dx s xRT_2016 TEÓRIA

hlavne to nezdielajte po skupinach a neukazujte Vilkovi :)ok (Dr. Solčány, have mercy please...)...

//[OFF] https://www.youtube.com/watch?v=1TxfYt25amQ
//(02.02.2016 12:59 Tony ->) som mal vcera na skuske chut mu to ta

nový doc pre skúšku 2016/17...pridávajte otázky čo boli, ak viete aj odpovede:
https://docs.google.com/document/d/1Dbg4SMcr5hNYIBTrdI5WQI25JgtEYKx5G2HhTLodODY/edit?usp=sharing

Udacity ma super kurz, Advanced Operating Systems,
kde je vysvetlena vacsina veci - napr. https://youtu.be/tMVj22EWg6A (inodes),
alebo https://youtu.be/V2Gxqv3bJCk (FAT) , je tam aj virtualizacia, multiprogramovanie, atd..
Mozno to niekomu pomoze :)

1.Vysvetlite pojem externá fragmentácia

Sú to voľné bloky, ktoré sú medzi obsadenými blokmi. Sú tak malé že do nich takmer nikdy nevojde to, čo potrebujeme alokovať
//mohol by to niekto prosím Vás napísať trocha, hmm lepšie? pochybujem, že za takúto odpoveď dá body
//Externá fragmentácia je, keď existuje dostatok celkového pamäťového priestoru na splnenie požiadavky, ale tento priestor nie je súvislý; pamäť je rozdelená na veľké množstvo malých dier//+1+1
//+ by som si daval pozor na slova, solcany a stein. velmi dobre rozlisuju medzi slovami pamat a diskovy priestor... zato daval body dole, mne tiez a na konzultaciach sa nechutne s vami hadali ohladom tejto tematiky ako by som im odkusol z prstu
2.Kedy sa proces môže dostať do stavu pripravený (2 príklady) a vysvetlit

1. Vyvolanie prerušenia na I/O
2. Preplánovanie aktuálneho procesu

// to nieje ukoncenie procesu, len OS naplanuje iny proces, po uplynuti casoveho kvanta // ja si myslim,ze (z blokovaneho) na pripraveny sa proces dostane ked:
1. nastane udalost na ktoru cakal
2. skonci I/O ktoru vykonaval /+1+1+1+ln(e)
// co myslis tym, nastane udalost na ktoru cakal, to moze byt viacero veci
// s tym I/O to je to iste, to mas pravdu
//neviem praveze, preto som to dal takto vseobecne, neviem ktore je spravne, ja som to mal zle. (takato odpoved figuruje aj v skuske 2013/14), ktore je teda spravne?

v prednáške č.3 strana 10 , tam je taký graf a prechod na pripravený je z
1.
+3
1. **Ak je proces blokovany a jeho podmienka blokovania skoncila**
2. **Ak bol proces beziaci a uplynulo mu casove kvantum**

//staci sa len pozriet na round robin tabulku tam je to iste…
//huradopice

3.Implementácia semafora monitorom (pozn. 18.01.2015 15:50 - to skôr bolo opačne, mali sme implementovať monitor pomocou semaforu)
// monitor pomocou semaforu je hovadina, zbytocne narocne //no neviem, je na to spraveny priklad v prednaskach... prednaska 6 strana 24 a 25
Riešenie ??
Na implemtenovanie semaforu pomocou monitora potrebujeme dalsie premenne. Kvoli tomu ze monitor ma len rad cakajucich procesov, a semafor ma aj hodnotu semaforu. Potrebujeme pomocou tych premennych zabezpecit aby sa csignal() nestratil ked nema aky proces spustit z radu cakajucich procesov a navysil v takomto pripade premennu, ktora reprezentuje hodnotu semaforu. Taktiez, aby cwait() nedal vzdy proces do radu cakajucich procesov, ale aby kludne dalej vykonaval dany proces, v pripade ze je hodnota (nasej novej premennej, ktora reprezentuje hodnotu semaforu, pre nas monitor) vacsia ako 0.

4.Linky a výhoda symbolickej linky oproti pevnej
- **softlinks can cross filesystems**
- **Symbolic links have the advantage that they can be used to link to files on machines anywhere in the world, by simply simply providing the network addresadresus of the machines where the files resides in addition to its path on that machine.**

//odkaz mimo akt. particie (napr. odkaz z C: do D:) je to výhoda, čo sa týka prístupu k dátam

b, výhoda pevnej linky oproti symbolickej

Hardlinked files stay linked even if you move either of them…

// dobrá organizácia dát, sú napojené na seba stále v jednej partícií na svojom mieste a je jasné s čím a kde systém pracuje, s akou množinou súborov môže pri prechodoch rátať

Výhodou symbolickej linky je jej univerzálnosť. Môže prekročiť hranice súborového systému, môže ukazovať aj na adresár, *nespôsobuje problémy pri aktualizácii.*
Výhodou hardlinky je, že nezaberá na disku dátový blok, ktorý obsahuje cestu k cieľu odkazu.

Diskusia:

//osobne si teda myslím podla toho čo som napísal pod to, tak v skratke SL = výhoda rýchlosti prístupu k dátam,
//-nesuhlasim, kedze pre kazdu sym. linku musi system vytvorit i-node a kym sa cez SL dostanes az k i-node toho suboru, tak to trva nejaky cas.
// HL = prehladnosť pri hladaní súboru podľa kľúča, ak niekto si mysli inak, povedzte…
//- skor: HL - vyhoda ze ak zmenis nazov suboru HL bude stale ukazovat na ten subor zatial co SL by uz nie. //toto solčány nepovažuje za výhodu (z konzultácii) //by som sa ho spytal ci je to teda nevyhoda...
wiki: výhoda - HL nezaberá na disku dátový blok, ktorý obsahuje cestu k cieľovému odkazu
// ok teda, tak sa asi mylim

http://askubuntu.com/a/801191/480822

A hardlink isn't a pointer to a file, it's a directory entry (a file) pointing to the same inode. Even if you change the name of the other file, a hardlink still points to the file. If you replace the other file with a new version (by copying it), a hardlink will not point to the new file. You can only have hardlinks within the same filesystem. With hardlinks you don't have concept of the original files and links, all are equal (think of it as a reference to an object). It's a very low level concept.

On the other hand, a symlink is actually pointing to another path (a file name); it resolves the name of the file each time you access it through the symlink. If you move the file, the symlink will not follow. If you replace the file with another one, keeping the name, the symlink will point to the new file. Symlinks can span filesystems. With symlinks you have very clear distinction between the actual file and symlink, which stores no info beside the path about the file it points to.

5.Napíš typy prostriedkov a urči klasifikáciu prostriedkov

Viacnásobne použiteľné
Opakovane použiteľné
Jednorázovo použiteľné

diskový blok - OP
stránkový rám - VP
záznam v tabuľke - OP alebo JP ak ide o dáta //ak date len OP pol boda dolu //solcany povedal ze je OP lebo tak je to v tannenbaumovi a tak to hovoril. Steiny vravel ze zalezi od implementacie ale byva to JP

vidličky pri filozofoch - OP

6. M bit pri stránkach:

a) **Ako a kde sa používa?**

Používa sa pri odstraňovaní stránky z rámu. V prípade, že má daná stránka nastavený M-bit na 1 (teda bola modifikovaná), tak sa uloží a potom odstráni z rámu. V prípade, že má M-bit na hodnote 0 (nebola modifikovaná), tak je stránka rovno odstránená z rámu.
Niektoré algoritmy ho tiež používajú pri výbere obete (NRU).

b) **Kedy a ako sa môže zmeniť?**

Pri vložení sa stránke inicializuje M-bit na 0.
Zmení sa v prípade, že stránka v ráme bola modifikovaná. 0 => 1.

7. Mame 6 CD mechanik a N procesov. Kazdy proces potrebuje 2 mechaniky, ktore si postupne zaberie. Pre ktore N nemoze nastat uviaznutie z dovodu sutazenia procesov o mechaniky? Odôvodni prečo. (Ja si pamatám ,že tých CD bolo 8 //to bola B skupina (alebo A :D))

Poznámka - riesenie berte s rezervou, vymyslel som ho ja, čakám na schválenie :)

n- pocet procesov, M - max prostriedkov na proces, k- pocet prostriedkov
n*(M-1) <= k -1 // Najhorsi stav, ked ma kazdy proces prideleny o 1 menej ako max, a musí ostať aspoň jeden prostriedok voľný, aby sme mohli naplniť požiadavku aspoň jedného procesu, ktorý uvoľní svoje prostriedky a teda nedôjde k uviaznutiu(deadlock) //+3
//otázka znie, pre ktoré N nemoze nastat uviaznutie, keď budem mať 5 procesov uviaznutie nastať môže nie ?? //nemôže, 4 procesy dostanú 1 mechaniku, 1 proces dostane 2 mechaniky, ten sa vykoná, vráti prostriedky, ktoré potom môžu použiť iné procesy//DÍK

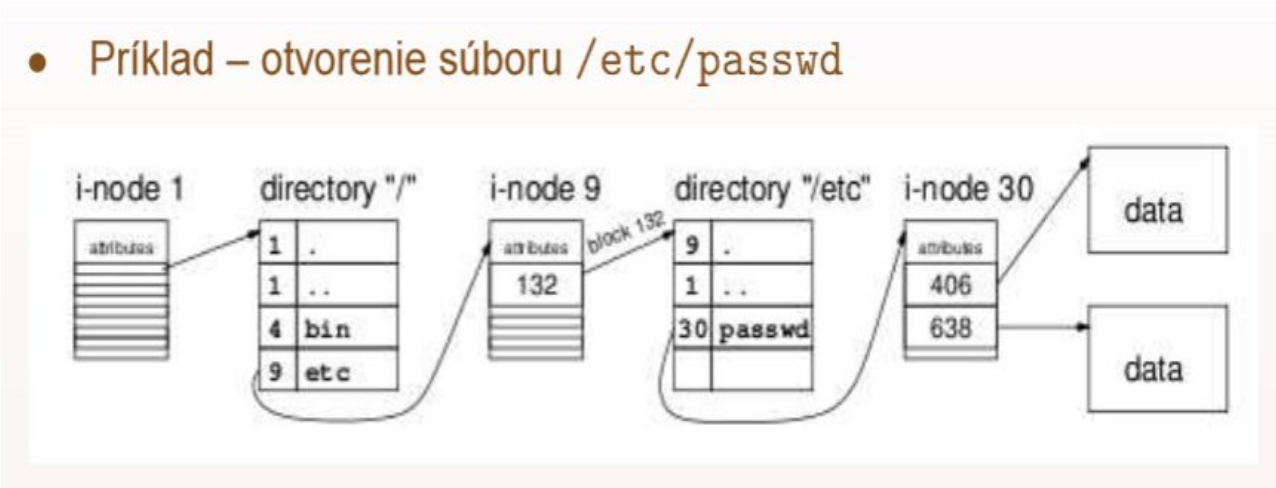
n*(2-1) <= 6 - 1
n<=5

8. tlačiareň - Čo sa používa pri riešení synchronizácie alebo tak nejak pri tlačiarni a opíš jeho činnosť

SPOOL , proces démon
S tlačiarňou priamo nekomunikujú PC ale démon, ktorý obsluhuje požiadavky.

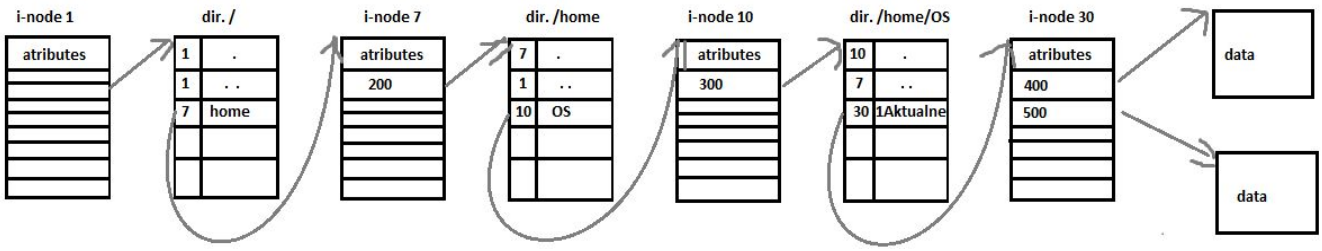
-Štefanovič : Pri obsluhe tlačiarne (nepreemptívne zariadenie) možno obísť použitie rady čakajúcich procesov pomocou metódy tzv. spooling, kedy je vytvorený špeciálny adresár, do ktorého môže ľubovoľný proces zapísať súbor dát určený na vytlačenie. Jeden osobitný paralelne bežiaci systémový proces (printer daemon) tieto súbory postupne číta, posiela ich dáta na tlačiareň a prečítané súbory vymazáva. **Je to teda jediný proces, splnomocnený komunikovať s tlačiarňou**. //+3

9. inody /home/OS/1Aktualne



+

//niekto vysvetlenie? //Aj ja by som poprosil vysvetlenie // tu Samo Kucik vyhadal od Solcanyho body, mohol by to vediet, steiny to vysvetil (ja som to neponal :D)// tie cisla napr 9 etc, 30 passwd … si mozeme vymysliet ci to je nejaka norma?:D //nejak takto pre /home/OS/Aktualne1? //No tie cisla by ma tiez zaujimali, odkial su … // to 9 etc - je cislo i-node a potom 132 je cislo bloku na disku, podla mna si to mozes vymydlíte. //+1



Vysvetlenie i-nodov, 15.12.2016 //to je Jurko Petrik :D //:DDDDD
https://photos.google.com/share/AF1QipMLwXhygS7C2ZYEwF1zDDVp3fDq4Wjgryglfp_3V181DuAG3J9vv-ovAiQXV0yahA/photo/AF1QipOjI8FsYT7UpF8lFrQdc2B_TUISGGxmBfHNmupe?key=UWQzenZvZ3RSRFZrOGtybVZjTGxZeDVyMGk1cVZR

10.preco su velkosti stranok a strankovych ramov mocniny 2?

Zjednodušuje to preklad adries. Namiesto sčítania (stránka+offset) ich stačí len zreťaziť. (Za tú druhú vetu ma včera na konzultáciach pekne zdrbal. Povedal preklad adries je správne, ale nie je to kompletná odpoveď.)// cize + asi praca s pamatou pri strankovani je rychlejsia, kvoli jej deleniu: +9
Zjednodušuje to aj indexovanie tabuľky stránok a výpočet offsetu stránky.+1 +1

//V prednaske ma, ze namiesto scitania staci len zretazit.. Vies teda kompletnu odpoved?
1.**page allocation**:
If the block of pages found is larger than that requested it must be broken down until there is a block of the right size. Because the blocks are each a power of 2 pages big then this breaking down process is easy as you simply break the blocks in half. The free blocks are queued on the appropriate queue and the allocated block of pages is returned to the caller.
2.To **translate a virtual address into a physical one, the processor must first work out the virtual addresses page frame number and the offset within that virtual page. By making the page size a power of 2 this can be easily done by masking and shifting.** Looking again at Figures 3.1 and assuming a page size of *0x2000* bytes (which is decimal 8192) and an address of *0x2194* in process *Y's* virtual address space then the processor would translate that address into offset *0x194* into virtual page frame number 1. // toto nam hovoril aj steiny na cviku, ze kvoli operacii shift treba mocninu dvojky
// tu je aj obrazok (ctrl+f → "power of 2") v http://www.tldp.org/LDP/tlk/mm/memory.html

PRÍKLADY

1. **Implementovať funkcie všeobecného semafora pomocou mutexu(binárny semafor) (pomocou binárnych semaforov, ktoré sú implementované tak, že ak nadobudnú hodnotu väčšiu ako 1 tak proces sa ukončí). Implementujte všetky tri funkcie.**

//doplnil som zadanie, presne som si to zapamatal z konzultacii 26.1.2016
//načo by som používal binárne semafóry keď by som rovno mohol použiť všeobecný? :/ Vilko už nevie čo by na tie príklady vymyslel aby to človek nespravil :(+70
// Taketo nieco som nasiel na googli, myslim si, ze je to spravne
BS1 : binary semaphore = 1;
BS2 : binary semaphore = 0;
Count: integer = initial value of counting semaphore.

```
Wait( s )
  wait( BS1 );
  Count := Count - 1;
  if Count < 0
```

```
    signal (BS1 ); wait( BS2 );
  signal ( BS1 );

Signal ( s )
  wait( BS1 );
  Count := Count + 1;
  if Count <= 0 then signal( BS2 )  // a tu ti nechyba signal(B1) ?, inak ak count<=0 tak si neodblokujes svoj mutex BS1 semafor ….
// nie, myslim ze nechyba, pretoze posledná instrukcia vo waite ( signal (BS1)) nie je podmienena if-om, ta sa vykona vzdy, nezaleziac na tom, ci je count < 0. Nizsie uvedena javovska implementacia by fungovala tiez, ale mohlo by dojst k viacerym prebiehajucim procesom signal, co vsak nie je chyba. // uz rozumiem, dakujem
  else signal( BS1 );

//  cize takto nejak?(3 funkcie, cize aj init treba)
      public class semaphore generalSem(){
        int val;
        binarySemaphore mutex1;
        binarySemaphore mutex2;

        void init (int temp){
          val = this.temp;
          mutex1.init(1);
          mutex2.init(0);
        }

        void wait(){
            mutex1.wait();
            val --;
            if (val < 0) {
              mutex1.signal();
              mutex2.wait();
            } else mutex1.signal();
        }

        void signal(){
            mutex1.wait();
            val++;
            if (val <=0) {
                  mutex1.signal(); //prečo je tu mutex1.signal? to nie je v tom riešení hore
      //Ako som uz pisal vyssie, tato implementacia je odlisna iba v tom, ze sa tu odblokuje mutex na vyvolanie signal() alebo wait(), zatialco horne riesenie caka kym sa dokonci predchadzajuci signal(). Asi to nie je chyba, ale ja osobne by som to tak radsej nedal.
                 mutex2.signal();
            } else mutex1.signal();
        }
      }  //+1


// tak ktore je teda  spravne ? mal za to niekto plny pocet ?
```

A co tak takto : ?
http://www.csc.uvic.ca/~mcheng/460/notes/gensem.pdf //4.riešenie //+1

public class GeneralSemaphoreByMutex(){==ú

```
        int val;
        binarySemaphore mutex1;
        binarySemaphore mutex2;

      void init (int temp){
         this.val = temp;
         mutex1.value = 1;
         mutex2.value = 0;
      }

      void wait(){
           mutex2.wait();      // mutex2=0 → nepreides ani prvy krat
           mutex1.wait();
           val --;
           if (val > 0) {
                mutex2.signal();
           }
            mutex1.signal();
       }

      void signal(){
                mutex1.wait();
                val ++;
                if (val == 1) {
                        mutex2.signal();
                }
                mutex1.signal();
       }
}
```

//Podla mna blbost, ved ak temp>0 tak pri zavolani wait() neprejde, ale bude cakat na prvy signal
++-++++-+*
//toto je spravne riesenie y definicie semafora vzpliva ye nemoye mat hodnotu mensiu ako 0

```
Semaphore s1;
Semaphore s2;
Int semval;
Int blocked;
Void init (int k) {
                s1.val(1);
                s2.val(0);
                semval=k;
                blocked=0;
}
Void wait () {
```

```
                s1.wait();
                If(semval >0) {
                        semval--;
                else {
                        blocked ++;
                        s2.wait();
                }
                S1.signal();
}
Void Signal () {
                S1.wait();
                If(blocked) {
                        Blocked--;
                        S2.signal();
                Else {
                        Semval++;
                }
                S1.signal();
}
```

//keď sa ti proces vo wait zabolkuje na s2 tak si skončil lebo sa nedostaneš do fukcie wait ani signal cez s1, ja by som to spravil takto:

```
Void init (i) {
        s1.init(1);
        s2.init(0);
        semval=i;
        blocked=0;
}



Void wait () {
                s1.wait();
                If(semval >0) {
                        semval--;
                        //s1.signal
                else {
                        blocked ++;
                        S1.signal();   //toto...
                        s2.wait();
                }
                S1.signal();   //...a toto tu nie je 2x? //podla mna to tu netreba ale musíš to dať hore do if za semval--; //si si isty?//treba pokial signal ostane taky aky je ale vraj to nie je dobre takze tak //alebo môžeš ešte prerobiť void signal
}

Void Signal () {
                S1.wait();
                If(blocked>0) {
                        Blocked--;
                        S2.signal();
                        //s1.signal
                Else {
                        Semval++;
```

```
                S1.signal();
            }
} //KTORE Z TYCHTO MILION RIESENI JE 100% DOBRE, MA NIEKTO TUSENIE????
//to vie iba Vilko //ja by som povedal že červený je dobre
```

//100% je iba to že umrieš
//otazka do plena, funkcia wait semval = 0 proces sa ma uspat, ale neuvolni predtym s1 nenastane deadlock? lebo s2 nie je podmienena premenna, ktora by prislusny mutex konkretne s1 uvolnila...

2. Bankový systém

Zadanie:

Máte dva účty medzi ktorými sa prevádzajú peniaze prostredníctvom funkcie `do_transfer()`. Jej kód je takýto:

```
do_transfer(unsigned int src, unsigned int dst, float amount)
{
      lock(src);
      lock(dst);
      move_money(src, dst, amount);
      unlock(src);    // tieto posledne 2 si myslim, ze boli
      unlock(dst);    // viete presne niekto ako boli? //takto
                      // aj keby boli otočené tak je to fuk,
nijako to nezabráni deadlocku.(Miko)
}
```

Funkcia `lock(unsigned int account)` - Uzamkne daný účet. Ak už účet bol uzamknutý preruší sa volanie a čaká.
Funkcia `unlock(unsigned int account)` - Ak je daný účet zamknutý tak ho odomkne. V prípade, že na jeho uzamknutie čaká iný proces, ten si ho následne opätovne zamkne.
Funkcia `move_money(unsigned int src, unsigned int dst, float amount)` - Prevedie peniaze o hodnote `amount` z účtu `src` na účet `dst`.

a) kedy a prečo môže pri tejto implementácii funkcie `do_transfer()` nastať deadlock.
b) Navrhnite vlastnú implementáciu funkcie do_transfer(), kde bude táto chyba opravená. Pri tejto implementácii môžete použiť iba funkcie `lock`, `unlock` a `move_money`.
// Pýtal som sa osobne Solčányho a môžeme tam prakticky používať všetky ostatné konštrukcie jazyka C ako aj vlastné premenné. (@Miko).

Riešenie (@Miko):

a) Deadlock nastane v prípade, že **bude existovať iný** `do_transfer()`, ktorý ale bude mať vymenené účty odosielateľa a príjmencu. Teda jeho volanie bude vyzerať takto: `do_transfer(dst, src, amount2);`. Ak by pri tomto prípade prebehol prvý riadok, uzamkneš **dst**, prerušil by si sa priebeh a nasledoval by prvý riadok ako je to v zadaní, uzamkneš **src**, tak máš uzamknuté oba účty a nastal deadlock a nie je možné sa prejsť druhým riadkov funkcie, pretože pokusy o uzmaknutie už zamknutých účtov zlyhajú.

//preco nam na toto dal iba 1 riadok? :D // ja som to napísal na druhú stranu :D // a ja som tam napisal uplnu picovinu :D ale miesta som mal dost :D // ja tiez :D .. mne dal tento koment: ???

b) ~~Moje riešenie pozostávalo z využitia jednej pomocnej premennej, pre "fiktívny účet".~~

```
do_transfer(unsigned int src, unsigned int dst, float amount)
{
    int tempAccount = 1;    // fiktívny účet

    // najprv presuniem prachy zo zdroja do tempu
    lock(src);
    move_money(src, tempAccount, amount);
    unlock(src);

    // následne až z tempu na cieľový účet
    lock(dst);
    move_money(tempAccount, dst, amount);
    unlock(dst);
}
```

~~/*~~
~~* Priznávam, že sa to bije s tým, že by som tam mal mať unsigned int ako~~
~~* parameter, ale toto nejako prežujeme :D~~
~~* Jak budú výsledky tak napíšem, čo mi zato dal nech viete :)~~
~~Pri unsigned int sa -1 v C-cku nastavi vsetky bity na jednotku teda na max mozne cislo, takze by to malo byt teoreticky OK~~
~~// ako je mi jasné, že by to prešlo kompilátorom aj by to šlo spustiť a dostali by sme číslo, ktoré zrejme nebude použité ako číslo účtu, len mi skôr šlo o princíp :D~~
~~*/~~
// Neuznal mi toto riešenie, pretože podľa popisu bolo nutné locknúť aj môj fiktívny účet, a to som nespravil. Či postačí na neho vykonať len jednoduchý lock(tempAccount) a zožral by to som sa už nedozvedel. Takže môj návrh považujte teda za zlý. (Miko)

Riešenie 2 po b): //za plný počet
//mohol by si dat komentar k tomu prosím Ťa?
//ide o to že stále **vyberieš účet s nižším číslom** teda ak ti príde že máš do_transfer zo src na dst a zároveň aj z dst na src tak ti stále lockne prvý ten menší čiže sa nemôže stať že by každý lockol jeden a nepohli sa.(Ak je src menší tak v oboch prípadoch sa najprv lockne src aj keď vystupuje v jednom ako src a v druhom ako dst) Môžeš to otočiť aby sa ti lockol stále ten väčší to je jedno.

```
do_transfer(unsigned int src, unsigned int dst, float amount)
{
        if(src<dst){
                lock(src);
                lock(dst);
        } else {
                lock(dst);
                lock(src);
        }
        move_money(src, dst, amount);
        unlock(src);
        unlock(dst);
} //+5
```

3. Monitor - Implementuj monitor mb2 tak, aby bol schopný poskytnúť funkciu void block(int i) , ktorá má za úlohu zosynchronizovať k - procesov tak, aby pred volanim funkcie niecoNechRobiaProcesy() sa procesy navzajom počkali. Pred touto funkciou sa musia zosynchronizovat volanim monitora mb2 volanim jeho funkcie barier(), ktoru implementujte + potrebne premenne a datove struktury tiez.

Ja si pamatam, ze proces nemohol dokoncit K-tu iteraciu pokial druhy proces nezacal K-tu iteraciu funkcie barrier a tak si ich mal zosynchronizovat.//+1

//časť z tohto kódu bola zadaná:

```
monitor mb2 {
  ...
  void barrier(int i){
    ...
  }
}
```

**************solution********************

```
monitor mb2 {
      int k[2]; //alebo k1,k2 whatever
```

```
	k[0]=k[1]= -1; //alebo vpodstate akakolvek hodnota
	cond queue;	m,

	void barrier(int i){
		k[i]++;

		~~while~~ if(k[0]!=k[1])
			queue.wait();
			//"sestersky" proces prejde a ja idem dalej ked
			ma zobudi signalom

		doStuff()..
		queue.signal();
	}
}
```

//takto som to skusil ja - mohol by sa niekto vyjadrit ci sa mu to zda zle/dobre budem vdacny //co si ja pamatam tak while nebol vo vnutri monitora a ani nevidim dovod preco by tam mal byt
//tak povedzme ze namiesto while by bol if - zmenilo by to nieco realne?
//mal som to cez tri ify podobnym sposobom ze sa cakali a full pocet
//tak by si mohol hodit sem to tvoje riesenie,
//tento while mi ohodnotil 6-timi bodmi, na 7 asi ten if treba (len prepisat while na if)
// ja som mal tento isty priklad pre N- procesov, to sa robi ako? //das si nejaky spolocny counter ktory bude musiet byt aspon N aby pustil dalsiu iteraciu // ano, to som mal za nula, lebo si zober ako spravis aby to vzdy fungovalo po kvantach pre N procesov a potom zasa pre N procesov, pricom musis nejako ten counter nulovat, no lenze ten si ho nuluje kazdy proces, a akonahle dekrementujes o 1 alebo 2 ci 3 moze prist v tom case iny proces a spusti to peklo znovu…
// nema tam byt csignal a cwait?

+1

```
monitor m2{
	int k[N]={0};
	int akt=1, preslo=0, blokovane=0; oldblock=0;
	cond rad1, rad2;

	void barier(int i){
		k[i]++;

		if(k[i]!=akt){
			blokovane++;
			if(akt%2==1)
```

```
                        rad1.cwait();
                else
                        rad2.cwait();
        }

        preslo++;

        nejakePičovinky();

        if(oldblock>0){
                oldblock--;
                if(akt%2==0)
                        rad1.csignal();
                else
                        rad2.csignal();
        }

        if(preslo==N){
                oldblock=blokovane;
                blokovane=0;
                preslo=0;
                akt++;
        }
    }
}
// však k[i] sa bude vždy rovnať akt, teda prvú várku procesov to vôbec
nezosynchronizuje :)
//prečo by nezosyncrhonizovalo? veď začne proces vždy prvú interáciu, a keď sa
nejaký pokúsi spraviť druhú tak sa zablokuje kým nespravia všetci prvú … kód tých
funkciá vyzerá asi tak že for(..){m2.barier(CISLO); robNieco();} => na začiatku sú
zosynronyzované tj krát to vykonali a teraz to majú spraviť prvýkrát preto sa pri prvom
zavolaní bude rovnať k[i] tomu akt, inak by to bol nezmysel :)


// Ide to o dosť jednoduchšie, malo by to vyriešiť všetko
monitor mb2{
        cond blocked;
        int block_cnt = 0;

        void barrier(){
                block_cnt ++;
                if(block_cnt < N){
```

```
                blocked.delay();
            }
            blocked.continue();
            block_cnt --;

        }
}
```

```
mb2 {
    cond blockit;

    blocked = 0;
      void barrier(){
       blocked++;
       if(blocked < N) blockit.cwait();
            else{
                  for (i = 0; i < N-1; i++){
                        blockit.csignal();
                        }
                  blocked = 0;
            }
    }
}+1
// to nad tym je to iste len tam nemusi byt  for loop ale kazdy proces
dekrementuje cislo o 1 a tym sa na konci dosiahne 0 a to iste je aj s
procesmi, budu sa navzajom odlokovavat
```

4. Nech vykonanie jednej inštrukcie trvá 100 nanosekúnd ale v prípade odvolávky sa na neprítomnú stránku pamäti (page fault) ďalších 8 milisekúnd ak sa stránka nemodifikuje a 16 milisekúnd ak sa modifikuje. Pravdepodobnosť, že sa stránka modifikuje je 50%. Priemerne každých koľko inštrukcií sa môže vyskytovať page fault, aby efektívny (priemerný) čas vykonávania jednej inštrukcie bol 800 nanosekúnd? [4b]

8*0,5+16*0,5=12

12 milisekund = (100k+12000000n)/k=800

100+12000000n/k=800

12000000n/k=700

12000000n=700k

n= 0,0000583k

k - celkovy pocet instrukcii

n - pocet instrukcii s page faultom

// **mal toto riesenie niekto za full na RT?**

// teda každých 0,0000583 inštrukcií môže byť page fault?
// vysvetlil by to niekto polopate prosim ? ******** //no vysledok asi hovori tolko, ze preto aby priemerna doba vykonavania jednej instrukcie bola 800 nanosekund potrebujeme zachovat pomer takeho poctu instrukcii ku pagefaultom ze n= … cize kazdych (0.0000583 krat pocet celkovych instrukcii) sa vyskytne jeden pagefault… este asi taka zaujimava uvaha: 1/0.0000583 = 17 153 so zaokruhlenim, co je pocet instrukcii ktore prejdu aby bol aspon jeden pagefault (n == pocet pagefaultov).. teda aby som mal aspon jeden pagefault potrebujem mat aspon 17153 instrukcii.. takze priemerne kazdych 17 153 instrukcii sa bude vyskytovat pagefault tak aby priem. cas vykonavania jednej instrukcie bol 800 nanosekund**
//ja nechapem co mam z toho vysledku vycitat ? a asi som debil ale mam 2 nezname ale len jednu rovnicu ?
// jaaaj no uz asi rozumiem dakujem ze si mi do toho vniesol svetlo :D
// diky moc za vysvetlenie pan panda // nz :D
//pomylil som sa v premienani jednotiek (mal som vysledok x1000) a dal mi 2/4 - for future generation
//ja som mal iba vzorec a mal som 2,5 so sometimes slightly fortunate
//podla mna tam to n-ko vobec nema byt a treba iba vypocitat k… a to vyjde k=**17142** +5

//niekto normalne vysvetlenie jak pre debilkov?

5. fs 1 mal subor 39 kb, velkost bloku bolo 4 kb a bol v sekvencnej pamati na mieste 572. subor bol vo fs na rovnakom fyzickom mieste a trebalo nakreslit ako by vyzerala ta tabulka.[4b]

Keďže súbor má 39 kB a veľkosť bloku 4 kB tak potrebujeme na tento súbor 39/4 čiže 10 blokov. Tie idú do FAT tabuľky štýlom 572
a v ňom 573. Ďalej 573 a v ňom 574. Bude ich 10 a posledný musí byť -1 čo indikuje koniec súboru. //nie je nahodou koniec suboru 0 a volny blok -1? // - 1 je ok
//čiže takto? FAT /file allocation table:
FAT - nema mat dva stlpce asi len jeden, bez toho adresa

adresa	odkaz
572	573
573	574

574	575
575	576
576	577
577	578
578	579
579	580
580	581
581	-1

(0)//nein

//mal som tu FAT takto ako je nakreslena a dal mi 4, resp. 6 s koeficientom
//ja som to mal tiez takto a dal mi 0 resp. 0 s koeficientom //:DDDDDDDDD
//a co ti povedal na konzultacii? aky bol dovod? ja som tam mal este nejaky obkec ze treba 9,75 =~ 10 blokov alokovat a pod..
// niekto, kto to mal dobre?
//ja by som radšej ten prvý stĺpec netabuľkoval lebo na konzultáciách sa pýtali, že koľko stĺpcov má FAT tabuľka a 2 stĺpce bola nesprávna odpoved a taktiež FAT tabuľka by mala byť reprezentovaná ako pole takže tak no …. ak sa mýlim tak ma prosím opravte //+1
// FAT moze mat 1-4 stlpce myslim. Povedal ze pre nas staci iba 1 a ze t nemame pisat takto lebo to ocislovanie policok nie je sucast tabulky to je iba pre nasu orientaciu
//skor tie indexy nedat do tabulkje loop iba vedla mimo tabulky

6. Výpočtové časy Round Robin 20 ms [6b]
procesy:
kazdy A proces bol nekonecny while cyklus ktoremu jedna iteracia trvala (vid nizsie ms na CPU) a raz v kazdej iteracii bola jedna I/O operacia.
A0 - cas na CPU 35ms a I/O zariadenie 10 ms
A1 - cas na CPU 25ms a I/O zariadenie 10 ms
kazdy B proces mal 100 iteracii a pri kazdej iteracii sa vykonala aj I/O operacia ktora trvala 10 ms.//každý B proces trval 1ms
B0, B1, B2, B3
Procesy prisli vsetky skoro naraz (nejak tak) v poradi A0,A1,B0,B1,B2,B3.
Otazky: Vyuzitie procesora?
Vyuzitie disku (I/O operacia) . // kludne ma po(o)pravte..
Ešte chceli aj napísať intervaly, kedy bol ktorý proces na CPU a kedy na HDD.

Urobte prehľad za prvých 120 ms.
čas (CPU)
bežiace
čakajúce
blokované (I/O)

//prosim opravte mi tabulku ak je to zle:

čas (CPU)	**bežiace**	**~~čakajúce~~ pripravené**	**blokované (v zátvorke dokedy je blokovaný)**
0-20	A0	A1,B0,B1,B2,B3	----------
20-40	A1	B0,B1,B2,B3,A0	----------
40-41	B0	B1,B2,B3,A0,A1	----------
41-42	B1	B2,B3,A0,A1	B0(51)
42-43	B2	B3,A0,A1	B0(51), B1(61)
43-44	B3	A0,A1	B0(51), B1(61), B2(71)
44-59	A0	A1, B0	B1(61), B2(71), B3(81)
59-64	A1	B0, B1	B2(71), B3(81), A0(91)
64-65	B0	B1	B2(71), B3(81), A0(91), A1(101)
65-66	B1	----------	B2(71), B3(81), A0(91), A1(101), B0(111)
66-71	----------	----------	B2(71), B3(81), A0(91), A1(101), B0(111), B1(121)
71-72	B2	----------	B3(81), A0(91), A1(101), B0(111), B1(121)
72-81	----------	----------	B3(81), A0(91), A1(101), B0(111), B1(121), B2(131)
81-82	B3	----------	A0(91), A1(101), B0(111), B1(121), B2(131)
82-91	----------	----------	A0(91), A1(101), B0(111), B1(121), B2(131), B3(141)
91-111	A0	A1	B0(111), B1(121), B2(131), B3(141)
111-120	A1	B0, A0	B1(121), B2(131), B3(141)

//upravená a doplnená tabulka o posledný stlpec, ja si myslim ze je to ok, ak nie tak ma opravte

\\ v 91-111 by v čakajúcich malo byť A1, ináč je to asi ok
\\a jak by sa tam prosim ta dostalo?

\\91+10=101 čo patrí do intervalu 91-111 // lenze ja som to robil naraz, naco budem ukazovat po 10 msekundach co urobim naraz(2 kroky), totiz tato tabulka tu v docu ma max. rozsah 20x20
\\ 44-59 Nemalo by B0 uz cakat ??? Mne to vychadza na 51. Na 41 ho bloknem a caka 10s 41+10 = 51. // to iste, setril som miesto spravil 2 kroky naraz...
otazocka: A0 a A1 nevykonavali 10 ms I/O operacie? alebo ako ste spocitali tie I/O?
//A0 zacalo pracovat s I/O az v 59... s I/O sa prvy krat zacne pracovat az po vykonani B0 cize od 41ms a pracuje sa az do konca cize 120-41 = 79/120

Nooo, lenze v zadani bol spomenuty aj FCFS . . . co s ním ? // tiez by ma zaujimalo...//z radu cakajucich vybaras na vykonavanie algoritmom FCFS //No jasne, keby som dal teoriu :D mal by som to spravne…
//FCFS keď je preemptívny tak robí to isté ako Robin Round … ak je nepreemprávny FCFS tak berieš tie úlohy po rade ale sú na CPU toľko koľko potrebujú a ptm začnú robiť I/0 operáciu .. ak ti tam dá že FCFS a nedá že je preemptívny / nepreemptívny tak rob nepreemptívny
Záver:

(color = unused)
Využitie CPU: ((120 - (9+5+9))/ 120) *100 = 80.8333% (green)
Využitie I/O-disku: ((120-(20+20+1))/120)*100 = 65.8333% (blue)

7. Fragment programu ,klasika [5b]
stránka 512 kb , n = 1024

Čiže polia boli na 2 stranky az.+1

Reference string: 0 19 0 19 (1.iterácia s prvými stránkami) (1.iterácia s druhými stránkami) 0 0
//mal som reference string "0 19 0 19 0 0 0 xy ..(nepamatam si)" a mal som to za 0, lebo pocas behu programu sa ten string bude vraj menit.. neviete niekto ako sa bude menit?
// myslim si, ze prvych 512 iteracii bude s prvymi cislami stranok (napr. A malo stranky 20 a 21, tak bude s 20) a dalsich 512 iteracii s druhymi strankami (pre A teda 21), kedze to bolo rozdelene
//Vedel by to sem niekto dat kto to mal na 100%
//To co je vyssie napisane je dobre (v teoretickej rovine), aj to co je cervenym. Len pre konkretne cisla by to chcelo cele zadanie prikladu

OPRAVÁK - OT **- mozem sa mylit, doplnte ma ako to ma byt, pripadne ak si lepsie pamatate veci dajte to sem, ale nezabudajte ze boli dve skupiny s roznymi cislami… [*made by frodo and followers*]**

TEÓRIA:

1. V stavovom diagrame procesov neexistuje slučka medzi stavom PRIPRAVENÝ a BLOKOVANÝ. Prečo sa nemôže proces dostať zo stavu PRIPRAVENÝ do stavu BLOKOVANÝ?
// Nemôže prejsť do stavu blokovaný, pretože ešte nepristupuje k žiadnym prostriedkom. Blokovaný môže byť len už bežiaci proces. (1,5b)##

2. Jedným zo spôsobov riešenia uviaznutia je metóda detekcie a zotavenia. Vysvetlite, ako funguje tá detekcia
//Deadlock detekujeme pomocou udrziavania grafu pridelenia prostriedkov v ktorom detekujeme cykly (monitoruje os) pripadne pomocou aproximacie - ak proces blokuje nejaky prostriedok prilis dlho zabijeme ho //-9 (motorku)
// z prednasky:
OS monitoruje prideľovanie a uvoľňovanie prostriedkov. Podla toho si udržuje graf pridelenia prostriedkov a detekuje v ňom cykly. Ak vznikne cyklus, ukončí sa niektorý z procesov v ňom. Ak uviaznutie pretrváva, ukončí sa ďalší proces, . . .
“Aproximácia” môže byt’ sledovanie doby blokovania procesov (bez grafu), ak niektorý proces je blokovaný dlhšie ako prahová doba, ukončí sa.
Iná možnost’ je vykonat’ postupne preempciu prostriedkov patriacich do cyklu.

3. Pri preklade logickej adresy stránky na fyzickú sa môže použiť číslo stránky. Vysvetlite:
a) ako sa ziskava cislo stranky pri preklade
//odrezanim offsetu ziskava ho mmu
b) ako sa pouziva cislo stranky
//na vyhladavanie cisla strankoveho ramu v asociativnej pamati alebo tabulke stranok

4. tlaciaren – ako funguje výlučný prístup k tlačiarni
//spooling
Metoda spooling procces demon vysvetlene v RT

5. FS1, FAT tabuľka:

a) koľko ma FAT tabuľka riadkov?

- tolko riadkov koľko diskových blokov

b) co sa v kazdom riadku FAT tabulky nachadza?

- nachádza sa tam číslo nasledujúceho bloku súbor

6. Vnutorne premenne semafora su vzhladom na semafor kriticke oblasti. Implementujte, resp. vysvetlite ako sa zabezpecuje vylucny pristup k tymto premennym pomocou monitora. (ako na RT)

operácie wait() a signal() su metódy monitoru, čím je zabezpečené vzájomné vylučovanie; hodnota semaforu by bola vnutorna premenna monitora

7. Vlákna sa delia do 2 kategórií - jadrové a používateľské.

a) aká je výhoda jadrového oproti tomu v používateľskom rozhraní?

Da sa zabezpecit skutocny hw paralelizmus

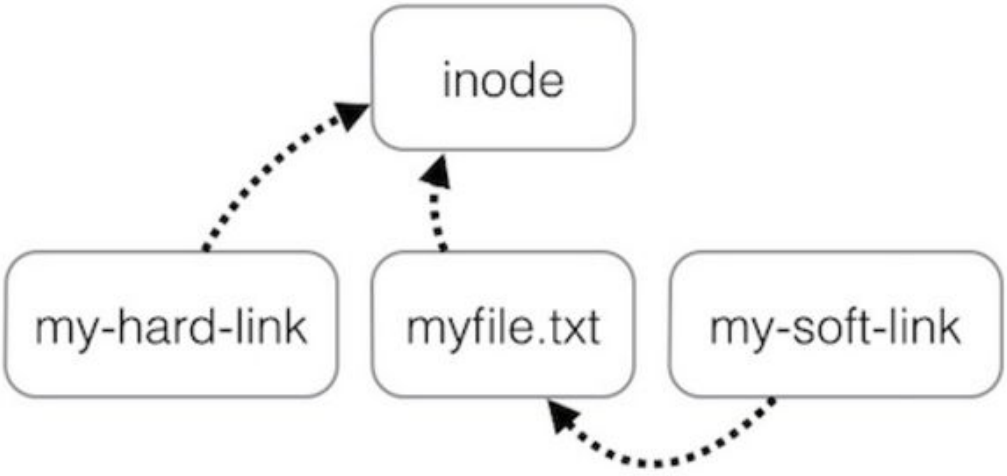
b) a naopak, aká je výhoda toho používateľského oproti jadrovému?

Nizsia rezia pri prepinani (nemusi sa prepinat medzi kernel/user mode), planovanie sa riesi VD swechat pouzivatelskom kode,programator si moze zvolit vhodny algoritmus

-nepotrebuje podporu OS, je možné ich použiť aj keď OS vlákna nepodporuje (1b)

8. Omáčka pred zadaním:

Pomocou unixového príkazu ls je možné zistiť mnoho užitočných vecí, vrátane čísla i-uzla daného súboru. Slovne popíšte, ako by ste spravili skript, ktorý vypíše všetko pevné linky k danému súboru, ktorý dostane zadaný ako parameter (absolútnu adresu k súboru). Nepíšte kód, len slovný popis (!)



pomocou prepinaca cez prikaz ls -i si najdem vsetky i-node, ak sa mi niektoré i-node zhodujú s i-nodom zadaneho suboru, potom som nasla hard linku prisluchajucu k zadanemu suboru/ceste

9. P bit (dačo na štýl M bit-u v RT):

a) ako a kde sa používa?

Present bit P - prítomnosť stránky v hlavnej pamäti
pri pokuse o pristup na stranku, ktora sa nenachadza v hlavnej pamati (P bit ma xdx sa mení jeho hodnota?
(1= prítomná v hlavnej pamäti, 0= neprítomná v hlavnej pamäti)

10. mal si dane prostriedky a mal si popisat ktory je OP,JP,VP …
- strankovy ram VP
- fat tabulka - (tusim ze zo bola inoda vo FAT tabulke - JP**) //OP**
- tlaciaren -OP

1. **- usek pamate v davkovom OS -** OP

- usek pamate v OS s multiprogramovanim a so swapovanim -VP
A dalej neviem…dalsie 2 …
//nieco na sposob i-uzol v strukture i-uzlov? :D taky nejaky bordel - bolo to OP

PRIKLADY:

1. K-ta iteracia procesov v cycle while, maju sa zosynchronizovat procesy P1 a P2 vdaka znova funkcii barrier ako na RT, avsak tentokrat nie pomocou monitoru ale semaforov

```
semaphore mutex=1;
semaphore block1=0;
semaphore block2=0;
void barrier(){
	mutex.wait();
	if(++count == N){
		for(int i=0;i<N;i++) block1.signal();
	}
	mutex.signal();
	block1.wait();
	…
	mutex.wait();
	if(--count==0){
		for(int i=0;i<N;i++) block2.signal();
	}
	mutex.signal();
	block2.wait();
}//ČO TAKTO? toto je pre N procesov, reusable barrier.
```

http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf //+1 str. 41

2. Bankovy system, ako na RT .. (toto bolo tusim jedine even eat we eat werewolf eed#)

3. Navrhni funkcie lock a unlock z ulohy 2 za pouzitia monitora

```
lock(int i){
  if(i.locked==true){
        c.wait();
   }
  i.locked=true;
}

unlock(int i){
  if(i.locked==true){
        i.locked=false;
  }
 c.signal();
}
//toto je dobre????
```

v prednaske 4/22 je nieco podobne

4. Podobny priklad ako na RT … vyriesit priklad kde pravdepodobnost modifikovania stranky je 25 % … modifikovane ak su tak trvaju 20 ms a bez modifikacie 8ms… vykonanie obycajnej instrukcie trva 100 nanosekund… aka je percentualna velkost vypadkov stranok tak aby priemerna doba vykonania instrukcie trvala 500 nanosekund?
(dodatok by Tony -> nebolo to nejako tak, ze ak nastane vypadok stranky, tak pri vypadku trva bez modifikacie 8 ms a s modifikaciou 20 >ms?)
//0.003636%

postup: (ak je to zle, opravte ma)

$$0.75*8 + 0.25*20 = 11ms$$

obyc. instr. 100 ns = 0.0001 ms

priemer doba instr. 500 ns = 0.0005 ms

$$(100k\ (ns) + 11n\ (ms)) / k = 500\ (ns)$$

prepis jednotiek na ms vsetko

$$(0.0001k + 11n) / k = 0.0005$$
$$0.0001 + (11n / k) = 0.0005$$
$$11n / k = 0.0004$$
$$11n = 0.0004k$$
$$n = (0.0004/11)k$$
$$n = (0.0004/11)k$$
$$n = 0.0000363636k$$

5. Bol dany system FS1 ktory ma fat tabulku – ta nebola dana… no a potom bola dana velkost suboru tusim 27kB a velkost bloku v systeme FS1 aj v FS2 bola 4kB...v FS1 dany subor zacinal v 526. bloku v poradi … FS2 je druhy system ktory je implementovany i-nodmi…trebalo prepisat FS1 na FS2 .. trebalo podrobne ku kazdej tabulke i-nodu napisat kde je co, a taktiez aj do buniek tabulky, kde aky blok a co… dalej bolo dane ze FS2 ma 3 priame bloky 1 nepriamy, jeden dvojito nepriamy a je den trojito nepriamy

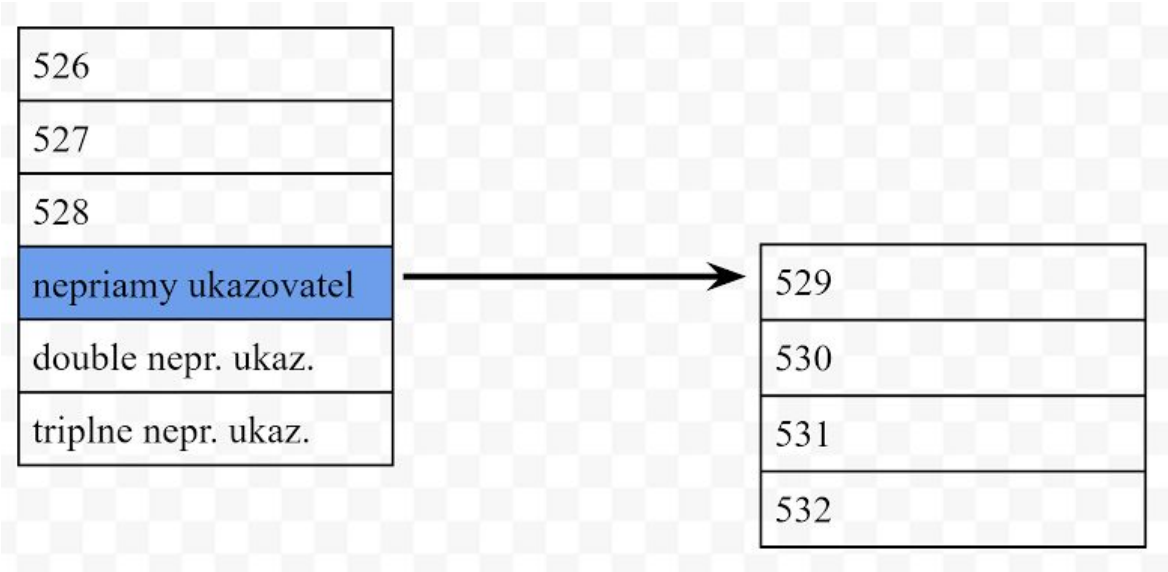
subor = 27 648 B = zabera 7 blokov (27 648 / 4096 B == 6.75)

FS1 - FAT table (ma len jeden stlpec - odkaz, prvy stlpec je pomocny - index)

index	odkaz
526	527
527	528
528	529
529	530
530	531
531	532
532	-1df

//how to convert FAT table into inode? to je otazka :D //tak ze prve 3 polozky tabulky budu v 3 priamych blokoch i-nodu a ostatne v nepriamom

i-node (uz spravne)



//a 526 si kde nechal moj ? //vilko vas zjebe za takyto nacrt deti moje musi tam byt aj ta 526 v inode
//spravne ma byt priamy 526-528 a nepriamy 529-532
//obraztok upraveny deti moje
//pri i-node ziadna -1 nie je. to nefunguje ako pri FAT s odkazmi. i-nody maju ulozene adresy blokov 526 -532 (zjednodusene, pouzivaju sa cisla z indexov)
//upraveny obrazok na zaklade komentara vyssie, **este vie mi niekto odpovedat jak je to s tym poctom prvkov v tej nepriamej tabulke na 100%????**

//to mas bud v zadani (skor nie) alebo si to vypocitas z udajov zo zadania.
Priklad: jeden ukazovatel ukazuje na blok (adresuje?) o velkosti 512B velkost ukazovatela je 32b = 4B t.j. nepriamy ukazuje na tabulku, ktora ma 128poloziek (ukazovatelov) a 2nepriamy na tabulku ktora ma 128ukaz. a kazdy jeden na tabulku, ktora ma tiez 128ukaz
Ked sa to zobere na tento priklad (chyba tam velkost ukazovatela) tak nepriama tabulka ma pre 32b ukazovatel 4096/4 = 1024 poloziek (ukazovatelov) pri 64b ukazovateli 4096/8 = 512
//super, vdaka

//otazka je kolko bude mat resp. moze mat ta nepriama tabulka?
//4096B/4B = 1024 prvkov v nepriamej tabulke????? (4 B je velkost ukazovatela)
//toto mate napicu chalani typek napise ze zabera 7 blokov ale nakresli 8. ma ich byt len 7 nakreslenych (uz to je fixnute ale na tej fotke to je zle)//fotka odstranena, prepisane do tabuliek sem

6. Bola dana funkcia Ti = ((i + 4) mod 9) +4 a i=0…8 … boli teda dane procesy 0..8 pre ktore dlzka casoveho kvanta pre kazdy sa mala vyratat pomocou funkcie Ti …bolo dane ze procesy nemaju pridelene I/O prostriedky za ich behu… kazdy process bezi len raz, ked ubehne jeho casove kvantum, skonci sa ten process… potom nasledne sme teda mali napisat tabulku time, running, ready … kde si mal uplatnit algoritmus ktory je modifikaciu algoritmu SJN, no proste to bol alg. SRTF Shortest Remaining Time First ktory je ako bolo dane – preemtivny.
(presné znenie otázok)

a) **napíšte jednotlivé intervaly trvania daných procesov**
b) **aká bola priemerná (casova?) odozva procesov? //toto niekto??**
   //prednaska 10, strana 33 :D tam to je

proces	**casove kvantum Ti\\a**
0 (A)	8
1 (B)	9
2 (C)	10
3 (D)	11

4 (E)	12
5 (F)	4
6 (G)	5
7 (H)	6
8 (I)	7

//neviem ci to ma byt takto… ak ano tak sa mi to zda prilis jednoduche, kedze dopredu vieme vsetky casy (alebo procesy prichadzaju postupne podla i?? neviem povedzte :DD)

čas	**running**	**ready**
0-4	F (5)	G,H,I,A,B,C,D,E
4-9	G (6)	H,I,A,B,C,D,E
9-15	H (7)	I,A,B,C,D,E
15-22	I (8)	A,B,C,D,E
22-30	A (0)	B,C,D,E
30-39	B (1)	C,D,E
39-49	C (2)	D,E
49-60	D (3)	E
60-72	E (4)	----------

to b) sa vypočíta že 9*4 + 8*5+ 7*6 + 6*7 + 5*8 + 4*9 + 3*10 + 2*11 + 1*12 a to celé vydelíš s 9 a máš priemernú dobu odozvy (všímaj že keď najprv plánuješ kratšie procesy tak je kratšia lebo tie prvé čísla pri násobení 9,8,7...1 tam ostávajú vždy od najväčšieho po najmenšie a začína sa od čísla = počtu procesov)

7. Typicky instrukcie… A[i] = B[i]+C[i] … bolo 5 strankovych ramov… dlzka slova 512… a chceli vediet reference string, kde n =1024 a taktiez chceli poziciu –fyzicku adresu prvkov: A[560] B[880] C[72] … cisla su cca,

nie presne… pri pocitani pozor na offset, ktory bude mat viac ako 9 bitov a bude posuvat o jednu hodnotu viac v cisle strankovych ramov //co sa BUDE Kam posuvat ?? ...algoritmus na vylucenie stranky bol FIFO.. (po 512 iteracii sa naplnili strankove ramy a trebalo asi vylucovat stranky)k

RT 2016/2017

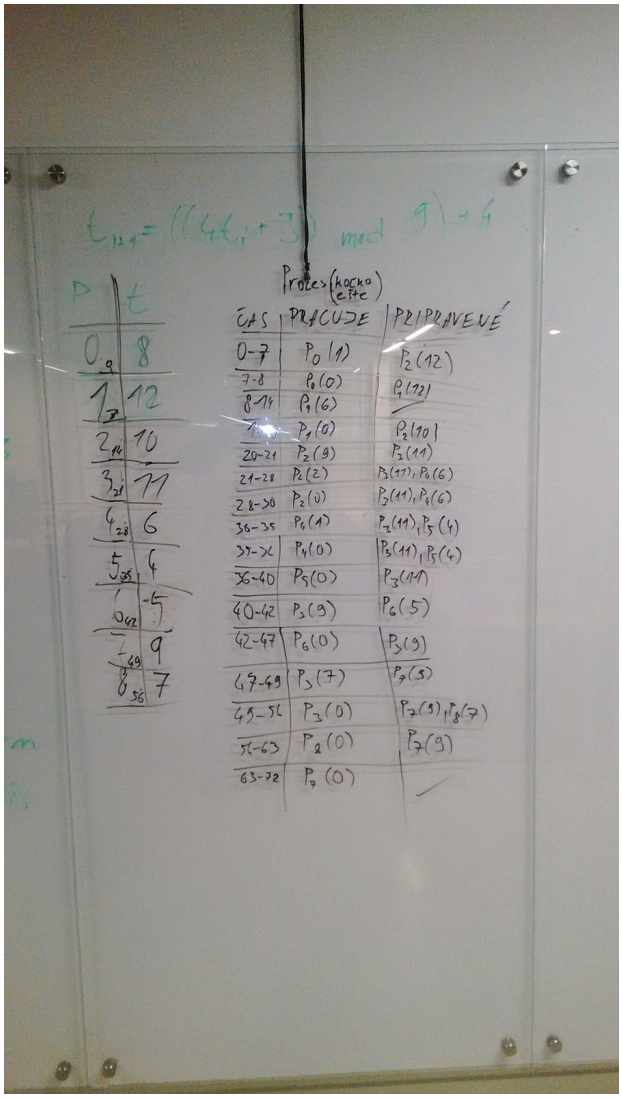
Ludia piste hlavne co si pamatate z konzultacii, na kolko mne priamo povedal ze vie ze je nejaky doc a ze to tam je zle -_-

Prakticke priklady

Mali sme zadanu funkciu: *t(i+1) = ((4*t(i)+3) mod 9) + 4.* Procesy boli vytvárané každých 7 milisekund a bolo ich 9(P0, P1, …. P8). Funkcia vyšie určovala ako dlho strávi proces na procesore. Zadaný bol proces P0 ktorý trval 8 milisekúnd.

a) **Aký bol časový priebeh procesov**
b) **aká bola priemerná casova odozva procesov**

a)



b) 73/9 = 8,1111111//asi, toto neviem //toto je zle

malo by to byt Doba kedy proces skoncil - doba vytvorenia procesu => napr pre P3 je Doba kedy proces skoncil = 56 a doba vytvorenia procesu = 21 => 56-21=35 toto sa spravi pre vsetky procesy a nakonci sa vydeli ta celkova suma, poctom procesov

(8+13+16+35+8+5+5+23+7)/9 = 13,33 (5bodov)