Funkce operačního systému, struktura a rozhraní operačního systému, mikrojádro.

Funkce

Abstrakce HW (vylepšení přístupu k HW pro programátora)

- Sjednocení a abstrakce zařízení,
  - Ovladače a jejich vrstvení (VFS)
  - Síťová komunikace
- Ulehčení programování,
  - Scheduler
  - Soubor namísto disk. Bloku
  - Virtuální paměť

Správa zdrojů

- Zabezpečení
  - Konkurence programů (pooling tiskárny)
  - Nekorektní použití, poškození špatným programem
- Efektivní sdílení (paměť, scheduler)
- Preempce

Rozhraní

- Uživatelské
  - Soubor aplikačních programů (*busybox*: shell, editory, cp, mv…)
  - Textové / Grafické rozhraní
- Programové
  - Soubor systémových volání

Struktura

- **monolitcké** systémy - hlavní program, obslužné procedury, podpůrné procedury
- **vrstvené** systémy - hierarchie vrstev, nejnižší je holý počítač, nejvyšší je aplikační program, používají mikrojádro

funkční hierarchie - někdy je problém rozdělit do vrstev podle úrovně abstrakce, proto dělení do vrstev podle funkčnosti

- klient-server - obsahuje mikrojádro, které poskytuje pouze základní funkce, většinu práce dělají servery, které jsou oddělené od jádra
- objektově orientovaná struktura - jádro spravuje řadu objektů (zastupují soubory, HW zařízení, ...), mezi objekty jsou tzv. capability = odkaz na objekt + množina práv definujících operace

Mikrojádro

Úplně nejmenší abstrakce nad holým strojem. Běží exkluzivně v privilegovaném režimu (i z toho plyne, že poskytuje jen ty nejželeznější služby – přerušení, hw, vlákna, meziprocesová komunikace apod.)

Všechno ostatní běží jako mikroservery nad mikrojádrem, v uživatelském režimu.

Případová studie OS Linux

Historie

Linus Torvalds, 1991, inspirace Minixem (Bez minixu to nejdřív vůbec nešlo, až po čase dospěli vývojáři k samoreprodukovatelnosti (přeložení linuxu na linuxu))

Charakteristika Linuxu

Linux je jádro operačního systému – není to operační systém (chybí mu zavaděč, základní uživatelské programy…).

- Unixový typ
- Preemptivní víceúlohový, víceuživatelský, víceprocesorový,
- Velmi rychle se rozvíjející, velmi rozšířený
- Velmi škálovatelný, všestranný, multiplatformní (openwrt … mobily … televize … pc … servery)
- Otevřený

Charakteristika programů pro Linux

- Úzce zaměřené
- Zpravidla otevřené (GNU…)
- KISS (příklad: tar)
- Pajpovatelnost

Případová studie: Ubuntu

- Operační systém pro osobní počítače a servery
- Postavený na Linuxovém jádře, jmenovitě se k Linuxu nehlásí („Ubuntu je operační systém“, „jádro Ubuntu“).
- Komunitně vyvíjený
- Zdarma
- Mnoho jazykových mutací (i česky)
- „Vše v jednom“ – na jednom cd (pravidlo) je instalátor OS i mnoha uživatelských programů pro různé účely (po instalaci je počítač hned připraven k práci)
- Vývoj a správa financována od jednoho sponzora a z placených služeb (online disk, podpora, nástroje pro správu serverů apod.)
- Nejen produkt, ale i komunita kolem něj, vlastní filosofie
- Příklad: servery Wikimedia

Alternativní P. S.

- Linux a busybox (minimální OS)

Zavedení operačního systému

BIOS

- program přítomný ve vestavěné paměti HW (většinou na základní desce)
- provádí testy a nastavení HW
- vybere zaváděcí jednotku

- načte první sektor (MBR), kde je umístěn program zavaděče a provede skok na adresu jeho programu, čímž mu předá řízení

EFI

- Extensible firmware interface
- Evoluce (náhrada) BIOSu
- Například podporuje grafické rozhraní a menu, nemá omezení BIOSu (16 bitový režim procesoru, 1 MB adresovatelného místa)
- Rozhraní mezi operačním systémem a firmwarem hardwaru

Zavaděč

- pro Linux LILO nebo GRUB (LILO musí dostat pevnou adresu, GRUB už umí číst většinu filesystémů a hledá podle jména)
- dává možnost zvolit startující OS a zavést parametry jádra
- načte jádro operačního systému do paměti a spustí ho

Jádro OS

- detekuje hardware a odpovídajícím způsobem nastaví ovladače zařízení
- připojí kořenový svazek pro čtení a provede kontrolu souborového systému
- spustí proces init

Proces INIT

- hlavní proces, který drží v provozu OS (OS se ukončuje killnutím INIT)
- rodič všech ostatních procesů
- konfiguruje se /etc/inittab
- nejdřív spustí démony, pak terminály

Embedded systémy

V Embedded systémech je spojená funkce BIOSu a zavaděče – poslední rutina Bootloaderu je zavolání programu uloženého na pevně dané adrese ve flash paměti

Proces a jádro

Proces = jedna instance vykonávaného programu

- Je umístěn v paměti
- Základní subjekt plánování (základní jednotka plánovače) – pozor: ne nejmenší! (thread)
- Má svůj **kontext**:
    - **User oblast (alokované místo pro data, která potřebuje proces)**
        - zásobník jádra procesu pro volání funkcí jádra
        - okamžitý adresář
        - kořenový adresář
        - pole pro argumenty systémového volání a pole pro návratovou hodnotu
        - spoustu dalšího
    - **Proc záznam (** = Process Control Block, položka seznamu procesů; **záznam, který potřebuje jádro pro správu procesů)**
        - identifikaci procesu (číslo procesu, PID)
        - reálné a efektivní UID/GID
        - obsah registrů procesoru, zejména
            - Program Counter – adresa následující strojové instrukce
        - Určení adresního prostoru procesu

        - Fronta nezpracovaných signálů
        - Priorita (info pro plánovač)
        - účtovací informace o procesu (kdy naposledy běžel, kolik času procesoru již spotřeboval apod.)
        - odkaz na další PCB
        - I/O informace (alokovaná I/O zařízení, seznam otevřených souborů apod.)
- **sdílení času**: čas (jednoho jádra vícejádrového) procesoru v jeden okamžik může být přidělen jenom jednomu procesu
    - přepínání kontextů procesů tak, aby to pro uživatele vypadalo, jako že procesy běží simultánně

(Další informace: http://140.120.7.20/LinuxKernel/LinuxKernel/node49.html)

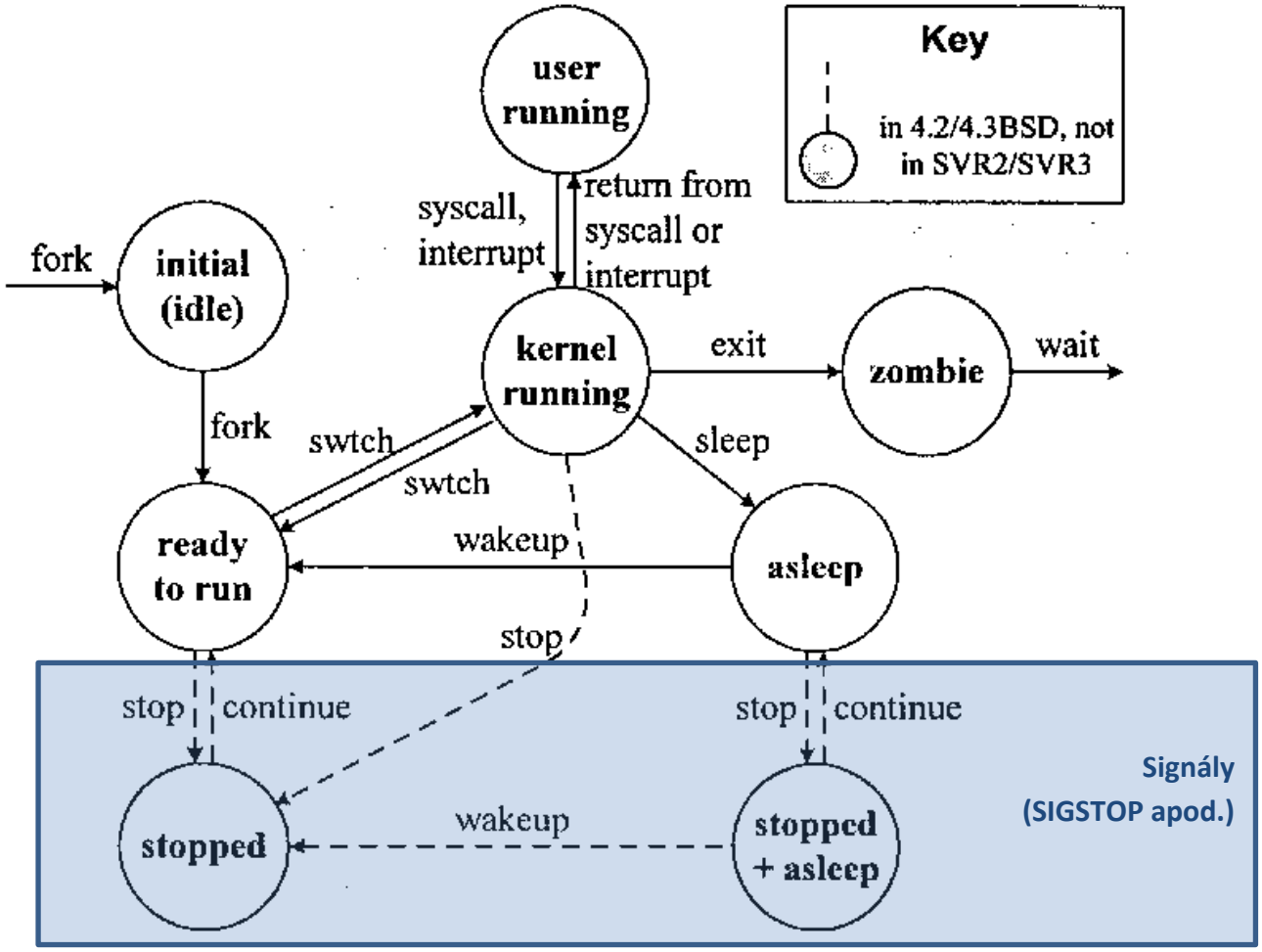
Jádro

- Speciální program zavedený do hlavní paměti při startu systému
- U pokročilých operačních systémů jádro nikdy neztrácí kontrolu nad počítačem a po celou dobu jeho běhu koordinuje činnost ostatních běžících procesů.
- Hlavní úkol jádra spočívá v
    - přidělování paměti a času procesoru (či procesorů) procesům (plánování),
    - ovládání zařízení počítače (pomocí ovladačů) a
    - abstrakci funkcí (systémová volání).
- Zajištění bezpečnosti:
    - User mode
    - Privilegovaný režim, Kernel mode

Různé druhy jader

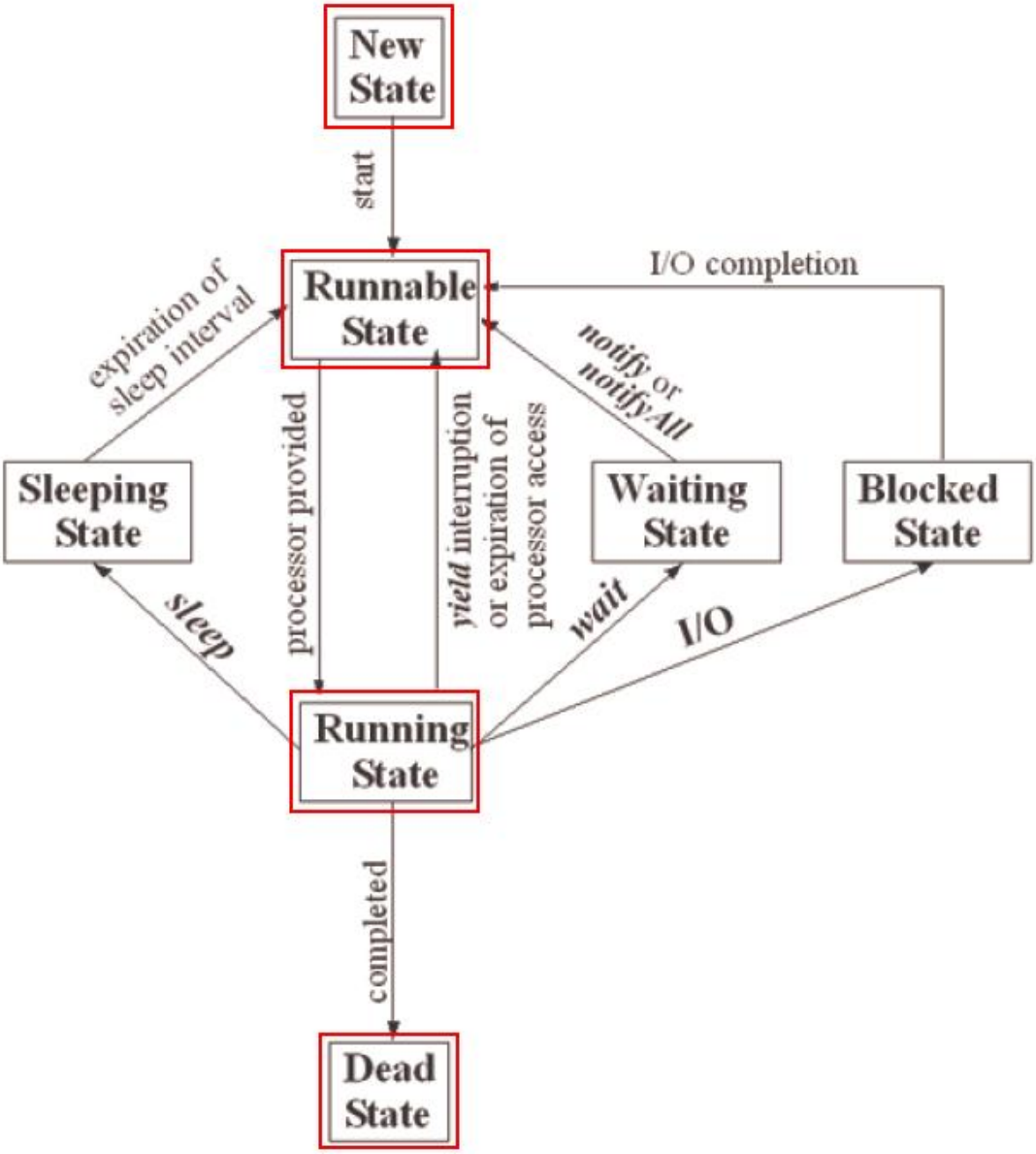
- Monolitické jádro
- Hybridní jádro
- Mikrojádro
- Nanojádro

Stavy procesu + stavy vlákna, oprávnění uživatele.



Mátoha (zombie) - proces končí voláním exit() anebo v důsledku signálu: prostředky jsou odebrány, ale zůstává záznam proc. Mátoha existuje dokud rodič nevykoná wait().

Sirotek – potomek mátohy. Když je mátoha zrušena (wait()) a sirotek běží, stane se jeho rodičem init.



Oprávnění uživatele

Uživatel identifikován svým číslem (UID) a číslem skupiny (GID).
K souborům mohu přistupovat ve smyslu

- Čtení
- Zápis
- Spouštění

A pro každý typ přístupu jsou u každého souboru zavedena oprávnění pro

- Vlastníka
- Členy skupiny (uživatel má nastavenou jednu výchozí skupinu pro nastavení práv při vytvoření souboru)
- Všechny ostatní

Příklad: výchozí práva na soubor: 644 (rw-r—r--) a na složku: 755 (rwxr-xr-x)

Reálné, efektivní UID a sticky bit

Procesy dědí oprávnění od rodičovských procesů.
Efektivní UID je programem používáno jako oprávnění pro přístup k všem požadovaným prostředkům. Používá se v případech, kdy chceme uživateli umožnit provedení akce, na které by potřeboval jiná nebo vyšší oprávnění:
SUID bit = 1:

- reálný UID (real UID) – ID uživatele, který proces spustil
- efektivní UID (effective UID) – ID uživatele, který vlastní spustitelný soubor

SUID bit = 0:

- Reálný UID = efektivní UID = ID uživatele, který proces spustil

SGID = to samé ale pro skupinu
Sticky bit: v adresáři, kde má víc uživatelů právo zápisu slouží sticky bit = 1 k tomu, aby uživatelé směli mazat jen své vlastní soubory a ne soubory ostatních (př: /tmp)

Proces, thread a fibre – implementace a využití.

Wiki: Thread je nejlehčí jednotka plánování, TXKoutný: Proces - *největší* výpočetní entita plánovače.

Je potřeba rozlišovat thread programátorský a thread z hlediska plánovače – dvě úplně odlišné věci, ale v literatuře se oboje jmenuje thread a pak jsou z toho zmatky.

Process

- Má svůj vlastní kontext a adresní prostor
- Může mít jedno nebo více vláken v jednom (svém) adresním prostoru
- Vlákna uvnitř procesu sdílejí prostředky procesu (otevřené soubory)

Thread

- Kernel thread: jednotka plánování plánovače jádra (činnost plánovače je založená na přepínání kernel threadů), jejich počet je nezávislý na počtu jader procesoru nebo počtu procesorů
- User thread: uživatelem vytvořený thread v rámci prostředků programovacího jazyka
  - Bez podpory plánovače jádra: fiber
  - S podporou plánovače jádra: lightweight proces (lehký protože nemá vlastní adr. prostor, ukazatele na soubory apod.) – fiber namapovaný na kernel thread
- Sdílená paměť a Thread-local storage (může mít a nemusí, v rámci adresního prostoru procesu)

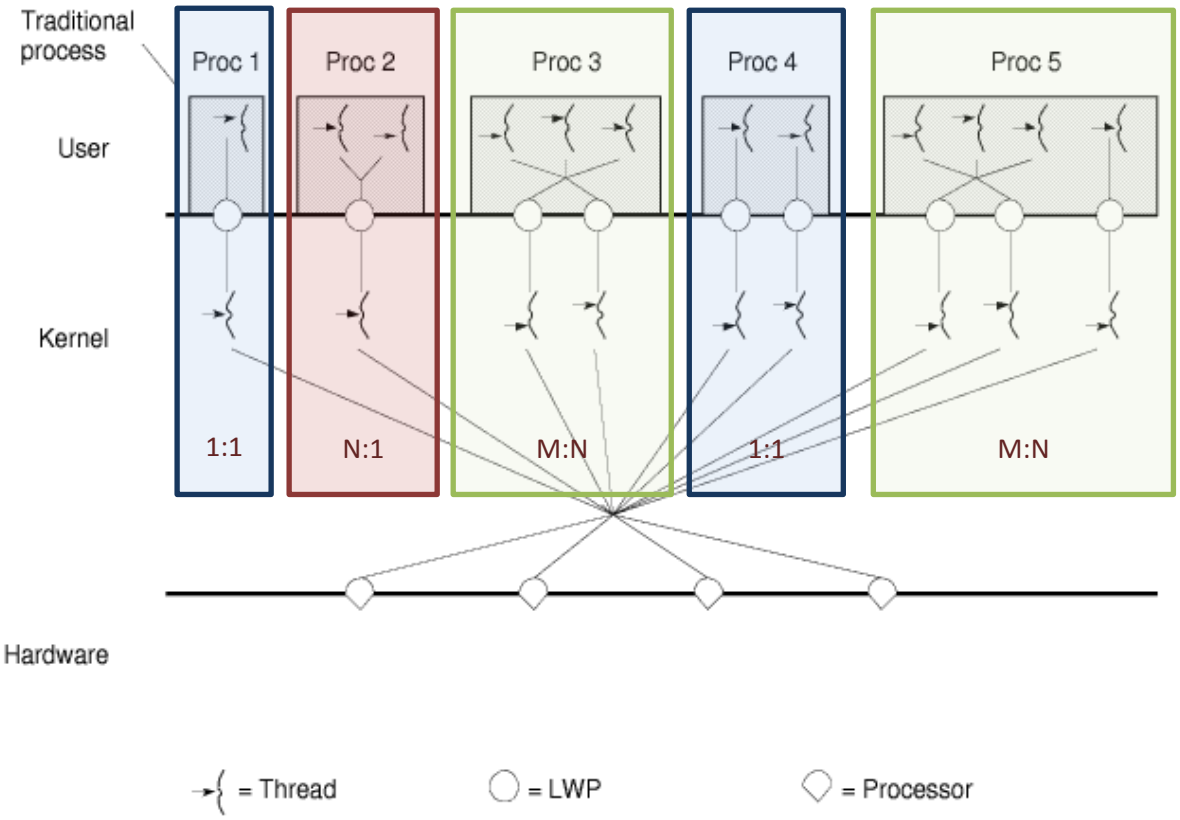
Fiber

- Běží v uživatelském prostoru a při přepnutí fiberů se nepřepíná kontext – rychlé a levné
- Spolupracují kooperativně (ne preemptivně) – musí udělat yield, jsou **implicitně synchronizované**
- Méně problematická thread-safety: nejsou potřeba spinlocky a atomické operace
- Vyžaduje menší podporu od OS, nemusí požadovat vůbec žádnou (třeba podle výhodnosti plánování –OS může a nemusí mít „lepší“ plánovač). Podporují je Unixové systémy, Microsoft, Symbian…
- Nemohou využívat víc procesorů (všechny fibery jsou v jediném kernel threadu)

- Sdílená paměť a Fiber-local storage

Vláknové modely

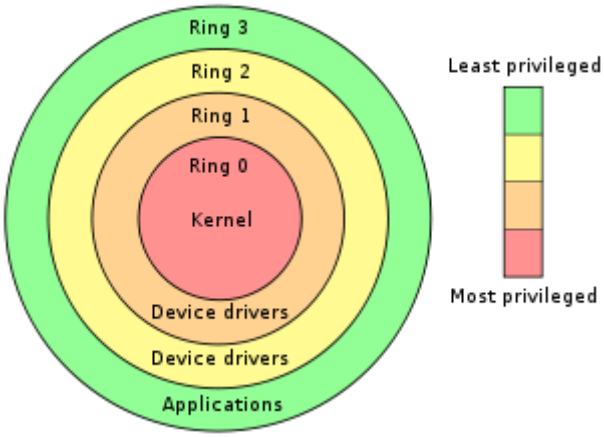
- **1:1 (kernel vlákna)** Vlákna vytvořená uživatelem odpovídají počtem 1:1 entitám, které plánuje jádro. Nejjednodušší přístup, ale je třeba uvážit preempci a thread-safety
- **N:1 (uživatelská vlákna)** všechna aplikační vlákna jsou namapována na jedno jádrové vlákno, jádro nemá tušení o tom, že aplikace běží vláknově.
- **M:N (hybridní)** komplexní na implementaci (vyžaduje změny v jádře i uživ. prostoru) ale umožňuje zvýšení efektivity výpočtu (například přiřazení více uživatelských vláken více vláknům jádra – proces může běžet na více procesorech).



Výpočet v módu jádro, systémová volání, výjimky, přerušení, signály

Procesor může běžet ve dvou nebo více režimech:

- Uživatelský
  - Omezení (manipulace s cizí pamětí apod)
  - Pro „nedůvěryhodné“ programy (uživateli se nesmí věřit!)
- Privilegovaný
  - Program může dělat cokoli a číst a psát kamkoli
  - Pouze pro důvěryhodné programy



Počet ringů a jejich přesná definice záleží na výrobci procesoru.
Ring -1: Speciální konstrukce pro virtualizaci – ve virtuálním stroji může OS běžet v Ring 0. Ring 0 je **superuživatel**, ring -1 pak **hyperuživatel** (-> odtud **hypervisor).**

OS může být navržen pro využití této vlastnosti,např. tak že pouze jádro je důvěryhodný program. Pokud uživatelský program potřebuje využít možnosti privilegovaného režimu, musí to provést prostřednictvím rozhraní jádra –> **systémové volání** (samotné činnosti v privilegovaném režimu tak zase provádí pouze jádro = cokoli spouštím v privilegovaném režimu je naprogramováno výrobcem OS a OS mi nedovoluje v p.r. spustit vlastní kód).

Např. DOS ve starších verzích běžel jen v privilegovaném režimu, ovladače zařízení tak mohly být spuštěny jako uživatelské programy.

Příklady výpočtu v módu játro: přerušení od zařízení, výjimky, sw. přerušení

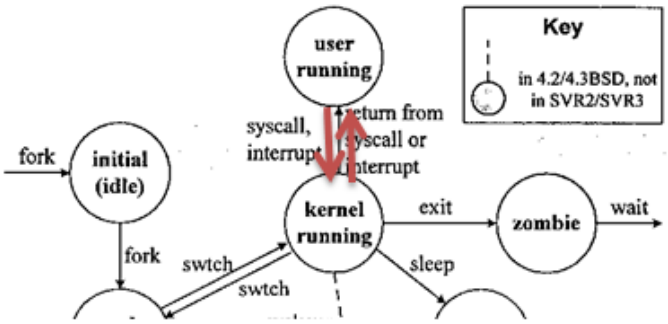
Systémová volání

Jazyk má pro systémové volání vlastní konstrukci (i proto se překládá vždy pro konkrétní OS) – obálka pro vlastní systémovou proceduru.

- Předání řízení jádru zevnitř uživatelského programu
- Řízení se předá SW přerušením (např. syscall())
- přepne se úloha (volající program je uložen a „odplánován“)
- funkce jádra vykoná požadovanou akci a uloží data
- funkce jádra vrací řízení
    - synchronní volání se vrací do volajícího programu
    - asynchronní program uspí a vrací řízení plánovači, aby spustil jiný připravený proces

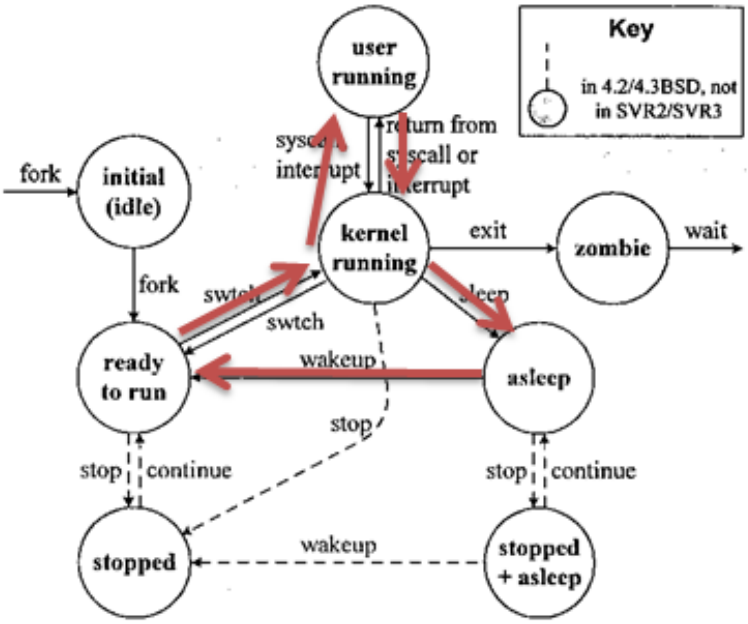
Výjimky

- Přerušení činnosti vzniklé **uvnitř** procesoru: např. overflow, neznámá instrukce apod.
- synchronní s procesem (proces čeká na vypořádání výjimky a pak pokračuje v práci, nezasahuje plánovač)
- procedury zpracování jsou podobné jako systémová volání



Přerušení

- přerušení způsobené **vně** procesoru, např. od hardwaru
- Postup: Program požádá systém o čtení z disku -> pomalá operace -> proces je zablokován a je spuštěn další naplánovaný proces -> … -> po dokončení diskové operace je ovladačem disku zasláno přerušení do procesoru, ten zjistí komu přerušení patří a majitele z blokovaných přesune do připravených. Po naplánování program pokračuje vesele dál tam, kde byl přerušen.



Signály

- umožňují oznámit procesům výskyt událostí v systému
- krátké zprávy, kde se procesům oznámí číslo signálu
- slouží tedy k uvědomění procesu, že nastala určitá událost, čímž jej přinutí vykonat funkci na zpracování signálu
- **asynchronní signál** - stisknutí CTRL-C -> generuje se přerušení (jako u každého stisknutí klávesy) -> ovladač rozpozná, že jde o kombinaci generující signál a odešle signál SIGINT procesu v popředí,
    - proces najde signál: při návratu do uživatelského módu po naplánování; při návratu z přerušení běžel-li
- **synchronní signál** - výjimka způsobí přechod do módu jádro, jádro vykoná její obsluhu a zašle se odpovídající signál běžícímu procesu, při návratu z obsluhy proces najde signál

- **nespolehlivá obsluha signálu** - funkce pro obsluhu signálů nejsou perzistentní – jednou zavoláme systémové volání signal(), jeho obsluha proběhne při přijetí prvního signálu a pak už se na žádný další signál nečeká -> případný další signál není zpracován (případně se zase musí zavolat signal()).
- **spolehlivá obsluha signálu** - perzistentní obslužné funkce signálů – při přijetí jakéhokoli signálu je zavolána obsluha pomocí sigaction().

Obsluha signálů: signály jsou obecně asynchronní a můžou přijít kdykoli (tedy handler je přerušen dalším signálem), proto by se v obsluze signálů neměly provádět „nebezpečné funkce“ – např. malloc.

Plánování, plánovací třídy, inverze priority

Víceúlohové systémy:

- systémy reálného času
- interaktivní systémy
- dávkové systémy

Ve víceúlohových systémech sdílení času je tradiční problém „vhodně“ a spravedlivě přidělovat čas jednotlivým procesům (adekvátně k typu – rt/dávkový) a zajišťovat vysokou průchodnost (tyhle tři požadavky jsou často v kontradikci). Je třeba hlídat aby nedošlo k vyhladovění a k inverzi priorit.

Problém velikosti časového kvanta: v interaktivních systémech je třeba přepínat často, aby byl vytvořen dojem okamžité odezvy, ale přepínání HW kontextu je náročné a zdržuje = čím kratší čas, tím větší režie systému.

Typy plánování:

- preemptivní – s předbíháním, proces je možno přerušit zvenčí, je potřeba zavést synchronizační primitiva
- nepreemptivní – proces se musí vzdát procesoru sám (fiber)

Plánování:

- **FIFO** – nejjednodušší, neadaptabilní, málokdy spravedlivý. Když nějaký proces nedoběhne, může dojít k vyhladovění
- **Shortest proces first / shortest remaining time –** může dojít k vyhladovění (v systému je hodně „krátkých“ úloh, vyhladovějí ty dlouhodobější)
- **Round robin** – procesy obdrží každý stejný čas a střídají se dokola, nedojde k vyhladovění protože není zavedena priorita
- **Prioritní plánování** – Každý proces má svou prioritu (procesy jádra nejvyšší, interaktivní vysokou, dávkové nízkou), procesy jsou podle priority roztříděny do tříd, ve kterých cyklují metodou Round Robin (vždy nejdřív první neprázdná třída od nejvyšší priority). Může dojít k vyhladovění procesů s nízkou prioritou (řešením je zvyšovat prioritu dlouho nenaplánovaným procesům).
    - Prakticky ve všech dnešních OS, mnoho různých metod:
        - **Fair Share** - V tomto prioritním plánování znamená vyšší numerická hodnota nižší prioritu. Každý proces k má nastavenu pevnou základní prioritu **Bk** a v každém časovém intervalu **i** používal proces **k** CPU po dobu **CPUk(i)**. Proces má v časovém intervalu **i** prioritu **Pk(i)**a na začátku každého časového intervalu je nastavena hodnota **CPUk(i)** na polovinu předchozí hodnoty. Hodnota **Pk(i)** je pak dána jako součet základní priority a nové hodnoty **CPUk(i)**. Plánovač vybere proces s nejnižší hodnotou **Pk(i)**.

- **Epochy** - čas procesoru je rozdělen do epoch
  - každý proces má specifikováno časové kvantum v rámci epochy
  - v jedné epoše proces může využívat své časové kvantum po částech
  - epocha končí, když všechny běhu schopné procesy vyčerpaly svá časová kvanta
  - Délka časového kvanta v epoše závisí na prioritě procesu.

RT plánování

- Soft RT
- Hard RT
- Deadline apod…

Viz níž.

Inverze priority

Inverzí priorit rozumíme situaci, která nastane, pokud vlákno s vyšší prioritou požaduje přístup k systémovým zdrojům, které v danou chvíli právě exkluzivně drží vlákno s nižší prioritou. V tomto případě dojde k preempci do vlákna s vyšší prioritou, které však nemůže běžet díky zablokovanému systémovému zdroji. To je velice nepříjemná situace, zejména v RTOS. Jediným řešením je umožnit vláknu, které systémový zdroj drží co nejrychleji doběhnout a umožnit tak i jiným vláknům pokračovat v jejich činnosti. **K vyřešení této situace se používá systém inverze priorit**, který umožní vláknu s nižší prioritou zdědit prioritu kritického vlákna, rychle vykonat potřebné operace až do chvíle uvolnění požadovaného systémového zdroje a dále pak nechat pokračovat v práci kritické vlákno.

Uvíznutí, vyhladovění, problém korupce dat – charakteristika a způsoby řešení.

Uvíznutí – deadlock

4 nutné podmínky (vlastnosti systému) pro vznik deadlocku:

1. podmínka **vzájemného vyloučení**: zdroj (prostředek (angl. resource)) může být vlastněn pouze jedním procesem
2. podmínka **drž a čekej**: proces který již nějaký prostředek vlastní smí požadovat další prostředek
3. podmínka **neodebratelnosti**: pouze proces držící zdroj ho může uvolnit
4. podmínka **kruhového (cyklického) čekání**: dva či více procesů tvoří uzavřený řetěz, kdy každý čeká na zdroj držený jiným procesem

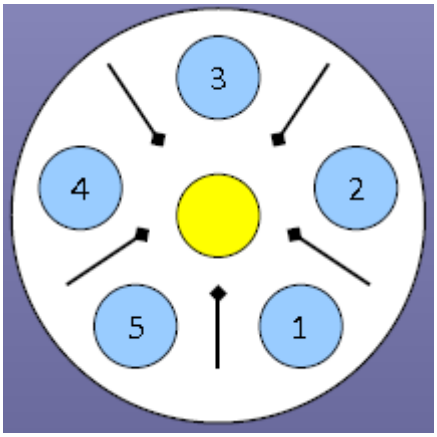
Řešení

- **Ignorace** - pštrosí metoda = předstíráme, že se nic neděje
- **Detekce a zotavení** - nesnažíme se zabránit, ale detekovat (např pomocí precedenčních grafů), pak provedeme rollback nebo zabití jednoho z procesů
- **Dynamické zabránění** - systém rozhodne, zda je přiřazení zdrojů bezpečné, pokud ano, zdroj přiřadí, jinak pozastaví žádající proces; Bankéřův algoritmus (v *praxi nepoužitelný*)
- **Prevence** - uděláme vše proto, aby k deadlocku nedošlo = **porušíme aspoň 1 ze 4 podmínek**
  - *vzájemné vyloučení* - před zdroj dáme frontu, neboli spool (např. u tiskárny)
  - *drž a čekej* - proces musí zabrat všechny zdroje najednou (použito např. u databází)
  - *neodebratelnost* - to je problém, odebrání způsobí chaos

- *kruhové čekání* - přidělování zdrojů ve předem určeném pořadí

Uvíznutí – livelock

- Procesy stále mění svůj stav v závislosti na druhém procesu, ale žádný nepostupuje vpřed ve svém výpočtu.
- Večeřící filosofové



Řešení

- Když nemůžu vzít druhou vidličku, položím tu první (stejně se můžou zas sejít)
- Strážce jídelny - semafor (další prvek navíc)
- 1 rebel (obtížně řešitelné)
- **Ohodnocení vidliček**

Vyhladovění

Proces se nedočká zdrojů tak dlouhou dobu, že už pak ani není třeba aby doběhl. Např. špatně implementované prioritní plánování.

Problém korupce dat

korupce dat může nastat, pokud správně neošetříme kritickou sekci a dva procesy/vlákna paralelně zapíšou do sdílených dat

Řešení: vzájemné vyloučení z kritické sekce

Základní a strukturované formy interakce procesů a vláken

Základní formy interakce vláken

Synchronizace nad kritickou sekcí (zabránění vstupu dvou vláken do kritické sekce).

Systémy se sdílenou pamětí:

- **Semafor –** fronta, příkazy P() a V(), semafor s jednoprvkovou frontou = **mutex**
- **Bariéra –** synchronizace více vláken v jeden okamžik. Typicky se používá v paralelních iteračních výpočtech. Vlákno přijde k bariéře, a uspí se dokud nedorazí určený počet vláken (pokud bylo poslední, probudí se všechna ostatní).
- **Spinlock** - aktivní čekání dokud není volný zámek, operace Test and set lock, spolu s mutexem se používá pro výhradní přístup k paměti

Systémy s distribuovanou pamětí:

- Zasílání zpráv – je potřeba vyrovnávací paměť pro zprávy
  - Synchronní – blokující send i receive
  - Asynchronní – blokující receive

Základní synchronizační prostředky kladou vysoké nároky na implementaci, může dojít k zablokování programu.

Strukturované formy interakce vláken

Oproti primitivním formám komunikace poskytují větší bezpečnost programování, jazyky pro ně často nabízejí speciální konstrukce.

Monitor (Synchronized)

- Metoda nebo blok (blok se nedoporučuje) který je vždy vykonán atomicky, takže jej nelze zavolat víckrát najednou ani zvenku přerušit
- Vlákna se mezi sebou vůbec neznají, každé pouze volá monitor
- Veškeré interakce vláken probíhají nepřímo prostřednictvím monitoru
- Monitor obsahuje frontu, zámek a zámek (pro obsluhu synchronizace)
- Volání monitoru je blokující (samo sebou)

Rendezvous

- Prostředek pro synchronizaci úkolů (tasks)
- V podstatě vnitroprogramový klient-server:
  - klient zavolá server
  - server si převezme parametry
  - server provede výpočet, klient spí
  - server předá výsledky
- Klient připraví kus výpočtu a pošle ho serveru ke zpracování
- Když je server busy, tak klienti čekají ve frontě

Meziprocesová komunikace

- Signály (asynchronní)
- Roury (zápis na jednom konci, čtení na druhém), uložená pojmenovaná roura
- Fronty zpráv
- Sdílená paměť (potřeba synchronizovat přístup)

Sdílená paměť

- Sdílená paměť – situace, kdy dva běžící procesy / vlákna / fibry můžou číst a zapisovat do stejné paměti
- Problém konkurenčního (současného) přístupu do sdílené paměti

Např. v programu C existuje přímo konstrukce pro alokaci sdílené paměti **shmget**() – program dostane paměť podobně jako při volání malloc(), ale může přidělenou paměť sdílet předáním identifikátoru.

Ostatní procesy můžou sdílenou paměť připojit pomocí funkce **shmat**() a následně zas odpojit pomocí **shmdt**.

Přístup do této paměti je stejně rychlý jako normální přístup do paměti, proto představuje velmi rychlou a efektivní možnost meziprocesové komunikace.

Sdílenou paměť je třeba chránit před možnou korupcí (mutex, semafor)

Víceprocesové systémy se sdílenou pamětí: viz předchozí otázku.

Synchronizace v jádře, symetrický multiprocesing

Kód jádrových funkcí se vykonává:

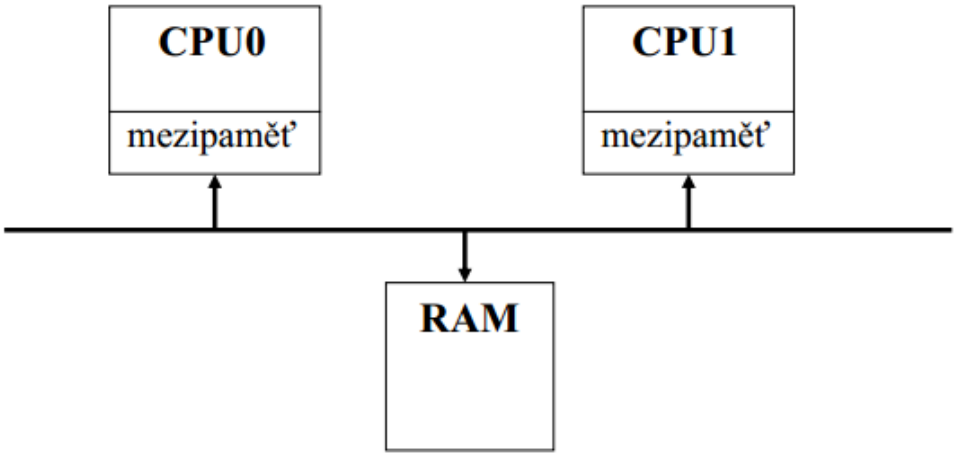
- Při systémovém volání
- Při obsluze výjimek a přerušení

Jádrové funkce sdílejí datové struktury jádra – vzniká problém synchronizace jádrových výpočtů.

Pro zabránění souběhu (race condition) v jádře se používají tyto metody:

- **Nepreemptivnost procesů jádra**: jádrový proces nemůže být přerušen a nahrazen procesem s vyšší prioritou (pokud samotný proces neudělá yield). I tak ale může být přerušen výjimkou nebo přerušením, a obsluha přerušení může být zase přerušena přerušením.
- **Atomické operace**: inc, dec…
- **Zákaz a maskování přerušení**: ne všechny operace lze vykonat atomickou instrukcí, data jádra můžeme zabezpečit tak, že při práci s nimi zakážeme přerušení. Velmi nebezpečné a nefunguje na víceprocesorových systémech.
  - Pozor na vnoření obsluh přerušení.
  - Kritické sekce se zákazem přerušení musejí být krátké.
  - Nesmí se čekat na oznámení události přerušením (to dá rozum).
- **Zamykání:** spinlock (atomická instrukce tsl, neefektivní na 1 procesoru, nemá kdo splnit podmínku), semafory

Symetrický multiprocesing



- Procesory a sdílená hlavní paměť jsou připojeny ke společné sběrnici, tu je nutno zamykat pro výhradní přístup jednoho procesoru (aby nemohly 2 procesy zapisovat na stejné místo v paměti naráz).

- Nutná synchronizace mezipaměti (cache) procesorů: když procesor modifikuje svou mezipamět, musí kontrolovat jestli stejná data nejsou v mezipaměti jiného procesoru a když ano, musí je aktualizovat nebo zneplatnit.
- Úloha může být zpracovávána postupně různými procesory, je umístěna ve sdílené hlavní paměti
- Na více procesorech naráz mohou být současně provedena systémová volání

Možnosti řešení souběhu:
- Semafory
- Kruhové čekání (spinlock)

Kromě SMP existují další technologie pracující na podobném principu:

- asymetrické multiprocesory ASMP - navíc vlastní lokální paměti a I/O připojení u procesorů, které tak mohou mít různé instrukční sady
- multiprocesor s distribuovanou pamětí - procesory mají jen lokální paměti, není sdílená paměť, komunikace zasíláním zpráv, topologie 2D mřížky nebo N-rozměrné krychle, výhoda odstranění společné sběrnice jako úzkého hrdla

Atomické operace, synchronizační objekty a funkce

Atomické operace

- Problém preemptivních víceúlohových systémů
- U některých operací musí být zaručeno, že proběhnou celé, že jejich provádění nebude v polovině přerušeno např. přepnutím na jinou úlohu. Takové operace se označují jako atomické.
- Každá atomická operace musí proběhnout buď celá, nebo vůbec.
- Příklady:
    - Instrukce, které přistupují k paměti pouze jednou
    - Čti-modifikuj-zapiš v jedné instrukci: inc, dec - na víceprocesorových systémech se musí zamknout sběrnice
    - tsl - instrukce, která paměť zároveň čte a modifikuje
    - v C mnoho funkcí (s předponou atomic, např. atomic_sub_and_test)

Synchronizační objekty a funkce

- Semafor (mutex)
- Bariéra
- Monitor
- Rendezvous

Všechny popsány výš, víš?

Systém reálného času

Víceúlohový systém s důrazem na včasné splnění úloh.

Dokončení výpočtu ve stanoveném termínu je kritické, termín musí být dodržen bez ohledu na zátěž. RT systémy nejsou vysoko-výkonnostní → jde o to dopočítat včas, ne vypočítat co nejvíc.

Mezi hlavní faktory RTOS patří:

- minimální latence při reakci na událost
- minimální latence při přepínání vláken
- někdy nutnost malých rozměrů, viz Embedded systémy
- minimalizace časových okamžiků, kdy je vypnuto přerušení
- preemptivní plánování založené na prioritách

Vyvinutý buď přímo jako RTOS a nebo upravený běžný OS (upravené jádro, plánovač apod.)

Determinismus

- Operace jsou prováděny ve fixovaných, předem určených časech nebo časových intervalech.
- Reakce na přerušení musí proběhnout tak, aby systém byl schopen obsluhy všech požadavků v požadovaném čase.

Dělení:

- **Hard real time OS**

Tzv. tvrdý real time OS. Zde je požadavek na stanovení času reakce absolutní. Na RTOS klademe většinou následující požadavky

  - preemptivní plánovač
  - velký počet nastavitelných priorit vláken
  - přesné hodiny reálného času

často je systém hard-real time nasazen tam, kde by případné nedodržení časových limitů mělo katastrofální následky

- **Soft real time OS**

Na rozdíl od hard real time OS se u soft real time OS dovolují drobné odchylky v reakcích.

Plánovač RTOS

RTOS mnohdy využívají plánovací algoritmy, které se liší od plánovačů běžných OS. Nejčastější z nich jsou:

- **Rate monotonic scheduling (RMS)**
  - Priorita je úlohám přidělována podle jejich periody: čím kratší perioda úlohy, tím větší priorita.
  - Úlohy jsou periodicky spouštěny za sebou podle přidělené priority.
  - Priorita je pro danou úlohu statická, nemění se – **static priority**.
  - **Liu & Layland** – pokud je zatížení procesoru n úlohami pod *ln(2)* (což je asi *70%*), pak lze všechny naplánovat tímto static-priority algoritmem
- **Earliest deadline First (EDF)**
  - Každý proces oznamuje při svém příchodu do fronty dobu platnosti (nejčastěji mezní termín splnění - deadline).
  - Plánovač udržuje informace o všech spuštěných úlohách ve frontě seřazené podle deadline.
  - Plánovač spouští úlohu s nejbližším časem deadline a kdykoliv se spustí úloha s bližším časem deadline, scheduler odstaví právě obsluhovanou úlohu a spouští novou - v tomto případě příchozí.
- **Round-robin scheduling**

Virtuální souborový systém

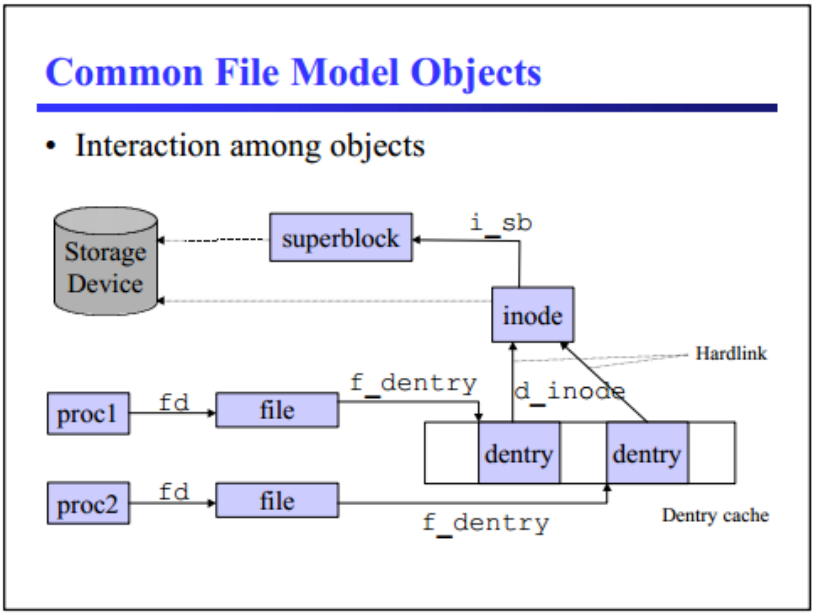
- Abstrakční vrstva mezi OS a konkrétním souborovým systémem.
- Na jedné straně je univerzální rozhraní pro operační systém a na druhé straně konkrétní ovladače pro všechny podporované souborové systémy
- Lze jednoduše přidávat podporu pro různé FS jenom novým ovladačem pro VFS bez potřeby upravovat jádro.
- Umožňuje zajistit dopřednou kompatibilitu OS s novými FS

Příklad: VFS v Linuxu

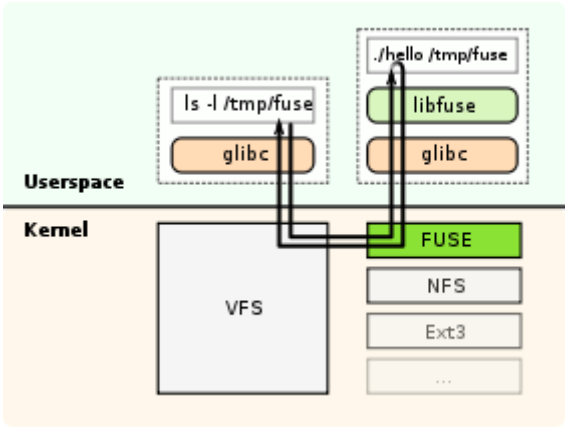
- Objektově orientovaný, každý typ objektu má seznam definovaných operací
  - Pro C existuje tabulka pointerů na funkce
- Podpora různých systémů – fyzické, síťové, speciální
- **Common file model** – zaměřen hlavně na unixové systémy, ostatní jsou mapovány (např. FAT nemá inode)
- Interakce mezi VFS a konkrétním systémem pomocí čtyř hlavních struktur:
  - **super_block:** globální informace o FS
  - **inode:** konkrétní soubor
  - **dentry:** adresář
  - **file:** otevřený soubor (procesem)

  a souvisejících struktur

  - **super_operations**
  - **inode_operations**
  - **file_operations**



FUSE



- Filesystem in Userspace
- Jakýkoli zdroj dat přístupný FUSE může být uživateli zpřístupněn jako souborový systém (GmailFS, SSHFS)
- Pomalý ale velmi flexibilní

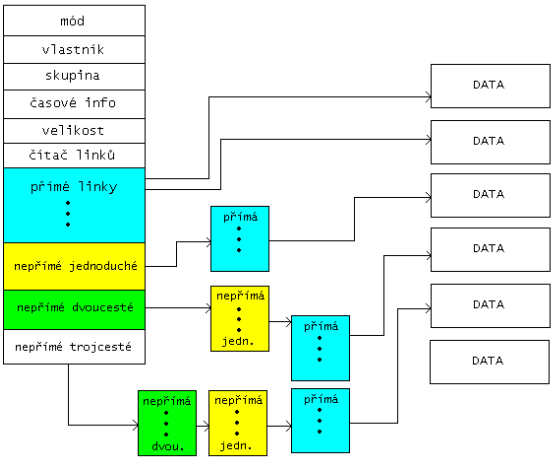
EXT

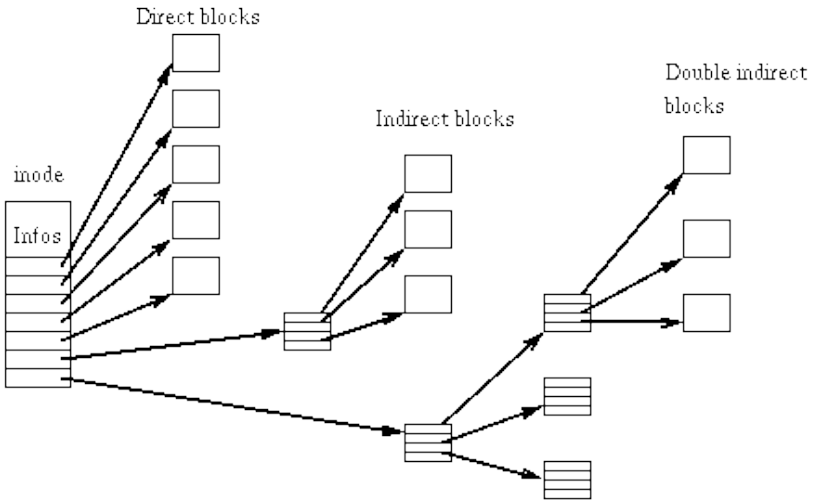
- Vychází z UFS, navržen a vytvořen pro Linux
- EXT, EXT2 až EXT4
- Různé typy souborů: obyčený soubor, blokové a znakové zařízení
- Pevné odkazy, symbolické odkazy
- **Ext3**
  - žurnálovací (3 způsoby žurnálování)
  - aktivně předchází fragemntaci (nelze jej defragmentovat když je připojený)
  - bezpečnější mazání (složitější obnova smazaných souborů)
  - zpětně kompatibilní s ext2
- **Ext4**
  - Zpětně kompatibilní s ext3 (fork a přidání funkčnosti) krom extentů
  - Extent: nahrazení tradičního blokového rozdělení, zmenšuje fragmentaci a zlepšuje výkon při práci s velkými soubory (alokační jednotka o velikosti až 128 MB) – při použití nelze mountnout jako ext3
  - Delayed allocation – alokování až při zápisu (zmenšuje fragmentaci)
  - Rychlejší kontrola (přeskakování nealokovaného místa)
  - Nanosekundový timestamp

Inode

- i-uzel obsahuje metadata pro každý libovolně velký soubor i adresář, například čas poslední změny, přístupová práva, seznam datových bloků a podobně.
- Univerzální struktura pro metadata
  - Adresář je specifický případ souboru
- Obsahuje
  - typ souboru
  - počet odkazů na tento objekt

- owner, group
- velikost objektu
- časové údaje
- 12 přímých odkazů, 1 single indirect, 1 double indirect, 1 triple indirect (strom)





Správa V/V zařízení

V/V (input/output) zařízení je hw zařízení které zprostředkuje kontakt počítače s okolím,

- Klávesnice/myš, čtečka kódů
- Obrazovka, tiskárna

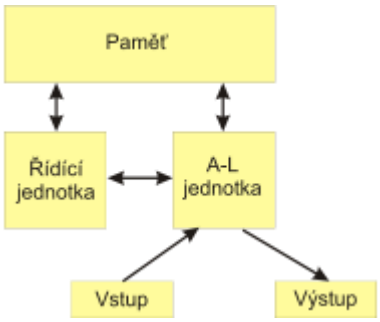
případně pseudo zařízení

- /dev/null
- /dev/random - generátor náhodných čísel

Procesor komunikuje s V/V zařízeními pomocí registrů (můžou sloužit jako vyrovnávací paměť)

- **Izolované (port-mapped)**: Přístupné pomocí speciálních instrukcí
- **Vnitřní (IO mapped)**: namapovaná paměť,
  - adresované jako paměť,

- přístupné pomocí běžných paměťových instrukcí
- např. DMA, velmi rychlé protože data nemusí do paměti přenášet procesor
- Porušuje Von Neumannovu architekturu:



V/V zařízení si vyžádá obsluhu procesorem pomocí přerušení – 1 bitový kanál, který pouze upozorňuje procesor, že je třeba věnovat se V/V

Využití V/V zařízení v programu vyžaduje systémová volání (procesor musí běžet v privilegovaném režimu, do kterého se uživatelský program nesmí přepnout).

- Speciální konstrukce jazyka – např. proudy v C (stdio)

Soubor typu zařízení

Unixové OS mapují V/V zařízení do souborového systému (protože se snaží do soub. Systému mapovat úplně všechno)

Soubor se vytváří voláním mknod s parametry

- Jméno
- Druh – blokové, znakové, roura
- Hlavní číslo – identifikuje skupinu (USB)
- Vedlejší číslo – konkrétní zařízení (třetí port zprava)

- **Bloková zařízení:** je potřeba zapisovat a číst vždy celý blok určené velikosti (video, zvuková karta)
  - Obsluhují se V/V operacemi s mezipamětí
- **Znaková zařízení:** zapsán a čten je vždy jeden znak (terminál, COM a LPT porty)
  - Protože jde o jeden znak, nemají mezipaměť
  - Mechanizmus proudu – duplexní zpracování

Síťová zařízení nemají svůj soubor.