Operační systémy

Základní součásti operačního systému

Klíčové pojmy

- **Proces**: Program v běhu, který zahrnuje aktuální stav programu a jeho data.
  Vlákno (Thread): Menší jednotka procesu, která může běžet paralelně s jinými vlákny.
- **Správa paměti**: Způsob, jakým operační systém spravuje paměť, včetně alokace a dealokace paměti.
- **Souborový systém**: Struktura, která umožňuje ukládání a organizaci souborů na disku.
- **Vstupně-výstupní (I/O) systémy**: Mechanismy pro komunikaci mezi počítačem a periferními zařízeními.
- **Plánování procesů**: Algoritmy a metody, které určují pořadí, ve kterém budou procesy vykonávány.
- **Synchronizace**: Techniky pro řízení přístupu k sdíleným prostředkům, aby se zabránilo konfliktům.
- **Deadlock (zablokování)**: Situace, kdy dva nebo více procesů čekají na zdroje, které jsou vzájemně blokovány.

Významná jména v historii vývoje operačních systémů

- **Ken Thompson** - "You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me.)"
- **Dennis Ritchie** - "UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity."
- **Bill Gates** - "Your most unhappy customers are your greatest source of learning."
- **Paul Allen**
- **Andrew S. Tanenbaum** - "Never underestimate the bandwidth of a station wagon full of tapes hurtling down the highway."
- **Linus Torvalds** – „Talk is cheap, show me the code.“
- **Steve Jobs**

Operační systémy

- **UNIX:** Jeden z prvních víceuživatelských a multitaskingových operačních systémů, vyvinutý v 60. letech. Stal se základem pro mnoho moderních operačních systémů
- **MS-DOS:** Microsoft Disk Operating System, který dominoval osobním počítačům v 80. letech
- **Windows:** Grafický operační systém od Microsoftu, který se stal standardem pro osobní počítače
- **Linux:** Open-source operační systém, který vznikl jako alternativa k UNIXu a stal se základem pro mnoho serverových a desktopových systémů
- **Mac OS:** Operační systém od Apple, známý pro své intuitivní grafické uživatelské rozhraní

Cíle studia operačních systémů

Procesy

V **MS Windows** je správa procesů zajišťována několika nástroji a mechanismy:

- **Správce úloh**: Tento nástroj umožňuje uživatelům sledovat a spravovat běžící procesy. Můžete jej spustit pomocí klávesové zkratky Ctrl+Shift+Esc nebo přes kontextovou nabídku hlavního panelu. Správce úloh zobrazuje využití procesoru, paměti, disku a sítě jednotlivými procesy
- **PowerShell**: Pomocí PowerShellu lze spravovat procesy pomocí příkazů jako `Get-Process`, `Stop-Process` a `Start-Process`. Tyto příkazy umožňují získávat informace o procesech, zastavovat je a spouštět nové procesy
- **Process Explorer**: Tento nástroj od Sysinternals poskytuje detailní informace o procesech a umožňuje sledovat problémy s knihovnami DLL nebo úniky paměti

V **Linuxu** je správa procesů prováděna pomocí různých příkazů a nástrojů:

- **ps**: Tento příkaz zobrazuje seznam běžících procesů. Můžete jej použít s různými přepínači, jako je `ps -e` pro zobrazení všech procesů nebo `ps -u` pro zobrazení procesů konkrétního uživatele
- **top**: Tento interaktivní nástroj zobrazuje aktuální stav systému, včetně běžících procesů, využití CPU a paměti. Umožňuje také ukončovat procesy
- **htop**: Vylepšená verze příkazu `top`, která poskytuje barevné rozhraní a další funkce pro správu procesů
- **nice a renice**: Tyto příkazy slouží k nastavení a změně priority procesů. Příkaz `nice` spouští proces s upravenou prioritou, zatímco `renice` mění prioritu již běžícího procesu
- **kill**: Tento příkaz slouží k zasílání signálů procesům, například k jejich ukončení. Příkaz `kill -9 PID` ukončí proces s daným PID

V **Unixu** jsou procesy spravovány podobně jako v Linuxu, protože Linux je odvozen od Unixu:

- **ps**: Stejně jako v Linuxu, příkaz `ps` zobrazuje seznam běžících procesů. Můžete použít přepínače jako `ps -l` pro podrobnější informace
- **top**: Tento nástroj poskytuje přehled o aktuálním stavu systému a umožňuje správu procesů
- **nice a renice**: Tyto příkazy slouží k nastavení a změně priority procesů. Příkaz `nice` spouští proces s upravenou prioritou, zatímco `renice` mění prioritu již běžícího procesu
- **kill**: Stejně jako v Linuxu, příkaz `kill` slouží k zasílání signálů procesům. Příkaz `kill -9 PID` okamžitě ukončí proces

a další.

Vlákno (Thread)

Vlákno je základní jednotka, pro kterou operační systém přiděluje čas procesoru. Každý proces může obsahovat jedno nebo více vláken, která sdílejí stejný paměťový prostor a systémové prostředky. Vlákna umožňují paralelní nebo kvaziparalelní provádění kódu, což zvyšuje efektivitu a rychlost aplikací.

Vlákna v MS Windows

V MS Windows jsou vlákna spravována pomocí několika nástrojů a API:

- **Win32 API**: Základní API pro správu vláken v Windows. Příkazy jako `CreateThread`, `TerminateThread`, `SuspendThread` a `ResumeThread` umožňují vytváření, ukončování a řízení vláken
- **Thread Pool**: Windows poskytuje fond vláken, který umožňuje efektivní správu a opětovné použití vláken pro různé úkoly. Třída `ThreadPool` v .NET Frameworku umožňuje snadné použití tohoto fondu
- **Spravovaná vlákna**: V .NET Frameworku jsou vlákna spravována pomocí třídy `Thread`. Tato vlákna mohou být vytvořena a řízena pomocí metod jako `Thread.Start`, `Thread.Abort` a `Thread.Join`

Vlákna v Linuxu

V Linuxu jsou vlákna implementována pomocí knihovny pthreads (POSIX threads):

- **pthreads:** Knihovna poskytuje funkce jako `pthread_create`, `pthread_join`, `pthread_cancel` a `pthread_mutex_lock` pro vytváření a správu vláken a synchronizaci mezi nimi
- **NPTL (Native POSIX Thread Library):** Moderní implementace vláken v Linuxu, která poskytuje lepší výkon a kompatibilitu s POSIX standardy
- **Thread Pool:** Linux také podporuje fondy vláken, které umožňují efektivní správu vláken pro paralelní úkoly

Proces x vlákno

Procesy:

- **Izolace**: Každý proces běží ve svém vlastním paměťovém prostoru, což znamená, že procesy jsou izolované od sebe navzájem. To zvyšuje bezpečnost a stabilitu, protože chyby v jednom procesu neovlivní ostatní procesy.
- **Přepínání kontextu**: Přepínání mezi procesy (context switching) je náročnější na zdroje, protože zahrnuje změnu celého paměťového prostoru a registrů procesoru.
- **Komunikace**: Procesy komunikují mezi sebou pomocí mechanismů jako jsou roury (pipes), fronty zpráv (message queues) nebo sdílená paměť (shared memory), což může být složitější a pomalejší než komunikace mezi vlákny.
- **Vytváření**: Vytvoření nového procesu je náročnější na zdroje, protože zahrnuje alokaci nového paměťového prostoru a inicializaci všech potřebných struktur.

Vlákna:

- **Sdílení paměti:** Vlákna v rámci jednoho procesu sdílejí stejný paměťový prostor, což umožňuje rychlejší a jednodušší komunikaci mezi vlákny. To však také znamená, že chyby v jednom vlákně mohou ovlivnit ostatní vlákna v rámci stejného procesu.
- **Přepínání kontextu:** Přepínání mezi vlákny je méně náročné na zdroje než přepínání mezi procesy, protože vlákna sdílejí stejný paměťový prostor a mnoho systémových prostředků.
- **Komunikace:** Vlákna mohou snadno komunikovat a synchronizovat se pomocí mechanismů jako jsou mutexy, semafory a podmínkové proměnné.
- **Vytváření:** Vytvoření nového vlákna je méně náročné na zdroje než vytvoření nového procesu, protože vlákna sdílejí mnoho systémových prostředků.

Příklad použití vláken Python:

```
import threading
import requests

# Funkce pro stahování souboru
def download_file(url, filename):
    response = requests.get(url)
    with open(filename, 'wb') as file:
        file.write(response.content)
    print(f"{filename} stažen.")

# Seznam souborů ke stažení
files_to_download = [
    ("https://example.com/file1.zip", "file1.zip"),
    ("https://example.com/file2.zip", "file2.zip"),
    ("https://example.com/file3.zip", "file3.zip")
]

# Vytvoření a spuštění vláken
threads = []
for url, filename in files_to_download:
    thread = threading.Thread(target=download_file, args=(url, filename))
    threads.append(thread)
    thread.start()

# Čekání na dokončení všech vláken
for thread in threads:
    thread.join()

print("Všechny soubory byly staženy.")
```

Vysvětlení:

1. **Funkce `download_file`**: Stahuje soubor z dané URL a ukládá jej do souboru s daným názvem.
2. **Seznam souborů ke stažení**: Obsahuje URL a názvy souborů, které chceme stáhnout.
3. **Vytvoření a spuštění vláken**: Pro každý soubor vytvoříme nové vlákno, které spustí funkci `download_file`.
4. **Čekání na dokončení všech vláken**: Pomocí `thread.join()` zajistíme, že hlavní program počká, dokud všechna vlákna nedokončí svou práci.

Příklad použití vláken v C++:

```
#include <iostream>
#include <thread>
#include <vector>
#include <string>
#include <curl/curl.h>

// Funkce pro zápis dat do souboru
size_t WriteData(void* ptr, size_t size, size_t nmemb, FILE* stream) {
    size_t written = fwrite(ptr, size, nmemb, stream);
    return written;
```

```
}

// Funkce pro stahování souboru
void download_file(const std::string& url, const std::string& filename) {
    CURL* curl;
    FILE* fp;
    CURLcode res;

    curl = curl_easy_init();
    if (curl) {
        fp = fopen(filename.c_str(), "wb");
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteData);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, fp);
        res = curl_easy_perform(curl);
        curl_easy_cleanup(curl);
        fclose(fp);
        if (res == CURLE_OK) {
            std::cout << filename << " stažen." << std::endl;
        } else {
            std::cerr << "Chyba při stahování " << filename << ": " <<
curl_easy_strerror(res) << std::endl;
        }
    }
}

int main() {
    // Seznam souborů ke stažení
    std::vector<std::pair<std::string, std::string>> files_to_download = {
        {"https://example.com/file1.zip", "file1.zip"},
        {"https://example.com/file2.zip", "file2.zip"},
        {"https://example.com/file3.zip", "file3.zip"}
    };

    // Vytvoření a spuštění vláken
    std::vector<std::thread> threads;
    for (const auto& file : files_to_download) {
        threads.emplace_back(download_file, file.first, file.second);
    }

    // Čekání na dokončení všech vláken
    for (auto& thread : threads) {
        thread.join();
    }

    std::cout << "Všechny soubory byly staženy." << std::endl;
    return 0;
}
```

Vysvětlení

1. Funkce `WriteData`: Tato funkce zapisuje data do souboru a je používána knihovnou libcurl pro ukládání stažených dat.
2. Funkce `download_file`: Tato funkce používá libcurl pro stahování souboru z dané URL a ukládá jej do souboru s daným názvem.
3. Seznam souborů ke stažení: Obsahuje URL a názvy souborů, které chceme stáhnout.

4. Vytvoření a spuštění vláken: Pro každý soubor vytvoříme nové vlákno, které spustí funkci `download_file`.
5. Čekání na dokončení všech vláken: Pomocí `thread.join()` zajistíme, že hlavní program počká, dokud všechna vlákna nedokončí svou práci.

Správa paměti

MS Windows

Správa paměti v MS Windows zahrnuje několik klíčových mechanismů:

- **Virtuální paměť:** Windows používá virtuální paměť, která umožňuje programům používat více paměti, než je fyzicky dostupné. Toho je dosaženo pomocí stránkovacího souboru (page file) na disku.
- **Správce paměti:** Windows má správce paměti, který přiděluje a uvolňuje paměť pro procesy a aplikace. Tento správce také monitoruje využití paměti a optimalizuje její rozdělení
- **Správce úloh:** Nástroj, který umožňuje uživatelům sledovat a spravovat využití paměti jednotlivými procesy. Uživatelé mohou ukončovat procesy, které spotřebovávají příliš mnoho paměti.
- **Optimalizace paměti:** Windows poskytuje nástroje jako RAMMap a CleanMem, které pomáhají optimalizovat využití paměti a uvolňovat zbytečně obsazenou paměť.

Příklady příkazů:

- `tasklist`: Zobrazí seznam běžících procesů a jejich využití paměti.
  `tasklist`
- `systeminfo`: Poskytuje podrobné informace o systému, včetně využití paměti.
  `systeminfo`
- `wmic memorychip get`: Zobrazí informace o paměťových modulech.
  `wmic memorychip get capacity, speed, manufacturer`
- `resmon`: Spustí Monitor prostředků, který poskytuje podrobné informace o využití paměti.
  `resmon`

Linux

- **Virtuální paměť:** Linux používá virtuální paměť, která umožňuje procesům používat více paměti, než je fyzicky dostupné. To je dosaženo pomocí stránkování a swapování
- **Alokace paměti:** Linux poskytuje dvě hlavní metody pro alokaci paměti: statickou a dynamickou. Dynamická alokace umožňuje procesům získat paměť podle potřeby
- **Správa paměti jádra:** Linuxové jádro má vlastní mechanismy pro správu paměti, včetně alokátorů paměti a algoritmů pro stránkování
- **Ochrana paměti:** Linux zajišťuje, že každý proces má přístup pouze k paměti, která mu byla přidělena, což zvyšuje bezpečnost a stabilitu systému

Příklady příkazů:

- `free: Zobrazí informace o využití paměti.`
  `free -h`
- `vmstat: Poskytuje informace o využití paměti, CPU a dalších systémových prostředcích.`

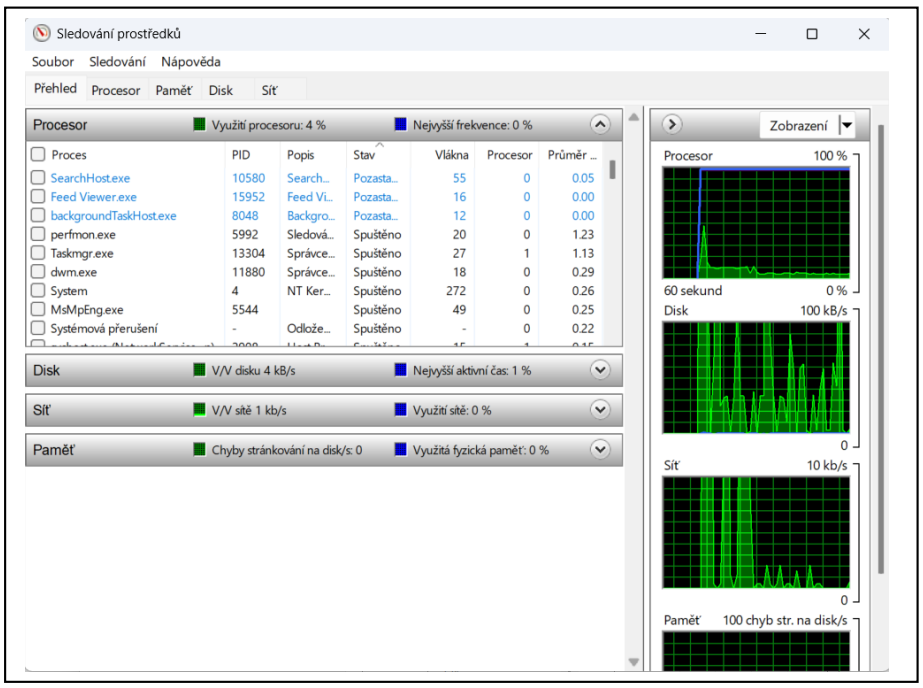
vmstat

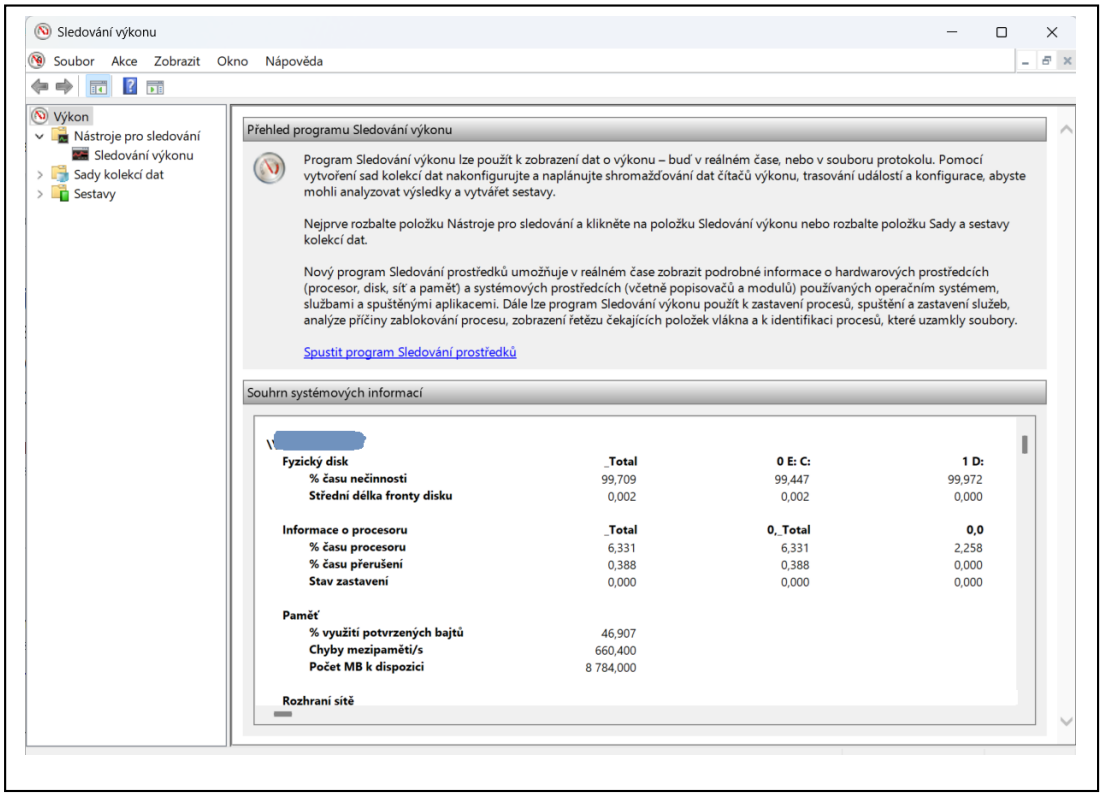
- top: Interaktivní nástroj pro sledování běžících procesů a jejich využití paměti.
- top
- htop: Vylepšená verze příkazu top s barevným rozhraním a dalšími funkcemi.
- Htop

Vstupně-výstupní (I/O) systémy

MS Windows

- **I/O Manager**: Spravuje všechny I/O operace a poskytuje jednotné rozhraní pro komunikaci s hardwarem. Zajišťuje, že I/O požadavky jsou správně směrovány a zpracovány
- **Asynchronní I/O**: Windows podporuje asynchronní I/O, což umožňuje aplikacím pokračovat v práci, zatímco I/O operace probíhá na pozadí. To zvyšuje výkon a efektivitu aplikací
- **Plug and Play (PnP)**: Umožňuje automatickou detekci a konfiguraci nově připojených zařízení bez nutnosti restartu systému
- **Power Management:** Správa napájení umožňuje systému a jednotlivým zařízením přecházet do nízkopříkonových stavů, což šetří energii





Linux

- **Virtuální souborový systém (VFS):** Abstraktní vrstva, která poskytuje jednotné rozhraní pro různé souborové systémy. Umožňuje aplikacím pracovat s různými typy souborových systémů bez nutnosti specifických úprav
- **Asynchronní I/O:** Linux podporuje asynchronní I/O pomocí systémových volání jako aio_read a aio_write, což umožňuje efektivní zpracování I/O operací na pozadí
- **Device Drivers:** Ovladače zařízení v Linuxu jsou modulární a mohou být načítány a uvolňovány za běhu systému, což zvyšuje flexibilitu a rozšiřitelnost
- **Buffer Cache:** Linux používá buffer cache pro ukládání často přistupovaných dat, což zvyšuje výkon systému tím, že minimalizuje počet přístupů na disk

Souborové systémy

MS Windows

- **FAT (File Allocation Table)**: Starší souborový systém, který se používá hlavně na menších discích a vyměnitelných médiích. Má omezení velikosti souborů a diskových oddílů
- **NTFS (New Technology File System)**: Moderní souborový systém, který nabízí pokročilé funkce jako šifrování, kompresi, kvóty a podporu velkých souborů a diskových oddílů. NTFS je standardním souborovým systémem pro většinu verzí Windows
- **exFAT** (Extended File Allocation Table): Navržený pro flash disky a externí úložiště. Podporuje větší soubory než FAT32 a je kompatibilní s více operačními systémy

Linux

- **ext4 (Fourth Extended Filesystem)**: Nejrozšířenější souborový systém v Linuxu, který nabízí vysoký výkon, spolehlivost a podporu velkých souborů a diskových oddílů
- **XFS**: Vysoce výkonný souborový systém, který je vhodný pro servery a aplikace náročné na I/O operace. Podporuje velké soubory a diskové oddíly
- **Btrfs (B-tree File System)**: Moderní souborový systém s pokročilými funkcemi jako snapshoty, kontrolní součty a podpora RAID. Je navržen pro vysokou spolehlivost a snadnou správu

Unix

- **UFS** (Unix File System): Tradiční souborový systém používaný v mnoha Unixových systémech. Nabízí spolehlivost a výkon, ale má omezenou podporu pro moderní funkce
- **ZFS** (Zettabyte File System): Pokročilý souborový systém s funkcemi jako snapshoty, kontrolní součty, komprese a podpora velkých souborů a diskových oddílů. Je navržen pro vysokou spolehlivost a snadnou správu
- **FFS** (Fast File System): Vylepšená verze UFS, která byla vyvinuta pro zvýšení výkonu a snížení fragmentace

Synchronizace přístupu ke sdíleným prostředkům

Pro synchronizaci přístupu ke sdíleným prostředkům a zabránění konfliktům existuje několik metod, které se používají jak ve Windows, tak v Linuxu. Zde je přehled některých z nich:

Windows

- **Mutex (Mutual Exclusion)**: Používá se k zajištění, že pouze jeden proces nebo vlákno má přístup ke sdílenému prostředku v daném okamžiku.
- **Semaphore**: Umožňuje omezený počet procesů nebo vláken přístup ke sdílenému prostředku. Je užitečný pro řízení přístupu k více instancím prostředku.
- **Critical Section**: Používá se k ochraně části kódu, která přistupuje ke sdíleným datům. Je efektivní pro synchronizaci vláken v rámci jednoho procesu.
- **Event**: Synchronizační objekt, který umožňuje procesům nebo vláknům čekat na určitou událost, než pokračují v činnosti.
- **Condition Variables**: Používají se v kombinaci s mutexy k čekání na určité podmínky v rámci více vláken.

Linux

- **Spinlock**: Používá se pro krátké kritické sekce, kde vlákno aktivně čeká (spinuje), dokud nezíská přístup ke sdílenému prostředku
- **Mutex**: Stejně jako ve Windows, mutexy zajišťují, že pouze jedno vlákno má přístup ke sdílenému prostředku v daném okamžiku.
- **Semaphore**: Umožňuje omezený počet vláken přístup ke sdílenému prostředku. Je užitečný pro řízení přístupu k více instancím prostředku.
- **Read-Write Locks**: Umožňují více vláknům číst sdílený prostředek současně, ale zápis je povolen pouze jednomu vláknu najednou.
- **Condition Variables**: Používají se v kombinaci s mutexy k čekání na určité podmínky v rámci více vláken

Tyto metody pomáhají zajistit, že přístup ke sdíleným prostředkům je řízený a nedochází ke konfliktům nebo nekonzistencím.

Příklad použití Mutex v MS Windows:

Viz mutex.txt

Zdroje:

Root.cz, copilot, microsoft.com, wikipedia