**1. Definujte proces**



Neformálne, ***proces*** je bežiaci program, vrátane všetkých informácií potrebné pre jeho opätovnému spusteniu od inštrukcie, kde bol prerušený (s tými istými prostriedkami a s tou istou prioritou).

Stav Procesu :Nový - proces je práve vytvorený. Bežiaci - vykonávajú sa inštrukcie programu.Čakajúci - proces čaká na nejakú udalosť, napr. dokončenie V/V operácie alebo na signál. Pripravený - proces čaká na pridelenie procesora. Ukončený - proces dokončil svoje vykonanie.

*Proces* je niečo viac ako ***kód programu***, ktorý sa vykonáva (niekedy nazývaný textový segment). Proces je definovaný aj kontextom, v ktorom sa vykonáva. Do kontextu patria hodnoty ***čítača inštrukcií*** a ***registrov procesora***. Proces zahŕňa aj ***zásobník***, ktorý obsahuje dočasne dáta procesu (ako sú parametre podprogramov, návratové adresy a lokálne premenné) a ***dátový segment***, ktorý obsahuje globálne premenné.

**2. Nakreslite stavový diagram procesu**

**Stavový diagram a rozhodnutie o pridelení CPU**

pri jednom z prechodov:

1.zo stavu bežiaci do stavu čakajúci 2.z bežiaci do stavu pripravený.

3. z čakajúci do stavu pripravený.

4.keď proces končí.

Pri prechodoch v bodoch 1 a 4 nemáme možnosť výberu. Nový proces sa musí vybrať pre vykonanie.

Keď sa plánovanie vykonáva len v prípadoch 1 a 4 - jedná o *nepreemptívne* plánovanie*,* ináč plánovanie je *preemptívne.*

**3. PCB**

Každý proces je prezentovaný v operačnom systéme dátovou štruktúrou, ktorá sa nazýva *riadiaci blok procesu (Process Control Block*) - obsahuje informácií o procese, z ktorých najdôležitejšie sú:

**a) Stav procesu** - nový, pripravený, bežiaci, čakajúci atď.

**b) Hodnota čítača inštrukcií** - indikuje adresu inštrukcie, ktorá bude vykonaná ako nasledujúca

**c) Registre CPU** - akumulátory, index registre, ukazovatele zásobníkov, univerzálne registre, informácie o podmienených kódov a iné.

**d) Informácie pre plánovanie procesu** - priorita procesu, ukazovatele na fronty pre plánovanie a iné.

**e) Informácie pre správu pamäte** - hodnoty limitných a bázových registrov, tabuľku stránok alebo segmentov, podľa použitej techniky správy pamäte.

**f) V/V informácie - obsahujú zoznam V/V zariadení, ktoré sú pridelené procesu, zoznam otvorených súborov atď.**

**4. Aku ulohu riesi dlhodoby planovac a aku kratkodoby planovac?**

V dávkových systémoch do systému postupuje viac úloh, ako môže byť naraz vykonávaných. Preto sa ukladajú na disk a plánujú sa v dvoch fázach:

**Dlhodobý plánovač** vyberá z týchto procesov na disku a ukladá ich do pamäte. Pridáva procesy do frontu pripravených procesov. Pracuje v **min** Snaží sa vytvárať mix úloh , viazaných na I/ O a úloh, vyžadujúcich prevažne CPU, veľa úloh, viazaných na I/O môže viesť k dlhšiemu čakaniu a multiprogramovanie potom nemá veľký vplyv. Zodpovedný za dobrú priechodnosť systému

**Krátkodobý plánovač** vyberá z pripravených procesov v pamäti a prideľuje CPU jednému z nich. Presúva úlohy do/z frontu pripravených procesov a poskytuje im CPU alebo iné prostriedky Veľmi dôležitý v time- sharingových systémoch kvôli dobe odozvy Spolupracuje s časovačom (timer) a plánovacou politikou, aby rozhodol kedy sa prepne proces

**5. Popis vlákien**

***Vlákno (thread),*** niekedy nazývané *odľahčený proces* *(LWP - Lightweight process)* je základná jednotka pre plánovanie činnosti procesora a pozostáva z:

**- čítača inštrukcií,**

**- sady registrov,**

**- a zásobníka.**

S ostatnými „príbuznými“ vláknami zdieľa kód, dáta a prostriedky (otvorené súbory, signály, atď.).

Porovnanie s tradičnými procesmi:

Tradičný proces ako sme ho doteraz poznali, je úloha s jedným vláknom. Tvorba vlákien a prepínanie medzi nimi je „lacnejšie“ ako u tradičných procesov a ochrana pamäte nie je potrebná.

**2 VLAKNA MOZU ZDIELAT PAMAT:** **pre užívateľa** - komunikácie sú jednoduchšie, efektívnejšie, ochrana nemusí byť **pre OS -** prepínanie je efektívnejšie, mapovanie pamäte zostáva nezmenené, nie vždy sa musí vyprázdniť cache pamäť

Vlákna v mnohom fungujú obdobne ako procesy. Vlákno môže byť v stave pripravený, zablokovaný, bežiaci alebo ukončený. Obdobne len jedno vlákno v danom čase využíva procesor. V rámci procesu sa vlákna vykonávajú sekvenčne. Vlákna môžu vytvárať potomkov a môžu sa zablokovať, kým sa uskutoční systémové volanie. Kým je jedno vlákno zablokované, vykonáva sa iné. Na rozdiel od procesov vlákna nie sú od seba nezávislé. Pretože vlákna majú prístup k celému adresnému priestoru úlohy, môžu čítať a zapisovať do zásobníkov iných vlákien. Tieto štruktúry nie sú chránené pred zásahom iných vlákien. Takáto ochrana nie je potrebná, pretože vlákna patria jednému užívateľovi a sú navrhované za účelom spolupráce v rámci jednej úlohy.

**6. Vlákna. Porovnanie s procesmi. Implementácia**

Vlákna v mnohom fungujú obdobne ako procesy. Vlákno môže byť v stave pripravený, zablokovaný, bežiaci alebo ukončený. Obdobne len jedno vlákno v danom čase využíva procesor. V rámci procesu sa vlákna vykonávajú sekvenčne a každé vlákno má svoj čítač inštrukcií a zásobník. Vlákna môžu vytvárať potomkov a môžu sa zablokovať, kým sa uskutoční systémové volanie. Kým je jedno vlákno zablokované, vykonáva sa iné.

- Na rozdiel od procesov vlákna sú od seba závislé.

- Pretože vlákna majú prístup k celému adresnému priestoru úlohy, môžu čítať a zapisovať do zásobníkov iných vlákien.

- Tieto štruktúry nie sú chránené pred zásahom iných vlákien.

- Takáto ochrana nie je potrebná, pretože vlákna patria jednému užívateľovi a sú navrhované za účelom spolupráce v rámci jednej úlohy.

Implementácia :

**1. Na úrovni užívateľa/rychlejsie - vytvorenie run-time prostredia**

Run-time modul je zostavený s užívateľským programom a pri inicializácii vykonateľného programu je mu odovzdané riadenie. Taktiež všetky funkcie súvisiace s vláknami sú vykonávané pod „dohľadom“ runtime modulu. V tomto prípade sa celý preložený a zostavený program javí z pohľadu operačného systému ako jediný proces (OS o vláknach nevie).

**2. Na úrovni OS /pomalsie –** plánované OS

môže byť vykonávané paralelne s ostatnými vláknami úlohy.

**7. Napíš 2 spôsoby implementácie vlákna**

a) Na úrovni užívateľa

i. OS nevie o nich

ii. implementované pomocou knižníc

iii. plánovanie v rámci úlohy

b) Vlákna na úrovni jadra

i. OS ich pozná – jednotka pre plánovanie

ii. OS plánuje ich vykonanie

**8. Popiste algoritmy planovania procesov FCFS a RR, ich vyhody a nedostatky**

**1. Spracovanie v poradí príchodu (FCFS - First Come, First Served)** - Proces, ktorý požiadal prvý o pridelenie procesora ho dostane ako prvý. Implementácia sa uskutočňuje pomocou frontu FIFO. Keď proces vstúpi do frontu pripravených procesov, jeho riadiaci blok (PCB) sa zaradí na koniec frontu. Keď sa procesor uvoľní, pridelí sa procesu, ktorý je na čele frontu. Bežiaci proces sa odstráni z frontu.

+/- :nevhodný v time-sharingových systémoch, **nepreemtívny-**proces sa prepína do stavu zablokovaný len v dôsledku svojho vlastného správania, najjednoduchší

**2. Cyklické plánovanie (Round Robin)** - Algoritmus cyklického plánovania (Round Robin - RR) bol navrhnutý špeciálne pre timesharing-ové systémy. Je podobný algoritmu FCFS, ale je **preemptívny**. Definuje sa malý časový úsek - časové kvantum (q), ktoré je obyčajne od 10 do 100 ms. Front pripravených procesov sa spracováva ako cyklický front. Plánovač prideľuje postupne každému procesu z frontu jedno časové kvantum.

+/- : preemptívny **(/**prepínanie stavov má aj iné príčiny **predchádza kritickým situáciam a výpadkom, vykonáva opatrenia),**  výkon silne záleží od voľby časového kvanta

**2. Popíšte transformáciu logickej adresy na fyzickú pri použití segmentácie**

Logická adresa pozostáva z dvoch častí: *číslo segmentu* - *s,* a *posuv v segmente* - *d.* Číslo segmentu sa použije ako index v tabuľke. Posuv *d* logickej adresy musí byť medzi 0 a veľkosťou segmentu. Ak tomu tak nie je, vygeneruje sa prerušenie pre pokus o prístup k adrese mimo segmentu. Ak posuv je legálny, pripočíta sa k počiatočnej adrese a získa sa fyzická adresa.

Užívateľ sa môže odkazovať na objekty dvojrozmernou adresou (číslo segmentu, posuv), ale

skutočná fyzická pamäť je stále jednorozmerná postupnosť bajtov. Takže musíme mapovať

dvojrozmernú užívateľskú adresu na jednorozmernú. Toto mapovanie uskutočňujeme

pomocou tabuľky segmentov. Každá položka tabuľky segmentov obsahuje bázu a dĺžku

segmentu. Báza je počiatočná fyzická adresa segmentu a dĺžka odzrkadľuje dĺžku segmentu.

Na obr.8.23 je ukázaná situácia, kedy máme 5 segmentov. V tabuľke segmentov je položka pre každý segment, kde je zaznamenáva dĺžka a začiatočná adresa segmentu. Fyzickú adresu pre bajt 53 zo segmentu 2 spočítame tak, že k počiatočnej adrese segmentu 4300 pripočítame posuv v rámci segmentu: 4300 + 53 = 4353.



**3. Napíšte malý script, ktorý Vám vypíše procesy, ktoré vykonáva užívateľ, ktorého meno zadaváte ako parameter**

#!/bin/bash

ps –ef | grep “^$1”

**4. Popíšte semafor – štruktúra, operácie, použitie**

- Celočíselná premenná S

- Pristupuje sa k nej pomocou dvoch atomických operácií

**a) WAIT** (niekedy označovaná ako P)

1. Pokiaľ S <= 0 do čakaj;

2. S := S-1;

**b) SIGNAL** (niekedy označovaná ako V)

3. S := S+1;

- Semafor je synchronizačný prostriedok

- Semafor je abstraktný dátový typ, ktorý je charakterizovaný svojou hodnotou a operáciami, ktoré sú definované nad ním.

- Semafory sa dajú použiť pre vyriešenie problému kritickej sekcie pre *n* procesov - použitie

* Vzájomné vylúčenie
* Serializácia
* Počítanie

Semafory popísané v predchádzajúcej časti sú známe ako **počítajúce (counting) semafory**,

pretože ich hodnota sa môže meniť neobmedzene.

**Binárny semafor** nadobúda hodnoty len 0 alebo 1. Implementácia binárneho semafora je niekedy jednoduchšia ako implementácia počítajúceho semafora, podľa toho, pre akú HW platformu sa navrhuje.

**5. Zistite podľa uvedeného grafu prideľovanie prostriedkov v akom stave je systém**

1. **Nahradzovací algoritmus stránok v pamäti FIFO**

FIFO je najjednoduchší algoritmus nahradzovania stránok. Tento algoritmus priraďuje každej stránke čas jej príchodu do pamäte. Keď sa má niektorá stránka nahradiť, ako obeť sa vyberie stránka, ktorá je najdlhšie v pamäti. Pri tom nie je potrebné presne zapisovať čas, ale stačí vytvoriť FIFO front v pamäti. Potom sa nahradí stránka, ktorá je na začiatku frontu a číslo novej stránky sa pridá na koniec frontu.

**8. Ktoré metódy prideľovanie voľného priestoru súborom na disku poznáte?**

* **súvislé prideľovanie** - súbor sa ukladá postupne na susedných blokoch na disku
* **zreťazené prideľovanie** - každý súbor je zreťazeným zoznamom blokov na disku
* **indexované prideľovanie** - každý súbor má svoj index blok, kde sú uložené adresy blokov súboru. *i-tá* položka ukazuje na *i-tý* blok súboru

Indexované prideľovanie podporuje priamy prístup a netrpí vonkajšou fragmentáciou.

Nedostatkom indexovaného prideľovania je plytvanie diskového priestoru pre

uloženie index bloku, pričom to plytvanie je väčšie ako u zreťazeného prideľovania.

**9. Ktoré metódy prístupu k súborom poznáte?**

* **Priamy** - súbor pozostáva z *logických záznamov s pevnou dĺžkou,* ktoré dovoľujú čítanie a zápis záznamov v ľubovoľnom poradí
* **Sekvenčný** - je najjednoduchšou metódou prístupu k súborom. Informácia sa sprístupňuje v poradí, záznam po zázname
* **iné**, na základe priameho – indexy ku súboru

**10. Mame disk, ktorý ma 200 stôp, očíslované od 0 po 200.**

**Momentálna pozícia ramienka je na 140 stope. Front požiadavkov je: 85, 145, 90, 175, 95, 150, 102, 180, 130**

**Spočítajte, koľko pohybov (v stopách) vykoná ramienko, ak je použitý algoritmus:**

1. **výťahu (ide najskôr hore)**

* Algoritmus výťahu (SCAN) odzrkadľuje dynamickú povahu požiadaviek. Pohyb ramienka pri použití tohoto algoritmu začína na jednom konci disku a pokračuje k druhému koncu (ako výťah) a potom naspäť, pričom obsluhuje požiadavky na stopy, ktoré sú po ceste pohybu.
* 140-130-102-95-90-85-145-150-175-180= 10+28+7+5+5+60+5+25+5=150

1. **najkratšieho presunu**

* Algoritmus najkratšieho presunu (Shortest Seek Time First - SSTF) obsluhuje najskôr z frontu požiadaviek tú požiadavku, ktorá bude požadovať najmenší pohyb vzhľadom na momentálnu pozíciu ramienka
* 140 – 145 – 150 – 130 – 102 – 95 - 90 – 85 – 175 – 180 = 5+5+20+28+7+5+5+90+5=170

1. **plánovanie podľa poradia príchodu (FCFS**)

* tento algoritmus obsluhuje požiadavky podľa poradia ich príchodu **(First Come, First Served)**.
* 140 - 85-145-90-175-95-150-102-180-130 = 55 + 60 + 55 + 85 + 80 + 55 + 48 + 78 + 50 = 566

**11. Popíšte FAT tabuľku. K čomu slúži**

Zaujímavou a efektívnou variáciou **prideľovania diskového priestoru zreťazením blokov** je použitie *tabuľky pridelenia súboru (File Allocation Table - FAT)*, ktorá bola použitá v systémoch OS/2 a MS-DOS. Jedna sekcia z disku na začiatku každej partition je oddelená pre uloženie FAT tabuľky. Tabuľka má jednu položku pre každý súbor a je indexovaná podľa čísel blokov. Tabuľka FAT sa používa ako zreťazený zoznam. Položka súboru v adresári obsahuje číslo jeho prvého bloku. Položka v tabuľke s týmto indexom obsahuje číslo ďalšieho bloku súboru atď. Tento reťazec pokračuje kým sa nepríde na koniec súboru, ktorý je označený špeciálnou hodnotou. Pridelenie nového bloku súboru spočíva v nájdení prevej položky s hodnotou 0, ktorá sa nahradí hodnotou konca súboru, a tam, kde predtým bol koniec súboru, sa zapíše adresa nového bloku.



Použitie FAT tabuľky vedie k nárastu pohybu ramienka disku, pokiaľ FAT tabuľka nie je v cache pamäti. Najskôr sa ramienko musí nastaviť v tabuľke na index požadovaného bloku a po prečítaní sa musí ešte nastaviť na ten blok.

Výhoda je, že priamy prístup je optimalizovaný, pretože adresa každého bloku sa dá prečítať v tabuľke FAT.

**12. Uveďte aspoň dva spôsoby implementácie zoznamu voľných blokov na disku**

Správa voľného priestoru na disku je potrebná, pretože je potrebné trvale udržiavať informáciu o zrušených súboroch, aby bol ich priestor znova využitý. Pre tento účel systém udržiava *zoznam voľného priestoru.* Tento zoznam obsahuje adresy všetkých voľných blokov na disku. Zoznam voľných blokov môže byť implementovaný ako:

**- bitový vektor** (Ak blok je voľný, bit má hodnotu 1, ak je pridelený, má hodnotu 0)

**- zreťazeným zoznamom** (adresy voľných blokov ako zreťazený zoznam a na určitej adrese na disku udržiavať ukazovateľ na prvý blok zoznamu)

**- zoskupenie** (Modifikácia predchádzajúcej metódy je ukladať adresy n voľných blokov do prvého bloku skupiny.)

- **zoznam obsahujúci počet voľných blokov** (obsahujúci začiatok súvislého

priestoru a počet voľných blokov v ňom. Takýto prístup je výhodný keď sa používa súvislé

prideľovanie alebo clustrovanie. Celková dĺžka zoznamu je menšia ako je u zreťazeného

zoznamu.)

**13. Ktoré zariadenie využíva spooling**

Veľmi často používaná technika pri správe tlačiarní je ***spooling*** (**s**imultaneous **p**eripheral **o**peration **o**n-**l**ine). Podstata je v tom, že keď proces požiada o tlačiareň, jeho požiadavke sa vyhovuje aj v prípade, že tlačiareň nie je voľná. Súbory, ktoré sa majú vytlačiť sa ukladajú do frontu na disk a služobný program zaisťuje ich postupnú tlač.

**14. Hlavný princíp monitora**

Monitor je abstraktný dátový typ, ktorý je charakterizovaný *zdieľanými dátami a množinou operácií, ktoré sú definované nad tými dátami.* Typu monitor pozostáva z deklarácií premenných, ktorých hodnoty definujú stav monitora a z množiny procedúr a funkcií, ktoré implementujú operácie nad monitorom.

*Monitory predstavujú vyšší stupeň abstrakcie ako semafory a sú jednoduchšie a bezpečnejšie pre použitie.*

Procesy, ktoré využívajú monitor, majú prístup len k procedúram monitora. Procedúry definované v monitore majú prístup len k premenným, ktoré sú definované vo vnútri monitora, a k formálnym parametrom. Tieto procedúry sa nesmú vzájomne volať a nesmú byť rekurzívne.

Konštrukcia monitora dovoľuje len jednému procesu byť vo vnútri, takže programátor nemusí explicitne programovať vzájomné vylúčenie vykonania jednotlivých procedúr monitora

* Monitory poskytujú dodatočný dátový typ, ktorý dovoľuje zablokovať proces v prípade, že nie je splnená určitá podmienka
* typ *condition*
* jediné operácie nad ním sú *wait* a *signal*.
* Operácia *wait(condition):*
* uvoľní uzamknutie monitora a „uspí“ proces. Keď sa proces „zobudí“, získa znova prístup k monitoru.
* Operácia *signal(condition):*
* „zobudí“ jeden z procesov čakajúcich na premennú typu *condition*. Ak vo fronte nie je žiadny proces, neurobí nič.

**15. Čím sa zaoberá pridelovanie diskového priestoru**

Disk je zariadenie pre ktoré priamy prístup ja najprirodzenejší a umožňuje veľkú flexibilitu pri

implementácii systému súborov. Hlavný problém pri tejto implementácii je efektívne

prideľovať diskový priestor a umožniť rýchle vyhľadávanie blokov súboru. Najrozšírenejšie sú

tri metódy prideľovania diskového priestoru*: súvislé prideľovanie, zreťazené prideľovanie a*

*indexované prideľovanie*. Každá metóda má svoje prednosti a nedostatky. Zvyčajne operačný

systém poskytuje len jednu metódu pre všetky súbory.

*Súvislé prideľovanie*

Pri súvislom prideľovaní diskového priestoru sa súbor ukladá postupne na susedných blokoch na disku. Diskové adresy nasledujú za sebou, čo znamená, že nie je potrebný pohyb hláv pre načítavanie ďalšieho bloku. Keď je potrebný pohyb (z posledného sektoru jedného cylindra na prvý sektor ďalšieho cylindra), ten pohyb je minimálny - len jedna stopa. To znamená, že čas pre hľadanie blokov súvisle uloženého súboru je minimálny. Súvislé uloženie súboru je definované diskovou adresou a dĺžkou prvého bloku. Prístup k súboru, ktorému bol pridelený súvislý priestor je veľmi jednoduchý. Pri sekvenčnom prístupe si systém pamätá adresu bloku, ktorý bol posledne sprístupnený, a keď je potrebné, prečíta sa ďalší blok.

Problém súvislého prideľovania diskového priestoru je podobný prideľovaniu súvislého pamäťového priestoru. Aj tu sa pri hľadaní vhodného úseku využívajú najčastejšie algoritmy first-fit a best-fit. Ani jeden z týchto algoritmov nie je lepší z hľadiska času a využitia diskového priestoru, ale first-fit je obecne rýchlejší.

Nedostatkom týchto algoritmov je vonkajšia fragmentácia. Ako sa súbory vytvárajú a rušia, diskový priestor sa postupne rozdrobí na malé kúsky. Vonkajšia fragmentácia začína byť problémom, keď aj najväčší súvislý úsek nestačí pre požadované uloženie súboru.

Problém pri rozšírovaní súborov v prípade súvislého prideľovania sa rieši tak, že na začiatku sa súboru pridelí súvislý priestor a pri potrebe rozšírenia sa pridelí dodatočný súvislý úsek, ktorý nemusí nutne ležať za prvým. Tento úsek sa nazýva rozšírenie (extent). Bloky súboru sa potom zapisujú do položky adresára ako počiatočná adresa a počet blokov plus adresa prvého bloku rozšírenia. Samozrejme problém vonkajšej fragmentácie zostáva, pričom podľa toho, akým spôsobom je vyriešené zadávanie veľkosti rozšírenia, môže byť aj vnútorná fragmentácia. Tá vzniká, keď užívateľ zadá väčšie rozmery, ako potrebuje.

**16. Mame log. adresný priestor z 8 stranok po 1024 slov, ktorý sa mapuje do fyz. priestoru s 32 rámcami. Koľko bitov ma log. adresa? Koľko bitov ma fyz. adresa?**

Fyzická adresa: vynásobíme počet rámcov (32) veľkosťou rámca (=veľkosti stránky, teda 1024 slov). Slovo je 16bitové, tj. fyzická adresa ma dĺžku 32\*1024\*(2^16).   
A s logickou adresou to imho bude tak isto, len namiesto 32 dosadíme 8.

**slovo** (word) - 16bit = 2 bajty

**17. Coffmanove podmienky - Nutné podmienky na uviaznutie**

Uviaznutie môže nastať ak sú splnené nasledovné štyri (Coffmanove) podmienky naraz

V danom čase:

**a) Vzájomné vylúčenie**. Aspoň jeden prostriedok musí byť pridelený výlučne, to znamená,   
že nemôže byť zdieľaný.

**b) Vlastniť a žiadať.** Musí existovať proces, ktorý má pridelený aspoň jeden prostriedok   
a požaduje ďalšie prostriedky, ktoré sú pridelené iným procesom.

**c) Používanie bez preempcie**. Prostriedok nemôže byť odňatý, t.j. proces môže uvoľniť prostriedok jedine dobrovoľne, keď s ním ukončí prácu.

**d) Kruhové čakanie.** Musí existovať množina *P0 , P1 . . . Pn* čakajúcich procesov takých, že *P0*čaká na prostriedok, ktorý drží *P1*, *P1*čaká na prostriedok, ktorý drží *P2, ......, Pn-1*čaká na prostriedok, ktorý drží *Pn*a*Pn*čaká na prostriedok, ktorý drží *P0* .

**18. Popis architekturu: klient-server, vrstvovu arch. OS**

Architektúra klient-server: základná myšlienka tejto architektúry spočíva v rozdelení operačného systému do niekoľkých procesov (serverov), z ktorých každý realizuje jednu sadu služieb - napr. služby pre prácu s pamäťou, služby pre vytváranie alebo plánovanie procesov a iné. Každý *server* beží v užívateľskom režime a čaká v nekonečnej slučke na požiadavky klientov. *Klient*, ktorý môže byť buď iný operačný systém alebo aplikačný program, žiada o službu tak, že pošle serveru správu. Jadro, ktoré beží v privilegovanom režime doručí správu serveru, server vykoná požadovanú operáciu a jadro vráti výsledok klientovi v inej správe.

Vrstvová architektúra OS: Hlavná výhoda viacvrstvovej architektúry systému je jeho modularita. Vrstvy sú rozdelené tak, že každá využíva funkcie (operácie) nižších vrstiev. Vrstva je implementácia abstraktného objektu, čo je vlastne zapuzdrenie dát a operácií s týmito dátami

§

**19. Popis komunikaciu medzi procesmi na základe volania vzdialenej procedury RPC**

Volanie vzdialenej procedúry je založené na mechanizme, ktorý je podobný mechanizmu volania lokálnej procedúry. Používa sa veľmi často v aplikáciách klient-server. Procesy takejto aplikácie sa vykonávajú na rôznych uzloch siete a komunikujú pomocou zasielania správ. Keď sa zavolá vzdialená procedúra, lokálny systém prenesie požiadavku a parametre volania do vzdialeného systému, ktorý vykoná procedúru. Po ukončení práce vzdialený systém posiela naspäť kód úspešnosti vykonania a výsledky.

Pre zabezpečenie komunikácie medzi procesmi, ktoré sa vykonávajú na rôznych systémoch v sieti, je potrebné správu adresovať na konkrétny port (identifikačné číslo koncového bodu spojenia), ktorý je pridelený príslušnej procedúre. Zistenie portu sa uskutočňuje buď staticky (port priradený pri preklade) alebo dynamicky (portmapper, pred nadviazaním spojenia ).

**20. Fazy spracovania programu, kedy je mozne mu priradit pamätové adresy**

- počas prekladu

- počas zavádzania

- počas vykonávania

**21. Ako je organizovany system suborov? Aké sú úlohy jednotlivych vrstiev?**

Súborový systém má viacej vrstiev. Každá vrstva používa vlastnosti nižších vrstiev, aby vytvorila nové vlastnosti, ktoré ponúka vyšším vrstvám (obr.10.11).

**Najnižšia vrstva** sú fyzické zariadenia. Nad ňou je vrstva, ktorá *riadi V/V operácie*. Tam patria ovládače zariadení a programy pre obsluhu prerušení. Ovládač zariadenia je program, ktorý „prekladá“ požiadavky systému do príkazov príslušného zariadenia.

**Vrstva *základného systému súborov*** odovzdáva generický príkaz príslušnému ovládaču zariadenia pre čítanie alebo zápis fyzického bloku na disk. Každý fyzický blok na disku je identifikovaný numerickou adresou, napr. mechanika 1, cylinder 73, plocha 2, sektor 10.

**Modul organizácie súborov pozná súbory a ich logické a fyzické bloky.** Vediac aký spôsob prideľovania diskového priestoru bol použitý a kde je súbor umiestnený na disku, tento modul prekladá logickú adresu bloku do fyzickej adresy a poskytuje ju nižšej vrstve. Logické bloky súboru sú očíslované od 0 (príp. od 1) po *N* a obyčajne číslo fyzického bloku, kde sú dáta, nie je také isté. Modul organizácie súborov zahrňuje aj správcu voľného priestoru na disku.

aplikačný program

⇓

logický systém súborov

⇓

modul organizácie súborov

⇓

základný systém súborov

⇓

riadenie vstupu/výstupu

⇓

Zariadenia

Obr.10.11 Vrstvy systému súborov

***Logický systém súborov*** poskytuje užívateľovi pohľad na systém súborov a využíva štruktúru adresárov, aby poskytol modulu organizácie súborov informácie o súboroch. Táto vrstva je zodpovedná aj za bezpečnosť súborov.

Keď aplikačný program chce vytvoriť súbor, volá vrstvu logického systému súborov. Logický systém súborov pozná štruktúru adresárov. Pri vytváraní nového súboru načíta príslušný adresár do pamäte, pridá novu položku a zapíše ho späť na disk. Potom logický systém súborov môže zavolať modul organizácie súborov, ktorý preloží logické adresy do fyzických a odovzdá ich nižším vrstvám. Keď sa zaktualizuje adresár, logický systém súborov ho použije, aby vykonal V/V operáciu. Pri otvorení súboru sa prehľadá adresár, aby sa našla položka súboru. Pre zefektívnenie prístupu k súboru sa používa tabuľka otvorených súborov (obr.10.12). Po prvom odkaze na súbor sa aplikačnému programu vráti index z tabuľky otvorených súborov, ktorý sa používa pri nasledujúcich operáciách nad súborom (v Unix-e sa nazýva *file descriptor*). Všetky zmeny položky adresára sa zapisujú do tabuľky otvorených súborov, ktorá je v pamäti. Až keď sa súbor uzatvorí, informácie o zmenách v ňom sa zapíšu na disk.

**22. Čo urobia príkazy:**

a) ps –ef | grep $USER - vypíše procesy užívateľa uloženého v premennej USER

b) cd - zmena pracovného adresára

c) PATH=$PATH - nastavenie cesty na aktuálny adresár

d) env - výpis systémových premenných – SHELL,TERM, SSH\_CLIENT,

SSH\_TTY,USER,MAIL,PATH,PWD,LOGNAME,...

e) echo $PATH - vypíše na obrazovku hodnotu premennej PATH (aktuálneho adresára)

f) cd $HOME - zmení adresár na cestu uloženú v premennej HOME (domovský adresár)

**25. Popíš cyklicke plánovanie (Round-Robin)**

Algoritmus cyklického plánovania (Round Robin - RR) bol navrhnutý špeciálne pre timesharing-ové systémy. Je preemptívny. Definuje sa **malý časový úsek - časové kvantum** (q), ktoré je obyčajne od 10 do 100 ms. Front pripravených procesov sa spracováva ako cyklický front. Plánovač prideľuje postupne každému procesu z frontu jedno časové kvantum.

Cyklické plánovanie sa implementuje tak, že front pripravených procesov je typu FIFO. Nový proces sa pridáva na jeho koniec. Plánovač vyberá proces vždy zo začiatku frontu, nastavuje časovač na 1 časové kvantum a spúšťa proces.

Ďalšia činnosť procesu môže byť nasledovná: proces môže potrebovať procesor na menší čas ako je časové kvantum a v takomto prípade uvoľní dobrovoľne procesor. Plánovač vyberie a spustí ďalší z pripravených procesov. Ak proces potrebuje čas dlhší ako je časové kvantum, po uplynutí kvanta časovač spôsobí prerušenie. Zapamätá sa kontext procesu a proces sa uloží na koniec frontu, z ktorého sa vyberie ďalší pripravený proces.

Priemerná doba čakania pri algoritme RR je niekedy dosť dlhá.

**26. Bola zadana uloha a matice z alg. Bankara. Urci, ci je podľa alg. Bankara proces v bezpečnom stave**

Algoritmus bankára dostal toto meno, pretože sa dá použiť v bankovníctve, kde banka

nesmie nikdy požičať celú svoju hotovosť, pretože to môže viesť k stavu, kedy nebude môcť

uspokojiť požiadavky svojich klientov.

Každý nový proces pri vstupe do systému musí deklarovať svoje požiadavky pre každý

typ prostriedkov. Tento počet samozrejme nesmie prekračovať celkový počet prostriedkov

systému. Systém zistí, či uspokojenie požiadaviek procesu ho nedovedie do nebezpečného

stavu. Ak tomu tak nie je, proces dostane, čo požaduje, inak musí čakať, kým iné procesy

neuvoľnia im pridelené prostriedky.

Algoritmus bankára používa nasledovné dátové štruktúry:

n je počet procesov,

m je počet typov prostriedkov v systéme,

prístupné: vektor s dĺžkou m, ktorý obsahuje počty prístupných prostriedkov z každého

typu. Ak prístupnéj]= k, to znamená, že z prostriedkov typu Rj je k dispozícii k jednotiek.

max: matica n x m definuje maximálne požiadavky každého procesu. Ak maxi,j]= k,

potom to znamená, že proces Pi môže požadovať maximálne k jednotiek z prostriedkov

typu Rj.

pridelené: matica n x m definuje počet prostriedkov každého typu, pridelených

momentálne procesu Pi . Ak pridelenéi,j]= k, potom to znamená, že proces Pi má

momentálne pridelených k jednotiek prostriedku typu Rj.

zostáva: matica n x m , ktorá označuje prostriedky, ktoré ešte musia byť pridelené

procesu. Ak zostávai,j]= k, potom to znamená, že proces Pi potrebuje ešte k prostriedkov

typu Rj, aby mohol svoju činnosť dokončiť.

**Všimnite si, že zostávai,j]= maxi,j] - pridelenéi,j.**

Tieto dátové štruktúry môžu meniť v čase svoje rozmery a hodnoty.

Aby sme zjednodušili prezentáciu algoritmu bankára, zavedieme niekoľko pravidiel

zápisu. Nech X a Y sú vektory dĺžky n. Hovoríme, že XY, vtedy a len vtedy ak XiYi

pre všetky i = 1, 2, ...n. Napr. ak X = (1, 7, 3, 2) a Y = (0, 3, 2, 1), potom Y X. Y X ak

YX a YX.

Riadky matíc pridelené a zostáva môžeme brať ako vektory a budeme ich označovať

pridelenéi a zostávai . Vektor pridelenéi špecifikuje všetky prostriedky pridelené procesu Pi a

vektor zostávai špecifikuje všetky prostriedky, ktoré proces Pi ešte potrebuje dostať do

svojho ukončenia.

**28. Ake sú spôsoby implementácie adresara**

* **lineárny zoznam**

Lineárny zoznam s ukazovateľmi na dátové bloky. Je to jednoduchý spôsob, ale prehľadávanie lineárneho zoznamu je časovo náročné (napr. pri zisťovaní, či pri vytváraní súboru už súbor s tým názvom neexistuje). Zoznam sa môže udržiavať aj v utriedenom stave; tým sa čas prehľadávania skráti, ale operácie vytvorenia a zmazania súboru budú náročnejšie.

* **hašovacia tabuľka**

**29. Popiš nahradzovanie stranok pri strankovani na žiadosť**

Viď otázku č. 42

**30. Bola dana tabuľka procesov a hodnoty: čas začiatku procesu a posledného použitia. Pri zavádzaní nového procesu, ktorý proces bude uvolneny z pamäte podľa algoritmu:**

**a) FIFO**

FIFO je najjednoduchší algoritmus nahradzovania stránok. Tento algoritmus priraďuje každej

stránke čas jej príchodu do pamäte. Keď sa má niektorá stránka nahradiť, **ako obeť sa**

**vyberie stránka, ktorá je najdlhšie v pamäti.** Pri tom nie je potrebné presne zapisovať čas, ale

**stačí vytvoriť FIFO front v pamäti**. Potom sa nahradí stránka, ktorá je na začiatku frontu a

číslo novej stránky sa pridá na koniec frontu.

1. **LRU** Least Recently Used - najdlhšie nepoužívaná

Pri použití algoritmu LRU ku každej stránke je priradený čas posledného použitia stránky. Keď

sa má nahradiť stránka, vyberá sa taká, ktorá najdlhšie nebola používaná.

**c) Modifikovaný 2. Šance**

**Algoritmus druhej šance** (Clock alebo Second chance) **je variantom algoritmu FIFO**. Využíva

tzv. **referenčný bit,** ktorý je pridaný ku každému rámcu. Tento bit sa hardvérovo nastavuje

vždy, keď sa so stránkou pracuje. Keď stránka je vybraná na odsunutie podľa algoritmu FIFO,

skúma sa jej referenčný bit. Ak je nastavený na 0, stránka sa odsunie, ak je nastavený na 1,

stránka dostáva „druhú šancu“, referenčný bit sa vynuluje a jej čas sa nastaví na momentálny

čas. A stránka sa presunie na koniec frontu.

Týmto spôsobom stránka, ktorá je stále používaná môže zostať trvalé v pamäti.

**Modifikácia -** Algoritmus druhej šance sa dá vylepšiť **použitím dvoch bitov**, ktoré sa nastavujú - jeden pri odkaze na stránku (R), druhý pri zápise na stránku (W). Keď zoberieme do úvahy všetky

možné kombinácie hodnôt tejto dvojice bitov (v poradí RW), sú tu nasledovné triedy:

00 - na stránku nebolo odkazované a nebola modifikovaná, najlepšia pre nahradenie,

01 - na stránku nebolo odkazované, ale bolo na ňu zapisované, nie je tak dobrá pre

nahradenie, lebo sa jej obsah bude musieť zapísať na disk,

10 - na stránku bolo odkazované, ale nebolo na ňu zapisované, pravdepodobne bude

v najbližšej dobe znova použitá,

11 - na stránku bolo odkazované a bolo na ňu zapisované, pravdepodobne bude

v najbližšej dobe znova použitá a jej obsah sa bude musieť zapísať na disk.

**31. Máme procesy čas vykonavania RR konstan. 10**

a) Požiadavky čas. Príchodu FCFS RR FSJ

b) 50 0 50 125 50

c) 30 5 80 90 125

d) 5 15 85 25 55

e) 25 25 110 100 95

f) 15 35 125 75 70

**33. Prostriedky na komunikáciu procesov**

* Správy (lokálne)
* zdielaná pamäť (lokálne)

- rúry (lokálne)

- RPC (po sieti)

- sockety (po sieti)

1. **Posielanie správ v distribuovanom systéme**

- RPC, RMI, sockety

- určenie portov: staticky či dynamicky (portmapper)

* vysielajúci alebo prijímajúci proces môže byť ukončený pred tým ako poslal alebo prijal správu
* správa sa môže stratiť počas prenosu vplyvom komunikačnej siete

**35. Hardverové prostriedky pre synchronizáciu aktívným čakaním**

a) Zákaz prerušenia - počas modifikácie hodnoty spoločnej premennej

b) Špeciálne inštrukcie - TSL, SWAP

**36. Stránkovanie**

Pri stránkovaní je fyzická pamäť rozdelená na časti s **pevnou veľkosťou, nazvané *rámce****.*

Logický adresný priestor procesu je rozdelený na rovnako veľké bloky, nazvané ***stránky***.

Stránka nesúvisí s logickou štruktúrou programu. Keď sa proces vykonáva, jeho stránky sa z disku presunú do voľných rámcov v operačnej pamäti.

Každá adresa vygenerovaná procesorom je rozdelená na dve časti - ***číslo stránky* *(p)*** a ***posuv* v rámci stránky *(d).*** Číslo stránky sa používa ako index v tabuľke stránok. V tabuľke stránok sú zaznamenané odpovedajúce počiatočné adresy stránok vo fyzickej pamäti. Počiatočná adresa stránky spolu s posuvom v rámci stránky vytvára skutočnú fyzickú adresu, ktorá sa posiela jednotke správy pamäte. Model stránkovania je ukázaný na obr. 8.12.

Veľkosť stránky a rámca závisia od HW. Rozmer stránky je obyčajne mocnina 2 a mení sa od 512 bajtov do 8 KB v závislosti od architektúry počítača. Výber mocniny 2 pre veľkosť stránky veľmi uľahčuje výpočet fyzickej adresy. Ak rozmer fyzického adresného priestoru je 2m  a veľkosť stránky je 2n (bajtov alebo slov), potom vyššie *m-n* bity logickej adresy určujú číslo stránky a *n* nižších bitov určuje posuv v stránke. Logická adresa má nasledovné položky:

kde *p* je index do tabuľky stránok a *d* je posuv v stránke.



**37. Algoritmy nahradzovanie stránok (je to popísané vyššie, ale ešte raz nezaškodí)**

**1. Algoritmus FIFO** - FIFO je najjednoduchší algoritmus nahradzovania stránok. Priraďuje každej stránke čas jej príchodu do pamäte. Keď sa má niektorá stránka nahradiť, ako obeť sa vyberie stránka, ktorá je najdlhšie v pamäti. Pri tom nie je potrebné presne zapisovať čas, ale stačí vytvoriť FIFO front v pamäti. Potom sa nahradí stránka, ktorá je na začiatku frontu a číslo novej stránky sa pridá na koniec frontu.

**2. Optimálny algoritmus -** Optimálny nahradzovací algoritmus je algoritmus, ktorý nahradzuje stránku, ktorá nebude potrebná najdlhšiu dobu. Použitie tohto algoritmu zaručuje najmenší možný počet výpadkov stránok pri danom pevnom počte rámcov.

**3. Algoritmus LRU - najdlhšie nepoužívaná -** Pri použití algoritmu LRU ku každej stránke je priradený čas posledného použitia stránky. Keď sa má nahradiť stránka, vyberá sa taká, ktorá najdlhšie nebola používaná.

**4. Algoritmus druhej šance –** Nastavuje sa hardvérovo bit vždy, keď sa so stránkou pracuje. Keď stránka je vybraná na odsunutie podľa algoritmu FIFO, skúma sa jej referenčný bit. Ak je nastavený na 0, stránka sa odsunie, ak je nastavený na 1, stránka dostáva „druhú šancu“, referenčný bit sa vynuluje a jej čas sa nastaví na momentálny čas. A stránka sa presunie na koniec frontu. Týmto spôsobom stránka, ktorá je stále používaná môže zostať trvalo v pamäti.

**5. Vylepšený algoritmus druhej šance -** Algoritmus druhej šance sa dá vylepšiť použitím dvoch bitov, ktoré sa nastavujú - jeden pri odkaze na stránku (R), druhý pri zápise na stránku (W). „00“ – najlepší kandidát na obeť.

**6. Aproximačný LRU pomocou dodatočných referenčných bitov –** Zaznamenáva sa regerenčný bit v pravidelných intervaloch pomocou posuvného registra pre každý rámec. Hodnota „00000000“: stránka počas posledných 8 intervalov nebola použitá a je najlepším kandidátom na obeť. Ak je ich viac s rovnakým číslom, vyberá sa stratégiou FIFO.

**38. Štruktúra adresárov**

Obyčajne sa súborový systém delí na menšie časti **- partitions**. Každý disk má aspoň jeden oddiel, čo je štruktúra nižšej úrovne, kde sa ukladajú súbory a adresáre. Niektoré systémy dovoľujú rozšíriť partition cez niekoľko diskov (obr.10.6). Takto užívateľ má možnosť organizovať svoje dáta logicky, bez ohľadu na fyzické zariadenia na ktoré sú dáta uložené.

Každý oddiel obsahuje informácie o súboroch. Tieto informácie sú uložené v adresároch zariadenia (volume table of contents). Adresár obsahuje informácie o všetkých súboroch tejto časti a to meno, umiestnenie, veľkosť a typ súboru.

Štruktúra adresára musí byť vybratá s ohľadom na operácie, ktoré sa vykonávajú nad adresármi.

Poznáme

**1. Jednoúrovňový adresár** – všetky súbory v jednom adresári

**2. Dvojúrovňový adresár** – každý užívateľ má súbory vo vlastnom adresári a systém tiež

**3. Stromová štruktúra adresára** – bežné, rušenie adresárov rekurzívne. Nedovoľuje zdieľať súbory

**4. Adresár s acyklickou štruktúrou** – dovoľuje zdieľanie súborov: duplikovať zdieľané súbory (problém s udržiavaním zmien), symbolické linky (soft linky): ako .lnk, .pif vo window$, hard linky: v unixoch: udržiava sa zoznam liniek odkazujúcich na súbor a keď je zmazaná posledná linka, počet klesne na 0 a súbor sa zmaže z disku.

**39. Algoritmy pohybu ramienka po disku**

**1. Plánovanie podľa poradia príchodu (FCFS)** - Tento algoritmus obsluhuje požiadavky podľa poradia ich príchodu (First Come, First Served).

**2. Algoritmus najkratšieho presunu -** Algoritmus najkratšieho presunu (Shortest Seek Time First - SSTF) obsluhuje najskôr z frontu požiadaviek tú požiadavku, ktorá bude požadovať najmenší pohyb vzhľadom na momentálnu pozíciu ramienka.

**3. Algoritmus výťahu -** Algoritmus výťahu (SCAN) odzrkadľuje dynamickú povahu požiadaviek. Pohyb ramienka pri použití tohoto algoritmu začína na jednom konci disku a pokračuje k druhému koncu (ako výťah) a potom naspäť, pričom obsluhuje požiadavky na stopy, ktoré sú po ceste pohybu.(+ vylepšenia tohto alg. C-SCAN – vybavuje sa len v 1 smere na ceste späť ide ramienko naprázdno, C-LOOK – ramienko nejde až na koniec ale iba po poslednú požiadavku)

**40. Preco je dolezita synchronizácia procesov, vysvetlit ake prostriedky existuju cize aktivnym a pasivnym cakanim,**

**vymenovat ich, a vybrat si jeden a ten popisat ( toto chcela dost do hlbky vysvetlit )**

Úlohou synchronizácie je zaistiť vzájomné vylúčenie procesov pri používaní zdieľaných prostriedkov.

Prakticky to znamená, že sa rýchlosti procesov musia zosúladiť tak, aby sa časy vykonania ich kritických sekcií neprekrývali.

Pri tom sa uplatňujú dva základné princípy:

Synchronizácia aktívnym čakaním znamená, že sa odsun kritickej sekcie uskutoční

vložením pomocných (obyčajne prázdnych) inštrukcií do kódu procesu.

Synchronizácia pasívnym čakaním znamená, že sa odsun kritickej sekcie uskutoční

dočasným pozastavením procesu, kým sa kritická sekcia neuvoľní.

**Poznáme synchronizačné prostriedky pre aktívne čakanie (spoločné premenné, Test and set, Swap) a pre pasívne čakanie (semafory, monitory).**

Semafor: abstraktná dátová štruktúra, skladá sa z hodnoty semafora a z frontu čakajúcich procesov (resp. ich PCB), implementovaným zväčša frontom FIFO.

Pred vstupom do KS sa zavolá wait(mutex) a po výstupe z nej signal(mutex). Na začiatku má semafor hodnotu 1. Proces dá wait-om najavo svoj záujem vstúpiť do KS. Zníži teda hodnotu semaforu o 1 (teda jeho nová hodnota je 0) a vstúpi do KS. Od toho okamihu, ak sa iný proces pokúsi vstúpiť do svojej KS, pred čím zavolá wait, tak zistí, že hodnota semafora je 0 a tak sa zaradí do zoznamu čakajúcich procesov semafora a zablokuje sa. Keď pôvodný proces zavolá signal, dáva tým najavo, že už skončil prácu v KS. Zvýši hodnotu semafora o 1 a pokiaľ nová hodnota semafora nie je 1 (tj. ešte sú vo fronte nejaké čakajúce procesy), tak sa vyberie z frontu jeden z čakajúcich procesov, podľa stratégie FIFO a odblokuje sa, čím môže vstúpiť do svojej KS.

**41. Komunikácia medzi procesmi. Jeden typ popísať aj do podrobna.**

**1. správy**

**-** Priama komunikácia

- Nepriama komunikácia

- Buffrovanie

**2. zdieľaná pamäť -** Zdieľaná pamäť poskytuje najrýchlejšiu komunikáciu medzi procesmi. Ten istý pamäťový segment je mapovaný do adresných priestorov dvoch alebo viacerých procesov. Ihneď ako sú dáta zapísané do zdieľanej pamäte, procesy, ktoré majú k nej prístup môžu tieto dáta čítať. Pri súbežnom prístupe k zdieľaným dátam je potrebné zaistiť synchronizáciu prístupu. V tomto prípade zodpovednosť za komunikáciu padá na programátora, operačný systém poskytuje len prostriedky pre jej uskutočnenie.

**3. rúry**

**4. RPC, RMI**

**5. sockety**

**42. Virtuálna pamäť - stránkovanie na požiadanie**

Procesy sú umiestnené na disku. Keď chceme vykonávať proces, presunieme ho do pamäte.

Avšak miesto presunutia celého procesu do pamäte presunieme len jeho časť. Používame tzv. **„lenivý“ swapper**, ktorý nepresúva stránku do pamäte, kým nie je potrebná.

Keď sa proces má zaviesť do pamäte, stránkovač predpokladá ktoré stránky budú potrebné, kým proces nebude znova odsunutý von z pamäte. Miesto presúvania celého procesu, stránkovač presunie do pamäte len vytypované stránky. Takto sa vyhne presúvaniu stránok, ktoré nebudú potrebné a tým sa zníži čas potrebný na výmenu procesov v pamäti.

Pri použití tejto schémy je potrebná HW podpora pre rozlíšenie, ktoré stránky sú v pamäti a

ktoré nie sú. Používa sa už zmienený spôsob nastavenia bitu platná/neplatná (valid/invalid). V tomto prípade keď bit je nastavený na hodnotu platná, to znamená, že stránka je platná a je v pamäti.

Ak bit je nastavený na neplatná, to znamená, že stránka je buď neplatná (nepatrí do adresného priestoru procesu), alebo nie je v pamäti. Ak sa proces nikdy neobráti na stránku, ktorá je označená ako neplatná, potom toto označenie nemá žiadny vplyv na proces. To znamená, že ak správne odhadneme, ktoré stránky proces skutočne potrebuje a len tie presunieme do pamäte, proces prebehne tak, ako keby bol celý v pamäti. Situácia sa zmení vo chvíli, keď sa proces obráti na stránku, ktorá nie je v pamäti.

Obrátenie sa na stránku označenú ako neplatná spôsobí výpadok stránky (page fault). Stránkovací hardvér si počas prekladu adresy všimne, že bit platná/neplatná je nastavený na neplatná a spôsobí prerušenie pre neprítomnosť stránky v pamäti. Toto prerušenie má za následok presunutie požadovanej stránky do pamäte.

Obsluha prerušenia výpadku stránky prebieha v nasledujúcich krokoch:

1. Skontroluje sa vnútorná tabuľka (obyčajne uchovaná v riadiacom bloku procesu) pre

tento proces, aby sa zistilo, či odkaz na danú stránku bol platný.

2. Ak odkaz bol neplatný, proces sa ukončí. Ak odkaz bol platný, ale stránka nie je v

pamäti, zaháji sa jej presun.

3. Nájde sa voľný rámec.

4. Odštartuje sa disková operácia pre načítanie požadovanej stránky do určeného rámca.

5. Po skončení operácie načítania sa vnútorná tabuľka modifikuje, aby odzrkadľovala

prítomnosť stránky v pamäti.

6. Reštartuje sa inštrukcia, ktorá spôsobila výpadok stránky.

Proces, ktorý spôsobil výpadok stránky, sa reštartuje presne od miesta, kde prerušenie nastalo. Toto je možné, pretože stav procesu sa uchováva - registre, podmienkové kódy,

čítač inštrukcií.

Extrémny prípad: odštartovať proces, ktorý nemá v pamäti žiadnu stránku. Po pokuse načítať prvú inštrukciu programu, nastane ihneď výpadok stránky - „čisté stránkovanie na žiadosť“

**44. Synchronizácia cez SWAP**

Táto inštrukcia zaisťuje atomickú výmenu obsahu dvoch premenných a je definovaná

nasledovne:

***procedure*** *Swap (* ***va****r a,b: boolean)*

***var*** *temp: boolean*

***begin***

*temp:**a*

*a:**b*

*b:**temp* 

***end***

Ak dva procesy potrebujú zaistiť vzájomné vylúčenie v KS, môžu použiť jednu spoločnú premennú lock, nastavenú na začiatku na false a jednu lokálnu premennú key.



**45. Uvedte zakladne systemove volania pre spravu procesov a ich ulohy**

1. príkaz **exit() –** normálne ukončenie procesu

2 . príkaz **kill()** – násilné ukončenie procesu

3. príkaz **abort()** – abnormálne ukončenie procesu

4. príkaz **wait()** – čakanie na ukončenie potomka

**47. Co je zahltenie, kedy vznika a ako sa mu predchadza?**

Zahltenie je nežiaduci stav v systéme, kedy frekvencia výpadkov stránok narastie na neúnosnú mieru – procesor venuje viac času riadeniu výmen stránok, než výpočtom.

Východiskom je zníženie počtu procesov v pamäti. Aby sa zahlteniu predišlo, procesy musia mať toľko stránok, koľko potrebujú.

Princíp lokality hovorí, že procesy majú počas vykonávania niekoľko lokalít, v ktorých využívajú len istú podmnožinu svojich stránok, takže stačí natiahnuť do pamäte len tie.

**Model pracovnej sady:** pracovná sada zahŕňa posledných Δ odkazov na stránky procesu, je to aproximácia lokality. OS prideľuje bežiacim procesom rámce tak, aby v pamäti mohla byť celá pracovná sada. Ak sa to nedá zabezpečiť, pozastaví niektorý proces, čím sa uvoľní potrebný počet rámcov.

**49. Bezpečnosť OS**

Bezpečnostné hrozby:

**Utajenie:** požadujeme, aby informácie uložené v počítačovom systéme boli dostupné len oprávneným osobám. Dostupnosťou budeme rozumieť možnosť zobrazenia, tlače, resp. iného spôsobu odhalenia včítane zistenia ich existencie.

**Integrita**: znamená podmienku možnosti modifikácie informácií iba oprávnenými osobami. Modifikácia zahŕňa zápis, zmenu, zmazanie a vytvorenie.

**Dostupnosť**: požadujeme, aby počítačový systém bol dostupný oprávneným používateľom.

**Autenticita**: znamená že počítačový systém je schopný overiť identitu používateľa.

Typy bezpečnostných hrozieb:

**Odopretie služieb**: znamená útok na **dostupnosť.** Neoprávnený používateľ zničí alebo znefunkční niektorú súčasť počítačového systému. Môže byť spôsobený napr. zničením niektorej hardvérovej časti počítača, prerušením resp. zahltením komunikačnej linky, prípadne znefunkčnením súborového systému.

**Zachytenie informácie**: znamená útok na **utajenie.** Neoprávnený používateľ získa prístup k súčasti počítačového systému. Môže sa jednať o odpočúvanie komunikačnej linky, resp. kopírovanie súborov.

**Modifikácia**: znamená útok na **integritu.** Neoprávnený používateľ manipuluje so súčastami počítačového systému ku ktorým získal prístup. Môže ísť napr. o zmenu obsahu súborov, modifikáciu programov, resp. obsah správ prenášaných komunikačnou linkou.

**Falzifikácia**: znamená útok na **autenticitu.** Neoprávnený používateľ podvrhne do počítačového systému falošné súčasti – napr. nežiadúce správy do komunikačnej linky, resp. súbory do súborového systému.

Pričom nežiadúci používateľ môže byť osoba, ale aj program.

**50. Algoritmus pekára**

Názov vyplýva zo spôsobu predaja v pekárňach a mäsiarstvach (na západe), kde každý zákazník pri vstupe do obchodu dostane číslo. Pri obsluhe sa zachováva poradie príchodu zákazníkov. Tento algoritmus bol vyvinutý pre distribuované prostredie, ale tu budeme brať do úvahy aspekty týkajúce sa centralizovaného systému.   
Pri vstupe do obchodu zákazník dostane číslo. Zákazník s najmenším číslom je obslúžený ako prvý. Tento algoritmus nemôže zaistiť, že dva procesy nedostanú rovnaké čísla. Ak nastane takýto prípad, obslúži sa ako prvý proces s menším menom. To znamená, že ak Pi a Pj dostanú rovnaké čísla a i<j, potom ako prvý sa obslúži Pi . **Pretože názvy procesov sú jedinečné**, algoritmus je **deterministický** (správanie a vykonávanie algoritmu je jednoznačne určené jeho stavom).