§ **OS**

* je program, který slouží jako prostředník mezi aplikacemi a HW počítače (pohodlné používání, efektivní využití zdrojů)

§ **Dělení OS**

* 1. Podle úrovně sdílení CPU (jedno X víceprocesorový)
* 2. Podle typu interakce/požadavků na odezvu (dávkový, interaktivní)
* 3. OS reálného času (RT)
* 4. Podle velikosti HW
* 5. Podle míry distribuovanosti
* 6. Podle počtu uživatelů
* 7. Podle funkcí

§ **Komponenty systému**

* 1. Procesy, 2. Správa paměti, 3. Soubory, 4. I/O podsystém, 5. Síť, 6. Ochrana a bezpečnost, 7. Uživatelské rozhraní

§ **Struktura OS**

* **1. Monolitický OS** – jeden soubor na holém počítači
* **2. Vrstvený OS** – hierarchie procesů;

§ **Funkční hierarchie**

* problém vrstvených systémů – obtížné rozčlenit OS do striktní hierarchie vrstev, správa procesů X správa paměti

§ **Objektově orientovaná analýza**

* Systém je množina objektů, capability = odkaz na objekt + množina práv definujících operace. Jádro kontroluje přístupová práva;

§ **Uživatelská rozhraní**

* 1.Command Line Interface
* 2. Graphic user interface

§ **Napište co je PCB a jaké obsahuje položky**

* OS udržuje tabulku nazývanou tabulka procesů, každý proces v ní má položku PCB(Process Control Block)
* PCB obsahuje všechny informace, které musejí být uchovány, je-li proces přepnut ze stavu ”běžící” do ”připraven” nebo ”blokován” tak, aby bylo možné proces znovu spustit
* Konkrétní obsah PCB se liší mezi systém, ale většina obsahuje pole týkající se **správy procesů** (PID, program counter), **správy paměti** a **správy souborů**

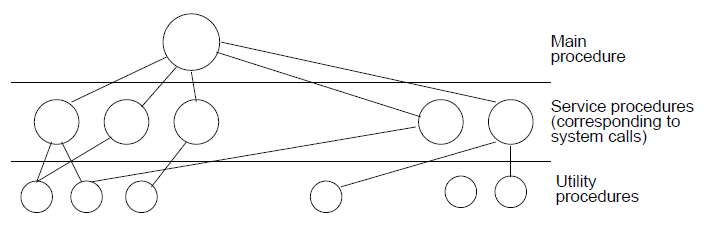
§ **Systemové volání**

* Do vybraného registru (EAX) uložím číslo služby, kterou chci vyvolat
* Do dalších registrů uložím další potřebné parametry
* Provedu instrukci
* V režimu jádra se zpracovává požadovaná služba
* Návrat, uživatelský proces pokračuje dále

**Přerušení**

* Přijde signalizace přerušení
* Dokončena rozpracovaná strojová instrukce
* Na zásobník uložena adresa následující instrukce,tj. kde jsme skončili a kde budeme chtít pokračovat !!!!!!!!
* Z vektoru přerušení zjistí adresu podprogramu pro obsluhu přerušení
* Obsluha -rychlá– Na konci stejný stav procesoru (hodnoty registrů) jako na začátku(pokud neslouží k předání výsledku)
* Instrukce návratu RET, IRET–Vyzvedne ze zásobníku návratovou adresu a na ní pokračuje !!!
* Přerušená úloha (mimo zpoždění) nepozná, že proběhla obsluha přerušení
* na zásobník uložíme návratovou adresu(CS:IP), kde budeme dále pokračovat po obsluze přerušení, přepnutí do privilegované horežimu
* na zásobník uložíme hodnoty registrů
* … provede se obsluha…
* ze zásobníku vybereme hodnoty registrů
* ze zásobníku vybereme návratovou adresu (CS:IP)určující instrukci, kde bude náš proces pokračovat a přepnutí do uživatelského (původního)režimu

**Monolitické jádro**

* Main procedure–vstupní bod jádra, na základě čísla služby (např. v EAX) zavolá servisní proceduru
* Service procedure –odpovídá jednotlivým systémovým voláním (zobrazení řetězce, čtení ze souboru, vytvoření procesu aj.)
* Service procedure volá pro splnění svých cílů různé pomocné utility procedures (lze je opakovaně využít v různých voláních)
* 

§ **Druhy přerušení**

* Hardwarové přerušení (vnejší)
  + Asynchronní událost –uživatel stiskne klávesu, kdy se mu zachce–
* Vnitřní přerušení
  + Vyvolá je sám processor–Např. pokus o dělení nulou, výpadek stránky paměti (!!)
* Softwarové přerušení
  + Je synchronní, vyvolané záměrně programem (chce službu OS)volání služeb operačního systému z běžícího procesu (!!)uživatelská úloha nemůže sama skočit do prostoru jádra OS, ale má právě k tomu softwarové přerušení

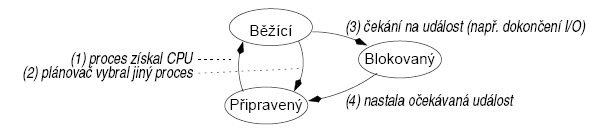
**Vyvolání HW přerušení**

* přerušení I/O zařízení signalizuje přerušení (něco potřebuji)
* Přerušení přijde na nějaké lince přerušení (IRQ, můžeme si představit jeden drát ke klávesnici, jiný drát k sériovému portu, další k časovači atd.)
* Víme číslo drátu (např. IRQ 1), ale potřebujeme vědět, na jaké adrese začíná obslužný program přerušení
* Kdo to ví? … tabulka vektorů přerušení
* Vektor přerušení je vlastně index do pole, obsahující adresu obslužné rutiny, vykonané při daném typu přerušení

**Tabulka vektorů přerušení**

* Od adresy 0 do adresy 1KB v RAM
* 256 x 4bytový ukazatel
* Ukazatel –adresa obslužného programu pro dané přerušení

§ **Proces**

* Program, který běží, má adresový prostor s kódem spustitelného programu, data, zásobník a s procesem sdružené registry
* 3 základní stavy procesu - 1.běžící, 2.připraven, 3.blokován
* 
* Proces–instance běžícího programu
  + Adresní prostor procesu
  + Proces používá typicky virtuální paměť (od 0 do nějaké adresy), která se mapuje do fyzické paměti (RAM –paměťové čipy)
  + MMU(Memory Management Unit) zajištuje mapování a tedy i soukromí procesu
  + kód spustitelného programu, data, zásobník
  + S procesem sdruženy registry a další info potřebné k běhu procesu = stavové informace
  + registry–čítač instrukcí PC, ukazatel zásobníku SP,univerzální registry
* Stavy procesů
  + Zombie
    - Proces dokončil svůj kód
    - Stále má záznam v tabulce procesů
    - Čekání, dokud rodič nepřečte exit status (voláním wait() ); příkaz ps zobrazuje stav “Z“
  + Sirotek
    - Jeho kód stále běží, ale skončil rodičovský proces
    - Adoptován procesem init

§ **Planovani procesu**

* **stavy procesů** - běžící, připraven, blokován, nový, ukončený
* Krátkodobé – CPU scheduling kterému z připravených procesů bude přidělen procesor; vždy ve víceúlohovém
* Střednědobé – swap out odsun procesu z vnitřní paměti na disk
* Dlouhodobé – job scheduling výběr, která úloha bude spuštěna dávkové zpracování (dostatek zdrojů – spusť proces)
* **charakteristiky plánovače**
  + 1. Rozhodovací mód - specifikuje časové okamžiky, ve kterých jsou vyhodnoceny priority procesu, dva typy plánovaní
    - *nepreemptivní* - proces bezi až do konce nebo do zablokovaní
    - *preemptivní* - proces může byt pozastaven;
      * problém kritických sekcí
      * časovač
  + 2. Prioritní funkce - funkce parametru procesu a systémových parametru, určující prioritu procesu v systému. Odvozeny od času, zatížení systému, paměťových požadavků
  + 3.Rozhodovaci pravidlo-řeší konflikty mezi procesy se stejnou prioritou
* CPU vázaný proces
* IO vázaný proces
* Cíle plánování (Dávkové systémy)
  + Propustnost(Throughput)
    - maximalizovat počet jobů za hodinu
  + Doba obrátky (Turnaroundtime)
    - minimalizovat čas mezi přijetím úlohy do systému a jejím dokončením
  + CPU využití
    - snaha mít CPU pořád vytížené
* **Plánovač**
  + rozhodovací mód
    - okamžik, kdy jsou vyhodnoceny priority procesu a vybrán proces pro běh
      * preemptivní a nepreemptivní
  + prioritní funkce
    - určí prioritu procesu v systému
      * dvě složky - statická a dynamická
      * priorita = statická+ dynamická
  + rozhodovací pravidla
    - jak rozhodnout při stejné prioritě
* Cíle plánovacích algoritmů
  + Každý algoritmus nutně upřednostňuje nějakou třídu úloh na úkor ostatních.
  + dávkové systémy–dlouhý čas na CPU, omezí se přepínání úloh
  + interaktivní systémy–Interakci s uživatelem, tj. I/O úlohy
  + systémy reálného času–Dodržení deadlines

**Plánovač x dispatcher**

* Dispatcher modul, který předá řízení CPU procesu vybraným short-term plánovačem
* Provádí
  + –přepnutí kontextu
  + –přepnutí do user modu
  + –skok na vhodnou instrukci daného programu, aby pokračovalo jeho vykonávání
* Plánovač určí, který proces (vlákno) by měl běžet nyní.
* Dispatcher provede vlastní přepnutí z aktuálního běžícího procesu na nově vybraný proces.

**fork–systémové volání pro vytvoření procesu(!!!!)**

* Vytvoří identickou kopii (klon) původního procesu
* Nový proces vykonává stejný kód (!!)
* Nový proces má jiný PID
* Návratová hodnota fork(!!)
  + rodič -nenulová hodnota (PID potomka)
  + potomek –nula (signalizuje, že je potomek)

§ **Precedenční grafy**

* grafy se hodí pro popis – konstrukce pro vytváření/rušení procesu musí být schopna vyjádřit různé precedenční relace mezi procesy

§ **Cobegin X Coend**

* explicitně specifikuji sekvenci programu, která má být spuštěna paralelně

§ **Explicitní deklarace podprocesu**

* 1. Statická – klíčové slovo „process“ označuje segment kódu mezi begin a end jako samostatnou jednotku behu
* 2. Dynamická – místo „process“ je „process type“- instance mohou být vytvářeny dynamicky za běhu pomoci „new“;
* **Časový souběh** muže nastat ve 2 případech – při přidávání prvku do seznamu, a pokud 2 procesy chtějí vytvořit soubor a zapsat do něj;

**Vlákna**

* Vlákna v procesu sdílejí adresní prostor, otevřené soubory (atributy procesu)
* Vlákna mají vlastní
  + Zásobník
  + Registry
  + Plánovací vlastnosti (policy, priority)
  + Množina pending a blokovaných signálů
  + Data specifická pro vlákno
* Mohou mít soukromé lokální proměnné
* Původně využívána zejména pro VT výpočty na multiprocesorech (každé vlákno vlastní CPU, společná data)
* Vlákna mohou být implementována:
  + V jádře
  + V uživatelském prostoru
  + Kombinace
* Modely vlákna
  + one-to-one (1:1) .. vlákna v jádře
    - Každé vlákno – separátní “proces” v jádře
    - Plánovač jádra je plánuje jako běžné procesy
    - Základní jednotkou plánování jsou vlákna
    - Linux clone()
  + many-to-one (M:1) .. vlákna jen v user space
    - User level plánovač vláken
    - Z pohledu jádra – vlákna 1 procesu jako pouze 1 proces
  + many-to-many (M:N)
    - Komerční unixy (Solaris, Digital Unix, IRIX)

**Bariéry**

* Synchronizační mechanismus pro skupiny procesů
* Použití ve vědecko-technických výpočtech
* Aplikace –skládá se z fází
* Žádný proces nesmí do následující fáze dokud všechny procesy nedokončily fázi předchozí
* Na konci každé fáze –synchronizace na bariéře
* Volajícího pozastaví
* Dokud všechny procesy také nezavolají barrier
* Všechny procesy opustí bariéru současně

**PTHREADS**

* - management vláken (create, detach, join)
* - mutexy (create, destroy, lock, unlock)
* - podmínkové proměnné (create, destroy, wait, signal)
* - další synchronizace (read-write locks, bariéry)

Vlákna

* - pthread\_create(&a, NULL, pocitej, NULL) - vytvoří nové vlákno
* - pthread\_join(a, NULL) - čekání na dokončení jiného vlákna
* - pthread\_detach - na vlákno nejde čekat joinem
* - pthread\_exit - vlákno skončí, když doběhne funkce, kterou vykonává

Mutex

* pthread\_mutex\_tm = PTHREAD\_MUTEX\_INITIALIZER;
  + Inicializace mutextu
  + Implicitněje odemčený
* pthread\_mutex\_destroy(&m)
  + Zrušení mutexu
* pthread\_mutex\_lock(&m)
  + Pokusíse zamknout mutex.
  + Pokud je mutex již zamčený, je volající vlákno zablokováno.
* pthread\_mutex\_unlock(&m)
  + Odemkne mutex
* pthread\_mutex\_trylock(&m)
  + Pokusí se zamknout mutex.
  + Pokud je mutex již zamčený, vrátí se okamžitě s kódem EBUSY

**Plánování vláken**

* Vlákna plánována OS
  + Stejné mechanismy a algoritmy jako pro plánování procesů
  + Často plánována bez ohledu, kterému procesu patří(proces 10 vláken, každé obdrží časové kvantum)
* Vlákna plánována uvnitř procesu
  + Běží v rámci času, který je přidělen procesu
  + Přepínání mezi vlákny –systémová knihovna
  + Pokud OS neposkytuje procesu pravidelné “přerušení“, tak pouze nepreemtivn íplánování
  + Obvykle algoritmus RR nebo prioritní plánování
  + Menší režie oproti kernel-level threads, menší možnosti

§ **Programové konstrukce pro vytváření vláken**

* Statické - obsahuje deklaraci pevné množiny pod procesu
* dynamické – procesy mohou vytvářet potomky dynamicky

**Kritická sekce (critical section, region)**

* místo v programu, kde je prováděn přístup ke společným datům

§ **Jak vznikne časový souběh**

* Pokud procesy sdílejí společnou paměť, kterou čtou a zapisují - race condition
* Pravidla pro řešení časového souběhu
  + Vzájemné vyloučení - žádné dva procesy nesmějí být současně uvnitř své kritické sekce
  + Proces běžící mimo kritickou sekci nesmí blokovat jiné procesy (např. jim bránit ve vstupu do kritické sekce)
  + Žádný proces nesmí na vstup do své kritické sekce čekat nekonečně dlouho (jiný vstupuje opakovaně, neumí se dohodnout v konečném čase, kdo vstoupí první)
* Možnosti řešení
  + Zákaz přerušení
    - vadí nám přeplánování procesů
    - zakaž přerušení;
    - kritická sekce;
    - povol přerušení;
  + Aktivní čekání
    - Průběžné testování proměnné ve smyčce, dokud nenabude očekávanou hodnotu
    - plýtvá časem CPU
  + Zablokování procesu

§ **Bariery**

* synchronizační mechanismus pro skupiny procesu. Skládá se z fázi, žádný další proces nesmí do další fáze, dokud není dokončena fáze předchozí;

§ **Semafor**

* proměnná, obsahuje nezáporné celé číslo, lze mu přiřadit hodnotu pouze při deklaraci, nad semafory pouze operace P(S) a V(S) – atomické akce, tj. jakmile započne operace nad semaforem, nikdo jiný k němu nemůže přistoupit
* P(S); KS; V(S)
* C
  + sem\_t s;
  + sem\_init(&s, 0, 1); // incializuje na 1
  + sem\_wait(&s); // operace P()
  + // kritická sekce
  + sem\_post(&s); // operace V()
  + sem\_destroy(&s);
* JAVA
  + Semaphore s1 = new Semaphore(1);
  + s1.acquire();
  + // kritická sekce
  + s1.release();

**Zámek**

* třída java.util.concurrent.locks.Lock
* Lock m= new ReentrantLock();
* m.lock();
* //kritická sekce
* m.unlock();

**TSL**

* většina počítačů – instrukci, která otestuje hodnotu a nastaví paměťové místo v jedné nedělitelné operaci
* Spin-lock s instrukcí TSL - proměnná ”zámek” - na počátku 0 – proces, který chce vstoupit do KS otestuje -> pokud 0 nastaví na 1 a vstoupí do KS. Pokud 1 čeká
* **Problém časového souběhu** - jeden proces přečte, vidí 0. Druhý proces je naplánován, přečte, vidí 0, nastaví na 1, vstoupí do KS. Po naplánování první zapíše 1, a máme dva procesy v KS => řešení vyžaduje {HW podporu}. Většina současných počítačů má instrukci, která otestuje hodnotu a nastaví paměťové místo v jedné nedělitelné operaci;

§ **Mutexy**

* Pamětové zámky - zajištují vzájemné vyloučení, efektivnší
* Mutexy C
  + #include <pthread.h>
  + pthread\_mutex\_t m;
  + pthread\_mutex\_lock(&m);
  + // kritická sekce
  + pthread\_mutex\_unlock(&m);

§ **Monitor**

* Vysokoúrovňové synchronizační primitivum
* na rozdíl od semaforů je jazykové konstrukce
* je speciální typ modulu, ve kterém jsou sdružena data a procedur, které s nimi mohou manipulovat
* procesy mohou volat proces monitoru, ale nemohou přímo přistupovat k datům monitoru
* v monitoru může být v jednu chvíli aktivní pouze 1 proces, ostatní jsou při pokusu o vstup do monitoru pozastaveny.
* Voláni yield - šetření CPU
* C mutex + podmínková proměnná
  + pthread\_mutex\_t zamek= PTHREAD\_MUTEX\_INITIALIZER;
  + pthread\_cond\_tc1= PTHREAD\_COND\_INITIALIZER;
  + pthread\_mutex\_lock( &zamek);
    - pthread\_cond\_wait( &c1, &zamek);
    - pthread\_cond\_signal( &c1);
  + pthread\_mutex\_unlock( &zamek);

**Operace nad podmínkami**

* C.wait
  + Volající bude pozastaven nad podmínkou C
  + Pokud je některý proces připraven vstoupit do monitoru, bude mu to dovoleno
* C.signal
  + Pokud existuje 1 a více procesů pozastavených nad podmínkou C, reaktivuje jeden z pozastavených procesů, tj. bude mu dovoleno pokračovat v běhu uvnitř monitoru
  + Pokud nad podmínkou nespí žádný proces,nedělá nic
* Řešení reakce na signál
  + Hoare
    - proces volající c.signal se pozastaví
    - Vzbudí se až poté co předchozí rektivovaný proces opustí monitor nebo se pozastaví
  + Hansen
    - Signal smí být uveden pouze jako poslední příkaz v monitoru
    - Po volání signal musí proces opustit monitor

§ **Semafory pomocí zpráv**

* zavedeme pomocný synchronizační proces, který bude pro každý semafor udržovat čítač a seznam blokovaných procesů
* operace P a V implementovat jako funkce, které provedou odeslání požadavku a poté čekají na odpověď pomocí receive
* synchronizační proces obsluhuje v jednom čase 1 požadavek, čímž je zaručena podmínka vzájemného vyloučení
* pokud synchronizační proces obdrží požadavek na operaci P a semafor > 0, odpoví ihned. Jinak neodpoví, čímž volajícího zablokuje
* pokud synchronizační proces obdrží požadavek na operaci V a je blokovaný proces, jednomu blokovanému procesu odpoví, čímž ho vzbudí.

§ **Uvíznutí**

* Večeřící filozofové, všichni vezmou levou, pravá už není.
  + Život filozofa –jí a přemýšlí
  + Když dostane hlad
    - Pokusí se vzít si dvě vidličky
    - Uspěje –nějakou dobu jí, pak položí vidličky a pokračuje v přemýšlení
  + Popis chyby:
    - Všichni filozofové zvednou najednou levou vidličku, žádný z nich už nemůže pokračovat,dojde k deadlocku
  + Deadlock:
    - cyklické čekání dvou či více procesů na událost, kterou může vyvolat pouze některý z nich, nikdy k tomu však nedojde
  + Pokud by filozofové vzali najednou levou vidličku,budou běžet cyklicky (vidí, že pravá není volná, položí..)
  + Vyhladovění(starvation)
    - proces se nedostane k požadovaným zdrojům
* Podmíněné uvíznutí
  + vzájemné vyloučení - Každý zdroj je buď dostupný, nebo je výhradně přiřazen právě jednomu procesu
  + hold and wait - Proces držící výhradně přiřazené zdroje může požadovat další zdroje
  + nemožnost odejmutí - Jednou přiřazené zdroje nemohou být procesu násilně odejmuty (proces je musí sám uvolnit)
  + cyklické čekání - Musí být cyklický řetězec 2 nebo více procesů, kde každý z nich čeká na zdroj držený dalším členem
* Všechny 4 podmínky musejí být splněny
* 4 strategie zacházení s uvíznutím
  + 1. Ignorovaní - pštrosí algoritmus
    - Předstíráme, že problém neexistuje
    - Vysoká cena za eliminaci uvíznutí
    - U uživatelských procesů uvíznutí neřešíme, snažíme se, aby k uvíznutí nedošlo v jádře OS
  + 2. Detekce a zotavení - detekuje se cyklus a zotavuje se pomoci preempce nebo pomoci zrušení změn nebo pomoci zrušení procesu
    - Systém se nesnaží zabránit vzniku uvíznutí
    - Detekuje uvíznutí
    - Pokud nastane, provede akci pro zotavení
    - Zotavení pomocí preempce
      * Vlastníkovi zdroj dočasně odejmout
      * Závisí na typu zdroje –často obtížné či nemožné
      * Tiskárna –po dotištění stránky proces zastavit, ručně vyjmout již vytištěné stránky, odejmout procesu a přiřadit jinému
    - Zotavení pomocí zrušení změn (rollback)
      * Častá uvíznutí – checkpointing procesů
      * Detekce uvíznutí–nastavení na dřívější checkpoint, kdy proces ještě zdroje nevlastnil (následná práce ztracena)
    - Zotavení pomocí zrušení procesu
      * Nejhorší způsob –zrušíme jeden nebo více procesů
      * Zrušit proces v cyklu
        + Pokud nepomůže zrušit jeden, zrušíme i další
  + 3. Dynamické zabránění - systém rozhodne, zda přirazení zdroje procesu je bezpečné, zda existuje posloupnost, ve které procesy mohou doběhnout
    - Ve většině systémů procesy žádají o zdroje po jednom
    - Systém rozhodne, zda je přiřazení zdroje bezpečné, nebo hrozí uvíznutí
    - Pokud bezpečné–zdroj přiřadí, jinak pozastaví žádající proces
    - Stav je bezpečný, pokud existuje alespoň jedna posloupnost, ve které mohou procesy doběhnout bez uvíznutí
    - I když stav není bezpečný, uvíznutí nemusí nutně nastat
  + 4. Prevence, pomocí strukturální negace jedné z dříve uvedených nutných podmínek pro vznik uvíznutí - Coffman
    - vzájemné vyloučení–výhradní přiřazování zdrojů
      * prevence –zdroj nikdy nepřiřadit výhradně
      * problém lze řešit pro některé zdroje(tiskárna)
      * spooling
    - hold and wait–proces držící zdroje může požadovat další
      * proces držící výhradně přiřazené zdroje může požadovat další zdroje
      * požadovat, aby procesy alokovaly všechny zdroje před svým spouštěním
      * Modifikace:pokud proces požaduje nové zdroje, musí uvolnit zdroje které drží a o všechny požádat v jediném požadavku
    - nemožnost zdroje odejmout
      * odejímat zdroje je velmi obtížné
      * Proces může zanechat zdroj v nekonzistentním stavu
    - cyklické čekání
      * Proces může mít jediný zdroj, pokud chce jiný, musí předchozí uvolnit –restriktivní, není řešení
      * Všechny zdroje očíslovány, požadavky musejí být prováděny v číselném pořadí

§ **Problém producent / konzument**

* dva procesy mají společnou paměť pevné velikosti N položek, jeden je „producent“ generuje nové položky a paralelně s ním běží „konzument“, který data vyjímá a spotřebovává;

§ **Čtenáři X písaři**

* první čtenář dostane přístup do databáze, provede P(w)
* další pouze zvětšuji čítač rc
* po skončení čtenáři zmenšují rc. Poslední provede v(w)
* semafor w zabraní vstupu písaře, pokud jsou čtenáři semafor w zabraní vstupu čtenářům, je-li písař
* čtenáři mají přednost před písaři

§ **Vlastnosti systémů založených na mikrojádře**

* model klient – server. Většinu činností OS vykonávají samostatné procesy mimo jádro (servery, např.systém souborů)
* mikrojádro = vrstva nad holým strojem, poskytuje pouze nejdůležitější nízkoúrovňové funkce potřebné pro implementaci OS (nízkoúrovňovou správu procesů, adresový prostor, komunikaci mezi adresními prostory, někdy obsluhu přerušení a vstupy/výstupy)
* pouze mikrojádro v privilegovaném režimu
  + **výhody** vynucuje si modulárnější strukturu, snadnější tvorba distributivního OS (klienti a servery mohou komunikovat po síti)
  + **nevýhody** - obecně složitější návrh systému, režie

**Meziprocesorová komunikace**

* Přes sdílenou paměť (předpoklad: procesy na stejném uzlu)
* Zasíláním zpráv (na stejném uzlu i na různých uzlech)
* Problém
  + Někdy není vhodné
    - Bezpečnost–globální data přístupná kterémukoliv procesu bez ohledu na semafor
  + Někdy není možné
    - Procesy běží na různých strojích, komunikují spolu po síti
* send(adresát, zpráva)-odeslání zprávy
* receive(odesilatel, zpráva)-příjem zprávy

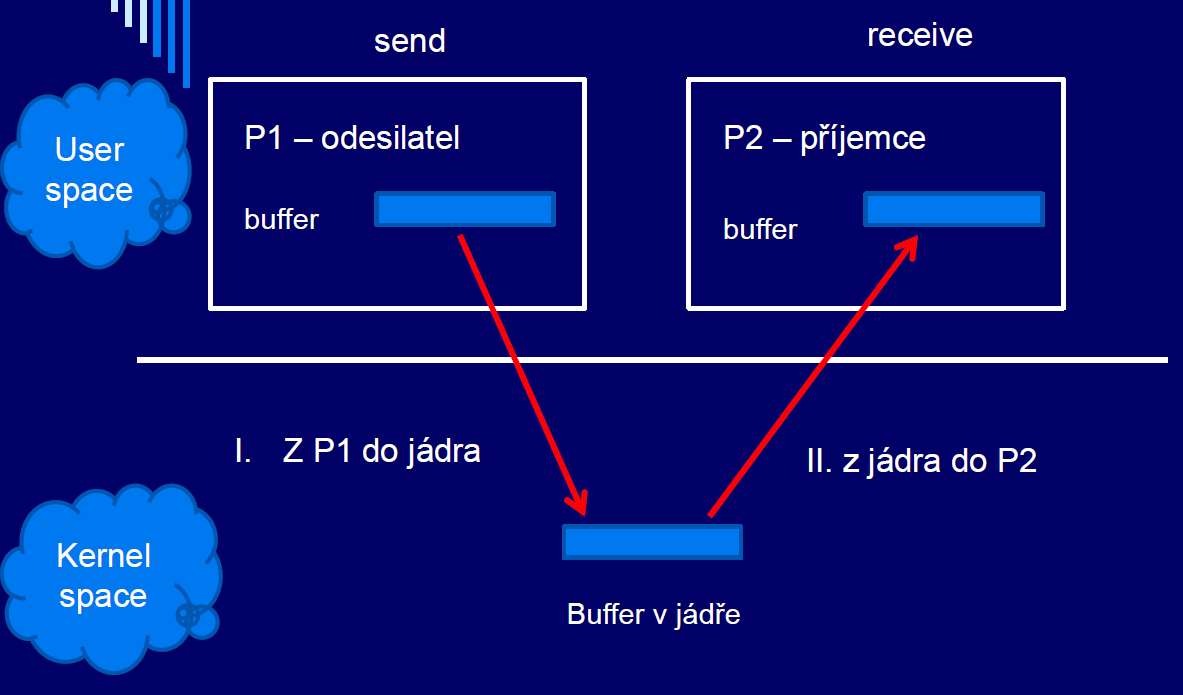
**Synchronizace zpráv**

* blokující (synchronní)
  + Volání se zablokuje, dokud požadovaná událost nenastane -typicky receive čeká na zprávu
* neblokující (asynchronní)
  + Volání hned pokračuje dále –typicky send předá zprávu jádru a dál se nestará

**Skupinové a všesměrové adresování**

* skupinové adresování (multicast)
* všesměrové vysílání (broadcast)

**Lokální komunikace**

* Na stejném stroji –snížení režiena zprávy
* Dvojí kopírování(!)
  + z procesu odesílatele do fronty v jádře
  + z jádra do procesu příjemce
  + 
* rendezvous
  + eliminuje frontu zpráv
  + např. sendzavolán dříve než receive–odesílatel zablokován
  + až vyvoláno obojí, sendi receive–zprávu zkopírovat z odesílatele přímodo příjemce
  + efektivnější, ale méně obecné(např. jazyk ADA)
* využití mechanismu virtuální paměti (!)
  + paměť obsahující zprávu je přemapována
  + zpráva se nekopíruje
  + Virtuální paměť umí “čarovat“ komu daný kus fyzické paměti namapuje a na jaké adresy

§ **Mailbox a port**

* **Mailbox** - fronta zpráv, která může byt využívána vice odesilateli a příjemci
* **Port** - omezená forma mailboxu, ze které může přijímat zprávy jen jeden příjemce

§ **RPC - komunikace mezi klientem a serverem při vzdáleném volání funkce**

§ dovolit procesům (klientům) volat procedury umístěné v jiném procesu (serveru)

§ průběh komunikace: **1**. Klient zavolá spojku klienta, reprezentující požadovanou vzdálenou proceduru;

**2**. Spojková procedura argumenty zabalí do zprávy a zprávu pošle serveru

**3**. Spojka serveru zprávu přijme, vyjme z ní argumenty a zavolá proceduru

**4**. Procedura se vrátí, spojka serveru pošle návratovou hodnotu zpět klientovi

**5**. Spojka klienta přijme zprávu obsahující návratovou hodnotu a návratovou hodnotu vrátí volajícímu

§ **Definujte OS reálného času**

* OS reálného času (real time, RT). Konkrétní aplikace mají přísné požadavky na čas odpovědi (řídící počítače, multimédia).
* OS, které se těmto požadavkům podřizují = OS reálného času. V řídících aplikacích často dobře definované, časově ohraničené požadavky na čas odpovědi (jinak problém: co kdyby ”zastav motor výtahu” přišlo pozdě?), jiným aplikacím stačí ”nejlepší snaha systému”
* **soft real time** – RT úlohy mají prioritu nad vším ostatním, ale nezaručuje odezvu v ohraničeném case, použiti - multimédia, virtuální realita
* **hard real time** – zaručuje odezvu systému v ohraničeném case – všechna zpoždění a režie systému ohraničeny – omezeni funkčnosti OS, použití: řízení výroby, robotika, telekomunikace;
* Vlastnosti
  + Malá velikostOS –>omezená funkčnost
  + Snaha spustit RT proces co nejrychleji
  + Multitasking + meziprocesová komunikace(semafory, signály, události)
  + Primitiva pro zdržení procesu o zadaný čas, čítače časových intervalů

**Plánovací algoritmy v RT**

* Statické nebo dynamické
* Statické
  + Plánovací rozhodnutí před spuštěnímsystému
  + Předpokládá dostatek informací o vlastnostech procesů
* Dynamické
  + Za běhu
  + Některé algoritmy provedou analýzu plánovatelnosti, nový proces přijat pouze pokud je výsledek plánovatelný

§ **Dávkové systémy**

* nejsou interaktivní uživatelé
* omezení přepínaní úloh, vyšší efektivita

§ **Interaktivní systémy**

* nutné přepínaní mezi procesy

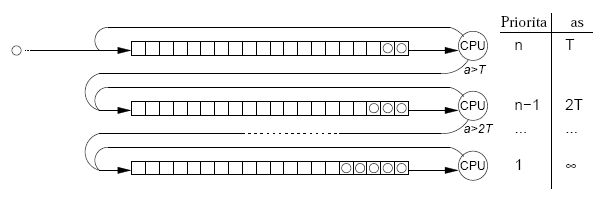
§ **Systémy reálného času**

* procesy po spuštění rychle obslouží požadavek a zablokuji se

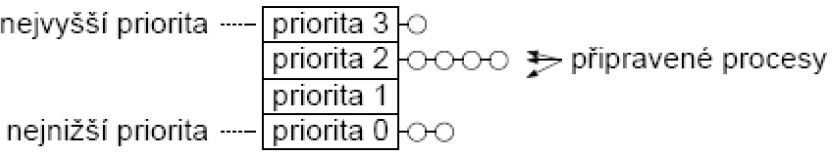
§ **cíle plánovacího algoritmu**

* spravedlivost, vynucovat stanovená pravidla, zaměstnávat všechny části systému, efektivita;

§ **Plánovaní procesu v dávkových systémech**

* **First-come First-Served (FCFS)**
  + zpracovaní podle casu příchodu úlohy
  + nepreemptivní FCFS je nejjednodušší plánovací algoritmus
  + jedna fronta, nově příchozí se řadí na konec
  + spravedlivý
* **Shortest Job First (SJF)**
  + předpoklad, že známe dobu vykonávání úlohy
  + nepreemptivní, jedna fronta příchozích úloh
  + optimalizuje dobu obrátky
* **Shortest Remaining Time (SRT)**
  + preemptivní varianta SJF
  + běžící 10 min, nová 1 min, stará se pozastaví, vykoná se nová, dokončí se stara
  + Možnost vyhladovění dlouhých úloh (!) => neustále předbíhánykrátkými
* **Multilevel Feedback**
  + na rozdílných prioritních úrovních, každá úroveň má svoji frontu úloh
  + úloha vstoupí s nejvyšší prioritou, na každé úrovni stanoveno max. času CPU, procesor obsluhuje nejvyšší neprázdnou frontu
  + Výhoda –rozlišuje mezi I/O-vázanými a CPU-vázanými úlohami
  + Upřednosťňuje I/Ovázané –déle se drží ve vysokých frontách
  + 

§ **Plánovaní procesu v interaktivních systémech**

* **Algoritmus cyklické obsluhy (RR)**
  + každému procesu přirazeno časové kvantum, po které může běžet. Pokud běží ještě na konci kvanta, je provedena preempce a spuštěn další proces
* **Prioritní plánovaní**
  + na rozdíl od RR nejsou všechny procesy stejně důležité, mají jinou prioritu, ta může byt přirazena staticky při startu procesu, nebo dynamicky (podle chovaní při běhu). Výsledná priorita je součet statické a dynamické priority.
  + plánovač snižuje dynamickou prioritu běžícího procesu při každém tiku časovače; klesne pod prioritu jiného-přeplánování
  + Dynamická priorita
    - dynamická priorita např. dle vzorce 1 /f (!)
    - f–velikost části kvanta, kterou proces naposledy použil
  + Prioritní třídy
  + 
* **Plánovač spravedlivého sdílené**
  + přiděluje čas každému uživateli proporcionálně bez ohledu na počet procesu. N uživatelů = 1/n času
  + každý uživatel –položka g
  + obsluha přerušení –inkrementuje g uživatele, kterému patří právě běžící proces
  + jednou za sekundu rozklad: g=g/2
    - Aby odrážel chování v poslední době, vzdálená minulost nás nezajímá
  + priorita P (p,g) = p –g
  + pokud procesy uživatele využívaly CPU v poslední době –položka g je vysoká, vysoká penalizace
* **Plánovaní pomoci loterie**
  + procesy obdrží losy, plánovač jeden vybere, vítěz obdrží 1 kvantum času na CPU
  + důležitější procesy mají vice losu, spolupracující procesy si mohou předávat losy
* **Plánovaní Win2000**
  + 32 prioritních úrovní - pole 32 položek. Každá položka má ukazatel na seznam připravených procesu. Priority ve skupinách: 0 - nulování stránek, 1 - 15 obyčejné procesy, 16 - 31 systémové procesy
* **CFS Completely Fair Scheduler**
  + od verze jádra 2.6.23 do současnosti
  + red-black tree místo front
  + rovnoměrné rozdělení času procesům
  + žádný idle procesor, pokud je co dělat
  + Místo priorit se používá decay faktor (do češtiny „zahnívání“)
  + Jak rychle se zmenšuje čas pro běh tasku
  + Tasky s vysokou prioritou
    - Akumulují vruntime pomaleji, potřebují více CPU času
  + Tasky s nízkou prioritou
    - Vruntime se zvyšuje rychleji

**Kvantum**

* krátké kvantum –snižuje efektivitu (režie)
* dlouhé–zhoršuje dobu odpovědi na interaktivní požadavky
* kompromis
* pro algoritmus cyklické obsluhy obvykle 20 až 50 ms
* kvantum nemusí být konstantní
* změna podle zatížení systému
* pro algoritmy, které se lépe vypořádají s interaktivními požadavky lze kvantum delší –100 ms

**Zdroje**

* Obecný termín zdroj–zařízení, záznam, …
* přeplánovatelné (preemtable)
  + lze je odebrat procesu bez škodlivých efektů
* nepřeplánovatelné(nonpremeptable)
  + proces zhavaruje, pokud jsou mu odebrány
* Sériově využitelné zdroje
  + Proces zdroj alokuje, používá, uvolní
* Konzumovatelné zdroje
  + Např. zprávy, které produkuje jiný proces
* Práce se zdrojem
  + Žádost (request)
    - Uspokojena bezprostředně nebo proces čeká
    - Systémové volání
  + Použití (use)
    - Např. tisk na tiskárně
  + Uvolnění (release)
    - Proces uvolní zdroj
    - Systémové volání

§ **Správa pamětí**

* část OS - správce paměti - udržuje info, jaká a která část paměti je a není volna, alokuje paměť procesům
* Alokuje paměť procesům podle potřeby
  + funkce malloc v jazyce C, (newv Pascalu)
* Zařazuje paměť do volné pamět ipo uvolnění procesem
  + funkce free v jazyce C, (releasev Pascalu)
* proces požádá o alokaci n bajtů paměti funkcí ukazatel = malloc (n)
* malloc je knihovní fce alokátoru paměti (součást glibc)
* paměť je alokována z haldy(heapu) !
* alokátor se podívá, zda má volnou paměť k dispozici, když ne, požádá OS o přidělení dalších stránek paměti (systémové volání sbrk)
* proces uvolní paměť, když už ji nepotřebuje voláním free(ukazatel)

**Přístup k souboru**

* Uživatel Pepa spustí program (textový editor), který poběží jako proces p1.
* Proces bude chtít otevřít soubor ahoj.txt.
* O otevření souboru musí proces požádat operační systém systémovým volánímopen().
* Soubor ahoj.txt bude ve filesystému chráněný pomocí ACL (AccesControlList), kdo k němu smí přistoupit.
* Jádro operačního systému zkontroluje, zda jej smí Pepa otevřít, a pokud ano, soubor otevře (naplní příslušné datové struktury).

§ **Rozdíl mezi ACL tabulkou a C-seznamy**

* **ACL** s každým objektem je sdružen seznam subjektů, které mohou k objektu přistupovat (subjekt = uživatel, respektive jeho procesy). Pro každý uvedený subjekt je v ACL množina přístupových práv k objektu
* **C-seznamy** -s každým subjektem (procesem) sdružen seznam objektů, ke kterým může přistupovat a jakým způsob.(tj. přístupová práva). Seznam je nazýván ”capability list” a jednotlivé položky ”capabilities”. Struktura ”capability” má prvky typu objekt. Práva obvykle bitmapová mapa popisující dovolené operace nad objektem. Odkaz na objekt, např. číslo i-uzlu, číslo segmentu apod.

§ **Jaké jsou výhody/nevýhody souboru mapovaného do části paměťového prostoru vůči read/write/atd.**

* Výhody -rychlejší, data jsou uložena v paměti a nemusí se načítat z disku
* Nevýhody - natahovaní stránky do paměti, řešení výpadku stránky

**Jak jádro rozhodne, že má uživatel k souboru přístup?**

* Implementace volání open() zjistí:
* na kterém filesystému(fs) ahoj.txt ležíntfs, fat32, ext3, ext4, xfs, …
* zda daný fspodporuje ACL (komplexní práva) nebo základní unixová práva (vlastník, skupina, ostatní)nebožádná kontrola práv (FAT)ACL slouží ke kontrole přístupových práv
* zkontroluje, zda ACL vyhovují pro daného uživatele a daný mód otevření souboru (uid, čtení/zápis)

§ **Interní a externí fragmentace**

* **externí fragmentace** - zůstávají nepřidělené (nepřidělitelné) úseky paměti např. dynamické přidělování paměti - díry
* **interní fragmentace** - část přidělitelné oblasti je nevyužita např. stránkování. V průměru polovina poslední stránky procesu je prázdná
* **čisté segmentování** - interní ne, externí ano. Stejné problémy jako přidělování paměti po sekcích: externí fragmentace paměti, mohou zůstávat malé díry (tj. dále již prakticky nepoužitelné)
* **čisté stránkování** - interní ano, externí ne. Při stránkování vnější fragmentace nenastává, protože všechny stránky jsou přidělitelné
* **zapisování souborů na disk** - interní i externí ano

**Správa hlavní paměti**

* Ideál programátora
  + Paměť nekonečně velká, rychlá, levná
  + Zároveň persistentní (uchovává obsah po vypnutí napájení)
  + Bohužel neexistuje
* Reálný počítač –hierarchie pamětí(„pyramida“)
  + Registry CPU
  + Cache paměť -malé množství, rychlá
  + RAM paměť –4GB, 8GB dnešní PC
  + Pevné disky –1-2TB, pomalé, persistentní, SSD vs. rotační

**Alokace paměti**

* 1.proces bude chtít alokovat 500B, zavolá malloc
* 2.alokátor koukne, nemá volnou paměť, požádá OS o přidělení stránky paměti (4KB) –sbrk
* 3.proces je obsloužen, dostane paměť
* 4.proces bude chtít dalších 200B, zavolá malloc
* 5.alokátor už má paměť v zásobě, rovnou ji přidělí procesu
* 6.když už proces paměť nepotřebuje, zavolá free

**Pointer**

* ukazatel = malloc(size)
* takto získaný ukazatel obsahuje virtuální adresu, tj. není to přímo adresa do fyzické paměti (RAM) !!!
* virtuální adresa se uvnitř procesoru převede na fyzickou adresu(s využitím tabulky stránek atd.)!!

**Mechanismy správy paměti**

* Základní mechanismy
  + Program je v paměti po celou dobu svého běhu
* Mechanismy s odkládáním
  + Programy přesouvány mezi hlavní pamětí a diskem

§ **Jednoprogramové systémy bez odkládaní a stránkovaní**

* nejjednodušší, uživatel zadá příkaz, OS zavede program do paměti. Dovoluje procesu využít všechnu volnou paměť
* OS ve spodní částiadresního prostoru v RAM(minipočítače)
* OS v horní částiadresního prostoru v ROM(zapouzdřené systémy)
* OS v RAM, ovladače v ROM(na PC –MS DOS v RAM, BIOS v ROM)

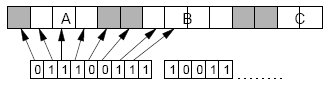
§ **Multiprogramování s pevným přidělením paměti**

* paralelní nebo pseudoparalelní běh více procesů. Rozdělení paměti na n oblasti.
* Nejjednodušší schéma –rozdělit paměť na n oblastí(i různé velikosti)
* **Strategie**
  + 1. více front, každá úloha do fronty s nejmenší oblastí
  + 2. jedna fronta - po uvolnění vybrat z fronty největší úlohu

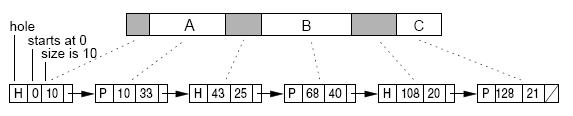
§ **Multiprogramování s proměnnou velikosti oblasti**

* každé úloze přidělena paměť podle požadavků, obsazení se s časem mění - zlepšeni využiti paměti
* V čase se mění
  + Počet oblastí
  + Velikost oblastí
  + Umístění oblastí

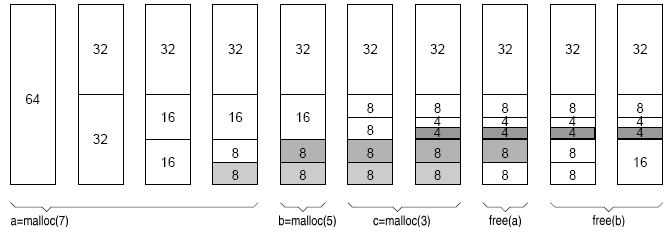
§ **Správa paměti pomoci bitových map**

* paměť rozdělena do alokačních jednotek stejné délky. S každou alokační jednotku sdružen jeden bit
* 
* Menší alokační jednotky –větší bitmapa
* Větší jednotky –více nevyužité paměti

§ **Sprava pameti pomoci seznamu**

* myšlenka udržovat seznam alokovaných a volných oblastí. Každá položka obsahuje info, zda je proces nebo díra, počáteční adresu oblasti, délku oblasti;
* 
* Proces skončí –P se nahradí H (dírou)
* Dvě H vedle sebe –sloučíse
* Alokace
  + First Fit (první vhodná)
    - Prohledávání od začátku, dokud se nenajde dostatečně velká díra
    - Díra se rozdělí na část pro proces a nepoužitou oblast (většinou „nesedne“ přesně)
    - Rychlý, prohledává co nejméně
  + NextFit (další vhodná)
    - Prohledávání začne tam, kde skončilo předchozí
    - O málo horší než firstfit
  + Bestfit(nejmenší/nejlepší vhodná)
    - Prohlédne celý seznam, vezme nejmenší díru, do které se proces vejde
    - Pomalejší –prochází celý seznam
    - Více ztracené paměti než FF,NF –zaplňuje paměť malými nepoužitelnými dírami
  + Worstfit (největší díra) –není vhodné
    - nepoužívá se
  + Quick Fit
    - Samostatné seznamy děr nejčastěji požadovaných délek
    - Díry velikosti 4KB, 8KB,…
    - Ostatní velikosti v samostatném seznamu
    - Alokace –rychlá
    - Dealokace –obtížné sdružování sousedů
* Urychlení
  + Oddělené seznamy pro proces a díry
    - Složitější a pomalejší dealokace
    - Vyplatí se při rychlé alokaci paměti pro data z I/O zařízení
  + Oddělené seznamy, seznam děr dle velikosti
    - Optimalizace best fitu
    - První vhodná –je i nejmenší vhodná, rychlost First fitu
    - Režie na dealokaci –sousední fyzické díry nemusí být sousední v seznamu

**Buddy system**

* Seznamy volných bloků 1, 2, 4, 8, 16 … alokačních jednotek až po velikost celé paměti
* Nejprve seznamy prázdné vyjma 1 položky v seznamu o velikosti paměti
* Př.: Alokační jednotka 1KB, paměť velikosti 64KB
* Seznamy 1, 2, 4, 8, 16, 32, 64 (7 seznamů)
* Požadavek se zaokrouhlí na mocninu dvou nahoru
* např. požadavek 7KB na 8KB
* Blok 64KB se rozdělí na 2 bloky 32KB (buddies)a dělíme dále…
* 

§ **Jak bude velká tabulka stránek**

* 48bitová VA, 32bitová FA a velikost stránky je 8kB.
* 8kb =2^18; 48-18 = 30; virtuální pamět 2^30; Pokud by zadal položky tabulky stránek 32b(4B), pak celková velikost=2^30\*4= 4GB

§ **NRU, LRU, SC**

* **LRU** – vyhodit nejdéle nepoužívanou stránku
* **NRU** – vyhodí stránku z nejnižší neprázdné třídy, výběr mezi stránkami ve stejné třídě je náhodný
* **SC** - podle bitu R nejstarší stránky - pokud R=0, stránka je nejstarší => vyhodíme, pokud R=1, nastavíme R na 0 a přesuneme na konec seznamu stránek (jako by byla nově zavedena)