1.Kapitola

1.1 Co je operační systém.

Úkolem operačního systému je zajistit řádný a řízený přístup k procesoru, paměti, V/V

zařízením mezi různými programy, o ně se ucházejícími

operační systém zajišťuje dvě základní, ne zcela související funkce:

Operační systém jako stroj s rozšířenými možnostmi

- Jedná se o pohled **z hora–dolů**
- Program, který ukrývá realitu hardware před programátorem a prezentuje „příjemný" jednoduchý pohled na pojmenované soubory pro čtení i zápis.
- Program který pro programátora prezentuje jednoduché rozhraní na kontrolu počítače (přerušení, časovače, správu paměti a další nízko úrovňové funkce a vlastnosti)
- abstrakce nabízená operačním systémem je jednodušší a snadnější pro použití, než základní hardware
- stroj s rozšířenými možnostmi
- virtuální stroj - je pro programování jednodušší

Operační systém jako správce prostředků

- Jedná se o pohled **zdola–nahoru**
- drží a spravuje všechny části komplexního systému
- Úkolem je zajistit řádný a řízený přístup k procesoru, paměti, V/V zařízením mezi různými program, o ně se ucházejícími. (hlavní úkol sledovat, kdo používá jaké zdroje, vyhovět žádostem o zdroje, řídit jejich používání a urovnávat konflikty mezi požadavky od různým programů i uživatelů)
- může přinést pořádek do potencionálního chaosu ukládáním všech výstupů na zařízení na disk
- Hlavní úkoly:
    - sledovat, kdo používá jaké zdroje
    - vyhovět žádostem o zdroje, řídit jejich používání
    - urovnávat konflikty mezi požadavky od různým programů i uživatelů

1.2 pojmy: dávka

- program, nebo skupina programů, na děrovacím štítku/štítkách (musel programátor napsat program na papír ve Fortranu nebo assembleru a pak vyděrovat na štítky), tištěný výstup.
- Jedna za druhou vykonávaná, vstupní místnost na příjem lístku obsluhou
- Struktura
    - $JOB Time, ID-Number, Name
        - Time = maximální množství času pro vykonání dávky (v minutách)
        - ID-number = identifikace účtu
        - Name = jméno programátora
    - $FORTRAN
        - Instruuje operační systém, aby spustil překladač FORTRANu z pásky, ten očekává zdrojový text programu (Fortran program ...)
    - $LOAD
        - Příkaz pro os na načítání přeloženého programu do paměti
    - $RUN
        - Příkaz pro os na spuštění programu s daty, které ho následují (Data for programm ...)
    - $END
        - označuje konec dávky

dávkový systém

- minimalize časové stráty dávek
- Postup
    - všechny dávky seřadí ve vstupní místnosti a pomocí malého počítače se nahrají na magnetickou pásku, Ta se přenese na samotný výpočetní počítač, kde jsou obvykle tři páskové mechaniky: vstupní, systémová a výstupní
    - Když jsou všechny dávky nahrány na pásku, páska se převine a zanese k hlavnímu počítači, kde se vloží jako vstupní mechanika
    - Obsluha spustí speciální program (předchůdce dnešního operačního systému), který čte jednotlivé dávky z pásky a spouští je
    - Výstup je zapisován na pásku, místo přímého tisku
    - Když je jedna dávka ukončena, načítá se z pásky další a spouští
    - Když jsou všechny dávky zpracovány, obsluha vymění vstupní a výstupní pásku a výstupy zanese k off-line tiskárně, která obsah výstupní pásky pomoci čtečky vytiskne

Multiprogramování

- Když na počítači aktuální dávka čekala na data z pásky nebo jinou V/V operaci, procesor zůstal ve stavu čekání na V/V požadavek (pro rozsáhlé vědecké výpočty kde jsou V/V Operace ojedinělé ztráta času není významná, u komerčního zpracování jsou V/V operace často až 90% a proto je třeba něco udělat aby CPU nečekal moc dlouho).
- **Řešení,** spočívalo v rozdělení paměti na několik bloků(partitions) s různými dávkami v každém z nich.
- **Nutnost,** mít jiný hardware na ochranu jednotlivých dávek mezi sebou proti sledování i chybám ostatních.

spooling

- Simultaneous Peripheral Operation On Line
- číst dávky z děrných štítků na disky
- jakmile běžící dávka skončila, operační systém mohl natáhnout z disku do uvolněného bloku paměti další dávku a spustit ji
- také použit pro výstup
- nebyla třeba počítače pro čtení štítků na pásky a zmizela částečně nutnost přenášet pásky

sdílení času

- každý uživatel má on-line terminál, více uživatelů nad jedním počítačem
- Ale jenom část aktivně pracuje, je tedy možné rozdělit „čas“ mezi aktivní a netřeba jím plýtvat pro neaktivní uživatele.
- počítač rychle a interaktivně pracuje s požadavky mnoha uživatelům a možná také pracuje na velkých dávkách na pozadí, když je CPU v čekacím stavu
- stalé dávky
- nestalo příliš oblíbené, dokud se potřebné ochrany hardware nestaly dostatečně rozšířené
- reakce na čas mezi přijetím dávky a získáním výsledků trvalo i několik hodin

1.3 Koncepce OS

- Rozhraní mezi operačním systémem a uživatelským programem je definováno jako množina „rozšiřujících instrukcí, které operační systém vykonává (systémová volání-system calls)
- systémová volání dostupná v rámci rozhraní OS se liší mezi jednotlivými operačními systémy (i přesto, že základní koncepce je podobná).
- Systémová volání MINIXu se dělí zhruba na dvě vymezující kategorie:
    - manipulace s procesy
    - manipulace se soubory

Procesy

- v podstatě vykonávaný program
- Ve spojitosti s každým procesem je jeho:
- adresový prostor
    - obsahuje proveditelný (spustitelný) program, jeho data a zásobník
- seznam alokací paměti od minima do maxima
    - ve kterých proces může pracovat, tedy číst a zapisovat
- Ke každému procesu také patří určitá skupina registrů, včetně čítače instrukcí (instruction pointer), zásobníku a jeho vrcholu (stack pointer) a další registry hardware potřebné pro běh programu
- Všechny informace procesu musí být uložené (např. ukazatel na aktuální pozici Souboru) aby se pozastavený proces mohl znova spustit tam kde skončil
- všechny informace o procesu jiné, než je obsah vlastního adresního prostoru (nazývaný obraz procesu – core image), jsou uložený v **tabulce procesů** (process table)
    - což je pole nebo zřetězený seznam s položkou pro každý existující proces zvlášť.
    - Jsou zde uloženy registry, včetně dalších informací
- Klíčovou činností pro správu procesů jsou **systémová volání** pro:
    - vytvoření a ukončení procesu
    - realokaci množství paměti
    - čekání na ukončení potomka
    - nahrazení jednoho programu jiným
- Když proces může vytvořit jeden nebo více procesů, říkáme jim dětské procesy, potomci (child process)
- Související procesy, které mají provést určitý úkol spolu často potřebují komunikovat a synchronizovat aktivity. Tuto komunikaci nazýváme **mezi-procesní komunikace - IPC** (interprocess communication)

- Po uplynutí časového limitu posílá operační systém programu **signál**. (**programová analogie s přerušením u hardware**)
- **Signál** zapříčiní, že
    - proces je dočasně pozastaven
    - uloží registry na zásobník
    - spustí vykonávání speciální funkce pro obsluhu signálu (např. přeposlání pravděpodobně ztracené zprávy)
    - po ukončení funkce se proces znovu spustí ve stavu, v jakém byl před signálem
- **Signál** může být generován z mnoha příčin
    - Časový limit
    - Vykonávání ilegální instrukce
    - porušení ochrany paměti

- Každý proces spuštěný v MINIXu má **uid** osoby, která jej spustila.
    - Jedno uid, nazývané **super-user** (root) = může porušovat mnoho ochranných mechanismů
- Interpret příkazů (command interpreter) nebo shell = proces na přečtení příkazů z terminálu.

Soubory

- Systémová volání jsou obvykle potřebná (pro práci se **soubory**)
    - vytvoření, smazání, čtení a zápis
- Než bude ze souboru čteno, musí se otevřít a po ukončení čtení se musí zavřít, což je úkol systémových volání.
- Pro umístění souborů používáme **princip adresářů**, jako metodu pro **uložení souborů do skupin**.
- Potrebné Systémová volání pro **Adresáře**
    - vytvoření a smazání adresáře
    - umístění souboru/adresáře do adresáře a smazaní souboru z adresáře
- Jednotlivé položku adresáře jsou soubory nebo adresáře
- Dostáváme tak hierarchický model souborového systému
- Hierarchie procesů i souborů je sice organizována jako strom, ale tím podobnost končí
- Hierarchie procesů není příliš hluboká, obvykle 3-4 úrovně. Zatímco hierarchie souborů je obvykle 5-6 i více úrovní. Hierarchie procesů má krátkou životnost, obvykle maximálně minuty, zatím co u souborů může existovat i roky. Je taky odlišné vlastnictví i ochrana
- Všechny soubory v adresářové struktuře lze specifikovat uvedením cesty (path name) od kořene (root directory) adresářové hierarchie
    - **úvodni lomítko / - absolútna cesta od koreňa,**
    - **úrovne oddelené lomítkom**
- **Procesy** mají aktuální pracovní adresář (working directory), ve kterém se hledají soubory nezačínající lomítkem
    - můžou měnit pomocí systémového volání
- Soubory a adresáře jsou v MINIXu chráněny přiřazením 9 bitové ochranné masky všem souborům
    - skládá ze tří trojic bitů(vlastníka, skupinu vlastníků, pro všechny ostatní)
        - bit určující přístup pro čtení (read)
        - bit pro zápis (write)
        - bit pro spuštění (execute)
        - prázdný bit – přístup není povolen
    - při otevírání se kontrolují přístupová práva
        - Pokud je přístup povolen, dostáváme popisovač souboru (file descriptor,handle) pro použití v následujících operacích
        - Pokud je přístup odmítnut, je návratovou hodnotou chybový kód

- Externí dátové nosiče (napr. disketa) můžou být dostupné až po systémovém volaní **MOUNT**, který **souborový systém** externého nosiče **připojí do koreňového souborového systému** kdekoliv (vybraný adresář jak něco obsahoval tak tyto dsta během připojení nosiče nejsou dostupné)
- **Speciální soubory (special file,devices)**
    - zajišťují přístup k zařízením jako k souborům
        - přistupujeme pro čtení i zápis stejnými systémovými voláními, používanými pro práci se soubory
    - Jsou dva druhy
        - Blokové
            - pro zařízení složené z náhodně adresovatelných bloků(disky)
            - přistoupí přímo na zařízení, bez znalosti souborového systému
        - Znakové
            - přijímají nebo vysílají proud znaků
            - tiskárny, modemy a další zařízení

náležící souborům i procesům

Roury

- typ pseudo-souboru
- může být použit ke vzájemnému spojení dvou procesů dohromady
- vytvoření roury zajišťuje systémové volání
- lze si představit jako soubor se dvěma konci
    - do jednoho konce zapisujeme a ze druhého čteme pomocí systémových volání pro práci se soubory

Systémová volání

Správa procesů:

- fork - vytvoří nový proces - potomka, identického s rodičovský procesem,
- waitpid - čeká na ukončení potomka,
- wait - starší verze waitpid,
- execve - nahradí obraz procesu jiným programem,
- exit - ukončí aktuální proces,
- brk - nastaví velikost datového segmentu,
- getpid - ID aktuálního procesu,
- getgrp - ID skupiny aktuálního procesu,
- setsid - nová session jako ID aktuálního procesu,
- ptrace - pro trasování.

Signály:

- sigaction - definuje obsluhu signálu,
- sigreturn - návrat ze signálu,
- sigprocmask - přebere nebo změní masku signálů,
- sigpending - vrátí množinu blokovaných signálů,
- sigsuspend - vymění masku signálů a pozastaví proces,
- kill - zašle signál procesu,
- alarm - nastaví hodiny alarmu,
- pause - pozastaví program do dalšího signálu

Správa souborů:

- creat - vytvoření souboru (zastaralá funkce),
- mknod - vytvoření souboru, speciálního souboru nebo adresáře,
- open - otevření souboru,
- close - uzavření souboru,
- read - čtení dat ze souboru,
- write - zápis do souboru,
- lseek - nastavení ukazatele v souboru,
- stat - status souboru,
- fstat - status souboru,
- dup - nový deskriptor pro otevřený soubor,
- pipe - vytvoření roury,
- ioctl - provádí speciální akce se souborem,
- access - ověří přístup k souboru,
- rename - přejmenuje soubor,
- fcntl - zamykání souborů a další operace.

Adresáře a správa souborů:

- mkdir - vytvoření nového adresáře,
- rmdir - smazání prázdného adresáře,
- link - nový odkaz na soubor,
- unlink - zmazání položky v adresáři,
- mount - montování souborového systému,
- umount - odmontování souborového systému,
- sync - zapíše neuložené bloky vyrovnávací paměti na disk,
- chdir - změna pracovního adresáře,
- chroot - změna kořenového adresáře.

Ochrana:

- chmod - změna ochranné bitové masky,
- getuid - UID volajícího,
- getgid - GID volajícího,
- setuid - nastaví UID,
- setgid - nastaví GID,
- chown - změna vlastníka a skupiny souboru,
- umask - změna masky.

Řízení času:

- time - čas v sekundách od 1. 1. 1970,
- stime - nastavení času,
- utime - nastavení času posledního přístupu,
- times - uživatelský a systémový čas procesu.

2. Kapitola

2.1 Úvod do procesů

- moderní počítače musí provádět více věcí současně
- **pseudoparalelismus -** rychlé a neustálé přepínání CPU mezi programy
- **skutečný paralelismus -** hardware v multiprocesorových systémech (s více procesory a sdílenou pamětí)
- **sekvenční model** - umožňuje snadné zvládnutí paralelismu (reakce na souběžnou starost člověka o více paralelních aktivit)

2.1.1 model procesu

- Všechny spustitelné programy(i OS), organizovány jako řada **sekvenčních procesů**
- Proces= běžící program + hodnoty programového čítače + registry a proměnné. (Obecněji má každý proces svůj vlastní virtuální CPU, Reálně ovšem CPU neustále přepíná mezi procesy)
- systém rychlého a neustálého přepínání se nazývá **multiprogramový**
    - postup nebude ve přepínání stejnoměrný a pravděpodobně ani nebude opakovatelný, pokud stejné procesy spustíme znovu=**procesy nesmí být programovány s pevnou vazbou na čas**.
    - Pokud má proces kritické nároky na požadavky v reálném čase, musí přijít určitá událost během specifikovaného časového intervalu v řádu milisekund a speciálním měřením musí být zajištěno, že se tak stalo
- Rozdíl mezi **procesem** a **programem** je malý, ale zásadní.
    - Program je recept/návod
    - Proces je aktivita která program načítá a vykonává z nějakými vstupními parametry (má program, vstup, výstup a stav)
- Jeden procesor může být sdílen mezi více procesy s určitým plánovacím mechanismem, rozhodujícím, kdy zastavit práci na jednom procesu a přejít na jiný

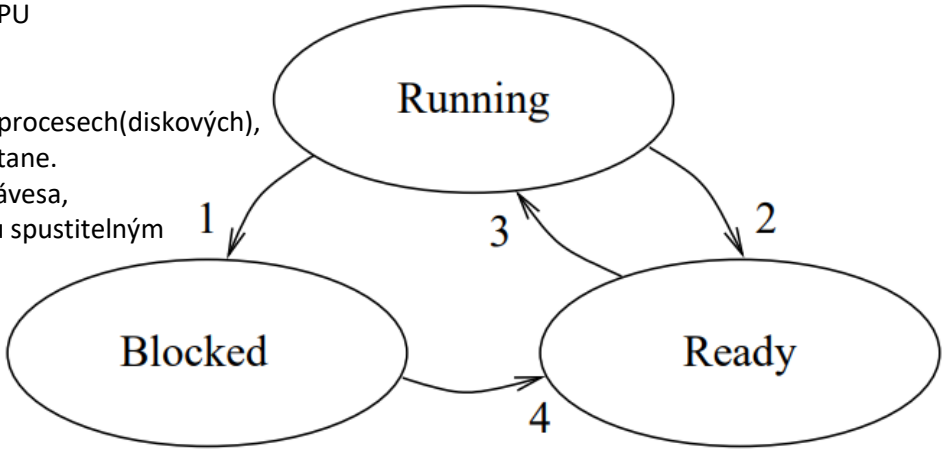
stavy procesu

Tři stavy:

- **Running - běžící** - právě využívá CPU
- **Ready - připravený** - dočasně pozastavený během jiného programu
- **Blocked - zablokovaný** - nemůže běžet, dokud nepřijde nějaká externí událost

Čtyři různé přechody mezi Stavy:

- **1 (Running -> Blocked)** – když zjistí, že nemůže pokračovat (některé systémy volají systémové volání BLOCK a některé systémy to dělají automaticky když během čtení nejsou dostupná data)
- **2 (Running -> Ready)** – vyvolán plánovačem procesů, rozhodl, že proces již běžel dostatečně dlouho a je čas předat CPU jinému procesu
- **3 (Ready -> Running)** – nastane pokud je plánovačem proces vybrán jako ten, komu bude předán CPU
- **4 (Blocked -> Ready)** – nastane, když přijde očekávaná externí událost, na kterou proces čekal. Pokud právě neběží žádný proces, přejde proces přechodem 3 ihned do stavu běžícího. Jinak bude ve stavu připravený čekat po krátkou dobu na přidělení CPU



Místo úvah o přerušení, můžeme uvažovat o procesech(diskových), které jsou zablokovány čekáním, až se něco stane.
Když je blok z disku načten, nebo stisknuta klávesa, proces čekající na odblokování se stává znovu spustitelným

Plánovač

- Správa všech přerušení
- podrobnosti o právě běžících a zastavených procesech
- mezi-procesní komunikace (nejspíše myšleno jako přerušení)

2.1.2 Implementace procesů

- Pro implementaci modelu založeného na procesech musí mít operační systém **tabulku procesů (process table)**
  - jeden záznam pro každý proces
  - obsahuje informace o:
    - alokaci paměti, status otevřených souborů, účtovací a plánovací informace
    - vše o procesu, co musí být uloženo, když se proces přepíná ze stavu (Running -> Ready), tak aby mohl být později znovu spuštěn, jako by nikdy nebyl zastaven
- Spojení s V/V zařízeními je uloženo na začátku paměti - **vektor přerušení**
  - Obsahuje adresy podprogramů obsluhy jednotlivých zařízení
- Aktuální číslo procesu a ukazatel na něj je uložen v globální proměnné, aby mohl být rychle vyhledán
- Obsluha přerušení a plánování
  - 1. Hardware uloží na zásobník čítač instrukcí atd.
  - 2. Hardware vezme nový čítač instrukcí z tabulky přerušení.
  - 3. Podprogram v assembleru uloží registry.
  - 4. Dále nastaví nový zásobník.
  - 5. Spustí obsluhu přerušení v jazyce C.
  - 6. Plánovač označí čekající proces jako připravený.
  - 7. Plánovač rozhodne, který proces bude následně spuštěn.
  - 8. Obsluha v C se vrátí zpět do assembleru.
  - 9. Podprogram v assembleru znovu spustí nový aktuální proces.

vlákna (pouze implementovaná systémem)

- jedno řízené vlákno a jeden programový čítač v každém procesu
- Řídící vlákna nazýváme právě jako vlákna, řídčeji lehkými procesy
- Každý proces má svůj adresní prostor a jedno řídící vlákno:
  - Tři procesy s jedním vláknem-každé z nich pracuje v jiném adresním prostoru
  - Jeden proces se třemi vlákny-všechna tři vlákna sdílí jeden adresní prostor
- Příklad použití
  - v rámci serveru, server môže paralelne spracovávať viač požiadavkou, a zároveň počas spracovávania ostatných, prípadne čakania na V/V operáciu príjmať nové
  - firefox, požiadanie na viac obrázkov naraz
- Pokud je v jednom adresním prostoru více vláken, pak některé položky z tabulky procesů nejsou pro každý proces, ale pro každé vlákno , v separátní **tabulce vláken**
- **Tabulka vláken**
  - jeden záznam pro každé vlákno
  - obsahuje registry, status a programový čítač
- Všechny tyto položky jsou nutné proto, že každé vlákno, stejně jako proces, může být pozastaveno a znovu spuštěno a může být i zablokováno
- Problémy (ťažko povedať či to treba)
  - Fork – rodič má viac vlákien, má ich mať aj potomok ? ak áno čo keď pri forku nejaké vlákno čakalo na vstup ? keď užívatel dá vstup kto to má dostať ?
  - Čo sa stane keď vlákno zavrie súbor a ďalšie stále používa ?
  - doalokovávanie pamäti ? spraví to ktoré vlákno ?
  - ako reagovať na signály ?
  - zásobníky, viac vlákien viac zásobníkov ako doalokovávanie ?
- Ačkoliv tyto dvě alternativy vypadají stejně, liší se zásadně ve výkonnosti. Přepnutí vlákna v adresním prostoru uživatele je výrazně rychlejší, než přepnutí voláním jádra

2.2 Mezi-procesní komunikace (interprocess communication) – IPC

- Procesy potřebují komunikovat s dalšími procesy, přednostně dobře strukturovaným způsobem, bez použití přerušení
- Například v rouře shellu se musí výstup prvního procesu předávat do druhého procesu a tak dále
- Problémy:
    - Jak může proces poslat informace jinému?
    - Aby si dva, nebo více procesů, vzájemně nelezlo do cesty, pokud provádějí libovolné kritické aktivity (např. pokud dva procesy chtějí alokovat poslední 1MB paměti).
    - A třetí problém předpokládá správné pořadí při vzájemných závislostech: pokud proces A produkuje data a proces B je tiskne, musí B počkat, dokud A nevyprodukuje nějaká data, než začne tisknout.

2.2.1 Souběh (Race Conditions)

- Situace, kdy dva nebo více procesů čtou a modifikují **sdílená data** a finální výsledek je závislý na tom, kdo přesně kdy běžel
- Těžké hledání chyb v programech,kde je souběh, výsledky vetšiny testu jsou správné ale ve výjimečných případech, kdy se stane něco neočekávaného, nastane nevysvětlitelný výsledek
- **Sdílená paměť** může být hlavní paměť, nebo sdílený soubor

2.2.2 Kritická oblast (Critical Section)

- část programu, kde se přistupuje ke sdílené paměti nazýváme **kritickou oblastí** nebo sekcí (**critical section** or region).
- Jak se vyhnout souběhu? zakázat, aby více než právě jeden proces nemohl číst a modifikovat sdílená data. Jinými slovy, potřebujeme vzájemné vyloučení (mutual exclusion)
- **Máme čtyři podmínky dobrého řešení**
    - 1. Žádné dva procesy nesmí být současně uvnitř stejné kritické sekce.
    - 2. Na řešení nesmí mít vliv počet a rychlost CPU.
    - 3. Žádný proces mimo kritickou sekci nesmí blokovat jiný proces.
    - 4. Žádný proces nesmí zůstat čekat nekonečně dlouho na kritickou sekci.

2.2.3 Vzájemné vyloučení s aktivním čekáním (busy waiting)

- několik návrhů pro dosažení vzájemného vyloučení tak, že zatím co jeden proces pracuje na aktualizaci sdílené paměti v kritické sekci, žádný další proces nesmí vstoupit do této kritické sekce a způsobit problémy

Zákaz přerušení

- zákaz všech přerušení procesy, které právě vstoupily do své kritické sekce a jejich znovu-povolení před opuštěním kritické sekce
- **Problémy**:
    - uživatelský proces zakáže přerušení, a nikdy je znovu nepovolí
    - pokud je systém dvou a víceprocesorový, zákaz přerušení se dotkne jen toho procesoru, který instrukci zákazu provede, další procesor bude pokračovat v běhu a může přistoupit do sdílené paměti
- **Závěr je**: zákaz přerušení je užitečná metoda uvnitř samotného operačního systému, ale není použitelná jako obecná metoda vzájemného vyloučení pro uživatelské procesy

Zamykací proměnné (Lock Variables)

- Máme jednu sdílenou zamykací proměnnou inicializovanou na 0
- Když proces požaduje vstup do své kritické sekce, musí nejprve otestovat zámek.
- Když je zámek 0, proces ji nastaví na 1 a vstoupí do kritické sekce
- Pokud je zámek již 1, proces počká, až bude 0.
- **Problémy**:
    - proces přečte zámek a má 0. Než může nastavit zámek na 1, je plánovačem přepnut jiný proces, ten se spustí a také nastaví zámek na 1 a dva procesy se ocitnou v jedné **kritické sekci** současně
- **Závěr je**: nepoužitelný

TSL instrukce (TSL Instruction)

- speciální instrukce **testuj a nastav zámek** (Test And Set Lock)
- přečte hodnotu proměnné do registru a do této proměnné uloží nenulovou hodnotu
- Procesor vykonávající TSL instrukci uzamkne paměťovou sběrnici, aby zakázal dalším procesorům přistupovat do paměti, dokud není hotov
- **Použití**:
    - použijeme sdílenou proměnnou lock, pro sladění přístupu do sdílené paměti
    - Když je lock 0, kterýkoliv proces může změnit hodnotu na 1 použitím TSL
    - Když je lock 1, tak procesy které chtějí přístup do sdílené paměti opakovaně kontrolují jestli už je hodnota lock 0 (busy waiting)
    - Následně může přistupovat do sdílené paměti
    - Když je hotov, proces nastaví lock zpět na 0 použitím instrukce MOVE

```
enter_region:
      TSL register, lock          ; zkopíruje lock do registru
                                  ; a nastaví lock na 1
      CMP register, 0             ; byl lock nulový?
      JNE enter_region            ; pokud nebyl nulový, testujeme znovu
      RET                         ; návrat z podprogramu

leave_region:
      MOV lock, 0                 ; uloží 0 do lock
      RET                         ; ukončení podprogramu
```

2.2.4 Uspání a probuzení (Sleep and Wakeup)

- Nedostatkem správných řešení jako je TSL je aktivní čekání (proces ve smyčce čeká na povolení)
- proces plýtvá procesorovým časem, může také mít neočekávaný efekt
- problém převrácené priority (priority inversion problem)
    - Dva procesy, H s vysokou prioritou a L s nízkou prioritou.
    - Plánovací pravidla říkají, že proces může běžet, pokud je připraven.
    - V určité chvíli je proces L v kritické sekci a H se stane připraveným k běhu.
    - H začne aktivně čekat, ale L nebude nikdy plánován, dokud H běží.
    - L tedy nedostane šanci opustit kritickou sekci a H se zacyklí navždy
- Řešením je mezi-procesní blokování, místo plýtvání procesorovým časem, když není povolen vstup do kritické sekce
    - Systémové volání
        - SLEEP - uspí volající proces, tj. pozastaví dokud jej jiný proces neprobudí
        - WAKEUP - má jediný parametr a sice proces k probuzení
    - Alternativně má SLEEP a WAKEUP jeden parametr - adresu paměti pro uspořádání (spojení) odpovídajících požadavků na uspání a probuzení.

2.2.5 Semafory (Semaphores)

- nový typ proměnné, které říkáme **semafor**
- Semafor může mít hodnotu 0 indikující, že nebylo uloženo žádné probuzení, nebo kladné číslo, indikující počet nezpracovaných probuzení.
- dvě operace:
    - DOWN(Sleep)
        - Kontroluje zda-li je hodnota 0, pokud ano, proces se uspí a operace DOWN se pro tento okamžik neukončí (jestli je hodnota větší než 0, sníží hodnotu a pokračuje)
        - Kontrola proměnné, její modifikace a případné uspání se vždy uskuteční jako jediná **neviditelná nedělitelná akce**.(Je zaručeno, že jednou započatá operace se semaforem nedovolí žádnému jinému procesu k semaforu přistupovat, dokud se operace neukončí)
    - UP(Wakeup)
        - zvýší hodnotu v semaforu
        - Pokud jeden nebo více procesů je uspáno semaforem, bez možnosti ukončit dřívější volání, jeden z nich je vybrán systémem, aby dokončil DOWN (hodnota semaforu stále 0, ale bude o jeden uspaný proces méně)
        - Operace zvýšení hodnoty semaforu a probuzení jednoho procesu je také **neviditelná**
- Další použití semaforu je pro synchronizaci.

Monitory

- Synchronizační prostředek vyšší úrovně
- skupina podprogramů, proměnných a struktur, které jsou zabaleny do speciálního typu modulu, či balíku (dnes asi nejčastěji třídy)
- Proces může volat podprogramy z monitoru kdykoliv potřebuje, ale nikdy nemůže přímo přistupovat k vnitřním datovým strukturám z podprogramů, deklarovaných vně monitoru
- v jakékoliv situaci smí být jen jeden proces aktivní v monitoru
- prvek programovacího jazyka a proto jej překladač rozpozná a zajistí volání jeho podprogramů odlišným způsobem, než pro ostatní podprogramy
    - když proces zavolá podprogram z monitoru, první instrukce podprogramu zkontrolují, zda již není v monitoru aktivní jiný proces
        - Pokud ano, proces se pozastaví, dokud jiný proces monitor neopustí
- úkolem překladače je implementovat vzájemné vyloučení, ale obecnou cestou je binární semafor
- potřebujeme způsob, jak proces pozastavit, pokud je to zapotřebí
    - Řešení spočívá v podmíněné proměnné se dvěma operacemi WAIT a SIGNAL
        - Když některý podprogram monitoru zjistí, že nemůže pokračovat, zavolá WAIT s konkrétní podmíněnou proměnnou - akce pozastaví volající proces a povolí vstup jinému procesu
        - Proces probudí další spící proces pomocí SIGNAL s podmíněnou proměnnou
    - musí být SIGNAL posledním příkazem podprogramu
    - Pokud zavoláme SIGNAL na kterou čeká více procesů, pouze jeden je spuštěn dle rozhodnutí plánovače procesů
    - zaslán signál a nikdo na ni nečeká, signál se ztrácí (Podmíněná proměnná není čítač jako např. Semafor)
- Automatickým vzájemným vyloučením kritické sekce monitorem je paralelní programování podstatně snadnější a odolnější proti chybám, než se semafory
- Monitor je prvek programovacího jazyka. Překladač jej musí rozpoznat a připravit vzájemné vyloučení
- V distribuovaných operačních systémech s více procesory, každý se svou privátní pamětí, spojené pomocí LAN, stanou se tyto principy nepoužitelné

2.2.6 Předávání zpráv (Message Passing) Fronta zpráv

- metoda IPC
- používá dvě operace: SEND a RECEIVE (stejně jako semafory, ale ne jako monitory, implementovány jako systémová volání)
    - SEND
        - zašle zprávu do daného cíle
    - RECEIVE
        - někdo zprávu z daného zdroje vyzvedne
            - není žádná zpráva k dispozici, příjemce se zastaví, dokud nějaká nepřijde
            - Alternativně může ihned skončit s chybovým kódem
- Abychom zabránili ztrátě zprávy, odesílatel a příjemce si potvrdí, že jakmile je zpráva přijata, příjemce odesílá zpět speciální potvrzovací (acknowledgement) zprávu
- Pokud odesílatel neobdrží potvrzovací zprávu v určitém časovém intervalu, tak zprávu pošle znovu
- Pokud příjemce obdrží zprávu nesoucí stejné číslo jako předchozí, tak ví, že jde o zprávu opakovanou a může ji ignorovat (ale musí ji potvrdit).
- Systém musí řešit
    - identifikace procesů(protože volání funkcí SEND a RECEIVE je nejednoznačné)
    - autentizace - jak může klient poznat, že posílá zprávu souborovému serveru a ne podvodníkovi ?
- důležité vlastnosti, pokud je odesílatel a příjemce na jednom počítači **propustnost**
    - Kopírování zprávy z procesu do procesu(pomalejší, než operace se semaforem)
    - Bylo uděláno mnoho práce, aby bylo předávání zpráv efektivní

2.3 Klasické IPC problémy

Večeřící filozofové

- 5 Filozofů, každý má vidličku po levé ruce a talíř špaget před sebou
- Filozof má 3 stavy
    - Thinking,Hungry,Eating
- Aby se Filozof mohl najíst potřebuje jak levou tak pravou vidličku
- Ve stejnou dobu může jíst maximálně 2 Filozofové
- Řešení a jejich uváznutí:
    - snadné řešení, podprogram take fork čeká, dokud se neuvolní vybraná vidlička a pak si ji vezme
        - všech pět filozofů vezme současně levou vidličku. Žádný už nemůže vzít pravou vidličku a dojde k zablokování
    - Můžeme opravit program tak, že po získání levé vidličky se podíváme, jestli je pravá volná. Pokud ne, tak položí levou vidličku zpět na stůl, chvíli počká a pak zkusí vše zopakovat
        - začnou všichni filozofové současně a když zjistí, že druhá vidlička není k dispozici, položí první zpět na stůl, chvíli počkají, vezmou současně levou vidličku a tak pořád do nekonečna
        - Taková situace, kdy všechny programy pracují do nekonečna, ale nejsou schopny cokoliv vyprodukovat, se nazývá **hladovění**
    - co kdyby filozofové čekali náhodný čas ?
        - potřebujeme řešení, které pracuje vždy a nemůže selhat zaviněním série náhodných čísel
    - vylepšení v příkladu bez zablokování a hladovění za volání funkce think vložíme binární semafor. Než filozof vezme ze stolu vidličky, může být zastaven voláním DOWN na mutexu. Jakmile vrátí vidličky, měl by zavolat UP mutexu
        - Z praktického hlediska je zde významná chyba výkonnosti: v kterémkoliv okamžiku může jíst jen jeden filozof
- **Řešení**
    - pole state pro sledování situace (Thinking,Hungry,Eating)
        - Filozof může začít jíst pouze v okamžiku, kdy žádný jeho soused nejí
        - Sousedi filozofa jsou definováni jako makro LEFT a RIGHTS
    - pole semaforů, vždy jeden pro každého filozofa
        - aby se hladovějící filozof mohl zablokovat, při požadavku na vidličky
    - binární semafor pro vstup do kritické sekce

Problém čtenářů a spisovatelů (Readers and Writers Problem)

- Můžeme připustit, aby více procesů současně v jeden okamžik ze souboru četlo, ale pokud jeden proces zapisuje do databáze, žádný jiný proces nesmí do databáze přistupovat, dokonce ani pro čtení
- řešení první čtenář se získáním přístupu do databáze zablokuje pomocí DOWN semaforu db. Ostatní čtenáři už jen inkrementují čítač numr. Jakmile čtenář končí, sníží čítač a poslední odemkne semafor db pomocí UP, čímž povolí případným zablokovaným spisovatelům zapisovat(Pokud přichází čtenáři každé 2 vteřiny, a řekněme, že pracují 5 vteřin, pak se spisovatel nikdy neprobudí)
- Abychom předešli této situaci
    - když přijde čtenář a je zde čekající spisovatel, je čtenář zablokován, místo aby byl okamžitě přijat. Takto čeká spisovatel jen na právě aktivní čtenáře a nemusí čekat na čtenáře, kteří přijdou až po něm
- Nevýhodou řešení je dosažení mírně nižší součinnosti

Spící holič (Sleeping Barber Problem)

- V holičství je jeden holič, jedno holičské křeslo a n židlí pro čekající zákazníky, pokud nějací jsou. Pokud nejsou zákazníci, sedí na křesle holič a spí. Když přijde zákazník, musí probudit spícího holiče. Pokud přijde další zákazník v době, kdy holič stříhá zákazníka, tak se posadí, je-li volná židle, jinak ihned odchází, jsouli židle obsazeny
- Když se ráno objeví holič v práci, provede podprogram barber, který jej přinutí zastavit (a usnout) na semaforu customers, dokud někdo nepřijde.
- Když přijde zákazník, zavolá podprogram customer, začínající uzamčením semaforu mutex před vstupem do kritické sekce. Pokud hned za ním přijde další zákazník, tak bude muset počkat, až se uvolní mutex.
- Zákazník pak zkontroluje, zda je počet čekajících menší, než počet židlí. Pokud ne, uvolní mutex a odchází bez ostříhání.
- Pokud je volná židle, inkrementuje proměnnou waiting a provede UP semaforu customers, aby probudil holiče. V tomto okamžiku jsou probuzeni oba, zákazník i holič.
- Jakmile zákazník uvolní mutex, přivlastní si jej holič, chvíli hospodaří a pak začne stříhat.
- Po ukončení stříhání zákazník ukončí podprogram a odchází z holičství.

2.4 Plánování procesů

- Když je připraveno více procesů ke spuštění, operační systém musí rozhodnout, který spustí jako první
- část operačního systému, která toto rozhodování provádí **plánovač (scheduler)** a použitý algoritmus rozhodování **je plánovací algoritmus (scheduling algorithm)**
- **Kritéria** dobrého plánovacího mechanismu
    - 1. férovost - dát všem procesů rovnocenné šance sdílení CPU,
    - 2. efektivnost - udržet procesor trvale vytížený na 100%,
    - 3. doba odezvy - minimalizovat odezvu pro interaktivní uživatele,
    - 4. doba běhu - minimalizovat čas, po který musí uživatel čekat na vykonání svého zadání (dávky),
    - 5. průchodnost - maximalizovat počet procesů, které budou vykonány během časové jednotky, např. za hodinu.
- Abychom měli jistotu, že proces neběží příliš dlouho, mají téměř všechny počítače vestavěny elektronické hodiny, které vyvolávají pravidelné **přerušení časovače**
    - spustí operační systém a rozhodne, zda aktuální běžící proces smí pokračovat, nebo zda již měl v této chvíli dostatek procesorového času a může být pozastaven, aby mohl předat CPU jinému procesu
- **preemptivní plánování (preemptive scheduling) -** proces schopný běhu dočasně pozastaven (jde o plánování v předstihu, nebo s předstihem opakem)
- **nepreemptivní plánování –** proces který se spustí a běží až do konce (případně se musí sám vzdát kontroly, a nebo se zablokovat a předat kontrolu)
- **pozastaven kdykoliv bez varování, aby mohl běžet proces jiný. To vede k souběhu -** existence semaforů, monitorů, zpráv

Plánovací Algoritmy

- **Round Robin**
  - Každý proces má přiřazen časový interval, nazývaný kvantum (quantum), po který smí běžet
  - Pokud je na konci svého kvanta proces stále běžící, je operačním systémem preemptivně přepnut CPU na jiný proces
  - Když se proces zablokuje nebo ukončí, před vyčerpáním svého časového kvanta, operační systém samozřejmě přepne CPU pro jiný proces. Round robin se snadno implementuje.
  - Když proces využije celé své kvantum, je zařazen na konec seznamu.
  - Přepnutí z jednoho procesu na druhý vyžaduje určité množství času pro režii – uložení a obnovení registrů, mapy paměti, aktualizace některých tabulek, seznamů a pod. Nastavení příliš krátkého časového kvanta vede k příliš častému přepínání procesů a snižuje tak efektivitu práce CPU, ale její neúměrné prodlužování způsobí špatnou odezvu na krátké interaktivní požadavky
- **Prioritní plánování**
  - každý proces má přiřazenu prioritu a spuštěn je spustitelný proces s nejvyšší prioritou
  - Abychom předešli situaci, kdy proces s vysokou prioritou může běžet trvale, plánovač snižuje prioritu aktivního procesu každou časovou periodu. Až se tímto sníží priorita pod hodnotu dalšího procesu v řadě, dojde k přepnutí.
  - Priorita se také může přiřazovat dynamicky, aby bylo dosaženo určitého cíle nebo chování. Jednoduchá metoda pro dobrou obsluhu procesů vázaných na V/V je nastavení priority jako 1/f, kde f je poslední použité časové kvantum. Proces, který použije ze svého kvanta jen 2 ms ze 100 možných, dostane prioritu 50, proces který použije 50 ms, bude mít prioritu 2 a proces, který využil celé své časové kvantum, bude mít prioritu 1
  - Je vhodné slučovat procesy do skupin se stejnou prioritou a používat prioritní plánovaní mezi skupinami, zatím co ve skupině se používá plánování round robin. Pokud nejsou priority nastaveny odpovídajícím způsobem, třídy s nízkou prioritou klidně „umřou hlady“
- **Více front**
  - Řešeni spočívá se stanovení prioritních tříd. Procesy s nejvyšší prioritou mohou běžet jedno časové kvantum. Procesy v další třídě priority mohou běžet dvě kvanta. V další třídě 4 kvanta, atd. Kdykoliv proces využije celé časové kvantum, které mu bylo v dané třídě přiděleno, je přesunut o jednu třídu priority níže
  - Mějme příklad procesu, který potřebuje procesor po dobu 100 kvant. Na počátku dostane jedno kvantum a je odložen. Při dalším spuštění už dostane 2 kvanta před odložením. V následujících bězích dostane 4, 8, 16, 32 a 64 kvant. Z posledního použije jen 37 ze 64 možných a ukončí se. Jenom 7 odložení bylo potřeba během celého běhu programu, na místo 100 odložení, kdyby se použil obyčejný algoritmus round robin. Dále by se mohl proces propadat stále hlouběji a hlouběji v jednotlivých prioritních frontách a bude spouštěn čím dál méně častěji, aby šetřil čas CPU pro interaktivní úlohy

3. Kapitola

3.1, 3.1.1 Principy V/V hardware

Inženýr elektrotechnik se na ně dívá jako na čipy, dráty, napájení, motor a všechny další komponenty tvořící hardware

- Programátory zajímá rozhraní pro programování
    - **Příkazy** které zařízení přijímá
    - **funkce** které vykonává
    - **chyby**, které může vrátit zpět

V/V zařízení

- dvě kategorie:
    - bloková zařízení (block device)
        - ukládá informace v blocích stejné velikosti a každý blok má vlastní adresu. Obvykle je velikost bloku od 512 do 32768 bytů.
        - jsou schopná číst a zapisovat každý blok nezávisle na všech ostatních
        - Nejobvyklejším blokovým zařízením je **disk**
        - **ovladač zařízení (device driver**) - systém komunikuje s abstraktním blokovým zařízením a nechává část závislou na zařízení nižší úrovni software
    - znaková zařízení (character device)
        - předává nebo přijímá proud znaků bez jakékoliv blokové struktury
        - Není adresovatelné a nezná operaci vyhledávání (seek)
        - Tiskárna, síť, myš, sériový port a mnoho dalších

3.2 Principy V/V software

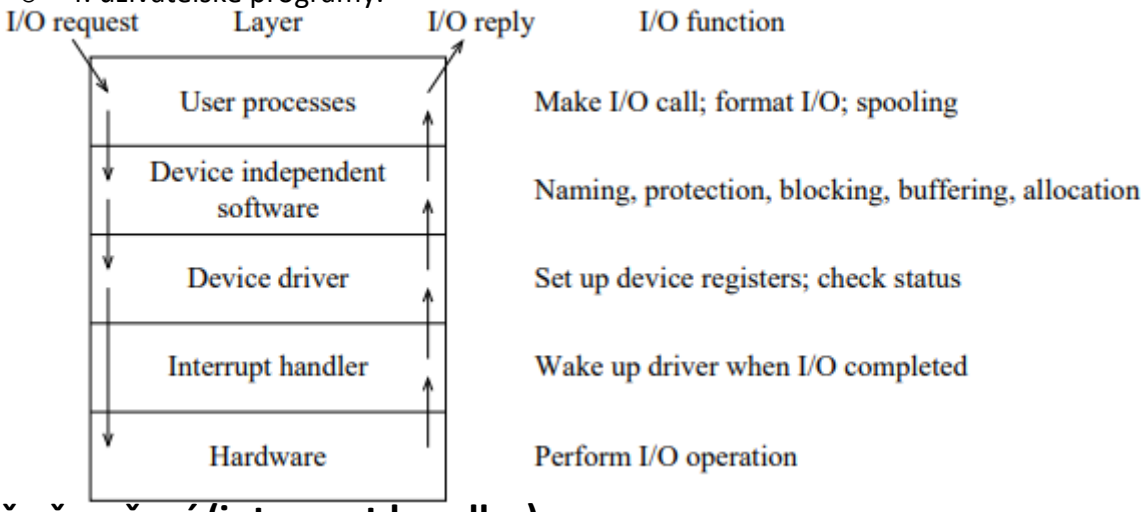
- Základní myšlenka je navrhnout software jako řadu vrstev
- Od nejnižší
    - eliminovat zvláštnosti hardware
- po nejvyšší
    - sloužící jako jednoduché, přehledné a regulérní rozhraní pro uživatele

Cíle V/V software

- **nezávislost na zařízení**
    - je možné napsat program, který může číst soubory z diskety, pevného disku nebo CD-ROMu, bez nutnosti měnit program pro každý typ zařízení
- **jednotnost pojmenovávání**
- **ošetření chyb**
    - musí řešit tak blízko hardware, jak je to jen možné
    - Jen v případě, že nižší vrstvy nejsou schopny si s problémem poradit, měly by o tom informovat vrstvy vyšší
- **synchronní (blokující) kontra asynchronní (přerušením řízené) přenosy**
    - Většina V/V zařízení je asynchronních – CPU spustí přenos a odejde dělat něco jiného, dokud nepřijde přerušení
    - Uživateli se ale daleko snadněji píšou programy, když jsou V/V operace blokující – zavoláním příkazu READ se program automaticky pozastaví, dokud nejdou data v bufferu
- **zařízení sdílená a vyhrazená**
    - otevřeno současně několik souborů nemůže způsobovat problémy
    - například pásky, jsou vyhrazena jednomu uživateli, dokud ten neukončí svou práci
    - operační systém musí být schopen manipulovat se sdílenými i vyhrazenými prostředky tak, aby problémům předcházel

Vrstvy V/V software - rozdělení do vrstev

- Všechny cíle lze dosáhnout srozumitelnou a účelnou formou - strukturováním V/V software do čtyř vrstev:
  - 0. Hardware – vykonávání V/V operací
  - 1. ovladač přerušení,
  - 2. ovladač zařízení,
  - 3. část operačního systému nezávislá na zařízení,
  - 4. uživatelské programy.



Ovladač přerušení (interrupt handler)

- Měly by být schovány hluboko v jádře operačního systému, tak aby o nich věděla co nejmenší část systému
- Funkce je **obudit Ovladač zařízení když je V/V operace hotová**.
- Nejlepší metodou utajení přerušení je pozastavit každý proces, který spustil blok V/V operace, dokud není V/V operace kompletní a dojde k přerušení
- Proces se může sám zablokovat voláním(DOWN,WAIT, RECEIVE…)
- Když dojde k přerušení, obsluha přerušení provede vše co má a odblokuje proces, který operaci odstartoval (UP, SIGNAL, pošle zprávu zablokovanému procesu SEND)

Ovladač zařízení(Device driver)

- část kódu závislá na zařízení
- Každý ovladač zařízení obsluhuje zařízení jednoho typu, nebo podobná zařízení ze stejné skupiny
- Ovladač zařízení **zadává příkazy a kontroluje**, zda byly **správně provedeny**
- řadič má jeden nebo více registrů používaných pro zadávání příkazů
- úkolem ovladače zařízení je přijmout abstraktní požadavky z vrstvy nezávislé na zařízení a dohlédnout, aby byl požadavek proveden
  - První krok je právě přenést V/V požadavek, což v případě disku znamená, přeložit abstraktní požadavek na konkrétní – musí rozhodnout, které operace řadiče jsou nutné a v jakém pořadí
  - Jakmile jsou příkazy, nebo příkaz, zadán řadiči , nastane jedna ze dvou věcí:
    - ovladač zařízení musí čekat, dokud mu řadič neprovede co potřeboval, tak pozastaví sám sebe, dokud nepřijde přerušení a neodblokuje ho
    - operace skončí bez čekání a ovladač zablokování nepotřebuje
    - V prvním případě je zablokovaný ovladač probuzen přerušením. Ve druhém případě nikdy neusne
  - po ukončení operace se musí zkontrolovat chyby
  - Pokud je vše v pořádku, ovladač může předat data vrstvě nezávislé na zařízení
- Fronta nevyřízených požadavků (ukládají se sem požadavky, když je aktuální zařízení zaneprázdněno), některé zařízení umí přijmout seznam příkazů

Část operačního systému nezávislá na zařízení(Device Independent software)

- provádění V/V funkcí, které jsou obecné pro všechna zařízení a nabízí jednotné rozhraní pro uživatelské programy
- Hlavní funkce
    - **jednotné rozhraní pro ovladače zařízení**
    - **pojmenovávání objektů**, jako jsou soubory a V/V zařízení
        - jméno zařízení je jednoznačně specifikováno i–nodem pro speciální soubory
        - i–node obsahuje hlavní číslo zařízení (major device number)
            - které se používá pro identifikaci odpovídajícího ovladače
        - obsahuje také vedlejší číslo zařízení (minor device number)
            - které se používá jako parametr ovladače pro identifikaci jednotky pro čtení nebo zápis
    - **ochrana zařízení**
        - Speciální soubor odpovídající V/V zařízení je chráněn bity rwx
    - **poskytuje velikost bloku nezávisle na zařízení,**
    - **bufferování (vyrovnávací paměť),**
        - bufferuje vstupy, výstupy
    - **alokace úložného prostoru na blokových zařízeních,**
    - **přidělení a uvolnění vyhrazených zařízení,**
        - vyhodnotit požadavek na zařízení a přijmout ho, nebo odmítnout, podle aktuální dostupnosti zařízení
    - **informace o chybách**
        - Když nastane chyba při čtení souboru uživatelem, je vhodné předat chybu volajícímu

Uživatelské programy(User processes)

- Úlohy
    - V/V volání
    - formátování V/V
    - spooling
- Standardní V/V knihovna obsahuje řadu podprogramů, které vyvolávají V/V operace a běží jako součást uživatelského programu
- důležitou kategorií je spooling
    - způsob manipulace s vyhrazenými zařízeními v multiprogramovém systému
    - typické zařízení je tiskárna
        - vytvoříme speciální proces nazývaný démon (daemon)
        - a dále speciální adresář pro souběžný tisk (spooling directory)
        - Abychom vytiskli soubor, musí proces nejprve vytvořit soubor pro tisk a vložit jej do adresáře souběžný tisk
        - Démon jakožto jediný proces majícího právo přistupovat ke speciálnímu zařízení, tiskne soubory z adresáře
        - ochranou speciálního souboru před přímým přístupem uživatelů eliminujeme možnost, že by si někdo mohl soubor otevřít zbytečně dlouho
    - není jen pro tiskárny, přenosy souborů po síti (taky používá démona)
        - Pro poslání souboru jej uživatel umístí opět do síťového adresáře
        - Časem si jej démon vyzvedne a pošle
        - Speciálním případem takového posílání souborů je elektronická pošta

Typický V/V požadavek

- Uživatelský program se pokouší číst blok ze souboru, OS je vyzvány systémovým voláním.
- Nezávislá vrstva se podívá zda blok není ve vyrovnávací paměti, ne-li volá ovladač zařízení aby předal požadavek na hardware
- proces je zablokován dokud není disková operace kompletní
- když disk skončí, hardware generuje přerušení
- ovladač přerušení zjistí co se stalo, které zařízení potřebuje pozornost, převezme status ze zařízení a probudí uspaný proces, aby dokončil V/V požadavek a proces mohl pokračovat

3.3 Deadlock

- každý systém má možnost dočasně zaručit výhradní přístup procesu k určitému prostředku
- K zablokování může dojít v mnoha případech, kromě požadavků na výhradní přístup
- každý zablokovaný proces čeká na **prostředky** vlastněné jiným zablokovaným procesem, žádný nemůže běžet a tedy uvolnit své **prostředky**, žádný nemůže být probuzen
    - **prostředkem** je cokoliv, co může být použito jedním procesem v kterémkoliv okamžiku
        - prostředky dělíme na **Odebíratelné prostředky (preemptable resources)**
            - jsou takové, jejichž odebrání procesu, který je vlastní, nebude nijak bolestivé
        - **neodebíratelné zdroje (nonpreemptable resources)**
            - nemohou být jeho vlastníkovi odebrány bez toho, že to způsobí chybu
- Obecně zablokování způsobují neodebíratelné prostředky

Podmínky pro zablokování

- Aby mohlo dojít k zablokování, musí být splněny 4 podmínky
    - 1. **Podmínka vzájemného vyloučení**.
        - Každý prostředek je buď právě vlastněn jedním procesem, nebo je dosažitelný.
    - 2. **Podmínka postupného přidělování prostředků**.
        - Proces který již vlastní dříve přidělené prostředky, může požádat o další.
    - 3. **Podmínka odebrání**.
        - Dříve přidělené prostředky nemohou být procesu násilím odebrány. Musí být uvolněny procesem, který je vlastní.
    - 4. **Podmínka čekání v kruhu**.
        - Musí se vytvořit zřetězený kruh dvou nebo více procesů, kde každý čeká na prostředek vlastněný následujícím procesem v kruhu

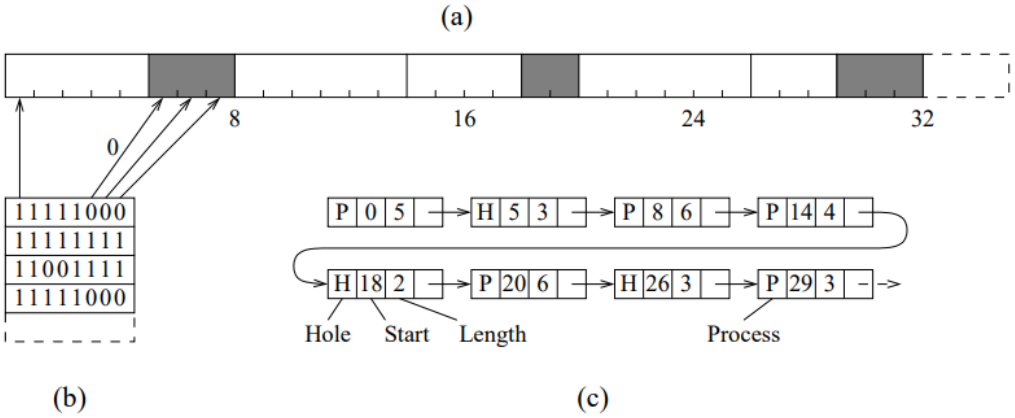
Řešení zablokování

- K řešení zablokování se dají použít čtyři strategie:
    - Ignorovat problém. (Pštrosí algoritmus)
        - ignorovat problém s předpokladem, že většina uživatelů upřednostní příležitostné zablokování před omezením všech uživatelů na jeden proces, jeden otevřený soubor a jeden kus ze všeho dostupného
    - Detekce a zotavení.
        - eviduje požadavky a uvolňování prostředků
        - Vždy, když je prostředek vyžádán a uvolněn, je aktualizován graf prostředků a je provedena kontrola, jestli se v něm neobjevil cyklus, pokud ano jeden proces z cyklu je zrušen (killed), pokud ne je zrušen další proces, dokud se smyčka neodstraní
        - Surovější metoda je – je-li proces zablokován déle než hodinu, je ukončen
        - Jen je nutno se věnovat pečlivému uvedení modifikovaných souborů do původního stavu, a všemu, kde došlo ke změnám
    - Dynamické zamezení pečlivou alokací prostředků. (Předcházení zablokování)
        - bankéřův algoritmus
    - Prevencí, kdy vhodným návrhem vyloučíme jednu ze čtyř podmínek zablokování.

Podmínka	Přístup
Vzájemné vyloučení	Vše přes spooling
Postupné přidělování prostředků	Vyžádat všechny prostředky předem
Neodebírání	Odebrání prostředků
Čekání v kruhu	Číselné pořadí prostředků

Přiřazuje-li se paměť dynamicky, musí jí operační systém spravovat

4.2.1 Správa paměti bitmapami (Memory Management with Bit Maps)



Obrázek 5: (a) Příklad paměti s pěti procesy a třemi mezerami (šedá barva). (b) Odpovídající bitmapa. (c) Stejná informace jako seznam.

Obecně existují dva způsoby, jak sledovat využití paměti: **bitmapami** (bit maps) a **seznamem** (free lists)

- U bitmapy je paměť rozdělena na alokační jednotky (allocation units)
  - velikost od několika slov, až po několik kilobajtů
  - Zvolíme-li velikost alokační jednotky větší, bitmapa bude menší, ale značná část paměti může být ztracena v poslední jednotce, pokud velikost procesu není přesný násobek alokační jednotky
- Ke každé alokační jednotce přísluší jeden bit v bitmapě
  - 0 znamená, že jednotka je volná
  - 1, když je obsazená (nebo naopak)
- Velikost bitmapy záleží jen na velikosti paměti a velikosti alokační jednotky
- **Hlavní problém**
  - rozhodneme se načíst proces délky *k* jednotek do paměti
  - **správce paměti musí najít v bitmapě celistvý úsek nul délky *k***
    - **jeto pomalá operace (protože sekvence může být rozložena mezi slovy v bitmapě)**
  - **Toto je také hlavní argument proti používání bitmap**

4.2.2 Správa paměti s propojenými seznamy (Memory management with linked lists)

- udržování **propojeného seznamu alokovaných a volných paměťových segmentů**
  - **segment** může být buď proces nebo mezera mezi procesy
- Každý záznam v seznamu specifikuje
  - mezeru (hole- H), nebo proces (P)
  - počáteční adresu,
  - délku
  - ukazatel na další záznam
  - (vhodné mít i ukazatel na předchozí – snazší slučování mezer)
- Ukončený proces má normálně dva **sousedy** (kromě případů kdy je na úplném začátku nebo konci paměti)
  - Soused může být proces nebo mezera
  - Nově vzniklá mezera po ukončeném procesu se sloučí se sousední mezerou/mezerami
- Máme-li seřazeny procesy a mezery podle adresy, můžeme použít několik algoritmů pro alokaci paměti pro nově vzniklé, nebo odložené procesy.
- **Algoritmy**(předpokládáme, že správce paměti ví, kolik paměti má alokovat)
  - **první vhodný (first fit)**
    - Správce paměti prochází seznam, dokud nenajde dostatečně velkou mezeru
    - Mezera je poté rozdělena na dvě části
      - jedna část pro proces
      - druhá bude nevyužita(kromě případů, kdy se proces přesně vejde do mezery)
    - **rychlý**, protože prohledává jen do prvního úspěchu

- **další vhodný (next fit)**
  - Funguje stejně jako první vhodný s tím rozdílem
    - uchovává záznam o poloze použité mezery
    - při příštím hledání pokračuje od místa, kde naposled skončil
  - **trošku horší** výsledky než první vhodný
- **nejlepší správný (best fit)**
  - prohledává celý seznam a vybírá nejmenší vhodnou mezeru (najít mezeru velikosti nejbližší požadavku)
  - je pomalejší než první vhodný (protože musí prohledat celý seznam) taky vede k většímu plýtvání pamětí(protože má sklon rozmělnit paměť na malé mezery)
- **nejhůře vyhovujícím (worst fit)**
  - vždy vybere největší mezeru, která po rozbití bude dostatečně velká aby byla ještě použitelná
  - není nejlepším řešením

- Všechny čtyři algoritmy mohou být urychleny udržováním oddělených seznamů procesů a mezer(věnují energii prohledáváním mezer a ne procesů)
  - zvýšená složitost
  - zpomalení při dealokaci paměti(protože uvolněný segment musí být odstraněn ze seznamu procesů a vložen do seznamu mezer)
  - Je-li seznam mezer oddělen od seznamu procesů, máme možnost drobné **optimalizace**
    - Prvním slovem každé mezery by byla velikost mezery
    - druhým slovem ukazatel na následující položku
- Algoritmus **rychlý vhodný (quick fit)**
  - **udržuje si oddělené seznamy pro některé často používané velikosti**
    - například tabulku n položek, ve které první záznam ukazuje na hlavičku seznamu mezer velikosti 4 kB, druhý záznam ukazuje na seznam mezer velikostí 8 kB, třetí..
    - mezery velikosti řekněme 21 kB mohou být buď zařazeny do seznamu 20 kB mezer nebo na zvláštní seznam mezer liché velikosti
  - **extrémně rychlé**
  - **nevýhody**
    - když proces skončí nebo je odložen, nalezení jeho sousedů a zjištění zda se vůbec můžou spojit, je časově náročné
    - Pokud nedojde ke spojení, tak se paměť velmi rychle fragmentuje na malé mezery, do kterých se již žádný proces nevejde

4.3 Virtuální paměť (Virtual Memory)

Před mnoha lety byli návrháři poprvé postaveni před problém, kdy programy byly příliš velké na to, aby se vešly do dostupné paměti. Řešení, které se obvykle užívalo, bylo rozdělit program na několik částí, nazývaných překryvnými moduly (overlays)

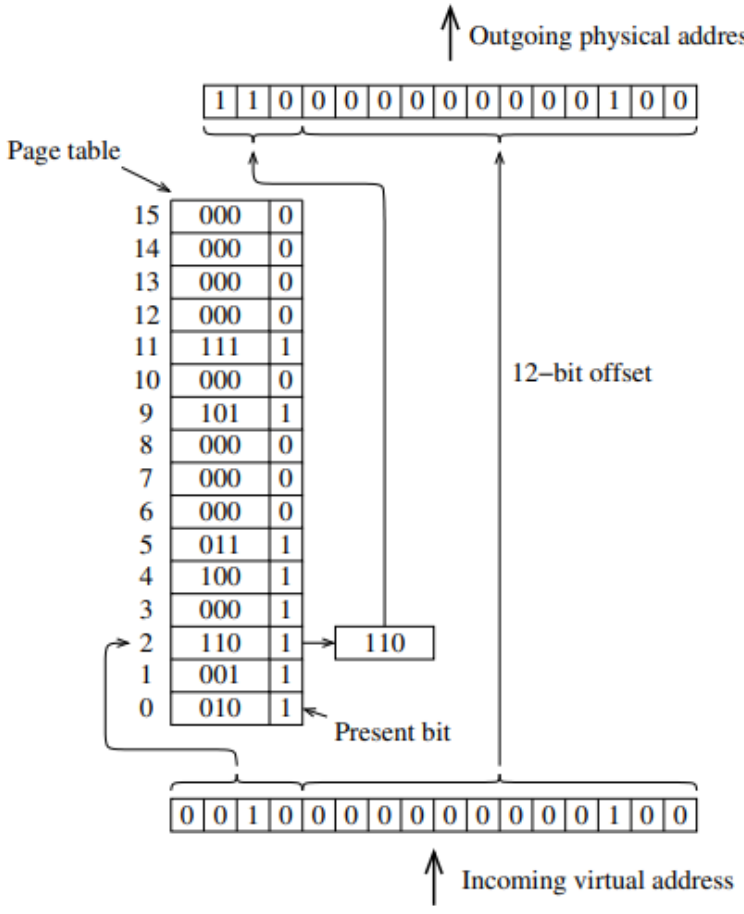
- Celková velikost programů, dat a zásobníků může překročit velikost dostupné fyzické paměti
  Hlavní myšlenka
- odstraňuje obavy z uspořádání programu do vrstev
- Operační systém uchovává v hlavní paměti jen ty části, které jsou právě používány, a zbytek má na disku
- Např. 16MB program může běžet na stroji s 4MB paměti(vybírá, které 4MB části programu uchovávat v paměti a které části má odkládat mezi pamětí a diskem)
- U víceprocesových systémů VP může odkládat částí programu na disk při čekání na V/V požadavek

4.3.1 Stránkování (paging)

Většina systémů virtuální paměti používá **stránkování**

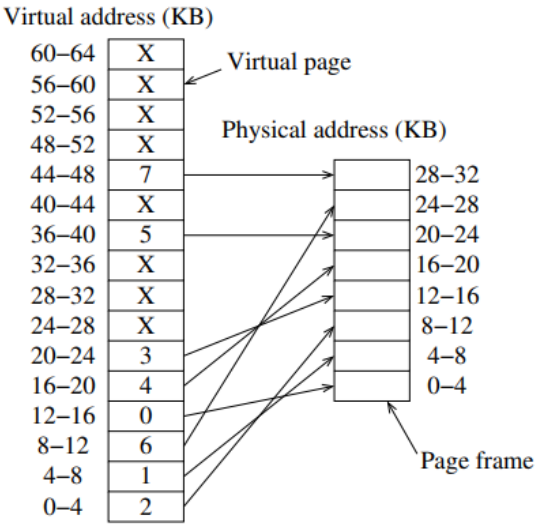
- Na každém počítači existují soubory adres paměti, které můžou programy vytvářet
- Adresy mohou být vytvářeny pomocí
  - Indexových registrů(indexing)
  - Bázových registrů (base)
  - Segmentových registrů (segment)
  - nebo jinými způsoby
- Tyto programově vygenerované adresy se nazývají **virtuálními adresami** a vytváří **virtuální adresový prostor** (virtual address space)
- Na počítačích **bez virtuální paměti**, se virtuální adresy posílají **přímo na paměťovou sběrnici**, což způsobí, že **se do fyzické paměti zapíše nebo přečte slovo** z dané adresy
- **Používá-li se virtuální paměť**, pak virtuální adresa **nejde přímo na paměťovou sběrnici**
- Místo toho jde do **jednotky správy paměti** (Memory Management Unit - MMU)
  - mapují virtuální adresy na fyzické adresy
  - při přesáhnutí velikosti paměti velikostí programu musí být jádro programu zkopírováno někde na disku tak, aby mohly být jednotlivé části nahrány, až bude potřeba
- Virtuální adresní prostor se dělí na několik jednotek zvaných **stránky** (pages)
- Odpovídající jednotky ve fyzické paměti se nazývají **rámce stránek** (page frames)
- **Stránky a rámce stránek jsou vždy stejné velikosti**
  - současných systémech používají stránky velikosti od 512 bytů do 64 kB
- Přenosy mezi pamětí a diskem jsou vždy realizovány po stránkách
  - (4kB stránky)virtuální adresa 0 pošle do jednotky MMU. Jednotka MMU vidí, že virtuální adresa spadá pod stránku 0 (0-4095), která odpovídá podle odpovídajícího mapování rámci stránky 2 (8192-12287)
    - Takto transformuje adresu na 8192 a pošle adresu 8192 na sběrnici
- Ve skutečném hardwaru je **ke každému záznamu** přiřazen **informující bit** (present/absent bit)
  - **přítomnosti** mapování dané stránky
  - **nepřítomnosti** mapování dané stránky
- Když jednotka MMU zjistí že stránka není namapována, oznámí toto CPU, to potom předá informaci operačnímu systému
  - Tento odchyt se nazývá **výpadek stránky** (page fault)
  - Operační systém vezme nejméně používaný rámec stránky a zapíše jeho obsah zpátky na disk. Poté přenese právě odkazovanou stránku do právě uvolněného rámce stránky, změní mapu a spustí znova odchycenou instrukci

- Jak Funguje MMU ?
  - přichází adresa (např. 16bitová) a je rozdělena na:
    - 4bitové číslo stránky (můžeme reprezentovat 16 stránek)
      - Použito jako ukazatel do **tabulky stránek (page table)**
        - převádí číslo rámce stránky, podle odpovídající virtuální stránky
        - Je-li bit přítomnosti stránky nastaven na nulu, pak je toto odchyceno operačním systémem
        - Je-li tam nastavena jednička, pak je číslo rámce stránky nacházející se v tabulce stránek zkopírováno do horních 3 bitů výstupního registru společně s 12-bitových offsetem, který je beze změny zkopírován z příchozí virtuální adresy
        - vytvářejí 15-ti bitovou fyzickou adresu
          - Hodnota výstupního registru je poté poslána na paměťovou sběrnici, jako fyzická adresa paměti
    - 12-bitový offset (adresovat 4096 bytů v rámci stránky)



Obrázek 9: Vnitřní operace MMU s $16 \times 4kB$ stránkami

4.3.2 Tabulky stránek (page tables)



Obrázek 8: Vztah mezi virtuální adresou a fyzickou pamětí je dán tabulkou stránek.

Virtuální adresa je rozdělena na číslo virtuální stránky (vyšší bity) a offset (nižší bity). Číslo virtuální stránky se použije jako ukazatel do tabulky stránek, aby se nalezl záznam virtuální stránky
Číslo rámce stránky je připojeno za konec (vyšší pozice) offsetu, nahrazuje tak číslo virtuální stránky a vytváří fyzickou adresu, která může být poslána do paměti

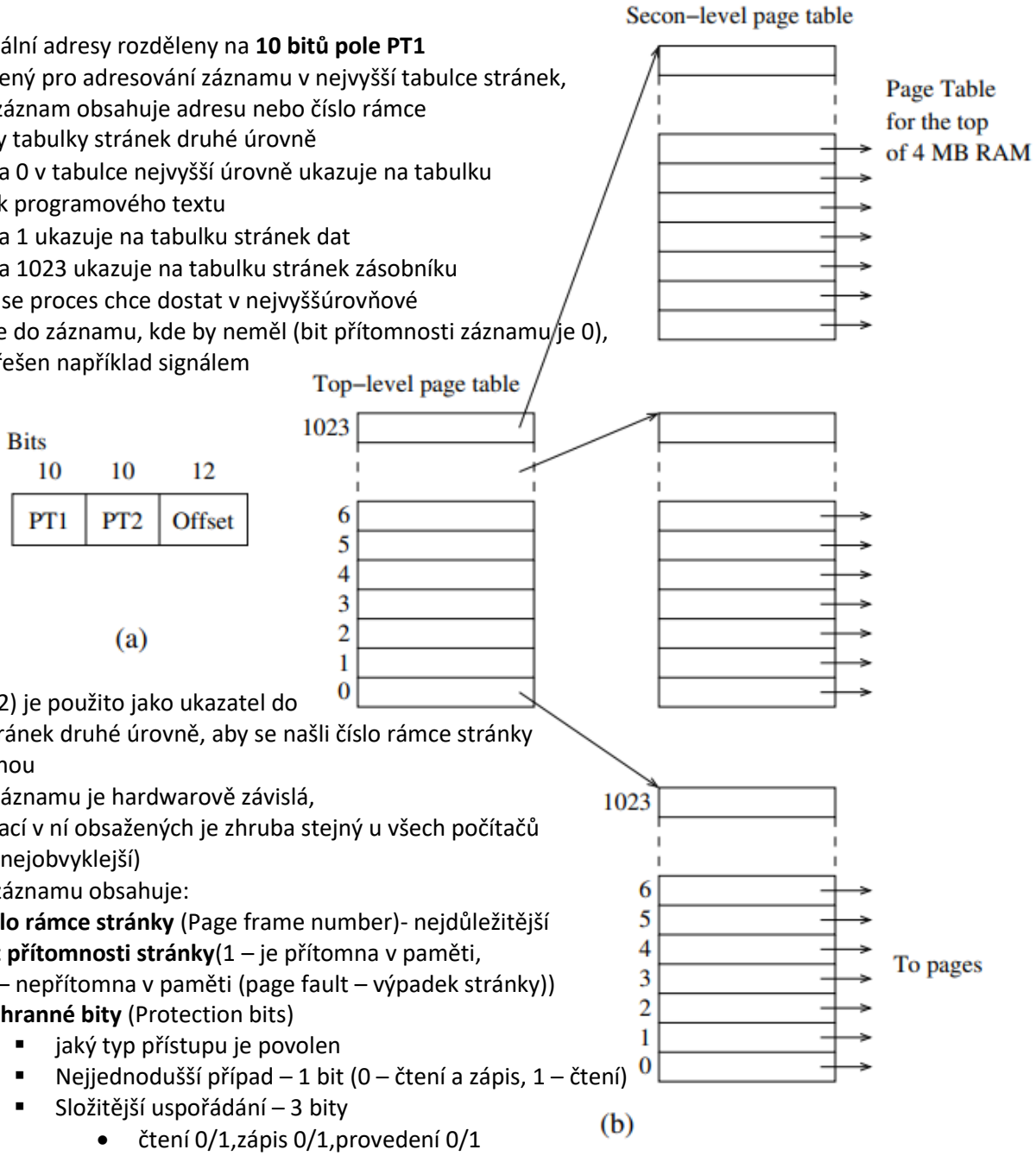
- Účelem je, **mapovat virtuální stránky na rámce stránek**
- Řečeno matematicky, tabulka stránek je funkce, která má jako argument číslo virtuální stránky a výsledkem je číslo fyzického rámce. Užitím výsledku této funkce, může být nahrazeno pole virtuální stránky ve virtuální adrese za pole rámce, což vytvoří fyzickou adresu paměti.
- Navzdory jednoduchému popisu se můžeme potýkat s dvěma podstatnými problémy:
    - 1. Tabulka stránek může být extrémně velká.
        - moderní počítače užívají minimálně 32-bitové virtuální adresy
            - stránky velikosti 4 kB a 32-bitovým adresným prostorem = 10^6
                - s 64 bitovým adresním prostorem 1 milión položek
        - a k tomu každý proces má vlastní tabulku stránek
    - 2. Mapování musí být rychlé.
        - mapování virtuální adresy na fyzickou adresu musí být prováděno při každém přístupu do paměti
- Nejjednodušší návrh (co do koncepčnosti)
    - mít jednu tabulku stránek
        - skládající se z pole rychlých hardwarových registrů s jednou položkou pro každou virtuální stránku
        - které jsou seřazeny podle čísla virtuální stránky
        - Když je proces spuštěn, operační systém naplní registry tabulkou stránek procesu, která je převzata z kopie uchovávané v hlavní paměti
        - V průběhu procesu nejsou třeba žádné další odkazy do paměti pro tabulku stránek
        - **Nevýhodou** je, že může být potenciálně nákladná (pokud jsou tabulky stránek velké)
        - Povinnost nahrát tabulku stránek v každém přepnutí kontextu - může také negativně ovlivnit výkon
- Jiným extrémem je
    - tabulka stránek celá v hlavní paměti, jeden registr ukazující na začátek tabulky stránek
        - umožňuje přepnout kontext mapy paměti pouhou změnou jednoho registru
        - **nevýhody** potřeba jednoho nebo více odkazů do paměti pro načtení záznamů tabulky stránky během vykonávání každé instrukce
    - Z tohoto důvodu je tento přístup jen zřídka používán

Víceúrovňové tabulky stránek (Multilevel Page Tables)

- Abychom se vyhnuli problému nutnosti držet po celou dobu velké tabulky stránek v paměť
- předcházet tomu, aby nebyly po celou dobu všechny tabulky v paměti (zvláště ty, které nejsou potřeba)
- Na obrázku vidíme, jak na tomto příkladu funguje dvouúrovňová tabulka stránek

32-bitové virtuální adresy rozděleny na **10 bitů pole PT1**

- vyhrazený pro adresování záznamu v nejvyšší tabulce stránek, tento záznam obsahuje adresu nebo číslo rámce stránky tabulky stránek druhé úrovně
- Položka 0 v tabulce nejvyšší úrovně ukazuje na tabulku
- stránek programového textu
- Položka 1 ukazuje na tabulku stránek dat
- Položka 1023 ukazuje na tabulku stránek zásobníku
- Pokud se proces chce dostat v nejvyššúrovňové tabulce do záznamu, kde by neměl (bit přítomnosti záznamu je 0), tak je řešen například signálem



Dalších 10 bitů (PT2) je použito jako ukazatel do vybrané tabulky stránek druhé úrovně, aby se našli číslo rámce stránky pro stránku samotnou

- struktura záznamu je hardwarově závislá, typ informací v ní obsažených je zhruba stejný u všech počítačů (32 bitů je nejobvyklejší)
- Struktura záznamu obsahuje:
  - **číslo rámce stránky** (Page frame number)- nejdůležitější
  - **bit přítomnosti stránky**(1 – je přítomna v paměti, 0 – nepřítomna v paměti (page fault – výpadek stránky))
  - **Ochranné bity** (Protection bits)
    - jaký typ přístupu je povolen
    - Nejjednodušší případ – 1 bit (0 – čtení a zápis, 1 – čtení)
    - Složitější uspořádání – 3 bity
      - čtení 0/1,zápis 0/1,provedení 0/1

Obrázek 10: (a) 32 bitová adresa se dvěma indexy do tabulek stránek. (b) Dvoúrovňové stránkování.

  - **Bit modifikace (Modified)**
    - Je-li proveden zápis do stránky, hardware automaticky nastaví bit modifikace
    - význam, když stránka musí být zapsána zpět na disk
      - pokud nebyla modifikovaná, může se zrušit (protože její kopie na disku je stále platná)
  - **Bit odkazu (Referenced)**
    - se nastavuje, kdykoliv je na stránku odkázáno, buď při čtení nebo zápisu
    - pomáhá operačnímu systému rozhodnout, kterou stránku vyklidit když dojde k výpadku stránky
  - **Bit umožňující vypnout rychlou vyrovnávací paměť(caching)**
    - vlastnost je důležitější pro stránky mapující zařízení na registry, než pro paměť

        - Čeká-li operační systém ve smyčce na odpověď příkazu zaslaného nějakému V/V zařízení, je nezbytné, aby hardware načítal slovo ze zařízení a nepoužíval starou kopii ve vyrovnávací paměti
        - Stroje, které mají oddělený V/V prostor a nepoužívají pamětí mapovaný V/V, tento bit nepotřebují
- diskové adresy užívané k ukládání stránek, když není místo v paměti, nejsou součástí tabulky stránek
    - Informace, které potřebuje operační systém k obsloužení výpadku stránky, jsou uchovávány v softwarových tabulkách v rámci operačního systému

4.4 Algoritmy výměny stránky

- Když se objeví chyba stránky, operační systém musí vybrat stránku k vyjmutí z paměti, aby vytvořil prostor pro stránku, která musí být vložena
    - Jestli nebyla pozměněna (například stránka obsahuje kód programu), disková kopie je tedy aktuální a žádný přepis není nutný

Optimální algoritmus náhrady stránky

- lehké popsat, ale nemožné naimplementovat
- Ve chvíli, kdy se objeví chyba stránky, je v paměti určitý soubor stránek. Jedna z těchto stránek bude odkazována na nejbližší instrukci (stránku obsahující tuto instrukci). Ostatní stránky mohou být odkazovány až o 10, 100, nebo možná 1000 instrukcí později. Každá stránka může být označena počtem instrukcí, které budou vykonány, než bude stránka poprvé odkazována
    - stránka s nejvyšším označením by měla být vyjmuta
- Ve zkratce vyjímáme stránku na kterou se bude odkazovat nejpozději, o co nejvíce instrukcí později
    - nemožnost toto vědět

Náhrada dříve nepoužité stránky (Not Recently Used -NRU)

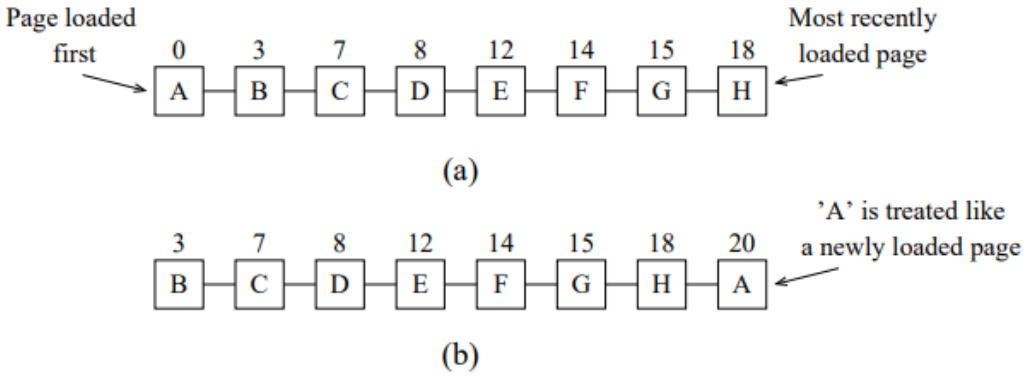
- Využívání bitu modifikace (M) a bitu odkazu (R)
- Pokud hardware nedisponuje těmito bity, mohou být simulovány (v interních tabulkách)
    - proces zahájen, všechny položky stránkové tabulky jsou označeny jako mimopaměťové. Jakmile je některá stránka odkazována, objeví se chyba stránky. OS poté nastaví R bit , změní položku tabulky stránky, aby ukazovala na správnou stránku s módem READ ONLY a znovu spustí instrukci. Jestliže je stránka následně modifikována nastaví se v interních tabulkách OS M bit a změnit mód stránky na READ/WRITE
- **Algoritmus**
    - proces je zahájen, oba stránkové bity pro všechny jeho stránky jsou operačním systémem nastaveny na 0
    - Pravidelně (např. na každé hodinové přerušení), je R bit vymazán, aby se odlišily stránky, které dosud nebyly odkazovány od těch, které již byly
    - Pokud se objeví chyba stránky, OS projde všechny stránky a rozdělí je do čtyř kategorií, podle R a M bitů:
        - Třída 0: neodkazovány, nepozměněny
        - Třída 1: neodkazovány, pozměněny
        - Třída 2: odkazovány, nepozměněny
        - Třída 3: odkazovány, pozměněny
    - algoritmus vyjímá stránku náhodně z neprázdné třídy s nejnižším číslem
    - snadná pochopitelnost, efektivní implementace a výkon, který jistě není optimální, ale často dostatečný

FIFO (First-In, First-Out) algoritmus náhrady stránky

- Operační systém si udržuje seznam všech stránek, které se momentálně nacházejí v paměti
- V čele seznamu je nejstarší stránka
- na jeho konci je pak stránka, která byla umístěna do paměti poslední
- Při chybě stránky je vyjmuta stránka z čela seznamu a na konec seznamu je přidána nová stránka

Náhrady stránky s druhou šancí (The Second Chance)

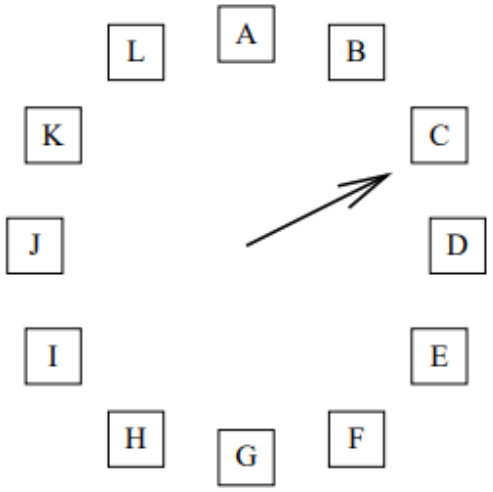
- nulování R bitu každý hodinový cyklus
- Jednoduchá modifikace FIFO pomocí ošetření R bitu nejstarší stránky
    - R bit roven 0 - stránka je stará a nepoužívaná
        - Stránka je pri výpadku stránky **okamžitě odstraněna**
    - R bit roven 1 - používaná stránka
        - Stránka je pri výpadku stránky **umístěna na konec seznamu R je vynulované**
- Princip druhé šance je hledání staré stránky, která nebyla odkazována v předchozím hodinovém intervalu. Jestliže byly odkazovány všechny stránky, druhá šance degeneruje v čistou metodu FIFO
- Nakonec opět dojdeme ke stránce, která již má nyní svůj R bit vymazán a je tedy odstraněna



Obrázek 12: Princip druhé šance. (a) Stránky v pořadí FIFO. (b) Pokud v čase 20 dojde k výpadku stránky, je A přesunuta nakonec seznamu a vynulován R bit.

Hodinový algoritmus náhrady stránky

- přístup je udržovat všechny stránky v kruhovém seznamu ve tvaru hodin
- Když se objeví chyba stránky, je vybrána stránka, na kterou ukazuje ručička
    - Jestliže je její R bit 0, stránka je odstraněna a na její místo je vložena stránka nová
    - Pakliže je R bit 1,je vynulován
- Ručička se posune o jednu pozici
- Od druhé šance se liší pouze implementací

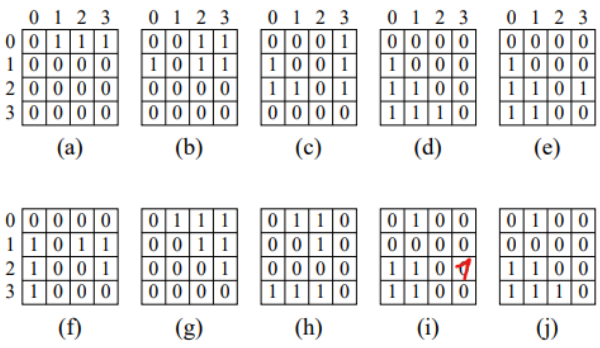


When a page fault occurs, the page the hand is pointing to is inpected.
The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

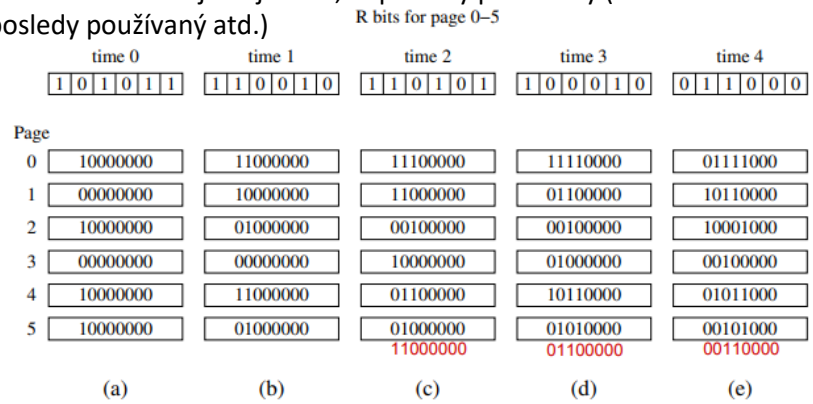
Obrázek 13: Hodinový algoritmus výměny stránek.

Náhrada nejdéle nepoužívané stránky (LRU - The Least Recently Used)

- pozorování, že stránky hojně využívané v několika posledních instrukcích budou pravděpodobně hojně využívány i v několika příštích. Naopak stránky již dlouho nepoužívané, zůstanou nejspíše nepoužité dlouho i nadále
- když se objeví chyba, vyžadí se stránka, která nebyla používaná nejdelší čas
- udržovat provázaný seznam všech stránek v paměti s nejdávněji používanou stránkou vepředu a s naposledy používanou vzadu
- Obtížné je, že seznam musí být aktualizován při každém paměťovém odkazu
    - Vyhledání stránky v seznamu, její smazání a následný přesun do čela seznamu
        - je dlouhá operace, spolyká mnoho času
- naimplementovat LRU se použitím speciálního hardware
    - 64 bitovým čítačem C, který je automaticky inkrementován po každé instrukci
    - každá položka tabulky stránky musí mít dostatečně velké pole k pojmutí čítače
    - Po každém paměťovém odkazu je současná hodnota C uložena v položce tabulky stránky pro právě odkazovanou stránku
    - Když se objeví chyba stránky, operační systém vyzkouší všechny čítače v tabulce stánky, aby našel ten s nejnižším údajem
    - Tato stránka je naposledy používaná
- druhý hardwarový LRU algoritmus
    - Pro stroj s n stránkovými rámci umí LRU hardware udržovat matici o n × n bitech, zpočátku všech nulových
    - Kdykoliv je stránkový rámec k odkazován, hardware nejdříve nastaví všechny bity řádku k na 1, poté nastav všechny bity sloupce k na 0
    - V každém okamžiku je řádek, jehož binární hodnota je nejmenší, naposledy používaný (řádek s 2 nejmenší binární hodnotou je 2 naposledy používaný atd.)



Obrázek 14: LRU s použitím matic.

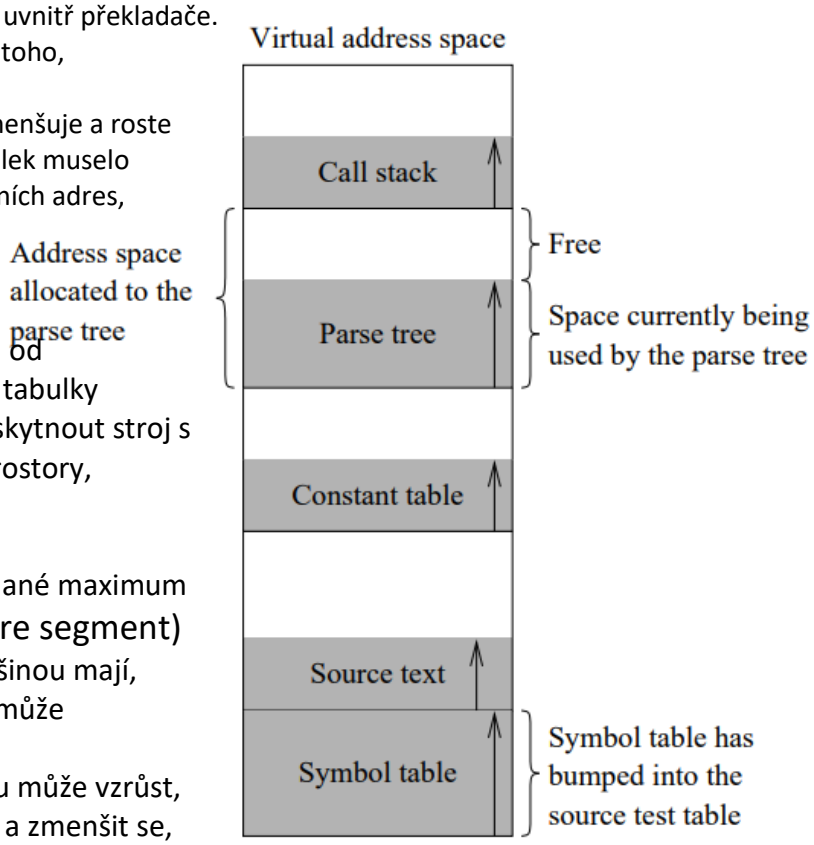


Obrázek 15: Programová simulace LRU. Šest stránek v pěti hodinových cyklech (a) až (e).

Programová simulace LRU

- LRU algoritmy v principu realizovatelné, jen málo strojů, pokud vůbec nějaké, disponuje tímto hardwarem
- Algoritmus **málo používaný (Not Frequently Used - NFU)**
    - Programový čítač asociovaný s každou stránkou na počátku nastavený na 0
    - Při každém hodinovém přerušení prohlédne OS všechny stránky v paměti a pro každou stránku je R bit(který může být 0 nebo 1) ,přičten do čítače
    - V podstatě jsou čítače jakousi snahou o udržování záznamu o tom, jak často byla každá stránka odkazována
    - Když se objeví chyba stránky, je k výměně vybrána stránka s nejnižším čítačem.
- **Problém NFU**
    - **nikdy nezapomene všechno**, čili pokud nějaká stránka byla velmi často používána na začátku programu a pak vůbec, tak se může stát, že zůstane brána jako často používaná až do konce a aktuálně potřebné stránky jsou odstraňovány
- **Uprava umožní simulovat NFU dobře (stárnutí)**
    - V první části jsou všechny čítače posunuty doprava o jeden bit dříve, nežli je přičten R bit
    - V druhé části je R bit přičten raději do nejlevějšího, než do nejpravějšího bitu
- Rozdíly NFU se stárnutím oproti LRU
    - v některých případech neumíme rozhodnout která stránka s nejnižší hodnotou čítače byla nejdříve použita, protože jich více může mít hodnotu 0
    - čítač máme na 8 bitech, tedy nejméně používané stránky je problém časově rozlišit

- Virtuální paměť je jednodimenzionální, virtuální adresy jdou od 0 po maximální adresu, jedna za druhou
- Při mnoha problémech by bylo mnohem lepší mít 2 nebo více rozdílných virtuálních adresových prostorů nežli pouze jeden
- Překladač má mnoho tabulek, které jsou vytvořeny jako výsledek překladu a nejspíše obsahují:
    - 1.zdrojový text uložený kvůli tištěnému výpisu (u dávkových systémů),
    - 2. tabulku symbolů, obsahující jména a atributy proměnných,
    - 3. tabulku obsahující všechny použité konstanty typu integer a s plovoucí čárkou,
    - 4. rozkladový strom, obsahující syntaktickou analýzu programu,
    - 5. zásobník, použitý pro volání procedur uvnitř překladače.
- První 4 tabulky se vytváří a rostou plynule podle toho, jak postupuje překlad
- Poslední se během překladu nepředvídatelně zmenšuje a roste
- V jednodimenzionální paměti by těchto pět tabulek muselo být alokováno na sousední části prostoru virtuálních adres, jak je tomu na obrázku
- Řešením je Segmentace



Obrázek 18: …

4.6 Segmentace

- Účel segmentace je osvobodit programátora od nutnosti spravovat zvětšující a zmenšující se tabulky
- Přímým a extrémně obecným řešením je poskytnout stroj s mnoha absolutně nezávislými adresovými prostory, zvanými **segmenty**
- **Každý segment se skládá:**
    - lineární posloupnosti adres od 0 po dané maximum (rozsah 0-povolené maximum pre segment)
    - Rozdílné segmenty mohou mít, a většinou mají, **rozdílnou délku** (délka segmentu se může měnit během vykonávání)
      Např. Délka zásobníkového segmentu může vzrůst, kdykoliv je něco uloženo na zásobník a zmenšit se, když je něco ze zásobníku vyjmuto.
    - každý segment představuje samostatný adresový prostor (segmenty se mezi sebou neovlivňují)
    - **Zdůrazňujeme**, že segment je logická entita
    - Adresování - tvořenou dvěmi částmi
        - číslem segmentu
        - adresou uvnitř segmentu
    - může obsahovat
        - podprogram
        - pole
        - zásobník
        - soubor skalárních proměnných, ale většinou neobsahuje směsici rozdílných typů
    - spojení odděleně kompilovaných procedur je jednodušší
        - Zkompilované a spojené procedury
            - Jedna procedura může zavolat proceduru v segmentu n použitím dvojdílné adresy (n,0) k adresování slova 0
            - Jestliže je procedura v segmentu n následně pozměněna a znovu zkompilována, nemusí se měnit žádné další procedury
                - U jednodimenzionální paměti jsou procedury nahuštěné těsně jedna na druhou, změna velikosti jedné procedury může ovlivnit počáteční adresy jiných, a musí se všechno překompilovat

- usnadňuje sdílení procedur, nebo dat, mezi několika procesy
  - Běžným příkladem je **sdílená knihovna (**V stránkovacích systémech je taktéž možno sdílet knihovny, ale je to mnohem komplikovanější)
- Mohou mít rozdílné segmenty rozdílné **druhy ochrany**
  - Jelikož každý segment obsahuje pouze jeden typ objektu, může mít ochranu vhodnou pro tento typ
  - Např. Procedurální segment může být určen jako pouze vykonávací (nelze číst/zapisovat)
  - Např. segment s proměnnými může být čistě pro čtení/zápis
- Obsahy stránek jsou v jistém smyslu náhodné. Programátor není obeznámen s faktem, že stránkování zrovna nastává
- Virtuální paměť Pentia obsahuje segmentaci i stránkování

Tabulka 2: Porovnání stránkování a segmentace.

Vlastnost	**Stránkování**	**Segmentace**
Potřebuje programátor vědět, že se používá příslušná technologie?	Ne	Ano
Kolik lineárních adresních prostorů je k dispozici?	1	Mnoho
Může celkový adresní prostor překročit velikost fyzické pamětí?	Ano	Ano
Mohou být oddělena data od kódu a odděleně zabezpečena?	Ne	Ano
Je možné se snadno přizpůsobit blokům proměnné velikosti?	Ne	Ano
Napomáhá technologie sdílení kódu mezi procesy?	Ne	Ano
Proč byla daná technologie vynalezena?	Získat velký adresní prostor bez nutnosti mít více fyzické paměti.	Oddělení adresního prostoru pro data a kód a pro sdílení a ochranu.