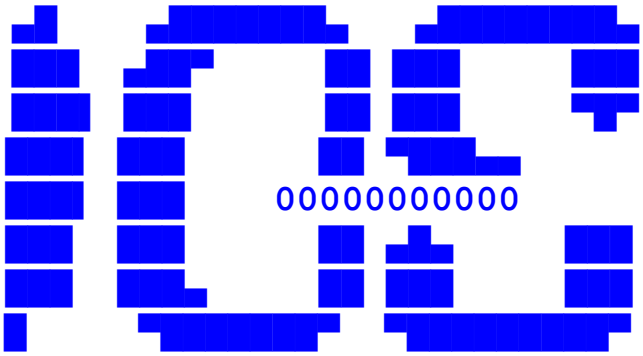


o

označit text, pravé tlačítko myši > Komentář

(Podakujte ročníku 1BIT 2013/2017)

Ked viete ešte o niečom kľudne to doplňte :)

Kto by chcel ešte nejaké materiály tak tu je môj archív k IOS: https://mega.co.nz/#F!o8RUFABR!icVROTdChf5arVvhvicp3A

Příkládám spojené všechny slidy z přednášek bez bashe...

https://dl.dropboxusercontent.com/.../ios_slides_nobash.pdf

Pridané nové zadania z fitušky na konci dokumentu (od stranky 15), ak sa niečo opakuje tak sorry but “no hate”, ved opakovanie matka mudrosti. :)

Pokud nehodláte nic editovat, tak si prosím nastavte místo Editing jen View.

(Dobrý postreh, vďaka M.N., !inak vznikajú zbytočné prepisy a je to zasekané!)

```
      _,      _   _      ,_
    o888P     Y8o8Y     Y888o.
   d88888      88888      88888b
 ,8888888b_  _d88888b_  _d8888888,
 888888888888888888888888888888888 spiderman?
 888888888888888888888888888888888
  Y8888P"Y888P"Y888P-Y888P"Y88888'
    Y888   '8'   Y8P   '8'   888Y
     '8o          V          o8'
```

why u do dis spoder

---

1. Problematika přepisu dat u SSD [pr 4; sl 8]

- Nutnost načíst blok do vyrovnávací paměti, změnit, na disku vymazat, zapsat zpět a to celý, byť se mění jen 1 stránka
- Řešení (lépe řečeno minimalizace problému):
    - Použití více stránek, než je oficiální kapacita
    - Příkaz TRIM - umožňuje souborovému systému sdělit SSD, které stránky nejsou používány a lze je opět považovat za prázdné
    - Samostatné uvolňování stránek

2. Maximální počet výpadků v N-úr.t.str [pr 7; sl 26]

- k výpadkům může dojít u dat i u instrukce
- Instrukce může zasahovat do dvou různých stránek *(záleží na zadání. V tomto zadání se data nachází jen na jedné stránce)*
- Každá z uvedených stránek může být překládána přes různé dílčí tab. str. (s výjimkou dílčí tab. str. nejvyšší úrovně), protože popis stránek, do nichž zasahuje instrukce, může být na rozmezí tab. str.
- Každá dílčí tab. str. (kromě nejvyšší úrovně) může vypadnout, vypadnout jich tedy může N-1 pro každou zpřístupněnou stránku.(1 + (N-1)) + 2 (1 + (N-1)) = 3N

  //dodatek -> ta 1 na začátku závorky je způsobena tím, že se první přeloží stránky (a až potom se překládá první,druhá.. a další úrovně, a jelikož je první chráněna proti výpadku je to (N-1).

3. Možný počet úrovní hierarch. tab. str. [pr 7; sl 16,17]

- pro posuv ve stránce je třeba 32b
- 128 - 32 = 96b zbývá pro stránkování
- 128b = 2^4B a tedy dílčí tab. str mají 2^{32} / 2^4 = 2^{28} řádků
- pro plné využití dílčích tab. str. je třeba 28b adresy
- 96 / 28 = 3.42 → 3 odkazy
- do 96b se tedy vejdou, při dané toleranci nepoužitých bitů adresy, 3 odkazy do tab.str a dostáváme tedy 3-úr. tab. str.

4. Plánování přístupu na disk [pr 4; sl 14]

- Příchozí I/O požadavky jsou ukládány do vyrovnávací paměti a jejich pořadí je měněno s cílem minimalizovat režie diskových operací
- Výtahový algoritmus: pohybuje hlavičkami od středu k okraji ploten a zpět, požadavky na I/O vyřizuje v pořadí odpovídajícím pohybu hlaviček
- Heuristiky:
    - Circular SCAN
    - LOOK, C-LOOK(NOOP,...)
    - sdružování operací
    - vyvažování operací různých procesů
    - priority operací
    - odkládání operací
    - časové limity operací+\

5. Max. počet přístupů do RAM u N-úr tab. str. [pr 7; sl 13, 16, 17]

- 4N + 4 = 4 (N+1)
- přístup pro data i kód instrukce
- obojí případně nezarovnáno - 4 stránky (pokud budou stránky zarovnány mohou být jen 2)
- pro každou stránku překlad přes N-úrovní

6. Počet bloků souboru o velikosti 360,5 1 kB, odkaz 10B [pr 4,7; sl 22]60,5 kB v UFS, al. blok

- 366 alokovaných bloků
  - 361 datových
  - 1 s přímými odkazy dostupnými z i-uzlu přes nepřímý odkaz 1. úrovně
  - 1 s nepřímými odkazy 1. úrovně
  - 3 s přímými odkazy dostupnými z výše uvedeného bloku

7. CFS (Completely Fair Scheduler) [pr 5; slajd 24]

- explicitně se snaží poskytnout procesům odpovídající % strojového času
- u procesů sleduje spotřebovaný čas (jejich strávený procesorový čas)
- vede si minimální spotřebovaný čas, pro nově připravené procesy
- procesy udržuje ve vyhledávací stromové struktuře, podle jejich spotřebovaného procesorového času
- vybírá proces s nejmenším spotřebovaným časem
- délka přidělovaného kvanta je ovlivněna prioritou
- podporuje skupinové plánování - rozděluje čas spravedlivě pro procesy spuštěné z jiných terminálů

8. Monitory pomocí semaforů [pr 6; sl 12,13,16]

Wait - Notify

```
struct sMonitor tMonitor;
void enter (tMonitor *m) {
   lock(m->mutex);
}
void wait (tMonitor *m) {
   m->condCount++;
   unlock(m->mutex);
   lock(m->condition);
   lock(m->mutex);
}
void notify (tMonitor *m) {
   if (m->condCount > 0) {
      unlock(m->condition);
      m->condCount--;
   }
}
```

```
void leave (tMonitor *m) {
   unlock(m->mutex);
}
```

Wait - Signal

```
struct sMonitor tMonitor;
void enter (tMonitor *m) {
    lock(m->mutex);
}
void wait (tMonitor *m) {
   m->condCount++;
   if(m->prioCount > 0)
       unlock(m->priority);
   else
       unlock(m->mutex);
   lock(m->condition);
   m->condCount--;
}
void signal (tMonitor *m) {
   if (m->condCount>0) {
      m->prioCount++;
      unlock(m->condition);
      lock(m->priority);
      m->prioCount--;
   }
}
void leave (tMonitor *m) {
   if(m->prioCount > 0)
      unlock(m->priority);
   else
      unlock(m->mutex);
}
```

9. Akce OS při operaci close() [pr 4; sl 59]

- kontrola platnosti fd
- uvolní položku v tabulce desktriptorů, sníží počítadlo odkazů v tabulce otevřených souborů
- pokud je počitadlo odkazů nulové, uvolní položku, sníží počet odkazů ve v-uzlu
- pokud je počitadlo odkazů nulové, i-uzel se z v-uzlu okopíruje do VP (virtuální paměť) a uvolní
- vrací 0, nebo při chybě -1

10. Účel a princip dvou-úrovňové obsluhy přerušení [pr 2; sl 13]

- minimalizace délky zákazu přerušení
- 1. úroveň:
    - zajišťuje minimální obsluhu HW, plánování běhu 2 úrovně
    - může být vyloučeno/zamaskováno přerušení
- 2. úroveň:
    - postupně dokončuje obsluhu zaznamenaných přerušení
    - nemusí zakazovat přerušení - vzájemné vyloučení se dá řešit běžnými synchronizačními prostředky

11. Počet čtení bloku u open (“/dir/symlink”) ... [pr 4; sl 22]

- blok s i-uzlem “/”, obsah ”/” //2
- blok s i-uzlem “/dir”, jeho obsah //2
- i-uzel symlink //1
- překlad “/dir” na i-uzel je ve vyrovnávací paměti, stejně tak jeho obsah
- blok s i-uzlem “/dir/file” a jeho obsah //2
- další čtení jde z vyrovnávací paměti
- celkem 7 čtení! //2 + 2 + 1 +2

12. Implementace lock s využitím spinlocku (Swap) [pr 6; sl 9,12,13]

```
struct sSemaphore
{
   int value;
   bool locked;
   queue *q;};

   void init(struct sSemaphore *s, unsigned value){
       s->value = value;
       s->locked = false;
   }

   void lock(struct sSemaphore *s)
   {
   bool key = true;
   while(key)
   {
      swap(&(s->locked), &key);
      }

      s->value--;

      if (s->value < 0)
      {
      process_t *C=get(ready_q);
      append(s->q,C);
      s->locked = false;

      switch();
   }
   else
      s->locked = false;
}
```

13. Implementace lock včetně spinlocku (Test&Set) [pr 6; sl 13]

```
struct sSemaphore{
   int value;
   queue *q;
```

```
   bool locked;
};

void init(struct sSemaphore *s, unsigned value){
    s->value = value;
    s->locked = false;
}

void lock(struct sSemaphore *s) {
   while(Test&Set(s->locked));
   s->value--;
   if (s->value < 0) {
      process_t *C=get(ready_q);
      append(s->q,C);
      s->locked = false;
      switch();
   } else
      s->locked = false;
}
```

Semafor, funkce unlock s využitím spinlock [př 6, sl 14]

```
   Test&Set
   struct sSemaphore {
       int value;
       queue *q;
       bool locked;
   };

   void unlock(sSemaphore *s) {
       while(Test&Set(s->locked));
       s->value++;
       if (s->value <= 0) {
           queue *p = get(s->q);
           append(ready_queue, p);
           s->locked = false;
       }
       else
           s->locked = false;
   }


   Swap
   struct sSemaphore {
       int value;
       queue *q;
       bool locked = false;
   };

   void unlock(sSemaphore *s) {
       bool key = true;
       while(key)
           swap(&(s->locked), &key);
```

```
        s->value++;
      if (s->value <= 0) {
          queue *p = get(s->q);
          append(ready_queue, p);
          s->locked = false;
      }
      else
          s->locked = false;
  }
```

14. Vyhledávání v tabulce stránek [pr 7; sl 18,19]

Invertovaná tabulka

```
struct inv_tab_item {
   pid_t pid;
   page_num_t pg;
};
page_num_t page_to_frame(...) {
   for (page_num_t i = 0; i < max_frame_num; i++) {//page_num_t i = 0; i<...
      if (inv_tab[i].pid == pid && inv_tab[i].pg == pg) {
         return i;
      }
   }
   return PAGE_FAULT;
}
```

Invertovaná tabulka s hashováním

```
struct inv_tb_item {
   pid_t pid;
   page_num_t pg;
   page_num_t next;   // Tohle místo struct
};

page_num_t page_to_frame(...) {
   page_num_t fr = hash(pg, pid);
   while (inv_tb[fr].pid != pid || inv_tb[fr].pg != pg){
      if (inv_tb[fr].next == NEGATIVE)
         return NEGATIVE;
      else
         fr = inv_tb[fr].next;
      }
   return fr;
}
```

Hashovací tabulka

```
struct hash_tab_item {
      pid_t pid;         // PID iba pri zdieľanej hash. tabuľke
      page_num_t pg;
      page_num_t fr;
      struct hash_tab_item* next;
};


page_num_t page_to_frame(struct hash_tab_item* hash_tab, pid_t pid, page_num_t
page_num)
{
      page_num_t index = hash(pid,page_num);
      struct hast_tab_item* itr = &hash_tab[index];
      while (itr) {
            if (itr->pid == pid && itr->pg == page_num)
                  return itr->fr;
            itr = itr->next;
      }
      return NEGATIVE;
}
```

Hashovací tabulka (regiony)

```
   struct hash_tab_item {
      page_num_t pg;
      reg_t reg;
      page_num_t fr;
      struct hash_tab_item* next;
   };
   page_num_t page_to_frame(struct hash_tab_item* hash_tab, pid_t pid, reg_t
      reg, page_num_t page_num)
   {
         reg_t glob_reg = loc_to_global_reg(pid, reg); // z lokalneho regionu
      na globalny
         page_num_t index = hashPg(pg, glob_reg);
         struct hash_tab_item* itr = hash_tab[index];
         while (itr)        {
                //if (/*itr->reg == reg && */itr->pg == page_num)

                if (itr->pg == page_num)
                        return itr->fr;
                itr = itr->next;
         }
         return PAGE_FAULT;
   }
```

Deklaracia hashPg

```
int hashPg(page_num_t pg, reg_t global_reg) { … }
```

15. Akce při výpadku str. [pr 7; sl 24-27]

- kontrola, zda se proces neodkazuje mimo přidělený adresový prostor
- alokace rámce: použijeme volný rámec, pokud nějaký je
    - pokud není vybereme vhodnou stránku s přiděleným rámcem (victim page)
    - tu odložíme na swap (je-li to nutné: příznak modifikace v tabulce stránek)
    - použijeme uvolněný rámec
- inicializace stránky po alokaci je závislá na předchozím stavu stránky

  ➔ první odkaz na stránku:
    - kód a inicializovaná data: načteme z programu
    - vše ostatní: vynulujeme (nelze ponechat původní obsah z bezpečnostních důvodů)

  ➔ stránka byla v minulosti uvolněna z FAP
    - kód a konstantní data znovu načteme z programu
    - ostatní: pokud byla stránka modifikována je ve swapu a musí se načíst do FAP
- úprava tab. str. (namapování zpřístupňované stránky na přidělený rámec)
- restart instrukce (proces je ve stavu připravený)

LRU, Least Recently Used [pr 7; sl 29-32]

- odkládá nejdéle nepoužitou stránku
- velmi dobrá aproximace *hypotetického ideálního algoritmu* (znal by budoucnost a dle budoucích požadavků by se rozhodoval, co odložit)
- problematická implementace -> vyžaduje výraznou HW podporu (označování stránek časovým razítkem posledního přístupu či udržování zásobníku stránek)
- používají se aproximace LRU
- Aproximace LRU pomocí **omezené historie**
    - při každém přístupu HW nastaví referenční bit stránky
    - jádro si vede omezenou historii tohoto bitu pro stránky
    - periodicky posouvá obsah doprava -> na pozici úplně vlevo (na začátek) uloží aktuální hodnotu ref. bitu a vynuluje ho
    - oběť (victim page) je vybrána jako stránka s nejnižší číselnou hodnotou historie

- Aproximace LRU algoritmem **druhé šance**
    - stránky jsou v kruhovém seznamu
    - postupujeme a nulujeme ref. bit
    - odstraníme první stránku, která již nulový referenční bit má (*pro lepší přehled obrázek (pr 7; sl 32))*
    - často používaný algoritmus (clock algorithm)

- výhody LRU: často používaný, velmi dobrá aproximace
- nevýhody LRU: problematická implementace (vyžaduje výraznou HW podporu)

FIFO, First In, First Out [pr 7; sl 28]

- Jednoduchá implementace
- Může odstranit “starou”, ale stále často používanou stránku

- Trpí tzv. Beladyho anomálií
- Odstraňuje stránku, která byla zavedena do paměti před nejdelší dobou a dosud nebyla odstraněna

16. Plánování [pr 5; sl 17, 19, 21, 23]

- *preemptivní:* ke změně běžícího procesu může dojít na základě přerušení (typicky od časovače, ale může se jednat i o jiné přerušení)
- *nepreemptivní:* ke změně běžícího procesu může dojít pouze tehdy, pokud to běžící proces umožní, předáním řízení jádru (I/O operace, nebo se sám vzdá procesoru - volání yield)
- **FCFS** - nepreempt. FIFO. Při vzniku procesu, jeho uvolnění z čekání, nebo vzdá-li se proces procesoru, je tento proces zařazen na konec fronty. Procesor se přiděluje prvnímu procesu ve frontě.
- **Round-robin** - preempt. FIFO. Obdoba FCFS, navíc má ale každý proces přiděleno časové kvantum, po jehož vypršení je mu odebrán procesor a proces je zařazen na konec fronty připravených procesů.
- **Víceúrovňové plánování**: procesy jsou rozděleny do skupin (typicky dle priority), v každé skupině může být jiný plánovací algoritmus (FCFS..), další plánovací algoritmus se používá pro určení skupiny, ze které se vybere proces, který má běžet (ale typicky na základě priority), může hrozit hladovění.
- **Víceúrovňové se zpětnou vazbou**: skupiny procesů rozdělené dle priorit, nově vzniklý proces je ve frontě s nejvyšší prioritou, postupně klesá, nakonec je plánován Round-Robin v nejnižší úrovni.
    - *modifikace*: proces zařazen do fronty na základě statické priority, dynamická priorita se zvyšuje, pokud hodně čeká na I/O, nebo naopak snižuje, pokud spotřebovává hodně času CPU. Cíl: rychlá reakce interaktivních procesů.

17. Kroky systému při volání open() [pr 4; sl 53]

- vyhodnocení cesty
- vytvoření v-uzlu
- položka v tab. otevření souborů
    - režim
    - pozice
    - čítač odkazů
- deskriptor
- návrat indexu

18. Deadlock [pr 6; sl 25-30]

Kostra:

- *definice*
- *Coffmanovy podmínky*
- *prevence*
- *vyhýbání*
- *zotavení se: detekce cyklu a restart / ukončení procesů po odebrání zdrojů*

- Uváznutím při přístupu ke zdrojům s omezeným přístupem označujeme situaci, kdy každý proces z určité množiny procesů je pozastaven a čeká na uvolnění zdroje s omezeným přístupem, vlastněného nějakým procesem z dané množiny, který jediný může tento zdroj uvolnit.

- Obecná definice uváznutí: je situace, kdy každý proces z určité množiny procesů je pozastaven a

čeká na událost, která by mohla nastat, pouze pokud by byl uvolněn nějaký proces z dané množiny

- **Coffmanovy podmínky:**
  - Vzájemné vyloučení při používání prostředků
  - Vlastnictví alespoň jednoho zdroje, pozastavení a čekání na další
  - Prostředky vrací proces, který je vlastní, a to po dokončení jejich využití
  - Cyklická závislost na sebe čekajících procesů

- **Řešení:**
  - Prevence uváznutí
  - Vyhýbání se uváznutí
  - Detekce a zotavení
  - (ověření, že uváznutí nemůže nastat)

- **Prevence Uváznutí:**
  (Zrušíme platnost některé z podmínek nutných pro uváznutí)

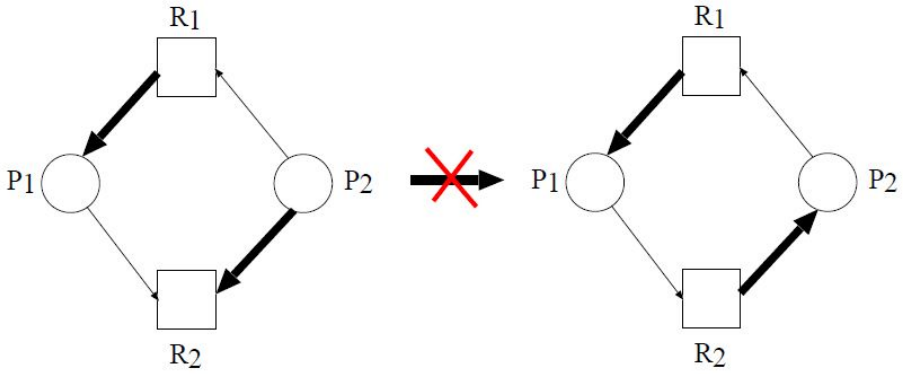
  - Nepoužívat sdílené prostředky nebo užívat sdílené prostředky, které umožňují (skutečně současný) sdílený přístup a u kterých není tedy nutné vzájemné vyloučení procesů.
  - Proces může žádat o prostředky pouze, pokud žádné nevlastní
  - Pokud proces žádá o prostředky, které nemůže momentálně získat, je pozastaven, všechny prostředky jsou mu odebrány a proces je zrušen, nebo se čeká, až mu mohou být všechny potřebné prostředky přiděleny
  - Prostředky jsou číslovány a je možné je získávat pouze od nejnižších čísel k vyšším, nebo jiným způsobem vylučujícím cyklickou závislost na sebe čekající procesů

- **Vyhýbání se uváznutí:**
  - Procesy předem deklarují určité informace o způsobu, jakým budou využívat zdroje. V nejjednodušším případě se jedná o maximální počet současně požadovaných zdrojů jednotlivých typů.

  - Předem známé informace o možných požadavcích jednotlivých procesů a o aktuálním stavu přidělování se využijí k rozhodování o tom, které požadavky mohou být uspokojeny (a které musí počkat) tak, aby nemohla vzniknout cyklická závislost na sebe čekajících procesů i v nejhorší možné situaci, která by mohla v budoucnu vzniknout při deklarovaném chování procesu.

  - Existuje řada algoritmů pro vyhýbání se uváznutí, např. algoritmus založený na grafu alokace zdrojů pro systémy s jednou instancí každého zdroje. Vytváří se graf vztahu se 2 typy uzlu (P-procesy, R-zdroje) a 3mi typy hran (R=>P) - zdroj je vlastněn procesem, (P=>R) - proces žádá o zdroj, (P->R) proces může požádat o zdroj.

  - Zdroj je přidělen pouze tehdy, pokud nehrozí vznik cyklické závislosti čekajících procesů, což by se projevilo cyklem v grafu při záměně hrany žádosti za hranu vlastnictví.



Zobecněním je tzv. Bankéřův algoritmus

- **Detekce uváznutí a zotavení:**
  - Uváznutí může vzniknout, periodicky se přitom detekuje, zda k tomu nedošlo a pokud ano, provede se zotavení
  - **Detekce uváznutí:** graf vlastnictví zdrojů a čekání na zdroje procesy (podobný jako graf alokace zdrojů, ale bez hran vyjadřujících možnost žádat o zdroj); cyklus indikuje uváznutí
  - **Zotavení z uváznutí:**
    - odebrání zdrojů alespoň některým pozastaveným procesům, jejich přidělení ostatním a později umožnění získat všechny potřebné zdroje a pokračovat (případně jejich restart či ukončení)
    - v každém případě anulace nedokončených operací (rollback), nebo nutnost akceptace možných nekonzistencí

19. OS, Jádro, mikrojádro [pr 1; sl 9-16]

- OS
  - program (kolekce programů) vytvářející spojující vrstvu mezi (příp. virtualizovaným) HW a uživateli a jejich aplikacemi
  - cíle
    - max využití zdrojů (dříve, drahé počítače x levná pracovní síla)
    - snadnost použití (dnes, levné počítače x drahá pracovní síla)
  - role
    - tvůrce virtuálního počítače - abstrakce, standardní rozhraní
    - správce zdrojů - bezpečnost a efektivita sdílení,,
- jádro
  - Jádro OS je nejnižší a nejzákladnější část OS
  - nejnižší správa prostředků a tvorba prostředí pro zbytek OS a uživ. app.
  - reaktivní program, zavádí se jako první při zapnutí a běží po celou dobu
  - Navazuje přímo na HW, typicky jej pro uživatele zapouzdřuje
  - Obvykle běží v privilegovaném režimu
- mikrojádro
  - jednoduché rozhraní, malý počet služeb, jednoduché abstrakce
  - část služeb monolitického jádra je implementována jako servery mimo privilegovaný režim
  - bezpečnost - útok na server neovládne celé jádro a OS

    - flexibilita - umožňuje zastavovat nebo spouštět služby/servery
    - nižší efektivita - vyšší režie
    - příklady: L4, Mach, QNX,...

20. Žurnálování [pr 4; sl 17-20]

- Žurnál slouží pro záznam modifikovaných metadat (dat) před jejich zápisem na disk. Implementován jako cyklicky přepisovaný buffer ve speciální oblasti disku. Operace pokryté žurnálováním jsou atomické: vytváří transakce – buď uspějí všechny dílčí kroky nebo žádný.
- systémy se žurnálem: Ext3, ext4, NTFS.
- Umožňuje spolehlivější a rychlejší návrat do konzistentního stavu po chybách.
- Data obvykle žurnálovaná nejsou – velká režie.
    - Kompromis: předřazení zápisu dat na disk před zápisem metadat do žurnálu.
- Implementace
    - Implementace na základě dokončení transakcí – *REDO* (ext3,4):
        - Sekvence dílčích operací se uloží do žurnálu mezi značky značící start a konec transakce. Poté se na disku provedou dílčí operace. Jestli uspějí všechny, transakce se ze žurnálu uvolní. Jestli ne, dokončí se všechny transakce, které jsou v žurnálu zapsány celé.
    - Implementace na základě anulace transakcí – *UNDO*:
        - Záznam dílčích operací do žurnálu a na disk se prokládá. Proběhne-li celá transakce, ze žurnálu se uvolní. Při chybě se eliminují nedokončené transakce.
    - UNDO+REDO (NTFS)
    - implementace musí zajišťovat správné pořadí zápisu operací, které ovlivňuje plánování diskových operací v OS, popř. jejich přeuspořádání v disku
- Alternativy
    - *Copy-on-Write*
        - nejprve zapisuje nová data/metadata na disk, pak je zpřístupní: Změny provádí hierarchicky v souladu s hierarchickým popisem obsahu disku (jde o vyhledávací, nikoli adresovací strom). Vytvoří kopii měněného uzlu -> upraví ji -> vytvoří kopii uzlu nadřazeného -> upraví ji tak, aby ukazovala na uzel vytvořený v předchozím kroku. Na nejvyšší úrovni se udržuje několik verzí kořenového záznamu se zabezpečovacím kódem a časovými razítky. Nabízí bázi pro implementaci: snímků souborového systému a klonů SS – vznikají částečně sdílené stromové struktury popisující různé verze obsahu SS

21. Soubor popsaný B+ stromem [pr 4; sl 27-29]

- v listech jsou 3 odkazy, jeden je potřebný pro provázání do seznamu, zbývají tedy dva odkazy na data a celkem jsou zapotřebí 3 listy
- 3 listy lze adresovat z jednoho kořene
- B+ stromy se dále užívají k popisu
    - obsahu adresářů
    - volného prostoru na disku

22. Alokační blok, Diskový sektor, Extent [pr.4; sl 11]

- *Diskový sektor:* nejmenší jednotka, kterou disk umožňuje načíst/zapsat
- Velikost sektoru: typicky 512B, u nových disků 4096B (s emulací 512B)
- Adresace sektorů:
    - CHS = Cylinder, Head (typicky 1-6 hlav), Sector

    - LBA = Linear Block Address (číslo 0..N)
- *Alokační blok:* skupina pevného počtu sektorů, typicky 2^N pro nějaké N, následujících logicky (tj. v souboru) i fyzicky (tj. na disku) za sebou, která je nejmenší jednotkou diskového prostoru se kterou umí OS (FS) pracovat. (nejmenší jednotka, kterou umožňuje OS načíst/zapsat)
- *Extent* - posloupnosti proměnného počtu bloků jdoucích za sebou logicky v souboru a uložených fyzicky za sebou na disku. Zrychlují práci s velkými soubory(menší indexovací struktury, menší objem metadat), naopak pro malé soubory to může znamenat zbytečně velkou režii

23.Překlad logické adresy na fyzickou přes 2-úrovňovou tabulku stránek (13b)

- LA se rozloží na prefix p sestávající z odkazu p1 do adresáře stránek a z odkazu p2 do konkrétní tabulky stránek. Tedy p=p1.p2
- Test zda TLB obsahuje dvojici (p,f), vyhledává se asociativně dle p (je možné částečně asociativní vyhledání), je-li položka (p,f) nalezena, je výsledná adresa (f,d).
- Přístup k položce na indexu p1 v adresáři stránek, který je v RAM na adrese uložené ve speciálním registru MMU (u intelu CR3)
- Má-li výše uvedená položka nastaven příznak neplatnosti, nastává výpadek v adresáři stránek
- Není-li tomu tak, pak se z uvedené položky získá bázová adresa dílčí tabulky stránek rovněž uložené v RAM. Použije se položka s indexem p2 této tabulky.
- Obsahuje-li položka na indexu p2 v dílčí tabulce stránek příznak neplatnosti, nastává výpadek stránky. Jinak se z položky zmíněné výše získá číslo rámce f a výsledná adresa je (f,d)

24. Synchronizace tří procesů

```
Init:     init(s1, 0);
          init(s2, 0);
          init(s3, 0);

p1:       printf("p1: krok 1\n");
          unlock(s2);
          lock(s1);
          printf("p1: krok 2\n");

p2:       lock(s2);
          printf("p2: krok 1\n");
          unlock(s3);
          lock(s2);
          printf("p2: krok 2\n");
          unlock(s1);

p3:       lock(s3);
          printf("p3: krok 1\n");
          printf("p3: krok 2\n");
          unlock(s2);
```

25. Inverze priorit [pr 5; sl 26, 27]

Problém inverze priorit:

- Nízko prioritní proces si naalokuje nějaký zdroj, více prioritní procesy ho předbíhají a nemůže dokončit práci s tímto zdrojem.
- Časem tento zdroj mohou potřebovat více prioritní procesy, jsou nutně zablokovány a musí čekat

na nízko prioritní proces.
- Pokud v systému jsou v tomto okamžiku středně prioritní procesy, které nepotřebují daný zdroj, pak poběží a budou dále předbíhat nízko prioritní proces.
- Tímto způsobem uvedené středně a nízko prioritní procesy získávají efektivně vyšší prioritu.

Inverze priorit nemusí, ale může, vadit; může zvyšovat odezvu systému a způsobit i vážnější problémy, zejména pokud jsou blokovány nějaké kritické procesy reálného času (ovládání hardware apod.).

Možnosti řešení inverze priorit:
- **Priority ceiling**: procesy v kritické sekci získávají nejvyšší prioritu.
- **Priority inheritance**: proces v kritické sekci, který blokuje výše prioritní procesy, dědí (po dobu běhu v kritické sekci) prioritu čekajícího procesu s největší prioritou.
- **Zákaz přerušení po dobu běhu v kritické sekci** (na jednoprocesorovém systému): proces v podstatě získá vyšší prioritu než všichni ostatní.

Další výraznou komplikaci plánování představují:
- **Víceprocesorové systémy** - nutnost vyvažovat výkon, respektovat obsah cache procesorů, přip. lokalitu paměti.
- **Hard real-time systémy** - nutnost zajistit garantovanou odezvu některých akcí.

27. Nežádoucí účinky využití operace spinlock

Při dlouhém čekání může blokovat prostředky jiným procesům, které pak nemohou pokračovat. Zatěžuje procesor, protože proces stále běží ve smyčce.

28. NTFS, odlišnosti různě dlouhých souborů, 3 varianty reprezentace a adresovaní.

- MFT (Master File Table)
  - alespoň jeden řádek pro každý soubor
  - malé soubory jsou celé uložené v MFT (i s obsahem)
  - střední – na řádku příslušeného souboru uložené odkazy na extenty
  - velké – přes 2 řádky jsou odkazy na pomocné odkazy, ve kterých jsou uložené odkazy na extenty (adresování podobné jako u B+ stromů)
- Extent
  - posloupnoust proměnného počtu alokačních bloků logicky jdoucích za sebou v souboru a uložených fyzicky na disku
- podpora žurnálování (REDO+UNDO)

Převzaté od : Pavel Filgas

Příloha

Konkrétní příklady

1. Zadání: Jaký je maximální počet výpadků stránek v systému se stránkami o velikosti 4 kB, 4-úrovňovou tabulkou stránek, u které pouze dílčí tabulka nejvyšší úrovně je chráněna proti výpadku, při provádění předem načtené instrukce o délce 4 B, která přesouvá 8 kB z jedné adresy paměti na jinou?

Řešení: (takhle bych to psal na písemku)

- instrukce se může rozdělit do dvou stránek, ale je předem načtená (0)
- data se mohou rozdělit až do tří stránek, čtení i zápis
  - výpadek může nastat při samotném čtení dat (3 * 1)
  - výpadek může nastat při cestě ke stránkám, avšak dílčí tab. je chráněná (N - 1 → 4 - 1)
  - cesta ke stránkám se může provést 2x, kvůli rozděleným datům do tří stránek (2 * (N - 1) → 2 * (4 - 1))
  - to vše se může opakovat 2x kvůli čtení a zápisu (2 * (3 + 2 * (4 - 1)))
- 2 * (3 + 2 * (4 - 1)) = **18**

2. Zadání: Architektura se šířkou slova 256 bitů používá stránkovaný přístup do paměti, velikost stránky je 2^{32} B, hierarchická tabulka stránek a dílčí tabulky mají velikost vždy právě 1 stránky a položky o velikosti 256 b. Kolika úrovňovou tabulku stránek může systém používat, když stránky jsou plně využitelné, v tabulkách nejsou nevyužité řádky a maximálně horních 16b adresy není použito pro adresování?

Řešení: (takhle bych to psal na písemku)

- 32 b pro posuv
- 256 - 32 = 224 b pro stránkování
- 256 = 2^5 B
- 2^32 / 2^5 = 2^27 řádků v dílčí tab. str.
- 27 b pro adresaci řádků
- 224 / 27 = 8,296 → **8 odkazů**
- kontrola: 8 * 27 = 216; 224 - 216 = 8 → 8 < 16 (správně)

3. Zadání: Jaký je maximální počet přístupů do paměti při použití 4-úrovňové tabulky stránek u instrukce o délce 4 B přistupující pro 4 B do paměti (čtení nebo zápis)? Jak se tento počet změní v případě přesunu v paměti? Jak se toto maximum změní, víme-li, že se jedná o instrukci, která v kódu lineárně následuje za aktuálně prováděnou instrukcí?

Řešení: (takhle bych to psal do písemky)

a. modrá
   - instrukce se může rozdělit do dvou stránek
     - cesta ke stránkám (N = 4)
     - čtení stránky (1)
     - to vše 2x, kvůli rozdělení (2 * (N + 1))
   - data se mohou rozdělit do dvou stránek
     - cesta k datům (N = 4)
     - čtení stránky (1)
     - to vše 2x, kvůli rozdělení (2 * (N + 1))
   - (2 * (4 +1)) + (2 * (4 + 1)) = **20**

b. červená
   - stejný postup jako v a), jen se přístup k datům zopakuje
   - 2 * (2 * (4 + 1)) + (2 * (4 + 1)) = **30**

c. zelená
   - instrukce se může rozdělit do dvou stránek, ale jedna stránka již je načtená z předešlé instrukce
     - čtení instrukce v načtené stránce (1)
     - čtení instrukce v nenačtené stránce (1)
     - čtení k nenačtené stránce přes úrovně (N = 4)
     - s daty je to totožné jako předešlé příklady
   - vyberu variantu a); (2 * (4 + 1)) + (1 + 4 + 1) = **16**

4. Zadání: Předpokládejme systém s virtuální pamětí, ve které se používá lokální výměna stránek a ve kterém se při výběru odkládané stránky používá algoritmus LRU. Předpokládejme dále, že v tomto systému je spouštěn proces, který má napevno přiděleny 4 rámce, do nichž na začátku není namapována žádná stránka. Ke kolika výpadkům stránky dojde, pokud daný proces postupně přistoupí k následujícím stránkám: 1 2 3 4 4 1 5 2 2 3 5 1.

Řešení:
- rámec je prázdný, vejdou se do něj první 4 stránky (4 výpadky) (obsah rámce: 1 2 3 4; zbývá: 4 1 5 2 2 3 5 1)
- 4 již v rámci je (0 výpadků) (obsah rámce: 1 2 3 4; zbývá: 1 5 2 2 3 5 1)
- 1 již v rámci je (0 výpadků) (obsah rámce: 1 2 3 4; zbývá: 5 2 2 3 5 1)
- 5 není v rámci, dle algoritmu nahradíme za příslušnou hodnotu, v tomto případě za 2 (1 výpade k) (obsah rámce: 1 5 3 4; zbývá: 2 2 3 5 1)
- 2 byla v předešlém kroku vyhozena (riziko LRU algoritmu), nahradíme za 3 (1 výpadek) (obsah rámce: 1 5 2 4; zbývá: 2 3 5 1)
- 2 již v rámci (0 výpadků) (obsah rámce: 1 5 2 4; zbývá: 3 5 1)
- 3 není v rámci, nahradíme za 4 (1 výpadek) (obsah rámce: 1 5 2 3; zbývá: 5 1)
- 5 je v rámci (0 výpadků)
- 1 je v rámci (0 výpadků)
- 4 + 0 + 0 + 1 + 1 + 0 + 1 + 0 + 0 = **7**

OTAZKY Z FITUSKY:

Zadání A
Počítání čtení bloků - /symlink, v symlinku /file - 6b
Implenentace monitoru s notify - 8b
Popsat CFS - 6b
Implementace hledání v invertované tabulce + struktura - 9b
Rozepisovací:
pojmy (co OS udělá po failu stránky, FIFO, LRU) - 16b
Deadlock - kompletně, do detailu - 15b

1. Copy-on-write
2. vzorec pre maximalny pocet vypadkov stranky (N-urovnova tabulka stranok, instrukcia 4 B, data 1 B alebo tak nejak)
3. priklad s hypotetickou vypocetnou architekturou
4. lock so Swap
5. popisat preemptivne, nepreemptivne planovanie, FCFS, round-robin, viacurovnove planovanie, viacurovnove planovanie so spatnou vazbou a popište jeho variantu používanou v praxi."(Myslím, že tím mysleli O(1))

(
No to pockej... O(1) Neni viceurovnove planovani...8
Tam se myslelo to, ze normalni se zpetnou vazbou ti dava dynamickou prioritu na max, a pak az klesa, kdezto v linuxu mas 100 statickych urovni, kde mas 1 az 99 pro Real-Time procesy a 0, kde je dynamicka priorita -20 az 19, a tam se pouzivalo kdysi O(1), a ted CFS
O(1) je tedy zakladni planovac, a tam mame popsat kompletni viceurovnove planovani v Linux 2.6

EDIT: Bud to, a nebo chtel jen viceurovnove se zpetnou vazbou s tim, ze statickou zakladnu dynamicke urovne nastavuje uzivatel (treba v polovine), a neni hned na nejvyssi
priorite.
)
6. deadlock

Otázky skupiny C:
1. Co udělá žurnál s jeho pokrytými operacemi? Jaké dva pozitivní dopady to má? Jak funguje u žurnálování REDO?
2. Deadlock
3. Jaký je max počet výpadku stránek v systému se stránkami o velikosti 4KiB, N-úrovňovou tabulkou stránek (N>=2), kde jen dílčí tabulka nejvyšší úrovně je chráněná proti výpadku, při provádění předem nenačtené instrukce o délce 4B, která zapisuje 8KiB do paměti RAM? Dilčí tabulky mívají velikost 1 stránky.
4. Napsat v pseudokódu jak projít čistě hashovanou tabulku.
5. Nějaký příklad na semafory.
6. Nějaký výtahový algoritmus či co.

1) Plánovač diskových operací, výtahový algoritmus, alespoň tři heuristiky používané plánovačem, modifikace výtahového algoritmu.
2) Vše o CFS.
3) Žurnálování, pokrytí instrukcí, dopady, fungování REDO.
4) Jaký je max počet výpadku stránek v systému se stránkami o velikosti 4KiB, N-úrovňovou tabulkou stránek (N>=2), kde jen dílčí tabulka nejvyšší úrovně je chráněná proti výpadku, při provádění předem nenačtené instrukce o délce 4B, která zapisuje 8KiB do paměti RAM? Dilčí tabulky mívají velikost 1 stránky.
5) Implementace semaforů - tři procesy, tři semafory, tisk pX: krok 1 (3třikrát s ruzným X) a pak v opačném pořadí pouze místo krok 1 je krok 2
6) Implementace čistě hashované tabulky stránek
7) Uváznutí

1) problémy okolo zápisu u SSD
2) diskový sektor, alokační blok, extent
3) hashovaná invertovaná tabulka
4) implementace lock s aktivním čekání v podobě swapu
5) postup při výpadku stránky, FIFO, LRU
6) deadlock

Skupina: A

Strčuně odpověz:

1) Plánovač (disk), výtahový algoritmus etc. (6b)
2) CFS - Completely Fair Sheduler (6b)
3) Implementovat monitor v C (s notify) (9b)
4) Implementovat překlad stránky na rámec pomocí invertované tabulky stránek (8b)
Rozveď:
5) Deadlock (15b)
6) Postup při výpadku stránky, FIFO, LRU (16b)

Skupina: B

Stručně odpověz:
1) Problematika zápisu u SSD (kromě omezeného počtu přepisů), 3 příklady minimalizace (6b)
2) Výpočet max. počet výpadku stránek pro N-úrovňovou tabulku stránek (nejvyšší chráněná proti výpadku) 4B instrukce čte 1B dat (6b)
3) Vypočítat max. počet úrovní stránek pro hypotetickou architekturu o délce slova 128 bitů (8b)
4) Implementovat semafor v C (s atomickou instrukcí swap) (9b)
Rozveď:

5) Preeptivní/Nepreemptivní polánování, FCFS, round-robin, víceúrovňové plánování, víceúrovňové plánování se zpětnou vazbou (16b)
6) Deadlock (15b)

1. co musi OS udelat pri vypadku stranky, LRU
2. co je to planovac a dispecer, preemptivni, nepreemptivni planovani
3. co to je diskovy sektor, alokacni blok, extent
4. priklad na pocet vypadku stranky, 4B instrukce, 24KiB data, N-urovni tabulek stranek pro N >=2, nejvetsi uroven chranena proti vypadku, velikost stranky 4KiB, a instrukce byla: zapsat 24KiB dat (napriklad nulovani)
5. kod na semafor je myslene ako SpinLock -> swap tam bol .. pseudokod a podobne ...
6. deadlock

ODPOVĚDĚT STRUČNĚ

1) (7b) Co zajišťuje v souborovém systému žurnálování s ohledem na jím pokryté operace? (Tedy jakým přívlastkem lze označit operace pokryté žurnálováním?) Jaké dva pozitivní dopady tato

skutečnost má? Popište základní prinncip strategie REDO používané při implemetaci žurnálování. (cca 5 vět)

2) (8b) Charakterizujte, jak je popisováno rozložení dat na disku v systémech Windows používajících
souborový systém NTFS, a to včetně odlišností reprezentace různě dlouhých souborů. Definujte pojem extent.
(cca 5 vět)

3) (8b) Mějme 3 procesy p1, p2 a p3 a 3 semafory s1, s2 a s3. Zapište v pseudokódu inicializaci semaforů s1, s2 a s3 a části řídicího kódu procesů p1, p2 a p3, které s pomocí semaforů s1, s2 a s3začínající "pi:" vypisuje proces pi pro i ∈ {1, 2, 3} a současně je každý řádek vypsán příslušným procesem samostatně:
zajistí, že tyto procesy vypíší následujících 6 řádků textu v zadaném pořadí, a to tak, že řádek

p1: krok 1
p2: krok 1
p3: krok 1
p3: krok 2
p2: krok 2
p1: krok 2
(limity: 4 úseky pseudokódu, nejdelší z nich o 6 řádcích)

4) (9b) Specifikujte v pseudokódu operaci TestAndSet používanou v atomické podobě pro synchronizaci s podporou hardware. Specifikujte v pseudokódu synchronizaci na kritické sekci s využitím této operace. Dále specifikujte v pseudokódu synchronizaci na kritické sekci s využitím atomické operace Swap. Jaká je základní nevýhoda uvedeného typu synchronizace? Kde je přesto možné tento typ synchronizace použít (uveďte obecnou charakteristiku, nikoliv konkrétní příklad)? Jaký problém je spojen s použitím uvedeného typu synchronizace na víceprocesorových systémech? Uveďte v pseudokódu modifikaci základního použití atomické operace TestAndSet, která tento problém řeší.
(limity: 4 krátké úseky pseudokódu, 3 rozvité věty)

ROZVEĎTE

5) (13b) S maximální přesností popište algoritmus převodu logické adresy na adresu fyzickou v systémech s dvouúrovňovou hierarchickou tabulkou stránek a TLB. Algoritmus převodu popište ve formě posloupnosti číslovaných kroků. Můžete se opřít o schematický obrázek, ale důležitý je zejména k němu uvedený popis algoritmu převodu (obrázek bez popisu je za 0b). U každého dílčího přístupu do paměti uveďte (a) s jakou pamětí se pracuje a (b) jakým způsobem se daná paměť adresuje. Uveďte, jak se detekuje, že dojde k výpadku stránky, ovšem samotná obsluha výpadku již není předmětem otázky. Kontroly práv pro čtení/zápis ignorujte.

Upozornění: Výroky typu "v paměti se najde" či "paměť se zaindexuje" jsou bez dalšího vysvětlení zcela nedostatečné! Kdykoliv se pracuje s nějakou tabulkou, uveďte, v které paměti se nachází (disk, RAM, či jiný typ paměti), jak konkrétně se určí její bázová adresa a jak index, případně jaký jiný typ adresování se použije.

6) (15b) Definujte pojem uváznutí (deadlock) při práci se sdílenými zdroji a uveďte 4 nutné (tzv. Coffmanovy podmínky) jeho vzniku. Uveďte základní princip prevence uváznutí a popište co nejdetailněji různé přístupy, které se k tomuto účelu používají. Uveďte základní princip vyhýbání

se uváznutí a zvláštní pozornost pak věnujte pečlivému popisu grafu alokace zdrojů a jeho použití pro vyhýbání se uváznutí. Jaký další klasický mechanismus řešení problému uváznutí se používá (neuvažujeme-li využití verifikace k ověření, že uváznutí nemůže nastat)? Stručně charakterizujte.

1. Odpovězte stručně

1. (6 bodů) Jaký je maximální počet čtení bloku z disku při provedení operací h = open(„/symlink“, O_RDONLY); read(h, buf, 10); read(h, buf, 10); Předpokládejte přitom, že adresáře jsou kratší, než 1 blok, symlink je symbolický odkaz na soubor /file, file je klasický soubor delší, než 20B, žádný blok není na počátku ve vyrovnávací paměti a používají se všechny běžně používané vrovnávací paměti, nedochází k interferenci s dalšími procesy, přepnutí kontextu, příchod signálů apod. (limity: 1 číslo [bez zdůvodnění za 0b] a cca 6 mírně rozvitých vět)

2. (6 bodů) Charakterizujte základní myšlenku a princip činnosti plánovače CFS (Completely Fair Scheduler) implementovaného v Linuxu 2.6. (limit cca 6 rozvitých vět)

3. (9 bodů) Uvažujme strukturu struct sMonitor se členy semaphore mutex, semaphore condition a int waiting, kterou hodláme použít pro simulaci monitoru v jazyce C, který monitory přímo nepodporuje. Předpokládáme přitom, že daný monitor bude mít napevnojednu podmínku pro čekání „uvnitř“ monitoru, pro jejíž implementaci bude použit semafor condition a čítač waiting. Simulace bude probíhat tak, že na začátku každé operace, která má být monitorem chráněna, bude volána funkce void enter(struct sMonitor *monitor) a na konci každé takové operace bude voláno void leave(struct sMonitor *monitor). S čekací podmínkou bude možno pracovat pomocí funkcí void wait(struct sMonitor *monitor) a void notify(struct sMonitor *monitor), u kterých předpokládáme, že jsou vždy volány mezi voláním enter a leave na patřičném monitoru. U funkce notify předpokládáme, že pokud nikdo na notifikaci nečeká, jedná se o prázdnou operaci. Poradí notifikovaných procesů není podstatné. Implementujte funkce enter, leave, wait a notify. (limity: nejdelší z funkcí cca 4 řádky kódu)

4. (8 bodů) Předpokládejte existenci struktury struct inv_tab_item popisující v jazyce C položku invertované tabulky stránek. Dále nechť číselný typ page_num_t popisuje čísla stránek a rámců a číselný typ pid_t čísla procesů. Implementujte v jazyce C funkci pro převod čísla stránky na číslo rámce s prototypem page_num_t page_to_frame(inv_tab_item *inv_tab, page_num_t max_frame_num, pid_t pid, page_num_t page_num); Předpokládejte přito, že parametr inv_tab ukazuje na první položku invertované tabulky stránek, max_frame_num je nejvyšší možné číslo rámce v systému, pid je číslo procesu, který převod provádí a page_num je číslo převáděné stránky. Funkce v případě, kdy je možno dané číslo stránky převést na odpovídající číslo rámce, vrací příslušné číslo rámce. Pokud převod není možno provést, funkce vrátí hodnotu PAGE_FAULT. Pro potřeby implementace fce page_to_frame si doplňte potřebné členy struktury inv_tab_item, Popište, se kterými v implementaci počítáte. (limity: patřičný počet členů struktury inv_tab_item, 4 řádky kódu pro tělo funkce bez deklarací) 2. ROZVEĎTE

5. (15 bodů) Definujte pojem uváznutí (deadlock) při práci se sdílenými zdroji a uveďte 4 nutné (tzv. Coffmanovy podmínky) jeho vzniku. Uveď jute základní princip prevence uváznutí, popište co nejdetailněji různé přístupy, které se k tomuto účelu používají. Uveďte základní princip vyhýbání se uváznutí a zvláštnípozornost pak věnujte pečlivému popisu grafu alokaci zdrojů a jeho použití pro vyhýbání se a uváznutí. Jaký další klasický mechanismus řešení problému uváznutí se používá (neuvažujeme-li využití verifikace k ověření, že uváznutí nemůže nastat)? Stručně charakterizujte.

6. (16 bodů) Co musí udělat operační systém při výpadku stránky? (Popište co nejpřesněji všechny možné varianty.) Jaká je základní myšlenka algoritmu FIFO, jeho výhody a nevýhody a v jaké podobě je možné ho v praxi používat? Jaká je základní myšlenka algoritmu LRU pro výběr tzv. victim page, jeho základní výhoda a nevýhody a základní princip dvou variant, ve kterých se běžně užívá v praxi

https://fituska.eu/download/file.php?id=11631 <<<pridavajte odpovede na otazky ako komentare z číslom otazky

https://fituska.eu/download/file.php?id=11640&mode=view <<<pridavajte odpovede na otazky ako komentare z číslom otazky

A dalšie otazky z fitušky nech je to tu pokope:

Vložené otázky

#242

Otazka:
Alokační blok (Cluster)

Nejmenší logická část souborového systému. Cluster je vždy 2^n sektorů následujících za sebou.ejmenší logická část souborového systému. Cluster je vždy 2^n sektorů následujících za sebou.

#108

Otazka:
extent

-posloupnost proměnného počtu bloků uložených na disku fyzicky za sebou
-extent udává, kde na disku začíná a kolik bloků obsahuje
-snižuje se počet metadat, zrychluje se čtení velkých souborů
-používají se např v B+ stromech

x

#107

Otazka:
žurnálování, žurnál ...

Slouží pro záznamy změn u metadat/dat před jejich zápisem na disk.
- používají je např ext3, ext4, ReiserFS, NTFS
- většina dat žurnálována není (příliš velká režie)

#106

Otazka:
start systému:

1.BIOS
2.inicializace služeb jádra+jádro samotné
3.swapper
4.init

#105

Otazka:
dynamická změna priorit?

#104

Otazka:
jak se vypíná obsluha přerušení?

#103

Otazka:
při přepínaní kontextu se zálohuje?
? pouze poslední instrukce a registry, nebo vše? celý PCB, pamět ... atd ?

registry a pokud to jde, tak na zbytek pouze ukazatele

#102

Otazka:
možná implementace semaforu (je nutné zajistit aby semafor proběhl jako atomická instrukce) využije se vlastnosti spinlocku ...

```
typedef Struct {
        int value;
        fronta_procesu *queue;
        bool lock;
} semaphore;
lock(S){
         while(testAndSet(S.lock)); // spinlock
```

```
        S.value--; //nevim proc
        if(S.hodnota < 0){
                C = get(ready_queue); // vytahnu z fronty procesu pripravenych bezet
                append(S.queue, C); // dam na forntu cekajicich procesu na semaforu
                S.lock = false; // vychzim z KS
                switch(); // vzdam se jadra
        }
        else{
        S.lock = false; // vychazim z KS
 }
// zkuste nekdo napsat jak by se zamykalo se spinlockem ...

z přednášky č. 6
bool TestAndSet(bool &target) {
        bool rv = target;
        target = true;
        return rv;
}
```

Tento kód popisuje chování spinlocku, který je ovšem řešen hardwarově za účelem dosažení atomicity.
S.value--; je tam proto, že při zamčení semaforu se dekrementuje jeho hodnota.

#101

Otazka:
Jak probíhá překlad adres při stránkování?

zde si myslim, ze by vojnar chtel implementaci v nejakem pseudo kodu... aspon o prednaskach to porad naznacoval, ze bychom to meli zvladnout...

```
pg_num page; //cislo stranky
pg_num PageTable; //velikost tabulky
int page_item[PageTable];
fr_num frame; //cislo ramce
if (page<=PageTable)
{
        frame=page_item[base+page];
        physic_adress=frame+offset;
}
else SIGSEG;
```

může být něco takového? vůbec nevím, jak si to oni představují...

#100

Otazka:
Napište pseudokód segmentace paměti.

#99

Otazka:

úrovně běhu, co znamenají? které se využívají?

0..halt
1..single user
6..reboot
2-5..víceuživatelské užití, grafický režim, síťový režim,
grafický+tíťový,.. záleží na distribuci
změna telinit N (kde N je 0-6 viz výše)Je to ono, na co se ptají? :D

ano je to presne ono, plus este urovne 's' a 'S'

#98

Otazka:
Co je hw přerušení, co je řadič přerušení, jak lze zakázat
přerušení, co je NMI, rozdělení na úrovně ...

HW přerušení je mechanismus, kterým HW zařízení oznamují jádru
asynchronně vznik událostí, které je zapotřebí obsloužit.
Řadič přerušení je řadič, do kterého přicházejí žádosti o HW
přerušení. Na PC se jmenuje APIC, kde každý procesor má vlastní lokální
APIC.
Řadič může být naprogramován tak, aby maskoval určitá přerušení.
Obsluhu přerušení lze také zakázat na procesoru, případně čistě
programově v jádře.
NMI (Non-mascable interrupt) je HW přerušení, které nelze maskovat na
řadiči, ani zakázat jeho přerušení na procesoru.
Obsluha přerušení bývá rozdělena na 2 urovně:
1) Zajišťuje minimální obsluhu HW a plánuje běh obsluhy 2. úrovně.
2) Postupně řeší zaznamenaná přerušení.

#97

Otazka:
Mikrojádra - popis, výhody, nevýhody

minimalizuju rozsah jadra, jednoduche rozhranie, jednoduche abstrakcie, maly
pocet sluzieb
vacsina sluzieb je implementovana mimo jadro v tzv. serveroch, teda nebezi v
privilegovanom rezime, teda je to bezpecnejsie a flexibilnejsie
nevýhody: vyšší režie -> nižší efektivita

malým počtem služeb = pouze základní správa procesoru, I/O, paměti a
meziprocesorové komunikace
výhody:
flexibilita - více současně běžících služeb
zabezpečení - servery neběží v privilegovaném režimu, chyba nevede hned k
selhání OS
nevýhody:
vyšší režie - vyšší problém u mikrojader 1.generace, lepší u 2.
generace - ale stále přetrvává

jabaduba ma ty vyhody a nevyhody spravne

J

#96

Otazka:
OS - definice, role, cíle atd.

Operační systém je program (případně soubor programů), které tvoří spojující mezivrstvu mezi HW počítače (který může být virtualizován) a uživatelem (aplikačními programy uživatele).
Cílem OS je maximálně využít zdroje počítače a zjednodušit práci s ním.
OS je správcem prostředků počítače a tvůrcem prostředí pro uživatele a jejich programy (poskytuje standardní rozhraní a abstrakce).
OS se dělí na jádro, systémové knihovny a utility a textové a/nebo grafické uživatelské rozhraní.

#95

Otazka:
Vše, co víte o deadlocku (cca za 15b)

Vzniká když více procesů chtějí dvě zařízení. A nastane situace že 1. proces si zabere jedno zařízení a 2. zařízení druhé a oba čekají na jejich druhé zařízení které je již zabrané.
řešení: výlučný přístup, postupné přidělování prostředků, odebrání zařízení po určité době
podmínky uváznutí:
vzájemné vyloučení při používání prostředků
vlastnictví alespoň jednoho zdroje a čekání na další
prostředky vraci pouze proces po dokončení jejich využití
cyklická závislost na sebe čekajících procesů
prevence:
-zruší se některá z platnosti podm. uváznutí
1.u prostředků, které umožňují (současný) sdílený přístup, nejsou zámky zapotřebí
2.proces může žádat o prostředky pouze pokud žádné nevlastní
3.pokud proces požádá o prostředky, které nemůže momentálně získat, je pozastaven, všechny prostředky jsou mu odebrány a čeká se, až mu mohou být všechny potřebné prostředky přiděleny
4.prostředky jsou očíslovány a je možné je získávat pouze od nejnižších čísel k vyšším
vyhýbání:
- procesy před spuštěním deklarují určité informace o způsobu, jakým budou využívat zdroje: v nejjednodušším případě se jedná o maximální počet současně požadovaných zdrojů jednotlivých typu.
- předem známé informace o možných požadavcích jednotlivých procesu a o aktuálním stavu přidělování se využijí k rozhodování o tom, které požadavky mohou být uspokojeny (a které musí

počkat) tak, aby nevzniklo cyklické cekání.
zotavení a uváznutí:
-ukoncení všech nebo nekterých zablokovaných procesu,
-odebrání zdroje nekterým procesum, anulace jejich nedokoncených operací (rollback) a pozdeji restart.

#94

Otazka:
cituji(z fb, snad dotycne nebude vadit): loni jsem měla otázku NTFS... co by ste k tomu kdo napsali? ono tam toho totiž moc není.

Také z fb: Nakreslil MFT, trosku ji popsal a potom nakreslil rozlozeni v te tabulce.
este tam napriklad mozes dodat, ze ma zurnalovanie....
Vyuziva istu modifikaciu B+ stromov, nazyvanych tiez H-stromy. Na rozdiel od Unixu metadata obsahuju aj nazov suboru. Ak sa specifikacia a pristupove prava nevojdu do vyhradneho priestoru, alokuje si dalsie riadky. Kazdy subor ma defaultne "nachystany" jeden riadok v tabulke. Oblast pre data
obsahuje adresu zaciatku extendu a to ako aj logicku tak aj fyzicku a velkost. Opat, ak je pocet riadkov adresuje sa priamo, ak nie alokuju sa dalsie riadky na ktore sa odkazuje, podobne ako v i-uzloch.
?+ to, že je to proprietární FS od Microsoftu (docela podstatná informace

tady je obrazek:
http://pages.cs.wisc.edu/~bart/537/lecturenotes/figures/mft-entry-extent.gif

Mi