1. Historie OS

Co to je OS? Základní části OS a v jakém modu běží?
umožňuje běh systému, aby s ním uživatel mohl manipulovat. Jádro OS (kernel) běží v kernel modu, utility (editory, shelly, kompilátory) běží v user modu.
Jaké znáte typy OS, uveďte příklady.
jednoduchý systém vypálený v ROM (bankomat); monitor (předchůdce OS, provádí např. načtení programu z děrných štítků); jednouživ. a jednoúlohové (DOS, CP/M, Win 3.11); jednouživ. a multitaskong (IBM 360); realtime (ovládá výrobní procesy v technologických zařízeních); víceuživ. a multitasking (UNIX, VMS), distribuované a síťové (Amoeba, Mach).
Charakterizujte 2. generaci OS.
1955-65; tranzistory; vědecké výpočty a komerční zpracování dat; uživatel nemá přímý přístup k počítači aby se omezily časové ztráty, batch system - programy v Job Control Language na děrných štítkách; Fortran; IBM 7090-1.
Charakterizujte 3. generaci OS.
1965-80; integrované obvody; vědecké počítače pro numerické výpočty, komerční počítače pro zpracování dat; batch system/online terminál; OS 360.
Charakterizujte 4. generaci OS.
1980-xx; LSI a VLSI integrované obvody; PC, workstations; user friendly interface (GUI - Windows, X-Window); UNIX, VMS, CP/M, DOS, Windows.
Popište monolitickou architekturu OS.
celý systém běží v kernel modu a volá se speciálními instrukcemi SVC které přepínají z/do user modu; IBM 360.
Popište architekturu vrstev OS.
systém je organizovaný v hierarchických vrstvách, vnitřní mohou být privilegovanější než vnější. Výhody: systém je navrhován systematicky od nižších vrstev k vyšším, když jsou nižší vrstvy bug-free tak se systém nemůže úplně zhroutit, potlačeno nebezpečí rekurzivního volání systémových procedur. Vrstvy systému T.H.E.: operátor - uživatelské programy - správa I/O - komunikace proces/operátor - správa paměti - procesor scheduling a multiprogramování; T.H.E., UNIX, DOS.
Popište architekturu "virtuální stroj".
virtuální stroje běží nad virtuálním monitorem, každý virtuální stroj poskytuje přesnou kopii skutečného hardwaru, na různých virtuálních strojích mohou běžet různé operační systémy.
Popište architekturu Client-Server.
klient zasílá požadavky (request) na služby, server je vykoná a vrátí výsledek (result). Server i Client běží v user modu, v kernel modu běží pouze Microkernel, který zajišťuje context switching a message passing.

2. Procesy

Vysvětlete co to je přepínání kontextu.
jeden procesor obsluhuje více procesů a každému dává k dispozici určité časové kvantum (10-100 ms), mezi je procesor zabrán jádrem a provádí se režie - uložení dat z předchozího procesu, přepnutí do dalšího (2-5 ms). Který proces má být puštěn určuje část jádra zvaná plánovač (nemusí se střídat - plánují se podle priority).
V jakém stavu se může proces nacházet a co to znamená?
proces je posloupnost akcí (dvojice stav+instrukce). Running (běžící), ready (čeká až bude naplánován - není volný procesor), blocked (proces je přerušen, např. probíhá čtení z disku), scheduled (proces je pozastaven na určitou dobu, pak se pustí), wait (proces čeká na splnění určité podmínky, např. dokončení I/O), suspended (proces byl kernelem uspán, např. málo paměti).
Vysvětlete co to je PCB a jaké informace obsahuje.
Proces Control Block je položka z Proces Table, která obsahuje všechna nezbytná data o procesu: p.name, p.identification, p.state, p.context (informace nutné pro restart spícího procesu), used memory, allocated OS resources (otevřené soubory, I/O zařízení), priorities, accounting information (doba běhu), reason(s) of blocking, PID of parent p., UID vlastníka, current directory, exit status… v UNIXu 35 položek.
Co to je časově závislá chyba. Uvedte příklad.
výsledek výpočtu závisí na konkrétním "interleaving" zúčastněných procesů. Např. dva procesy modifikují stejný soubor, zapisují do stejné proměnné zároveň.
Jak je definována kritická sekce?
část kódu, ve které se přistupuje ke sdíleným prostředkům (proměnné, segmenty paměti, soubory).
Jaký je rozdíl mezi aktivním čekáním a blokováním procesů?
Při aktivním čekání když je první proces v KS, tak druhý čeká před KS a provádí prázdné instrukce dokud první nevyjde. Tato technika plýtvá výkonem proceosru. Při blokování je proces OS přesunut do stavu blocked.
Které synchronizační metody jsou založeny na aktivním čekání?
zákaz přerušení (Disable Interrupt), zámky (Lock Variable), striktní střídání (Strict Alternation), Dekker-Petersonův alg., instrukce TSL (Test and Set Lock).
Které synchronizační metody jsou založeny na blokování procesů?
operace kernelu sleep a wakeup, semafory, čítač událostí (Event Counters), monitory, zasílání zpráv (Message Passing).
Vysvětlete jak funguje synchronizace pomocí zákazu přerušení. Její výhody a nevýhody.
(?) zabráníme přepnutí kontextu pomocí instrukce DI. Nemůže být použita v multiprocesorovém systému (přerušní jednoho uživatele blokuje i ostatní), přidělování je nespravedlivé, může dojít ke ztrátě přerušení.
Napište algoritmus pro striktní střídání 2 procesů.
Proces A: `while(1) {while (turn != 0); KS(); turn = 1; nonKS(); }`
Proces B: `while(1) {while (turn != 1); KS(); turn = 0; nonKS(); }`
Napište Dekker-Petersonův algoritmus pro synchronizaci 2 procesů.
```
#define N 2
int interested[N] = {0, 0}; int turn; /* kdo je na rade? */
void enter_region(int process) {int other; other = 1 - process;
interested[process] = 1; turn = process;
while (turn == process && interested[other]); }/* tady se provede KS() */
void leave_region(int process) {interested[process]=0; }
```
Vysvětlete, jak funguje TSL instrukce. Její výhody a nevýhody.
(?) přečte obsah paměťového slova (proměnná flag) do registru. Když je v registru 1, tak je v KS proces a čeká se, když 0, můžeme do KS vstoupit. `enter_region: mov register,#1`
```
tsl register, flag
cmp register,#0
jnz enter_region ret
leave_region: mov flag,#0
ret
```
Vysvětlete, jak fungují operace jádra sleep a wakeup. Jejich výhody a nevýhody.
sleep() zablokuje proces, který ji zavolal, wakeup() uvolní volající proces z fronty spících procesů. Když pošleme signál wakeup procesu, který nespí, tak se (oba) procesy zablokují. Řešení: zavedeme pomocný wakeup waiting bit, který udává, zda byl probuzen již probuzený proces.
Vysvětlete, jak jsou definovány semafory.
semafor je struktura, která obsahuje čítač (definuje kolik procesů může najednou manipulovat se sdíleným prostředkem) a frontu procesů (které jsou zablokovány a čekají na uvolnění). navíc jsou definovány operace: init() (nastaví čítač na 1 a vyprázdní frontu), down() (if (čítač¿0) dec(čítač) else zablokuje proces a uloží do fronty), up() (if (prázdná fronta) inc(čítač) else spustí se proces ve frontě). Operace down() a up() musí být atomické, tj. během jejich chování nesmí dojít k přerušení.
Vysvětlete, jak funguje synchronizace pomocí čítačů událostí.
čítač událostí je integerová hodnota E, nad kterou jsou definované tři základní operace: read(E) (vrátí hodnotu E), advance(E) (inkrementuje E, je atomická!), await(E,v) (if (E¡v) proces je zablokován else nic se neděje).
Vysvětlete, jak jsou definovány monitory.
monitor je objekt ve kterém jsou definována sdílená data, operace nad nimi a fronta (slouží k ukládání a vybírání běžících procesů). Nad frontou jsou definovány operace wait(q) (pozdrží volání procesu ve frontě q) a send(q) (resumes první proces, který je uložen ve frontě q).
Vysvětlete, jak funguje synchronizace pomocí zasílání zpráv.
zasílání zpráv je buď asynchronní (neblokující; proces pošle zprávu a pokračuje bez ohledu na to, zda zpráva došla) nebo synchronní (blokující; proces je zablokován (pozdržen) do té doby, než přijde potvrzení o správném přijetí zprávy). Směrování (routing) zpráv probíhá buď přímo procesu nebo nepřímo (přes pomocný objekt). U jednoprocesorových systémů je jím obvykle mailbox. Mailbox je datová struktura,

která umožňuje ukládání zpráv do a odebrání z fronty a podržení procesů, které musí čekat na nějakou zprávu, ve frontě. Operace nad mailboxem: send(process, message) (vloží do mailboxu zprávu pro proces), receive(process, message) (odstraní zprávu pro proces z mailboxu).

Napište inteligentní alg. pro řešení problému večeřících filosofů pomocí semaforů.

(řešení pomocí jednoho semaforu je nevhodné, protože by mohl jíst pouze jeden filozof)

```
#define N 5;
#define LEFT ((i-1) % N);
#define RIGHT ((i+1) % N);
enum stat(thinking, hungry, eating);
enum stat state[N]; semaphore mutex;
semaphore s[N];
void philosopher(int i) {
  while (1) { think(); take_forks(); eat(); put_forks(); }
}
void take_fork(int i) {
  down(&mutex); state[i] = hungry; test(i); up(&mutex); down(&s[i]);
}
void put_forks(int i) {
  down(&mutex); state[i] = thinking; test(LEFT); up(&mutex);
}
void test(int i) {
  if (state[i] == hungry && state[LEFT] != eating && state[RIGHT] != eating) {
    state[i] = eating; up(&s[i]);
  }
}
```

Napište alg. pro řešení problému Producent-konzument pomocí semaforů.

```
#define N 100;
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer(void) {
  itemtype item;
  while (1) {
    produce_item(&item); down(&empty); down(&mutex);
    enter_item(item); up(&mutex); up(&full);
  }
}
void consumer(void) {
  itemtype item;
  while (1) {
    down(&full); down(&mutex); remove_item(&item);
    up(&mutex); up(&empty); consume_item(item);
  }
}
```

Napište alg. pro řešení problému Producent-konzument pomocí čítačů událostí.

```
#define N 100;
typedef int semaphore;
event_counter in = 0; event_counter out = 0;
void producer(void) {
  int item, sequence = 0;
  while (1) {
    produce_item(&item); sequence = sequence + 1;
    await(out, sequence - N); enter_item(item); advance(&in);
  }
}
void consumer(void) {
  int item, sequence = 0;
  while (1) {
    sequence = sequence + 1; await(in,sequence);
    remove_item(&item); advance(&out); consume_item(item);
  }
}
```

Napište alg. pro řešení problému Producent-konzument pomocí operací sleep a wakeup.

```
#define N 100;
int count = 0; /* pocet polozek v bufferu */
void producer(void) {
  itemtype item;
  while (1) {
    produce_item(&item);
    if (count == N) sleep();
    enter_item(item); count++;
    if (count == 1) wakeup(consumer);
  }
}
void consumer(void) {
  itemtype item;
  while (1) {
    if (count == 0) sleep();
    remove_item(&item); count--;
    if (count == N - 1) wakeup(producer);
    consume_item(&item);
  }
}
```

Napište alg. pro řešení problému Producent-konzument pomocí zasílaní zpráv.

```
#define N 100;
#define MSIZE 4;
typedef int message[MSIZE];
void producer(void) {
  int item; message m;
  while (1) {
    produce_item(&item); receive(consumer,&m);
    build_message(&m,item); send(consumer,&m);
  }
}
void consumer(void) {
  int item, i; message m;
  for (i = 0; i < N; i++) send(producer,&m);
  while (1) {
    receive(producer,&m); extract_item(&m,&item);
    send(producer,&m); consume_item(item);
  }
}
```

Napište alg. pro řešení problému Čtenáři-písaři pomocí semaforů.

```
semaphore mutex = 1; semaphore db = 1; /* access to database */
int rc = 0; /* readers counter */
void reader(void) {
  while(1) {
    down(&mutex); rc++;
    if (rc==1) down(&db);
    up(&mutex); read_data_base(); /* KS() */
    down(&mutex); rc--;
    if (rc==0) up(&db);
    up(&mutex); use_data_read(); /* nonKS().*/
  }
}
void writer(void) {
  while(1) {
    think_up_data(); down(&db); write_data_base(); up(&db);
  }
}
```

Napište alg. pro řešení problému Spící holič pomocí semaforů.

```
#define CHAIRS 5;
semaphore customers = 0; semaphore barbers = 0;
semaphore mutex = 1; int waiting = 0;
void Barber(void) {
  while (1) {
    down(customers); down(mutex);
    waiting = waiting -1; up(barbers); up(mutex); cut_hair();
  }
}
void Customer(void) {
  down(mutex);
  if (waiting < CHAIRS) {
    waiting = waiting + 1; up(customes); up(mutex);
    down(barbers); get_haircut();
  } else {
    up(mutex);
  }
}
```

Vysvětlete, jak funguje alg. Round Robin a jeho nevýhody.
každý proces běží po dané časové kvantum. Procesy jsou uloženy ve frontě. Když vyprší časové kvantum, je běžící proces přesunut na konec fronty a je spuštěn první proces.
Vysvětlete, jak funguje prioritní plánování.
priorita je celé číslo z předem stanoveného rozsahu, které určuje prioritu přístupu k prostředkům (obvykle CPU). Priotita je nastavena systémem ještě než je proces spuštěn. V některých systémech (UNIX) se může priorita měnit během života procesu. Víc druhů: statická priorita (procesy se vykonávají v pořadí podle hodnoty), dynamická priorita (během života procesu se priorita mění), kombinovaná (proces má stat. i dyn. prioritu, pokud se vyskytnou dva procesy se stejnou stat. prioritou, tak se rozhodne podle dyn.), fairness dynamic priority (všechny procesy dostanou stejný čas).
Vysvětlete, jak funguje alg. Multiple Queues a proč se používal.
u operačního systému CTSS bylo přepínání procesů velmi pomalé (IBM 7094 mohl mít v paměti pouze jeden proces), proto bylo výhodnější dávat procesům větší časová kvanta, aby se snížil počet přepínání procesů. Procesy s nejvyšší prioritou p=max běžely časové kvantum q, procesy s nižší prioritou p=max-1 2q, p=max-2 4q, atd. Když proces trval déle než jemu přidělený počet kvant, byl posunut o třídu níže (tj. byl mu zdvojnásoben počet přidělených časových kvant).
Vysvětlete, jak funguje alg. Shortest Job First a kdy lze použít.
nejdříve se spouštějí procesy s nejnižším odhadem doby běhu. Tím získáme nižší průměrnou dobu běhu ((?) a kratší dobu odezvy). Vhodné pro dávkové systémy (batch systems), kde známe dobu běhu (run time) předem. Dá se použít i pro interaktivní procesy, kdy si ukládáme průměr z dosažených časů.
Vysvětlete, jak funguje alg. Guaranteed Scheduling.
každý z n nalogovaných uživatelů dostane 1/n výkonu procesoru.

3. Uváznutí (deadlock)

Vysvětlete, co to jsou Coffmanovy podmínky.
Pokud nastanou všechny podmínky najednou, dojde k deadlocku: 1. výlučný přístup (prostředky nemohou být sdílené), 2. hold and wait podmínka (proces využívající již dříve přidělené prostředky může žádat o další), 3. prostředky jsou neodnímatelné, 4. podmínka kruhového čekání.
Vyjmenujte strategie řešení uváznutí.
ignorování (Ignore), detekce a vzpamatování (Detection and Recovery), vyhýbání se deadlocku (Dynamic Avoidance), prevence (Prevention).
Vysvětlete, jak funguje řešení uváznutí pomocí "detekce a zotavení".
v okamžiku, kdy dojde k deadlocku se snažíme vrátit poslední akci. Tři metody zotavení: 1. Recovery Through Preemption (dočasně sebereme procesu prostředek a dáme ho jinému), 2. Recovery Through Rollback (ukládáme si informace o procesech; když se vyskytne deadlock, tak se vrátíme do stavu, kdy proces ještě daný prostředek neměl a daný prostředek přiřadíme jinému deadlocknutému procesu), 3. Recovery Through Killing Processes (můžeme killnout proces v cyklu (čeká na prostředek) nebo proces držící prostředek; výsledný stav může být nekonzistentní!).
Vysvětlete, jak funguje řešení uváznutí pomocí "zamezení" (deadlock avoidance).
před přidělením prostředku zvažujeme, zda nedojde k deadlocku. K tomu se používá bankéřův algoritmus (plánovací algoritmus, který se dokáže vyhnout deadlocku), potřebujeme ale vědět, kolik je procesů a kolik každý proces potřebuje prostředků (=¿ spíše teoretický alg.).
Vysvětlete, jak funguje řešení uváznutí pomocí "prevence".
snažíme se porušit aspoň jednu Coffmanovu podmínku: 1. sdílet prostředky (např. printer spooling - každý proces může zařadit soubor na tisk, ale pouze printer daemon může přistupovat k prostředku tiskárna; nelze u všech), 2. všechny procesy žádají o všechny prostředky najednou (zbytečné plýtvání prostředky), 3. odnímat prostředky (in fact impossible - třeba odebrat tiskárnu během tisku), 4. prostředky, které jsou k dispozici se očíslují a procesům jsou přidělovány pouze v určitém pořadí (tj. mohou žádat pouze o prostředky číslem větším, než již mají přiděleny).

4. Správa paměti

Vysvětlete, jak funguje přidělování paměti ve víceúlohovém OS s úseky pevné délky.
Paměť je rozdělena do n částí nestejné délky. Velikost oblastí se nemění za běhu OS. Správa paměti se děje dvěma způsoby: 1. ke každé oblasti je přiřazena fronta procesů (nevýhoda - fronta pro velkou oblast může být prázdná zatímco fronta pro malou oblast může být plná), 2. procesy jsou v jedné frontě a do oblastí jsou rozdělovány pomocí strategie best fit (nejlépe vyhovující oblast - diskriminuje malé procesy) nebo first fit (první vhodná oblast - plýtvá velkými oblastmi na malé procesy).

Vysvětlete, jak funguje přidělování paměti ve víceúlohovém OS s úseky proměnné délky.
Počet, umístění a velikost oblastí se mění dynamicky v závislosti na potřebách procesů.
Popište řešení problému relokace a ochrany úseků paměti.
Problém posunutí (relokace) - programy neví, na jaké fyzické adrese budou umístěny; řešením je linkování adres proměnných, skoků atp. Problém ochrany - programy nemohou využívat jiné oblasti RAM; řešení: paměť je rozdělena do bloků a ke každému bloku je připojen klíč, který specifikuje úlohy, které mohou tento blok využívat.
Co to je interní a externí fragmentace paměti? Kdy se vyskytuje?
Vyskytují se při přidělování paměti ve víceúlohovém OS. Interní fragmentace - v oblastech zůstávají nevyužité části paměti; řešení: proměnná velikost oblastí. Externí fragmentace - vytvoří se díry volné paměti mezi oblastmi které jsou příliš malé na využití; řešením je sesypání, to je ale velice pomalé (resp. náročné na výkon CPU).
Vysvětlete metody přidělování paměti: first fit, next fit, best fit a worst fit.
First fit hledá první využitelnou díru (nejlepší), next fit - hledání díry neprobíhá od začátku, ale od místa, kde se v předchozím kroku skončilo, best fit - hledá se nejmenší díra, kde se dá proces nahrát a spustit, worst fit - hledá se co největší díra (aby byl zbytek ještě využitelný). Všecny algoritmy se dají urychlit vytvořením oddělených seznamů (linked list) pro díry a procesy.
Vysvětlete metodu přidělování paměti: quick fit.
Používáme seznamy volných děr rozdělených podle velikostí (např. díry 1-5 KB, 5-10 KB, ...). Nevýhodou je složité hledání sousedních děr ke spojení.
Vysvětlete, jak funguje "Buddy system".
Jedná se o algoritmus quick fit s dírami velikosti 2n bytu.
Popište princip "odkládání procesů" (swapping).
Jedná se o přesun celých procesů mezi fyzickou pamětí a diskem. Ve většině OS se swapovací prostor používá pouze k ukládání dat běžících procesů. Kód programu bývá read-only a není jej tedy třeba ukládat na disk.
Popište jak funguje virtuální paměť.
Pokud je program větší než fyzická RAM, tak programátor rozdělí program na části (overlays) a do paměti se vždy zavedou jen ty nezbytně nutné (root overlay je v paměti stále, transient overlays se zavádí z disku jen v případě potřeby). Virtuální paměť je technika, který vytváří iluzi, že máme víc fyzické paměti než tomu je ve skutečnosti. Virtuální paměť je tvořena zčásti pamětí RAM a zčásti pamětí na disku (swap space, page space), která je namapována do paměti RAM. Procesy jsou automaticky (tj. OS) rozděleny na části (mohou být stejné - stránkování, nebo různé - segmentace) které se ukládají do paměti nebo na disk.
Vysvětlete, co to je strankování paměti.
Systém, kdy je virtální paměť rozdělena na stránky o stejné velikosti (typicky 4 KB).
Vysvětlete, co to je segmentace paměti.
Systém, kdy je virtální paměť rozdělena na stránky o různé velikosti.
Jaký je rozdíl mezi stránkou a rámcem?
Fyzická paměť (RAM) je rozdělena na rámce, virtuální je rozdělena na stránky.
Jaký je rozdíl mezi stránkou a segmentem.
stránka má vždy stejnou velikost (typicky 4 KB), segment ne.
Jak funguje tabulka stranek a jak ji lze realizovat?
Tabulka stránek slouží k převodu virtuální adresy na fyzickou (číslo rámce = f(číslo virtuální stránky), f je implementována tabulkou stránek). Dá se realizovat hardwarově (+rychlý převod; - drahé, dá se použít pouze u malých tabulek stránek, přepínání kontextu je pomalé) nebo v paměti (+ levnější, rychlé přepínání kontextu; - pomalý převod adresy). Tabulka stránek může být hodně velká, mapování musí být velmi rychlé (provádí se při každém odkazu do paměti).
Jak funguje více úrovňová tabulka stánek, výhody a nevýhody?
skládá se z tabulek 1. až n-té úrovně. (?) Vyhneme se udržování všech tabulek stránek v paměti. Tam uchováváme pouze adresy použitých stránek.
Popište princip TLB (Translation Lookaside Buffer).
Translation Lookaside Buffer je organizován jako asociativní paměť. Obsahuje nejčastěji používané stránky paměti (obvykle desítky). Než začne MMU překládat adresu, ověří, zda se hledané číslo virtuální stránky nevyskytuje v TLB. Pokud ano, tak se veme přímo z TLB a už se nemusí jít do tabulky stránek.
Jak funguje invertovaná tabulka stánek, výhody a nevýhody?
(?) invertovaná tabulka stránek je menší než klasická tabulka, používá hashovací funkci. I-tý zápis obsahuje informaci o virtuální stránce, která právě zabírá rámec i.
Vysvětlete jak funguje optimální alg. pro náhradu stránek.
Odstraní se stránka, která bude použitá jako poslední. Pokud bychom každou stránku očíslovali počtem instrukcí, které musí proběhnout, než bude stránka poprvé zavolána (referencována), tak optimální algoritmus odstraní stránku s nejvyšším číslem.
Vysvětlete jak funguje NRU alg. pro náhradu stránek.
Ke každé stránce přiřadíme dva bity: R - nastaví se, když je stránka volána (ať už read nebo written), M - nastaví se, když je do stránky zapisováno (je modifikována). Tyto bity má každá stránka a jsou nastavovány hardwarem. Při startu procesu jsou všechny bity nastaveny na 0. Periodicky (v každém cyklu hodin; např. po 10 ms) jsou R bity vynulovány, aby se odlišily stránky, které byly "čerstvě" referencovány. Pokud není volná stránka (page fault), tak systém rozdělí všechny stránky do čtyř kategorií a odstraní náhodnou stránku z co nejnižší (neprázdné) třídy.
Vysvětlete jak funguje FIFO alg. pro náhradu stránek.
OS udržuje seznam všech stránek; na začátku (head) nejstarší a na konci (tail) nejpozději načtená. Při page fault se odebere první stránka a nová se přidá na konec. Moc se nepoužívá.
Vysvětlete jak funguje Clock alg. pro náhradu stránek.
Každá stránka má bit R, který určuje, zda se ke stránce přistupovalo. Stránky jsou uloženy v kruhu a ukazatel ukazuje na začátek. Při page fault se prozkoumá stránka, na kterou ukazuje ukazatel. Jestliže má bit R=0, je odstraněna, na její místo se dá nová stránka a ukazatel se posune na další. Pokud je bit R=1, je změněn na 0 a ukazatel se posune na další. To se opakuje tak dlouho, dokud se nenalezne ztránka s bitem R=0. Dobrá efektivita - blíží se ideálnímu, používá se.
Vysvětlete jak funguje LRU alg. pro náhradu stránek.
Vyhodí se stránka, ke které se přistupovalo před nejdelší dobou (tj. nejdéle nepoužitá). Je efektivnější než NRU a Clock, ale má náročnou implementaci.
Popište princip HW implementace LRU algoritmu.
1. LRU implementovaný speciálním HW - každá stránka má čítač, který se inkrementuje po každém volání stránky. Stránka s nejnižším číslem je při page fault vyhozena. 2. LRU používající matici (matrix) - pro n stránek je hardwareově implementována matice n x n s počátečními hodnotami 0. Pokud je volána stránka k, nastaví HW všechny bity řádku k na 1 a všechny bity sloupce k na 0. Odebrána je stránka, jejíž řádek má nejnižší hodnotu (tj. bylo k ní přistupováno před nejdelší dobou).
Popište princip SW implementace LRU algoritmu.
Předchozí algoritmy byly závislé na speciálním hardwaru. Softwareové řešení NFU spočívá v tom, že každá stránka má svůj vlastní čítač (iniciovaný 0). V každém cyklu hodin se přidá do čítače hodnota registru R. Vyhazuje se stránka s nejnižším čítačem. Problém: stárnutí - program nezpomíná, je třeba nějak zvýhodnit stránky, ke kterým se přistupovalo nedávno. Řešení: před připočtením bitu R provede SHR čítačů a R se připočte do nejlevějšího bitu.

5. Souborové systémy

Vysvětlete princip různých přístupů k souboru.
1. Náhodný přístup (Random Access) - záznamy mohou být čteny nebo zapisovány v libovolném pořadí (diskové soubory). 2. Sekvenční přístup (Sequential Access) - OS může číst a zapsovat záznamy pouze v sekvenčním pořadí (magnetická páska, tiskárna). 3. Virtuální sekvenční přístup - OS simuluje sekvenční přístup na náhodně přístupných souborech.
Vysvětlete princip "memory mapped files".
Jedná se o mapování souborů do adresního prostoru ve virtuální paměti. Při mapování souboru do virtuální paměti přiřadíme k začátku souboru virtální adresu. Při čtení nebo zápisu na onu virtuální adresu dojde k page fault, načte se první stránka (4 KB) dat do RAM atd. Soubory mapované v paměti jsou updatovány do originálních souborů, ne do swap space.
Jaké znáte způsoby alokace dat. bloku souborů? Vysvětlete jejich princip.
1. Souvislá alokace (Contiguous Allocation) - soubor je mapován do souvislého místa na disku; + jednoduchá implementace, stačí jedna adresa (prvního datového bloku), výborný výkon (celý soubor přečteme jednou operací); - složitá alokace místa (neznáme dopředu velikost vytvářeného souboru), fragmentace disku. 2. Alokace spojovým seznamem (Linked List Allocation) - každý blok souboru obsahuje ukazatel na další blok; zastaralé, nepoužívá se; + stačí jedna adresa (prvního datového bloku), není fragmentace; - pomalý náhodný přístup, nemůžeme využít na uložení dat celý blok. 3. Alokace spojovým seznamem

s použitím tabulky (Linked List Allocation Using Table) - v RAM je tabulka s pointery na každý blok; + snažší náhodný přístup, k uložení dat můžeme použít celý blok; - celá tabulka musí být v paměti dokud soubor používáme. 4. I-nodes - i-node je datová struktura která obsahuje 12 přímých adres a tři nepřímé adresy (prvního, druhého a třetího řádu); + cachujeme pouze i-nody v RAM; - přístup k velkým souborům je pomalejší než k malým, průměrně 10% datových bloků je použito na nepřímé pointery.

Jaké znáte způsoby implementace adresařů? Vysvětlete jejich princip.

CP/M, MS-DOS, UNIX.

Jakým způsobem si systém souborů udržuje informaci o volných dat. blocích na disku?

1. Zřetězený seznam (Linked List) - seznam obsahuje čísla volných bloků, doplňuje se (po smazání souboru) od začátku. 2. Bitová mapa - potřebuje méně místa, seznam je výhodnější pouze když je disk téměř plný.

Vysvětlete, jak se liší jednotlivé typy RAID.

Redundant Array of Inexpensive Disks. RAID 0 - Disk Stripping (celý diskový prostor je použit pro uložení dat bez redundance), RAID 1 - Disk Mirroring (zrcadlení, každý disk má identického dvojníka), RAID 3 - Byte Stripping (zabezpečení bytů pomocí parity; - zpoždění při zápisu), RAID 5 - Block Stripping with Parity (zabezpečení bloků pomocí parity, ale parita je rozprostřená na víc disků). Nepoužívá se: RAID 2 (zabezpečení pomocí kódu), RAID 4 (zabezpečení bloků pomocí parity), RAID 6 (zabezpečení pomocí dvojí parity).

Vysvětlete jak fungují virtuální disky.

Virtuálním diskem je buď část (partition) fyzického disku. Pak je ale omezená velikost adresáře (např. /var, /home, /tmp). Virtuálním diskem je ale i logický disk. Fyzické disky se rozdělí do částí (extents) o pevné velikosti 4 MB. Tyto části se mohou sloučit do logických disků (logical volume). Logické disky se mohou skládat z částí více fyzických disků a jejich velikost může být změněna bez formátování.

Popište systém souborů ufs.

Inode 128 B, data blocks 8 KB default size (other 4, 16, 32, 64 KB), reálná velikost souboru max. 2 GB, jméno souboru 255 B, nelze defragmentovat.

Popište systém souborů vxfs.

(?) Systém ufs se žurnálováním. Pokud dojde k chybě, projde systém intent log a pak provede potřebné operace rollback nebo rollforward. Zápis všech metadat (struktura dat ve FS, všechny ionformace ve FS kromě jmen a obsahu souborů) je atomickou transakcí. Logují se pouze celé (dokončené) transakce.