Co je OS?

- nepopiratelná je pouze příslušnost OS k softwaru

Rutina

- opakující se činnost
- funkčně vymezená část kódu, nejčastěji ve formě podprogramu

Funkční pohled

- OS je softwarový správce prostředků
- Prostředky lze rozdělit do 2 skupin:
    - **1.** Hardwarové nebo Fyzické prostředky (všechny komponenty + periferie)
        - Centrální prostředky: CPU, RAM
        - Vnější paměti: HDD, DVD, SSD, ..
        - IO: Monitor, klávesnice, ..
    - **2.** Softwarové nebo Logické prostředky:
        - Jsou vytvářeny rutinami OS za využití jiných prostředků, jak Fyzických, tak Logických
        - **Centrální prostředky:** Proces, Paměťový region
        - **Nízkoúrovňové log. prostředky:** Logický disk, logický terminál
        - **Dílčí prostředky:** Soubor, GUI
        - **Virtuální prostředky:** Virt. paměťový nebo síťový disk, Virt. tiskárna
        - **Synchronizační a Komunikační prostředky:** Semafor, Mutex | Roura
        - **Bezpečnostní prostředky:** Šifry, Účty, Kanály
        - **Síťové prostředky:** TCP IP Socket, HTTP server/klient

- **Proces:**
    - je instancí *(Kučera: "íí")* programu
    - Vykonání od okamžiku provedení 1. instrukce programu po provedení instrukce poslední označujeme termínem proces
    - Poznámka o procesu:
        - Každá rutina může být vykonávaná pouze v rámci procesu
        - V každém okamžiku musí existovat v systému alespoň jeden proces
        - OS není vykonáván jako jediný proces, ale jeho jednotlivé rutiny jsou sdíleny všemi procesy v systému
        - Procesy vznikají, jako reakce na požadavek jiného procesu, vyjímka Boot. proces
    - Bootovací proces (Linux: Shell proces)

- **Paměťový region:**
    - je to souvisle adresovatelná oblast paměti, která je přidělena procesu

- **OS jako správce**
    - Hlavní účel správy spočívá ve vytváření přesně definovaného a bezpečného prostředí pro procesy a to ve 2 směrech:
        - vytvoření tzv. virtuálního PC
            - OS vytváří pro programy jednotné rozhraní, které skrývá jemné rozdíly na úrovni fyzických zařízeních, toto rozhraní má charakter virtuálního PC
        - nezávislost jednotlivých procesů

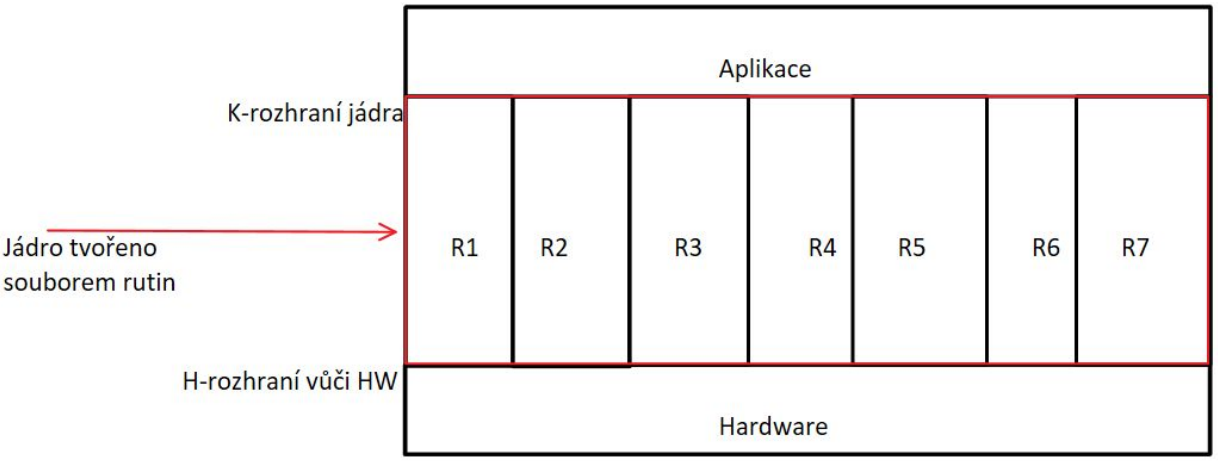
            - Pro každý proces běžící v rámci OS musí rutiny OS vytvářet iluzi, že je jediným procesem, který kdy běžel, běží a bude běžet v rámci dané instance OS

Systémový pohled

- OS je tvořen množinou rutin, které je možno na základě vzájemného volání organizovat do několika vrstev
- Nejnižší vrstva rutin přistupuje přímo k Hardwaru
- Jádro (eng. Kernel), tam patří ty rutiny, které bezprostředně přistupují k Hardwaru PC, respektive zajišťující virtualizaci
- **<u>Systémové procesy:</u>**
    - zvláštní procesy, rutiny jádra zajišťují správnou funkčnost těchto procesů
- Linux je balíčkový OS

Architektura OS

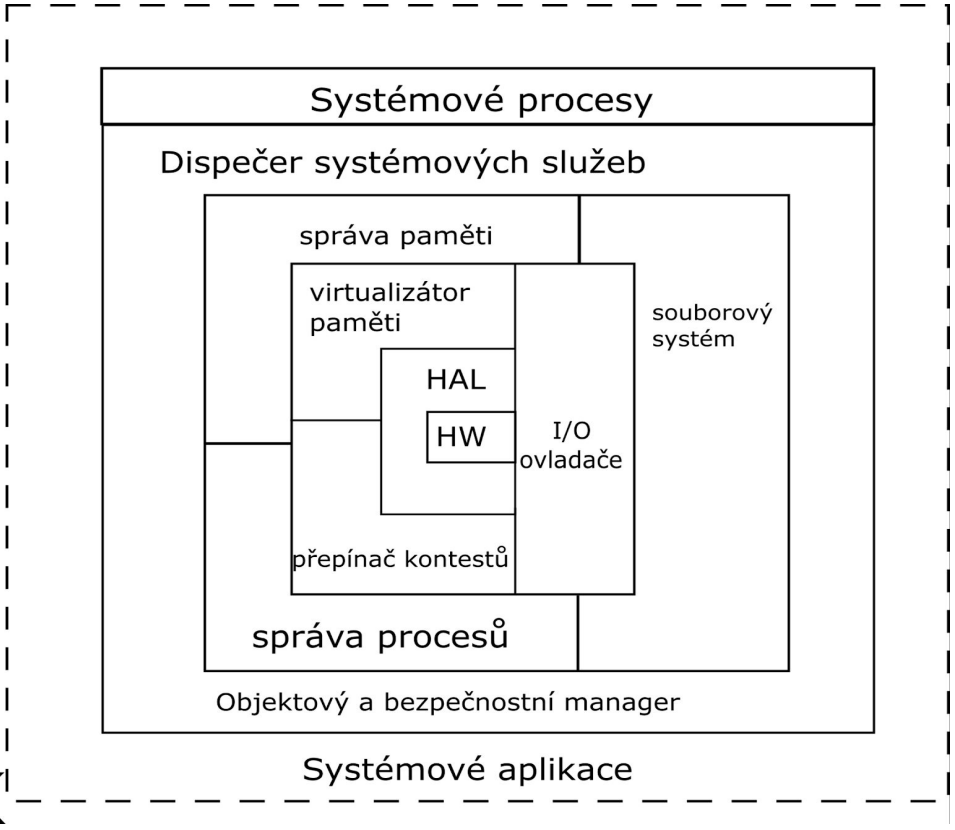
- rutiny jsou vykonávány pomocí procesů
- základní kritérium klasifikace je dle úrovně spolupráce jednotlivých rutin:
    - **1.** S těsným propojením rutin (monolitické)
        - Monolitické OS



- Rozhraní jádra využívají systémové procesy i běžné aplikace

- tato architektura je možná pouze u nejjednodušších OS vývoj vedl k její plné eliminaci
- hlavní výhoda byla jednoduchost
- rozhraní jádra využívají jak systémové procesy, tak aplikace
- jádro je tvořeno paralélně spolupracujícími rutinami
- pokud proces vykonává kód aplikace, běží v takzvaném uživatelském (neprivilegovaném režimu)
- **neprivilegovaný** - uživatelský, přistupuje pouze k části OP a nesmí provést privilegované instrukce
- v režimu jádra (privilegovaném režimu) se vykonávají pouze rutiny jádra bez omezení přístupů k celé OP
- příklad: MS DOS

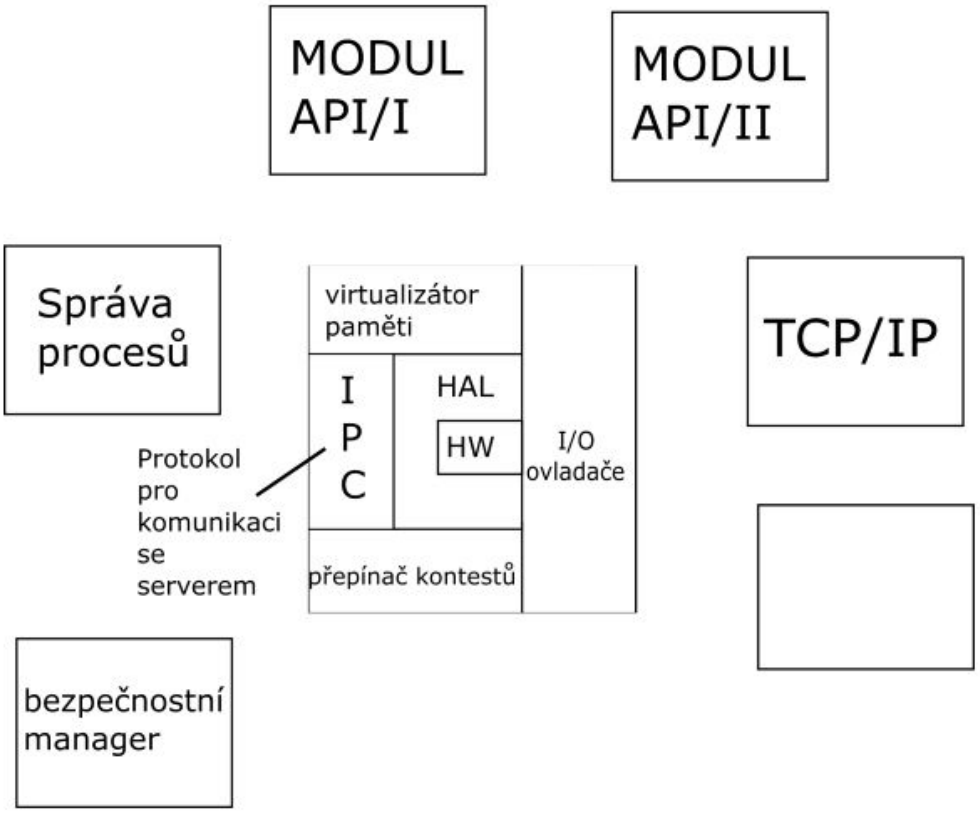
- **2.** Spolupráce rutin omezena hierarchicky (hierarchické->vrstevnaté)
  - Hierarchické OS



K

  - v dnešní době neexistuje OS, který by neobsahoval rysy Hierarchické architektury
  - Typickým → UNIX
  - zde jsou rutiny uspořádány do vrstev, které postupně obalují hardware a nabízejí vyšším vrstvám které postupně obalují hardware a nabízejí vyšším vrstvám pevně definovaná rozhraní
  - systém vyžaduje, aby rutiny přímo volaly pouze vrstvy bezprostředně nižší
  - bohužel nelze zcela dodržet → setkáváme se i s přímým voláním hlouběji zanořených vrstev
  - **hal:**
    - podobné API
    - uvnitř OS vytvořena mezivrstva, která usnadňuje programování ovladačů jednotlivých zařízení
    - **API** - rozhraní pro volání systémových služeb (vrstva s jednoduchými funkcemi pro programátory)
  - **dispečer systémových služeb:** řídí provoz

- **3.** Plně distribuované (klient-server)
    - OS typu klient-server



- **IPC =** Prostředek pro komunikaci mezi procesy
- Distribuovaný přístup (pozornost na několik objektů současně) → snižuje výkonnost systému
- Hlavní změnou oproti Hierarchickému modelu je vyčlenění "zbytečných" rutin z Jádra do specializovaných systémových procesů (Serverů)
- V Jádře zůstávají pouze zcela nezbytné rutiny pro virtualizaci paměti, přepínání a komunikaci procesů, respektive pro přístup k dalším HW prostředkům
- Výsledné redukované Jádro (označeno Mikrojádro) nabízí ostatním procesům pouze základní funkce, vše ostatní zajišťují servery prostřednictvím meziprocesorové komunikace (rychlá a efektivní komunikace je klíčová u této architektury)
- Počet serverů je formálně neomezený, mezi nejdůležitější patří - servery souborových systémů, servery bezpečnostní, servery nabízející API(nejdůležitější) s nimi komunikuje při volání služeb OS

Správa operační paměti

- OP je paměť přímo přístupná procesoru
- Uchovává kód programů, procesů, mezivýsledky jejich činností, stav ostatních prostředků
- Souvisle adresovatelná paměť rozdělená do buněk
- OP souvislý prostor paměťových buněk o velikosti 1Byte, jež jsou lineárně adresovány
- Pokud je tento adresový prostor přímo reprezentován fyzickou pamětí(RAM), označujeme jej jako Fyzický adresový prostor(FAP)
- Velikost prostoru je daná

- **Správa FAP musí plnit následující funkce**
  - Přidělování paměťových regionů na požádání procesů
  - Uvolňování paměťových regionů na žádosti procesů(Vždy musí skončit úspěchem)
  - Udržování informací o obsazení adresového prostoru
  - Zabezpečení ochrany paměti, brání procesům přistupovat k paměti mimo regiony, které vlastní
  - Podporovat střídavý běh více procesů

  - **Metody správy FAP:** triviální x netriviální
  - **1. Triviální**
    - **A) monolitická aplikační paměť**
      - Nejjednodušší správa OP
      - **Rozděluje adresový prostor na 2 bloky** = souvislá oblast adresového prostoru určená počáteční adresou - Bází a Velikostí
      - **První blok** přidělen rutinám jádra (Kernel Memory-KM)
      - **Druhý blok** přidělován na požádání procesům (Application memory-AM)
      - Paměť jádra je sdílena všemi procesy, protože rutiny Jádra, jsou užívány všemi procesy.
      - Paměť aplikační je soukromá a přístup by k ní měl mít pouze proces vlastník
    - **Algoritmus přidělení regionu**
      - Pokud je AM volná, je přidělena procesu celá bez ohledu na jím požadovanou velikost
      - Pokud není volná, je požadavek odmítnut → s fatálními důsledky pro proces
      - Alokace paměti se tudíž děje jenom jednou při spuštění procesu, kterou využívá po celou dobu svého života
      - Paměť je uvolněna při ukončení procesu
      - Stačí si pamatovat vlastníka paměti(jediný proces)
    - **Bázový registr**
      - Uložena adresa toho prostoru
      - Jeho obsah je automaticky přičítán k adrese použité ve strojovém kódu a teprve tento součet je užit k fyzické adresaci paměťového místa
      - Ochrana paměti se omezuje pouze na ochranu paměti jádra(protože v paměti nejsou žádné další regiony)
      - Nejjednodušší metodou je použití bázového registru, který fyzicky znemožňuje použití fyzických adres nižších než je báze
      - V režimů jádra obsahuje bázový register nulovou adresu a proces tak může přistupovat k celému adresovému prostroru
      - Příklad OS: CPM(některé rysy v MSDOS)
    - **B) Statické bloky**
      - Pro informovanost u statických bloků se používá pole
      - Komplikuje se i ochrana paměti, protože použití bázového registru chrání bloky jen na nižších adresách
      - Pro ochranu bloků na vyšších adresách používáme tzv. **limitní registr** (obsahuje velikost aktuálního regionu)
      - Hodnota lokální adresy je ještě před přičtením bázového registru porovnána s hodnotou limitního, pokud je větší je vyvolána výjimka
      - Největší změnou je alokace více bloků jedním procesem
      - Adresový prostor procesu je rozdělen na 3 hlavní regiony
      - **Kódový region [CR] -** obsahuje strojové kódy(kódy programů) - není do něj zapisováno
      - **Datový region [DR] -** obsahuje statická data programů - vyžaduje zápis a čtení
      - **Zásobníkový region [SR] -** obsahuje lokální proměnné a návratové adresy funkcí organizované jako zásobník

- **LIFO** - last in first out
- Strategie statických bloků umožňuje souběžnou existenci více procesů i bez odkládání na disk
- Počet souběžně existujících procesů je však omezen počtem (paměťových) bloků
- Hlavní nevýhodou je malá pružnost systému alokace → Statické bloky se používají pro správu paměti jádra, kde lze požadavky jednotlivých rutin odhadnout

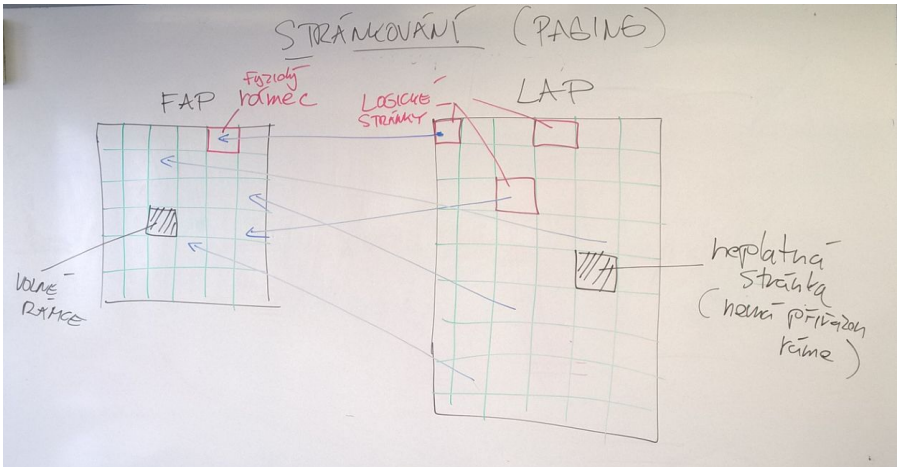
- **C) Dynamické bloky**
  - aplikační paměť lze rozdělit i na bloky, jejichž velikost se přispůsobuje požadavkům procesů
  - v počátečním stavu (před alokací 1. regionu) tvoří aplikační adresový prostor jediný volný blok
  - Strategie alokace:
    - **1. First-Fit**
      - při alokaci vyhledá 1. přípustný blok
      - pokud je jeho velikost rovna požadavků je blok přidělen celý, v ostatních případech je rozdělen na 2 bloky, 1. požadované velikosti je přidělen procesu, 2. zůstává volný
    - **2. Best-Fit**
      - Volí nejmenší blok z přípustných (pomalé a vede k fragmentaci)
  - Při uvolňování se musí provádět tzv. **scelování nových bloků**
  - Základní informace o bloku umisťujeme bezprostředně před něj → tvoří tzv. **hlavičku bloků**
  - ochrana paměti je obdobná blokům statickým (bázový a limitní registr)
  - Hlavním problémem je fragmentace OP to jest ke vzniku mnoha malých nesouvislých bloků volné paměti
  - Důsledkem je situace, kdy je
  - sice relativně velké množství volné paměti, ale nikoliv souvislé bloky rozumné velikosti
  - Fragmentace vede k degradaci celého systému (Procesy stále čekají na pamět a nové nelze spouštět)
  - Problém s fragmentací lze odstranit tzv. Setřásáním bloků = přesunem bloků vedoucím k uvolnění paměti

- **2. Netriviální**
  - **A) Virtualizace paměti**
    - Metody správy OP u triviálních metod nebyly příliš optimální
    - řešením na vyšší úrovni je úplná virtualizace paměti
    - Vzniká souvislý adresový prostor, který nemusí být reprezentován fyzicky souvislým blokem OP
    - Velikost tohoto tzv. virtuálního lépe logického adresového prostoru (LAP) není omezena velikostí skutečné OP, ale pouze velikostí adresy (u 32-bit adres 2˘32 = cca 4GB | u 64-bit adres 2˘64 = 16 EB)
    - tento LAP Lze rozdělit do 2 zákl. částí:
      - 1. část tvořena adresami, které nebyly použity (nebylo z nich čteno nebo do nich zapisováno)
        - tato část nemusí mít a většinou ani nemá žádnou fyzickou reprezentaci (data z této části nejsou nikde uložena)
        - tato část tvoří v mnoha případech větší část LAP
      - 2. část obsahuje adresy, na nichž jsou uložena skutečná data a musí mít i skutečnou fyzickou reprezentaci

- tuto část lze rozdělit na několik podčástí, podle místa uložení dat
- data jsou ukládána buď v operační (primární) paměti - OP nebo na vnějších (sekundárních) paměťových zařízeních - HDD
- důvodem jsou charakteristiky obou paměťových technologií
- OP je rychlá, ale relativně malá, disk má vyšší kapacitu, ale je relativně pomalý
- proto existuje virtualizátor (= systémové rutiny zajišťující virtualizaci paměti) řeší tento rozpor tím, že právě užívaná data jsou umístěna v OP, data delší dobu neužívaná, jsou odložena na disku
- Hlavní podstatou virtualizace je převod požadavků na přístup k LAP na přístup k fyzickým datům na skutečném paměťovém zařízení
- Virtualizane není koncovým správcem paměti a musí být doplněn klasickým správcem paměti, který už nespravuje FAP, ale prostor logický, pro správu LAP lze použít libovolnou dříve uvedenou strategii
- Díky charekteru logické paměti se nevýhody minimalizují či eliminují
- Při použití strategie dynamických bloků pro logickou paměť může sice stále docházet k fragmentaci logické paměti, která se však při vhodně zvolené metodě virtualizace (stránkvání) neprojevuje na fyzické paměti
- Fragmentace LAP je sice nepříjemná, avšak méně nebezpečná (rozsah log. prostoru řádově vetší a navíc je užíván jen jedním procesem)
    -

Stránkování

- Je hardwarový mechanismus, umožňující plnou virtualizaci paměti
- Je prováděno procesorem
- MMU (Management memory unit)
- Mechanismus stránkování má 2 základní části:
- **Překlad adres** - Logická adresa je překládána na fyzickou
- **Výpadek stránky** - Přerušení při přístupu na neplatnou stránku

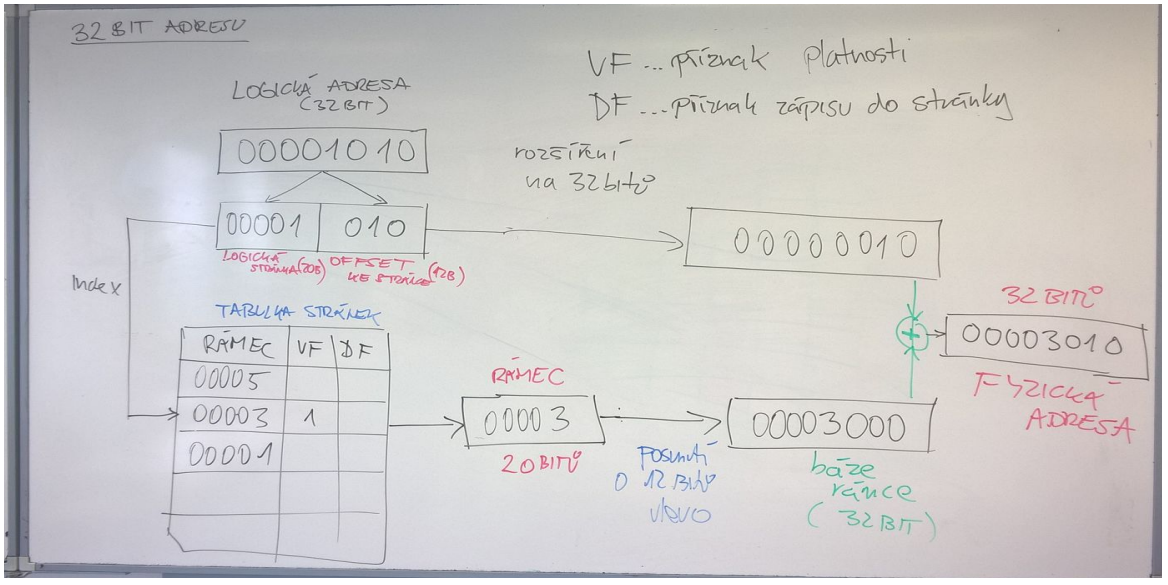


- Rozděluje LAP na **logické stránky** stejné velikosti
- Velikost stránek se může u různých architektur lišit, typicky v řádech jednotek kilobytech
- Na stejné stránky se myšleně rozdělí i FAP
- Označujeme je jako stránky fyzické či lépe jako **Rámce**

- Mezi jednotlivými fyzickými a logickými stránkami existuje zobrazení, které však nemůže být úplné (Množiny na množinu)
- Protože velikost LAP je výrazně větší než velikost FAP
- Proto vždy existují logické stránky, které nemají přiřazený rámec tzv. **Neplatné stránky** Kuči říká: “Pamatovat”
- Mohou však existovat i rámce (v FAP), na nichž není mapována žádná logická stránka tzv. **Volné rámce**(Tvoří nevyužitou fyzickou paměť) Kuči říká: “Pamatovat”

Mechanismus překladu adres při stránkování

- Každá logická adresa se rozdělí na 2 části
- První vyšší označuje logickou stránku(tj. její pořadí)
- Druhá nižší obsahuje adresu paměťového místa vztaženou k dané stránce(tzv. **Offset**)
- Přímým procesem překladu projde pouze číslo logické stránky
- Překlad se děje převodem čísla(pořadí) logické stránky nad číslo(pořadí) rámce prostřednictvím tzv. **tabulky stránek**
- Tabulka obsahuje položky, které jsou indexovány a obsahují číslo rámce a skupinu příznaků
- Získané číslo rámce fyzické stránky je bitovým posunutím vlevo převedeno na bázovou adresu rámce ve fyzické paměti
- Posledním krokem je přičtení stránkového offsetu k bázové adrese rámce, čímž se již získá fyzická adresa
-



Druhý mechanismus stránkování

- Co však nastane pokud proces přistoupí k neplatné stránce**?**
  - Za této situace nastane **<u>výpadek stránky</u>**
  - Výpadek stránky(V
  - výjimka) - přeruší instrukci a předá řízení obslužné rutině v jádře systému, tato rutina má 2 možnosti jak na situaci reagovat:

1. Nalezne odpovídající rámec a zajistí jeho propojení s logickou stránkou a tu následně splatní(V
2.
3.
4.
5. tabulce stánek se uloží index rámce a nastaví bit platnosti[VF-validity flag])
6. Ukončí proces - pošle se výjimka(win 32)

- O tom jak obslužná rutina zareaguje, rozhoduje umístění logické stránky v LAP
- Poslední fáze výpadku stránek spočívá v opětovném provedení přerušené instrukce, která již na druhý pokus výpadek stánky nezpůsobí, protože stránka už je platná
- Je nutno zdůraznit, že restartování instrukce není v moderních procesorech triviální operací(MMU - tvoří relativně významnou část procesoru)

Stránkování na žádost

- Základním principem je tzv. **lenivé vykonávání**
- Lenivé vykonávání - vše je provedeno až v okamžiku, kdy je toho skutečně třeba
- Po přidělení regionu jsou všechny jeho stránky neplatné a platnými se stávají až po prvním přístupu k nim.
- Jinak řečeno fyzická paměť je procesu přidělována až v okamžiku, kdy ji skutečně potřebuje.
- Logické stránky uvnitř regionů, které jsou neplatné(protože k nim proces ještě ani jednou nepřistoupil) se označují jako **panenské stránky**
- Na začátku jsou všechny stránky panenské
- Jak systém vyřeší výpadek stránky při přístupu k panenské stránce, to závisí na druhu regionu v němž tato logická stránka leží

1. V kódovém regionu

- **A)** Aby obslužná rutina zajistila splatneni stránky musí nalézt volný rámec(pokud není žádný rámec volný, musí jej ukrást)
- **B)** Dále musíme do tohoto rámce nahrát odpovídající segment spustitelného souboru
- **C)** následně se rámec propojí s logickou stránkou(V tabulce stránek se uloží identifikace rámce a nakonec se stránka splatní)
- Nastaví se bit platnosti v danné položce tabulce stránek
- Poté výpadek stránky skončí návratem z přerušení a procesor opakovaným čtením požadovanou instrukci načte

2. V datovém regionu

- Reakce se liší v závislosti na požadované počáteční hodnotě daného paměťového místa

A. Nejjednodušší je situace v případě, že na počáteční hodnotě nezáleží.

- Zde stačí logickou stránku propojit s libovolným rámcem a logickou stránku splatnit

B. Pokud je požadována nulová hodnota, je použit podobný přístup jako v předchozím případě, pouze při vyhledávání volného rámce jsou preferovány rámce, které jsou vyplněny pouze nulovou hodnotou

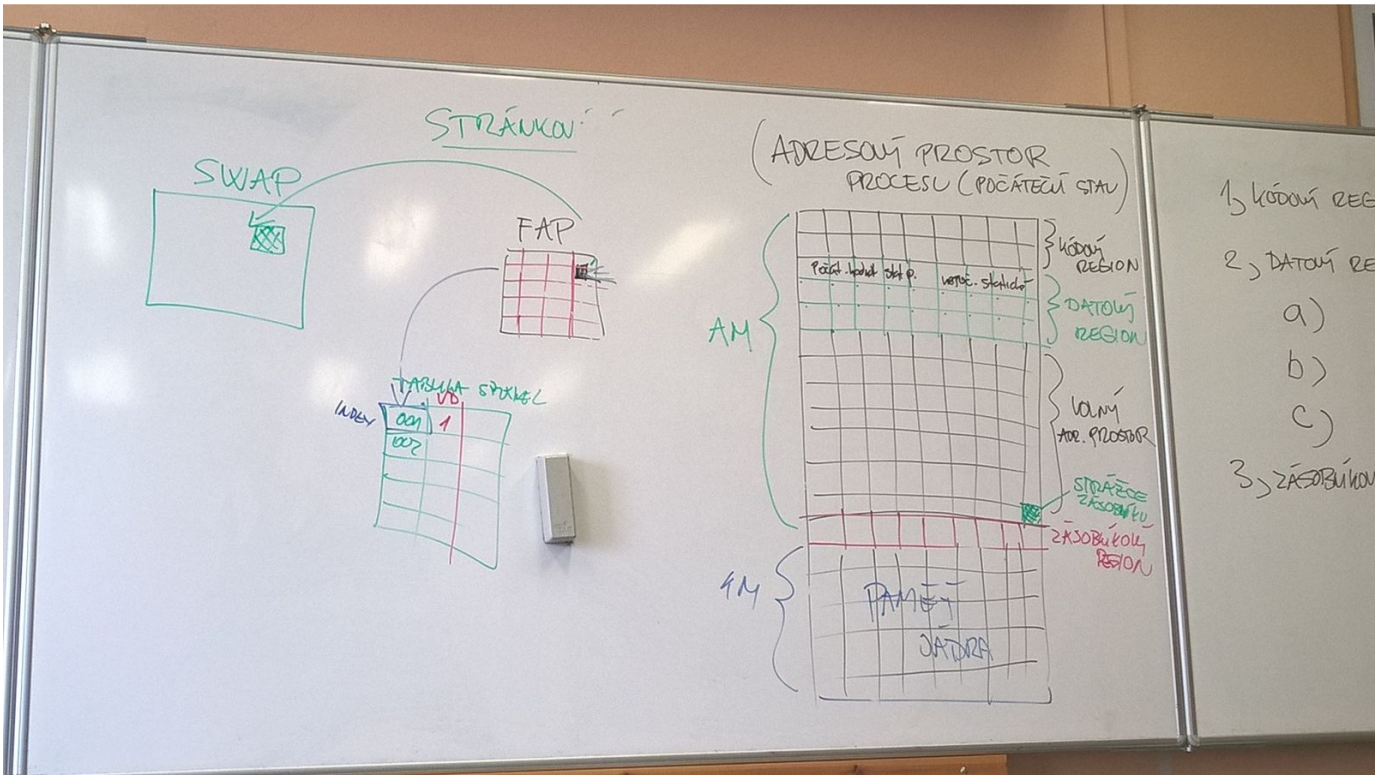
- Pokud se nenajde, vezme systém za vděk libovolným volným rámcem, který však před připojením vynuluje

C. Posledním podtypem jsou stránky na pozicích statických proměnných s nenulovou počáteční hodnotou.

- Počáteční hodnoty těchto proměnných jsou uloženy ve spustitelném souboru, kde tvoří zvláštní datový blok(Tj. při výpadku panenské stránky je vzat volný rámec, do něj je zkopírována odpovídající část spustitelného souboru a následně je rámec s danou logickou stránkou propojen)

3 V zásobníkové regionu

- Při výpadku stačí připojit libovolný volný rámec , při čemž nulování není vyžadováno



- Zvětšuje se zásobníkový region, výjmečně i datový
- V oblasti paměti jádra jsou panenské stránky za běhu systému spíše vyjímkou, protože většina logických stránek je využita již ve fázi bootování, nebo jsou dokonce od své alokace pevně vázány na rámce, nemohou být ani ukradeny

11/10/2017

Jak si ukrást stránku

- Předchozí přehled reakcí systému na výpadky stránek předpokládal, že fyzické OP je k dispozici téměř neomezené množství, tj. obsluha výpadku stránky vždy najde volný rámec
- OP není nikdy dost a systém je tak nucen odebírat rámce procesům, kteří tyto rámce vlastní, ale dostatečně je nevyužívají
- Tento proces se označuje jako **kradení stránek** (Promítá se do mechanismů stránkování na žádost, činí ho složitějším)

Kradení probíhá v následujících fázích

1. **Počáteční podmínka(nenalezení volného rámce)**
    - Virtualizátor si udržuje informace o využití všech rámců
    - Každá položka tzv. **tabulky rámců**(popisuje stavy, volných, obsazených, pozice odloženosti na SWAPU **)**
    - Při výpadku stránky je tabulka prohledána a pokud není nalezena volná stránka je zahájen proces **kradení rámce**

2. Vytipování vhodného rámce

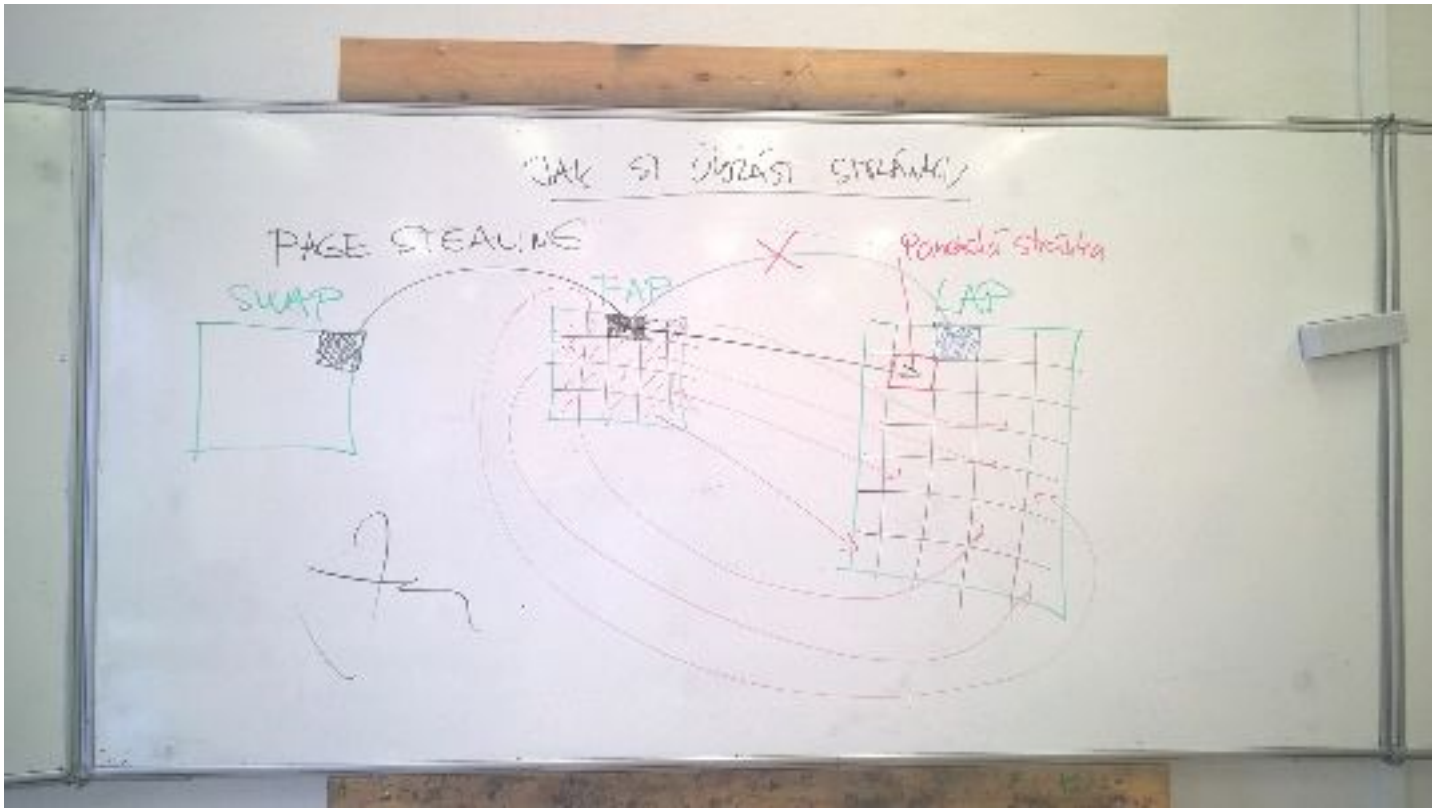
- Základní strategie pro nejvhodnějšího kandidáta na ukradení = rámce jehož ukradení by co nejméně zpomalilo systém a dotčené aplikace
- Nyní předpokládáme, že výsledkem této fáze je jediný rámec, který bude v následujících fázích, který ho využívá

3. Uložení obsahu vytipovaného rámce do SWAPU

- Prvním krokem je odložení obsahu rámce do **SWAPu** (na disku)
- Pokud nemá zvolený rámec svůj obraz na **SWAPu**(nebyl ještě nikdy ukraden)
- Je nalezen volný blok není-li na **SWAPu** místo musí být proces pro nějž je paměť kradena bezpodmíněčně ukončen (tzv. vyčerpání fyzické paměti)
- Má-li však rámec svůj obraz (již byl dříve ukraden) je rámec odložen do bloku původního obrazu
- Uložení obsahu paměti na disk je velmi pomalá operace, z tohoto důvodu neukládá ty rámce, jejichž obraz je již na disku a které nebyli od posledního uložení změněny
- Pro zjištění zda nebyla data změněna slouží tzv. **dirty bit** (rámec který obsahuje jiná data, než jeho obraz ve SWAPu, se označuje jako špinavý rámec)
- V případě přesunu dat do sekundární paměti, naplánuje proces uložení daného rámce na SWAP, a poté se zablokuje (opustí dobrovolně procesor a čeká pozastaven na dokončení této operace)

4. Zneplatnění logické stránky, která na kradený rámec odkazovala

- Po uložení rámce na disk, je proces odblokován a poté co opět získá procesor, zneplatní logickou stránku, s níž byl rámec propojen a nastaví, že při jejím potenciálním výpadku má být obnovena ze SWAPu (v tabulce stránek - identifikace příslušného bloku na SWAP)
- Nyní když je stránka volná, a může být propojena s logickou stránkou, jejíž výpadek celý proces kradení odstartoval
- Pokud požadavky na paměť výrazně převýší kapacitní možnosti OP a procesy se tudíž přetahují o rámce, dochází k velkým výpadkům tj. systém nedělá nic jiného než hořečně SWAPuje na disk a procesy stojí tzv. **zahlcení** (**trashing**)



18/10/2017

Strategie kradení stránek

- Strategie určuje, který z obsazených rámců je nejvhodnějším kandidátem pro ukradení a přidělení jiné logické stránce
- Nejznámější a nejčastěji používanou strategií je LRU(**Least recently used** - volba nejdéle nepoužívané stránky)
- Platí pravděpodobnost dalšího přístupu při delší nečinnosti klesá
- Platnost tohoto tvrzení je dána tzv. zachováním **principu lokality**
- Pracovní množina stránek procesu(PMSP) p v čase t
- PMSP je množina těch rámců, k nimž proces přistupoval v čase <t-**Δ**t>, kde delta je časový interval krátký vzhledem běhu programu
- PMSP se v čase mění (přibírání a mizení rámců dle vývoje procesu)
- Míra zachování lokálnosti lze např. vyjádřit

$$\frac{\text{Card}(Wp(\Delta t) \cap Wp(t))}{\text{card}(Wp(t))} \ (0,1)$$

- Blíží-li se tato míra 1 je princip lokality zachován(LRU je vhodnou strategií)
- Blíží-li se k 0 není LRU efektivní strategií
- Bohužel i když je LRU proveditelné, vyžádala by si jeho podpora neúměrnou režii, protože by od procesoru, vyžadovala při každém přístupu k paměti zanechání časového razítka(timestamp) v tabulce stránek a v okamžiku kradení nalezení minimální hodnoty tohoto razítka mezi všemi logickými stránkami
- Z tohoto důvodu se používají strategie jako pseudo LRU(málo společného s LRU)

Pseudo LRU

- Většina těchto strategií volí libovolnou stránku, která není v aktuálních pracovnách množinách procesů (NRU-not recently used)
- OS vynuluje dirty-bity všech logických stránek v alokovaných regionech a nechá běžet aplikaci
- Jedničkovým dirty bitem jsou identifikovány všechny stránky v PMSP
- Naopak stránky u nichž je dirty bit nulový jsou kandidáty na ukradení
- Závěrečné rozhodnutí, která stránka bude nakonec ukradena může být libovolné

Zloděj stránek

- Tento systémový proces je většinu času pozastaven a probouzí se pouze tehdy, když podíl volných rámců klesne pod stanovenou dolní mez(mezi 3-7%)
- Po probuzení zjistí jaké rámce nejsou využívány, a potom je začne krást, nikoliv však pro sebe, ale pro ostatní procesy(tj. nepřipojuje je ke svým logickým stránkám, ale nechává je volné)

Výhody zloděje stránek oproti strategii „Ukradni si sám"

- Urychluje ukládání odcizených stránek(krade po stovkách)
- Brání nestabilitě, která může vzniknout pokud se více procesů snaží ukrást jedinou stránku(procesu je ukradená stránka znovu ukradena a to dříve než ji stačí použít)

19/10/2017

Lokální strategie FIFO

- Strategie LRU není jediná v praxi používaná strategie
- Zvolen rámec, který je nejdéle používán(nejdéle propojen s logickou stránkou)
- Každý nově propojený rámec je zařazen na konec fronty(FIFO) a z přední části vybíráme stránky, které budou ukradeny
- Musí existovat správce pracovních množin - kontroluje míru využití přidělených rámců a dle ní přiděluje či odebírá procesům rámce. Ty které odebere jsou nulovány a ihned přidělovány jiným procesům(Odebírání se děje pouze při nedostatku OP)

Sdílená paměť

- Do teď jsme předpokládali, že každý region paměti je vlastněn pouze 1 procesem a že každý rámec OP je propojen nejvýše s jednou log. stránkou
- Virtualizace paměti umožňuje realizaci tzv. sdílených pamětí = Fyzický paměť prostor(OP+SWAP) přístupný z několika LAP
- Základní myšlenka je ve sdílení obsahu tabulky stránek, tj. položky tabulky stránek, příslušné danému bloku, obsahují ve všech LAP shodné údaje
- Správa(Kučera: “ Správa se s”) sdílené paměti je určena 2 principy
    - princip konzistence - všechny procesy musí mít v každém okamžiku stejný obraz paměti
    - Principem lenivého vykonávání - vše je provedeno až tehdy, kdy již není zbytí
- Pokud dojde k ukradení rámce, který je spojen se sdíleným regionem, musí být zneplatněny odpovídající log. stránky ve všech dotčených adres. prostorech
- Naopak pokud dojde ke zplatnění stránky v rámci výpadku stránky způsobené jedním z procesů, stačí když je zplatněna jen ta jediná výpadkem dotčená log. stránka, při výpadku dalších stran, může systém detekovat, že obraz této stránky již v OP existuje a následně zajistit její propojení
- Kódový region je sdílený a připojí všechny instance daného programu do svého LAP
- V každém případě se šetří swap a urychluje se běh aplikací
- Použití sdíleného kódu je ve vetšině současných OS dále rozvinut podporou tzv. linkovaných knihoven (ve Win - .dll | Unix - .so) a ty jsou na požádání procesu připojeny až při jeho spuštění, jsou sdíleny všemi aplikacemi, které je v daném okamžiku potřebují

Dočasně sdílená paměť

- (Copy on Write)
- Kromě klasické sdílené paměti podporují některé OS i tzv. **dočasně sdílenou paměť**, tvoří ji minimálně 2 paměťové regiony, které zpočátku odkazují na rámec, pokud provádějí jen čtení situace se nemění, pokud ovšem jeden z procesů se pokusí do paměti zapsat je vyvolána procesorová vyjímka a v jejím rámci dojde k rozštěpení fyzické paměti
- Proces, který způsobil vyjímku získá svůj vlastní soukromý rámec, do něhož je zkopírován obsah z původního sdíleného rámce
- Po návratu z vyjímky proces zápis dokončí, ale tentokrát již do vlastní a nesdílené paměti, pro ostatní procesy nedojde ke změně
- Tato strategie se označuje jako COPY ON WRITE neboli kopírování při zápise
- COPY ON WRITE je výhodné především v případě, pokud oba procesy z paměti pouze čtou
- implementace mechanismu je snadná, u každé logické stránky spojené s COPY ON WRITE rámcem nastavit zákaz zápisu spolu s bitovým příznakem COPY ON WRITE.

02/11/2017

Správa procesu

- Procesy patří mezi základní prostředky všech OS
- Proces popisujeme z různých pohledů, které se navzájem doplňují
- **3 základní pohledy:**

  1. Genetický
  - Proces je instancí programu
  - Program je nejčastěji reprezentován souborem, který obsahuje kód aplikace a její iniciální data(**spustitelný soubor**)

  2. Dynamický
  - Proces je postupným vykonáváním instrukcí
  - Proces ovlivňuje a může být ovlivňován prostředky a okolím

  3. Systémový pohled
  - Udržuje informace o užívaných prostředcích a jejich stavech
  - Důležitou vlastností procesu je jeho jednoznačná identifikace mezi všemi ostatními procesy
  - Systémové hledisko se promítá do dynamického

- Proces z dynamického hlediska předpokládá změnu stavu, po provedení každé instrukce
- Závislost budoucích stavů na stavu aktuálním je velmi vysoká

Kontext procesu

- Sjednocení stavů všech prostředků OS, jež proces v daném okamžiku používá
- Jak se budou chovat v krátké budoucnosti
- **Co znamená, že v daném okamžiku používá? (představit jako řez procesu)**

Determinismus
- Do kontextu procesu patří stavy všech prostředků u nichž je možnost, že by jejich změna mimo proces vedla ke změně chování programu
- Většina OS vyžaduje, aby dali najevo, jaké prostředky chtějí v budoucnu využívat
- Například do kontextu běžného programu mohou patřit stavy následujících prostředků (Procesor, Kódový paměťový region, Datový paměťový region, Zásobník, Paměť grafické karty, Paměť disku, buffery a registry zabudované do periferních zařízení, napůl vytištěná stránka v tiskárně, vypálené stopy na právě vypalovaném DVD)

- Velikost takto široce pojatého prostoru je v řádech MB -> GB
- Kontext se musí ukládat i obnovovat v rámci přepínání procesů při multitáskingu a to by trvalo až vteřiny

Jak to vyřešit?
- Východiskem je skutečnost, že pokud je prostředek využíván výhradně jedním procesem (**vyhrazený prostředek**) není nutné jeho stav obnovovat a tím ani ukládat
- Důvodem je neměnnost stavu prostředku
- Úplného vyhražení dosáhneme pouze virtualizací prostředků
- Jsou 2 základní strategie při virtualizaci

Strategie rozdělení sdílených prostředku

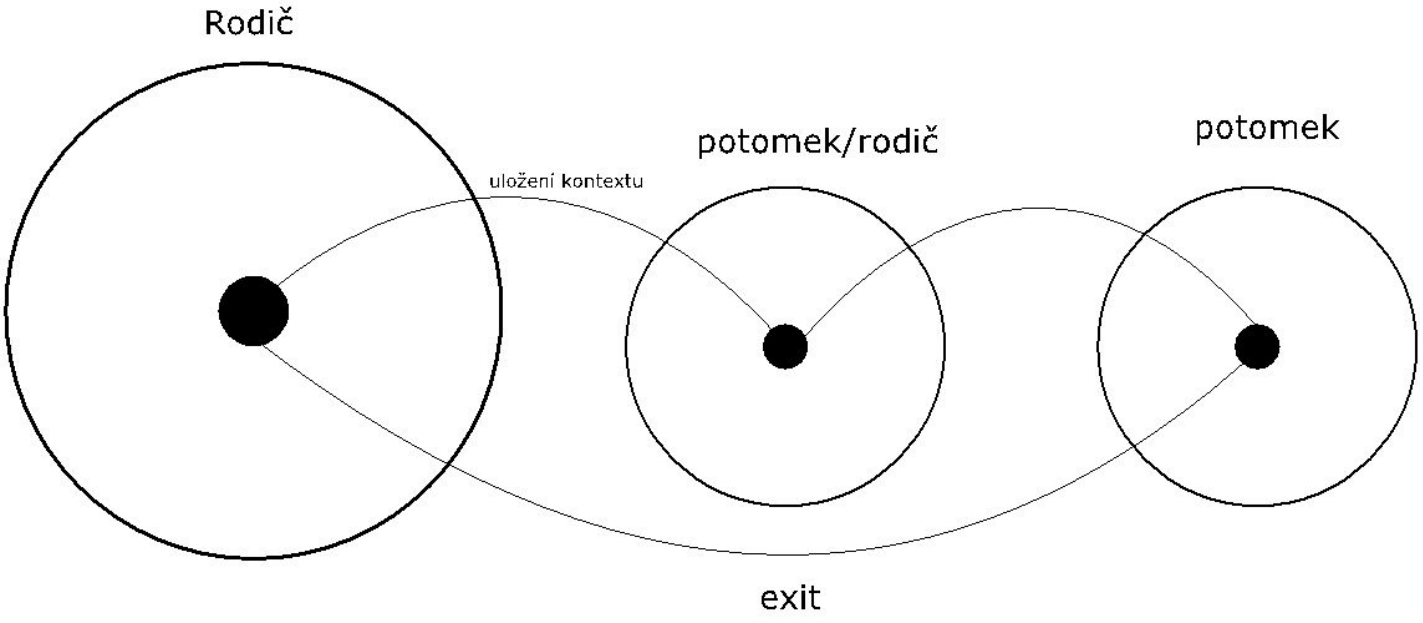
- Je užívána u většiny prostředků, neboť téměř vždy je možno prostředek virtuálně rozdělit na menší části, které mohou být přiděleny procesům do výhradního užívání
- Klasickým příkladem této strategie je rozdělení vnějšího paměťového zařízení na soubory, které jsou ve většině případů ve výhradním vlastnictvím procesu
- Dalším příkladem je rozdělení monitoru na okna

Strategie vyhrazených serverů

- Užívána u prostředků, které lze obtížně rozdělit
- V tomto případě se jeden z procesů stává výhradním vlastníkem prostředku
- Jeho funkčnost nabízí nejčastěji ve formě fronty požadavků
- klasický příklad - správa tiskárny tiskovým serverem (pouze proces server má přístup k tiskárně, další procesy musí využívat nepřímý prostředek - tiskovou frontu)

Multitasking

- správa procesů, která umožňuje existenci více nezávislých procesů v jednom okamžiku
- Efekt skutečného **multitáskingu** je při dostatečně rychlé výměně procesů(milisekundy) iluze souběžného běhu více procesů i na jednoprocesorových strojích



Vzájemné volání procesů

- Systémy, které nabízejí pouze vzájemné volání procesů jsou zastaralé
- **Postup:**
  - Na začátku existuje jeden proces(shell)
  - Nový proces může vzniknout pouze v rámci volání služby jádra tímto procesem
  - Volání pozastaví aktuální proces, uloží jeho kontext a vytvoří počáteční kontext nového procesu, kterému následně předá řízení
  - Nyní běží jen tento nový proces, který po jisté době zavolá službu jádra exit, aby byl ukončen

Kooperativní multitasking

- K výměně procesů nedochází pouze na požádání běžícího procesu, ale vždy, když aktuální proces nemůže pokračovat v běhu, neboť čeká, až bude splněna podmínka (nemůže ovlivnit, př. Stisk klávesy, nebo pohyb myši)
- Proces se vzdá procesoru tehdy a jen tehdy pokud ho v danou chvíli nepotřebuje (dobrovolně-kooperativně)
- Systém musí zajistit, že běh procesu bude obnoven poté, co se podmínka stane pravdivou(nastane událost na niž proces čekal)
- **To vede k následujícím požadavkům na OS:**

1. K výměně procesů může docházet pouze ve službách jádra(protože pouze jádro řídí přístup k prostředkům a zajišťuje komunikaci procesů)

2. Speciální rutina nazývaná dispečer musí rozhodnout, který proces bude obnoven(protože může existovat více pozastavených procesů schopných běhu)

3. Systém musí vyřešit i situaci, kdy neexistuje žádný proces schopný běhu(všechny čekají na událost), systém v takovém případě spouští tzv. **idle proces**, jenž v nekonečném cyklu volá dispečera, bez toho, že by cokoliv jiného provedl

- Pokud aplikace komunikuje s okolím tj. v pravidelných intervalech čeká na událost(stisk klávesy, příchod paketu, zpráva, odpověď) je velmi vhodnou strategií
- Méně vhodný je pro procesy, které si vystačí pouze s výpočty nad OP (vědecko technické výpočty)

- Největší nevýhodou KM je možnost časově neohraničeného držení procesoru jedním procesem(např. pokud proces omylem vstoupí do nekonečné smyčky, bez volání systémových služeb, pokud tato situace nastane, není postižen jenom daný proces, ale i všechny ostatní, včetně procesů systémových, neboť i ty již nikdy nezískají procesor)
- Z těchto důvodů se tyto systémy prosadily pouze z hlediska snadnější implementace a menších nároků na hardware

Preemtivní multitasking

- Omezení a nevýhody Kooper. Mul.lze vyřešit relativně malou změnou strategie přepínání procesů
- U KM může dojít k výměně procesů dojít pouze tehdy, čeká-li proces na externí událost a nepotřebuje tudíž procesor
- **Existují však i další místa, v nich může potenciálně dojít k výměně procesů:**

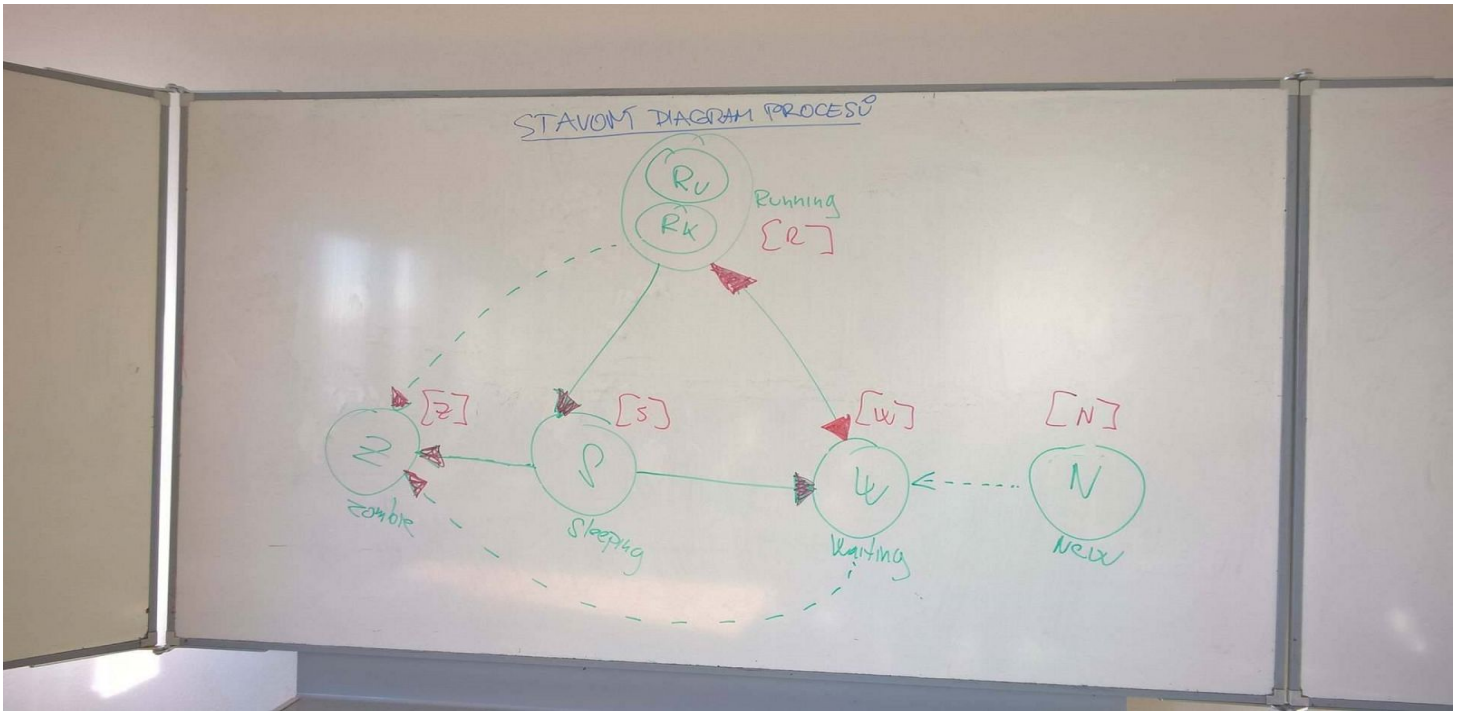
1. Okamžik bezprostředně před návratem z libovolné služby jádra (jádro vždy uvolní nevyhrazené prostředky před skončením služby)
2. Při návratu z obsluhy přerušení do uživatelského režimu (tj. jen u přerušení, která byla vyvolána za běhu procesu v uživatelském režimu)

- Důsledky této relativně malé změny strategie jsou obrovské
- Klady
- Proces může být zbaven procesoru i nedobrovolně(mohl by procesor využívat, ale místo toho bude nuceně vystřídán jiným procesem)
- Existence speciálního hardwarového zařízení tzv. časovače, který pravidelných intervalech (milisekund) vyvolává externí přerušení, což umožňuje výměnu procesů v pravidelných a relativně krátkých intervalech
- Bohužel i tak malá změna strategie výměny procesů si vyžádá rozsáhlé změny ve filozofii systémů

- Povinná systémová správa všech hardwarových prostředků a jejich úplná virtualizace(proces může být v uživatelském režimu přerušen v libovolném okamžiku)
- Nutná minimalizace časových závislostí mezi procesy-> nutnost virtualizace paměti
- Proces musí poskytovat alespoň částečnou informaci o svém nároku na využití procesoru tj. musí být vytvořen dynamický systém priorit
- Je nutné zdůraznit, že vynucené odebírání je spíše výjimkou, ale i zde převažuje kooperace, preempce jen tehdy, drží-li proces procesor příliš dlouho a systém je zároveň částečně zatížen

- Systém silněji zatížen procesy, které příliš navzájem nekomunikují(vědecko technické výpočty, multimediální aplikace) Průběh:
- Každý proces získává procesor na určitý pevný časový interval. Dán jeho statickou prioritou
- Přidělené intervaly jsou relativně krátké(desítky milisekund) a jsou tudíž neregistrovatelné člověkem.
- Vzniká tak sdílení času(time-slicing), které vytváří iluzi souběžného běhu procesů

Stavovaný diagram procesů



- Proces začíná ve stavu **Nový[N]** a končí ve stavu **Zombie[Z]**
- Většinu života však proces stráví však ve **dvou základních cyklech:**
    - Kratší obsahuje stavy **Wait[W]** (proces čeká na procesor)
    - **Running[R]** kdy proces běží
        1. 2 podstavy
        2. **RU** a **RK** podle aktuální úrovně privilegovanosti
        3. Delší cyklus navíc obsahuje stav **Sleep[S]** (proces čeká na externí událost)

New[N]

- Proces si alokuje své paměťové regiony v LAP
- Nový proces je vždy vytvářen na popud jiného procesu
- Rodičovský proces zavolá systémovou službu, po vytvoření nového dětského procesu rutiny této služby nový proces vytvoří
- Vytváření počíná vytvořením záznamu v tabulce procesů
- Pokračuje alokováním základních logických prostředků (především paměťových regionů)
- A končí vytvořením počátečního stavu
- Během celé této doby se proces nachází ve stavu **New[N]**, ale v této chvíli není schopen běhu
- Teprve po úplném vytvoření je proces přesunut do stavu **Waiting[W]** a tím začíná jeho samostatná existence
- Oba procesy(rodič i dítě) jsou zcela nezávislé a nemají žádné zvláštní vztahy

Waiting[W]

- Nacházejí se zde procesy, které jsou připraveny na vykonání, ale musí čekat na procesor, který je v danou chvíli obsazen
- Proto jsou procesy ve stavu waiting organizovány do jediné fronty FIFO - tato fronta je prioritní(s předbíháním)
- Princip prioritní fronty
- U procesů se stejnou prioritou je funkce prioritní funkce stejná jako u klasické fronty FIFO
- Při neshodě priorit procesů ve frontě předbíhají ty procesy, jejichž priorita je nižší.
- Pro označování priorit používáme reálná čísla(-20….20) kde menší číslo symbolizuje vyšší prioritu
- Výpočet priority **AP = SP + DP**
- Pokud se priority liší jsou vždy naplánovány jen procesy s nejvyšší prioritou
- Ostatní procesy získají procesor jen tehdy, pokud se všechny procesy z vrcholové skupiny vzdají dobrovolně procesoru (čekají na událost ve stavu **Sleeping[S]**)
    - Má to kooperativní charakter
    - Kdyby byli na řadě pořád jen procesy, které mají vyšší prioritu, ty méně důležité by se ke slovu nikdy nedostali

- Pro preemptivní plánování procesů je nutné využít složitějšího mechanismu - **Systém dynamických aktuálních priorit**, který spočívá v dynamické změně priorit zvýhodňováním čekajících procesů.
- Mechanismus předpokládá, že aktuální
- Priority statické, jež se během života procesu nemění a priority dynamické, které se mění podle aktuálního stavu
- Dynamická priorita se zvyšuje o stupeň u všech procesů ve Wait frontě na každý přeplánovaný cyklus
- Nulová je u procesů, které do fronty nově vstoupili
- Snižování priority je u běžícího procesu

Běžící proces[Running]

- V jednoprocesorovém systému existuje právě jeden běžící proces.
- Běžící proces může běžet buď v režimu neprivilegovaném(uživatelském) nebo privilegovaném(jádra) -> rozdělení stavu na 2 podstavy
- Proces může opustit tento stav dobrovolně(vzdá se procesoru a přechází do SLEEPINGu) jestliže čeká na systémovou událost
- Nedobrovolně - je-li mu odebrán procesor při přeplánování -> přechází do stavu WAITING a řadí se do prioritní fronty

Zablokovaný proces[SLEEPING]

- Ve stavu sleeping se proces nachází, pokud se dobrovolně vzdal procesoru a čeká na určitou systémovou událost
- Mezi hlavní typy událostí patří:
    - Vstupní událost na vstupní periferii(pohyb myši, stisk klávesy, příchod paketu…)
    - Výstupní událost(zápis na disk, odeslání znaku na monitor)
    - Uvolnění fyzického/logického prostředku
    - Uvolnění synchronizačního prostředku

- Při přechodu do stavu **SLEEPING**(zablokování procesu) by měl proces využívat minimum dalších prostředků, protože by je zbytečně blokoval
- Ve stavu sleeping se může nachzet více procesů čekajících na stejný vyhrazený prostředek
- Pro jejich uvolňování existují 2 mechanisny
    - Fronta čekajících procesů - procesy čekající na prostředek jsou řazeny do fronty a po uvlonění prostředku je pouze první z nich(nejdéle čekající) odblokován
    - Probuďte se a předbíhejte - probuzeny jsou všechny procesy a o tom který proces prostředek získá rozhoduje absolutní priorita

Proces Mátoha[Zombie]

- Procesy končí svůj život jako zombie
- Ze stavu zombie nevede cesta zpět a proces v něm zůstává až do svého úplného odstranění
- **Proces se do stavu zombie může dostat ze 3 příčin:**
    1. **Vražda**
    - proces je ukončen jiným procesem(pouze běžícím) a do stavu Mátoha přechází ze stavu **WAITING/SLEEPING**
    - vzájemné zabíjení procesů je přirozeně omezené běžný uživatel smí zabíjet pouze smí zabíjet pouze své procesy

    2. **Sebevražda**
    - běžící proces zavolá systémovou službu pro své ukončení(většinou exit) v jejímž rámci je přesunut do stavu zombie z **RUNNINGu**

    3. **Smrtelný úraz**
    - chyba v kódu, porušení ochrany paměti z **RUNNINGu,** pokud program poruší ochranu paměti, použije v uživatelském režimu prvilegovanou instrukci, či se jinak snaží narušit stabilitu OS respektive jiných aktuálních procesů, může být nedobrovolně ukončen. Aplikace je přerušena vyvoláním přerušení od procesoru(výjimkou) např výpadkem stránky

- Po přechodu do stavu Mátoha jsou programu odebrány všechny prostředky

23/11/2017

Vlákna (Threads)

- Pro preemptivní systémy je typická paralelnost na úrovni všech procesů(ale ne stejných programů) např. pokud program čeká na stisk klávesy může v nevyužívaném čase provádět na pozadí pomocné operace(textové procesory hledají překlepy, grafické programy optimalizují zobrazení…)

- Kdybychom aplikaci rozdělili do několika procesů a tím dosáhli požadovaného paralelizmu, je to nepohodlné a náročné na výpočetní prostředky
- Řešením jsou takzvaná vlákna
- Jsou obdobou procesů, sdílejí však datový region, což usnadňuje jejich vzájemné působení, protože mohou komunikovat prostřednictvím globálních statických proměnných
- Vlákna mají svou aktuální instrukci a svůj programový zásobník a souběžně běží nad společnou pamětí

- **Hlavní nevýhoda** - **globální zablokování** - jestliže dojde v jenom z vláken k zablokování procesu, jsou pozastavena i ostatní vlákna procesu
- Vlákna procházejí stejnými stavy jako klasické procesy.
- Proces lze popsat jako množinu vláken sdílejících společný datový region
- Každý proces má tzv. hlavní vlákno, které je vytvořeno během vzniku procesu a po přidělení prostředků prochází běžným stavovým cyklem
- OS může na požádání(volání služby jádra) vytvořit pro proces nové vlákno, které začne vykonávat část programu v souběhu s ostatními vlákny procesu
- Hlavní výhodou vláken je snadnější a efektivnější kooperace paralelních toků řízení -> umožňuje užití většího počtu souběžně běžících vláken bez výrazného zpomalení systému

Základní správa vláken

- Nejdůležitější operací je vytvoření vlákna a následně jeho spuštění
- Vytvoření vlákna se provádí voláním specializované služby, mezi jejíž nejdůležitější parametry patří ukazatel na funkci, v níž počne a často i končí vykonávání daného vlákna a ukazatel na případné parametry vlákna
- Vlákno ukončí svou činnost buď dosažením konce funkce, nebo může být ukončeno z vnějšku stejně jako proces(předčasně jsou ukončována všechna vlákna procesu, pokud svou činnost končí hlavní vlákno)
- OS může nabízet i další služby související s vlákny, stanovení statické priority

Synchronizace a meziprocesová komunikace

- kritický kód a vzájemné vyloučení
- Pro preemptivní multitásking je typická téměř úplná vzájemná nezávislost procesů
- Jedním z prostředků dosažením tohoto stavu je striktní oddělení tohoto prostředku užívaných jednotlivými procesy, což zcela brání kolizi procesů při jejich užívání
- Tento ideální stav panuje pouze na uživatelské úrovni a to výhradně u procesů, které s okolními procesy nikdy nekomunikují. Zcela jiná situace je u rutin jádra, které přistupují ke sdíleným prostředkům -> vznikají kolize

- **2 procesy:**

  Producent (P) - zapisuje, generuje šifrovací klíče
  Konzument (K) - čte, používá klíče

- Spolupráce mezi těmito procesy nemusí fungovat
- **Důvody:**

1. **K** využívá prostředek, který ještě není k dispozici(čte dříve)

2. **K** přečte 2x či dokonce vícekrát tentýž klíč

3. Nepřečtení klíče z důvodů jeho bezprostředního přepsání producentem(tj**. K** nestačí klíče dostatečně rychle číst)

4. Nejsložitější případ [překryv zpráv](situace daná náhodných souběhem je komplikovaná**, K** přečte sdílenou paměť, která obsahuje fragmenty dvou klíčů, novějšího a staršího)

- Každý přístup ke sdílenému prostředku respektive pokus o nesynchronizovanou komunikaci je potenciálně nebezpečný a **označujeme ho jako kritický kód** (jakýkoliv přístup ke sdílenému prostředku **)**
- Potřebujeme synchronizaci tj. předcházení negativním souběhům a komunikačním ztrátám
- **2 základní typy synchronizace:**

1. **Čekání na událost**
- proces čeká na událost, jež je výsledkem jiného procesu, synchronizace zajistí časovou následnost a zabrání propásnutí události čekajícím procesem
- Tento typ řeší problémy dané rozdílnou rychlostí
- producenta a konzumenta
- První 3 příklady s klíčem

2. **Vzájemné vyloučení**
- Synchronizace musí zabránit současnému vykonávání dvou kritických kódů nad stejným prostředkem tj. pokud jeden proces vykonává kritický kód zahájil, nebo dokončil vykonávání první instrukce a ještě neprovedl instrukci poslední nesmí jiný proces vstoupit do kritického kódu nad týmž prostředkem

- **oba typy synchronizace lze kombinovat**

Synchronizační prostředky

- Binární semafor patří mezi nejdéle známe a používané synchronizační prostředky
- Hlavní výhodou použití bin. Semaforů je jejich jednoduchá sémantika
- Nevýhoda při rychlosti
- Binární semafor je sdílený logický prostředek se dvěma stavy(červená - zelená) a dvěma operacemi Wait a Signal

- **WAIT:** pokud je semafor ve stavu zelená, je bezprostředně přepnut do stavu červená, jinak proces vyčká změny stavu na zelenou(Ta je provedena voláním operace signál jiným procesem a až poté změní sám stav semaforu na červenou, stav semaforu je změněn na zelenou)

- **SIGNAL:** zelená vstoupí a hned se změní na červenou, aby tam byl proces sám

- Z definice vyplývá, že semafory využívané pro řízení dopravy jsou i přes stejnou základní funkci pouze částečnou obdobou binárních semaforů
- Silniční semafory nejsou také na rozdíl od binárních semaforů přímo řízeny prostředky, které je využívají(auta, motorky…), ale jejich stavy jsou funkcí času
- Nejbližší analogií jsou jednoduché dvoustavové semafory železniční tratě například hradla
- Čekání před červeným semaforem a automatické nastavení na zelenou po projetí vlaku celým úsekem zajišťují vzájemné vyloučení tj. nejvýše jeden vlak může v libovolným okamžiku projíždět daným úsekem
- Aby byl semafor funkční, musí se testování hodnoty semaforu a jeho nastavení v operaci dít atomicky tj. nesmí být přerušeno přepnutím kontextu
- Kromě požadavku na atomičnost si zpracování v jádře vynucuje i požadavek na zablokování a odblokování procesu, které musí být provedeno rutinou jádra

,

Použití semafor

- Lze použít k zajištění vzájemného vyloučení nad kritickým kódem, kde má však silnou konkurenci v MUTEXU(pokud je MUTEX k dispozici, měl by být použit přednostně)
- Nezastupitelnou roli hraje v synchronizaci producenta a konzumenta nad sdílenou pamětí

12/12/2017

MUTEX

- **Mutual exclusion**
- Synchronizační prostředek, jenž se od binárního semaforu liší v jediném avšak velmi podstatném směru: MUTEX může být uvolněn pouze procesem, který daný MUTEX drží (vstoupil přes něj do kritického kódu)

- **MUTEX je určen dvěma hodnotami:**

    1. Identifikátor procesu, jenž MUTEX drží(pokud takový proces existuje, jinak není tato hodnota definována)
    2. Počet uzamčení MUTEXU nad nímž jsou definovány 2 atomické operace.

- **LOCK -** uzamčení/získání MUTEXU
    - Je-li MUTEX volný proces se stává držitelem MUTEXU a počet uzamknutí nabývá hodnoty 1
    - Je-li vlastněn aktuálním procesem, je pouze zvýšen počet uzamknutí
    - Je-li MUTEX vlastněn jiným než aktuálním procesem, je proces zablokován a čeká na uvolnění MUTEXU

- **UNLOCK -** odemčení/uvolnění MUTEXU
    - Je-li MUTEX volný, je chování nedefinováno
    - Je-li vlastněn aktuálním procesem, zmenšuje se počet uzamknutí o 1 a jel-li poté nulový je MUTEX uvolněn a jeden z čekajících procesů je odblokován
    - Je-li MUTEX vlastněn jiným než aktuálním procesem, je chování nedefinováno, ale MUTEX není v žádném případě uvolněn

- Základním pravidlem je absolutní zamezení držení MUTEXU více procesy najednou
- I Mutex musí být implementován jako semafor na úrovni jádra a musí být zaručena atomičnost operací
- **Použití Mutexu:** je používán pro zajištění vzájemného vyloučení nad kritickým kódem
- Hlavní výhodou ve srovnání s BS je snažší použíti, především v rozsáhlejších projektech dané požností vícnásobného zamykání

- Základním rozdílem mezi Mutexem a Událostí je bezestavový charakter události, tj. událost si nepamatuje svůj stav a je dostupná na uživatelské úrovni
- Jak tedy zajistit, aby ty procesy navazovali a nepřekrývali se?
- Řešením je použití sdílené stavové proměné, která ukazuje dosažený stav (nastavována je prvním procesem a testována druhým)
- Protože nastavování ani testování není atomickou činností, musí být obě činosti chráněni Mutexem

Obecný semafor

- Vhodný pro synchronizaci přístupu procesů k víceprvkové množině procesů
- Binární semafor je pouhým speciálním případem obecného semaforu
- Stavy obec. semaforu nabývají hodnot 0-max(hodnota semaforu určuje počet aktuálně volných prostředků)
- Konstanta max je neměnnou charakteristikou semaforu a je určena při jeho definici
- **Obecný semafor je dále určen 2 operacemi:**
    - **POST(Je-li stav nulový -> nemá žádné prostředky -> proces se zablokuje dokud není semafor uvolněn, jinak se o 1 sníží čítač semaforu)**
    - **Release(Zvýšení čítače o 1, je-li stav menší jak max)**

- Binární semafor je Obecný semafor s hodnotou max rovnou 1, kde operace WAIT odpovídá operaci POST a singál RELEASEu

20/12/2017

Uváznutí

- Problematická místa veškeré synchronizace -> trvalé zablokování procesů, při chybném použití synchronizačních prostředků
- př. proces, který získal semafor opětovně zavolá jeho operaci WAIT
- stav v němž se procesy vinou chybné synchronizace navždy zablokují (tj. bez ukončení nemohou ukončit stav SLEEP) označujeme jako uváznutí (DEADLOCK)
- Výše zmíněný př. patří do skupiny méně nebezpečných, uváznutí tohoto typu vznikají důsledkem hrubých chyb v použití synchrnizačních prostředků a označují se jako triviální uváznutí
- Druhou a nebezpečnější skupinu uváznutí, tvoří uváznutí vzniklá nepříznivým souběhem dvou procesů tj. když 2 procesy používají 2 spol. prostředky(Obtížně se detekují)

Eliminace uváznutí

- Nejjednodušší metodou vzniku uváznutí je vyloučení překryvu kritických kódů, není-li vyžadována přímá spolupráce od několika prostředků, v mnoha případech není přímá spolupráce nutná
- Nutnost uspořádat prostředky(respektive jejich MUTEXY) tak, aby alokace byla prováděna ve správném pořadí -> avšak ani tato strategie není bez nedostatků(neexistence uspořádání prostředků)
- Naštěstí lze zavést přijatelné a snadno použitelné uspořádání ve formě abecedného uspořádání jejich identifikátoru
- Závažnějším nedostatkem je že vynucené pořadí alokací může vést k větší časové závislosti procesů -> výrazné zpomalení běhu dotčených aplikací
- Uváznutí lze zabránit i aplikací přístupů vše nebo nic tj. proces se pokusí získat všechny prostředky, které potřebuje a když se mu to nepovede, všechny prozatím uvolní(aby je neblokoval, když sám nemůže běžet)
- Další možností je detekce potenciálního uváznutí ještě před jeho vznikem, bohužel algoritmy, které jsou schopny potenciální hrozbu odhalit, jsou buď velmi pomalé, nebo příliš defenzivní(jsou obranné až moc, vyloučí i to co by nakonec uváznutí nebylo)
- Nejznámějším algoritmem tohoto typu je takzvaný bankéřův algoritmus

Základní komunikační prostředky

- Současné OS nabízejí celou škálu prostředků
- Umožňuji přesun dat mezi jednotlivými procesy
- Tyto prostředky se liší svou rychlostí, způsobem použití
- Z toho uděláme pouze základní přehled nejdůležitějších a nejtypičtějších komunikačních prostředků
- Klasifikace komunikačních prostředků

1. **Spoluúčast jádra**
    - Jádro se komunikace neúčastní
    - Data proudí přes jádro tj. jsou minimálně 2x kopírována

2. **Vnitřní struktura dat**
    - Proudově orientované datovody - data tvoří jediný nečleněný proud
    - Zprávově orientované - data jsou organizována do zpráv

3. **Směrování přenášených dat**
    - Všesměrové -> z jednoho procesu do více procesů
    - Dostředné -> z více procesů do jednoho procesu
    - Jednosměrné
    - Dvousměrné

4. **Přenášený objem dat (přenosová rychlost)**
    - Malé(kB/s) - fronty zpráv a signály
    - Střední(desítky kB-MB/s) - roury a sokety
    - Velké(stovky MB - GB/s) - sdílená paměť

5. **Transparentní použití**

Roura

- Je klasickým komunikačním prostředkem OS Unix
- Jednosměrným datovodem vhodný především k výměně malých a středních objemů dat mezi dvěma procesy
- V případě obousměrného přenosu je nutno použít dvojici rour
- - Implementace roury je jednoduchá -> Kruhová roura (kruhová fronta) je propojena se souborovým systémem

Typy:

- V Unixu existují 2 typy rour lišící se mechanismem své identifikace

1. **Nepojmenované/anonymní roury - nemají vlastní identifikátor a jsou přístupné pouze prostřednictvím souborových deskriptorů**
    - Z tohoto důvodu nelze použít pro komunikaci mezi libovolnými procesy, ale pouze mezi procesy, které jsou potomky procesů, jenž danou rouru vytvořil

2. **Pojmenované roury - viditelné v souborovém systému tj. mají jméno a umístění v adresářové struktuře a mohou je tudíž užívat libovolné 2 procesy**
    - Pojmenované roury je možné otvírat stejně jako běžné soubory(jak pro čtení tak pro zápis)
    - Při otevírání může dojít k zablokování procesu, neboť služba open čeká jestliže není přítomen proces na druhém konci roury

Socket (schránka)

- síťový komunikační prostředek
- umožňují navázat 2 typy spojení
    - Datagramové - zprávově orientované(neposkytují záruku úspěšného doručení ani pořadí)
    - Proudové - obousměrný datovod

- Určen pěticí údajů
    - První dvojici tvoří adresa počítače, na němž běží proces poskytující určitou službu(IP adresa serveru a port)
    - Druhá dvojice údajů je tvořena údaji o procesu klienta(IP klienta a číslo portu)
    - Poslední údaj pětice určuje použitý nízkoúrovňový protokol, při čemž se výhradně užívá TCP nebo UDP

- Navazování spojení začíná otevřením tzv. půlsoketu na straně serveru, ten nabídne potenciální spojení na dané IP adrese a daném portu.
- Na druhé straně komunikace klient vyšle požadavek na spojení, který obsahuje identifikaci serveru(adresu, port, protokol)

- Pokud je na specifikovaném místě nalezen soket, jenž čeká na spojení dojde ke vzniku spojení(vzniká tzv. relace)

04/01/2018

Fronta zpráv

- Zprávově orientovaným komunikačním prostředkem
- Umožňuje cílenou i všesměrovou výměnu zpráv
- Omezené velikostí mezi dvěma či více procesy
- Každá zpráva se skládá z hlavičky a z těla
    - Hlavička - informace
    - Tělo - data
- Fronta zpráv zajišťuje směrování řazení zpráv
- Vysílající proces se blokuje pokud je fronta plná, přijímací proces se blokuje není-li ve frontě žádná použitelná zpráva
- Zprávy jsou nezbytné u OS klient-server, ale jsou užívány i v jiných systémech
- Je často užívána jako nízkoúrovňový základ složitějších a snadněji použitelných prostředků(př. GUI)

Vzdálené volaní procedur

- Umožňuje volání funkcí v jiném adresovém prostoru, nebo v jiné instanci OS
- Vzdáleně volaná procedura může přijímat parametry
- Hodnoty těchto parametrů jsou přeneseny z adresového prostoru volající funkce do adresového prostoru volané funkce a tato funkce může sama hodnotu vracet
- Jedná se tedy o zprávově orientovaný obousměrný komunikační prostředek

- **Výhody**
    - Předávání dat do funkcí
    - Automatické řazení dat

- **Nevýhody**
    - Identifikace vzdálené funkce se musí dít na základě poskytované služby(Adresa ,port)
        - Nikoliv, jen prostřednictvím adresy funkce
    - Ošetřování chybových vztahů
        - Může dojít k chybě během přenosu parametrů a návratové hodnoty
    - Přerušení spojení
    - Chybná identifikace protistrany

- pozn. Jako komunikační kanál se použije socket resp. jiný vhodný prostředek

Souborový systém (SS)

- Způsob organizace informací, tak aby se snadno našli a přistupovalo se k nim
- Uloženy ve vhodném typu elektronické paměti

- SS není nezbytný v operačním systému, ale existuje jen málo systémů bez jeho podpory
- Důvod je že pro běh OS je nutné trvalé uložení dat

- Klasickým SS je SS UNIXu, který se stal vzorem pro většinu současných OS

Struktura SS

Správa otevřených souborů
Adresářová struktura
i-uzly
Vyrovnávací paměti
Správa svazků
I/O subsystémy
HW

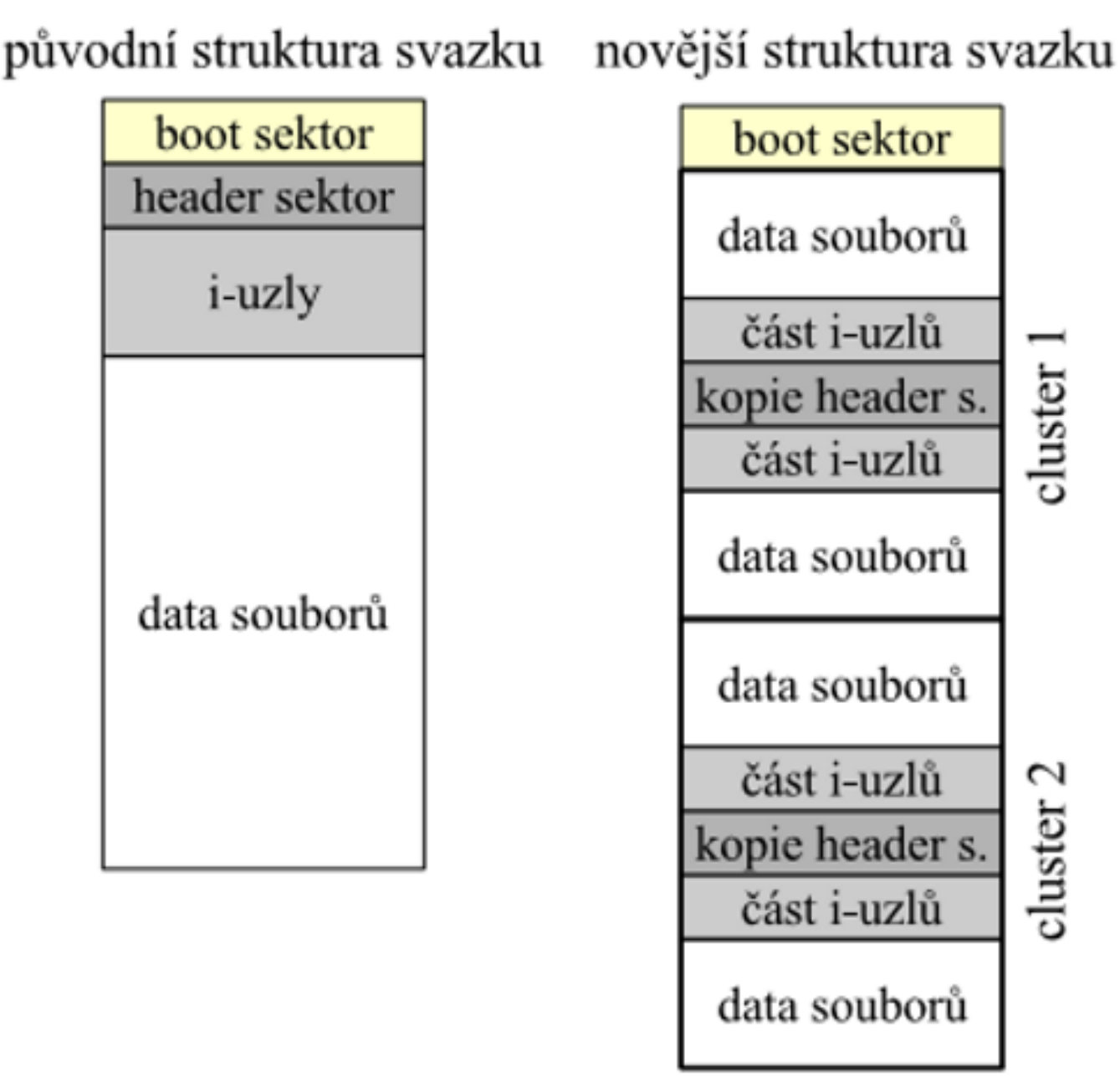
Svazek

- **Správa svazků** - Nejnižší logická vrstvou SS
- Stará se o fyzické uložení SS na blokovém zařízení. Tj., zařízení, které je tvořeno bloky o stejné velikosti.
- **Př:** disk, různé elek. paměti (fleška,regiony OP)

- **I/O Subsystém** - Propojení mezi vrstvami, nabízí vyšším vrstvám tzv. log disky, jejichž bloky jsou adresovány lineárně, na log. disku vytváří SS log. prostředek -> svazek

- **SVAZEK** - Je tvořen několika sekcemi, některé obsahují globální informace o dané instanci SS, jiné metadata -> (data o datech) o uložených souborech a přirozeně i vlastní data souborů. (Reference na data)
- Struktura svazku se liší na novější a starší strukturu svazku

obrázek na další stránce



Starší (jednodušší) přístup rozděluje log. disk na několik málo sektorů:

- **Boot sektor** – Zavádí operační systém
- **Header sektor** – Obsahuje globální data celého SS (informace o obsazení svazků, o volných blocích atd.)
- **Metadata sektorů** – Tabulka I-uzlů
- **Data souborů** - Uložená data
- Metadata jsou uložena jen jednou, pokud jsou poškozené celý svazek je (zničen)
- Při přístupu k datům je nutno číst jejich metadata, které jsou uloženy v jiné a často vzdálené sekci **-> prodlužuje přístupové doby.**

- modernější svazky (systémy). využívají schémata vníž jsou klíčová data duplikována (blíže k datům která popisují)
- tato struktura je nejvhodnější pro cylindrické disky s jednoduchým adresováním založeným na geometrii disku
- současnosti se však často používají virtuální geo. bez cylindru (bez cylindrického uspořádání) a tak výhody zanikají a zůstává jen duplicita dat.

25/01/2018
Vyrovnávací paměti (Buffery)

- Paměťové bloky v OP, které zrcadlí obsah fyzických bloků na blokových zařízeních

- **+3 základní funkce**
    1. Tvoří sdílenou paměť mezi jádrem a nesynchronizovaným čtením a zápisem na disk(zajišťován DMA řadičem)
    2. Sjednocují rozhraní k blokovým zařízením, Vyšší vrstvy očekávají logický disk s jednotnými bloky a lineárním adresováním, Buffery právě takové rozhraní vyšším vrstvám nabízejí
    3. Urychlení přístupu na disk,

- Většina čtení a zápisů se děje do bufferů a nikoliv do pomalých vnějších zařízeních

- Návrhy systému bufferů vycházejí z následujících principů
    1. **Princip konzistence -** každý blok fyzického zařízení je zrcadlem nejvýše v jednom bufferu
    2. **Princip shodnosti a sdílenosti -** Všechny buffery jsou funkčně shodné a mohou být sdíleny všemi blokovými zařízeními v systému (tj. konkrétní buffer může být použit během své existence pro různá bloková zařízení)
    3. **Princip konečných prostředků -** Buffery jsou přiděleny již při zavádění OS a dále se nemění
    4. **Princip lenivost**i **-** zápis bufferů se odkládá tak dlouho jak je možno (tzv. odložený zápis)

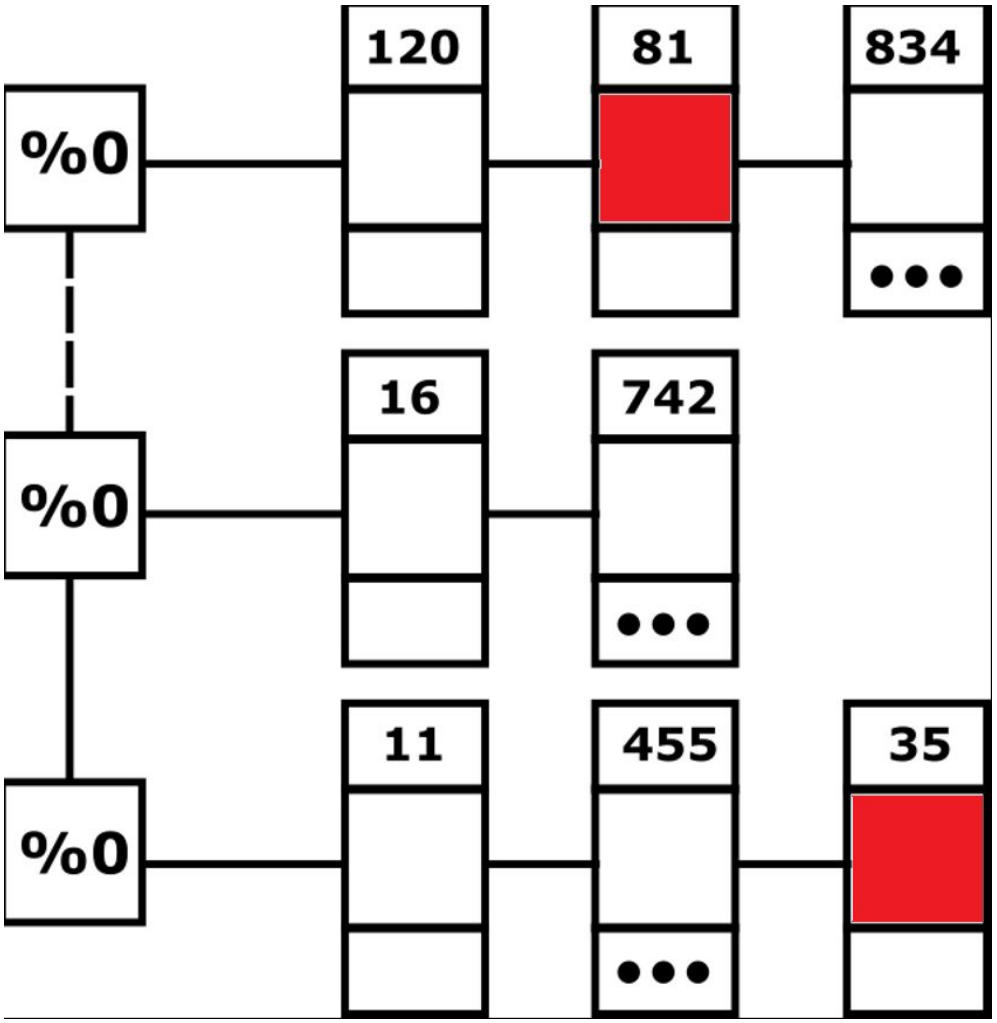
Buffery v Unixu:

- Implementace bufferu -> Složen z **HLAVIČKY a BUFFERU**
- **Hlavička:** Obsahuje informace o právě zrcadleném bloku, informaci zda je buffer právě uzavřen a ukazatel, který zařazuje paměť do některého ze seznamů, ve kterých jsou buffery organizovány

Seznamy:

- **Seznam volných bufferů-** Stránek, které nezrcadlí žádný blok. Tento font obsahuje na začátku všechny buffery a během běhu systému se postupně vyprazdňuje, až se zcela vyčerpá, neboť za normálních situacích se nedoplňuje
- **Fronta odemčených bufferů-** V ní jsou všechny přidělené (nikoliv volné!) buffery, které nejsou uzamčeny
- **Otevřená Hashovací tabulka všech přidělených bufferů-** Skládá se z většího počtu oddělených seznamů, umožňuje rychlé vyhledání bufferů pro daný blok a svazek

obrázek na další stránce



- **Proces:** možnosti, které mohou nastat při požadavku na zápis do určitého bloku na svazku resp. čtení z tohoto bloku

1 - pokud má tento blok již existujíci buffer (zjistí z hashovací tabulky) mohou nastat 2 situace:

- a) pokud je Buffer odemčen, pak ho proces uzamkne a provede příslušnou operaci a ihned potom ho odemkne a probudí případné procesy čekající na tuto paměť
- b) je již zamknut, proces se musí zablokovat

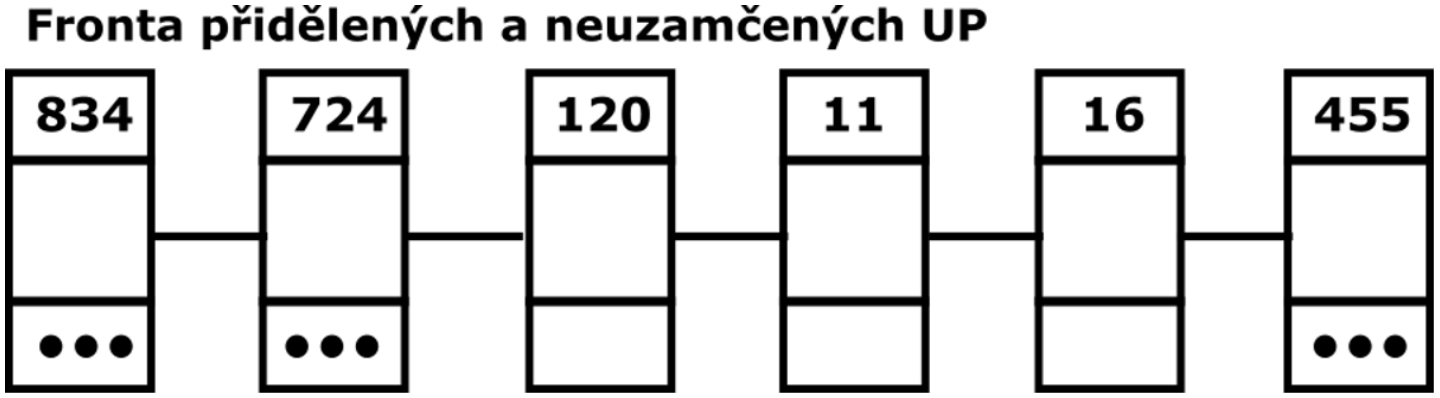
2 - Blok není aktuálně zrcadlen

- **a) Stále existují nepřidělené buffery**
  - proces odebere buffer z fondu nepřidělených, změní jeho hlavičku, na hlavičku požadovaného bloku a zařadí jej do hashovací tabulky (do příslušného seznamu) a zamkne jej, všechny tyto operace musejí být řešeny atomicky
  - Během jeho zablokování je přesun dat z , a poté se může odemknout

- **b) Neexistují žádné nepřidělené buffery**
  - Musí být ukradeny některé z odemčených(ale již samožřejmě přidělených) bufferů
  - Měl by to být buffer ze začátku fronty(je již dlouhou dobu nepoužíván)

  - **Zde mohou nastat 2 podsituace zda je buffer na začátku fronty čistý/špinavý:**
    - **Čistý -** nezměněn oproti diskovému
    - **Špinavý -** změněn oproti diskovému(s odloženým zápisem)

- Pokud je čistý, je další činnost obdobná přidělení volné paměti
- Nejdříve je však nutno paměť vyjmout jak z fronty tak hashovací tabulky.
- Dále je již vše stejné
- změněno označení bloku
- Paměť je uzamčena a vložena na jiné místo v hashovací tabulce(nyní již podle nového čísla bloku) a následně je naplánováno čtení daného bloku. Po odblokování je buffer použit a odemčen(opět se objeví ve frontě - na konci)
- Buffer není ještě uložen tj. se špinavým neboli odloženým zápisem



Subsystém I/O

- zajišťuje komunikaci s hardwarovými zařízeními (s výjimkou OP a procesoru, o než se stará správce paměti, resp. správce procesů)
- tento subsystém využívá i navíc další moduly jádra, především modul SS, ale i modul správy paměti -> SWAP
- největší význam 3 periferních zařízeni:
    - Vnější paměťová zařízení s blokovým přístupem
    - Vstupně výstupní zařízení se sekvenčním přístupem (myš, klávesnice, ..)
    - Počítačová síť
- Skládá se z ovladačů (driverů)

07/02/2018

Ovladače

- Leží mezi hardwarem a softwarem
- Nabízejí relativně jednotné rozhraní vůči vyšším vrstvám os
- V nejnižší vrstvě přistupují přímo k hardwaru(registry)
- HAL ->často naprogramována v assembleru a jako jedinná je přímo závislá na zvoleném hardwaru

Rozhraní ovladačů

- Na rozhraní ovladačů jsou kladeny dva protikladné požadavky
- Pro IO zařízení jsou vyžadována speciální rozhraní
- Na straně druhé vyšším vrstvám vyhovovalo rozhraní co nejjednotnější a nejjednodušší -> komplikace celého OS
- Proto vnější rozhraní ovladačů rozdělujeme do dvou částí

1. **Základní rozhraní poskytuje pouze operace, které lze definovat u většiny ovladačů**
2. **AD 1 - Základní obecně podporované operace**
    a. **INIT -** počáteční inicializace zařízení)
        i. Provádí se pouze jednou
        ii. Buď při startu pc, nebo po připojení zařízení
    b. **TERMINATE -** odpojení zařízení
    c. **OPEN -** otevření zařízení procesem
    d. **CLOSE -** uzavření zařízení
    e. **GET -** Vrátí informace o daném zařízení
    f. **READ -** u znakových čte byty
    g. **WRITE** - zapisuje blokové byty
    h. **STRATEGY -** u blokových zajišťuje zápis nebo čtení bloků
    i. **CONTROL -** vstupní bod ke specializovan

Identifikace zařízení a ovladačů

- Zařízení jsou identifikována dvouúrovňovým identifikátorem (tvořen dvěma čísly)
    - **Major (hlavní) -** identifikuje ovladač, index do tabulky ovladačů
    - **Minor (vedlejší)**

- **Zavoláme rutinu ->** předání parametru číslo Minor, které identifikuje zařízení, které je aktuálně , obsluhováno daným ovladačem -> index do tabulky zařízení, která je tentokrát uložena v rámci daného ovladače

- Tabulka ovladačů - obsahuje odkazy na rutiny jednotlivých služeb daného ovladače (ukazatele na funkci)

Horní a dolní polovina ovladače

- Ovladače většinou přistupují k hardwarovým zařízením která pracují asynchronně vzhledem k procesoru
- Ovladač se skládá ze dvou částí (Polovin), liší se umístěním v jádře OS a svou funkcí
- **Horní polovina** je běžnou rutinou jádra a zajišťuje rozhraní vůči vyšším vrstvám
- **Dolní polovina** je obslužnou rutinou přerušení a je vyvolána daným zařízením (není vyvolána zbytkem jádra) => běží asynchronně
- Mezí horní a dolní polovinou ovladače leží softwarová vyrovnávací paměť (jedině přes ní mohou obě asynchronní části komunikovat)
- Tato komunikace mezi horní a dolní polovinou je komunikací mezi producentem a konzumentem

Terminál

- vstupní a výstupní zařízení sloužící k interakci s uživatelem, původně se jednalo o fyzická zařízení s klávesnicí a textovým výstupem dnes o virtuální zařízení
- Speciální rysy jsou dané tzv. **linkovou disciplínou** ta spočívá ve speciálním zpracování vstupu a výstupu
- Terminál se může nacházet ve více režimech nejtypičtější je tzv. **kanonický režim**
- V tomto režimu používá ovladač terminálu trojici bufferů

- 1) Výstupní - Klasický Buffer (Data v něm čekají dokud nejsou zaslána na výstupní část terminálu, typicky monitor)
- 2) Vstup - Zde se nachází dvojce bufferů
    - a) Klasický vstupní Buffer (Do něj zapisuje dolní polovina ovladače, která obsluhuje přerušení od vstupní části terminálu, typicky klávesnice)
    - b) Výstupní Buffer (tzv. Echo znak stisklý na klávesnici se ihned objevuje na monitoru i bez toho že by byl zpracován aplikací)
- Všechny tyto akce jsou velmi rychlé a mohou být tedy prováděny v obsluze přerušení, to jest na úkor jiného procesu

Příkazy

- Hledání adresářů "**/home/uživatel/adresář který hledáme**"
- **ls** - Vypíše složky v adresáři např. " ls /home/uživatel/"
    - **ls -l** - víc info o složce
        - **ls -l -h** - info o složce s velikostí
- **pwd** - Zjistí jméno aktualního adresáře
- **date** - Datum
- **cal "Může být i rok"** - KalendářZ
- **history** - Historie příkazů
    - **!!** - vykoná poslední příkaz
    - **!-5** - vykoná 5. příklad ze spoda
    - **!ls** - Poslední ls příklad
- **grep** - Vyhledá slovo v textu např. " grep "word" filename "
- **echo** - vypíše proměnou, či text
- **echo $PATH** - Vypíše hodně adresářů :| **"/usr/bin/custom:/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games"**
- **env** - hledá ve výstupu pomocí **grep**, používá se pro export comandů, lze tím exportovat i proměnné
  npř. **export variable**
  env | grep variable
  which " comand " - hledá umístění comandu
- **alias** - lze vytvořit " vlastní příkaz "
- Využití **Asterisk** - hledá složky podle čísla/písmena např. " **echo /etc/t*** "
- **Question Mark** - hledá soubor, či složku podle délky a počátečního písmena např. " **echo /etc/r??????** ", lze použít i " **echo /etc/??????** ", díky tomu najde soubor či složku, která má délku, jako počet " ? ".
- **Brackets [ ]** - používají se pro udání rozsahu u příkazu **např. " echo /etc/[g-u]* "** vypíše jen soubory, které jsou mezi G - U
- **Point (!)** - neguje range v **Brackets**
- **Character (\)** - pro formátovaní textu, když chceme vypsat

Down arrow ↓	Go down one line
Space	Go down one page
s	Search for term
[	Go to previous node
]	Go to next node
u	Go up one level
TAB	Skip to next hyperlink
HOME	Go to beginning
END	Go to end
h	Display help
L	Quit help page
q	Quit info command

- **"/" - root** (je vrcholem adresářové struktury -> všechny soubory jsou pod ním)
  - **"/bin" -** binární spustitelné soubory, které se uplatňují při zavádění systému
  - **/boot -** obsahuje souboru pro zavední systému
    - **/boot/group -** group2 zavádí systém(základní config)
  - **/dev -** obsahuje všechny hw zařizení
  - **/etc -** obsahuje config a nelze jej sdílet
  - **/init.d -** nacházejí se zde programů, které umožní jakoukoliv službu
  - **/home -** domovský adresář všech uživatelů(až na admina)
  - **/req -** zde jsou věco pro běh programů
  - **/bin/lib**
  - **/bin/ub**
  - **/lost**
  - **/found**
  - **/media -** multimediální zařízení
  - **/mnt**
  - **/opt -** instaluje se zde sw, který není součástí distribuce
  - **/sbin -** systémové nástroje, které se uplatní při zavádění či konfiguraci systému
  - **/tmp -** jsou to dočasné pracovní soubory
  - **/user -** obsahuje lib, bin atd
  - **/var -** obsahuje podadresář log(log -> záznamy o činnosti systému)
    - /cache/apt - jsou tu dočasně dostahované balíky

Vytvoření složky

Mkdir [název složky] [název složky] [název složky]

Mkdir [název složky]/[název složky]

Smazání složky

Rmdir [název složky]

Rmdir [název složky]/[název složky]

Smazání souboru

Rm[název souboru]

Rm [název složky]/[název souboru]

Přejmenování souboru/složky

Mv [starý název souboru] [nový název souboru]

Mv [název složky]/[starý název souboru] [název složky]/[nový název souboru]

Přesunutísouboru/složky

Mv [starý název složky]/[název souboru] ~/[nový název složky]/[název souboru]

Vytvoření souboru

Touch [nazev_souboru]

Kopírování

Cp [název složky]/[název složky] [název složky]/[název složky]

Uživatele

Sudo useradd [jmeno_uziv]

Sudo userdel [jmeno_uziv]

Skupiny

Sudo Groupadd -g [cislo] [nazev_skupiny]

Sudo Groupdel [nazev_skupiny]

Balíky

Sudo apt-get update

Sudo apt-get install [nazev_baliku]

Pwd - vypsání aktuálního adresář

Ls -L vypsání aktuálního adresáře

arch - napíše architekrutu
lscpu - to identify the type of CPU in your system
cat /proc/cpuinfo - detailnější lscpu
free -m ukáže vlastnosti pamětí(velikost, využití, atd)

The lspci command shows detailed information about devices connected to the system via the PCI bus.

To display the devices connected to the system via USB, execute the lsusb

The lshal command allows you to view the devices detected by HAL

apt-get install název=verze - instalace balíčku

apt-get remove název - odstraní balíček

apt-get purge název - odstraní vše(i config)

apt-get update nazev - update

apt-get upgrade - upgrade(aktualizuje vše)

apt-get dist upgrade nazev - update

apt-get autoremove

apt-get autoclean - smaže všechny nepotřebené nenainstalované balíky

apt-cache search název - vyhledá balíky

apt-cache show(pkg) název- ukáže info o balíku(pkg - podrobné info)

dpkg

man apt