

Operační systém je kolekce programů, které vytváří spojnici mezi HW výpočetního systému (který může být virtualizován), uživateli a jejich uživatelskými aplikačními programy.

Cíle: maximální využití zdrojů x jednoduchost použití

Role: správa prostředků + tvůrce prostředí pro uživatele → standardní rozhraní + abstrakce (proces, virt. p.)

Zahnuje: jádro (+ systémové aplik. programy - utility knihovny + rozhraní)

Jádro: reaktivní program v privilegov. režimu (přímá ved. HW)

správa prostředků + základní prostředí pro vyšší vrstvy - bezprostřední interakce s HW

Monolity: řada služeb, těsně provázání (efektivní x neflexibilní)

Monolit s moduly: umožňuje přidávat a odebírat subsystemy za běhu

Mikrojádra: jednoduchá, malý rozsah

většina služeb v **serverech** mimo privileg. režim

flexibilní, bezpečnější x vyšší **režie**

Hybridy: mikrojádra, kde kód jenom běží těsněji u jádra, v jeho režimu

Preempt. přerušení právě vykonávaného procesu, aniž by byl vyžadován jeho spolupráce (na základě příchodu přerušení při nějaké události)

Klasifikace OS: univerzální x specializ., jednov. x vícev., jednashitové x vícev. -ve/preempt, realtime

Snahy o vývoji: pokročilé architektury (kombinace dvou jader...), snižování režie!

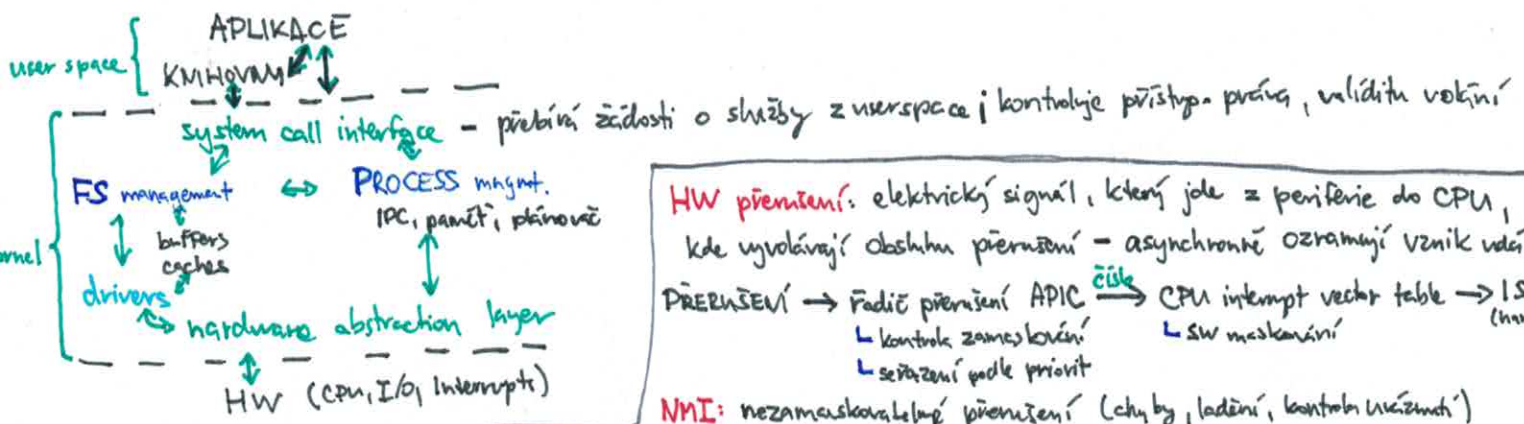
bezpečnost, spolehlivost - formální verifikace (spolehlivost: respektování sém. od sebe)

podpora více jader (efektivní) / procesorů

distribuované zpracování

POSIX: standard definující jednotné rozhraní OS pro programátory (API) i uživatele (standardní prog., utility), striktní podmnínka **SUS**.

UNIX: dvě zkl. abstrakce: **procesy** a **soubory**, mezi nimi IPC a I/O
komunikace s jádrem pomocí **syscalls** na služby jádra



HW přerušení: elektrický signál, který jde z periferie do CPU, kde vyvolávají obsluhu přerušení - asynchronně oznamují vznik události

PŘERUŠENÍ → řadič přerušení **APIC** → CPU interrupt vector table → **ISR** (handler)
- kontrola zameškování
- seřazení podle priorit
- SW maskování

NMI: nezamaskovatelné přerušení (chyby, ladění, kontrola údržby)

IPI: interprocessor - správa cache mezi více CPU

SMT: system management → firmware (vybití baterie)

- mohou vznikat i v CPU: trap, fault, abort

- obsluha musí být synchronizovaná

(aby nebyl nekonzistentní stav) → **zákazy přerušení + rozdělení do dvou úrovní** (speciální procesy - softIRQ)

Ovladače: registrace k obsluze IRQ → komunikace s řadičem služeb nebo sdělování
sdělování IRQ - jádro vede zřetelný seznam zejména o obsluhu IRQ /proc/interrupts

evný disk: plotny (platters) + stopy (tracks) = jedna kružnice → cylinder (stopy na plotnách pod sebou)

stopa dělení na sektory (nejmenší RW jednotka), obvykle 4 KiB

adresace: dřív **Cylinder Head Sector**, dnes **Linear Block Address** (blok = sektor) - elektronice si to spočítá

Přístupová doba: doba vystavení (pohyb) hlav + rotační zpoždění; typicky jednotky ms

sustained (až nízké stavy MB/s) × maximum transfer rate

SSD: organizační stránky (4 KiB) → bloky (128 str. → 512 KiB)

prázdné stránky lze zapsat jednotlivě, ale přepisování po **blocích** (problém menší u seq zápisu)

TRIM: OS (FS) řekne SSD, které stránky už nejsou používány → zresetování → možnost zápisu

ideálně volné bloky → SSD si přesouvá data, jak se mu to hodí

(k ničemu u obkazi FS, databází...)

minimalizace přepisů → problém s mazáním (bezpečným)

RAID 0: striping **RAID 1:** mirroring **RAID 5:** block-level striping + distribuce parity → rovnoměrné zatížení

RAID 6: ukládá paritu 2x → mohou selhat dva disky může selhat jeden disk

ALOKAČNÍ BLOK: skupina pevného počtu sektorů (typicky 2^n), které za sebou následují logicky v souboru a současně za sebou vždy následují i fyzicky na disku. Je to nejmenší jednotka, se kterou OS běžně pracuje (a s jakou umožňuje pracovat uživateli).

EXTERNÍ FRAGMENTACE rozumíme jev, kdy (v důsledku delší činnosti FS) na disku vzniká posloupnost použitých a nepoužitých oblastí (přičemž použité oblasti navíc mohou být použity různými soubory).

→ příliš malé nevyužité oblasti, které se nedají rozumně použít (při spojitelném přidělování po různých velikostech)

→ nespojitě ukládání → větší množství metadat → více prostoru, pomalejší (nebo není možnost ukládat ani metadata)

- minimalizace: rozkládání ukládaných souborů po disku + předalokace + odložení alokace (zápisy se střídají)

INTERNÍ FRAGMENTACE: soubory nejsou velké jako alok. bloky → v posledním bloku zůstává volné místo

- některé FS umí tail-packing - toleruje se

Přístup na disk: příkazy: IO porty nebo paměť. mapované operace i přík. definované rozhraním (ATA)

přenos dat: DMA ukončení: interrupt

Plánování: plánovač disk. operací - subsystém jádra - snaží minimalizovat režii (mezi FS a driverem)

- fronty, přeuspořádání **Vytahání/SCAN alg.:** pohybuje hlavami tam a zpět, vyřizuje požadavky v pořadí

Circular SCAN: vyřizuje jen při pohybu v jednom směru odpovídajícím pohybu hlav

LOOK: podobně, ale pauze v mezích daných akt. frontou

- sdružování operací, vyřezávání požadavků uživatelů, odkládání požadavků, priority operací...

- více vstupních i výstupních front

IBR: nulový sektor, 1-4 primární disk. obl., jedna z nich mohla být odkaz na zřet. seznam logických obl. (EBR záznam na začátku každé z nich)

IBPT: tabulka až 128 odkazů na logické disk. obl. i kontrolní součet, duplikace tabulky

LVM: logical volume manager, pokročilý správc. log. oblastí - mezi VFS a plánovačem nebo přímo ve FS

Žurnálování: technika umožňující rychlý a bezpečný návrat FS do konzistentního stavu založená na udržování žurnálu

= záznam o modifikacích metadat (občas i dat) i pokynů operací **atomicke** i cyklický přepisovaný buffer

- musí spolupracovat s plánovačem - velmi závislé na správném pořadí operací se žurnálem

- obvykle se režíruje data (režie) - kompromis: první zapsat data, pak žurnál metadat, pak ostatní data

REDO: sekvence do žurnálu (zabezpečení) → provedení na disku → úspěch - uvolnění

UNDO: dílčí op. - žurnál - dílčí op. - žurnál... pokud je v žurnálu nedokončená op., všechno se vrátí

COW: hierarchický popis celého obsahu disků - vyhl. strom popisující uložení dat a metadat
 vytvoření kopie listových bloků, postupně vytvoří kopii celého stromu, až při změně (publikaci) do kořene
 jsou nové data dostupné - udržuje se několik verzí kořenového záznamu

- snímky (uložení staršího kořene) + klony (kopie kořene)

Soft updates (uzpůsobuje pořadí zápisu dat/metadat, může vznikat garbage) **Log-structured FS** (jeden velký log)

UNIX fs: boot block + super block (informace o FS) + **tabulka i-uzlů** + data

modifikace: ~~uložení~~ rozdělení na skupiny bloků, každá má své i-uzly ... UID, GID, práva

i-uzel: stav (blok.x.volny) + typ soub. + délka souboru (v B) + [mlatc]time + AC + počet jmen + odkazy na datové bloky
 jména souborů jsou uložena v adresáři

- až 10 přímých odkazů na blok. bloky + nepř. odkaz 1. úrovně + 2./3. úrovně (novější 12 přímých)
- velikost souboru omezena: max počet odkazů + použitelná dat. strukt. FS + VFS + jádro + architekt. systém

proč: minimalizace režie - seek, změna velikosti i snadnost vyhledání bloků, přidání/ubrání bloků a alokace/dealokace
 → min. ext. frag.

fs: optimalizován pro malé soubory - občas data přímo v i-uzlu (mychle symlinky...)

kontinuální ukládání (blbě se zvětšuje) **zřetězované seznamy blok. b.** (blbě se posouvá, rozproštěná metadata - náhled na

FAT: na začátku je tabulka, která pro každý blok říká, jaký blok je dál; může se zduplikovat i simple AF

Extenty: posloupnost alokačních bloků (proměnného počtu), které jdou logicky i fyzicky za sebou
 → zrychlení práce s velkými soubory, zmenšení objemu metadata
 - snadno kombinovatelný s B+ stromy - ne s klasickými UNIX fs!

ext4: používá **strom extenti** (podobný B+, bez vyvažování, max 5 úrovní) - v i-uzlech informace o extentech - pokud se
 soubor vkráčí do 4 extenti, všechny uloženo v i-uzlu

NFTS: používá **Master File Table** - alespoň 1 řádek per soubor / obsah přímo v MFT / v extentech odkazovaných přímo z MFT
 (startovní logický blok + délka bloků + kde začíná na disku) / v ext. odkazovaných z pomocných řádků MFT

volný prostor: bitové mapy (nebo seznamy, zřetězení v tab. bloků - FAT, B+ strom), mohou být organizovány po extentech

deduplikace: odhalování opakovaně ukládaných stejných dat, při zápisu nebo na přání, nízké úrovně, zabývá se hashováním
 může ušetřit prostor na disku/paměti, ale při menším objemu shodných dat zbytečná režie

adresář: soubor dvojic (jméno, číslo), vždy . a .., impl. jako seznam/B+ strom/hashtables...

sticky bit: uživatel v takto označeném adresáři může mít/přijmenovat jen soubory, které vlastní (++, /tmp)

cache: cílem použití vyrovnávacích pamětí je minimalizace počtu pomalých operací s periferiemi

čtení: mám blok v cache? (zahashuju, hledám) → ne: nakopíruje se blok cache (vezme se předalok.) → načten se ten data → nakopíruje
 do adr. prost. už.
 → ano: vezmu blok → nakopíruje se do adr. pr. už.

zápis: dirty bit

Otevření souboru: 1) vyhodnocení cesty, získání čísla i-uzlu (používá se cache celých cest: d-entry cache)
 2) alokuje se **V-uzel:** rozšířená paměťová kopie i-uzlu s počítadlem použití (tabulka V-uzlů: vyhledávací
 nebo se najde u alokovaného V-uzlu pro danou položku, zůstane čítač

3) systémová tab. **otevřených souborů:** alokuje se nová pol. - odkaz na V-uzel, režim, pozice, čítač užít
 4) položka v **poli deskriptoru souborů** - v záznamu o procesu v jádře - odkaz na položku
 L nebo v uživatelské oblasti procesu
 5) index v ↗

- kontrola přístupu. práv - nízké další modifikátory (např. synchronní zápis)

čtení: kontrola fd, načtení dat do cache / čtení z ní **zápis:** podobně (od dirty bit), kontroluje a alokuje prostor ve FS

Zavření: kontrola fd, uvolnění fd, snížení počítadla v tab. otevř. soub. i příp. snížení počítadla ve V-uzlu i příp. uvolnění V-uzlu,
 naplnění zápisu bloků s i-uzlem

mazení: vyhodnocení cesty, odstranění jména (hard link) z adresáře, zmenšení počítadla jmen v i-uzlu i příp. uvolnění i-uzlu
 (unlink) z disku se maže, když soubor nemá žádné jméno a počítadlo otevření = 0

blok./znak. speciální soubory: b/c, v jádře **tab. blok. zř.** a **tab. znak. zř.**, v i-uzlu typ, major num. (index v tabulce) a minor num.

konky: nepojmenované - nejsou ve FS, volání pipe → dva fd, pouze příbuzné procesy, vytváří se v kolonách (dají x přechyt jen socke
 pojmenované - mknod/mkfifo, existují ve FS
 typický kruhový buffer, procesy jsou "synchronizovány" (read blokuje, zápis do pipe blokuje)

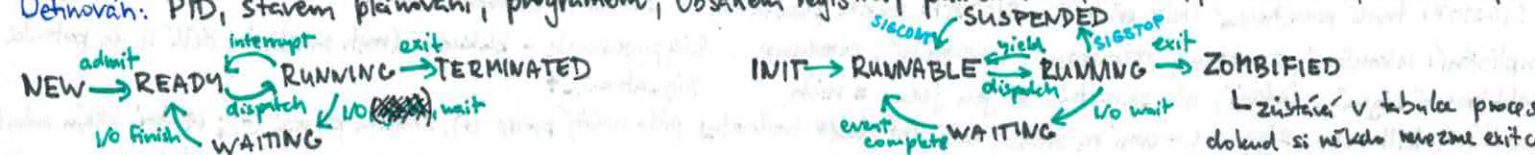
sockety: síťová i lokální komunikace, blokující i nablokující, lze na ně pasivně čekat na minizím i svázaný se sítí nebo souborem
 mohou vyhrážet aplikace, které můžou, ale nemusí běžet distribuovaně

NFS: kaskadování; autentizace smutná (UID, GID) → potřeba důvěry; v3: bezstavový, v4: stavový, client-side cache, zamýšlený
 (ale jsou i jiné možnosti) **Spooling:** umění přeměnit data z jednoho místa na druhé, které nemají nikde uložené

správa procesů: přepínání kontextu (dispatcher) na základě rozvržení plánovače (scheduler)
+ správa paměti + IPC kernel space (většinou registry) L kernel/user space

Proces je aktivní entita, abstrakce nějaké aktivity v systému (řízení programem)

Definován: PID, stavem plánování, programem, obsahem registrů, pamětí (zís., holda...), vztahy na OS (soub., seín)



Process Control Block v OS reprezentuje proces: identifikátory, stav plánování, obsah registrů, plánovací informace, informace o paměti (tab. stránek), účtování, I/O zdroje

- může být rozdělena

Uživatel'ský adresový prostor: přístupný proces - kód + data + stack + sdílené knih. a jejich data, sdílená paměť

uživatel'ská oblast: v Linuxu ne, součást user addr. space, přístupná jen jádru, může si tam ukládat nějaké info o běhu (číslo)

záznam v tab. procesů: trvale v jádře - identifi., stav plánování, na co čeká, odkaz na tab. paměťových regionů (včetně LA)

kontext proc. = stav: uživatelský - kód, data, zásob., sdílená data registry systémový - uživ. obl., položka v tab. proc., ...

identifikátory: PID, PPID, UID, GID, EUID, EGID, saved set-UID/GID, FSUID, FSGID (Linux), PGID (skupina procesů), SID

fork: potomek dědí kód, data, zásobník, sdílenou paměť, otevřené soubory, synchr. prost. nedědí čekající sig., soub. zámký...

exec: zánikují vazby a zdroje vázané na původní kód (obsluhy signálů, sdílené soubory, paměť. mapované soub., semafony)

Plánovač rozhoduje, který proces poběží (a případně jak dlouho). **Nepreemptivní:** je zvrh může dojít, jen pokud běží proc. předs. jinde řízení (ukládání služby, I/O, yield) - vzdá se proc. e

Vlastní přepnutí je dispatcher. Ovlivněno **swapováním** - rozhoduje, co zůstane v paměti.

dluhodobé plánování: rozhoduje, které úkoly budou připuštěny do systému **střednědobé** **krátkodobé:** co zvrh poběží

Přepnutí kontextu: 1) úschova stavu registrů do PCB 2) úprava řídicích struktur 3) obnova stavu registrů z PCB 4) předání řízení neukládá se celý stav procesu! (třeba paměť - pokud je dost)

FCFS: first come first served, nepreemptivní, FIFO fronta

Round-robin - II -, ale preemptivní - procesy mají přidělen časová kvanta

Shortest Job First - přiděluje CPU procesu, který vyžaduje nejkratší dobu bez I/O - CPU burst

- preemptivní - nutnost znát dobu běhu dopředu -> obvykle špatně proveditelný (jen specifické syst.)

- hrozí **starnutí** (hladovění, stacion) - preemptivní varianta **shortest remaining time**

Víceúrovňové plánování - procesy ve skupinách (typický priority), v rámci skupin různé jiné plánovací algoritmy

s různými parametry + alg. vybírající skupinu

II - se zpětnou vazbou - striktně podle **priorit** - nově připravovaný proc. ve frontě s nejvyšší prioritou, postupně klesá

- může používat statickou a dynamickou priority - podle spotřeby času, čekání na I/O (+priority) -> interaktivní procesy

Plánování v Linuxu: víceúrovňové prioritní se 100 základními statickými úrovněmi: 1-99: procesy reálného času (FCFS/round-robin)

- priority 0: běžné procesy -> CFS - podúrovně NICE - 20 až 19 (nejnižší) + tři typy: běžné, dávkové, idle

+ sporadické periodické úkoly s odhaditelnou délkou trvání

complete Fair Scheduler - snaží se všem procesům dát stejně (± priority) - vede si údaj o virtuálním CPU čase

- má procesy ve vyhledávací struktuře uspoř. podle využitého virt. času, vybírá proces s nejmenším, dává mu kvantum podle priority

- nové procesy dostávají virtuální CPU čas podle **minimálního skutečného CPU času** (→ začíná od 0) - zvládá skupinové pl. (multi)

Plánování ve WinNT: víceúrovňové prioritní se zpětnou vazbou na základě interaktivnosti 32 priorit: 1-15 běžné, 16-31 RT, 0 - nulová

- základní priority (statická) ± dynamická - zvyšuje procesy s oknem v popředí, do kterých tečou události (mys), které jsou uvolněny

- zvýšená priority se po vyčerpání kvanta snižuje o 1, až do základu -> priority bursts

inverze priorit: nízkoprioritní proces zamlkne zdroj -> více prioritní proces jej potřebuje -> čeká -> virtuálně vyletí priority nízkop.

+ středně prioritní procesy vždy předbehnou nízkop. -> čekají nízkop. i vysoko priorit. procesy

-> **priority ceiling:** v kritické sekci získá proc. vysokou priority **zákaz přeměnění:** jen na jednoho CPU syst. -> nemůže dojít k přepnutí (nejvyšší priority)

priority inheritance: v KS dědí priority procesů s vyšší prioritou, které čekají na daný zdroj

reprocesové sys. - možnost využít výkon, cache, paměť **hard RTOS:** nutnost zajistit garantovanou odezvu

vládkna: odlehčený proces - vlastní registry a stack, sdílí kód a další zdroje, řízený jedním programem - rychlejší spuštění, př

úlohy: skupina paralelně běžících procesů spouštěných jedním příkazem (v klatě) **skupiny procesů:** možnost poslat nejednu všem signál

možnost čekat na všechny nebo libovolný z nich může mít vedoucího PGID

řízení: skupina skupin procesů může mít řídicí terminál, jedna skupina na popředí (rovinné term.), vedoucímu jde SIGHUP

PC: signály, rouny (nepojm.), fronta zpráv, sdílená paměť, sockety, semafony

signal je číslo, které je procesem získáno pomocí zvláštní dovolené rozhraní. Získání je asynchronní, zajišťuje to OS.
obsluha - pozor na race conditions, náročné ledění (\rightarrow testování, form. verif.)
Generování: při chybách (SIGSEGV); externích událostech (SIGALRM); na žádost procesu (\Rightarrow IPC)
SIGPIPE: nruka bez čtenáře SIGTERM: měkké ukončení SIGKILL: tvrdé uk. (nebe předdefinovat) SIGUSR1/2 SIGCHLD: pozat./ukon. p.
SIGSTOP: tvrdé pozastavení (nelze předef.) SIGTSTP: měkké pozastavení SIGCONT: odblokování proc.
implicitní: ukončení, zmrazení, rozmrazení, ignorování, core dump **Sigprocmask** - blokování (nast. masku), dělí se do potních
zablokované signály čekají, ale pamatuje se jen jeden z nich **Sigatomic_t**
získání: kill nebo sigqueue - umí na začech nést int/void* hodnoty pid - určit proces (+), skupina procesů (-); věřící, kterým mohou
čekání: pasivní - pause() nebo sigsuspend() - atomicky může přepnout mezi signály blokováními mimo a po dobu čekání (avg. mask)

RACE CONDITION = časově závislá chyba, **souběh**, je chyba vznikající při přístupu ke sdíleným zdrojům kvůli různému
pořadí provedení jednotlivých paralelních výpočtů / operací v systému. (data race) - dva přístupy ke zdroji s výsle. př. aspoň jeden zápis pro

SYNCHRONIZACE slouží k tomu, aby procesy navzájem nekolabovaly a v systému nedocházelo k nekonzistentním stavům (data).
Zajišťuje správné pořadí provedení spolupracujících procesů.

SDÍLENÍ KŘITICKÝMI SEKCEMI dané neprázdné množiny procesů rozumíme ty úseky jejich řídicích programů přistupující
ke sdíleným zdrojům, jejich provedení jedním procesem vylučuje současně provedení libovolného z těchto úseků ostatními.
- může existovat více sad sdílených KS, které navzájem sdílené nejsou (\Rightarrow různé sady prom...)

Problém KS = problém zajištění korektní synchronizace procesů na dané množině SKS
 \Rightarrow **vzájemné vyloučení** (navzájem jeden (oba) k) procesů je v daném okamžiku v dané množině SKS

dostupnost: je-li KS volná, proces nemůže neomezeně čekat na přístup do ní \Rightarrow **deadlock** (uvěznění)

Blokování při přístupu do KS: proces, který žádá o vstup do KS, musí čekat, přestože je KS volná **blokování** **stárnutí** (starvation)
(se nenachází v žádné sdílené KS) a ani o žádnou z dané množiny SKS žádný další proces nežádá. (žádný proces
"Sdílený prostředek je volný, proces na něj ale musí dlouho čekat.")

Stárnutí je situace, kdy proces čeká na podmínku, která nemůže nastat. (na zdroje, které nikdy nedostane.)
 \Rightarrow **Stárnutí při přístupu do KS** - podmínka je umožnění vstupu do KS. ! Může být způsobeno tím, že plánovač nikdy nevybere proces
ve správný okamžik

Livelock je situace, kdy každý proces z určité neprázdné množiny procesů běží, ale provádí jen
omezený úsek kódu, ve kterém opakovaně žádá o nějaký zdroj s vyloučeným přístupem, který je vlastně nekterým
z procesů dané množiny a jen ten by jej mohl uvolnit, pokud by mohl pokračovat. (= deadlock s aktivním čekáním)

Offmanovy podmínky: nutné a postačující pro deadlock: 1) vzájemné vyloučení při používání prostředku
2) vlastnictví alespoň jednoho zdroje, pozastavení a čekání na další 3) prostředky vrací pouze proces, který je vlastní, po dobu
4) cyklická závislost na sebe čekajících procesů

Prevence uvěznění: ① \rightarrow nepoužívat sdílené prostředky, nebo pouze sd. pr., které nevyžadují vzájemné vyloučení (atomické)

② \rightarrow proces může žádat pouze tehdy, když žádný neexistuje, zamyká tak **všechny najednou**

③ \rightarrow proces může zamlouvat jen to, na co nemusí čekat (jinak je třeba zabít) (může dojít k nekonzistenci)

④ \rightarrow zamezit cyklem - např. striktním pořadím zamykání zdrojů, spec. algoritmy (\rightarrow je nutná pořádná verifikace)

Vyhýbání se uvěznění: procesy předem deklarují, které zdroje budou používat a jakým způsobem
 \rightarrow systém přidělování zdrojů má info o možných požadavcích + aktuální situaci \rightarrow vyhnutí žádosti, pouze pokud by
ani v nejhorším možném případě nemohlo dojít k cyklické závislosti

\rightarrow **graf alokace zdrojů** - dva typy uzlů (P, R), tři typy hran (zdroj je vlastní, proces žádá, proces může žádat)

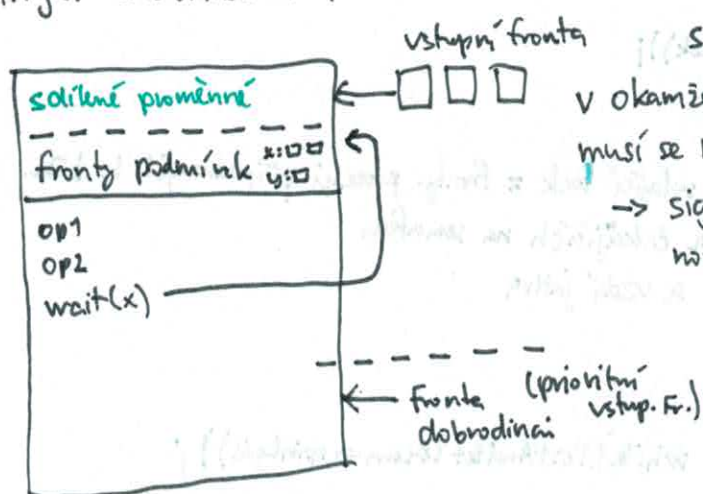
- cyklus při záměře některé hrany žádosti za hranu vlastnictví

Detekce a zotavení:
Uvěznění může vzniknout - pomíneme-li to, že je v ně uvězněných proc. dle speciálního systému k jeho rozřešení -
periodicky se přitom detekuje, jestli k němu nedošlo; pokud ano, provede se zotavení.

Zotavení: odebrání některého zdroje, příp. restart, ukončení \rightarrow pozor na nekonzistence (rollback)

Monitory: vysokoumrovne synchronizacni prostredky, definovana struktura se sdilenymi promennymi, ke kterym se da pristoupit jen pres metody, které pristup k nim zahrnuji

Synchronizace UVNITŘ monitору: pomocí **PODMÍNEK**, které mají operace
wait(x) - zařadí do fronty i někdo z enty queue je vybrán a může začít pracovat v monitru
signal() - dobrodinec předává řízení čekateli → dobrodinec jde do fronty dobrodinců
notify() - dobrodinec říká, že se někam dostal → čekatel je ve frontě dobrodinců a čeká na dokončení dobrodince



```
typedef struct {
    sem_t mutex, condition, priority;
    int prioCount, condCount;
} mon_t;
```

```
void mon_enter(mon_t *m) {
    sem_wait(&m->mutex); // postav se do fronty k mutexu
}
```

```
void mon_wait(mon_t *m) { // čekání na podmínku (synchronizace uvnitř monitru)
    m->condCount++; // zvýšit počet čekajících na podmínku - synchronizace, v monitru je jen jeden
```

```
    if (m->prioCount > 0) {
        sem_post(&m->priority); // umožní vstup z prioritní fronty
    } else {
        sem_post(&m->mutex); // v prioritní nikdy, umožní vstup zrujším
    }
}
```

```
2) sem_wait(&m->condition); // začni čekat
x m->condCount--; // běžím jako první a jediný (opět), můžu schytat m cC
}
```

```
void mon_signal(mon_t *m) {
    if (m->condCount > 0) { // jen pokud někdo čeká na podm., jinak no-op
        m->prioCount++;
        sem_post(&m->condition); // pokračuje ten, kdo čeká na podm.
        sem_wait(&m->priority); // já sám se zařadím do priority fronty
        m->prioCount--; // už nečekám, mám exkluzivní řízení
    }
}
```

```
void mon_leave(mon_t *m) {
    if (m->prioCount > 0) { // pustím buď někoho z prio, nebo celý monitor
        sem_post(&m->priority);
    } else {
        sem_post(&m->mutex);
    }
}
```

```
void mon_init(mon_t *m) {
    sem_init(&m->mutex, 1);
    sem_init(&m->condition, 0);
    sem_init(&m->priority, 0);
    m->prioCount = m->condCount = 0;
}
```

```
void mon_notify(mon_t *m) {
    if (m->condCount > 0) {
        m->condCount--;
        sem_post(&m->condition);
    }
}
```

to je vlastně leave

3) sem_wait(&m->mutex); // dobrodinec dostal k notify(), j. výjezd ven do čekací fronty, dobrodinec poklidí

Konceptuální implementace semaforu:

```
typedef struct {
    int val;
    bool spinlock;
    process-queue *q;
} sem_t;
```

```
void init (sem_t *sem) {
    sem->val = 1;
    sem->spinlock = false;
    //init-queue (sem->q);
}
```

```
void lock (sem_t *sem) {
    /* synchronizace */ while (TestAndSet (&sem->spinlock));
    sem->val--;
    if (sem->val < 0) {
        process_t p = pop (readyQueue); // odstraní proces volající lock z fronty procesů připravených k běhu
        push (sem->q, p); // přidá volající proces do fronty čekajících na semafor
        switch (); // přepne kontext; volající proces se vzdá jádra
    } else { sem->spinlock = false; }
}
```

```
void unlock (sem_t *sem) {
    /* synchronizace */ while (TestAndSet (&sem->spinlock));
    sem->val++;
    if (sem->val > 0) {
        process_t p = pop (sem->q); // vezme čekající proces
        push (readyQueue, p); // přidá jej do fronty p. připravených k běhu
    } else { sem->spinlock = false; }
}
```

Swap pro spinlock:

```
bool spinlock = false;
...
bool key = true;
while (key) Swap (&spinlock, &key);
// KS
spinlock = false;
```

Multiproc. T&S:

```
bool slock = false;
...
while (TestAndSet (&slock)) {
    while (slock); // počká se pkine cache, nemusíme do paměti
    // potom ale musíme znova atomicky zjistit,
    // jestli je tam opravdu stále false
}
}
```


Aby mohl být program proveden, musí mu být přidělen procesor a přidělena **paměť**.
LAP = virtuální adres. prostor (procesor s tím pracuje při provádění) **FAP** - společný pro všechny (jde na stránci...)
- mohou se překrývat (kus AP jádra v LAP každého procesu)

Contiguous Memory Allocation: jednoduchý - ke každému procesu má MMU 2 údaje - limit + relocation
→ externí fragmentace → minimalizace dopadů (strategie - first/best/worst fit, binary buddy - štipu nejvhodnější kus pro pil
+ problém se zvětšováním + není možné jemné řízení práv, sdílení kódu

Segmentace: LAP rozdělen na segmenty, ty nemusí být všechny ulož. spojitě v paměti pole segmentů, adresa = č. seg. + displacement
- mohou být nastaveny různé práva, sdílení... ale problém ext. frag. furt je + je viditelná procesu (→ bad)

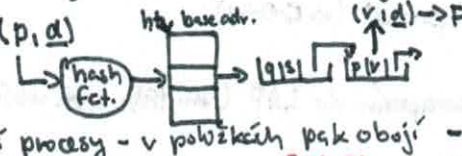
Stránkování - LAP dělen na stránky, FAP na **rámce**, pevná velikost. Paměť přidělována po rámcích. Minimalizace ext. frag. (ale i nevýhodné pro procesy) jemné jednotky ochrany (r/w, user/sys, NX bit...) a sdílení a odkládání (swap)

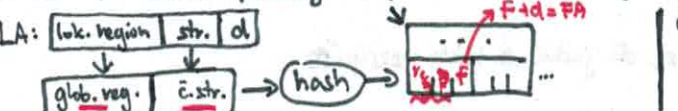


Organizace volných rámců a tabulky stránek pro procesy drží jádro! (Podle MMU daného procesoru.)

Tab. stránek: mapování + příznaky - **platnost** (je psát?), **přístup** (byl zpřístupněn? - swap), **modifikace**, práva, možnost provádění
příznak **globality** → při přepnutí kontextu nemusím měnit

TLB: translation look-aside buffer: dvojice (číslo str., číslo rámce) + některé příznaky
asociativní paměť → paralelní vyhledávání → obvykle částečně - několik bitů použito pro odhadování nejčastějších asoc. bloků
- může být řízena HW nebo SW (MIPS) - v PT hledá jádro a uloží výsl. do TLB - nikdy víc vrstev, hierarchie
- při změně kontextu invalidace (x př. globality, PID v TLB), při změně PT teky - TLB pre fetching - **lokalita**! stránky ad.

Hierarchické PT: stránkování tabulky stránek - rozdělení LA na displacement (offset) v rámci stránky + n indexů do vrstev PT
x86-64: 4úrovňové: 12b offset + 4. 9b indexy do PT (= pouze 48 b), PML4T, PDPT, PDP, PT, jedna položka PT má 8 B
- ne vždy použity všechny vrstvy → „velké“ stránky - výrazně roste význam TLB - podpora virtualizace

Hashování PT: $LA(p, d)$  různé mod.: více HT, jako klíč jen několik bitů
na jednom řádku více záznamů, jádro si vede vlastní záznamy (extra PT), kde dohledává, co nenajde v HT
- může být sdílené procesy - v položkách pak obě - možné rozdělit na regiony, adresa daná číslem reg → nějaká speciální tab.

LA:   

Inventurní PT: pro každý rámec udává, který proces má do něj kterou stránku → jedna pro všechny, číslo rámce = index v tab.
musí (sekvenčně) prohledávat pro dvojice (PID, P) → kombinuje se s hashováním

Virtualizace paměti umožňuje procesům a jádru pracovat s oddělenými lineárními LAP. V případě mapování pomocí stránek a segmentů je možné zajistit, aby ne všechny využitý LAP byl celý umístěn ve fyzické paměti.
→ menší spotřeba paměti, rychlejší swapping - pro uložení části LAP se využívá prostor na disku.

Stránkování na žádost: stránky zaváděny, jen když jsou potřeba. Informace o stavu stránek si vede jádro v pomocných str.
v PT (TLB) pouze příznak „je přidělen rámec?“ - pokud ne, **výpadek stránky** → TRAP interrupt

Obsluha výpadků: kontrola limitů → absence rámce (pokud není → victim page, kterou odhazíme/zmažeme) → inicializace
rámce: (první odkaz) → kód + inicializ. data načtená z programu; vše ostatní se **nuluje**
(v minulosti už uvolňováno) → -||- konst. data načtená z progr. i ostatní, modifikované stránky načteny ze swapu, jinak nulování
→ namapování rámce na stránku → proces zpět do st. ready, zavážen do plánování

Výpadky: nezarování instrukce, nezarování dat, data delší než stránka, výpadky (vznikají úloví) PT
odkládání: nastává (nejm) při výpadku jiné stránky, **lokální**: odkládá se jiná stránka daného procesu **globální**: odkládá se jakákoliv str.
obvykle je snaha mít vždy nějaký počet volných → page daemon (zloděj str.)

FIFO: odstraňuje nejstarší zavedené stránky, primitivní x může odstranit často používanou str., Bědého anom. = více paměti → více výpadků
dá se zlepšit nějakou heuristikou - odkládá hned, jen si ho zasl. jako kandidáta a čeká, jestli se k němu bude přistupovat

LRU: least recently used - aproximace ideálního alg. (který by znal budoucnost) u velkých polích v cyklech (bubblesort) problém
- problematické imp. - detekce použití, časová razítka → používají se aproximace

Aproximace LRU pomocí omezené historie referenčního bitu stránek: v PT u stránek uložen **referenční bit** - při přístupu
jej HW nastaví na 1; jádro si vede o bitu historii (kousek) - posouvá historii, při uložení do hist. bit v PT vynuluje;
oběť = stránka s nejnižší hodnotou **Algoritmus druhé šance**: stránky v konkrétním seznamu, posouvá ukazatel „další oběť“
nuluje ref. bity, pokud najde 0, použije jako oběť (→ další jediná šance) **modifikace**: upřednostnění nemodifikovaných,
dva ukazatele pro nulování a označování obětí s rozstupem **Linux**: fronta reaktivních str., vyhledává stránky pro caching

Alokace rámců: suchá, aby proces byl dříve v mezích → lokální výběr obětí → kolik je dobré?

- vždy potřeba počet pro provedení jedné (nejšíšíšíší) instrukce
- různé heuristiky - podle velikosti programu, priority, **pracovní množiny str.** (aproximace pomocí ref. b.)
 - sledování frekvence výpadků

Windows: minimální a maximální počet rámců při dosažení maxima lokální výměna
pokud je nedostatek výrazný, prochází všechny procesy - glob. vým., omezení hist. přístupů

- upřednostňuje méně často běžící - vyhýbá se proc. s BUI na popř.; těm, u kterých bylo moc výpadků
- obětí ještě chvilu nechá být
- dlouho neaktivní odswapeje pryč celé

Trashing: jev, kdy (systém) proces strávil víc času nahrazení stránek než užitečným výpočtem.

- často v systémech s vysokou mírou paralelismu
- **swapper:** pozastaví některé procesy a odloží vše (při faktálním nedostatku paměti)
- když dojde swap, je třeba zabít něco.

Prepaging: přednastavení většího počtu stránek na jednom

Zamykání stránek: zabránění odložení

u I/O stránek (DMA), částí PT, některých stránek jádra, např. (v omezené míře)

Sdílení stránek:

- kódy programu
- konstantní data, nemodifikovaná data u kopií procesů (→ COW)
- IPC
- paměťově mapované soubory → celý soubor namapován do LAP (není třeba syst. volání)
- **sdílení knihovny**

COW fork: - stránky označené jako COW, jsou sdíleny, dokud se do jedné z nich nezapiše

- nemusí se na jednom kopírovat všechno