéž o známých síťových protokolech.

Soubory `hosts` a `networks` můžeme využít pro urychlení překladu názvů, který je za normálních okolností prováděn DNS serverem. Jestliže u některé často používané adresy známe dvojici název–IP adresa a víme, že IP adresa je staticky přidělena (tudíž se nemůže změnit), můžeme tuto dvojici zaznamenat do souboru `hosts` (pokud jde o adresu konkrétního uzlu v síti) anebo `networks` (pokud jde o adresu sítě či podsítě) a pak o překlad nemusí být zdlouhavě žádán DNS server.

Poznámka

Dostat se k těmto souborům může být trochu problém, pokud pracujeme v 64bitových Windows a jsme v 32bitovém procesu (což je většina souborových manažerů třetích stran). V 64bitovém systému je totiž adresář `System32` také 64bitový a při přístupu z 32bitového procesu dochází k přesměrování v režii Windows. Rozhodně bychom se do adresáře `etc` nedostali postupným procházením adresářovou strukturou, pouze přímým zapsáním celé cesty k tomu souboru, se kterým chceme pracovat (například `C:\windows\system32\drivers\etc\hosts`), anebo použitím 64bitového správce souborů.

4.4.2 Základní příkazy pro správu sítě

Zde si ukážeme jen několik nejtypičtějších úloh, s dalšími se setkáme v jiném předmětu.

`IPCONFIG`

zobrazí konfiguraci protokolu TCP/IP a umožňuje také jeho (omezenou) konfiguraci

`ipconfig /?` zobrazí se nápověda

`ipconfig /all` zjištění všech potřebných informací o síťových kartách včetně IP adresy, brány, masky podsítě a MAC adresy

`ipconfig /release název_karty` uvolnění IP adresy přidělené zadané síťové kartě (když neuvedeme název karty, jsou uvolněny IP adresy všech karet)

`ipconfig /renew název_karty` obnovení přidělení IP adresy pro zadanou síťovou kartu (když není název karty uveden, provede se pro všechny karty)

`ipconfig /displaydns` systém si ukládá do pomocné paměti předchozí provedené DNS překlady, aby je při opakovaném použití adresy nemusel pořád odesílat; tento příkaz zobrazí obsah této pomocné paměti

V PowerShellu zjistíme všechny naše IP adresy takto:

```
get-NetIPAddress | ft
```

Příklad

Chceme zjistit údaje o své síťové kartě – svou IP adresu, fyzickou (MAC) adresu, masku podsítě, adresu svého DNS serveru a další:

```
ipconfig /all
```

Zjistíme, že je nějaký problém – máme zřejmě špatně přiřazenu IP adresu (například stejnou jako jiný počítač v síti, to se občas může stát), proto IP adresu uvolníme a pak znovu kartu aktivujeme (znovu požádáme o IP adresu) – předpokládejme, že IP adresu dostáváme dynamicky přes DHCP, potom si znovu ověříme, jakou máme adresu:

```
ipconfig /release
ipconfig /renew
ipconfig /all
```

`PING`

ověřuje průchodnost připojení k zadanému počítači v síti, odesílá k tomuto počítači pakety a podle zaslaných odpovědí určuje, zda je počítač dostupný a jaká je odezva připojení (pokud má vzdálený počítač firewall nakonfigurovaný tak, aby požadavky `ping` byly ignorovány, může se počítač jevit jako nedostupný)

`ping www.google.com` zjistí, zda je zadaný server dostupný (je dostupný prakticky vždy, tedy pokud se zobrazí hláška, že tomu tak není, zřejmě nejsme připojeni k síti nebo je někde na cestě porucha)

`ping -n 2 www.google.com` tento parametr omezí (nebo u většího čísla navýší) počet odesílaných testovacích paketů, zde na 2; výchozí hodnota je 4 pakety

`ping -t www.google.com` počet odesílaných paketů není stanoven, posílají se opakovaně až do přerušení klávesovou zkratkou Ctrl+C

Příklad

Jako parametr příkazu `ping` lze použít i číselnou IP adresu. Dokonce to může být i adresa *loopback*, což je vlastně zpětná smyčka, díky této adrese může počítač komunikovat i sám se sebou (resp. procesy na témže počítači mezi sebou) přes síťové protokoly. Adresa loopback je 127.0.0.1, tedy můžeme otestovat:

```
ping 127.0.0.1
```

Tento příkaz je užitečný například tehdy, když chceme zjistit, zda je naše síťová karta funkční. Pokud by příkaz hlásil nedostupnost, máme problém se síťovou kartou.

Jestliže máme správně přidělenu IP adresu, ale spojení v některých síťových aplikacích přesto nefunguje, může být problém v nesprávné adrese DNS serveru (to je server, který překládá jmenné adresy na číselné IP adresy a případně zpět). Ověříme si příkazem, ve kterém použijeme IP adresu:

`ping 8.8.8.8`

(to je adresa veřejného DNS serveru Googlu). Pokud příkaz nehlásí chyby, pak je určitě problém právě v DNS serveru našeho poskytovatele internetu. Můžeme si nastavit jiný DNS server, třeba zrovna tento (jen je celkem vytížený, má horší propustnost).

`test-connection`

je obdoba příkazu `ping` pro PowerShell. Opět máme k dispozici různé parametry – především IP adresu nebo název testovaného zařízení, ale je možné zadat i další parametry. Například:

`test-connection www.google.com -count 2` zadali jsme také počet ICMP dotazů, které budou poslány (standardně 4, jako u příkazu `ping`)

`ROUTE`

pracuje se směrovacími tabulkami (routing tables); tam je především uvedena výchozí brána (tj. když není stanoveno jinak v pravidlech směrovací tabulky, je paket směrován právě na tuto bránu) a taky směrovací pravidla („pakety s cílem xxx veď na bránu yyy“).

`route print` vypíše směrovací tabulky (lze zadat také upřesňující parametry)

`route add` ⟨`IP adresa cíle`⟩ `MASK` ⟨`maska podsítě`⟩ ⟨`brána`⟩ `METRIC 1`
přidá do směrovací tabulky záznam se zadanou IP adresou, maskou podsítě, bránou a metrikou (zde je metrika 1), také je možné zadat rozhraní (to je IP adresa síťové karty v tomto počítači), lomené závorky samozřejmě *nepíšeme*, jsou zde jen pro usnadnění orientace

`route delete` ⟨`IP adresa`⟩ odstraní ze směrovací tabulky záznam pro zadanou IP adresu

Příklad

Směrovací tabulka určuje, kam má být který odchozí paket směrován. Na desktopu (Windows XP) může směrovací tabulka vypadat třeba takto:

```
C:\> route print

===========================================================================
Interface List
  5...70 b5 e8 aa 9e 34 ......Realtek PCIe GbE Family Controller
  3...........................Wintun Userspace Tunnel
  7...9c fc e8 fd 77 c7 ......Microsoft Wi-Fi Direct Virtual Adapter
 15...9c fc e8 fd 77 c6 ......Intel(R) Wireless-AC 9560 160MHz
  1...........................Software Loopback Interface 1
....
===========================================================================

IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0     193.84.196.1   193.84.199.123     40
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
     192.168.56.0    255.255.255.0         On-link      192.168.56.1    281
     192.168.56.1  255.255.255.255         On-link      192.168.56.1    281
   192.168.56.255  255.255.255.255         On-link      192.168.56.1    281
     193.84.196.0    255.255.252.0         On-link    193.84.199.123    296
   193.84.199.123  255.255.255.255         On-link    193.84.199.123    296
   193.84.199.255  255.255.255.255         On-link    193.84.199.123    296
```

```
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link       192.168.56.1    281
        224.0.0.0        240.0.0.0         On-link     193.84.199.123    296
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
  255.255.255.255  255.255.255.255         On-link       192.168.56.1    281
  255.255.255.255  255.255.255.255         On-link     193.84.199.123    296
===========================================================================
Persistent Routes:
  None

IPv6 Route Table
===========================================================================
Active Routes:
 If Metric Network Destination             Gateway
  1    331 ::1/128                         On-link
 22    281 fe80::/64                       On-link
 15    296 fe80::/64                       On-link
 22    281 fe80::c4fa:2bc0:8483:ad73/128   On-link
 15    296 fe80::d349:5fe5:dce3:93d2/128   On-link
  1    331 ff00::/8                        On-link
 22    281 ff00::/8                        On-link
 15    296 ff00::/8                        On-link
===========================================================================
Persistent Routes:
  None
```

Počítač má v tuto chvíli přidělenu IP adresu 193.84.199.123. Podívejte se, kde v tabulce se tato adresa nachází. Všimněte si, že pakety adresované sobě samému jsou směrovány na loopback (127.0.0.1). Brána pro místní segment lokální sítě je 193.84.196.1. Tuto adresu najdeme především na začátku sloupce *Gateway*.

IPv6 adresu toto zařízení zrovna nemá přidělenou, ale vidíme alespoň formát tabulky.

`TRACERT`

určuje trasu k zadaným počítačům včetně její délky pomocí protokolu ICMP:

- v paketu ICMP využívá hodnotu TTL, podle případné reakce směrovačů odhadne délku trasy,
- nejdřív vyšle paket s TTL=1, při předčasném vypršení TTL je vyslán paket s TTL=2, pak TTL=3, atd., dokud neobdrží kladnou odezvu.

Parametrem `-h` můžeme omezit počet směrování, aby při nemožnosti dosažení cíle příkaz nepracoval zbytečně dlouho.

`tracert www.google.com` zobrazí cestu k danému serveru včetně časových údajů (nejkratší, průměrná a nejdelší doba)

`tracert -h 5 www.google.com` zajímá nás pouze prvních 5 skoků přes směrovače na cestě

`PATHPING`

vypíše cestu k zadané adrese a vypočítá statistiky související s jednotlivými úseky cesty (vpodstatě náročnost úseků cesty). Jde o kombinaci funkcí příkazů `ping` a `tracert` – je nejen určena trasa přes směrovače k danému cíli, ale ke každému směrovači jsou posílány pakety podobně jako u příkazu `ping`. Takto lze zjistit spolehlivost cest k jednotlivým směrovačům (počet ztracených paketů, dobu odezvy apod.) a tím například odhadnout, kde na cestě dochází k problémům s doručováním paketů.

`pathping www.google.com` zjistí stav cesty k zadanému serveru, vypíše podrobnou statistiku cesty (generování statistiky může trvat i několik minut)

Výstup můžeme upřesnit zadáním parametrů, také lze stanovit jiný výchozí počítač. Příkaz je užitečný například tehdy, když cesta k některému cíli vykazuje hodně špatnou propustnost, a my potřebujeme zjistit, na kterém úseku je problém.

`NETSTAT`

tento program zobrazuje statistiku protokolů TCP/IP

`netstat` vypíše základní statistiku (nevšímá si aplikací, které pouze naslouchají, vypisuje pouze TCP spojení)

`netstat -a` vypíše celou statistiku

`netstat -a -o` vypíše celou statistiku, přidá sloupec s PID příslušného procesu

`netstat -ao` totéž (přepínače můžeme sdružovat do jediného řetězce)

`netstat -ano` navíc místo doménových vypisuje IP adresy

`netstat -abno` podobně jako předchozí, ale ke každému řádku přidá informaci o názvu procesu (ne PID, opravdu název spustitelného souboru)

`netstat -ab > seznam.log` vypíše všechny procesy (názvy i PID), které právě komunikují se sítí (jakkoliv), výsledek uloží do zadaného souboru (opět jsme přepínače shrnuli k jedné pomlčce)

`netstat -e` základní statistika pro Ethernet (počet odeslaných a přijatých paketů, počet chyb apod. – týká se protokolů rodiny TCP/IP)

`netstat -es` podrobná statistika *všech* protokolů z rodiny TCP/IP

`netstat 2` vypisuje svůj výstup opakovaně každé dvě sekundy (můžeme zadat jakýkoliv interval), činnost příkazu lze ukončit jen klávesovou zkratkou Ctrl+C

`ARP`

ARP je protokol pro evidenci adres nejbližších „sousedů“ – pro každé zařízení, s nímž jsme propojeni, tu najdeme IPv4 adresu a do páru MAC adresu; tento příkaz umožňuje pracovat s tabulkou těchto údajů.

`arp -a` zobrazí ARP tabulku, také vidíme, které záznamy jsou dynamické (automaticky vytvořené z komunikace) a které statické (ručně vložené)

`arp -s 123.123.123.123 00-12-34-56-78-9A` přidá do ARP tabulky statický záznam se vztahem zadané IP adresy a MAC (fyzické) adresy, MAC adresa se zadává v hexadecimálních číslech s pomlčkami

`arp -d 123.123.123.123` odstraní z ARP tabulky záznam pro zadanou IP adresu

`arp -d *` odstraní z ARP tabulky všechny záznamy

Lze také přidat parametr s určením IP adresy síťové karty, pro kterou daná ARP tabulka platí (každá karta má svou tabulku), což má smysl pouze v případě, kdy je v provozu více než jedna síťová karta.

Příklad

Na desktopu s jedinou síťovou kartou je dost pravděpodobné, že ARP tabulka (tedy tabulka sousedů) bude vypadat takto:

```
C:\> arp -a

Rozhraní: 193.84.195.63 -- 0x2
  internetová adresa    fyzická adresa         typ
  193.84.195.1          00-0d-66-2c-7a-00      dynamická
```

Je to z toho důvodu, že v současných ethernetových sítích mívají tyto uzly jediného souseda – přepínač nebo směrovač, ke kterému jsou připojeny (a to je také jim přiřazená brána, přes kterou kumunikují se světem, všimněte si adresy). Pokud jsou v tabulce další záznamy, zřejmě budou statické, a obvykle se bude jednat o skupinové adresy.

`netsh interface ipv6 show neighbors`
zatímco příkaz `arp` funguje jen na IPv4 adresy, takto získáme obdobné informace pro IPv6 (odpovídající protokol se nejmenuje ARP, ale NDP, vypisujeme NDP tabulku, tedy tabulku IPv6 sousedů) – tento příkaz budeme brát později, zde jen pro úplnost

`get-netneighbor`
takto v PowerShellu získáme výpis ARP/NDP tabulky sousedů pro IPv4 a IPv6

`new-netneighbor`
vytvoření nového záznamu v ARP/NDP tabulce v PowerShellu, podobně `remove-netneighbor` je pro odstranění záznamů z dané tabulky

Úkoly

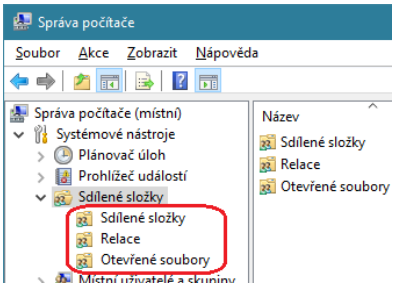
1. Prohlédněte si obsah souborů `networks`, `hosts`, `services` a `protocols` v adresáři `...\system32\drivers\etc` (pozor, pokud pracujete na 64bitovém systému).
2. Zjistěte svou IP a MAC adresu, a také masku podsítě.
3. Zobrazte seznam informací o DNS názvech (včetně IP adresy, evidovaných hodnot TTL atd.).
4. Vyberte si jakýkoliv server na síti (třeba svůj mail server nebo server, který používáte při vyhledávání) a zjistěte jeho dostupnost, při testu použijte pouze dva pakety. Na tentýž server vyzkoušejte příkaz `ping` v Příkazovém řádku a cmdlet `test-connection` v Powershellu. Získané údaje srovnejte.
5. Vypište směrovací tabulky na vašem počítači. Všimněte si, jakou metriku má zařízení s IP adresou začínající čísly 127.0.0.
6. Porovnejte výpisy příkazů `tracert` a `pathping` pro tentýž server.
7. K serveru použitému v úkolu 4 zjistěte trasu (adresy) včetně určení časové náročnosti trasy k jednotlivým uzlům (směrovačům) na cestě.
8. Zjistěte, které procesy (všechny) právě pracují se sítí. Zjistěte, jak přidáním příslušného filtru vypsat pouze ty procesy, které naslouchají na některém portu.
9. Projděte si nápovědu příkazu `netstat` (v Příkazovém řádku) a zjistěte, kterým přepínačem lze omezit výpis pouze na protokol TCP.
10. Zobrazte ARP/NDP tabulku síťové karty na vašem počítači. Vyzkoušejte i příkaz v PowerShellu.

4.4.3 Varianty příkazu `NET` pro práci se sdílenými prostředky

Příkaz `NET` je komplexní nástroj pro práci především se sítí, ale taky je určen pro některé úkoly související se správou počítače a uživatelů (Windows řady NT). Už jsme se s ním setkali – seznámili jsme se s variantami pro práci s uživateli, skupinami, zásadami správy účtů.

K jednotlivým variantám příkazu získáme krátkou nápovědu v textovém režimu – stručný popis syntaxe (například `net view /?`), podrobnější nápovědu najdeme v grafickém režimu, když do vyhledávacího okna napíšeme net view.

Každý by měl mít přehled o tom, co je na jeho počítači nasdíleno „ven“ a je tedy dostupné z jiného počítače, podobně by měl vědět, jak zjistit, co je dostupné na jiných počítačích a jak se k tomu dostat. Využijeme několik podpříkazů příkazu `net`: varianta `net share` se vztahuje k tomu, co máme na svém počítači a může být viditelné směrem ven, zatímco varianta `net view` se naopak vztahuje k tomu, co se nachází na jiných počítačích a chceme to používat na tom našem. K tomu, co máme u sebe a je nasdíleno směrem ven, jsou také příkazy `net file` a `net session`.



Obrázek 4.6: Ekvivalent příkazů `net share`, `net session` a `net file` v GUI

`NET SHARE`

(podobně ve *Správě počítače*, položka *Sdílené složky* ⇨ *Sdílené složky*) práce se sdílenými prostředky (obvykle adresáři/složkami nebo tiskárnami) na našem počítači (nasdílet prostředek, ukončit sdílení, bez dalších parametrů zobrazí seznam nasdílených prostředků), umožňuje také omezit sdílení jen na určitý počet uživatelů

`net share` zobrazí seznam všech prostředků (složek, tiskáren apod.), které jsou na tomto počítači nasdíleny (jsou přístupné z jiného počítače na síti). V seznamu jsou také položky neviditelné v nástroji *Síť* (končí symbolem $).

`net share fakturydod=d:\faktury /remark:"Faktury dodavatelů"` vytvoří nové sdílení – adresář `d:\faktury` bude na síti přístupný, a to buď přes nástroje s grafickým rozhraním nebo přes UNC adresu `\\pocitac\fakturydod` (předpokládáme, že pracujeme na počítači `pocitac`), je také připojen komentář prostředku

`net share fakturydod /delete` odstraní sdílený prostředek, ten přestane být přístupný ze sítě

`NET FILE`

(podobně ve *Správě počítače*, položka *Sdílené složky* ⇨ *Otevřené soubory*) práce se sdílenými otevřenými soubory (soubory z nasdílených složek, s nimiž právě někdo pracuje)

`net file` vypíše seznam otevřených sdílených souborů s informací (uživatel na síti, který soubor používá, cesta k souboru, apod.), taky zde zjistíme ID (identifikační číslo) nasdíleného souboru

`net file 2` vypíše informace o využívání nasdíleného souboru, jehož ID je 2 (to jsme zjistili z předchozího výpisu)

`net file 2 /close` uzavřeme soubor (se zadaným ID), který někdo otevřel ze sítě, tím také soubor odblokujeme (soubor používaný ze sítě je uzamčen, blokován proti změnám)

`NET SESSION`

(podobně ve *Správě počítače*, položka *Sdílené složky* ⇨ *Relace*) práce s vnějšími připojeními na náš počítač (session neboli relace je navázané připojení mezi naším počítačem a jiným počítačem v síti)

`net session` vypíše seznam počítačů, ze kterých jsou navázány relace na náš počítač (včetně dalších informací – uživatele, jeho operační systém, počet prostředků, se kterými na našem počítači pracuje a dobu nečinnosti)

`net session \\pocitac` podrobnější informace o relaci navázané ze zadaného počítače

`net session \\pocitac /delete` ukončení relace (pokud nezadáme název počítače, ukončí se všechny navázané relace)

`NET VIEW`

zobrazí seznam sdílených prostředků na určeném počítači v doméně (zjišťujeme, jaké prostředky jsou dostupné na jiném počítači v síti)

`net view` zobrazí seznam počítačů v doméně

`net view \\pocitac` vypíše seznam sdílených prostředků, které nabízí daný počítač

`net view /domain` vypíše seznam domén v síti

`net view /domain:ucetni` vypíše seznam počítačů, které jsou v zadané doméně

`net view /network:NW` vypíše seznam serverů v síti Novell Netware

`net view /network:NW \\pocitac` v síti Novell Netware vypíše prostředky dostupné na zadaném počítači

`NET USE`

práce se sdílenými prostředky na ostatních počítačích v síti

`net use` zobrazí existující síťová připojení (cesta na síti, ale pokud je prostředek namapován jako logická jednotka s písmenem, pak i označení jednotky)

`net use W: \\pocitac\fakturydod` přiřadíme písmeno W zadanému prostředku ze sítě, od této chvíle můžeme k tomuto prostředku přistupovat přes `W:`

`net use W: \\pocitac\fakturydod /user:uživatel` podobně, ale na cílovém počítači budeme k jednotce přistupovat jako zadaný uživatel (vyžaduje zadání hesla), pokud je cílový počítač v jiné doméně, musíme zadat i tuto doménu

`net use W: \\pocitac\fakturydod /delete` odstraníme vazbu

`net use W: /delete` totéž (název prostředku nemusíme zadávat)

`net use /persistent:a` momentálně platná připojení budou uložena a zůstanou platná i po restartu (příp. vypnutí a dalších startech) počítače

Poznámka

Všimněte si, že pokud chceme určitému datovému úložišti (konkrétně složce či adresáři na něm) „přiřadit písmeno“, jsou kompetence rozděleny takto:

- příkaz `subst` použijeme, pokud se jedná o složku na místním datovém médiu, případně se na nižší úrovni dá použít příkaz `mountvol`,
- příkaz `net use` použijeme, pokud se jedná o složku někde v místní síti.

Pozor, nepleťte si příkazy `net use` a `net user` – rozdíl je v jediném písmenu, ale funkčnost zcela jiná.

Příklad

Předpokládejme, že máme v lokální síti dva počítače – desktop a notebook. Zjistíme, co je z desktopu viditelné v lokální síti:

```
C:\> net share

Název sdílené
položky                  Prostredek                      Poznámka
-------------------------------------------------------------------------------
ADMIN$                   C:\Windows                      Vzdálený správce
C$                       C:\                             Výchozí sdílená položka
print$                   C:\Windows\system32\spool\drivers Ovladace tiskárny
IPC$                                                     Vzdálený IPC
themes                   P:\zaloha\themes
```

```
Users                       C:\Users
zaloha                      P:\zaloha
EPSON BX620FWD Series       USB001                  Zarazeno   EPSON BX620FWD Series
EPSON BX620FWD Series(FAX) USB001                   Zarazeno   EPSON BX620FWD Series (FAX)
Príkaz byl úspešne dokoncen.
```

Teď se podíváme, co je ve skutečnosti vidět z jiného počítače. Pokud chceme zůstat u téhož počítače a jen se na sebe podívat „zvenčí“, stačí použít příkaz

`net view 127.0.0.1`

(nebo dosadíme svou konkrétní IP adresu). To sice funguje, ale bohužel i tehdy, když samotná síť není zcela vpořádku a může to vytvořit falešný pocit jistoty, že je vše funkční. Jinou možností je sednout si k jinému počítači v síti a provést následující postup (začátek je relevantní jen tehdy, pokud je předem nutné zfunkčnit sdílení v lokální síti).

Povolíme služby související s lokální sítí (především Prohledávání počítačů, může být třeba povolit ještě několik dalších). Počítače v lokální síti by měly být ve stejné pracovní skupině (ověříme) a dále může být problém ve vynucení přístupu s heslem (buď správně nastavíme heslo na počítačích nebo tuto možnost vypneme, například ve Windows 7 v *Centru síťových připojení a sdílení* ⇨ *Změnit pokročilé nastavení sdílení*.

Seznam viditelných strojů v lokální síti bude stejný na obou počítačích, pořadí je podle abecedy.

```
C:\> net view

Název serveru           Poznámka
-------------------------------------------------------------------------------
\\DOMACIPC
\\NOTEBOOK
Príkaz byl úspešne dokoncen.
```

Podíváme se, co konkrétně je viditelné na desktopu (sedíme u notebooku). Počítač můžeme určit buď jeho lokálním názvem (tím, který je ve výpisu výše) anebo jeho IP adresou. Ukážeme si přístup pomocí IP adresy (předpokládejme, že jsou používány soukromé IP adresy třídy „A“, desktop má 10.0.0.1, notebook 10.0.0.2):

```
C:\> net view \\10.0.0.1

Sdílené prostředky na \\10.0.0.1

Název sdílené položky        Typ   Použito jako  Komentář
-------------------------------------------------------------------------------
EPSON BX620FWD Series        Tisk                EPSON BX620FWD Series
EPSON BX620FWD Series (FAX)  Tisk                EPSON BX620FWD Series (FAX)
themes                       Disk
Users                        Disk
zaloha                       Disk
Příkaz byl úspěšně dokončen.
```

V seznamu je tiskárna, fax a pak tři sdílené složky. Všimněte si, že položky z `net share` začínající symbolem `$` zde nejsou. Můžeme to provést také naopak (na desktopu si vypsat seznam z notebooku):

```
C:\> net view notebook

Sdílené prostredky na notebook

Název sdílené položky  Typ   Použito jako  Komentár
-------------------------------------------------------------------------------
```

```
Public                Disk
Users                 Disk
zaloha                Disk
Príkaz byl úspešne dokoncen.
```

Zůstaneme na desktopu. Pokud máme příslušná přístupová oprávnění, lze nasdílené položky z jiného počítače volně využívat. Například si můžeme vypsat obsah adresáře nasdíleného na jiném počítači a otevřít některý ze souborů ve sdíleném adresáři:

```
C:\> dir \\notebook\zaloha

 Svazek v jednotce \\notebook\zaloha je DATA.
 Sériové číslo svazku je 5A37-8E6F.
 Výpis adresáre \\notebook\zaloha

18.10.2010  08:39    <DIR>          .
18.10.2010  08:39    <DIR>          ..
19.04.2002  11:32           378 945 disk6.zip
18.10.2010  08:39    <DIR>          nejnovejsi
19.04.2000  09:38             8 512 ukazka1.pas
11.06.2000  16:59            58 122 ukazka2.pas
           Souboru:      2,   Bajtu:                274 065
           Adresáru:     3,   Volných bajtu:  9 649 033 216

C:\> notepad \\notebook\zaloha\ukazka1.pas
```

Pozor, stejným způsobem se dají využívat prostředky, které ve výpisu `net view` nejsou vidět (končí symbolem `$`):

```
dir \\notebook\ADMIN$
```

Na notebooku si zkontrolujeme, které položky a soubory využívá někdo z jiného počítače:

```
C:\> net file

ID          Cesta                                Uživatelské jméno   Počet
                                                                     uzamčení
-------------------------------------------------------------------------------
57          D:\pocitac\zaloha\                   Guest               0
59          D:\pocitac\zaloha\ukazka1.pas        Guest               0
Příkaz byl úspěšně dokončen.
```

Vidíme, že někdo přistupoval ke sdílenému adresáři a pak také k jednomu souboru. Žádná položka není uzamčena (což je normální, protože žádný z použitých příkazů uzamčení nepožaduje). V případě, že soubor otevřeme v programu, který soubory uzamyká (například ve Wordu), můžeme zkusit příkaz `net session` a zjistit, ze kterých počítačů v síti jsou využívány prostředky našeho počítače.

```
C:\> net session

Počítač             Uživatel      Typ klienta   Otevření   Doba nečinnosti
-------------------------------------------------------------------------------
\\10.0.0.1          Guest                       2          00:03:27
Příkaz byl úspěšně dokončen.
```

Zjistíme si údaje pro dotyčný počítač, o kterém teď víme, že využívá prostředky našeho počítače:

```
C:\> net session \\10.0.0.1
Uživatelské jméno     Guest
Počítač               10.0.0.1
Přihlášení hosta      Ne
```

```
Typ klienta
Doba relace           00:04:40
Doba nečinnosti       00:03:36

Název sdílené položky     Typ     Počet otevření
-------------------------------------------------------------------------
zaloha                    Disk    2
Příkaz byl úspěšně dokončen.
```

Uživatel z cizího počítače zřejmě dvakrát přistupoval k nasdílenému adresáři, soubor už žádný otevřený nemá. Všimněte si, že se na náš počítač vlastně ani nepřihlašoval, pracuje pod uživatelským účtem Guest (host). Kdybychom měli zakázán účet hosta, uživatel by takto naše prostředky nemohl využívat.

Úkoly

1. Zobrazte seznam všech prostředků, které jsou z vašeho počítače přístupny v síti (které máte nasdíleny).
2. Na disku, kde máte právo zápisu (předpokládejme `D:`, může to být i hlouběji v adresářové struktuře) vytvořte adresář s názvem `Pokusný adresář` (do něj umístěte nějaké nepříliš důležité soubory a adresáře). Potom tento adresář zpřístupněte na síti pod názvem `pokusny`. Zkontrolujte, zda je opravdu dostupný (název svého počítače zřejmě znáte, pokud ne, zjistěte ho). Nakonec zrušte sdílení a adresář odstraňte z disku. Rušení můžete odložit na dobu po provedení následujících úkolů v této posloupnosti.
3. Zjistěte, zda na některém počítači v síti nejsou nějaké sdílené prostředky (týká se i vašeho počítače). Dále si vyberte některý počítač a zobrazte seznam prostředků, které na síti nabízí. Pokud je některým z těchto prostředků adresář, připojte si ho jako disk pod nějakým vhodným volným písmenem a pak si obsah tohoto adresáře prohlédněte v některém souborovém manažerovi (případně Průzkumníkovi) s využitím takto vytvořeného přístupu k síťovému disku. Potom připojení zrušte (můžete odložit na dobu po provedení posledního úkolu v této posloupnosti).
4. Zjistěte, jestli z některého počítače na síti vede relace do vašeho počítače (zda někdo zvenčí používá prostředky vašeho počítače). O zjištěné relaci zjistěte podrobnější informace (z předchozího výpisu znáte název počítače, ze kterého relace vede).

4.4.4 Další varianty příkazu `NET`

Podíváme se ještě na zbývající varianty příkazu `NET`.

`NET HELPMSG`

poskytuje nápovědu k chybovým zprávám Windows (parametrem je číslo zprávy)

`net helpmsg 2182` pokud se nám při spouštění některé služby objevila hláška s číslem chyby 2182, použijeme tento příkaz, který nám zobrazí podrobnější komentář, v tomto případě informace, že jsme se pokusili spustit službu, která je již spuštěna

Některé zprávy jsou poněkud kuriózní – při neúspěšném pokusu o spuštění jedné ze služeb můžete být odkázáni na chybu číslo 3534, ale po zobrazení příslušné informace zjistíte, že to znamená „Služba neoznámila chybu.“

`NET COMPUTER`

přidá nebo odstraní počítač z domény

`net computer \\pocitac /add` přidá zadaný počítač do domény

`net computer \\pocitac /del` odstraní zadaný počítač z domény

`NET TIME`

synchronizace hodin počítače s hodinami jiného počítače nebo domény, nebo můžeme zobrazit aktuální čas na jiném počítači (souvisí s protokolem NTP – Network Time Protocol)

`net time \\CasovyServer` synchronizujeme systémové hodiny s hodinami zadaného časového serveru (obecně nějakého zařízení, které tuto síťovou službu poskytuje)

`net time /domain` synchronizujeme čas s primárním řadičem domény (také můžeme zadat konkrétní doménu, pokud je jich více), ovšem musíme patřit do domény

Úkoly

1. Zjistěte, k čemu slouží příkazy `net pause` a `net continue`. Najděte (třeba v nápovědě) informaci o tom, co se stane, když první z uvedených příkazů použijete na službu *Server*.
2. Zjistěte, co znamená chybová zpráva
   - č. 3515 (vypíše se při použití příkazu `net group` na lokálním počítači),
   - č. 1058 (vypíše se při pokusu o spuštění některých služeb),
   - č. 3912 a 3913 (vypíšou se při nesprávných způsobech získání informací o synchronizaci hodin).
3. Pokud jste připojeni k síti, pokuste se zjistit, zda jsou systémové hodiny na vašem počítači synchronizovány s některým serverem (může jít o server v doméně nebo na internetu, může nebo nemusí to být server s protokolem NTP).

4.4.5 NetShell

Příkazem `NETSH` spustíme textovou konzolovou aplikaci *NetShell* (Network Services Shell). Tato konzola slouží ke konfiguraci některých částí systému většinou souvisejících se sítí. Pracujeme s *moduly* (také *helper*, obvykle dynamické knihovny) a *kontexty* (sada úloh – příkazů pro jednotlivé moduly, každý kontext může odpovídat jednomu modulu, některé kontexty mohou mít další podkontexty).

Takže se vlastně jedná o jakési společné pracovní a programové prostředí pro běh kontextů uložených v dynamicky linkovaných knihovnách, na momentálním vybavení systému těmito knihovnami záleží, které kontexty budeme mít k dispozici. Ve Windows 10/11 se Microsoft pokouší potlačovat využívání NetShellu a směrovat uživatele k PowerShellu, nicméně pořád existují úlohy, pro které v PowerShellu není cmdlet.

To, se kterými kontexty můžeme pracovat, tedy závisí na verzi, vybavenosti a nastavení operačního systému. Obvykle jsou kromě jiných dostupné tyto kontexty:

- `interface` – konfigurace protokolů rodiny TCP/IP (tj. síťového *rozhraní*), má podkontext `ip` a případně také `ipv6` (pro IP verze 6)
- `routing` – funkce související se směrováním, konfigurace směrovacích serverů, má podkontexty:
  - `ip` – konfigurace protokolů IP, obsahuje kontexty `autodhcp`, `dnsproxy`, `igmp`, `nat`, `ospf`, `relay`, `rip`, `routerdiscovery`
- `ras` – konfigurace RAS (Remote Access Server, server vzdáleného přístupu), podkontexty:

  - `ip`, `netbeui`
  - `aaaa` – práce s databází *AAAA* (Authentication, Authorization, Accounting, Auditing), která se používá například při autentizaci na síti
- `dhcp` – používá se na DHCP serveru ke konfiguraci DHCP (je v serverových variantách Windows), lze například přidat nový DHCP server do seznamu autorizovaných serverů
- `http` (konfigurace protokolu HTTP a HTTPS), `ipsec` (pracujeme se zásadami a statistikami protokolu IPSec), atd. pro další protokoly
- `advfirewall` – konfigurace rozšířených možností firewallu
- `lan`, `wlan` (wireless), `mbn` (mobile) – konfigurace ethernetové, bezdrátové a mobilní sítě
- případně další, tento nástroj lze rozšiřovat

Do vybraného kontextu se přepneme zadáním jeho názvu (pohybujeme se ve „stromové struktuře“), do nadřízeného kontextu se dostaneme zadáním `..` (dvě tečky, o úroveň výše).

Pracuje se zde v textovém režimu, používáme pouze příkazy specifické pro *NetShell*. Každý kontext má svou sadu příkazů, část je stejná ve všech kontextech, například (mnohé příkazy se někdy používají s dalšími parametry):

- `help` zobrazí všechny použitelné příkazy, je možné také použít symbol `?`
- `exit` odchod z `netsh`, také je možné použít příkaz `bye`
- `show` zobrazí informace o kontextu
- `set` nastavení konfigurace pro daný kontext
- `add` přidá položku konfigurace do seznamu (skriptu)
- `delete` odstraní položku konfigurace
- `dump` zobrazí konfigurační skript, přesměrováním ho můžeme uložit do souboru
- `exec` spustí soubor skriptu (obvykle jde o načtení konfigurace, kterou jsme předchozím příkazem někdy v minulosti uložili do souboru)

Příklad

Příkazem `netsh` spustíme prostředí *NetShell*u; zobrazí se typický prompt

```
netsh>
```

(při změně kontextu je promptem momentální kontext) a dále zadáváme tyto příkazy:

`routing ?` zobrazíme nápovědu ke kontextu *routing*

`routing` přesuneme se do kontextu *routing*, prompt se změní na `routing>` a můžeme zadávat příkazy patřící do vybraného kontextu

`show` zobrazí se informace o tom, co v tomto kontextu můžeme zjistit

`show helper` zobrazí se seznam základních příkazů typických jen pro tento kontext s informací pro tyto příkazy (jakých DLL knihoven se týkají a jejich GUID), příkazy jsou `ip` a `ipx`

`ip` přesuneme se do kontextu *ip* uvnitř kontextu *routing*

`show` zobrazí se informace o tom, co v tomto kontextu můžeme zjistit (protože jsme v kontextu *routing* a v jeho podkontextu *ip*, jedná se většinou o různé směrovací informace týkající se protokolu IP)

`show rtmdestinations` v předchozím výpisu jsme zjistili, že můžeme použít tento příkaz – vypíšeme seznam adres ve směrovací tabulce (ke každé položce IP adresa, brána, MAC adresa, metrika apod.)

`..` přesuneme se o úroveň výše (opustíme kontext *ip*, jsme v kontextu *routing*)

`..` přesuneme se o úroveň výše (opustíme kontext *routing*)

`interface` přesuneme se do kontextu *interface*

`ip` přesuneme se do kontextu *ip* uvnitř kontextu *interface* (oba poslední příkazy můžeme sjednotit v jediném příkazu, a to `interface ip`)

`show` zobrazí se informace o tom, co v tomto kontextu můžeme zjistit (všimněte si, že možnosti jsou jiné než v kontextu *ip* uvnitř *routing*, teď jde o informace týkající se rozhraní protokolu IP)

`show config` zobrazíme základní konfiguraci rozhraní protokolu IP

`show tcpconn` zobrazí se tabulka existujících připojení přes protokol TCP (TCP connections)[2]

`dump` zobrazí se konfigurace rozhraní protokolu IP

`exec ?` zobrazíme nápovědu k příkazu, který slouží k obnově konfigurace ze zálohy (načtení skriptu s kontextem)

`exit` ukončíme práci v interaktivním rozhraní NetShellu

Kterýkoliv z výše uvedených příkazů můžeme zadat i mimo prostředí NetShellu, například:

```
netsh -c interface show config
```

zobrazí konfiguraci rozhraní IP bez nutnosti přechodu do prostředí NetShellu (přepínač `-c` slouží k zadání kontextu). Další příkaz

```
netsh -c ras aaaa show authserver
```

zobrazí RADIUS server, který se používá k ověřování identity (pokud ovšem takový server používáme, autentizaci lze provádět různými způsoby).

To byla krátká ukázka základního ovládání NetShellu. V následujících příkladech se zaměříme na konkrétní úlohy.

Příklad

Kterýkoliv kontext (jako celek) můžeme zálohovat (záloha se však provádí zvenčí, abychom mohli přesměrovávat výstup):

```
netsh -c interface ip dump > kontextinterip.dat
```

Výsledek je uložen do textového souboru a umístěn do pracovního adresáře, pokud nezadáme cestu. Tento textový soubor můžeme buď pouze zálohovat, nebo exportovat a načíst na jiném počítači, anebo pozměnit a načíst se změnami.

Výsledkem předchozího příkazu je textový soubor, který má přibližně tento obsah (na počítači s jednou síťovou kartou, jejíž MAC adresu vidíme – pozměněnou, IP adresa je přiřazována přes DHCP):

```
# ----------------------------------
# Konfigurace rozhraní protokolu IP
# ----------------------------------
pushd interface ip
# Konfigurace protokolu IP rozhraní pro "418DA6F5-8B14-4F21-B6AD-43452BF31CA7"
set address name = "418DA6F5-8B14-4F21-B6AD-43452BF31CA7" source = dhcp
set dns name = "418DA6F5-8B14-4F21-B6AD-43452BF31CA7" source = dhcp
set wins name = "418DA6F5-8B14-4F21-B6AD-43452BF31CA7" source = dhcp
popd
# Konec konfigurace protokolu IP rozhraní
```

[2] Všimněte si, že se v záhlaví výpisu objevuje řetězec „MIB-II“. Jedná se o objektovou databázi, ve které jsou uloženy informace o uzlech v síti. S touto databází pracuje protokol SNMP a pokud umíme k MIB II přistupovat (a máme přístupová oprávnění), můžeme získat hodně zajímavých informací o síti.

Příkazy `pushd` a `popd` slouží k uložení původního kontextu, načtení nového a opětovném návratu k původnímu kontextu.

Ze zálohy můžeme kontext kdykoliv obnovit (načíst), a to i zvenčí:

```
netsh -c interface ip exec kontextinterip.dat
```

Protože je přechod do určitého kontextu součástí skriptu, lze skript načíst (spustit) také jinak – pomocí parametru `-f`:

```
netsh -f kontextinterip.dat
```

Příklad

Kdykoliv můžeme získat nápovědu, a to i k syntaxi jednotlivých příkazů. Například pokud jsme v kontextu *ip* uvnitř kontextu *interface*, lze se zeptat:

```
show ?
```

```
show dns ?
```

Zjistíme, že dotaz na nastavení DNS serveru můžeme formulovat jednoduše tako:

```
show dns
```

Zjistili jsme, jaké DNS servery jsou nastaveny (může jich být i více, pokud jde o statické adresy). Zjistíme, co lze nastavovat – příkaz `set` slouží k nastavování existujících vlastností.

```
set ?
```

Vybereme si nastavení prostředí DNS, ale protože neznáme syntaxi, zeptáme se:

```
set dns ?
```

Vypíše se krátký popis včetně ukázkových příkladů. Teď už by nám mělo být všechno jasné. Pokud chceme nastavit statickou adresu DNS serveru, použijeme příkaz

```
set dns "Připojení místní sítě" static 190.52.192.10
```

(doplníme řetězec označení svého rozhraní a skutečnou IP adresu DNS serveru).

NetShell je součástí Windows od verze 2000 (na serverech i desktopech) a různé verze se liší vybavením moduly. Například od verze XP přibývá modul (helper) s kontextem *firewall*.

Příklad

Ukážeme si práci s firewallem ve Windows pomocí NetShellu. Tedy zobrazujeme pravidla, přidáváme nová, odstraňujeme je.

`advfirewall firewall` přesuneme se do kontextu *advfirewall* a hned v tomtéž příkazu do jeho podkontextu `firewall`

`show` ověříme, co vše lze zobrazit, zjistíme, že použitelné je `show rule`

`show rule ?` požádáme o podrobnosti

`show rule name=all` pokusíme si nechat vypsat všechna nastavená pravidla, ale seznam je příliš dlouhý, pravidla vypadají nějak takto:

```
Název pravidla:                         Windows Live Messenger (UPnP-In)
----------------------------------------------------------------------
Povoleno:                               Ano
Směr:                                   In
Profily:                                Doména,Privátní,Veřejná
Seskupení:                              Windows Live Messenger
LocalIP:                                Any
```

```
Vzdálená IP adresa:                     LocalSubnet
Protokol:                               TCP
Místní port:                            2869
Vzdálený port:                          Any
Funkce Edge traversal:                  No
Akce:                                   Allow
```

Podobně vypadají i další pravidla. Pravidlo je povoleno, směr je „In“, tedy se jedná pravidlo pro příchozí provoz (pro odchozí by bylo „Out“), jsou stanoveny síťové profily, pro které pravidlo platí, pravidla jsou seskupována (group) pro snazší správu, dále jsou určeny adresy, protokol, port. Pokud nám nestačí konec výpisu (záleží na velikosti vyrovnávací paměti Příkazového řádku, kolik údajů bude viditelných), pak musíme příkaz spustit mimo prostředí NetShellu a výstup směrovat do souboru:

```
netsh advfirewall firewall show rule name=all > d:\vystupfirewall.txt
```

(ale pak je třeba použít editor, který zvládá znakovou sadu Latin2, například PSPad (je třeba nastavit v menu *Zobrazit* ⇨ *Zobrazení znaků MSDOS*)

Bohužel musíme zadat buď „all“ nebo konkrétní název pravidla, dále lze filtrovat – omezovat výpis (ale moc možností nemáme)

`show rule name=all type=dynamic dir=in profile=public` zkonkrétnili jsme dotaz, chceme všechna pravidla, která jsou dynamicky vytvořená, pro příchozí provoz a veřejnou síť

`add rule name="povolit udp 5000" dir=out protocol=udp localport=5000 action=allow` povolili jsme odchozí komunikaci na UDP portu 5000

`add rule name="šifrovat 80" protocol=TCP dir=in localport=80 security=authdynenc action=allow` takto jsme si vynutili šifrování příchozí komunikace na TCP portu 80 (o jaký port se asi tak jedná?)

Příklad

Pro notebook, který přenášíme mezi umístěními se statickou a dynamickou IP adresou, si můžeme vytvořit dva zástupce nebo dávkové soubory. Nejdřív si v nápovědě ověříme formát příkazu:

```
netsh -c interface ip set ?
netsh -c interface ip set address ?
```

Teď už víme, které parametry a v jakém pořadí máme zadat, tedy vytvoříme zástupce nebo dávkové soubory (podle potřeby) s následujícími příkazy. První z nich je určen ke spuštění na umístění se statickou adresou:

```
netsh -c interface ip set address name="Připojení místni sítě"
    source=static addr=123.456.0.5 mask=255.255.0.0 gateway=123.456.0.1
    gwmetric=1
```

Celý příkaz by měl být na jednom řádku. Takto nastavíme statickou IP adresu, masku podsítě, bránu, doplníme vlastní skutečné údaje.

Další je pro umístění, na kterém používáme dynamickou adresu získanou od DHCP serveru:

```
netsh interface ip set address name="Připojení místni sítě" source=dhcp
```

Nastavíme dynamickou IP adresu, která bude získávána z DHCP.

Vytvoříme dva zástupce na pracovní ploše – jednoho pro statickou adresu a druhého pro dynamickou adresu, do příkazového řádku zástupců napíšeme výše uvedené příkazy a oba zástupce vhodně pojmenujeme. Pak stačí jen poklepat na jednoho ze zástupců.

Prostředí NetShell je svými možnostmi přizpůsobeno práci v síti. Počítá se přirozeně i s tím, že příkaz (prostředí) budeme chtít využít ke vzdálené správě počítačů. Přesun na jiný počítač provedeme buď při spouštění příkazu *netsh* přepínačem *-r*, anebo přímo v interaktivním prostředí příkazem *set*:

`netsh -r počítač` umožní pracovat v interaktivním prostředí vzdáleně na jiném počítači

`netsh -r počítač příkaz` tento přepínač můžeme použít v neinteraktivní formě příkazu, když chceme provést jednorázovou akci na jiném počítači (přidáme jakýkoliv platný příkaz NetShellu)

`set machine počítač` použité v interaktivním režimu v prostředí NetShellu slouží k „přesouvání“ mezi počítači

`set machine` pokud nezadáme cílový počítač, přesuneme se na náš, lokální, počítač

Úkoly

1. Spusťte prostředí NetShell a vyzkoušejte z příkladu na straně 134 vše, co půjde (na co máte přístupová oprávnění). Ke každému příkazu, který vyzkoušíte, zobrazte nápovědu, abyste měli přehled o možnostech příkazu.
2. Vypište seznam cílů ve směrovací tabulce vašeho počítače.
3. Vypište tabulku existujících spojení přes protokol TCP.
4. Přesuňte se do kteréhokoliv kontextu (raději hlouběji ve struktuře kontextů) a vypište jeho konfiguraci. Potom opusťte prostředí NetShell a proveďte totéž, výstup přesměrujte do souboru `vypisnetsh.dmp` (umístěte tam, kde máte právo zápisu). Tento soubor si pak prohlédněte v některém textovém editoru.
5. Vypište konfiguraci firewallu. Zjistěte, které aplikace mají povoleno pracovat se sítí, které služby naslouchají na portech a které porty jsou otevřené.

   Můžete vyzkoušet výpis čehokoliv, co pro tento kontext lze vypsat. U otevřených portů a povolených aplikací se tytéž informace pokuste zjistit i v grafickém prostředí.

Souhrn kapitoly 4

V této kapitole jsme se zabývali správou zařízení a sítě ve Windows. Nejdřív jsme se zaměřili na ovladače, naučili jsme se vypsat informace o ovladačích, také zjistit, jestli jsou všechny ovladače podepsané. Následovala sekce o paměťových médiích, kde jsme se nejdřív naučili synchronizovat a zálohovat úložiště, seznámili jsme se s diskovými kvótami, naučili jsme se zkontrolovat stav disku, pracovat s oddíly na disku, vytvořit virtuální disky a připojit oddíl do složky. Také jsme se seznámili s programem `fsutil`, zejména s jeho možnostmi pro zjišťování informací o souborových systémech. Také jsme zjistili, že v souborech mohou existovat alternativní datové streamy. K operační paměti jsme si zopakovali, jak je to se stránkovacím souborem.

V poslední sekci kapitoly jsme se zaměřili na správu sítě. Naučili jsme se používat základní příkazy pro tyto účely, také jsme se naučili konfigurovat sdílení prostředků a seznámili jsme se s NetShellem.

Kapitola 5

Windows: nasazení systému

Rychlý náhled: V této kapitole se zaměříme na pokročilejší témata jako je správa registru, průběh startu systému a zobrazení startovací nabídky s možnostmi opravy problémů Windows, podíváme se na instalaci Windows včetně bezobslužné instalace. Budeme se také zabývat aktualizací systému, různými chybami při běhu systému a pokročilou správou softwaru.

Klíčová slova: Registr, start systému, nouzový režim, Windows Recovery Environment, msconfig, BCDEdit, Boot Manager, Boot Loader, Media Creation Tool, bezobslužná instalace, soubor odpovědí, wim, Windows ADK, Windows PE, WSIM, WDS (Windows Deployment Service), aktivace, Product Key, Product ID, aktualizace, WSUS (Windows Server Update Services), BSOD, služba Windows Installer, přiřazení softwaru počítači/uživateli, publikování softwaru, System Center Configuration Manager (ConfMgr, SCCM), balíček MSI, WinGet.

Cíle studia: V této kapitole se naučíme přistupovat k obsahu registru Windows i v textovém režimu, naučíme se řídit stav systému včetně přechodu do nouzového režimu a využití prostředí Windows Recovery Environment a podrobně se seznámíme se samotným průběhem startu systému. Pak se seznámíme s možnostmi bezobslužné instalace Windows, nástroji používanými pro tyto účely, také Službou pro nasazení systému Windows (WDS) a principem, na jakém je založena aktivace systému. Naučíme se, jak se dostat ze zamrzlého celoobrazovkového módu a jak reagovat při různých typech chyb při běhu systému. Naučíme se, jak vlastně probíhá instalace aplikací ve Windows, a které nástroje se k ní používají, včetně možností automatizace a přístupu pomocí nástroje WinGet.

5.1 Registr

V této podsekci shrneme a rozšíříme naše dosavadní znalosti o registru Windows (v angličtině Windows Registry). Z předchozího semestru víme, že rozlišujeme několik pojmů:

- *větev registru* – začíná vždy některým z hlavních klíčů HKLM (Local Machine), HKU (Users), HKCU (Current User), HKCR (Classes Root), HKCC (Current Config),
- *podstrom registru* – dva z hlavních klíčů jsou na počítači unikátní (to jsou podstromy) – HKLM a HKU, ostatní hlavní klíče pouze odkazují na některý z podklíčů těchto podstromů,

- *podregistr* (hive, úl) – tento pojem se vztahuje k souborům, ve kterých je registr uložen:
  - několik systémových podregistrů s názvy `sam`, `security`, `software`, `system`, všechny jsou uloženy v `...\system32\config`, jde o podklíče v klíči *HKLM*,
  - pro každého uživatele jeden s názvem `ntuser.dat` uložený v profilu uživatele, tyto podklíče jsou v klíči *HKU*, resp. *HKCU* pro právě přihlášeného uživatele,
  - další pomocné podregistry, především *HKU/.Default* (tento podregistr najdeme ve stejném adresáři jako systémové podregistry, soubor `default`), a dále klíče *HKU/⟨SID⟩_Classes* (tento podregistr je v souboru `%userprofile%/Local Settings/Data Aplikací/Microsoft Windows/usrClass.dat`, kde podřetězec `%userprofile%` je cesta k profilu uživatele s daným SID).

Poznámka

V klíči *HKLM* je ještě jeden podklíč – *hardware*. Pro tento podklíč však neexistuje žádný podregistr, protože všechny údaje, které v něm najdeme, jsou dynamicky generovány při každém startu systému a proto nejsou nikde uloženy.

Víme, že ve Windows řady NT se konfigurace systému a aplikací obvykle ukládá do registru. Protože se jedná o binární soubory, je zpracování údajů registru pro procesy o něco náročnější, nelze použít přímou úpravu těchto souborů. Shrňme si, jaké máme možnosti:

- program `regedit.exe` – pouze pro uživatele „klepajícího“ myší u počítače, i vzdáleného,
- soubory s příponou REG – lze jen buď exportovat z výše zmíněných programů, anebo načíst do registru nová data, ale pro proces není možné tímto způsobem data z registru získat,
- příslušné API funkce anebo funkce, třídy, objekty v programovacích nástrojích (pro procesy),
- program `reg.exe` (pro uživatele i procesy),
- další.

Teď nás bude zajímat program `reg.exe`. Tento program je určen k běhu v Příkazovém řádku a s jistými omezeními ho lze použít i procesem, třebaže není vhodný pro skriptování. Má syntaxi podobnou příkazu `net`, ale množství variant je výrazně menší. Probereme postupně jednotlivé varianty.

`reg query`

takto získáváme údaje z registru; zadáváme vždy větev registru, která nás zajímá (větve začínají některým z řetězců *HKLM*, *HKU*, *HKCU*, *HKCC*, *HKCR*), můžeme zadat také větev ze vzdáleného registru (na jiném počítači), řetězec obsahující mezery uzavřeme do uvozovek

`reg query "HKLM\system\currentcontrolset\Control\Session Manager"`
vypíšou se všechny záznamy (položky) zadaného klíče a všechny jeho podklíče

`reg query "HKCU\Control panel\desktop /v autoendtasks` přepínač `/v` vypíše momentální nastavení hodnoty *AutoEndTasks* v zadaném klíči

`reg query "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\ Memory Management" /v clearpagefileatshutdown` takto zjistíme momentální nastavení hodnoty (položky) *ClearPageFileAtShutdown* v zadaném klíči

`reg query HKCU\Software\OpenOffice.org /s` vypíšou se všechny údaje, a to rekurzívně i v podklíčích

`reg query "\\pocitac\HKU"` takto přistupujeme ke vzdálenému registru (před název hlavního klíče dáme název počítače podle UNC); na vzdálených počítačích můžeme přistupovat pouze k podstromům registru, tj. ke klíčům *HKLM* a *HKU* (a jejich podklíčům), nelze zadat např. *HKCU*

`reg copy`

kopíruje obsah jednoho klíče do druhého (spíše pro kopírování mezi dvěma různými počítači)

`reg copy \\RefPocitac\HKLM\software \\JinyPocit\HKLM\software /s /F`
: zkopíruje celou softwarovou konfiguraci (obecně) z jednoho počítače na jiný (pozor, pokud chceme tímto způsobem přenášet konfiguraci mezi počítači, musíme toho přenést poněkud více), a to rekurzívně (`/s`) a bez požadavku na potvrzování kopírování (`/F`)

Pokud nezadáme UNC počítače, pracuje se s větví registru lokálního počítače.

`reg delete`

odstraní z registru podklíč nebo položku

`reg delete HKCR\nejaky-klic` odstraní zadaný podklíč

`reg delete "HKLM\software\program /v polozka` odstraní uvedenou položku v zadaném klíči

`reg delete "HKLM\software\program /ve` v zadaném klíči odstraní všechny položky, které nemají vloženu žádnou hodnotu (empty)

`reg delete "HKLM\software\program /va` v zadaném klíči odstraní všechny položky (ale podklíče nechá)

`reg export, reg import`

slouží k exportu větve do souboru s příponou REG, resp. importu z REG souboru do registru

`reg export "HKCU\AppEvents\EventLabels d:\uzivatelUdalosti.reg` exportujeme zadanou větev registru do REG souboru

`reg import uzivatelUdalosti.reg` import zadaného REG souboru

Příkaz `reg` má ještě další parametry. Kromě běžných větví registru můžeme také zvlášť pracovat s podregistry. Podregistr můžeme dočasně připojit, pak znovu odpojit nebo uložit podregistr do souboru s příponou HIV (zkratka z „hive“), připojovat můžeme pouze pod klíče *HKLM* a *HKU*. Pro tyto účely slouží podpříkazy `save`, `load`, atd.

Poznámka

Používání příkazu `reg` přímo na Příkazovém řádku je poměrně jednoduché (když neznáme syntaxi, stačí použít přepínač `/?`, stejně jako u kteréhokoliv jiného příkazu), horší je to ve skriptu (dávkovém souboru) nebo v procesu. Příkaz totiž vypisuje poněkud hodně informací (nejen to, co chceme). Na začátku výpisu se vždy zobrazí řádek

`! REG.EXE VERSION xxx` (zobrazí se číslo verze souboru)

a zbytek výpisu je prokládán prázdnými řádky. Při automatickém zpracování tedy musíme přeskočit řádky začínající symbolem „`!`“ a prázdné řádky, a dále je třeba rozpoznat názvy klíčů od názvů položek

Pokud nás zajímá, které podregistry jsou kde uloženy (a obvykle takto také můžeme zjistit přiřazení SID konkrétnímu uživateli), podíváme se do klíče *HKLM/SYSTEM/CurrentControlSet/Control/hivelist*, ve kterém jsou položky nazvané podle objektů registru, jejichž hodnotou je umístění daného souboru.

Úkoly

1. V nápovědě příkazu `reg` zjistěte, jak konkrétně se dá pracovat s podregistry.
2. Vyberte si kteroukoliv větev registru a vyzkoušejte na ní příkazy `reg query` (včetně přepínačů) a `reg export`.

3. Vlastníkem načtených podregistrů (tedy souborů `system`, `sam`, atd.) je jádro. Ve struktuře procesů zobrazených v aplikaci Process Explorer je jádro představováno řádkem *System*. Zobrazte seznam manipulátorů vlastněných tímto procesem (handlů) a najděte mezi nimi podregistry (jedná se o objekty typu soubor, *file*).

5.2 Start systému

5.2.1 Možnosti spuštění Windows

Windows obvykle spouštíme běžným způsobem (tj. do paměti se natáhne jádro a pak paralelně různé moduly – ovladače, antivirus apod., o chvíli později dostaneme možnost se přihlásit). Nicméně někdy potřebujeme spustit Windows trochu jinak, typicky v nouzovém režimu.

Ve Windows s NT jádrem startovací menu (startup settings) nabízí tyto možnosti:

Nouzový režim (Safe Mode): Po načtení jádra se načtou pouze základní obecné ovladače pro nejdůležitější zařízení (monitor, myš, klávesnice, některé filtry, apod.) a jenom nejdůležitější systémové služby. Síťové rozhraní není v provozu.

Nouzový režim se sítí (Safe Mode with Networking): Oproti předchozímu se načtou i standardní ovladače pro síťové rozhraní a síťové protokoly, takže na síť se dostaneme.

Nouzový režim s příkazovým řádkem (Safe Mode with Command Prompt): Neaktivuje se grafické rozhraní, není spuštěn proces explorer.exe – žádná okna, žádné nástroje závislé na GUI (včetně konzol typu Správa počítače). Pokud je v GUI problém se stabilitou, kvůli kterému nelze systém reálně spustit, v tomto režimu problém odpadá.

Spustit s nízkým rozlišením (Enable low-resolution video): Pokud selhávají ovladače grafických zařízení (hlavně grafické karty a monitoru), můžeme použít tento režim. Použijí se pouze standardní ovladače grafických zařízení (které neumějí vysoké rozlišení). GUI bude (včetně oken a všech nástrojů), rozdíl bude jen v rozlišení a barevné paletě. To stačí k tomu, abychom dokázali například odinstalovat a znovu nainstalovat příslušné ovladače.

Povolit protokolování při spuštění (Enable boot logging): Během startu systému se vytvoří soubor `ntbtlog.txt` (ve složce Windows) se seznamem modulů načtených do jádra (včetně informace o úspěšnosti), většinou jde o ovladače. Pokud máme podezření na problém s konkrétním ovladačem, v tomto souboru si ho můžeme ověřit.

Zakázat vynucení podpisu ovladače (Disable driver signature enforcement): V 64bitových Windows lze používat jen podepsané ovladače, ale tuto podmínku lze dočasně zmírnit. Pokud je například poškozen některý certifikát, tato volba může problém odhalit (ale jednodušší by bylo prostě použít nástroj pro kontrolu podpisu ovladačů).

Zakázat automatické restartování při selhání systému (Disable automatic restart after failure): Některé chyby běhu systému se projevují tak, že se systém okamžitě restartuje. Restartování opravdu některé problémy napraví, ale v jiných případech k nápravě nedojde, důsledkem pak je cyklické restartování Windows, do kterého se jen těžko dá zasáhnout a zjistit příčinu.

Zakázat brzy spuštěnou antivirovou ochranu (Disable Early launch anti-malware protection): Někdy může problém způsobit modul antiviru v jádře (podobně jako ostatní ovladače). Tato možnost slouží právě k otestování této možnosti. Windows obvykle dokážou odlišit antivirus od ostatních

modulů jádra (certifikovaný antivirový modul se načítá jako první z modulů „třetích stran“), takže další ovladače by tento režim postihnout neměl.

Výběr provádíme podle toho, jak nám samotné Windows na dané obrazovce poradí v závislosti na konkrétní verzi – buď kurzorovými klávesami nebo stisknutím příslušné klávesy F1 , F2 , . . . , v některých verzích musíme také potvrdit klávesou Enter .

5.2.2 Jak se dostat ke startovací nabídce

Ve Windows 7 a starších před načtením jádra (tj. hned po zobrazení hlášení BIOSu, ještě před začátkem bootování) stiskneme klávesu F8 . Pokud nepoznáme tu správnou chvíli, je dobré mačkat tuto klávesu hned od startu počítače v intervalech cca jedné sekundy, nebo ji prostě držet stisknutou.

Ve vyšších verzích Windows je složitější se k startovací nabídce dostat. Může fungovat Shift+F8 nebo původní F8 (pokud by šlo o zařízení se starším BIOSem). Vzhledem k tomu, že na většině zařízení je UEFI, nebude tato možnost fungovat.

Může pomoci zadat na příkazovém řádku s oprávněními správce příkaz

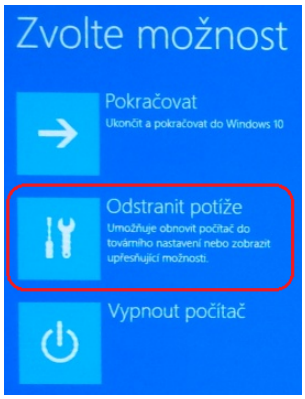
```
bcdedit /set {default} bootmenupolicy legacy
```

(včetně složených závorek), pak by už měla klávesa F8 fungovat. Ovšem interval pro její použití je tak krátký, že je problém to stihnout.

V jakékoliv verzi Windows si můžeme vynutit po restartu přechod do nouzového režimu v nástroji *Konfigurace systému* (`msconfig.exe`), kde na záložce *Spuštění počítače* vybereme *Bezpečné spuštění* a určíme, který režim se má použít, případně na dalších záložkách můžeme určit, které služby nebo programy po spuštění mají nebo nemají být spuštěny.

Jinak se k startovací nabídce dostaneme přes prostředí *Windows Recovery Environment*, jehož „okno“ vidíme na obrázku 5.1.

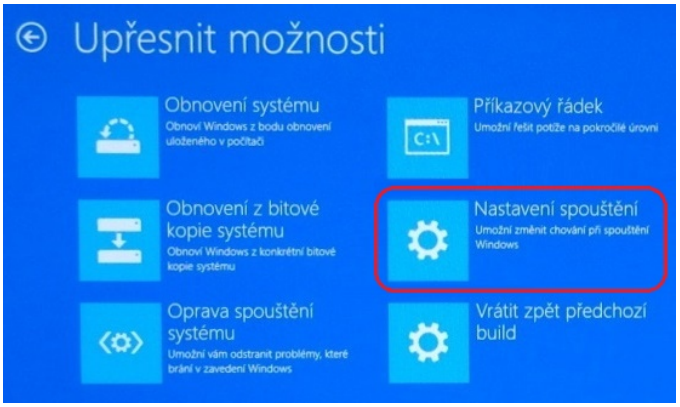
Zvolíme *Odstranit potíže* (v anglické variantě *Troubleshoot*), pak *Upřesnit možnosti* (v anglické variantě *Advanced Options*), viz obrázek 5.2. Tam je volba *Nastavení Spouštění* (resp. *Startup Settings*), která nás zavede na obrazovku s volbami typu Povolit/Zakázat (například *Povolit Nouzový režim*). Povolíme či zakážeme položky dle vlastního výběru, klepneme na tlačítko *Restartovat*, čímž se konečně (po restartu) objeví startovací nabídka – pokud jsme ovšem v předchozím kroku něco vhodného vybrali.



Obrázek 5.1: Windows Recovery Environment

Teď sice víme, jak se z nástroje *Windows Recovery Environment* dostat ke startovací nabídce, ale jak se dostat k tomu nástroji?

- Přes tlačítko *Start* se dá (někde, v závislosti na verzi) dostat k možnosti restartu Windows – obvykle v kontextovém menu tohoto tlačítka najdeme volbu pro vypnutí nebo odhlášení. Stiskneme a držíme klávesu Shift a zároveň zvolíme restart. Před začátkem bootování by se měla objevit obrazovka s nabídkou jako na obrázku 5.1.
- Přes tlačítko *Start* spustíme nástroj *Nastavení*, tam vybereme položku *Aktualizace a zabezpečení*, vlevo *Obnovení*, možnost *Spuštění s upřesněným nastavením*, *Restartovat*.
- Pokud máme instalační médium Windows (třeba DVD), nabootujeme z tohoto média. Případně pokud máme vytvořen disk pro obnovení systému, můžeme ho použít i pro tento účel.



Obrázek 5.2: Nabídka *Upřesnit možnosti* ve Windows Recovery Environment

Postup

Někdy může nastat opačný problém – po restartu z nouzového režimu či startovací nabídky by správně měl naběhnout operační systém, ale co když se zase dostaneme do startovací nabídky? Pak postupujeme takto:

- stiskneme ⊞+R, abychom mohli zadat textový příkaz,
- zadáme `msconfig`,
- v okně přejdeme na záložku *Spuštění počítače* (Boot) a změníme volby (především zrušíme zatržení položky *Bezpečné spuštění*).

5.2.3 Start systému a registr

Informace o startu systému jsou v registru. Na předchozích stranách je zmíněn nouzový režim a nouzový režim se sítí. V klíči `HKLM/System/CurrentControlSet/Control/SafeBoot` máme dva podklíče:

- `Minimal` obsahuje seznam ovladačů a služeb (ve formě podklíčů), které se mají načíst v případě, že systém nastartuje v nouzovém režimu,
- `Network` obdobně seznam ovladačů a služeb pro nouzový režim se sítí.

V klíči `HKLM/System/Select` najdeme několik důležitých položek:

- `Current` obsahuje číslo momentální platné konfigurace; obvykle je zde číslo 1, což znamená, že platná konfigurace je v klíči `HKLM/System/ControlSet001`, do tohoto klíče tedy odkazuje klíč s momentálně platnou konfigurací `HKLM/System/CurrentControlSet`,
- `Default` označuje výchozí volbu, obvykle obsahuje stejné číslo jako `Current`,
- `LastKnownGood` obsahuje číslo poslední známé platné konfigurace (před změnami, které mohly způsobit problémy), například pokud je zde číslo 2, poslední známá platná konfigurace je v klíči `HKLM/System/ControlSet002`,
- `Failed` obsahuje číslo konfigurace, jejíž načtení se nezdařilo (pokud je zde 0, znamená to, že žádné problémy se nevyskytly).

Do verze Windows 7 byla ve startovacím menu také volba *Poslední známá funkční konfigurace*, která vrátila systém do konfigurace zaznamenané v klíči `LastKnownGood`.

5.2.4 Jak startují Windows

Průběh startu Windows se liší podle konkrétní verze. My se zaměříme na postup platný v novějších systémech (Windows Vista, 7, 8, 10, 11).

Pro načtení a inicializaci systému se používá dvojice programů:

- *Windows Boot Manager* (soubor `bootmgr`) – správce zavádění systémů (zobrazuje nabídku zjištěných operačních systémů, abychom si mezi nimi mohli vybrat),
- *Windows Loader* (soubor `winload.exe`) – provede vlastní zavedení systému.

Start systému se dá ovlivnit těmito nástroji:

- použitím *Windows Recovery Environment* včetně startovacího menu a nouzového režimu, viz výše,
- v nástroji *Konfigurace systému* (`msconfig.exe`),
- programově a ve skriptech přes rozhraní WMI,
- použitím *BCDEdit*.

Program *BCDEdit* (`bcdedit.exe`) slouží ke konfiguraci startu Windows od verze Vista SP1 výše, vpodstatě nahrazuje dřívější konfiguraci přes soubor `boot.ini`. Vyžaduje oprávnění správce.

Slouží ke konfiguraci boot manažera a jako frontend k *úložištím BCD* (Boot Configuration Data Stores). V každém takovém úložišti je uložena kompletní konfigurace pro některý způsob spuštění (některých) Windows, a vše je napojeno na rozhraní WMI.

Pokud chceme konfigurovat boot manažera v multiboot prostředí, máme možnost například určovat bootovací sekvenci, pokud se má zobrazovat seznam s výběrem operačních systémů, tak v jakém budou pořadí, který má být výchozí a jak dlouho se má počkat, než bude načtena výchozí volba (timeout), případně možnost přidání dalších nástrojů do seznamu (třeba nástroj pro kontrolu operační paměti).

Postup

Ukážeme si, jak u programu *BCDEdit* získat nápovědu a pár dalších úloh.

`bcdedit` (bez parametrů) zobrazí momentální obsah BCD úložišť a bootovací konfiguraci,

`bcdedit /?` vypíše se nápověda příkazu – seznam možných základních parametrů,

`bcdedit /? /bootsequence` podrobnosti k parametru `/bootsequence` ovlivňujícímu bootovací sekvenci v případě více instalovaných operačních systémů,

`bcdedit /timeout 8` v případě více instalovaných operačních systémů tímto nastavíme dobu čekání na volbu systému ze seznamu na 8 sekund,

`bcdedit /displayorder {current} /addfirst` pokud je více instalovaných operačních systémů a zobrazuje se boot menu s jejich seznamem, tímto jsme určili, že operační systém, který je označen jako `current` (to bychom zjistili z výpisu příkazu bez parametrů) bude v boot menu jako první a bude výchozím systémem (když během doby čekání nezvolíme jiný, spustí se tento).

Další informace

- https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/bcdedit-command-line-options?view=windows-11
- https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/bcd-boot-options-reference

Program *BCDEdit* není zrovna „uživatelsky přítulný“. Aplikace *EasyBCD*[1] od firmy NeoSmart Technologies sloužící ke stejnému účelu má přehledné grafické rozhraní a je volně ke stažení. Umožňuje dokonce snadné zařazení unixových operačních systémů do nabídky Windows Boot Managera.

Úkoly

1. Najděte na systémovém disku všechny soubory, které jsou v této sekci zmíněny (podle verze Windows – například buď `ntldr` nebo `bootmgr` plus `winload.exe`). Některé budou v kořenové složce systémového disku, další například u Windows 7 a vyšších v `windows\boot`.
2. Na internetu najděte stránky s popisem aplikace EasyBCD a zjistěte, co vše v ní lze nastavovat.
3. V registru najděte klíč `HKLM/System/Select` a prohlédněte si momentální hodnoty položek.

5.3 Instalace Windows

5.3.1 Obecně o instalaci

Rozlišujeme několik základních typů instalací Windows:

Čistá instalace: instalujeme na prázdný zformátovaný logický disk (budoucí systémový disk) nebo provedeme formátování na začátku instalace, instalace probíhá bez jakékoliv vazby na předchozí operační systém, který byl případně na daném logickém disku nainstalován.

Inovace (přeinstalování): na budoucím systémovém disku je již nainstalován některý operační systém typu Windows, chceme ho přeinstalovat na vyšší verzi. Když provádíme opravdu pouze upgrade na novou verzi, můžeme se setkat s problémy (některé části starší verze nebudou správně upgradovány), v takových případech je lepší provést čistou instalaci.

Za speciální typ inovace můžeme považovat také restart konfigurace nástrojem *Obnovení počítače do továrního nastavení* dostupný od Windows 8. Ve Windows 10 a vyšších se provádí v nástroji *Nastavení* ⇨ *Obnovení*. Různé verze Windows také nabízejí možnost vytvoření záchranného CD/DVD pro obnovení systému. Také můžeme vytvořit bitovou kopii systému (tj. vpodstatě obraz systémového oddílu).

Možnosti instalace na desktopu:

Běžná instalace: instalujeme z instalačního média (DVD), ze sítě nebo jiným způsobem, v každém případě sedíme u počítače a přímo zadáváme údaje.

Bezobslužná instalace: použijeme ji, pokud máme instalovat systém na větší množství počítačů a nemíníme u každého během instalace sedět, ručně zadávat údaje a hlídat instalační proces.

Bezobslužná (automatizovaná) instalace probíhá s minimem zásahů uživatele (správce), pro vstupy se používají *soubory odpovědí* (answer file, to jsou textové soubory s informacemi, které by bylo jinak nutné zadávat během instalace, případně tady mohou být příkazy pro spuštění určitých programů včetně instalačních a konfiguračních nástrojů). V novějších verzích Windows je soubor odpovědí ve formátu XML. Soubory odpovědí můžeme vytvářet buď ručně, anebo s využitím specializovaného programu (taky záleží na tom, co je určeno pro konkrétní verzi Windows).

[1] EasyBCD je ke stažení na http://neosmart.net/dl.php?id=1.

Postup

Windows v některých edicích jsou ve skutečnosti volně ke stažení. To, že je můžeme používat (a také kterou edici) se určuje během aktivace pomocí obdoby licenčního klíče – Product Key (viz sekci o aktivaci).

Hlavním přístupovým bodem je web https://www.microsoft.com/cs-cz/software-download, resp. můžeme se zeptat Googlu – „windows download“. Po výběru verze systému máme tři možnosti:

1. Pomocník s instalací: použijeme, pokud potřebujeme provést jen upgrade (například z Windows 10), přičemž se nacházíme na tomtéž počítači,
2. stažení nástroje *Media Creation Tool* pro vytvoření instalačního média (DVD nebo USB flash disku): použijeme, pokud potřebujeme instalační médium, například když budeme instalovat na jiný počítač než na kterém jsme,
3. stažení bitové kopie (ISO souboru): pokud budeme chtít s ISO souborem dále pracovat, například ho upravit pomocí nástroje Rufus.

Windows jsou od verze Vista výše distribuovány na DVD (u starších verzí bylo CD) ve formátu *WIM* (Windows Imaging). Přesněji – na paměťovém médiu může být víc souborů s příponou WIM, třeba pro různé hardwarové platformy nebo předpřipravené varianty s různými edicemi či přidanými aplikacemi.

Vnitřní struktura připomíná ZIP archiv. Uvnitř WIM souboru (třeba `install.wim`) mohou být ve „složkách“ různé edice dané platformy, v XML souboru pak jejich popis. V jednotlivých složkách jsou komponenty systému, také určitým způsobem členěné. Pokud se má komponenta nainstalovat, průběh opět připomíná rozbalení ZIP archivu na dané místo (včetně konfigurace, která je taktéž popsána v příslušném XML souboru). Během instalace se zvolí příslušná složka a z ní konkrétní komponenty.

Soubory s příponou WIM se dají modifikovat, pokud máme k ruce příslušné nástroje. Především jde o nástroj `dism`, jehož účelem je řešit problémy s instalačními daty nebo upravovat či obohacovat WIM soubory.

Další informace

- https://www.optimalizovane-it.cz/deployment/windows-deployment-zaklady.html
- https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/mount-and-modify-a-windows-image-using-dism?view=windows-11

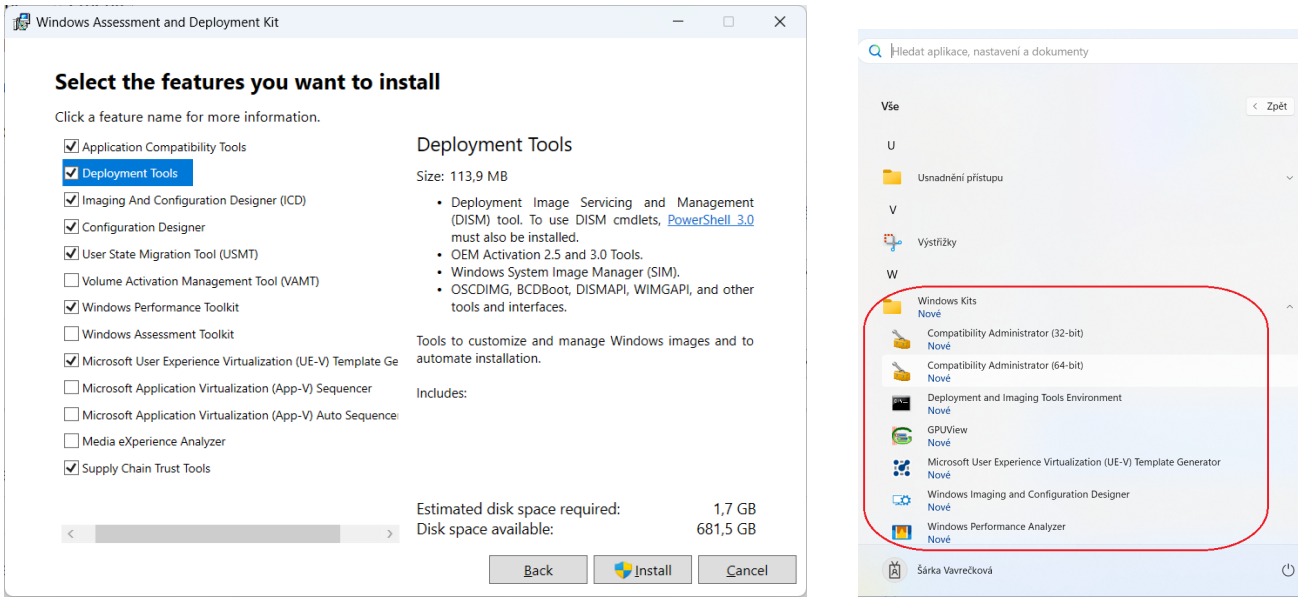
5.3.2 Nástroje k řízení instalace a oprav systému

Windows ADK (Assessment and Development Kit, také WADK) je sada nástrojů k úpravám, nasazení a optimalizování (např. co se týče výkonu) systémů Windows. Zde obsažené nástroje nám pomáhají i s bezobslužnou instalací.

Na rozdíl od svého předchůdce AIK (Automated Installation Kit) má trochu širší využití a navíc se mnohem snadněji instaluje a používá (AIK se stahoval ve formě ISO obrazu, musel se vypálit na CD nebo připojit jako virtuální mechanika a pak instalovat instalovat). WADK si jednoduše stáhneme z tohoto odkazu:

https://learn.microsoft.com/en-us/windows-hardware/get-started/adk-install

Jen musíme dát pozor, ať vybereme správnou variantu, jsou různé pro x64 a arm64. Během instalace si volíme, které součásti chceme instalovat (viz obrázek 5.3).



Obrázek 5.3: Volba nástrojů při instalaci Windows ADK

Nástroje nainstalované v balíku WADK najdeme v menu Start v položce *Windows Kits*. Po instalaci není nutno restartovat. Další nástroje se do WADK dají přidat ve formě modulů, například nástroj pro tvorbu Windows PE.

DISM (Deployment Image Servicing and Management) je nástroj pro příkazový řádek sloužící k vytváření, opravám nebo úpravám instalačních obrazů Windows, virtuálních disků (VHD, VHDX) s Windows nebo i aktuálně nainstalovaného systému. Systém, který právě běží, je *online*, ktežto instalační obraz (typicky .WIM soubor) je *offline*. Tento nástroj vždy spouštíme v příkazovém řádku s oprávněním správce.

Dism bývá součástí Windows, nebo kdyby náhodou ne, doinstaluje se s WADK (viz výše). Jak víme, na instalačních médiích vlastně máme WIM soubory: nástroj DISM umí vytvořit WIM soubor a upravovat ho.

Příklad

Pokud máme vytvořený obraz v souboru `c:\testovaci\obraz\install.wim` a chceme ho upravit, nejdřív ho připojíme:

```
dism /Mount-image /imagefile:c:\testovaci\obraz\install.wim /Index:1 /MountDir:c:\testovaci\pripojeny
```

Teď můžeme provést potřebné úpravy, například přidat ovladač (musíme ho mít stažený na tentýž počítač a zadáváme jeho .INF soubor):

```
dism /Image:c:\testovaci\pripojeny /Add-Driver /Driver:C:\ovladace\xxxx.inf
```

Pokud pomocí DISM upravujeme instalační obraz (který jsme před úpravou připojili jako virtuální jednotku), po úpravě je třeba odpojit tento obraz s potvrzením (volba `/commit`) nebo odmítnutím (volba `/discard`) změn.

```
dism /unmount-image /mountDir:c:\testovaci\pripojeny /commit
```

Pokud se zaměříme na úpravy existující instalace Windows (na počítači, kde zrovna pracujeme), pak se použít například tento příkaz pro hledání problémů v systému:

```
dism /online /cleanup-image /scanhealth
```

Tento příkaz pouze prověří, jestli jsou v systému problémy, například poškozené systémové soubory (je to o něco důkladnější sken než pomocí `sfc /scannow`). Pokud příkaz najde problémy, takto zjistíme, jestli se dají opravit:

```
dism /online /cleanup-image /checkhealth
```

Jestli je oprava možná, spustíme ji takto:

```
dism /online /cleanup-image /restorehealth
```

Pro opravu můžeme zadat také zdroj, ze kterého se mají brát nepoškozené soubory, to může být jiný připojený obraz. Pak bychom přidali parametr `/source:` a dodali bychom přípojný bod tohoto zdroje. Pokud neuvedeme zdroj, bude zdrojem Windows Update, a pak bychom měli mít zajištěno kvalitní připojení na síť, jinak bychom mohli vyrobit další problémy.

Takto můžeme opravit i offline připojený obraz, nemusí jít o aktuálně běžící (online) systém.

Další informace

- https://www.techtarget.com/searchenterprisedesktop/definition/Microsoft-Windows-Deployment-Image-Servicing-and-Management-DISM
- https://learn.microsoft.com/en-us/windows/deployment/windows-adk-scenarios-for-it-pros
- https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/repair-a-windows-image?view=windows-11
- https://matthewhard.com/windows-command-line-dism

Pokud chceme nasadit systém na počítači (nebo více počítačích) podle jednoho, na kterém už jsme provedli plnou instalaci včetně dalšího softwaru, pak můžeme z toho předinstalovaného stroje sejmout obraz, generalizovat (tedy zbavit „specifických“ nastavení platných jen pro ten dotyčný stroj) a pak tento obraz přenést na další stroje.

Příklad

Zachytit obraz systému (toho, na kterém zadáváme příkaz) můžeme takto:

```
dism /capture-image /imagefile:c:\cesta\obrazWindows.wim /capturedir:c:\ /name:"disk-c"
```

Bude to poměrně velký soubor, ale lze přidat i parametr pro kompresi, pokud to bude nutné. Lze ho brát jako zálohu disku, pak bychom ho obnovili takto:

```
dism /apply-Image /imagefile:c:\cesta\obrazWindows.wim /name:"disk-c" /applydir:C:\
```

Pro jistotu lze použít také parametr `/CheckIntegrity`.

Nebo zachycený obraz použijeme jako referenční pro další instalace, a tedy nejdřív generalizujeme:

```
dism /generalize /imagefile:c:\cesta\obrazWindows.wim
```

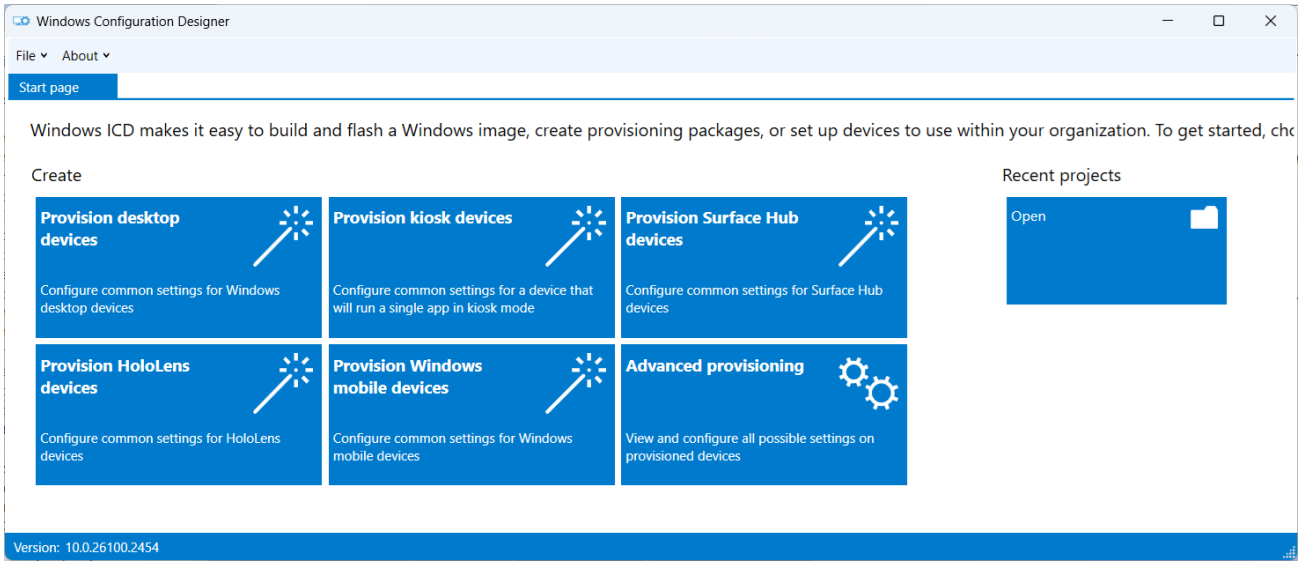
Pak tento obraz můžeme použít pro nasazení na jiná zařízení. Tam je třeba vytvořit oddíl(-y) a nakopírovat obraz. Samozřejmě bychom museli buď vhodně zpřístupnit zařízení přes síť, nebo nabootovat pomocí Windows PE. Celý návod včetně ukázkových skriptů je zde:

https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/capture-and-apply-windows-using-a-single-wim?view=windows-11

(doporučuji zrušit automatický překlad z angličtiny, je nepoužitelný). Nebo zde:

https://www.tenforums.com/general-support/43132-using-dism-capture-apply-windows-10-images-partitions.html

Výše jsme se seznámili s nástrojem WADK (Windows ADK). Nástroje v něm obsažené nám mohou zjednodušit výše uvedené úlohy, mohou sloužit jako frontend k nástroji DISM. Například Imaging and Configuration Designed (ICD) pomáhá s přípravou WIM obrazu.



Obrázek 5.4: Nástroj Imaging and Configuration Designer z balíku WADK

Na obrázku 5.4 je úvodní obrazovka tohoto nástroje, při instalaci WADK jsme měli tento nástroj zatrhnutý (na obrázku 5.3 to byla třetí položka).

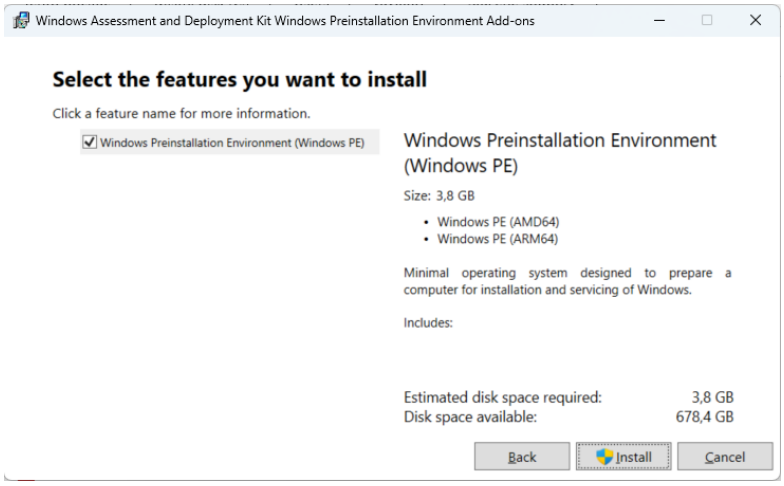
Windows PE je odlehčená bootovací varianta Windows určená pro různé opravy systému (takového, který třeba nelze spustit, z bootovacího média se k tomu systému dostaneme), také je lze použít v případě, že na zařízení chceme nainstalovat systém z obrazu získaného výše naznačeným způsobem pomocí DISM nebo ICD.

Windows PE si můžeme vytvořit sami. Obvyklá cesta je instalace WADK, a následně do této sady nástrojů přidáme modul (tedy další nástroj) použitelný právě pro vytvoření Windows PE. Na stejném webu, kde jsme si stáhli WADK, je ke stažení i modul Windows PE add-on, tedy tento modul.

Další informace

- https://learn.microsoft.com/cs-cz/windows-hardware/manufacture/desktop/download-winpe--windows-pe?view=windows-11
- https://learn.microsoft.com/en-us/windows-hardware/manufacture/desktop/winpe-intro?view=windows-11

WSIM (Windows System Image Manager) je nástroj s grafickým rozhraním, který lze kromě jiného použít pro vytvoření souboru odpovědí a bezobslužnou instalaci. Je součástí sady WADK.



Obrázek 5.5: Instalace modulu Windows PE do WADK

Okno aplikace obsahuje několik podoken, nás zajímají podokna *Windows Image* a *Answer File*. Podokno *Windows image* slouží k vytvoření a modifikaci bitové kopie, ze které bude systém instalován, a v podokně *Answer File* vytváříme a modifikujeme soubory odpovědí (můžeme jich mít i více).

5.3.3 Bezobslužná instalace Windows

Běžná instalace Windows, kdy u instalovaného stroje opravdu sedíme a reagujeme, je popsána na straně 147.

Jak bylo výše uvedeno, Windows jsou distribuovány na DVD ve formátu *WIM*. Jedná se vlastně o obraz disku, který je komprimovaný (pomocí různých postupů je zmenšena jeho velikost) a neredundantní (každá informace je zde jen jednou, i když po instalaci může být na disku na více různých místech).

Moduly z obrazu, které se mají nainstalovat, jsou stanoveny při instalaci podle typu zakoupené licence (podle *Product Key*, který uživatel obdržel při koupi). Pokud Product Key není během instalace zadáván, vybíráme variantu ze seznamu během instalace a Product Key zadáme později.

Postup (Bezobslužná instalace z DVD)

Bezobslužná instalace z DVD znamená instalaci s využitím instalačního média (DVD) a souboru odpovědí. Celý postup je následující:

1. Stáhneme a nainstalujeme WADK. Spustíme nástroj WSIM a připojíme instalační obraz Windows (buď vložíme do mechaniky nebo připojíme jeho obraz do virtuální mechaniky).
2. V připojeném obrazu najdeme složku `sources` a v ní soubor `install.wim`. Zkopírujeme ho někam na pevný disk, předpokládejme umístění `c:\sources\install.wim`. Pozor, soubor by neměl být nastaven „read-only“, WSIM do něj bude chtít zapisovat.
3. Stanovíme bitovou kopii, pro kterou budeme vytvářet soubor odpovědí:
   - V podokně *Windows Image* klepneme pravým tlačítkem a v kontextovém menu zvolíme *Select Windows Image*.
   - V dialogovém okně najdeme cestu k souboru `install.wim`, který jsme uložili na pevný disk (zde `c:\sources\install.wim`). Otevře se okno se seznamem bitových kopií, které jsou k dis-

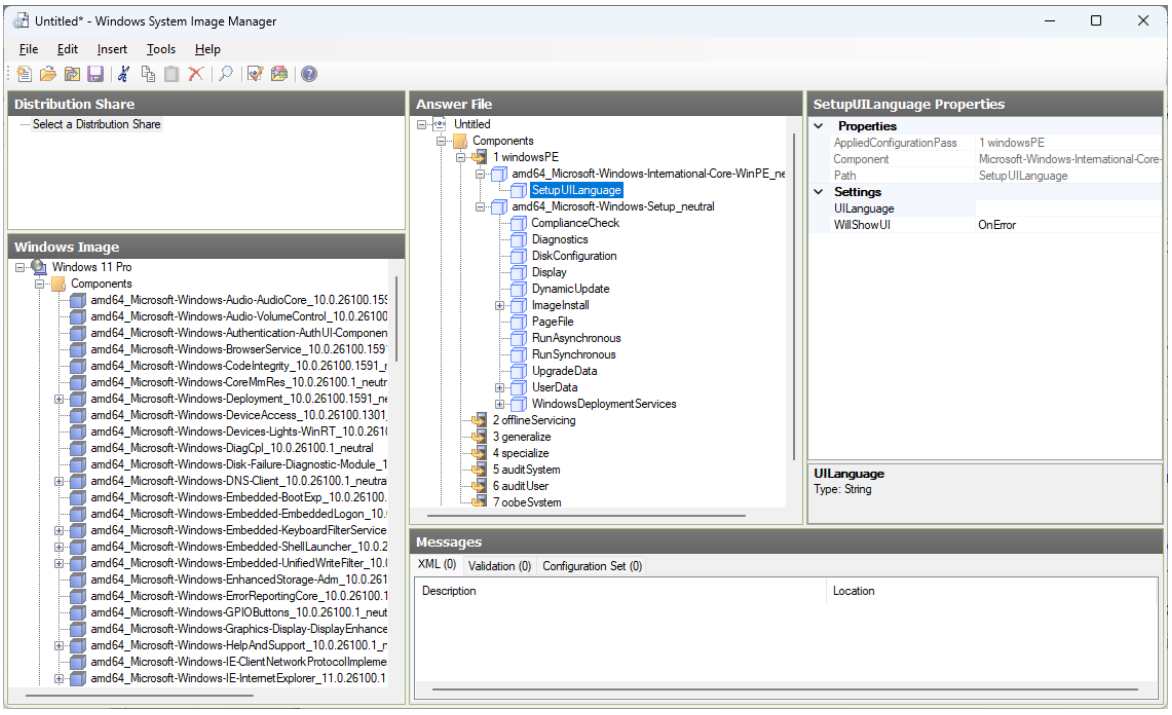
pozici (odpovídají variantám Windows pro danou verzi). Vybereme variantu, kterou potřebujeme (na kterou máme licenci).

- Může se zobrazit upozornění, že pro vybranou bitovou kopii ještě není vytvořen soubor katalogu, což provedeme (klepneme na tlačítko Ano, resp. OK v tomto okně).

4. Začneme vytvářet soubor odpovědí `autounattend.xml` tak, že v WSIM v podokně *Answer File* klepneme pravým tlačítkem a vybereme *New Answer File*.
5. Naplníme soubor odpovědí tak, že v podokně *Windows Image* postupně projdeme položky zde uvedené (jsou ve stromové struktuře) a na ty, které budeme chtít zařadit, klepneme pravým tlačítkem myši. V zobrazeném kontextovém menu klepneme na konkrétní položku, kterou chceme vložit do souboru odpovědí. Můžeme například stanovit, jak se má rozdělit pevný disk apod.
6. Položka, kterou jsme vybrali v podokně *Windows Image*, se objeví i v podokně *Answer File*, taktéž ve stromové struktuře. Procházíme jednotlivé položky; po klepnutí na položku se ve vedlejším podokně *Properties* zobrazí konkrétní vlastnosti vybrané položky, které můžeme pozměnit.
7. Po provedení změn uložíme soubor odpovědí pod názvem `autounattend.xml` na pevný disk a pak zkopírujeme na USB flash disk do kořenového adresáře (nesmí být v žádné složce!).

(Přesun k cílovému počítači)

8. Na počítači, kde chceme provést instalaci, připojíme USB flash disk se souborem odpovědí.
9. Dále do DVD mechaniky vložíme instalační DVD Windows nebo instalační USB flash disk.
10. Restartujeme tento počítač a necháme proběhnout instalaci. Soubor odpovědí by měl být automaticky rozpoznán a použit. Kdyby došlo k nějaké chybě (například kdyby nebylo správně detekováno médium se souborem odpovědí), probíhala by instalace klasickým způsobem, zadávali bychom všechny informace „ručně“.



Obrázek 5.6: Vytváření souboru odpovědí

Příklad (Klonování systému pomocí Clonezilly)

Pokud máme nainstalovaný referenční počítač (včetně aplikací), můžeme ho naklonovat buď pomocí výše uvedených nástrojů, což používáme například tehdy, když obrazy spravujeme centrálně přes Active Directory, ale pokud nepotřebujeme nic moc složitého, můžeme prostě použít nástroj Clonezilla.

Clonezilla je ve skutečnosti linuxová distribuce, která má jediný účel: vytvořit obraz nainstalovaného systému a přesunout ho na další počítače. Clonezillu si nejdřív stáhneme na bootovací USB flash disk (nebo ve formě ISO na CD, pokud máme optickou mechaniku), nabootujeme (tedy spustíme Clonezilla server), určíme zdrojový disk, určíme cílové disky/zařízení (zařízení musí být spuštěná) – to jsou Clonezilla klienti.

Je třeba brát v úvahu, že zejména u větších diskových oddílů to nebude za chvilku. K uložení obrazu potřebujeme dostatečně velké úložiště, typicky to bude externí disk (ne USB flash disk, ty mívají jen malou kapacitu).

V případě Windows bude na klientech po klonování následovat aktivace.

Další informace

https://clonezilla.org/

5.3.4 Služba pro nasazení systému Windows

Služba pro nasazení systému Windows (*WDS*, Windows Deployment Service) umožňuje instalovat také po síti. WDS je služba na straně serveru, musíme tedy mít v síti Windows Server vhodné verze. V síti je vyžadováno funkční Active Directory, DHCP a DNS server.

Aby WDS mohla řádně fungovat, musí být na serveru nainstalována a musíme mít uloženu

- alespoň jednu *spouštěcí* bitovou kopii, která slouží ke startu klientského počítače přes síť (lze pracovat na klientovi, i vzdáleně, ale systém běží na serveru), ale také např. k řízení instalace klientského počítače z některé instalační bitové kopie,
- alespoň jednu *instalační* bitovou kopii (z té se instalují klientské počítače).

Ve Windows Server se pracuje s *rolemi*, z nichž jedna je také WDS. Tedy instalace WDS probíhá přidáním této role. Instalujeme běžným způsobem jako cokoliv jiného – na serveru máme k dispozici jakousi hlavní grafickou konzolu *Správce serveru* (Server Manager) dostupnou hned z tlačítka *Start*, v této konzole najdeme instalování nové role a několika klepnutími myší přidáme roli WDS.

Popis činnosti a správy WDS je nad rámec těchto skript, proto nezbývá než odkázat na literaturu (vše je popsáno prakticky v každé knize určené pro správce Windows serveru). WDS dokáže ulehčit mnohé z úkolů, které jsme probírali výše o instalaci Windows.

5.3.5 Aktivace Windows

Některé edice Windows je nutné *aktivovat*. Bez aktivace běží systém po určitou dobu (30 dnů, *Grace Period*) bez restrikcí. Aktivace nebývá nutná u některých typů licencí, např. OEM (produkt byl aktivován při instalaci) nebo některých hromadných digitálních licencích. Aktivaci také můžeme provést při automatizované instalaci úpravou souboru odpovědí (funguje, pokud je počítač připojen k internetu).

Pro aktivaci potřebujeme *Product Key*, což je 25místný řetězec při zápisu rozdělený do skupin po pěti znacích, může vypadat třeba takto (samozřejmě s jinými znaky):

```
12345-67890-ABCDE-FGHIJ-KLMNP
```

Product Key najdeme buď na štítku nalepeném na počítači nebo na krabici od počítače (pokud se jedná o OEM instalaci), případně na krabici s instalačkami Windows (pokud je máme), nebo v potvrzujícím e-mailu, pokud jsme Windows koupili digitálně. Pokud máme systém Vista SP1 nebo novější a zároveň místo BIOSu používáme UEFI včetně Secure Boot, je Product Key uložen v UEFI. Product Key se dá zjistit i z běžícího systému, ale není to úplně jednoduché.

Další informace

Na https://www.thewindowsclub.com/find-windows-product-key je několik postupů, jak zjistit Product Key, což se rozhodně hodí, pokud přeinstalováváme. Další možnost je použít specializovaný program jako například *Abelssoft MyKeyFinder* (dostupný na https://www.abelssoft.de/en/windows/Helpers/MyKeyFinder), který vytáhne z registru všechna produktová čísla, která najde, včetně toho pro Windows.

Poznámka

V každém případě je dobré mít někde Product Key poznamenán. Pokud například chceme přeinstalovat Windows nebo provádíme razantnější změny hardwaru (procesor apod.), budeme ho potřebovat. Pokud máme systém Vista SP1 nebo novější a zároveň místo BIOSu používáme UEFI se zapnutým Secure Boot, obvykle není nutné při reinstalaci zadávat Product Key, načte se automaticky. Nicméně když se něco týká Windows, není dobré spoléhat na to, že to „obvykle“ půjde.

Samotná aktivace spočívá ve vygenerování kódu typického pro určitý počítač (Product ID – PID), vytváří se na základě hardwarové konfigurace počítače (to je vlastně důvod, proč se po „razantnějších“ a opakovaných změnách konfigurace začne systém chovat jako neaktivovaný). Závisí na 10 parametrech, po změně 5 z nich je považován za neplatný. Tento kód používá Microsoft pro ověření, zda komunikuje s právoplatným majitelem licence (například při stahování aktualizací).

Product ID je 20místné hexadecimální číslo, při zápisu se dělí do čtyř skupin po pěti číslicích. Pokud na dvou různých počítačích použijeme při aktivaci Product Key (třeba když starý počítač nahrazujeme novým a máme krabicovou verzi Windows, tedy můžeme migrovat), vznikne pokaždé jiné Product ID, protože to závisí na hardwaru.

Poznámka

Pozor, nepleťte si Product Key a Product ID. To první je něco na způsob „klíče“ odemykajícího licencované funkce Windows (použije se jednou, při aktivaci), to druhé je identifikátor vzniklý během aktivace a používá se neustále při komunikaci se servery Microsoftu. Pokud při instalaci zadáte místo Product Key číslo Product ID, bude hlášena chyba a aktivace se neprovede.

Product ID najdeme například v nástroji *Nastavení* v části *O aplikaci* jako „ID produktu“. O něco níže je odkaz *Změnit kód Product Key nebo upgradovat edici Windows*, který použijeme, když chceme aktivovat Windows nebo přejít na jinou edici (a tedy zadat odpovídající Product Key).

Úkol

Najděte u svého počítače Product ID.

5.4 Aktualizace systému

Microsoft stanovil druhé úterý každého měsíce pro den pravidelných bezpečnostních záplat (tomu dni se také říká „Patch Day“, den záplat), ale bezpečnostně závažné záplaty jsou distribuovány i mimo tyto dny.

Z předchozího semestru už víme, jaké typy aktualizací Windows existují a jak lze řídit jejich využívání na lokálním počítači. Jestliže však ve větší organizaci chceme zajistit aktualizaci všech počítačů v síti, není vhodné nechávat každého uživatele zvlášť stahovat aktualizace ze serveru Microsoft Update.

Obvyklým řešením je stanovení lokálního serveru *WSUS* (Windows Server Update Services), který stahuje aktualizace z webu Microsoftu a dále je distribuuje v lokální síti.

Další informace

Na stránce https://learn.microsoft.com/en-us/windows/deployment/ najdeme dokumentaci k službám souvisejícím s nasazením systému.

Úkol

Na výše uvedené webové stránce si projděte jednotlivé části (všimněte si, že menu vlevo není přesnou kopií toho, co je v dlaždicích). Zaměřte se také na Deployment Tools. Není tam něco o Windows PE?

5.5 Chyby při běhu aplikací a systému

Někdy se stane, že se aplikace při své práci dostane do nekonečné smyčky nebo dojde k uváznutí při čekání na prostředek (bude probíráno na přednáškách), a nereaguje. U Win aplikací to poznáme ve *Správci úloh* (známe z předchozího semestru) označením „neodpovídá“ u této aplikace. Ve *Správci úloh* také můžeme tuto aplikaci násilně ukončit označením aplikace a stisknutím tlačítka *Ukončit úlohu*.

Pokud je některá aplikace spuštěna v celoobrazovkovém režimu (DOS aplikace, Flash, prezentace – PowerPoint, Adobe Reader), a „zasekne se“ nebo nenabízí možnost pro své ukončení, přesuneme se do Windows, kde se dá „zlikvidovat“:

Alt+Enter (do okna), pak v kontextovém menu tlačítka aplikace na *Hlavním panelu* zvolíme *Zavřít*,

Alt+Tab (přepnutí úlohy), podobně jako v předchozím případě,

Ctrl+Esc nebo ⊞ (otevření menu *Start*), podobně jako v předchozích případech,

Ctrl+Alt+Del nebo Ctrl+Shift+Esc zobrazíme *Správce úloh*, v něm aplikaci ukončíme.

Jestliže je poškozen některý z chráněných systémových souborů (nebo přepsán při instalaci nové aplikace), lze ho opravit programem *System File Checker*. Tento program obnovuje systémové soubory ze zálohy. Spustíme ho příkazem `sfc` (vlastně ho už známe, viz str. 82) při použití vhodných parametrů (viz nápovědu příkazu – například je možné spustit okamžitou kontrolu, kontrolu při příštím spuštění Windows nebo pravidelnou kontrolu při každém spuštění).

Modrá obrazovka („obrazovka smrti“, BSOD – Blue Screen of Death, podle terminologie Microsoftu „obrazovka Stop“) se zobrazí při narušení jádra systému (chybějící nebo poškozená část systému, zaseknutí systémové části, provádění některého kódu ve smyčce, spuštění více antivirových programů či firewallů zároveň, konflikty u ovladačů, chybně pracující ovladač...), někdy za to může virus. Modrou obrazovku generuje *vždy* jádro systému.

Často je jedinou možností restart počítače. Pokud se tato obrazovka objeví při startu systému, je to zřejmě způsobeno nesprávnou instalací nebo konfigurací některého zařízení. Řešením je spustit Windows v nouzovém režimu a prohlédnout protokol `ntbtlog.txt`.

Pokud se objeví modrá obrazovka, systém (s NT jádrem) vytvoří *soubor výpisu stavu paměti*, ze kterého je možné zjistit přesnou příčinu tohoto stavu (je použitelný také v případě, že se nepodaří nastartovat systém, jen zkopírujeme soubor):

- *omezený výpis paměti* (ve složce `Windows\Minidump` na systémovém disku, zabírá jen 64 kB paměti), obsahuje pouze zásobník vlákna, které způsobilo pád jádra,
- *výpis stavu paměti jádra* (tamtéž, rozsáhlejší), obsahuje výpis paměti používané jádrem,
- *úplný výpis stavu paměti* (v souboru `Memory.dmp` na systémovém disku, zabírá místo odpovídající fyzické kapacitě operační paměti plus 1 MB) a obsahuje výpis paměti jak jádra, tak i uživatelského prostoru (tj. všech běžících procesů).

O tom, který výpis bude generován, rozhodujeme pomocí nástroje *Systém*, *Upřesnit nastavení systému*, záložka *Upřesnit*, část *Spuštění a zotavení systému*, tlačítko *Nastavení*.

Další informace

V souboru s výpisem paměti bychom se jen těžko orientovali. Pro jeho analýzu se používají nástroje na ladění jádra. Informace najdete v příloze anebo na http://www.zezula.net/cz/prog/pm_os_debug.html.

Jestliže se objeví chyba při startu Windows (nelze nastartovat Windows), reagujeme některým z těchto způsobů:

1. Použijeme některou možnost pro *nouzový režim*, jak je popsáno v sekci o startu systému. V nouzovém režimu můžeme pro obnovení systému použít mimo jiné i nástroj *Obnovení systému*, obnovit registr ze zálohy, spouštět různé programy včetně nástrojů pro konfiguraci a úpravu registru (samozřejmě podle typu spuštění nouzového režimu), případně přeinstalovat vadné ovladače.
2. Nabootujeme *Windows PE* (ale musíme mít k dispozici, předem si vytvořit). Případně nabootujeme některou ze záchranných linuxových distribucí jako je *System Rescue*.
3. Ve Windows je možné si použít záchranný disk, samozřejmě je nutné vytvořit předem! Postup se pro různé verze Windows liší, obvykle je tato možnost v zálohovacím nástroji.
4. Ve Windows od verze 8 lze systém resetovat do továrního režimu, to je ovšem až poslední možnost, kterou volíme jako méně destruktivní alternativu přeinstalování.

Chyby na ovladači: Ve Windows máme v případě instalace chybného ovladače možnost *vrátit změny na ovladači*. Ve *Správci zařízení* (ten je dostupný třeba v nástroji *Systém*, nebo přes `devmgmt.msc`) najdeme zařízení, které dělá problémy, zobrazíme jeho vlastnosti a na kartě *Ovladač* použijeme tlačítko *Vrátit změny ovladače*. To je samozřejmě použitelné pouze tehdy, když se dostaneme do grafického prostředí Windows, funkce má pouze odbourat nutnost ovladač ručně odinstalovat a nainstalovat ten původní (ale to samozřejmě jde taky).

Jinou možností nápravy je nástroj *Obnovení systému*.

Pokud je problém s externím paměťovým zařízením připojovaným přes USB (externí disk, USB flash disk), například není správně rozpoznáno jako paměťové zařízení, obvykle pomůže toto řešení:

- ověříme, zda je zařízení připojeno k funkčnímu USB portu (případně vyzkoušíme jiný port) – chyba může být i v hardwaru,
- přejdeme do *Správce zařízení* a pokusíme se najít ovladač daného zařízení (ve větvi *Diskové jednotky*); když ho tam nenajdeme, ověříme, zda je zapnuto zobrazování skrytých zařízení (v menu *Zobrazit*),
- pokud je zapnuto zobrazování skrytých zařízení, ale ovladač se přesto nezobrazuje, přejdeme do nástroje *Systém*, volba *Upřesnit nastavení systému*, na kartě *Upřesnit* klepneme na tlačítko *Proměnné prostředí*, vytvoříme novou proměnnou s názvem
  `devmgr_show_nonpresent_devices`
  nastavíme na hodnotu `1`, čímž ve *Správci zařízení* zapneme zobrazování poškozených ovladačů,
- pokud se ve *Správci zařízení* konečně náš ovladač objeví, odstraníme ho (stačí klepnout na Del),
- zařízení znovu připojíme, ovladač se automaticky nainstaluje znovu (a už by měl být v pořádku).

Úkoly

1. Zjistěte, jaký typ výpisu stavu paměti při modré obrazovce je generován a kam konkrétně se ukládá. Zjistěte, zda má být systém při modré obrazovce automaticky restartován.
2. Zobrazte nápovědu programu *System File Checker* (`sfc`) a zjistěte, co znamená použití přepínačů `/quiet` a `/cancel`.
3. Vyzkoušejte, jak je možné vrátit změny na ovladači nebo ovladač odinstalovat (nemusíte postup dovést až do konce).
4. Prohlédněte si seznam bodů obnovy, které jsou vytvořeny za poslední týden.

5.6 Správa softwaru

5.6.1 Instalace aplikací

Většina aplikací dnes nabízí *instalační proces*, který provede vše potřebné od detekce ovladačů přes editaci registru a registraci DLL knihoven až po přesun adresářů a souborů na pevný disk. Tento instalační proces si programátor buď vytvoří sám, anebo využije některý existující program, který pouze upraví. Instalační programy využívají do určité míry službu *Služba Windows Installer* (resp. *Instalační službu systému Windows*) zabudovanou do Windows, ale jinak se jejich vlastnosti mohou hodně lišit.

Hodně používaný je nástroj *InstallShield Developer*[2], který nabízí prakticky vše, co si programátor může přát, ale možná i z toho důvodu jsou používány i jiné programy, například *CreateInstall*[3], nebo *Inno Setup*[4].

MSI a MSIX balíčky. Podobně jako v Linuxu máme software distribuovaný v .DEB nebo .RPM souborech, ve Windows se už desítky let používají .MSI balíčky. V poslední době k nim přibyly .MSIX balíčky obsahující aplikace z Windows Store.

[2] https://www.revenera.com/install/products/installshield
[3] https://www.createinstall.com/
[4] https://jrsoftware.org/isinfo.php, freeware, je velmi jednoduchý, třebaže použitelný

Tyto typy balíčků mají na starosti následující služby:

- *Instalační služba systému Windows* (msiserver) – zajišťuje správu klasických aplikací, které často bývají instalovány z .msi balíčků, výkonnou částí je program `msiexec.exe`,
- *AppX Deployment Service* (appxsvc) – tuto roli plní pro Windows Store aplikace distribuované často v balíčcích .msix, výkonnou částí je program `appinstaller.exe`.

S instalací a provozem aplikací souvisí poměrně hodně systémových služeb. Kromě výše zmíněné *Instalační služby systému Windows* (msiserver) je to třeba služba *Identita aplikace* (AppIDsvc) pro ověření identity aplikace (pomocí AppLockeru je možné aplikaci zakázat), nebo například *Služba pro klientské licence* (ClipSVC) a *Služba správce licencí Windows* (License Manager), které jsou nezbytné, pokud chceme používat aplikace z Windows Store.

Automatizace instalace softwaru. Pomocí nástroje *Zásady skupiny* můžeme na počítačích v doméně se službou Active Directory řídit instalaci následujícím způsobem: aplikaci neinstalujeme osobně, ale „přenecháme tuto čest“ uživateli, i když mu neponecháme mnoho volnosti a proces bude automatizovaný. Aplikaci můžeme

- *přiřadit počítači* – aplikace se tváří, jakoby byla již nainstalována, její odkaz je například v menu *Start* ⇨ *Programy*, ale samotná instalace se spustí v okamžiku, kdy se uživatel počítače pokusí aplikaci poprvé použít,
- *přiřadit uživateli* – podobně jako předchozí, ale aplikace je „nasimulována“ v menu *Start* ⇨ *Programy* pouze při přihlášení toho uživatele (a to v jakémkoliv počítači, ke kterému se může přihlašovat), kterému je přiřazena, po „spuštění“ se nainstaluje tak, aby byla dostupná tomuto uživateli,
- *publikovat* – uživatel může sám rozhodnout, jestli si aplikaci nainstaluje; aplikace je uvedena v nástroji *Přidat nebo odebrat programy*.

Poznámka

Aby uživatel mohl na klientském zařízení využívat přiřazení nebo publikování aplikací (tj. přes skupinové zásady v Active Directory), musí mít spuštěnu službu *Správa aplikací* (AppMgmt) – ruční spuštění.

Pokud máme v síti server s nainstalovanými Windows Server verze minimálně 2008, můžeme použít nástroj *System Center Configuration Manager* (ConfMgr, také SCCM). Je to hodně komplexní nástroj, který umožňuje spravovat nejen instalovaný software a jeho distribuci v síti, ale také distribuci bezpečnostních záplat, vzdálené řízení, nasazení operačního systému (funguje tedy jako konsolidační rozhraní k jiným nástrojům a rolím včetně WSUS pro aktualizace, bezobslužné instalace na klientech apod.).

SCCM může mít plnou kontrolu nad softwarem instalovaným na zařízeních v síti Active Directory – u každého zařízení „vidí“ seznam aplikací s jejich verzemi, může přiřazovat či publikovat instalační balíčky, hlídá případné závislosti či kolize mezi aplikacemi, zajišťuje aktualizace (systému i softwaru), atd.

Uživatelé mají přístup do katalogu softwaru zajištěn buď přes speciální webovou stránku (to je třeba nakonfigurovat na serveru, *SC Configuration Manager Application Catalog*), nebo je možné na klientská zařízení nainstalovat klientskou aplikaci *Software Center*, která pak slouží jako rozhraní ke katalogu.

Další informace

- https://www.samuraj-cz.com/clanek/system-center-configuration-manager-2012-instalace/
- https://www.samuraj-cz.com/clanek/sccm-2012-sprava-instalace-aplikaci/
- https://www.samuraj-cz.com/clanek/sccm-2012-software-update-management/
- https://prajwaldesai.com/sccm-2012-r2-step-by-step-guide/
- https://www.zive.cz/clanky/prvni-pohled-na-system-center-configuration-manager-2012/sc-3-a-161369/default.aspx
- https://learn.microsoft.com/en-us/mem/configmgr/

5.6.2 Instalační soubory aplikací

Soubory .MSI jsou tedy instalační balíčky. V těchto souborech bývá (skoro) všechno, co je třeba při instalaci aplikace, ale .MSI soubor může být doprovázen jedním nebo více soubory s příponou .CAB, ve kterých kromě dat může být uložena bezpečnostní informace (včetně digitálního podpisu).

Kromě MSI jsou používány ještě další přípony, například MSP pro záplaty (patch), MSU (MS Update) pro aktualizace v novějších Windows, nebo MST pro transformace (tento soubor lze vytvořit zároveň se souborem MSI, obsahuje transformace, které je nutné provést po instalaci, například v registru, často jsou takto řešeny jazykové varianty).

Na Příkazovém řádku můžeme použít nástroj `msiexec`:

`msiexec /?` zobrazí se okno (v grafickém režimu) s nápovědou

`msiexec /i cesta\balíček.msi` nainstalujeme zadaný balíček

`msiexec /x cesta\balíček.msi` odinstalujeme zadaný balíček

`msiexec /i cesta\balíček.msi /L*V logsoubor.log` do zadaného LOG souboru se zapíše informace o průběhu instalace; tímto způsobem můžeme zjistit, proč instalace aplikace selhává

`msiexec /x {56D35A37-90B7-45BC-B3AA-9C5233F8E3D7}` instalace balíčku zadaného pomocí identifikátoru GUID (identifikátor GUID zjistíme například v LOG souboru podle předchozího příkazu), tento postup někdy funguje i v případě, že odinstalace aplikace se nedaří při zadání názvu balíčku

`msiexec /ju cesta\balíček.msi` přiřadí (inzeruje) aplikaci uživateli (právě přihlášenému)

`msiexec /jm cesta\balíček.msi` přiřadí aplikaci všem uživatelům počítače (tj. počítači)

`msiexec /a cesta\balíček.msi` administrativní (síťová) instalace – jsme dotázáni, kam má být aplikace umístěna, a potom se balíček na zadaném místě rozbalí podobně, jako by se opravdu instaloval, ale neprovedou se žádné změny v systému (předpokládáme, že jde o server, tedy aplikace nebude provozována na serveru), vytvoří se „menší“ MSI balíček, který je publikován klientským počítačům

`msiexec /a soubor.msi /p záplata.msp` použití záplaty na aplikaci, která byla publikována na síti

Dalšími přepínači lze například podrobně řídit logování, zakázat zobrazování grafického rozhraní instalátoru (pro bezobslužný mód instalace), apod.

Pomocí Active Directory můžeme instalovat po síti takto:

- přihlásíme se na server, který je doménovým řadičem AD, jako Domain/Administrator,
- provedeme administrativní instalaci (`msiexec /a`), teď by nám mělo být známo umístění MSI souboru ve sdílené složce na tomto serveru (tato složka musí být přístupná všem klientům, na kterých chceme instalovat),

- na serveru spustíme MMC konzolu *Group Policy Management* (správa zásad skupiny), zobrazí se struktura položek, ve které najdeme *Group Policy Objects* (GPO),
- v kontextovém menu položky zvolíme *Nový* (New), vytvoříme nový GPO (vpodstatě jde o novou *zásadu*, která má být aplikována na zadané počítače), vhodně nazveme (podle instalované aplikace),
- po vytvoření musíme GPO nastavit – klepneme na vytvořenou položku a nastavíme vše, co potřebujeme (v pravém podokně na všech záložkách), například určíme objekty, na které se má zásada aplikovat (v našem případě počítače),
- potom v kontextovém menu vytvořeného GPO zvolíme *Edit*; otevře se konzola *Group Policy* pro tuto položku, ve které najdeme *Computer Configuration* ⇨ *Software Settings* ⇨ *Software Installation*, v kontextovém menu této položky zvolíme *New* ⇨ *Package* a najdeme MSI balíček, který jsme vytvořili na začátku tohoto postupu,
- dále určíme, jestli na balíček chceme aplikovat nějaké transformace (volba *Advanced* – rozšířené možnosti nasazení), v tom případě musíme ve vlastnostech položky „naťukat“ příslušné volby pro transformace, potvrdíme,
- v konzole *Group Policy* pro námi vytvořený GPO ještě nastavíme podrobnosti instalace – najdeme *Computer Configuration* ⇨ *Administrative Templates* ⇨ *Windows Components* ⇨ *Windows Installer* a nastavíme zde zásady podle našich požadavků (například můžeme nastavit, že se má aplikace instalovat se zvýšenými oprávněními apod.).

InstallShield vytváří místo MSI souborů spustitelné instalační EXE soubory. Hlavní spustitelný soubor instalace se většinou nazývá `setup.exe` a podporuje některé běžné přepínače (například pro vytvoření souboru odpovědí `setup.iss`, bezobslužné instalace apod.).

`setup.exe /r` pokud nemáme soubor odpovědí, takto ho vytvoříme (spustí se modelová instalace, ve které „naklikáme“ vše potřebné a všechna zvolená nastavení se uloží do souboru `...\windows\setup.iss`), vytvořený soubor odpovědí pak přesuneme do složky, ve které máme instalační soubor `setup.exe`

`setup.exe /s` pokud máme ve stejné složce i soubor odpovědí `setup.iss`, je takto spuštěna bezobslužná instalace

Další informace

Zajímavý článek o instalátorech aplikací najdeme například na https://unattended.sourceforge.net/installers.php.

Úkol

Zobrazte nápovědu příkazu `msiexec` a zjistěte, co vše lze během činnosti tohoto programu ukládat do LOG souboru a jakým způsobem se tyto volby nastavují.

5.6.3 WinGet

Podobně jako v Linuxu, také ve Windows existuje systém správy softwarových balíčků, například MSI. Software se nachází v různých zdrojích, k těmto zdrojům lze přistupovat pomocí určitých nástrojů. V novějších verzích Windows máme k dispozici nástroj WinGet, který dokáže přistupovat k různým zdrojům a vyhledávat a instalovat v nich obsažené aplikace.

Nástroj WinGet je obvykle už nainstalován (ve Windows 10 od verze 1809, Windows 11 a novějších). Kdyby ne, dá se doinstalovat přes Microsoft Store (tam se nazývá „App Installer“, resp. „Instalační program aplikací“).

Příklad

Pokud napíšeme pouze příkaz `winget` bez parametrů, zobrazí se nápověda.

Když chceme najít balíček, použijeme klíčové slovo `search`, například nás zaujal nástroj Powertoys (pravděpodobně bude třeba odsouhlasit podmínky použití):

```
C:\Users\sarka> winget search powertoys

Name                Id                            Version Match          Source
-------------------------------------------------------------------------------
Microsoft PowerToys XP89DCGQ3K6VLD                Unknown                msstore
PowerToys (Preview) Microsoft.PowerToys           0.79.0                 winget
EverythingPowerToys lin-ycv.EverythingPowerToys 0.78.0  Tag: powertoys winget
```

Každý balíček má svůj název (první sloupec) a své ID (druhý sloupec). Když chceme balíček nainstalovat, můžeme použít krátký název (ten, který jsme zde použili při vyhledávání) nebo ID daného balíčku. Krátký název nemusí být jednoznačný, například zde je stejný pro tři různé balíčky, ID je vždy jednoznačné. Při instalaci je přehlednější využít ID ve formě autor.název, například takto:

```
winget install microsoft.powertoys
```

Vyhledávat se dá také podle vhodného klíčového slova, včetně kategorie. Například pokud si chceme stáhnout a nainstalovat nějaký další webový prohlížeč, můžeme se zeptat:

```
winget search browser
```

V seznamu najdeme nejen Edge, ale také Firefox, Brave, Avast Secure Browser, Operu, Chrome, DuckDuckGo, Safari a další. Kromě webových prohlížečů tam jsou i jiné (prohlížeče souborů, LDAP, obrázků,...).

Všimněte si, že pro jeden produkt jsou často k dispozici různé varianty. Může jít o různé vývojové větve (stabilní, experimentální, vývojová nestabilní, canary = raně vývojová, atd.).

Poznámka

V nápovědě se k některým možnostem dostáváme postupně. Například pokud chceme podrobněji řídit instalaci nového softwaru, je dobré se podívat na možnosti: `winget install -?`. Tak například zjistíme, že můžeme ovlivnit, zda se aplikace nainstaluje pro všechny uživatele nebo jen pro jednoho (parametr `--scope`), popřípadě jak donutit instalátor, aby nám umožnil i pokročilejší nastavení a neuvrhl nás do defaultních (parametr `--custom`).

Pomocí WinGet se dá na příkazovém řádku provést i aktualizace aplikací (přesněji: balíčků, ke kterým má WinGet přístup), a to takto:

```
winget upgrade --all
```

A pokud chceme tento proces nechat běžet na pozadí, můžeme přidat parametr `--silent`.

Program WinGet má i grafické rozhraní – vlastně dokonce několik. Oblíbený je například WinGetUI, který můžeme najít a nainstalovat pomocí samotného programu WinGet.

Program WinGet pracuje s programy distribuovanými přes MS Store, ale také s dalšími úložišti (zdroji softwaru). Dokonce má ve své databázi i to, co bylo nainstalováno jinak. Pokud chceme zjistit, které balíčky jsou již nainstalovány (nejen pomocí WinGet), můžeme použít příkaz:

```
winget list
```

Jde o poměrně dlouhý seznam (jsou tam například také aktualizační balíčky některých aplikací), tedy je praktické si výstup profiltrovat (třeba pomocí `findstr`). Ještě praktičtější je si seznam exportovat ve vhodném formátu, ideálně tak, aby se dal využít třeba při migraci na jiné zařízení.

Příklad

Předpokládejme, že si chceme udělat seznam instalovaného softwaru na daném počítači. Můžeme použít tento příkaz:

```
winget export -o seznamsw.json
```

Pravděpodobně dostaneme spoustu chybových hlášení, protože některé aplikace odmítají s WinGet spolupracovat. Ale alespoň část se v seznamu objeví. Výsledný soubor je ve formátu .JSON (a to i v případě, že napíšeme jinou příponu), což je textový formát.

Tento soubor pak můžeme přenést na jiný počítač, kde ho importujeme:

```
winget import -i seznamsw.json -accept-package-agreements
```

(ten poslední parametr není nutný, ale určitě nechceme při větším množství balíčků během instalace pořád potvrzovat, že souhlasíme s licenčními podmínkami).

Úkol

Zjistěte, zda je na vašem systému nainstalován `winget` (stačí vyzkoušet bez parametrů). Pokud ano:

- Zjistěte, které webové prohlížeče se takto dají nainstalovat.
- Zjistěte, zda jsou k dispozici nějaké jednoduché textové editory (klíčové slovo „notepad“).
- Najděte přes tento nástroj programy CPU-Z, Thunderbird, Gimp, Sysinternals Suite (pozor – víceslovný název je třeba dát do uvozovek), FreeCommander.
- Prohlédněte si nápovědu pro instalaci nového softwaru pomocí `winget`.

Souhrn kapitoly 5

V této kapitole jsme se krátce zabývali přístupem do registru v textovém režimu. Pak jsme se soustředili na průběh a možnosti ovlivnění startu systému a dále jsme se zaměřili na instalaci Windows: jak probíhá, jaké nástroje lze použít, jak se dělá bezobslužná instalace, jak použít službu WDS na serveru. Následovala aktualizace systému a chyby při běhu systému. Poslední část kapitoly byla věnována správě softwaru, včetně programu WinGet.

Část II

Linux

Kapitola 6

Návrat k shellu bash

Rychlý náhled: V této kapitole si upevníme a rozšíříme své znalosti a dovednosti v shellu bash. Ukážeme si, že existují i jiné shelly a jak se k nim dostat, zopakujeme si používání nápovědy, naučíme se vytvořit nový soubor v textovém režimu. K proměnným zde je sekce o jejich využití pro výpočty. Následují sekce o psaní skriptů v bash a vysvětlení dalších užitečných příkazů pro větvení a cykly. Na konci kapitoly si ukážeme, jak vytvořit a přeložit program s využitím gcc, a taky jaké jsou programy pro konverzi textových souborů.

Klíčová slova: Shell, bash, tcsh, ksh, dash, zsh, login shell, here document, filtrování souborů, prohledávání adresářové struktury, prohledávání textů, proměnná, skript, propojení příkazů, příkazy větvení, cykly, překlad programů, knverze souboru.

Cíle studia: V této kapitole se naučíte přepínat mezi různými instalovanými shelly, upevníte si postupy využívání nápovědy v bash, naučíte se vytvořit nový soubor mechanismem here document a naučíte se provádět výpočty v shellu bash. Dále se naučíte psát a spouštět skripty v bash, a také používat různé složené příkazy včetně příkazů větvení a cyklů. Naučíte se překládat programy v jazyce C pomocí programu gcc a zjistíte, jaké jsou možnosti konverze textových souborů do různých kódování.

6.1 Textové shelly v UNIX-like systémech

Původním shellem v UNIXu byl *Bourne Shell* (vznikl roku 1978 a označoval se jednoduše `sh`), jehož autor je Stephen Bourne. O něco později (v době přepisu UNIXu do jazyka C) vznikl jako alternativa *C Shell* (označoval se `csh`). Zatímco Bourne Shell byl určen spíše pro psaní skriptů a umožňoval lepší spolupráci programů (včetně směrování), C Shell hodně inspirovaný jazykem C se orientoval hlavně na interaktivní práci přímo v příkazovém řádku (například používání historie pomocí speciálních příkazů) a také přinesl pokročilejší správu úloh.

V současné době se už nesetkáme přímo s Bourne Shellem a C Shellem, ale s jejich potomky:[1]

- *Toronto C Shell* (`tcsh`) je rozšíření `csh` o další možnosti, například používání šipek při práci s historií příkazů, `tcsh` také odstranil některé chyby související s činností ve skriptu, které se vyskytovaly v `csh`,

[1] Pěkné porovnání shellů (dokonce včetně Příkazového řádku ve Windows) najdeme například na stránce https://en.wikipedia.org/wiki/Comparison_of_command_shells.

- *Korn Shell* (`ksh`) – syntaxe příkazů odpovídá Bourne Shellu, ale jinak většinu vlastností přejal z `tcsh`, jde vlastně o hybrid shellů `sh` a `tcsh`,
- *Bourne Again Shell* (`bash`) je nejobvyklejším shellem v Linuxu a je velmi podobný `ksh` (je jednodušší, například oproti `ksh` neobsahuje podporu racionálních čísel a vícedimenzionálních polí),
- *Debian Almquist Shell* (`dash`) je potomkem shellu `ash` (Almquist Shell) ze systému FreeBSD, jak název napovídá, můžeme se s ním setkat u Debianu a jeho potomků (včetně Ubuntu); je podobný shellu `bash`, ale mírně osekaný (některé vlastnosti shellu `bash` nepodporuje) a rychlejší,
- *Z Shell* (`zsh`) je naopak vybavenější než `bash`, a to směrem k vědeckým výpočtům (je srovnatelný spíše s shellem `ksh`). Pro zsh existují dokonce plug-iny. Velmi oblíbený je framework „Oh My Zsh“, po jehož instalaci získáme přístup ke spoustě dalších plug-inů, témat, funkcí a dalších prvků vytvořených komunitou fanoušků zsh. Ve většině linuxových distribucí není defaultně nainstalován, ale balíček zsh není problém doinstalovat.

V Linuxu je vždy (nebo téměř vždy) nainstalován shell `bash`, a kromě něho i několik dalších.

Seznam použitelných shellů (těch, které máme k dispozici) je v souboru `/etc/shells`. Mezi shelly se přepínáme příkazem `chsh` (je to zkratka z CHange SHell).

Příklad

Ukážeme si použití příkazu `chsh`.

`man chsh` spustíme prohlížeč nápovědy pro příkaz `chsh`, zjistíme si syntaxi příkazu, tedy jaké parametry podporuje, program `man` ukončíme stiskem klávesy Q ,

`chsh -s /bin/zsh` právě jsme svůj *login shell* (tj. shell spouštěný při našem přihlašování) nastavili na *Z Shell*, změna se projeví po odhlášení/přihlášení,

`chsh -s /bin/bash` nastavili jsme zpět shell *bash*.

Pokud je shell, který chceme takto zapnout, nainstalován, ale přitom není uveden v souboru `/etc/shells`, nebude to fungovat a zobrazí se chybové hlášení. Úpravou tohoto souboru může totiž administrátor zakázat používání těch shellů, které nepovažuje za bezpečné (prostě jejich řádky vymaže), resp. povolit pouze konkrétní shelly, které sám určí.

Další možnou příčinou selhání příkazu je zadání nesprávné cesty k souboru shellu (jde přece o program), zde stačí zjistit, kde konkrétně je shell nainstalován.

Také v UNIXových shellech existují buď vnitřní (vestavěné) příkazy, které jsou součástí příslušného shellu, nebo vnější (externí) příkazy reprezentovné spustitelným souborem, jako další typ specifický pro UNIXové systémy můžeme uvést aliasy.

Příkaz `type` nám řekne, o jaký typ příkazu se jedná – vnitřní nebo vnější. Jako parametr jednoduše zadáme příslušný příkaz.

Příklad

Zadáme tyto příkazy:

```
type cd          type echo          type passwd
type find        type ls            type ping
```

Některé jsou vnitřní příkazy shellu bash, další jsou buď vnější příkazy nebo aliasy (jak víme, aliasy se někdy používají pro nastavení výchozích parametrů příkazů, často se s tím setkáváme právě u `ls`).

Úkoly

1. Zobrazte si obsah souboru `/etc/shells`.
2. Vyzkoušejte příkaz `type` podle předchozího příkladu, pak dosazujte jiné příkazy, na které si vzpomenete.

6.2 Nápověda

Zopakujeme si, jak se dostat k nápovědě. Nápovědu k příkazům můžeme získat více způsoby:

- zobrazením manuálové stránky příkazu, a to příkazem `man`, mohou být k dispozici i `info` stránky,
- příkaz `apropos` použijeme, když nevíme, jak se příkaz nazývá,
- příkaz `whatis` použijeme, když jsme narazili na příkaz (spustitelný soubor), ale nevíme, co provádí,
- existují *dokumenty HOWTO*, případně *dokumenty FAQ* (Frequently Asked Questions),
- v internetovém prohlížeči (přesněji – zeptáme se Googlu) se také dostaneme na manuálové stránky, například zadáme `man chsh` do adresního řádku.

Další informace

Na internetu je hodně stránek věnovaných shellu `bash`, například:

- https://www.gnu.org/software/bash
- https://tldp.org/LDP/abs/html (skripty)
- https://www.abclinuxu.cz/clanky/navody/bash-i

6.3 Adresáře a soubory

6.3.1 Možnosti vytvoření nového souboru

V předchozím semestru jsme se setkali s několika možnostmi vytvoření nového souboru, některé z nich dovolují zároveň do souboru něco uložit. Máme tyto možnosti:

- kopírováním a směrováním do souboru
- `touch soubor`
- `cat /dev/null > soubor`
- `cat > soubor`, napíšeme pár řádků, pak stiskneme Ctrl+D

Poslední uvedený způsob funguje i u Windows (použití příkazu `type`, kterému na vstup směrujeme vstup z konzoly `CON`). V Linuxu (na rozdíl od windowsovského `type`) příkaz `cat` dokáže pracovat i interaktivně, proto nemusíme explicitně zadávat vstup (konzolu, zde třeba `/dev/tty`).

Příklad

V pracovním adresáři vytvoříme nový soubor s názvem `soubor.txt`:

```
cat > soubor.txt   Enter

Toto je první řádek,
a teď píšeme další řádek.
Tento řádek bude poslední.

 Ctrl+D , pak  Enter
```

Klávesová kombinace Ctrl+D je důležitá a v tomto případě bychom si ji měli pamatovat, vytvoří textový znak znamenající konec souboru.

Pokud si „nemůžeme zapamatovat“ ukončující sekvenci Ctrl+D, která představuje konec souboru, máme možnost použít tzv. *vložené soubory* (here documents). Je to (téměř) totéž, jen navíc stanovíme ukončující sekvenci. Použití si ukážeme na příkladu:

Příklad

Vytvoříme nový soubor pomocí vloženého souboru (here document), tedy stanovíme ukončující sekvenci, po jejímž zadání bude soubor uzavřen, příkaz ukončen, a tato sekvence se nestane součástí vytvořeného souboru. Zadáme:

```
cat > soubor.txt << KONEC   Enter

Toto je první řádek,
a teď píšeme další řádek.
Tento řádek bude poslední.
KONEC

 Enter
```

Po napsání ukončující sekvence („`KONEC`“) se příkaz ukončí a soubor je vytvořen. Řetězec `KONEC` nebude součástí výsledného souboru.

Úkol

Vyzkoušejte si vytvoření nového souboru v textovém režimu pomocí příkazu `cat`, včetně využití mechanismu *here document*.

6.3.2 Filtrování souborů

Pro jednoduché filtrování seznamu souborů se dá použít příkaz `ls` nebo `echo` (obojí jsme probírali v předchozím semestru), přičemž používáme zástupné symboly (globing characters) – je to zjednodušená obdoba regulárních výrazů:

- hvězdičku (jakýkoliv řetězec jakýchkoliv symbolů), například `a*` znamená, že hledáme řetězce, kde první písmeno je `a` a následuje jakýkoliv počet čehokoliv dalšího,
- otazník (jeden jakýkoliv symbol), například `m???` znamená, že hledáme čtyřpísmenná slova, která začínají písmenem `m`,
- hranaté závorky (zadáváme množinu povolených symbolů, přičemž jeden z nich je na daném místě použit), např. `[a-z]` určí, že na daném místě může být jakékoliv malé písmeno, kdežto schéma `[a-z]*` znamená řetězec začínající (jakýmkoliv) malým písmenem,
- hranaté závorky s vykřičníkem určují negaci obsahu množiny – např. `[!a-z]` znamená, že na tomto místě může být cokoliv kromě malého písmene, nebo `ab[!0-9]*` určuje, že hledáme řetězec se třemi znaky, z nichž první dva jsou písmena `ab`, následuje jeden znak, který není číslicí a za ním je cokoliv (takže schématu by vyhovoval například řetězec `abcxyz21`).

Tyto zástupné symboly jsou vyhodnocovány textovým shellem (např. bash), nikoliv příkazem, v jehož parametru se použijí. Příkazu bude naservírován až zpracovaný výsledek.

Příklad

Předpokládejme, že máme v pracovním adresáři soubory
`abc.txt`, `amx.jpg`, `bcd.gif`, `abdca.txt` a adresáře `alibaba`, `Mercedes`
a zadáme `ls a*`, ve skutečnosti se postupně provede tento příkaz:

```
ls abc.txt amx.jpg abdca.txt alibaba
```

přičemž u souborů se prostě vypíše jejich název a u adresáře se příkaz pokusí vypsat jeho obsah. K samotnému příkazu `ls` se hvězdička vůbec nedostane.

Pokud bychom v tomto případě zadali příkaz `echo a*`, shell by ho „přeložil“ na

```
echo abc.txt amx.jpg abdca.txt alibaba
```

takže důsledkem by prostě bylo vypsání těchto řetězců hned za sebou. Ovšem pro příkaz `echo` nemáme k dispozici další možnosti formátování výstupu ani rekurze.

Kdybychom chtěli v našem případě vypsat všechny položky, které začínají písmenem `b` nebo `M`, napsali bychom:

```
echo [bM]*
```

Bash tento příkaz „přeloží“ na

```
echo bcd.gif Mercedes
```

tedy budou na výstupu právě tyto dva řetězce.

Úkol

Zopakujte si toto učivo z předchozího semestru a procvičte si filtrování na názvech souborů a adresářů ve vašem domovském adresáři. Například vypište seznam všech položek, které začínají písmenem `P`, začínají tečkou, za kterou následuje písmeno `g`. Dále vypište všechny položky s názvem končícím písmenem

6.3.3 Prohledávání adresářové struktury

K prohledávání adresářové struktury (hledání souborů) používáme příkazy `find` a `locate`, k prohledávání obsahu souborů zase `grep` (viz dále). Program `grep` lze použít také k nalezení názvu souboru v zadaném adresáři – stačí na vstup programu poslat výpis adresáře. Pro nalezení cesty ke spustitelnému souboru (příkazu) používáme příkazy `whereis` a `which`.

Zde se podíváme na použití příkazu `find`. Hledáme soubory (a adresáře) se zadaným názvem nebo jinými parametry, příkaz také poskytuje možnost omezeného zpracování souborů nebo ovlivňování formátu vlastního výstupu. Jako první parametr se obvykle zadává místo, od kterého má začít vyhledávání. Pokud tento parametr nezadáme, dosadí se pracovní adresář. Můžeme také používat zástupné symboly (globes) – hvězdičku apod., a dokonce máme dvě možnosti:

- buď necháme zástupné symboly interpretovat shellem (podobně jako v předchozí sekci),
- nebo jejich interpretaci svěříme samotnému příkazu `find` (protože ten to na rozdíl od mnoha jiných příkazů umí).

Jestliže chceme použít druhou možnost, uzavřeme vyhledávací výraz do jednoduchých apostrofů (pozor – ne opačných), čímž zamezíme shellu v interpretaci zástupných symbolů.

Příklad

Projdeme si možnosti příkazu `find` postupně podle jednotlivých kritérií. Nejdřív se zaměříme na samotný hledaný řetězec, jeho formát a vypnutí rozlišování malých a velkých písmen.

`find *.txt -print` vypíše všechny soubory se zadanou příponou v pracovním adresáři, přičemž interpretaci zástupných symbolů necháváme na shellu, přepínač `-print` se dává celkem běžně, ale ve skutečnosti není až tak nutný (způsobí výpis celého názvu nalezeného souboru, což je ale výchozí chování, pokud nepoužijeme modifikátory výstupu)

`find -name '*.txt'` vypíše všechny soubory se zadanou příponou rekurzívně; jak vidíme, hvězdička je přenechána příkazu `find`, nikoliv shellu (apostrofy jsou nutné!)

`find /usr/xyz -name abcd` hledá soubor, který má v názvu řetězec *abcd*, v adresáři `/usr/xyz`, přepínač `-print` způsobí výpis úplného jména; rozlišují se malá a velká písmena (takže třeba pokud by v tom adresáři byl soubor, v jehož názvu je řetězec „AbCD“, nebyl by vypsán v nalezených)

`find ~ -iname 'desk*'` vypíše s rekurzí vše, co začíná řetězcem „desk“, bez rozlišování malých a velkých písmen (protože jsme použili parametr `-iname` místo `-name`)

Příkaz `find` umí výborně filtrovat podle různých kritérií, některá z nich si zde ukážeme (další najdete v manuálové stránce tohoto příkazu: `man find`).

Příklad

Můžeme vyhledávat například podle vlastníka či přidružené skupiny souboru (jak víme, každý soubor má svého vlastníka a přidruženou skupinu):

`find -user uzivatel` hledá soubor vlastníka *uzivatel* (prohledáváme pracovní adresář)

`find -uid 1520 -name prac*` hledáme soubor (rekurzívně), jehož název začíná zadaným řetězcem a jeho vlastník je uživatel se zadaným UID (lze také zadat GID, parametr `-gid`)

Také se dá vyhledávat podle typu nebo velikosti souboru:

`find -type d -name abcd` hledáme rekurzívně z pracovního adresáře všechny soubory, které jsou adresáři (parametr `-type d`) a v názvu mají řetězec `abcd`

`find -type f -name abcd` podobně, ale chceme běžné soubory (file), nikoliv adresáře (jako typ souboru je možné použít `d` pro adresář, `f` pro běžný soubor, `c` pro znakové zařízení, `b` pro blokové zařízení, `p` pro pojmenovanou rouru, `l` pro symbolický odkaz a `s` pro socket)

`find -type f,d,l -name abcd` zajímá nás víc různých druhů souborů

`find \( -type f -o -type d -o -type l \) -name abcd` totéž, jen trochu víc rozepsané

`find -type d -size +10k` vypíšou se soubory typu adresář (d), jejichž velikost je více než 10 kB

`find -type f -size -2M` vypíšou se běžné soubory (f), jejichž velikost je méně než 2 MB

`find /usr/bin /usr/sbin -executable` vypíšeme všechny soubory ze zadaných adresářů, které jsou spustitelné (týká se i skriptů coby textových spustitelných souborů)

Další kritérium je časové – vyhledáváme podle data/času změny (modifikace), posledního přístupu (včetně přístupu „pro čtení“), atd.:

`find -mtime 2` hledáme soubor, který byl modifikován (modified) naposledy před právě 2 dny (přesněji před 2*24 hodinami)

`find -mtime +2` hledáme soubor, který byl modifikován (modified) naposledy před více než 2 dny

`find -mtime -2` hledáme soubor, který byl modifikován (modified) naposledy před méně než 2 dny

`find -atime 3` k souboru bylo přistupováno (accessed) naposledy před 3 dny; obecně: vyhledáme vše, co kdy bylo modifikováno (m) či přistupováno (a), zde jde o dny, ale existují volby i pro minuty

`find ~ -mtime 0` najde vše v pracovním adresáři, co bylo modifikováno před méně než 24 hodinami (tj. uplynul méně než jeden den)

Každý soubor má přiřazena přístupová oprávnění pro vlastníka, přidruženou skupinu a „zbytek světa“. Také toto může být vyhledávací kritérium pro příkaz `find`.

Příklad

Můžeme vyhledávat jak podle „písmenného“ zadání oprávnění, tak i číselného módu:

`find -perm -u+w,g+w` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A ZÁROVEŇ právo zápisu pro skupinu (ostatní oprávnění nejsou testována)

`find -perm /u+w,g+w` chceme vypsat celé názvy souborů, v jejichž přístupových oprávněních je právo zápisu pro uživatele A/NEBO právo zápisu pro skupinu (ostatní oprávnění nejsou testována)

`find -perm /u=w,g=w` totéž

`find -perm -664` hledáme soubory, které mají nastavena oprávnění `rw-rw-r-`, ale pokud je nastaven i jiný bit (například některé `x` nebo `w` ve třetí části), taky bude takový soubor zahrnut

`find -perm /222` hledáme soubory, kde alespoň někdo má právo zápisu – vlastník, skupina nebo ostatní, nebo třeba všichni (ostatní oprávnění nejsou testována, klidně může být i právo čtení nebo spouštění)

`find -perm 664` hledáme soubory, které mají *přesně* nastaven řetězec oprávnění na `rw-rw-r-`, bude ignorovat soubory, které mají v řetězci oprávnění jakoukoliv odlišnost

`find / -perm -4000` vypíše celé názvy souborů (rekurzívně v kořenovém adresáři), které mají nastaven SUID bit (vzpomeňte si z předchozího semestru, co to znamená), SUID, SGID a Sticky bity se dají zadat pouze zadáním číselného módu souboru

`find / \( -perm -4000 -o -perm -2000 \) -type f -exec ls -la{} \;`
teď je příkaz o něco složitější; všímáme si všech souborů s nastaveným SUID *nebo* SGID bitem (takto zapisujeme disjunkci, závorky jsou nutné), má jít o běžné soubory (včetně spustitelných, ne adresáře), na konci je místo obyčejného výpisu volba na spuštění příkazu, v jehož výstupu zjistíme informace o souboru

Pokud před řetězec nebo číselnou reprezentaci oprávnění u `-perm` dáme

- *pomlčku*: stanovíme minimální požadovaná oprávnění (u souboru mohou být i další oprávnění, nebudou vadit), například `-444` by znamenalo, že vlastník, skupina i ostatní musí mít právo čtení,
- *lomítko*: alespoň některé z uvedených oprávnění je požadováno, nemusejí být všechna (a mohou být u souboru i další), například `/444` by znamenalo, že alespoň někdo musí mít právo čtení,
- *nic*: striktní, stanovíme naprosto přesná oprávnění, která musí hledaný soubor mít, například `644` by znamenalo, že všichni musí mít právo čtení a vlastník i právo zápisu, nikdo nesmí mít žádná další oprávnění.

Poznámka

Zejména způsob vyhledání všech souborů s nastaveným SUID bitem bychom si měli zapamatovat; jak víme, SUID bit je sice někdy nevyhnutelný, ale taky může být nebezpečný. Takže zjednodušeně (chceme jen názvy):

`find / -perm -4000`

Úkoly

1. Pomocí příkazu `find` vyhledejte všechny soubory, jichž jste vlastníky a jejichž název začíná tečkou. Dále vyhledejte všechny soubory ve svém domovském adresáři, ke kterým bylo (jakkoliv) přistupováno před méně než dvěma dny.
2. Pokud máte administrátorská přístupová oprávnění (třeba přes mechanismus sudo), použijte postup nalezení všech programů s nastaveným SUID bitem (je naznačen v poznámce).

Další příkaz sloužící k prohledávání adresářové struktury je `locate`. Pracuje na trochu jiném principu než `find` – obsah připojených zařízení se pravidelně indexuje (jednou denně) a vytváří se databáze položek v adresářové struktuře, a příkaz `locate` vyhledává právě v této databázi. Důsledkem je výrazně rychlejší vyhledávání. Nevýhodou je, že databáze nemusí být plně aktuální (pak ji ale můžeme aktualizovat ručně spuštěním příkazu `updatedb`, k tomu potřebujeme administrátorská oprávnění).

Vzhledem k vlastnostem tohoto příkazu je výhodné používat `locate` v případech, kdy hledáme soubor, o kterém sice víme, jak se jmenuje, ale nemůžeme si vzpomenout, kde se nachází, přičemž tento soubor existuje víc než jeden den. Příkaz pracuje pouze s těmi položkami, ke kterým máme přístupová oprávnění, takže nejsme obtěžováni upozorněními typu „Přístup zamítnut“.

Příklad

Pokud hledáme soubor `faktury` s nějakou příponou (nevíme jistě, jakou jsme souboru dali) a nevíme, kde hledat, napíšeme:

```
locate faktury.*
```

Vypíše se seznam souborů s plnou cestou, jejichž název odpovídá zadanému schématu. Prohledává se opravdu celá indexovaná adresářová struktura, nikoliv jen z pracovního adresáře.

Hledaný řetězec může být i v cestě k hledanému objektu, takže pokud například chceme seznam dynamických knihoven s moduly programu Gimp, napíšeme

```
locate *gimp*modules*
```

vypíšou se nám řádky s moduly (knihovnami s příponou .so) a pár dalších řádků.

Konfigurace mechanismu `locate` je v souboru `/etc/updatedb.conf`. Také tento soubor je přístupný pouze s administrátorskými oprávněními.

Úkoly

1. Vyzkoušejte si mechanismus `locate`. Pokud máte nainstalován Gimp, vyzkoušejte postup naznačený v předchozím příkladu (ten s moduly). Dále vyhledejte všechny soubory s názvem `passwd`.
2. Porovnejte chování těchto dvou příkazů (doporučuji co nejvíc rozšířit okno terminálu):

   ```
   locate '*.jpg'
   find / -name '*.jpg'
   ```

   Všimněte si také, jak příkaz (ne)reaguje v případě nedostupných položek.

Pokud chceme vyhledat spustitelný soubor konkrétního příkazu, použijeme jeden z těchto příkazů: `whereis` nebo `which`. První z nich vyhledá opravdu všechny výskyty souboru s tímto názvem (tj. nejen spustitelné soubory, ale i třeba stejně pojmenované konfigurační soubory, manuálové stránky, atd.) –

takže se ptáme „Kde se nachází soubor s tímto názvem?“, druhý se zaměřuje opravdu jen na spustitelné soubory.

Úkol

Vyzkoušejte příkazy `whereis` a `which` s parametrem `passwd`, srovnejte výstup. Dále tyto příkazy vyzkoušejte na jiných příkazech – například `locate`, `ls`, `cut`, `shutdown`, `man`.

6.3.4 Prohledávání textů

Když chceme prohledávat textový soubor nebo třeba textový výstup jiného programu, použijeme příkaz `grep`, případně s přepínačem `-E`. Od předchozích příkazů se odlišuje nejen tím, že prohledává texty (nikoliv strukturu souborů), ale i tím, že dokáže používat plnohodnotné regulární výrazy.

Přepínač `-E` zapíná rozšířené reg. výrazy, často bývá defaultní. Další důležité přepínače:

`-i` nerozlišuje malá a velká písmena

`-l` pouze vypíše názvy souborů, ve kterých nalezl shodu

`-n` vypíše také číslo řádku

`-r` rekurzívně zpracovává i podadresáře

`-o` vypíše jen nalezený řetězec, ne celý řádek

`-c` v souborech pouze spočítá výskyty nalezeného řetězce

V regulárním výrazu můžeme používat symboly shrnuté v tabulce 6.1.

Tabulka 6.1: Možnosti pro regulární výrazy v příkazu `grep`

Prvek	Význam
`.`	Libovolný znak
`?`	Nula nebo jeden výskyt předcházejícího řetězce
`*`	Nula nebo více výskytů předcházejícího řetězce
`+`	Jeden nebo více výskytů předcházejícího řetězce
`{m}`	`m` opakování předcházejícího řetězce
`{m,n}`	`m` až `n` opakování předcházejícího řetězce
`{m,}`	`m` nebo více opakování předcházejícího řetězce; `a{0,}` je totéž jako `a*`
`^`	Začátek řádku
`$`	Konec řádku
`[třída]`	Jakýkoli (jeden) znak z množiny v závorkách
`[^třída]`	Jakýkoli znak mimo prvky množiny
`[x-y]`	Jakékoli znaky v daném rozsahu (jeden)
`r1 | r2`	logické nebo – buď řetězec `r1`, nebo řetězec `r2`
`(výraz)`	uzávorkování výrazu; metaznaky `*`, `?` apod. za závorkou platí pro celou závorku

Rozšířené (extended) regulární výrazy (tj. pokud použijeme přepínač `-E` nebo rovnou příkaz `egrep`) berou všechny symboly uvedené v tabulce 6.1 jako speciální symboly, a pokud je chceme speciálního významu zbavit, použijeme zpětné lomítko. Například `*` má speciální význam: určuje, že to, co je před ní uvedeno, se může opakovat jakýkolivpočetkrát, ale pokud chceme v textu prostě vyhledat hvězdičku, zadáme do výrazu `\*`.

Základní regulární výrazy příkazu `grep` to mají naopak – téměř všechny symboly z tabulky 6.1 (kromě tečky, hranatých závorek `[]`, stříšky `^` a dolarovky `$`) ve výchozím stavu nemají speciální význam, ale pokud jim ho chceme dát, musíme před ně napsat zpětné lomítko.

Symboly tečka hranaté závorky, stříška, dolarovka mají ve výchozím stavu speciální význam u základních i rozšířených výrazů; v obou případech například `[a-z]` je množina malých písmen.

Poznámka

Jaký je rozdíl mezi jednoduchými výrazy se zástupnými symboly, kterým rozumí programy typu `bash`, `find` apod., a regulárními výrazy v programu `grep` (ať už základními nebo rozšířenými)? U plnohodnotného regulárního výrazu:

- symboly hvězdička, plus a otazník se vztahují k předchozímu symbolu/množině/podvýrazu, nikoliv k „čemukoliv“, navíc můžeme určit číslem/intervalem počet opakování předchozího prvku,
- můžeme vyjádřit fakt, že hledáme řetězce na začátku či konci řádku, začátku či konci slova,
- dá se vyjádřit disjunkce či konjunkce podvýrazů,
- pro jeden výskyt jakéhokoliv znaku se používá symbol tečka, nikoliv otazník, kdežto otazník znamená jeden nebo žádný výskyt předchozího uvedeného podvýrazu,
- negace uvnitř množiny se nevyjadřuje vykřičníkem, ale stříškou, přičemž stříška se používá také k indikaci začátku řádku.

Posix sekvence jsou jakési „zkratky“ pro konkrétní typy znaků. Například místo `0-9`, můžeme napsat posix sekvenci `[:digit:]`. Další možnosti najdeme v tabulce 6.2. Protože jsou posix sekvence uzavřeny do hranatých závorek, pak při použití v regulárním výrazu jsou tyto závorky „zdvojeny“ – vnitřní patří posix sekvenci, vnější uzavírají množinu, ze které se vybírá prvek.

Tabulka 6.2: Posix sekvence při vyhledávání

Sekvence	Význam
`[:digit:]`	číslice
`[:xdigit:]`	hexadecimální číslice
`[:alpha:]`	písmena
`[:alnum:]`	písmena a číslice
`[:lower:]`	malá písmena
`[:upper:]`	velká písmena
`[:blank:]`	mezera a tabulátor
`[:space:]`	prázdné znaky (mezera, tabulátor, konec řádku, ...)
`[:graph:]`	viditelné znaky
`[:print:]`	viditelné znaky a mezera

Příklad

Ukážeme si použití příkazu `grep`.

`grep -i "vypis" *.txt` hledáme v souborech s příponou `.txt` slovo `vypis` bez rozlišování malých a velkých písmen

`grep -il "vypis" *.txt` hledáme v souborech s příponou `.txt` slovo `vypis` bez rozlišování malých a velkých písmen, pouze vypíše názvy souborů, ve kterých se toto slovo nachází

`grep -cr "#include.*\.[hc]" *.c` u každého souboru s příponou `.c` vypíše počet vkládaných souborů s příponou `.h` nebo `.c`, a protože tečka je významový znak (určuje jeden jakýkoliv symbol), musíme před ni dát zpětné lomítko, aby byla brána jako součást řetězce

`grep -c "Dear \(Mr.\|Ms.\|Miss\)" *.txt` chceme počet anglických oslovení v zadaných souborech

`grep -Ec "Dear (Mr.|Ms.|Miss)" *.txt` totéž, ale pomocí rozšířených výrazů

`egrep -c "Dear (Mr.|Ms.|Miss)" *.txt` totéž

`grep '[0-9]\{6\}/[0-9]\{3,4\}' soubor.txt` v souboru hledáme rodná čísla ve tvaru 123456/1234 – nejdřív 6 číslic, pak lomítko a tři nebo čtyři číslice; jinak:

```
grep '[[:digit:]]\{6\}/[[:digit:]]\{3,4\}' soubor.txt
```

`grep '[[:digit:]]\{1,3\}\(\.[[:digit:]]\{1,3\}\)\{3\}' soubor.log` v zadaném souboru hledáme všechny IP adresy ve tvaru 123.123.123.123 – čtyři skupiny číslic, v každé skupině 1 až 3 číslice; jinak:

```
grep '[0-9]\{1,3\}\(\.[0-9]\{1,3\}\)\{3\}' soubor.log
egrep '[0-9]{1,3}(\.[0-9]{1,3}){3}' soubor.log
```

Poznámka

V příkazech si můžeme povšimnout, že je nutno rozlišit vyhledávané znaky a „metaznaky“, které v regulárním výrazu slouží k upřesnění vyhledávacího řetězce (například v posledním příkazu kulaté závorky). Kdybychom před symboly kulatých a složených závorek nepsali zpětné lomítko, shell by je „sebral“ a pokusil je interpretovat.

Program `grep` je pružnější než `find` (už pro rozsáhlé možnosti regulárních výrazů), proto je často používán nejen k prohledávání textových souborů, ale také k prohledávání struktury adresářů v kombinaci s příkazem `ls`.

Příklad

Ukážeme si možnosti prohledávání adresářové struktury příkazem `grep`:

`ls -la | grep -ic "^-"` vypíše počet běžných souborů v pracovním adresáři

`ls -la | grep "^-..x"` vypíše všechny běžné soubory (řádek začíná pomlčkou), které jsou spustitelné (vlastník má právo spouštění)

Příklad

Ukážeme si, jakým způsobem vypíšeme login shell některého uživatele. Předpokládejme, že chceme znát shell uživatele novak. Pak zadáme:

```
cat /etc/passwd | grep "^novak:" | cut -d : -f 7
```

Vypadá to trochu zvláštně. V koloně (rouře) je první příkaz `cat`, který na svůj výstup pošle obsah souboru `/etc/passwd`, což je soubor se seznamem uživatelů a jejich parametrů. Další na řadě je příkaz `grep`, který vybere pouze řádek obsahující na svém začátku jméno hledaného uživatele.

Třetím příkazem v koloně je `cut` (pozor, neplést si s `cat`), který „usekne“ řádek a zobrazí pouze jeho konec. Jeho první volba, `-d`, stanoví dvojtečku na oddělovač „sloupců“. Druhý parametr stanoví, že chceme pouze sedmé pole v pořadí (pole jsou oddělena oddělovačem zadaným v předchozí volbě).

Úkoly

1. Zaměřte se na soubor `.bashrc` ve vašem domovském adresáři (nebo některý jiný konfigurační soubor). Nejdřív vypište všechny řádky tohoto souboru, které začínají symbolem `#` (tj. hledaný řetězec je na začátku řádku). Pak naopak vypište všechny řádky souboru, které nezačínají tímto symbolem (tj. potřebujete vědět, jak určit, že řetězec je na začátku řádku, a navíc potřebujete negovat – bude to negace jednoprvkové množiny).
2. Najděte ve svém domovském adresáři všechny soubory, které obsahují řetězec (dvojznak) `#!`.
3. Ve výpisu domovského adresáře najděte všechny položky, jejichž název obsahuje řetězec `rc`.
4. V tabulce 6.1 stojí, že `x*` se dá zapsat jako `x{0,}`. Jak podobným způsobem zapíšete `x+` a jak `x?`?
5. Vypište počet běžných souborů ze svého domovského adresáře, které jsou skryté (začínají tečkou).
6. Vypište seznam všech symbolických odkazů ve svém domovském adresáři. Využijte vhodný regulární výraz na výstup příkazu `ls -l`.

6.4 Speciální znaky pro uvozování

Různé typy uvozovek a dalších symbolů mohou měnit způsob zacházení s řetězcem.

Apostrofy určují, že vše, co je mezi nimi, tvoří jednolitý celek, i kdybychom mezitím klepli na klávesu Enter. Znaky mezi apostrofy jsou brány jako obyčejný text, zamezuje se interpretaci čehokoliv (zástupných symbolů, proměnných, aliasů apod.). Jsou používány například v těchto případech:

- v příkazu `echo` (nebo jiném) chceme text umístit do více řádků, pak na prvním řádku (který začíná příslušným příkazem) je jen levý apostrof, pravý je na posledním řádku,
- názvy souborů obsahující mezery uzavíráme do apostrofů,
- `find / -name '*' -print` (zde zabraňujeme interpretaci hvězdičky shellem; vyzkoušejte i variantu bez apostrofů, obdržíte chybové hlášení)
- atd., setkali jsme se s nimi u regulárních výrazů, kde určovaly hranice výrazu a zamezovaly předčasné interpretaci, setkáme se s nimi také například u proměnných.

Obrácené apostrofy naopak způsobují okamžitou interpretaci svého obsahu. Tedy obsah mezi obrácenými apostrofy je nahrazen svým výstupem. Takto lze vložit výstup jednoho příkazu do parametru jiného příkazu.

Příklad

Jaký bude výstup těchto příkazů?

- `echo Datum je anglicky date`
- `echo Datum je anglicky ‘date‘`

V prvním případě se na obrazovku vypíše prostě to, co je v paramtru příkazu `echo`, v druhém případě se však řetězec `date` interpretuje (bude považován za příkaz, nikoliv za obyčejný řetězec) a místo něj se na výstupu dosadí momentální datum a čas.

Příklad

Příkaz `expr` slouží k výpočtu matematických výrazů. Srovnejme (pozor, mezery kolem operátoru jsou povinné!!!):

Příkaz	Výstup příkazu
`expr 1 + 1`	`2`
`echo Výpočet: 'expr 1 + 1'`	`Výpočet: expr 1 + 1`
`echo Výpočet: ‘expr 1 + 1‘`	`Výpočet: 2`

Výstup prvního příkazu je číslo 2 (tento příkaz vyhodnotí svůj argument jako aritmetický výraz). V druhém případě se pouze vypíše celý řetězec, nic nebude vypočteno. V třetím případě jsme do parametru příkazu `echo` vložili příkaz pro výpočet, a to v obrácených apostrofech, tedy nejdřív je proveden vnitřní příkaz, pak je jeho výsledek (číslo 2) vložen na místo tohoto příkazu a teprve potom je vypsán celý řetězec.

Zpětné lomítko se na rozdíl od ostatních uvozovacích symbolů píše pouze před symbol, ne za něj. Zpětná lomítka používáme k přepnutí významu následujícího prvku (tj. buď vypnutí nebo zapnutí, opačně vzhledem k původnímu nastavení).

S některými způsoby použití jsme se už setkali, například u regulárních výrazů nebo u escape sekvencí v příkazu `echo`. Zde je to například `\n`, tedy výsledek není chápán jako písmeno „n“, ale escape sekvence přechodu na nový řádek.

Zpětné lomítko také umístíme na konec řádku (opravdu na konec řádku, nesmí za ním být třeba mezera), pokud má příkaz pokračovat na následujícím řádku – pak je jeho účelem přepínat speciální význam symbolu pro konec řádku (už to nebude speciální symbol vynucující zařádkování).

Příklad

```
echo první řádek\
>  druhý řádek \
>  třetí řádek
```

vypíše

```
první řádekdruhý řádek třetí řádek
```

(všimněte si chybějící mezery; tato mezera chybí, protože jsme ji do původního příkazu před zpětné lomítko nenapsali).

Uvozovky vypínají speciální význam zástupných symbolů (takže například hvězdička uzavřená do uvozovek ztrácí speciální význam a stává se pouhým znakem), ale přitom zachovávají význam obrácených apostrofů, proměnných a znaků uvozených zpětným lomítkem. Uvozovky využijeme také jako náhradu běžných apostrofů, pokud je chceme vnořit (běžné apostrofy nelze vnořovat, proto místo „vnějších apostrofů“ použijeme uvozovky).

Pozor, uvozovky a apostrofy mají svůj význam (vypnutí speciálního významu) pro daný shell – často je používáme tehdy, když chceme řetězec „uchránit“ před zásahem shellu, aby byly v původním tvaru bez interpretace předány jako parametr příkazu. Samotný příkaz už si s řetězcem samozřejmě může dělat co chce.

Úkol

Využijte obrácené apostrofy k výpisu informací o povolených shellech, vyzkoušejte:

```
cat /etc/shells
ls -la `cat /etc/shells`
```

Ve výpisu také zjistíte případné symbolické odkazy, které jedním názvem shellu odkazují na jiný shell.

6.5 Proměnné

6.5.1 Opakování práce s proměnnými

Proměnné se definují v souboru `.bashrc` nebo `.profile` nebo jsou či v jiném konfiguračním souboru (včetně těch systémových). Proměnnou vytváříme standardně přiřazením, například napíšeme `prom=abc` a tak vytvoříme proměnnou `prom` včetně incializace. Pak ale vzniká *lokální proměnná* platná jen v nejbližším okolí (například shellu, funkci).

Pokud chceme proměnnou *exportovat do prostředí* (tedy zveřejnit do všech procesů, které byly z daného shellu vytvořeny), použijeme příkaz `export` (parametrem bude příslušná proměnná), případně můžeme spojit vytvoření proměnné a export do jediného příkazu:

```
export PROM=abc
```

čímž rovnou vytvoříme proměnnou, která bude přístupná i mimo lokální určení ve všech procesech, jejichž (pra)rodičem je shell, v němž pracujeme.

Je zvykem, že proměnné, které chceme exportovat do prostředí, se zapisují velkými písmeny a proměnné, které mají zůstat lokální, malými písmeny.

Nejdůležitější proměnné jsou:

`HOME` domovský adresář uživatele

`TERM` typ terminálu

`SHELL` cesta k používanému shellu

`USER` přihlašovací jméno uživatele

`PATH` seznam adresářů, ve kterých se hledá spouštěný soubor, jednotlivé cesty jsou odděleny dvojtečkou

`PS1` prompt, výzva příkazového řádku konzoly nebo terminálu

`PWD` pracovní adresář

Poznámka

V proměnné `PATH` není ve výchozím nastavení cesta k pracovnímu adresáři a při spuštění programu také na rozdíl od Windows není spouštěný program hledán v pracovním adresáři! Proto pokud chceme spustit například program `mujprogram` umístěný v adresáři, který je právě naším pracovním adresářem, provedeme to takto:

```
./mujprogram
```

Teoreticky by se tento „problém“ dal vyřešit přidáním adresáře „`.`“ do proměnné `PATH`, ale toto řešení se z bezpečnostních důvodů nedoporučuje. Představte si, že ve vašem domovském adresáři (který velmi často bývá pracovním adresářem) bude podstrčen škodlivý software pojmenovaný stejně jako některý z běžně používaných příkazů, nejlépe příkazu obvykle se nacházejícího v adresáři `/sbin`, který často nebývá zahrnut v proměnné `PATH`. Pokud zařadíme do proměnné `PATH` tečku, můžeme tento škodlivý software omylem spustit (a pokud je jeho programátor dost chytrý na to, aby po dokončení své vlastní činnosti spustil původní program na správném umístění, tak to ani nezjistíme). Mohlo by se zdát, že stačí tečku přidat až na konec proměnné `PATH`, aby všechny dříve uvedené adresáře měly přednost, ale toto nefunguje, pokud některé jinak důležité cesty nejsou v této proměnné zahrnuty.

Existuje také obdoba rozdělení proměnných na běžné a dynamické, s čímž jsme se setkali už ve Windows. Například proměnná `PS1` je dynamická.

Příkazy pro práci s proměnnými:

`echo $proměnná`
: vypíše obsah proměnné (vyhodnotí ji), pracuje s běžnými i dynamickými proměnnými

`proměnná=výraz`
: změní obsah proměnné

`export proměnná`
: exportuje proměnnou do prostředí – podřízených procesů (aby ji mohly využívat všechny skripty i příkazový režim), dá se spojit s přiřazením hodnoty do proměnné

`env`
: vypíše proměnné s jejich obsahem (proměnné prostředí)

`set`
: vypíše veškeré proměnné a funkce, které jsou definovány v dané oblasti, ve které pracujeme (výstup je poměrně rozsáhlý)

`unset proměnná`
: odstraní proměnnou

`read proměnná`
: načte ze standardního vstupu řetězec do proměnné

Znak $ vyhodnotí vše, co se za ním nachází (vše až po první mezeru nebo konec řádku považuje za název zpracovávané proměnné), proto když zadáváme za názvem proměnné ještě něco dalšího, uzavřeme tuto proměnnou do lomených závorek:

```
export PATH=${PATH}:/usr/TeX/bin
```

Když chceme, aby se takováto „vnitřní proměnná“ vyhodnocovala při každém volání „vnější proměnné“, uzavřeme výraz do jednoduchých uvozovek (apostrofů).

Příklad

Předpokládejme, že někde třeba ve skriptu používáme proměnnou. Nejdřív ji vytvoříme a inicializujeme, pak vypíšeme její obsah. Pro výpis obsahu proměnné použijeme různé způsoby uvození.

Příkaz	Výstup příkazu	Komentář
`prom="A B  C"`		jen jsme vytvořili proměnnou
`echo $prom`	`A B C`	vypsali jsme obsah proměnné, ale pozor – mezery
`echo "$prom"`	`A B  C`	mezery zachovány
`echo '$prom'`	`$prom`	apostrofy zamezují interpretaci čehokoliv, proto berou speciální význam symbolu `$`
`echo ‘$prom‘`	`A: příkaz nebyl nalezen`	zpětné apostrofy naopak vynucují interpretaci u čehokoliv, tedy po dosazení obsahu proměnné se shell marně pokouší interpretovat řetězec `A B C` jako příkaz `A` s parametry `B` a `C`
`echo \$prom`	`$prom`	vzali jsme speciální význam symbolu `$`

Všimněte si důsledku použití uvozovek v parametru příkazu `echo`. Výstup je vpodstatě podobný jako u předchozího příkazu (tj. bez uvozovek), ale zabránili jsme shellu v interpretaci mezer coby oddělovačů podřetězců, tedy aby sekvenci oddělovačů nezjednodušil na jediný. Nicméně uvozovky nezabránily interpretaci proměnné.

Příklad

Do proměnné nemusíme ukládat jen řetězec nebo číslo. Jak víme, obrácené apostrofy jsou prostředkem, jak vynutit vyhodnocení výrazu, přičemž nám nic nebrání vnořit je do uvozovek. Například:

```
prom="Výstup: ; ‘ls -la‘"
echo $prom
echo "$prom"
```

V prvním příkazu jsme do proměnné uložili příkaz (vlastně dva příkazy oddělené středníkem, aby mohly být na jednom řádku) k interpretaci. Obrácené apostrofy uvnitř uvozovek jsou zde nutné.

V dalších příkazech vypisujeme obsah proměnné, čímž vyvoláme interpretaci příkazů v ní uložených. Vyzkoušením si u posledního příkazu opět můžeme ověřit význam uvozovek.

Příklad

Další ukázky s proměnnými:

`export PS1='$PWD'` do proměnné `PS1`, tj. do momentálního promptu, si uložíme pracovní adresář

`PS1="\W"` pracovní adresář bez cesty, s cestou je `\w` (a také zobrazuje domovský adresář jako vlnovku)

`PS1="\d"` promptem bude aktuální datum

`PS1="\A"` promptem je aktuální čas – jen hodiny a minuty, jiné možnosti pro čas jsou `\t` nebo `\T`

`PS1="\u\h\$ "` prompt je ve formě uživatel počítač`$` (plus mezera)

`PS1="\u\h:\w> "` prompt je ve formě uživatel počítač: pracovní adresář> (plus mezera)

`PS1='pracuji tak dlouho: $SECONDS'` opět změníme prompt, proměnná `SECONDS` je dynamická

`mkdir $HOME/novy` využili jsme proměnnou uvnitř parametru příkazu; všimněte si složených závorek kolem názvu proměnné, zde jsou nutné

Kromě proměnné `PS1` existují obvykle ještě další proměnné určující prompt. Většinou se setkáme alespoň se sekundárním promptem `PS2`, který je zobrazován například při psaní víceřádkových příkazů, případně při vytváření here documents.

Příkaz `set` je ve skutečnosti mnohem silnější, nejde jen o vypisování seznamu proměnných a funkcí. Můžeme například do proměnné přiřadit výsledek funkce (podrobnosti najdeme v manuálové stránce příkazu `set`).

Úkoly

1. Vyzkoušejte příkazy vypisující proměnné. Vyberte si kteroukoliv proměnnou a vypište její obsah.
2. Vyzkoušejte postupy ukázané v příkladech v této sekci (pokud ovšem nejsou destruktivní).

6.5.2 Výpočty

Používání proměnných úzce souvisí s prováděním výpočtů. Pro tyto účely máme sice rovnou dva příkazy: `let` a `expr`, ale ne vždy jsou nainstalovány (nicméně je lze doinstalovat), tak existuje i „závorkový“ (obecnější) postup, který je nejoblíbenější (není nutno pamatovat si příkaz).

Příklad

Příkaz `let` slouží ke *změně hodnot proměnných* s vyhodnocením výrazu, v některých případech jsou uvozovky nutné. Ukážeme si, jak se používá.

`let "prom=3+4"` do proměnné je uložen výsledek výrazu

`let "prom+=4"` je podporována „céčkovská“ notace přiřazování

`let "prom+=$prom"` na pravé straně může být také proměnná, ale samozřejmě příslušně označená symbolem dolaru, aby byla interpretována

`let "prom-"` snížení hodnoty proměnné o 1

Příklad

Příkaz `expr` slouží k podobnému účelu (ovšem nemusíme výsledek ukládat do proměnné), jen si musíme dávat pozor, abychom kolem operátorů měli mezery. Opět pár ukázek použití:

`prom=‘expr 3 + 4‘` nejdřív vypočte výraz, pak výsledek přiřadí do proměnné (kolem rovnítka nesmí být mezery)

`expr $prom + 1` vypíše číslo o 1 větší než je hodnota proměnné `$prom` (tj. dosadí za proměnnou, vypočte výraz a vypíše)

Výsledek si můžeme vždy ověřit vypsáním `echo $prom`.

Pro označení řetězce za matematický výraz a vynucení jeho interpretace se používají dvojité kulaté závorky. Pokud se tato konstrukce objeví za rovnítkem nebo kdekoliv jinde, kde může být i něco jiného než výpočet, předsadíme před levou dvojzávorku symbol dolarovky.

Příklad

Ukážeme si počítání s využitím dvojitých závorek. Syntaxe samotného výrazu odpovídá céčkovské syntaxi.

```
prom=2
(( prom+=8 ))
echo $prom
```

Pozor – kdybychom v druhém příkazu nepoužili dvojité závorky, bylo by to vpodstatě správně, ale dostali bychom špatný výsledek – pracovalo by se totiž s řetězci (zřetězili bychom „2“ a „8“, výsledek by byl „28“ místo čísla 10). Mezery uvnitř dvojzávorek jsou nepovinné (ale nejsou chybou), zde je jejich úlohou jen zvýšení přehlednosti.

```
x=$(( prom*2 ))
echo Výsledek výpočtu 259-prom/4 je $(( 259-prom/4 ))
```

Takže dvojité závorky značí, že s tím, co je uvnitř, se má zacházet jako s matematickým výrazem. Pokud se tam objeví řetězec něčeho jiného než číslic, bude s ním automaticky zacházeno jako s proměnnou, nemusíme před název proměnné dávat dolarovku. Vlastně mají podobnou roli jako zpětné apostrofy.

Příklad

Hodnotu proměnné lze načíst i od uživatele (proměnná, do které načteme vstup od uživatele, nemusí předem existovat, lze ji vytvořit i voláním funkce `read`):

```
echo -n "Zadej číslo: "
read prom1
echo -n "Byla načtena hodnota ${prom1}"
prom2=$prom1
```

Všimněte si závorek kolem názvu proměnné. Zde nejsou přímo nutné, ale je dobré si na ně zvyknout, protože v určitých případech jsou naopak nevyhnutelné. Celý vypisovaný řetězec také nemusí být v uvozovkách, v tomto případě.

Na následujících výrazech si ve výsledku všimněte rozdílu, když použijeme dvojité závorky:

```
prom1+=80
(( prom2+=80 ))
echo prom1=${prom1}, prom2=${prom2}.
```

Úkoly

1. Proveďte příkazy ze dvou posledních příkladů. Zjistěte, jaký je rozdíl ve výstupech příkazů, kde se k proměnné přičítá číslo 80. Ve kterém případě se s proměnnou zachází vždy jako s řetězcovou?
2. Vyzkoušejte výpočty, kde je na pravé straně výrazu proměnná. Načtěte od uživatele hodnotu proměnné a vypište její dvojnásobek.

6.6 Skripty

6.6.1 Co je to skript

Skript je textový spustitelný soubory, který pro své spuštění potřebuje interpret (třeba některý shell). Soubory se skripty obvykle mívají buď příponu SH, anebo jsou bez přípony.

Každý skript je psán vždy pro určitý shell nebo programovací jazyk (například `bash` nebo `perl`) a v jiném nemusí fungovat. Proto často bývá na prvním řádku skriptu *identifikace shellu*. Tento řádek vždy začíná dvojicí znaků `#!`, pokud ovšem se ve skriptu nachází.

Příklad

První řádek skriptu může vypadat třeba takto:

- `#!/usr/bin/tcsh` znamená, že skript má být interpretován shellem Toronto C Shell,
- `#!/usr/bin/bash` určuje skript s příkazy shellu Bourne Again Shell,
- `#!/usr/bin/perl` (adresa může být jiná, záleží na umístění spustitelného programu `perl`) je skript psaný v programovacím jazyce Perl.

Aby bylo možné skript spouštět v textovém režimu napsáním jeho názvu, musí být označen jako spustitelný, tj. je třeba nastavit příznak `x` v přístupových oprávněních. V opačném případě musíme skript spouštět jako parametr interpretačního programu (například `bash soubor.sh`), ale i tak je vhodné tento soubor označit jako spustitelný nastavením příslušného příznaku.

Příklad

Vytvoříme jednoduchý skript, ve kterém bude jediný příkaz. Nejdřív vytvoříme nový soubor, který pojmenujeme třeba `testik.sh`. Do tohoto souboru uložíme následující řádky:

```
#!/usr/bin/bash
echo Toto je muj prvni skript
```

Dále předpokládejme, že tento soubor se nachází v našem pracovním adresáři. Skript už nyní můžeme spustit tak, že jeho název předáme jako parametr interpretačnímu programu:

```
bash testik.sh
```

Jenže my chceme, abychom mohli skript spouštět i jako samostatný spustitelný soubor. Pak musíme nastavit oprávnění „`x`“, třeba v grafickém režimu. V textovém režimu by se to provedlo takto (teď trochu předbíháme) – platilo by pro všechny uživatele:

```
chmod +x ./testik.sh
```

Soubor bychom pak spustili takto:

```
./testik.sh
```

Tečka a lomítko jsou nutné, jak už jsme si vysvětlili v sekci o proměnných, konkrétně se to týká vlastností proměnné `PATH`.

Pokud nastavíme přístupové oprávnění „`x`“, můžeme skript spouštět i poklepáním myší v grafickém režimu (ovšem musíme počítat s tím, že po ukončení se okno se skriptem zavře a nebude možné si přečíst případné výstupy).

Skripty můžeme použít pro automatické zálohování, shromažďování informací, přidávání nových uživatelů a obecně jakoukoliv automatizaci posloupnosti operací.

Úkol

Vytvořte jednoduchý skript vypisující řetězec, datum apod. (případně podle své fantazie využijte některé nedestruktivní příkazy z těch, které jsme už probírali). Zajistěte, aby bylo možné tento skript spustit tak, jak je ukázáno v příkladu.

6.6.2 Parametry a návratové hodnoty

Parametry skriptu jsou přístupné přes proměnné `$0` (název skriptu), `$1` (první parametr), `$2` (druhý parametr), ..., `$9`. S těmito parametry zacházíme naprosto stejně jako s běžnými proměnnými, tedy pokud například chceme vypsat na obrazovku obsah druhého parametru, zadáme

```
echo Druhý zadaný parametr skriptu je: $2
```

Pokud bychom chtěli „ještě něco dodat“, uzavřeme číslo parametru do složených závorek:

```
echo Ve čtvrtém parametru je číslo znamenající násobnost: ${4}krát větší náklady...
```

Pokud potřebujeme přistupovat k více než devíti parametrům, pak použijeme příkaz `shift` posouvající obsah parametrů o jeden krok doleva (takže pod `$9` pak bude dostupné to, co bylo původně v desátém parametru, apod., posouvá se obsah všech parametrů). Jako nepovinný parametr můžeme použít číslo označující počet kroků posunutí, implicitně je to 1.

Skript může také vracet *návratový kód*, a to příkazem `exit`. Můžeme jako nepovinný argument zadat číslo návratového kódu, výchozí je číslo 0. Návratová hodnota nemusí být konstantní číslo, můžeme takto předat třeba i obsah proměnné.

`exit` konec skriptu, vracíme hodnotu 0

`exit 8` konec skriptu, vracíme hodnotu 8

`exit $prom` konec skriptu, vracíme hodnotu obsaženou v zadané proměnné

Obdobou proměnné `errorlevel` z Windows je v Unixu proměnná `$?`. Obsahuje status (stav) posledního spouštěného příkazu, což samozřejmě může být i skript.

Běžně používané návratové hodnoty (chybové kódy) jsou 0 (proces proběhl úspěšně), 1 (proces se spustil, ale proběhl neúspěšně – nastaly chyby při běhu), 2 (proces se nespustil), 127 (proces nebylo možno spustit, protože program nebyl nalezen).

Příklad

Trochu předběhneme a použijeme cyklus until pro procházení všemi parametry, které uživatel při spuštění našeho skriptu použil. Soubor se skriptem obsahuje tento kód:

```
until [ -z "$1" ] ; do      # "z" jako "zero" - provádí se tak dlouho, dokud
                            # nebude splněna podmínka "první parametr je prázdný"
  echo -n "$1 "             # vypíšeme první parametr
  shift                     # posun, do prvního parametru se dostane obsah druhého apod.
done                        # konec cyklu
echo                        # zařádkování na konec (prázdný řádek)
```

Předpokládejme, že tento náš skript je uložen v souboru s názvem `pokus.sh` a uživatel ho spustil s těmito parametry:

```
./pokus.sh kalendar auto 54 posledni
```

Pak výstupem bude následující:

```
kalendar auto 54 posledni
```

V cyklu sice vypisujeme pořád první parametr (`$1`), ale díky příkazu posunu se do tohoto parametru postupně přesouvají všechny následující. Cyklus končí tehdy, když je v prvním parametru prázdný řetězec, tj. další parametr již není zadán.

Mohli bychom použít také cyklus `for` (viz dále) – počet použitých (plných) parametrů je totiž uložen v proměnné `$#`.

Úkol

Napište jednoduchý skript, ve kterém budete počítat se třemi parametry. Tyto tři parametry postupně vypište na výstup.

6.6.3 Další možnosti použití skriptů

Na prvním řádku skriptu bývá uveden příkaz spouštějící shell, který má provést interpretaci následujících řádků. Ve skutečnosti tam může být téměř kterýkoliv příkaz, kterému je pak přesměrován zbývající obsah souboru jako vstup.

Nahrazení shellu následujícím skriptem způsobí, že pokud se uživatel, pro kterého byla změna provedena, pokusí přihlásit do systému, vypíše se mu uvedená hláška a uživateli bude odmítnut přístup.

```
#!/usr/bin/tail +2
Pozor, tvůj účet byl zablokován z důvodu ...
Pokud chceš vše napravit, ...
```

Samotné nahrazení shellu se provede příkazem

```
chsh -s /.../soubor_skriptu uživatel
```

6.7 Podmínky, větvení a cykly

6.7.1 Jednoduché propojení příkazů

V UNIXu se setkáváme s podobnými možnostmi propojení příkazů, jaké známe z Windows (ostatně, odkud se to všechno ve Windows asi vzalo). Nejdřív jednoduché možnosti propojení příkazů:

`příkaz1 ; příkaz2 ; příkaz3` sekvenční provádění příkazů, totéž, jako kdyby byly zvlášť na samostatných řádcích

`příkaz1 | příkaz2 | příkaz3` řetězení vstupů/výstupů, tedy roury (kolony) (pozor, to není plně sekvenční provádění, následující příkaz může začít zpracovávat vstup ještě před ukončením předchozího příkazu – výstupy/vstupy se předávají postupně po blocích, typicky stránkovací filtry)

V UNIXových systémech máme možnost využít speciální symboly pro podmíněné vyhodnocování. Pokud vezmeme symboly `&` a `|` a zdvojíme je (bez mezery mezi nimi), budou se chovat následovně:

- dvojsymbol `&&` znamená konjunkci:
  - pokud předchozí příkaz skončil s úspěchem (true), bude vyhodnocen i následující příkaz,
  - pokud předchozí příkaz skončil s neúspěchem (false), nebude následující příkaz vyhodnocován (protože false v konjunkci s čímkoliv znamená vždy false v celém výrazu),
- dvojsymbol `||` znamená disjunkci:
  - pokud předchozí příkaz skončil s úspěchem (true), další příkaz již nebude vyhodnocován (protože jedno true v disjunkci znamená true v celém výrazu),
  - pokud předchozí příkaz skončil s neúspěchem (false), bude vyhodnocen i další příkaz.

Shrňme si syntaxi:

`příkaz1 && příkaz2 && příkaz3` logické AND, následující příkaz se provede pouze tehdy, když předchozí skončil úspěchem (jeho návratový kód byl 0, to zde znamená „true“).

`příkaz1 || příkaz2 || příkaz3` logické OR, následující příkaz se provede pouze tehdy, když předchozí skončil neúspěchem (tj. vrátil nenulový návratový kód).

Příklad

Chceme ukončit skript při výskytu chyby a zároveň do proměnné `$?` uložit číslo, ze kterého by bylo možné poznat, jaká chyba nastala.

```
...
cd xyz 2>/dev/null || exit 1
cp *.txt /media/usb/zaloha || exit 2
...
```

Pokud došlo k chybě u příkazu `cd` (přesun do neexistujícího adresáře), chybové hlášení bude zahozeno, ale skript se ukončí s chybovým kódem 1, následující příkaz už nebude proveden. Pokud ale `cd` skončí úspěchem (přesuneme se do `xyz`), spustí se příkaz `cp` (kopírování). U něj opět platí, že když dojde k chybě, skript bude ukončen (ale chybové hlášení se objeví).

Uživatel našeho skriptu může zjistit důvod problému tak, že po ukončení našeho skriptu otestuje obsah proměnné `$?`. Když v ní bude číslo 0, je vše v pořádku; číslo 1 znamená, že chyba nastala při přesunu do neexistujícího adresáře; číslo 2 indikuje problém s kopírováním.

Oba operátory můžeme také kombinovat. Typické použití je vypisování hlášení při správném nebo nesprávném provedení prvního příkazu v pořadí.

Příklad

Podívejme se na tento příkaz:

```
pgrep nejaky_proces >/dev/null && echo proces běží || echo proces neběží
```

Příkaz `pgrep` v sobě kombinuje funkce příkazů `ps` (výpis běžících procesů) a `grep` (filtrování). Vypíše efektivního vlastníka zadaného procesu (zjistíme, jaká oprávnění daný proces má).

Výsledek je směrován do odpadkového koše, tj. vlastně nechceme nic vypsat. Pouze v případě, že zadaný proces existuje, vypíše se hláška „proces běží“ (protože při vyhodnocení `true AND něco` potřebujeme znát výsledek toho `něco`, aby byl výpočet kompletní), a v případě, že zadaný proces neexistuje (`pgrep` nám „vynadá“ hodnotou `false`, že po něm chceme nemožné), se vypíše druhá hláška (při vyhodnocení `false OR něco` potřebujeme znát výsledek toho `něco`, aby byl výpočet kompletní).

Symbol `&` *samotný* na konci názvu příkazu je něco trochu jiného. Znamená asynchronní provádění příkazu, tedy příkaz následovaný tímto symbolem je spuštěn na pozadí a můžeme v shellu dále běžně pracovat. S touto možností se blíže seznámíme ve správě procesů a úloh.

Pro sdružování několika příkazů spojených těmito symboly můžeme použít závorky (ale jen v případě, že za nimi ještě má něco následovat, třeba symbol `&`). Pozor, za levou a před pravou závorkou musí být mezera. Takto se dá řešit problém poslání na pozadí celé sekvence příkazů.

Příklad

```
sleep 30; echo "Konec prace!"
```

30 sekund se nic neděje (ani prompt), pak se vypíše hláška

```
sleep 30; echo "Konec prace!" &
```

30 sekund se nic neděje, pak se na pozadí provede příkaz

```
( sleep 30; echo "Konec prace!" )&
```

30 sekund můžeme pracovat (je zobrazen prompt), pak se objeví hláška

Úkol

Sestavte příkaz, který provede toto:

- z pracovního adresáře rekurzívně dále vyhledá všechny soubory, jejichž název obsahuje řetězec „satelit“ (použijte příkaz `find`), výstup nasměrujte tak, aby se nezobrazil ani neuložil,
- pokud hledání skončí úspěchem, nechte vypsat hlášku „soubor existuje“,
- pokud hledání skončí neúspěchem, nechte vypsat hlášku „soubor neexistuje“.

Předpokládejme, že takový soubor nikde není (jestli ano, pak zvolte jiný řetězec).

Otestujte. Nejdřív by příkaz měl vypisovat hlášku o neexistenci souboru. Pak vytvořte soubor s takovým názvem (třeba příkazem `touch`) a otestujte znovu. Teď by měl příkaz vypisovat pozitivní hlášku.

6.7.2 Příkazy větvení

Předně je třeba si uvědomit jednu důležitou věc – složené příkazy jsou doopravdy složené z více příkazů, a tedy buď jejich části (ty, které se vyhodnocují sekvenčně) oddělujeme středníkem, anebo pokračujeme na dalším řádku (víceřádkový příkaz, klepneme na Enter).

Příkaz `if` je základním příkazem pro větvení kódu v shellu bash. Blok začíná klíčovým slovem `if` a končí sklíčovým slovem `fi` (tj. zrcadlovým obrazem prvního klíčového slova). Jako podmínku můžeme použít cokoliv, co má nějakou návratovou hodnotu (třeba příkaz, který při svém úspěšném provedení vrátí 0 jako true nebo nenulovou hodnotu jako false).

Podmínku vždy ukončujeme buď symbolem středník (jak v ukázce níže vidíme v prvním sloupci), nebo musíme přejít na další řádek (druhý sloupec), taky je možné využít i větev `else` (třetí sloupec):

```
if podmínka ; then
  příkazy
fi
```

```
if podmínka
then
  příkazy
fi
```

```
if podmínka ; then
  podmínka splněna
else
  podmínka nesplněna
fi
```

V případě, že při nesplnění podmínky chceme testovat ještě další podmínku, máme dvě možnosti: buď vnoříme jiný blok s druhou podmínkou, nebo použijeme klíčové slovo `elif`:

```
if podmínka1 ; then
  příkazy pro splněnou podmínku1
else
  if podmínka2 ; then
    příkazy nesplněnou podmínku1,
    ale splněnou podmínku2
  else
    žádná podmínka není splněna
  fi
fi
```

```
if podmínka1 ; then
  příkazy pro splněnou podmínku1
elif podmínka2 ; then
  příkazy nesplněnou podmínku1,
  ale splněnou podmínku2
else
  žádná podmínka není splněna
fi
```

Negaci vyjadřujeme vykřičníkem (ostatně, to je v bash i v množinách zadaných hranatými závorkami). Za vykřičníkem vždy píšeme mezeru (on je to ve skutečnosti vlastně taky příkaz).

Příklad

Píšeme skript, jehož první parametr je název souboru, který má být vytvořen v daném adresáři. Pokud ten adresář neexistuje, bude předem vytvořen. První způsob je tento:

```
if cd ~/dokumenty/faktury ; then
  touch ~/dokumenty/faktury/$1
  echo Soubor $1 vytvoren
else
  mkdir ~/dokumenty/faktury
  cd ~/dokumenty/faktury
  touch ~/dokumenty/faktury/$1
  echo Adresar s fakturami neexistoval => vytvoren, soubor $1 vytvoren
fi
```

Jde to i jednodušeji, třeba s využitím negace (vykřičníku):

```
if ! cd ~/dokumenty/faktury ; then
  mkdir ~/dokumenty/faktury
  echo Adresar s fakturami neexistoval => vytvoren
fi
touch ~/dokumenty/faktury/$1
echo Soubor $1 vytvoren
```

Nebo jinak (s využitím podmíněného vyhodnocování, které jsme probírali v předchozím textu), všimněte si způsobu přechodu na další řádek uvnitř složeného příkazu:

```
cd ~/dokumenty/faktury 2>/dev/null || mkdir ~/dokumenty/faktury ; \
   cd ~/dokumenty/faktury ; echo Adresar s fakturami neexistoval => vytvoren
touch ~/dokumenty/faktury/$1
echo Soubor $1 vytvoren
```

Příklad

Pokud přidáváme nové uživatele do systému, můžeme si předem ověřit, zda už náhodou takový uživatel v systému není. K vytváření nového uživatele se dostaneme později, teď si ukážeme jen to otestování. Použijeme příkaz `grep` a necháme ho najít daného uživatele v souboru `/etc/passwd`, kde, jak víme, je seznam registrovných uživatelů.

Předpokládejme, že název uživatele máme v lokální proměnné `uzivatel`.

```
if ! grep ${uzivatel} /etc/passwd >/dev/null 2>&1 ; then
  ...
  teď toho uživatele vytvořím, protože podmínka je splněna, pokud zadaný uživatel neexistuje
  (vykřičník neguje podmínku)
fi
```

Abychom nemuseli složitě řešit směrování výstupu i chybového výstupu do `/dev/null`, můžeme u příkazu `grep` použít přepínač `-q` pro tichý režim (quiet), jen vrátí hodnotu `true` (=0) nebo `false` (>0):

```
if ! grep -q ${uzivatel} /etc/passwd ; then
  ...
  teď toho uživatele vytvořím, protože podmínka je splněna, pokud zadaný uživatel neexistuje
  (vykřičník neguje podmínku)
fi
```

Poznámka

Pokud v příkazu `if` chceme nechat první větev prázdnou a psát příkazy až do větve `else`, do první větve dáme alespoň prázdný příkaz (to je „:“, dvojtečka).

Testování parametrů jako je srovnávání hodnot čísel/řetězců a proměnných, existence souboru či adresáře apod., provádíme pomocí příkazu `test`. Tento příkaz má za parametr testovací logický výraz, vrací buď hodnotu `true` (jako každý proces – návratová hodnota =0) nebo `false` (>0).

Buď použijeme přímo klíčové slovo `test`, nebo použijeme jeho „závorkovou“ podobu. Níže vidíme vlevo syntaxi tohoto příkazu v obou podobách, vpravo pak jeho začlenění jako podmínku příkazu `if`.

```
test výraz                          if test výraz; then
[ výraz ]                           if [ výraz ] ; then
```

Teď se podíváme na podobu testovacího výrazu. Používají se řetězce obvykle představující proměnné a dále operátory reprezentované těmito přepínači:

Pro řetězce:

`řetězec`	= true, pokud je řetězec neprázdný
`-z řetězec`	= true, pokud je řetězec prázdný
`řetězec1 = řetězec2`	= true, pokud jsou řetězce stejné
`řetězec1 != řetězec2`	= true, pokud jsou řetězce různé

Pro názvy souborů:

`-f soubor`	= true, pokud soubor existuje a je běžný soubor
`-d soubor`	= true, pokud soubor existuje a je to adresář
`-r soubor`	= true, pokud soubor existuje a je nastaveno právo čtení
`-w soubor`	= true, pokud soubor existuje a je nastaveno právo zápisu
`-x soubor`	= true, pokud soubor existuje a je nastaveno právo spouštění
`-s soubor`	= true, pokud soubor existuje a má nenulovou délku

Pro čísla a proměnné s čísly:

`číslo1 -eq číslo2`	= true, pokud číslo1 = číslo2
`číslo1 -ne číslo2`	= true, pokud číslo1 <> číslo2
`číslo1 -gt číslo2`	= true, pokud číslo1 > číslo2
`číslo1 -lt číslo2`	= true, pokud číslo1 < číslo2
`číslo1 -ge číslo2`	= true, pokud číslo1 ≥ číslo2
`číslo1 -le číslo2`	= true, pokud číslo1 ≤ číslo2

Skládání výrazů:

`! výraz`	negace výrazu
`výraz1 -a výraz2`	logické AND
`výraz1 -o výraz2`	logické OR
`\( výraz \).`	uzávorkování výrazu

Poznámka

Pokud existuje třeba jen malá pravděpodobnost, že v proměnné použité ve výrazu by mohl být prázdný řetězec, musí být název proměnné (i se znakem $) uzavřen do uvozovek. Totéž platí v případě, že v proměnné je uložen řetězec s mezerami nebo obecně jinými než základními znaky.

Příklad

Ukážeme si použití jednoduchého testování na zjištění existence adresáře. Před přesunem do zadaného adresáře si můžeme takto otestovat, zda existuje (vlevo je zápis s klíčovým slovem, vpravo závorková forma):

```
if test -d adresar
then
  cd adresar
else
  ztratil(-a) jsem se!!!
fi
```

```
if [ -d adresar ]
then
  cd adresar
else
  ztratil(-a) jsem se!!!
fi
```

Parametr `-d` slouží k testování existence adresáře, jiné parametry zase slouží k testování existence jakéhokoliv souboru, blokového či znakového speciálního souboru, souboru s nastaveným SUID bitem, je možné testovat také přístupová oprávnění, zda je adresář prázdný, atd.

Příklad

Názvy adresářů jsou vlastně řetězce, tedy se s nimi dá zacházet jako s jinými řetězci. Následující příkaz zjišťuje, zda jsme v kořenovém adresáři (uvedeme si už jen závorkovou formu):

```
#!/bin/bash
if [ "$PWD" = "/" ] ; then
  echo Pracovní adresář je root, tam mě nikdo nedostane!
  exit 1
fi
```

Příklad

Víme, že pro spuštění souboru se skriptem potřebujeme právo „`x`“. Můžeme otestovat, zda ho máme:

```
if [ ! -x ./testik.sh ] ; then
  echo Nemám dostačující přístupová oprávnění. Končím.
  exit 1
fi
./testik.sh
```

Vykřičník nám neguje výraz v podmínce. Kolem hranatých závorek (obou) a za vykřičníkem bychom měli udělat mezeru.

Úkoly

1. Vyzkoušejte si ověřování existence uživatele pomocí příkazu `grep` podobně, jak je to v příkladu na straně 187 (místo proměnné použijte své vlastní přihlašovací jméno nebo třeba `root`, případně i řetězec, který není žádným přihlašovacím jménem). Ve větvích `then` a `else` jen vypište informaci o existenci nebo neexistenci uživatele. Vyzkoušejte obě větve.

   Nemějte obavy v shellu na těch správných místech klepnout na Enter – na dalších řádcích složeného příkazu se jednoduše dostanete do promptu nižší úrovně a budete pokračovat ve psaní. Pokud budete chtít v shellu psát celý příkaz na jeden řádek, musíte na místech, kde byste jinak dali konec řádku, umístit středník (třeba za `then` nebo před `fi`).

2. Vyzkoušejte i to, co je v následujících příkladech (práce s adresáři, varianty příkazu `test` nebo hranaté závorky). Nejdřív v pracovním adresáři vytvořte skript, ve kterém zjistíte, jestli pracovním adresářem je či není kořenový adresář (root), a pak buď v jiném skriptu nebo přímo v shellu otestujte, zda k původnímu skriptu máte oprávnění spuštění.

   To otestování se dá provést i bez příkazu `if`:

   ```
   [ -x skript.sh ] || echo nemam opravneni
   ```

 **Větvení pomocí příkazu `case`** je praktičtější, pokud chceme otestovat více alternativ pro danou hodnotu či výraz. Princip je podobný jako v jiných programovacích jazycích – stanovíme výraz, jehož výsledek chceme otestovat (typicky proměnnou, podle které větvíme) a určíme jednotlivé větve.

V syntaxi je za hodnotou větve pravá závorka, každá větev se ukončuje dvojicí středníků, celý složený příkaz ukončujeme zrcadlovým obrazem prvního klíčového slova – `esac`. Můžeme (nemusíme) použít větev typu default (místo konkrétní hodnoty dáme hvězdičku).

```
case $prom in
hodnota1)
  příkazy pro případ, že $prom=hodnota1
  ;;
hodnota2)
  příkazy pro případ, že $prom=hodnota2
  ;;
hodnota3 | hodnota4)
  příkazy pro případ, že $prom=hodnota3 nebo $prom=hodnota4
  ;;
*)
  toto je větev default, resp. else; provede se, když žádná z předchozích větví není použita
esac
```

6.7.3 Cyklus přes množinu

Je dána množina prvků (řetězců, čísel apod., cokoliv, co můžeme dosadit do proměnné). Touto množinou může být například množina všech názvů souborů, které odpovídají zadané masce (pomocí hvězdičky apod.), nebo prvky množiny můžeme rovnou vyjmenovat. Přes všechny prvky této množiny potřebujeme provést určitý příkaz nebo sekvenci příkazů.

 Příkaz `for` slouží právě k tomuto účelu. Stanovíme množinu prvků a určíme proměnnou, do které jsou postupně prvky dané množiny dosazovány, a pak samozřejmě ty příkazy. Základní syntaxe je tato (jsou tu ukázány dva způsoby reprezentace množiny řetězců):

```
for prom in řetězec1 řetězec2 řetězec3 ; do
  příkazy používající proměnnou $prom
done
```

```
mnozina="řetězec1 řetězec2 řetězec3"
for prom in $mnozina ; do
  příkazy používající proměnnou $prom
done
```

Pokud bychom chtěli klíčové slovo `do` napsat až na další řádek, středník si můžeme odpustit. Všimněte si rozdílu oproti předchozím složeným příkazům: klíčové slovo pro ukončení složeného příkazu tentokrát není zrcadlovým obrazem prvního klíčového slova.

 Klíčové slovo `in` je ve skutečnosti příkazem, jehož parametry jsou prvky zadané množiny řetězců, středník za množinou tento příkaz odděluje od příkazu `do`, jehož úkolem je přijmout výsledek příkazu `in` a předat ho vnořeným příkazům.

Příklad

Protože víme, že zástupné symboly, jejichž interpretaci jsme nezakázali (zpětným lomítkem, uvozovkami, apostrofy), interpretuje shell ještě před předáním řetězce zadanému příkazu, je jasné, že v řetězcích klidně můžeme tyto zástupné symboly používat.

```
for i in *.txt ; do
  echo Kopíruji $i
  cp ${i} /media/sitovy/zalohatxt
done
```

Příklad

Množina, přes kterou jde proměnná v cyklu `for`, může být vygenerována příkazem. Nesmíme však zapomenout na to, že dotyčný příkaz generující množinu musíme uzavřít do zpětných apostrofů, aby byl vyhodnocen ještě před vyhodnocením samotného cyklu `for`.

```
for i in ‘find . -name ’*.txt’‘
do
  cp $i ${i%txt}bak
done
```

Do proměnné `$i` postupně přiřazujeme všechny názvy souborů s příponou TXT, a to rekurzívně, kopírujeme je na soubory do stejného adresáře, ale s jinou příponou (BAK). Všimněte si, jakým způsobem bylo provedeno „odstřihnutí“ původní přípony a zřetězení s novou příponou.

Kdybychom chtěli celý příkaz umístit na jeden řádek:

```
for i in ‘find . -name ’*.txt’‘ ; do cp $i ${i%txt}bak ; done
```

Příklad

Následující skript (zapsaný v souboru) v cyklu prochází všechny soubory z pracovního adresáře včetně skrytých, u každého se zeptá, jestli ho má smazat. Pokud ano, použije příkaz pro rekurzívní mazání. Všimněte si, jakým způsobem je ošetřena chyba při mazání souboru.

```
#!/bin/bash
#přepneme se do adresáře docasny v domovském adresáři
cd ~/docasny

#vytvoříme proměnnou
souhlas="N"

pwd
for i in ‘ls -a‘ ; do
  echo Smazat $i .... (A/N)?
  read souhlas
  if [ "$souhlas" = "A" || "$souhlas" = "a" ] ; then
    #rekurzívní mazání souborů a adresářů:
    rm -fr $i
    if [ $? -eq 0 ] ; then
      echo OK, soubor $i smazán
    else
      echo CHYBA při mazání souboru $i!
    fi
```

```
  else
    echo OK, soubor $i zůstane
  fi
done
```

Řetězec ‘`ls -a`‘ bude interpretován shellem, výsledek zahrnuje všechny skryté soubory a adresáře z pracovního adresáře. Tento řetězec můžeme zapsat i jinak:

```
$( ls -a )
```

Příklad

V parametru určujícím množinu mohou být jakékoliv řetězce. Také tam mohou být proměnné, jejichž interpretaci si chceme vynutit. Dokonce můžeme naopak vynucení pozdržet.

```
#!/bin/bash
for c in 1 2 3 ; do
  echo c = $c
done

for i in $( "$HOME" "$SHELL" "$PS1" ) ; do
  echo $i je ‘$i‘
done
```

Příklad

Ve skriptu chceme do proměnné uložit abecedně seřazený seznam uživatelů oddělených mezerou (abychom si procvičili řetězení obsahu proměnných).

```
SOUBOR=/etc/passwd
uzivatele=""
for i in $( cut -d ":" -f 1 $SOUBOR | sort ) ; do
  uzivatele="$uzivatele $i"
done
echo $uzivatele
```

Místo zápisu podmínky pomocí dolarovky se závorkami jsme alternativně mohli použít zápis pomocí zpětných apostrofů (je to jedno, obojí způsobí interpretaci obsahu). „Rozřezali“ jsme řádek souboru podle oddělovačů „dvojtečka“ a vybrali jsme první sloupec.

Úkoly

1. Zapište do shellu následující složený příkaz (tak jak je, na víc řádků):

   ```
   for i in ‘ls -Ra‘ ; do
     if [ -x $i -a ! -d $i ] ; then
       echo spustim $i
     fi
   done
   ```

   Výstupem příkazu `ls -Ra` je seznam všech názvů souborů z pracovního adresáře, a to rekurzívně. Každý z těchto názvů je profiltrován příkazem `if` – projdou jen ty, ke kterým máme nastaveno oprávnění „`x`“ a zároveň to nejsou adresáře (vykřičník). Co tedy bude výstupem příkazu?

2. Pokud jste správně zapsali složený příkaz z předchozího úkolu (a na více řádků), zkuste teď udělat krok do historie – stiskněte jednou kurzorovou klávesu pro pohyb zpět v historii. Jak se vám zobrazil předchozí zadaný příkaz?

   Zde si všimněte, kam konkrétně patří středníky, pokud budeme chtít složený příkaz zapsat na jediném řádku.

6.7.4 Cykly pro rozsah hodnot

V shellu bash dále máme cykly `while` a `until`. Jejich syntaxe je následující:

```
while [ podmínka ] ; do
  příkazy
done
```

```
until [ podmínka ] ; do
  příkazy
done
```

Cyklus `while` provádí příkazy ve svém těle tak dlouho, *dokud platí podmínka*, cyklus `until` naopak provádí příkazy ve svém těle, *dokud nezačne platit podmínka*. Opět je třeba dávat pozor na umístění mezer, formát podmínky je stejný jako u příkazu `if`, resp. `test`.

Příklad

Následující dvě varianty kódu jednoduše čekají na stisknutí jakékoliv klávesy kromě klávesy „N“.

```
#!/bin/bash
klavesa="N"
until [ $klavesa != "N" ] ; do
  echo Stiskni něco jiného než N... a Enter
  read klavesa
done
```

```
#!/bin/bash
klavesa="N"
while [ $klavesa = "N" ] ; do
  echo Stiskni něco jiného než N... a Enter
  read klavesa
done
```

Příklad

Proměnná `CITAC` postupně roste o 1, přičemž určuje, kolikrát bude proměnná `vysledek` násobena číslem 2. Výsledek celého výpočtu bude mocnina čísla 2. Skript můžeme vytvořit v textovém editoru, například xed nebo nano.

```
#!/bin/bash
CITAC=1
POSLEDNI=5
vysledek=1
while [ $CITAC -le $POSLEDNI ] ; do
  (( vysledek*=2 ))
  (( CITAC+=1 ))
done
echo 2^${POSLEDNI} = $vysledek
```

Úkol

Projděte si příklady v této sekci. U posledního příkladu pozměňte kód tak, aby obsah proměnné POSLEDNI bral z parametru skriptu.

6.8 Překlad programů

V shellu můžeme spouštět i aplikace grafického režimu, existují zde i textové programy pro úpravu textů (např. nano, pico, vi, vim, emacs, kwrite, kedit), kompilátory jazyka C (gcc, c++, cc, ...) apod.

Pro překlad programů v jazyce C (nebo také C++) se obvykle používá překladač `gcc` (GNU C Compiler). Jedná se opravdu o překladač, nikoliv editor. Jako editor můžeme použít některý z běžných textových editorů, ale existují i speciálně určené pro programování, například Kate nebo prostředí KDevelop, prostředí Eclipse a další. Program `gcc` se v jednodušším případě používá takto:

`gcc -o vystupni_soubor zdrojovy.c` přeloží zadaný zdrojový soubor, také je stanoven výstupní soubor

Program má velmi mnoho nejrůznějších voleb, často určených pro překlad konkrétního jazyka, některé volby jsou hardwarově závislé (pro konkrétní hardwarovou platformu). Existují také například volby pro překlad vícevláknových aplikací.

Úkol

Vytvořte textový soubor `world.c` s následujícím obsahem (například v editoru kwrite nebo gedit, nebo přímo v shellu příkazem `cat > soubor.c << KONEC`, ovšem nepočítejte s tabulátory):

```
#include<stdio.h>
int main(void){
 printf("Hello World\n");
}
```

Uložte do vašeho domovského adresáře a pak přeložte:

```
gcc -o world world.c
```

Program spusťte. Nezapomeňte, že váš domovský adresář zřejmě není v proměnné obsahující cesty ke spustitelným souborům, tedy je nutné zadat cestu do pracovního adresáře:

```
./world
```

6.9 Konverze textových souborů

`iconv` je program pro konverzi textových souborů mezi různými kódováními (filtr), například

`iconv -f cp1250 -t iso8859-2 < souborwin.txt > souboriso.txt`

zkonvertuje soubor s Windows kódováním cp1250 na ISO8859-2,

`iconv -f iso8859-2 -t utf-8 < souboriso.txt > souborutf.txt`

zkonvertuje soubor v kódování ISO8859-2 na unicode UTF-8.

Seznam všech podporovaných kódování se zobrazí `iconv --list` nebo `iconv -l`.

Programem s rozsáhlejšími funkcemi je například `recode` (dokáže kromě jiného také odstranit diakritiku ze souboru), dále `enca` (dokáže také detekovat znakovou sadu). Program `cstocs` je zase konvertor specializovaný na češtinu a slovenštinu:

```
cstocs cp1250 utf-8 < soubor.txt > vysledek.txt
```

Program `convmv` dokáže opravit názvy souborů, které nezodpovědný uživatel Windows pojmenoval s použitím háčků a čárek a odeslal jinému uživateli:

```
convmv -f cp1250 -t utf8 soubor
```

Souhrn kapitoly 6

V této dlouhé kapitole jsme si zopakovali a prohloubili své znalosti shellu bash. Po krátkém připomenutí, že existují i jiné shelly, jsme se už soustředili pouze na shell bash. Naučili jsme se vytvořit nový soubor přímo v textovém režimu, také formou here document, kde není nutno pamatovat si ukončující klávesovou zkratku, pak jsme si shrnuli různé možnosti vyhledávání: a to prohledávání adresářové struktury a prohledávání textů. Seznámili jsme se s významem různých druhů uvozujících symbolů (apostrofů, zpětných apostrofů a uvozovek) a připomněli si význam zpětného lomítka, pak jsme si ukázali, jak se v bash počítá.

Následovaly sekce o psaní skriptů, používání složených příkazů (pro větvení kódu a různých druhů cyklů), také jsme si ukázali, jak se překládá program pomocí gcc a jak můžeme konvertovat textové soubory do různých kódování.

Kapitola 7

Řízení přístupu a správa uživatelů v Linuxu

Rychlý náhled:

Klíčová slova:

Cíle studia:

Kapitola 8

Správa procesů v Linuxu

Rychlý náhled:

Klíčová slova:

Cíle studia:

Kapitola 9

Správa zařízení v Linuxu

Rychlý náhled:

Klíčová slova:

Cíle studia:

Kapitola 10

Správa sítě v Linuxu

Rychlý náhled:

Klíčová slova:

Cíle studia:
spojit?????

Kapitola 11

Linux: nasazení systému

Rychlý náhled:

Klíčová slova:

Cíle studia:

Přílohy

Příloha A

Ladění programů a jádra Windows

Zde se budeme věnovat nástrojům usnadňujícím sledování a nalezení chyb při běhu procesů, služeb, ovladačů a samotného jádra.