IOS - Survival guide 2018

Operační systém (OS):

Je program, resp. kolekce programů, která vytváří spojující mezivrstvu mezi hardware a uživateli a jejich aplikačními programy.

Cíle OS: Do jisté míry protichůdných cílů

- Maximální využití zdrojů počítače
- Jednoduchost použití

Role OS:

- **Správa prostředků**
  - paměť procesor, disk, periferie
- **Tvůrce prostředí pro uživatele a jejich programy**
  - Standartní rozhraní
    - zjednodušuje přenositelnost aplikací
  - Abstrakce
    - Vybavení je složité a různorodé -> potřeba zjednodušit
    - Problém: menší efektivita

OS obsahuje:

- Jádro
- Systémové knihovny a utility
- Grafické nebo textové uživatelské rozhraní

Přesná definice, co vše OS zahrnuje však neexistuje.

Jádro OS:

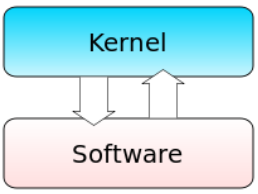
- Jádro OS je nejnižší a nejzákladnější část OS.
- Zavádí se první a běží po celou dobu běhu počítačového systému.
- Navazuje přímo na hardware.
- Běží obvykle v privilegovaném režimu.
- Zajišťuje základní správu prostředků a tvorbu prostředí.

Rozlišujeme dva typy rozhraní:

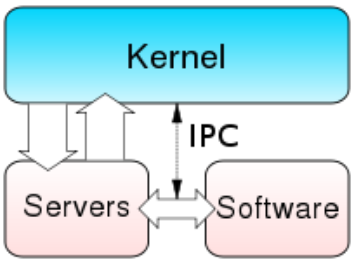
- **Kernel Interface:** přímé volání jádra specializovanou instrukcí
- **Library Interface:** volání funkcí ze systémových knihoven, které mohou (ale nemusí) vést na volání služeb jádra

Typy jader:

- **Monolitické jádro:** Jehož veškerý kód běží ve stejném (jaderném) paměťovém prostoru, který se označuje jako kernel space. Tím se liší od tzv. mikrojádra, které většinu tradičních činností monolitického jádra, jako je třeba správa souborových systémů, implementuje v procesech, které běží v uživatelském paměťovém prostoru. Přestože jsou monolitická jádra psána tak, aby byla činnost jednotlivých subsystémů oddělená, jsou jednotlivé části velice silně provázány. A navíc, protože sdílejí stejný paměťový prostor, může chyba v jednom subsystému zablokovat jiný, nebo dokonce shodit celé jádro. Na druhou stranu, pokud je dbáno na správnou implementaci jednotlivých částí, je monolitické jádro velice efektivní.

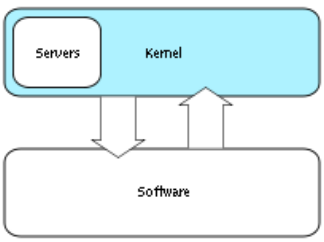


- **Mikrojádro:** je velmi malé a obsahuje jen nejzákladnější, čímž se minimalizuje objem běžícího kódu v privilegovaném režimu. Ostatní potřebné části jádra jsou řešeny v uživatelském prostoru jako běžné procesy (resp. démoni, u mikrojader se označují *servery*), například správa souborového systému, ovladače zařízení, podpora protokolů pro počítačové sítě a další.



  - Výhody: flexibilita, zabezpečení
  - Nevýhody: vyšší režie

- **Hybridní jádra:** Mikrojádra rozšířená o kód, který by mohl být implementován ve formě serveru, ale je za účelem menší režie těsněji provázán s mikrojádrem a běží v jeho režimu.



- **Exojádra:** Experimentální jádra poskytující velmi nízké rozhraní zaměřené hlavně na bezpečné sdílení prostředků. Doplněno o knihovny implementující služby jinak nabízené běžně jádrem. Prozatím se neprosadilo v praxi.

Klasifikace počítačů:

- **Podle účelu:**
  - Univerzální
  - Specializované
- **Podle výkonnosti:**
  - Vestavěné
  - Osobní
  - Pracovní stanice
  - Servery
  - Střediskové
  - Superpočítače

Klasifikace OS:

- **Podle účelu:**
  - Univerzální (Unix, Windows, Linux..)
  - Specializované:
    - Real-time
    - Databáze, web
    - Mobilní zařízení (Android, iOS, Windows Phone..)
- **Podle počtu uživatelů:**
  - Jednouživatelské (MS-DOS..)
  - Víceuživatelské (UNIX, Windows..)
- **Podle počtu současně běžících úloh:**
  - Jedno-úlohové
  - Víceúlohové (Multitasking)

Komunikace s jádrem

Služby jádra – operace, jejichž realizace je pro procesy zajišťována jádrem. Explicitně je možno o provedení určité služby žádat prostřednictvím systémového volání.

HW přerušení:

Mechanismus, kterým HW zařízení asynchronně oznamuji jádru vznik události, které je zapotřebí obsloužit.
Synchronní: proces-jádro
Asynchronní: hardware-jádro

- Žádosti o HW přerušení přichází jako elektrické signály do **řadiče přerušení.**
- Přerušení mají přiřazeny **priority**, dle kterých se oznamují procesoru.
- Přijme-li procesor přerušení s určitým číslem vyvolá odpovídající obslužnou rutinu na základě tabulky přerušení, přičemž automaticky přejde do privilegovaného režimu.
- Řadič může být naprogramován tak, že některá přerušení jsou **maskována**, případně jsou maskována všechna přerušení až po určitou prioritní úroveň.

Přerušení také vznikají přímo v procesoru:

- **Trap:** po obsluze se pokračuje další instrukcí (breakpoint, přetečení apod.)
- **Fault:** po obsluze se opakuje instrukce, která výjimku vyvolala (např. výpadek stránky, dělení nulou, chybný operační kód apod.)
- **Abort:** nelze určit, jak pokračovat, provádění se ukončí (např. chyby HW detekované procesorem).

Správa souborů

Pevný disk:

- Využívá tzv. **sekvenční přístup**
- **Diskový sektor:** nejmenší jednotka, kterou disk umožňuje načíst/zapsat.
- **Velikost sektoru:** typicky 512B, u novějších 4096B (s emulací 512B).
- Adresace sektorů:
    - CHS = Cylinder, Head (typicky 1-6 hlav), Sector
    - LBA = Linear, Block, Address (číslo 0..N)

Pro připojení disků se používá řada různých **diskových rozhraní:** primárně ATA/SATA či SCSI/SAS, ale také USB, FireWire, FibreChannel, Thunderbolt aj. Liší se zejména rychlostí, počtem připojených zařízení, max. délkou kabelů, architekturou připojení, podporovanými příkazy.

Hierarchie pamětí:

- **primární paměť:** RAM (nad ní ještě registry, cache L1-L3)
- **sekundární paměť:** pevné disky, SSD (mají také své cache)
- **terciární paměť:** pásky, CD, DVD, ...

Parametry pevných disků:

- **Přístupová doba** = doba vestavění hlav + rotační zpoždění
    - Kapacita
    - Průměrná doba přístupu
    - Otáčky
    - Přenosová rychlost

SSD:

Založeno na nevolatilních pamětech **NAND flash**, ale vyskytují se i řešení založena na **DRAM** či na kombinacích.

- Výhody:
  - Náhodný přístup
  - Rychlý náběh
  - Větší přenosové rychlosti
  - Tichý
  - Magnetická a mechanická odolnost
  - Nižší spotřeba (neplatí pro DRAM)
- Nevýhody:
  - Vyšší cena za jednotku prostoru
  - Omezený počet přepisů
  - Možné komplikace se zabezpečením

Problematika zápisu u SSD:

SSD jsou organizovány do **stránek** (typicky 4KiB) a ty do **bloků** (typicky 128 stránek). Prázdné stránky lze zapisovat jednotlivě, ale pro přepis je nutno načíst celý blok do VP, v ní přepsat, poté na disku vymazat a pak zapsat.

- Problém je menší při sekvenčním než při náhodném zápisu do souboru.

Aby se problém minimalizoval může SSD mít více stránek, než je oficiální kapacita. Řadič SSD může přepisovanou stránku zapsat na jinou pozici, případně i přesouvá dlouho neměněné stránky, aby mohl některé bloky uvolnit.

Zabezpečení disků:

- **ECC (Error Correction Code)** – k užitným datům sektoru si ukládá redundantní data, která umožňují opravu nebo alespoň detekci chyb.
- **S.M.A.R.T. data** – Disk si shromažďuje statistiky o svém provozu, které lze použít k předpovídání/diagnostice chyb.
- Rozpoznávání a označování vadných bloků. (Raději vyměnit disk)

Disková pole:

- **RAID 0** – Více disků tvoří kapacitu jednoho velkého disku. Žádná ochrana.
- **RAID 1** – Zrcadlení. Obsah se současně zaznamenává na 2 disky.
- **RAID 2** – Data jsou uložena po bitech mezi jednotlivé disky. Jsou zabezpečena pomocí Hammingova kódu na zvláštním disku.
- **RAID 3** – Je použito N+1 disků. Na N disků se ukládají data a na poslední disk je uložena parita.
- **RAID 4** – Data jsou uložena po blocích na každém disku a parita na zvláštním disku.
- **RAID 5** – Jako RAID 4, ale parita je rozložena na všech discích střídavě.
- **RAID 6** – Stejný jako RAID 5, ale parita je uložena 2x, vyrovná se i se ztrátou 2 disků.

Uložení souboru na disku:

Alokační blok – Skupina pevného počtu sektorů následujících logicky i fyzicky za sebou, která je nejmenší jednotkou, kterou OS čte/zapisuje.

Fragmentace:

- **Externí fragmentace** – Na disku vzniká posloupnost volných oblastí a oblastí použitých různými soubory což má dva důsledky:
  - Vzniknou oblasti příliš malé na to, aby se dali využít.
  - Při nespojitém přidělování prostoru po alok. blocích výše uvedený problém nevzniká, ale data souboru jsou na disku uložena nespojitě -> složitější a pomalejší přístup.

  #### Techniky k minimalizaci externí fragmentace:

  - Rozložení souboru po disku
  - Předalokace – Alokuje víc místa, než je potřeba.
  - Odložená alokace – Odkládá zápis, než se nasbírá více požadavků a lepší podvědomí, kolik je třeba alokovat.
  - Defragmentace

- **Interní fragmentace** – nevyužité místo v posledním přiděleném alok. bloku -> plýtvání místem.
  - Některé souborové systémy umožňují sdílení posledních alokačních bloků.

Přístup na disk:

Prostřednictvím I/O portů a/nebo paměťově mapovaných I/O operací se řadiči.

Přenos z/na disk je typicky řízen řadičem disku s využitím technologie přímého přístupu do paměti (DMA). O ukončení operací či chybách informuje řadič procesor (a na něm OS) pomocí přerušení.

Plánování přístupu na disk:

Pořadí bloků čtených/zapisovaných na disk ovlivňuje plánovač diskových operací. Přicházející požadavky na čtení/zápis jsou ukládány do VP a jejich pořadí je případně měněno tak, aby se minimalizovala režie diskových operací. Slouží k tomu různé algoritmy:

- **Výtahový algoritmus** – pohybuje hlavičkami od středu k okraji ploten a zpět a vyřizuje požadavky v pořadí odpovídajících pozici a směru pohybu hlaviček.
- **Circular SCAN** – Vyřizuje požadavky vždy při pohybu jedním směrem -> rovnoměrnější.
- **LOOK** – Pohybuje se jen v mezích daných aktuálními požadavky -> nižší prům. doba přístupu.

Plánovač může sdružovat operace, vyvažovat požadavky různých uživatelů, implementovat priority nebo časová omezení, odkládat operace v naději, že je bude možno později propojit apod..

Logický disk – Rozdělení fyzického disku na logické disky tzv. diskové oblasti.

Virtuální souborový systém (VFS) – Vrstva, která zastřešuje všechny použité souborové systémy a umožňuje s nimi pracovat jednotným a abstraktním způsobem. Odděluje vyšší vrstvy OS od konkrétní implementace jednotlivých operaci na jednotlivých souborových systémech. Pro popis souborů používá rozšířené i-uzly (tzv. v-uzly).

Network file systém (NFS) – Transparentně zpřístupňuje soubory uložené na vzdálených systémech.

- Umožňuje **kaskádování:** lokální připojení vzdáleného adresářového systému do jiného vzdáleného adresářového systému.
- Autentizace často prostřednictvím UID a GID

Spooling:

- VP pro zařízení, které neumožňují prokládané zpracování dat různých procesů.
- Výstup je proveden do souboru, požadavek na jeho fyzické zpracování se zařadí do fronty, uživatelský proces může pokračovat a zpracování dat se provede, až na ně přijde řada.

Žurnálování:

- Slouží pro záznam modifikovatelných metadat před jejich zápisem na disk.
- Umožňuje spolehlivější a rychlejší návrat do konzistentního stavu po chybách.
- Implementován jako **cyklický přepisovaný buffer** ve speciální oblasti disku.
- Operace pokryté žurnálováním jsou atomické – vytváří **transakce.**

Implementace na základě dokončení transakcí (REDO):

- Sekvence operací se uloží do žurnálu mezi značky označující začátek a konec transakce.
- Poté se dílčí operace provádí na disku.
- Uspějí-li všechny dílčí operace, transakce se ze žurnálu uvolní.
- Při selhání se dokončí všechny transakce, které jsou v žurnálu.

Implementace na základě anulace transakcí (UNDO):

- Záznam dílčích operací do žurnálu a na disk se prokládá.
- Proběhne-li celá transakce, ze žurnálu se uvolní.
- Při chybě se eliminují nedokončené transakce.

UNDO a REDO je možno kombinovat.

Implementace žurnálování musí zajišťovat správné pořadí zápisu operací.

Alternativy k žurnálování:

Copy-on-write – nejprve zapisuje nová data na disk, pak je zpřístupní.

Soft updates – Sleduje závislosti mezi daty a metadaty a zaručuje zápis na disk v takovém pořadí, aby v kterékoli době byl obsah disku konzistentní.

Log-structured file systems – celý souborový systém má charakter logu s obsahem disku vždy dostupným přes poslední záznam.

Organizace souborů:

- **Kontinuální uložení:**
  - Problém se zvětšováním souborů díky externí fragmentaci nebo obsazení prostoru hned na koncem souboru.
- **Zřetězené seznamy bloků:**
  - Při přístupu k náhodným blokům či ke konci souboru nutno projít celý soubor.
  - Chyba kdekoliv na disku může způsobit ztrátu velkého objemu dat.
- **FAT (File allocation table):**
  - Seznamy uložené ve speciální oblasti disku.
  - Nachází se na disku a je pro vyšší spolehlivost rozdvojená.
  - Má položku pro každý blok.
  - Vedou do ní odkazy z adresářů.
  - Položky tabulky mohou být zřetězeny do seznamu, příp. označeny jako volné či chybné.
  - Opět vznikají problémy s náhodným přístupem.
- **B+ stromy:**
  - Vkládá se na listové úrovni. Dojde-li k přeplnění list se rozštěpí a přidá se nový odkaz do nadřazeného vnitřního uzlu. Při přeplnění se pokračuje směrem ke kořeni. Nakonec může být přidán nový kořen.
  - Ruší se od listové úrovně. Při nenaplnění minimální kapacity, pokus o přerozdělení. Klesne-li zaplněnost sousedních uzlů na polovinu, uzly se spojí a ruší se jeden odkaz na nadřazené úrovni. Pokračuje se směrem ke kořeni, nakonec může jedna úroveň ubýt.

V moderních systémech se často indexuje alokovaný prostor po tzv. **extentech**, tj. posloupnost proměnného počtu bloků jdoucích za sebou logicky v souboru a uložených i fyzicky na disku za sebou:

- Rychlejší práce s velkými soubory.
- Snadno se kombinují s B+ stromy.
- Pro malé soubory může B+ strom představovat zbytečnou režii. Používá se přímé uložení v i-uzlu nebo přímé odkazy na extenty z i-uzlu.

Organizace volného prostoru:

Bitová mapa s jedním bitem pro každý blok. Umožňuje zrychlit vyhledávání volné souvislé oblasti pomocí bitového maskování.

Další způsoby:

- Seznam – Zřetězení volných bloků
- Označení (zřetězení) volných položek v tabulce FAT
- B+ strom (adresace velikostí nebo offsetem)
- Někdy se také eviduje jen obsazený prostor podle pozice na disku
- Extenty

Deduplikace:

- Snaha odhalit opakované ukládání těchže dat, uložit jednou a odkázat vícenásobně.
- Podporována na různých úrovních: byty, bloky, extenty, soubory.
- Může uspořit diskový prostor
- Při menším objemu duplikace může naopak zvýšit spotřebu procesorového času, paměťového i diskového prostoru.

Princip montování disků:

- Všechny soubory jsou v jednom stromu adresářů.
- V systému je pouze jeden kořenový logický disk, další lze připojit již do adresářového stromu.

Datové struktury a algoritmy pro vstup/výstup

I/O buffering:

- **Buffer** = Vyrovnávací paměť
- Cílem je minimalizace pomalých operací s periferiemi (typicky s disky).
- Dílčí vyrovnávací paměti mývají velikost alok. bloku a jsou sdružovány do kolekce pevné či proměnné velikosti umožňující snadné vyhledávání.

Sparese files – „řídké soubory“:

- Vnikají nastavením pozice za konec souboru a zápisem.
- Bloky, do kterých se nezapisovalo nejsou alokovány a nezabírají diskový prostor. Při čtení se považují za vynulované.

Ovladač – sada programů pro řízení určitého typu zařízení.

Terminály – Fyzická nebo logická zařízení umožňující (primárně) textový vstup/výstup systému po řádcích.

Roury (pipes):

- Jeden z typů speciálních souborů.
- Rozlišujeme roury **pojmenované** a **nepojmenované**.
- Reprezentují jeden z mechanizmů meziprocesové komunikace.
- Implementace: kruhový buffer s omezenou kapacitou.
- Procesy komunikující přes rouru (producent a konzument) jsou synchronizovány.

Sokety – Umožňují síťovou komunikaci.

Správa procesů

Dispatcher – přepínání kontextu

Plánovač – přiděluje CPU procesům

Správa paměti – přiděluje paměť

Meziprocesová komunikace (IPC) – signály, RPC

Proces = běžící program

Je definován:

- Identifikátorem (PID)
- Stavem jeho plánování
- Programem, kterým je řízen
- Obsahem registrů
- Daty a zásobníkem

Stavy procesu:

- Vytvořený (new) – ještě neinicializovaný
- Připravený (ready) – mohl by běžet, ale nemá CPU
- Běžící (running) – používá CPU
- Čekající (waiting) – čeká na událost (např. dokončení read)
- Odložený (terminated) – zamrazený signálem SIGSTOP

V OS bývá proces reprezentován strukturou **PCB (Process Control Block)**, ta zahrnuje:

- Identifikátory
- Stav
- Obsah registrů
- Plánovací informace (priorita, ...)
- Informace spojené se správou paměti (tabulky stránek, ...)
- Informace spojené s účtováním (spotřeba procesoru, …)
- Využití I/O zdrojů (otevřené soubory, používaná zařízení, …)

Uživatelský adresový prostor přístupný procesu:

- Kód
- Data
- Zásobník
- Knihovny, sdílená paměť

Kontext procesu = stav procesu

Rozlišujeme:

- **Uživatelský kontext**
  - Kód
  - Data
  - Zásobník
  - Sdílená data
- **Registrový kontext**
- **Systémový kontext**
  - Uživatelská oblast
  - Položka tabulky procesů
  - Tabulka paměťových regionů procesu…

Hierarchie procesů v Unixu

- Předek všech procesů je **init** s PID=1
- Existují **procesy jádra**, jejichž předkem init není:
  - Jejich kód je součástí jádra
  - Vyskytuje se i proces s PID=0: obálka pro vlákna jádra+ idle smyčka
  - Proces jádra kthreadd, který spouští ostatní procesy jádra a je jejich předkem.
- Pokud procesu skončí předek, jeho předkem se automaticky stane init

Zombie proces: Je takový proces, který skončil, ale i nadále se vyskytuje v tabulce procesů. Protože jeho rodič nepřevzal jeho návratový kód. Pokud tak učiní a převezme jeho návratový kód, OS odebere záznam z tabulky procesů, uvolní jemu přidělené prostředky a uvolní jeho PID. Procesy, co patři mezi zombie delší dobu jsou obvykle chybou a způsobují únik prostředků.

Start systému

Typická posloupnost akcí při startu systému:

1. **Firmware** (BIOS)
2. Načtení a spuštění **zavaděče OS**, někdy v několika fázích
3. Načtení **inicializačních funkcí jádra** a samotného jádra, spuštění inicializačních funkcí
4. Inicializační funkce jádra vytvoří **proces 0**, ten vytvoří další procesy jádra a proces init
5. **Init** načítá inicializační konfigurace a spouští další démony a procesy

Démon: je proces, který je dlouhodobě spuštěn a není v přímém kontaktu s uživatelem. Jeho úkolem je vyčkávat na nějakou událost, tu poté obsloužit a zajišťovat tak různé úkoly bez nutnosti interakce s uživatelem.

Plánování procesů:

- **Nepreemptivní plánování** – pouze běžící proces se sám může vzdát CPU
- **Preemptivní plánování** – proces se může sám vzdát CPU, ale také mu může být odebrán na základě přerušení
- Vlastní přepnutí kontextu řeší na základě rozhodnutí plánovače tzv. **dispatcher**
- Plánování může být též ovlivněno systémem **swapování**

Přepnutí procesu (kontextu):

- Dispatcher odebere procesor procesu A a přidělí ho procesu B, což zahrnuje:
  - Úschovu stavu registrů v rámci procesu A do PCB
  - Úpravu některých řídicích struktur v jádře
  - Obnovu stavu registrů v rámci procesu B z PCB
  - Předání řízení na adresu, kde bylo dříve přerušeno provádění procesu B
- Neukládá se/Neobnovuje se celý stav procesů
- Přepnutí trvá typicky stovky až tisíce instrukcí, interval mezi přepínáním musí být tedy volen tak, aby režie přepnutí nepřevážila běžná běh procesů.

Klasické plánovací algoritmy:

- **FCFS (First come, first served):**
  - Procesy čekají na přidělení procesoru ve FIFO frontě
  - Při vzniku procesu, jeho uvolnění z čekání nebo vzdá-li se proces procesoru, je tento proces zařazen na konec fronty
  - Procesor se přiděluje na začátku fronty
  - Algoritmus je nepreemptivní a k přepnutí kontextu dojde pouze tehdy, pokud se běžící proces vzdá procesoru.
- **Round-robin** – preemptivní obdoba FCFS:
  - Pracuje podobně jako FCFS, navíc má ale každý proces přiděleno **časové kvantum**, po jehož vypršení je mu odebrán procesor
- **SJF (Shortest Job First):**
  - Přiděluje procesor procesu, který požaduje nejkratší dobu pro své další provádění na procesoru bez I/O operací
  - Nepreemptivní algoritmus
  - Minimalizuje průměrnou dobu čekání, zvyšuje propustnost systému
  - Nutno znát dopředu dobu běhu procesu na procesoru nebo mít možnost tuto dobu rozumně odhadnout na základě předchozího chování
  - Používá se ve specializovaných systémech
  - Hrozí **stárnutí** (hladovění) procesů mající dlouhé výpočetní fáze

- **SRT (Shortest Remaining Time):**
    - Obdoba SJF s preempcí
- **Víceúrovňové plánování:**
    - Procesy jsou rozděleny do skupin (typicky podle **priority**, ale lze i jinak)
    - V rámci každé skupiny může být použit jiný dílčí plánovací algoritmus (FCFS, …)
    - Je také použit algoritmu, který určuje, ze které skupiny bude vybrán proces, který má aktuálně běžet (často na základě priority)
    - Hrozí hladovění některých nízko prioritních procesů
- **Víceúrovňové plánování se zpětnou vazbou:**
    - Víceúrovňové plánování se skupinami procesů rozdělenými dle priorit.
    - Proces nově připravený běžet je zařazen do fronty s nejvyšší prioritou, postupně klesá do nižších prioritních front, nakonec plánován round-robin na nejnižší úrovni.
    - Používají se varianty, kdy je proces zařazen do počáteční fronty na základě své statické priority. Následně se může jeho dynamická priorita snižovat, spotřebovává-li mnoho procesorového času, nebo naopak zvyšovat, pokud hodně čeká na I/O operacích.
- **Víceúrovňové prioritní plánování se 100 základními statickými prioritními úrovněmi:**
    - Priority 1-99 pro procesy reálného času plánované **FCFS** nebo **round-robin**
    - Priorita 0 pro běžné procesy plánované tzv. **CFS** plánovačem
    - V rámci úrovně 0 se dále používají podúrovně v rozmezí -20 (nejvyšší) až 19 (nejnižší)
    - V rámci úrovně 0 dále rozlišujeme plánování pro **běžné, dávkové** (delší kvantum) a **idle** procesy (jen pokud nejsou jiné úlohy)
    - Základní prioritní úroveň a typ plánování mohou ovlivnit procesy s patřičnými právy.
- **CFS (Completely Fair Scheduler)**
    - Snaží se explicitně každému procesu poskytnout odpovídající procento strojového času (dle jejich priorit)
    - Vede si u každého procesu údaj o tom, kolik (virtuálního) **procesorového času** strávil.
    - Navíc si vede údaj o **minimálním stráveném procesorovém čase**, který dává nově připraveným procesům.
    - Procesy udržuje ve **vyhledávací stromové struktuře** (red-black strom) podle využitého procesorového času.
    - Vybírá jednoduše proces s **nejmenším stráveným časem**.
    - Procesy nechává běžet po dobu časového kvanta spočteného na základě priorit a pak je zařadí zpět do plánovacího stromu.
    - Obsahuje podporu pro **skupinové plánování**: Může rozdělovat čas spravedlivě pro procesy spuštěné z různých terminálů

- **Víceúrovňové prioritní plánování se zpětnou vazbou na základě interaktivity:**
  - **32 prioritních úrovní:** 0 – nulování volných stránek, 1 – 15 běžné procesy, 16 – 31 procesy reálného času.
  - **Základní priorita** je dána staticky nastavenou kombinací plánovací třídy a plánovací úrovně v rámci třídy.
  - Systém může prioritu běžných procesů **dynamicky zvyšovat a snižovat:**
    - Zvyšuje prioritu procesů spojených s oknem, které se dostane na popředí
    - Zvyšuje prioritu procesů spojených s oknem, do kterého přichází vstupní zprávy (myš, časovač, klávesnice…)
    - Zvyšuje prioritu procesů, které jsou uvolněny z čekání (např. I/O operaci)
    - Zvýšená priorita se snižuje po každém vyčerpání kvanta o jednu úroveň až do dosažení základní priority

Inverze priorit:

- Nízko prioritní proces si naalokuje nějaký zdroj, více prioritní procesy ho předbíhají a nemůže dokončit práci s tímto zdrojem.
- Časem tento zdroj mohou potřebovat více prioritní procesy, jsou nutně zablokovány a musí čekat na nízko prioritní proces.
- Pokud v systému jsou v tomto okamžiku středně prioritní procesy, které nepotřebují daný zdroj, pak poběží a budou dále předbíhat nízko prioritní proces.
- Tímto způsobem uvedené středně a nízko prioritní procesy získávají efektivně vyšší prioritu.

Inverze priorit může zvýšit odezvu systému.

Možnosti řešení:

- **Priority ceiling:** procesy v kritické sekci získávají nejvyšší prioritu.
- **Priority inheritance:** proces v kritické sekci, který blokuje výše prioritní procesy, dědí (po dobu běhu v kritické sekci) prioritu čekajícího procesu s největší prioritou.
- **Zákaz přerušení po dobu běhu v kritické sekci** (na jednoprocesorovém systému): proces v podstatě získá vyšší prioritu než všichni ostatní.

Další výraznou komplikaci plánování představují:

- **Víceprocesorové systémy** – nutnost vyvažovat výkon, respektovat obsah cache procesorů.
- **Hard real-time systémy** – nutnost zajistit garantovanou odezvu některých akcí.

Vlákna:

- Odlehčený proces
- V rámci jednoho klasického procesu může běžet více vláken.
- Vlastní obsah registrů a zásobník.
- Sdílí kód, data a další zdroje.
- Výhody: rychlejší spuštění, přepínání.

Komunikace procesů (IPC = Inter-Process Communication):

- Signály
- Roury
- Zprávy
- Sdílená paměť
- Sockety
- RPC = Remote Procedure Call

Signály:

- **Signál** je číslo zaslané procesu prostřednictvím pro to zvláštně definovaného rozhraní. Signály jsou generovány:
  - **Při chybách** (aritmetická chyb, chyba práce s pamětí...)
  - **Externích událostech** (vypršení časovače, dostupnost I/O…)
  - **Na žádost procesu** – IPC (kill…)
- Signály často vznikají **asynchronně** k činnosti programu – není tedy možné jednoznačně předpovědět, kdy daný signál bude doručen.
- **Nutno pečlivě zvažovat obsluhu**, jinak mohou vzniknout „záhadné“, zřídka se objevující a velice špatně laditelné chyby.
- Obsluhu signálu lze **předefinovat** (většinou).

Synchronizace procesů

- **Současný přístup** několika paralelních procesů ke **sdíleným zdrojům** může vést k nekonzistencím zpracovávaných dat.
- **Časově závislá chyba (race condition)** – chyba vznikající při přístupu ke sdíleným zdrojům kvůli různému pořadí provádění jednotlivých paralelních výpočtů v systému, tj. kvůli jejich různé relativní rychlosti.
- Zajištění konzistence dat vyžaduje mechanismy **synchronizace procesů** zajišťující **správné pořadí** provádění spolupracujících procesů.

Kritická sekce:

- **Sdílenými kritickými sekcemi** daných procesů rozumíme ty úseky jejichž řídící program přistupuje ke sdíleným zdrojům, jehož provádění jedním procesem vylučuje současné provádění ostatními procesy.
- **Problém kritické sekce** rozumíme problém zajištění korektní synchronizace procesů na množině sdílených kritických sekcí, což zahrnuje:
    - Vzájemné vyloučení – nanejvýš jeden (někdy víc) proces je v daném okamžiku v dané množině sdílených KS.
    - Dostupnost KS:
        - Je-li KS volná, proces nemůže neomezeně čekat na přístup do ní.
        - Je zapotřebí se vyhnout:
            - Uváznutí
            - Blokování
            - Stárnutí

Problémy vznikající v kritické sekci:

- **Data race (časově závislá chyba nad daty)** – dva přístupy ke zdroji s výlučným přístupem ze dvou procesů bez synchronizace, alespoň jeden přístup je pro zápis
- **Blokování (blocking)** při přístupu do KS – situace, kdy proces, jenž žádá o vstup do kritické sekce, musí čekat, přestože je kritická sekce volná a ani žádný další proces o ni nežádá.
- **Stárnutí (aneb hladovění)** – situace, kdy proces čeká na podmínku, která nemusí nastat. V případě kritické sekce je umožnění vstupu do ní.
- **Livelock** – situace, kdy procesy běží, ale provádějí jen omezený úsek kódu, ve kterém opakovaně žádají o určitý zdroj (aktivní čekání).

Semafory:

Synchronizační nástroj nevyžadující aktivní čekání – resp. Aktivní čekání se v omezené míře může vyskytnout uvnitř implementace operací nad semaforem, ale ne v kódu, který tyto operace využívá.

Jedná se o celočíselnou proměnnou přístupnou dvěmi základními **atomickými** operacemi:

- **Lock** – zamknutí, volající proces čeká, dokud není možné operaci úspěšně dokončit.
- **Unlock** – Odemknutí semaforu.

Pokud je hodnota semaforu S <= 0, tak je semafor zamčený a S > 0 je odemčený.

Používají se také:

- **Read-write zámky** – pro čtení lze zamknout vícenásobně
- **Reentrantní zámky** – proces mže stejný zámek zamknout opakovaně
- **Mutexy** – binární semafory, které mohou být odemknuty pouze těmi procesy, které je zamkly

Monitory – Jeden z vysokoúrovňových synchronizačních prostředků. Zapouzdřuje data, má definované operace, jen jeden procesů může provádět nějakou operaci nad chráněnými daty.

Deadlock (uváznutí):

Situace, kdy si proces zabere první zdroj a poté čeká na uvolnění dalšího zdroje, ovšem jiný proces, který tento zdroj vlastní čeká na uvolnění prvního zdroje -> paradox.

Nutné podmínky pro uváznutí (Coffmanovy podmínky):

1. **Vzájemné vyloučení při používání prostředků.**
    - Prostředek může v jednom okamžiku používat jen 1 proces.
2. **Drž a čekej**
    - Proces může žádat o další prostředky, i když už má nějaké přiděleny.
3. **Neodnímatelnost**
    - Jakmile proces vlastní prostředek, nelze mu ho bezpečně odejmout, musí ho vrátit sám.
4. **Cyklické čekání**
    - Pokud procesy na sebe čekají ve smyčce, dojde k deadlocku.

Pokud aspoň jedna z podmínek není splněna, nemůže dojít k uváznutí.

Řešení:

- **Předcházení** – napadení jedné z Coffmanových podmínek
    - Používat sdílené prostředky bez vzájemného vyloučení.
    - Proces musí zažádat o prostředky, které bude potřebovat najednou, buď dostane všechny nebo žádný.
    - Proces může žádat o prostředky pouze pokud žádné nevlastní.
- **Vyhýbání** – OS vyhoví žádosti o přidělení prostředku pouze pokud zůstane v bezpečném stavu. (Nevzniknou **cyklické závisloti**)
- **Detekce a zotavení** – Hledání kružnic v orientovaném grafu, pokud tam je, nastalo uváznutí a je třeba ho řešit:
    - Odebráním prostředků
    - Zabíjením procesů
    - Rollback

Správa paměti

- **Logický adresový prostor (LAP)** – virtuální adresový prostor, se kterým pracuje procesor při provádění kódu (každý proces i jádro má svůj).
- **Fyzický adresový prostor (FAP)** – adresový prostor fyzických adres paměti (společný pro všechny procesy i jádro).

MMU (Memory Managment Unit) = HW jednotka pro překlad logických adres na fyzické (dnes součástí procesoru):

- Využívá speciálních registrů a případně i hlavní paměti systému; pro urychlení překladu může obsahovat různé VP např. TLB
- Pokud si proces zažádá o paměť pomocí LAP, tak jádro si musí zapamatovat jakou adresu jakému procesu přidělil a jak ji překládat na FAP.
    - Spojité bloky (contiguous memory allocation)
    - Segmenty
    - Stránky
    - Kombinace výše uvedeného