

# **Zbierka úloh k predmetu Operačné systémy**

**Zostavil: Juraj Štefanovič a kolektív  
verzia 2, november 2008**

# Zbierka úloh k operačným systémom

## Verzia 2, november 2008

Fakulta informatiky a informačných technológií, STU Bratislava  
Zostavil: Juraj Štefanovič a kolektív

**Ďakujeme všetkým, ktorí si príklady prečítali (alebo ešte prečítajú)  
a upozornili na chyby a nedostatky.**

Poznámka - riešenia sú naznačené do tej miery, aby bolo možné pochopiť ako sa realizujú a viesť k tomu kľásť otázky. Predpokladáme, že čokoľvek je tu nejasné alebo chybné, to sa nás budete pýtať v rámci diskusného fóra a predovšetkým seminárov pri tabuli, ktoré sú určené práve na toto. Najlepšie, keď to bude ešte počas semestra... Hodnotné otázky a poznámky na fóre sú registrované a prispievajú k Vášmu hodnoteniu.

## Synchronizácia

Poznámka k synchronizačným úlohám: ak riešenie vedie na zablokovanie alebo vyhľadovanie procesu, tak je to riešenie, ktoré možno považovať za jednoduché a základné. Odporúčame zamyslieť sa nad jeho nevýhodami a možnosťami zdokonalenia. Samozrejme jednoznačne zistiť a dokázať, či v danom riešení existuje zablokovanie, to nie je vždy triviálna úloha a záleží od prípadu k prípadu. Úlohy, pri ktorých sa navrhuje/programuje synchronizačné riešenie, ponúkajú obvykle rôzne možnosti ako riešenie urobiť.

### 1) Unisex toaleta (alebo sprcha, sauna, ...)

#### Zadanie:

Navrhnete synchronizačné riešenie pre unisex toaletu (určenú pre mužov aj ženy) s nasledovnými obmedzeniami:

- na toalete sa nesmú nachádzať muži a ženy zároveň
- na toalete sa nesmú nachádzať viac ako traja ľudia súčasne

Riešenie by sa malo vyhnúť zablokovaniu. Starvacia – vyhľadovanie procesov sa neberie do úvahy. Môžeme predpokladať, že toaleta je vybavená všetkými potrebnými semaforami.

#### Riešenie:

Príchod niekoho k toalete je reprezentovaný bežiacim procesom, ktorý je typu žena alebo muž a požaduje vstup do kritickej oblasti. V kritickej oblasti môžu byť najviac tri procesy – takže použijeme semafor. Môžu tam byť len procesy jedného druhu, použijeme na to napríklad riadiacu premennú, ktorú stráži semafor. Problém jednoduchého riešenia je, že

zmenu pohlavia možno uskutočniť až keď je toaleta prázdna, čo pri rušnej prevádzke nemusí nastať. Občas treba všetkých mužov vyhnať a prepnúť panáčika na ženy (a naopak). Nasledujúci príklad je v princípe o tom istom.

## 2) Jednosmerný most

### Zadanie:

Navrhňte synchronizačné riešenie pre most s jedným jazdným pruhom, ktorým môžu prechádzať autá len v jednom smere, pričom tento most neznesie väčšiu záťaž ako 3 vozidlá súčasne.

### Riešenie:

Predpokladajme, že v systéme je každé auto reprezentované jedným procesom, ktorý vykonáva procedúru `OneVehicle()`, keď dorazí k mostu:

```
OneVehicle(int direc)
{
    ArriveBridge(direc); // vstup do kritickej oblasti
    CrossBridge(direc);  // tu je jedno čo sa deje, ideme, ideme...
    ExitBridge(direc);   // výstup z kritickej oblasti
}
```

Premenná `direc` označuje smer, ktorým vozidlo prechádza mostom pričom nadobúda hodnoty 0 a 1. Vstup a výstup okolo kritickej oblasti bude obsahovať semafor a inak je to vlastne unisex toaleta ako v predošlom príklade, namiesto toalety je most a namiesto pohlavia smer jazdy. Riešením úlohy je rozumne napísať obsah procedúr `ArriveBridge()` a `ExitBridge()`.

## 3) Autobusová zastávka

### Zadanie:

Napíšte synchronizačný kód, ktorý simuluje nasledovnú situáciu so všetkými uvedenými obmedzeniami:

Na autobusovú zastávku postupne prichádzajú ľudia, ktorí čakajú na autobus. Keď autobus dorazí, všetci čakajúci vyvolajú procedúru `BoardVehicle()`, pričom pasažier nemôže nastúpiť do autobusu na zastávke (musí počkať na ďalší autobus), ak dorazil na zastávku až po príchode autobusu. Kapacita autobusu je 50 ľudí, ak sa na zastávke v čase príchodu autobusu nachádza viac ľudí ako je tento počet, niektorí z nich musia počkať na ďalší autobus.

Po nastúpení všetkých pasažierov autobus vyvolá procedúru `Departure()`. Pokiaľ na zastávke nikto nečaká, vyvolá túto procedúru okamžite.

### Riešenie:

Príchod niekoho na zastávku aby čakal na autobus je reprezentovaný čakaním na nejakej podmienke či je prítomný autobus. **BoardVehicle()** je procedúra vstupu do kritickej oblasti, pričom semaforom je možné vpustiť najviac 50 procesov. Procedúra **Departure()** znamená zrušenie možnosti žiadať o vstup do kritickej oblasti.

#### 4) Jedáleň

##### Zadanie:

Napíšte synchronizačný kód, ktorý simuluje nasledovnú situáciu so všetkými uvedenými obmedzeniami:

Každý študent po príchode do jedálne vyvolá procedúru **Jedlo()** a následne **Odchod()**. Medzi týmito dvomi procedúrami sa nachádza v stave “pripravený na odchod”. Synchronizačné obmedzenie pre túto situáciu je, že študent nikdy nesmie sedieť pri stole sám, pričom študent sedí pri stole sám vtedy, keď všetci ostatní, ktorí vyvolali **Jedlo()**, vyvolajú **Odchod()** pred tým, ako bola ukončená procedúra **Jedlo()** pre tohto študenta.

##### Riešenie:

Tie dve procedúry predstavujú vstup a výstup okolo kritickej oblasti. Výstup môže byť pozdržaný podmienkou, ak povedzme počet procesov v kritickej oblasti po vydelení štyrmi poskytuje zvyšok 1 (jeden je sám pri stole so štyrmi stoličkami).

#### 5) Výťah – riadiaci systém

##### Zadanie:

S použitím semaforov a podmienených premenných napíšte synchronizačný kód pre riadiaci systém výťahu. Výťah je reprezentovaný procesom, každá osoba, ktorá ho použije je taktiež reprezentovaná osobitným procesom. Je potrebné implementovať procedúru volanú prichádzajúcim pasažierom:

**ArrivingGoingFromTo(int atFloor,int toFloor)**

Táto procedúra má zobudiť výťah, povedať mu, na ktorom poschodí sa daná osoba nachádza a počkať, kým sa výťah na toto poschodie dostaví, až potom mu povedať, na ktoré poschodie má ísť. Výťah sa pohybuje z jedného poschodia na druhé úžasne rýchlo ale nie okamžite – trvá mu to presne 100 tikov. Pre zjednodušenie môžeme predpokladať, že existuje len jeden výťah, ktorým môže cestovať ľubovoľný počet pasažierov súčasne. Tiež je potrebné dávať prioritu odchodom/príchodom na piate poschodie.

##### Riešenie:

Procedúra pozastaví proces osoba, kým sa proces výťah nenastaví na číslo jej poschodia. Obsahuje teda podmienku s premennou. Potom sa proces osoba pozastaví ďalšou podmienkou, kým nie je výťah na cieľovom poschodí. Proces výťah cyklicky mení

hodnotu premennej POSCHODIE a vlastne nič iné ani robiť nemusí v jednoduchom riešení, len s prestávkami počítať túto premennú od nula po max a potom zase dole, atď.

## 6) Hľadaj-Vlož-Odstráň

### Zadanie:

Tri druhy vlákien zdieľajú prístup k jednoduchému zreťazenému zoznamu: hľadajúce, vkladajúce a odstraňujúce vlákna. Prehľadávajúce vlákna zoznam iba prezerajú, môžu sa teda vykonávať súbežne. Vkladajúce vlákna pridávajú nové položky na koniec zoznamu, operácie vloženia sú vzájomne sa vylučujúce (mutually exclusive), aby sa zamedzilo vkladaniu nových položiek v rovnakom čase. Operácia vloženia však môže prebiehať paralelne s ľubovoľným počtom operácií prehľadávania. Odstraňujúce vlákna odstraňujú položky z ktoréhokoľvek miesta v zozname. V jednom čase môže k zoznamu pristupovať maximálne jedno odstraňujúce vlákno, operácia odstránenia sa vzájomne vylučuje s operáciami prehľadávania a vkladania.

Napíšte kód pre vyššie uvedené vlákna, ktorý splňa všetky požiadavky zadania.

### Riešenie:

Zreťazený zoznam je údajová štruktúra, na ktorú každé vlákno pozná smerník a môže zavolať niektorú z operácií **hľadaj** (\*čo,\*zoznam), **vlož** (\*čo,\*zoznam), **odstráň** (\*čo,\*zoznam). Vlákna počas behu generujú požiadavky na vkladanie, hľadanie a zmazávanie položiek, napríklad náhodne generovaných čísiel od 1 do 9 a podobne. Ďalej je to podobné na úlohu zapisovateľa a čitateľa, keď prístup k zoznamu je strážený semaforom. Rozdiel oproti zapisovateľom a čitateľom je ten, že sa tam ešte vyskytujú likvidátori záznamov a zápis+čítanie sa považujú za vzájomne znesiteľné operácie (môžu prebiehať paralelne).

## 7) Misionári a kanibali I.

### Zadanie:

Cez rieku plnú krokodílov premáva loď s kapacitou N osôb. Na jeden breh prichádzajú v náhodnom poradí misionári a kanibali, aby sa odviezli na druhú stranu. Z dôvodu osobnej bezpečnosti nesmie v loďke počet kanibalov prevýšiť počet misionárov. Implementujte simuláciu takejto situácie prostredníctvom mnohých paralelne spúšťaných procesov, ktoré budú predstavovať jednotlivé prichádzajúce osoby a prevoz osoby bude reprezentovaný dokončením procesu.

### Riešenie:

(jedna z možností)

Napíšeme program, ktorý bude pomocou volania fork() cyklicky generovať stále nové a nové paralelne bežiacie procesy Proces\_Osoba(), pričom každému procesu je pridelená informácia, či je typu misionár, alebo kanibal. Aby sa proces mohol dokončiť (pod čím rozumieme že sa osoba previezla loďkou na druhú stranu), musí mať možnosť inkrementovať príslušné počítadlo kanibalov alebo misionárov, nesmie však narušiť ich

povolený pomer a loďka nesmie byť preplnená. Aby si procesy vzájomne striedavo neprepisovali organizačné údaje v kritickej oblasti, tak pomocou jedného semaforu je vždy len jeden proces v jednom okamihu pripustený k premenným misionári a kanibali. Pritom neustále beží paralelne proces Proces\_Prievozník(), ktorý kontroluje stav loďky a ak je plná, tak vynuluje počty cestujúcich (vykoná prevoz), takže môžu nastúpiť ďalší. Semafor je teda len jeden a je inicializovaný na hodnotu 1, pričom tu existuje len v stavoch 1,0 ako zámka.

```
Proces_Osoba(int typ) {          // tieto procesy stále pribúdajú
    while() {                    // čakám v cykle, pokiaľ sa nedostanem k lodi
        down();                  // pripustenie k premenným
        if(kanibali + misionári == N) // loď je plná
            { up(); continue; }   // musíme čakať prievozníka
        if(typ == kanibal) {
            if(kanibali < misionári)
                { printf("nastupuje kanibal"); kanibali++; }
            else { up(); continue; } // veľa nás, musíme čakať
        }
        else
            { printf("nastupuje misionár"); misionári++; }
        up();                     // uvoľníme semafor
        break;
    }                             // koniec procesu, už nastúpil na loď
}

Proces_Prievozník() {           // tento proces beží neustále
    while(1) {
        down(); // čakanie na semafor
        if(kanibali + misionári == N) { // loď ide až keď je plná
            kanibali = misionári = 0;
            printf("vrátila sa prázdna loď, nastupujte");
        }
        up(); // uvoľnenie semaforu
    }
}
```

## 8) Santa Claus

### Zadanie:

Santa Claus spí vo svojom obchode na Severnom póle a môže sa zobudiť, len ak (1) všetkých 9 sobov sa vrátilo z dovolenky v južnom Pacifiku alebo (2) niektorí škriatkovia majú problémy pri výrobe hračiek, aby sa však Santa dobre vyspal, škriatkovia ho môžu budiť len ak majú súčasne traja problémy. Pokiaľ sa riešia problémy troch škriatkov, ostatní škriatkovia s problémom musia čakať. Ak sa Santa prebudí kvôli trom škriatkom, ktorí majú problém a všetky soby sa už vrátili z dovolenky, Santa ich ignoruje, pretože je pre neho dôležitejšie pripraviť si sane na Tour de USA (predpokladá sa, že sobom sa príliš nechce vracieť z trópov a preto tam zostávajú do najposlednejšej chvíle). Posledný sob musí zohnať Santu, zatiaľ čo ostatné soby čakajú v maštali pre tým, ako budú zapriahnuté do saní. Navrhnite synchronizačné riešenie, ktoré simuluje túto situáciu, pričom musí spĺňať nasledovné špecifikácie:

- po návrate posledného soba vyvolá Santa procedúru **PrepareSleigh** a následne všetkých deväť sobov musí vyvolať procedúru **GetHitched**.
- po príchode tretieho škriatka musí Santa vyvolať **HelpElves**. Súbežne by si mal každý škriatok vyvolať **GetHelp**.
- všetci traja škriatkovia musia vyvolať **GetHelp** pred tým ako vstúpi ďalší škriatok (inkrementuje počítadlo škriatkov)

Santa by mal bežať v slučke takže môže pomôcť mnohým skupinám škriatkov. Môžeme predpokladať, že sobov je presne 9, ale škriatkov môže byť ľubovoľný počet.

#### Riešenie:

Je to vzájomne poprepájané riešenie viacerých čiastočných úloh. Zobudenie Santu pri probléme minimálne troch škriatkov je ekvivalentné zobudeniu prievozníka, keď má práve troch cestujúcich na prepravu loďkou. Pri menšom počte prievozník nerobí nič, pri počte tri všetkých “prevezie”, t.j. Santa obslúži a do čakárne môžu vstupovať ďalší. Zbieranie sa sobov je podobné, pričom obsluha sa uskutoční až pri počte 9 čakajúcich procesov. Keď deviaty sobí proces vstúpi do čakania v kritickej oblasti, zistí že počítadlo je 9 a zavolá Santu – všetky soby opustia kritickú oblasť a ako procesy skončia. Vybavenie sobov má vyššiu prioritu ako vybavenie škriatkov, teda pri vybavení sobov sa nastaví podmienková premenná pri ktorej nemôže pokračovať vybavovanie škriatkov, musí počkať.

## 9) Suši bar

#### Zadanie:

Predstavte si suši bar s piatimi stoličkami. Ak dorazíte k baru, pričom je aspoň jedna zo stoličiek neobsadená, okamžite ju obsadíte. Ale ak je všetkých 5 stoličiek obsadených, znamená to, že večerajú spolu a budete musieť počkať, kým všetci odídu predtým, ako sa usadíte. Napíšte kód, ktorý simuluje túto situáciu.

#### Riešenie:

Kritická barová oblasť je strážená semaforom, ktorý je inicializovaný na hodnotu počtu stoličiek a teda prepustí dovnútra najviac päť procesov. Netriviálne je, že ďalšie procesy môžu byť pripustené až po úplnom vyprázdnení baru. Pridáme povedzme premennú, ktorá sa inkrementuje až po hodnotu 5 a pri tejto hodnote nemôže žiadny proces prísť k vstupnému semaforu, aj keď by ho už semafor mohol pustiť na práve uvoľnenú stoličku. Pomôcť by mohli teda dve premenné: spomenuté počítadlo počtu procesov vo vnútri a logická premenná, ktorá sa nastaví na 1 keď dosiahneme 5 a nastaví na nulu keď počítadlo zase dosiahne nulu. Logická premenná bude brániť prístupu k semaforu.

## 10) Obedujúci filozofi

### Zadanie:

(The Dining Philosophers Problem – autor E. W. Dijkstra, 1965) Opis problému: okolo okrúhleho stola sedí  $N$  filozofov (alebo rytierov?), každý má pred sebou tanier s jedlom (špagety) a na každej strane taniera má jednu vidličku, ktorú zdieľa so susednou osobou. Počet vidličiek je teda tiež  $N$ . Aby osoba mohla jesť, potrebuje mať v rukách obidve vidličky a vtedy jej susedia musia čakať.

### Riešenie podľa učebnice - A.S. Tannenbaum: Modern Operating Systems -

Ako formalizovať tento problém: činnosť každej osoby možno implementovať ako proces, ktorý pravidelne volá procedúru `stolovanie()` s parametrom identifikujúcim miesto osoby pri stole. Poradie pri stole je číslované v kruhu od 0 do  $N - 1$ . Jedno z možných riešení tohto problému pomocou semaforov je uvedené v literatúre, pričom sa pripomína, že tiež nezaručuje korektnosť za všetkých okolností. Riešenie používa pole semaforov a pole premenných, pričom každá položka z týchto polí patrí jednému filozofovi. Okrem toho je použitý jeden semafor `mutex` na výlučný prístup len jedného zároveň k manipulácii s riadiacim polom `stav`. Ak testovacia funkcia overí, že  $i$ -ty filozof môže začať jesť, tak stav  $i$ -teho semaforu je nastavený na 1 a procedúra `zober_vidličky()` sa dokončí bez uspania, takže  $i$ -tý proces vstúpi do kritickej oblasti: filozof má obidve vidličky a začne jesť. Semafor `mutex` zabezpečuje prístup vždy len jedného procesu k polu stavových premenných `stav` a k testovaniu situácie. Pri štúdiu tohoto príkladu je potrebné si všimnúť umiestnenie operácií so semaforom `sem[i]`.

```
void stolovanie (int osoba) {
    while(1) {
        vezmi_vidličku(osoba);           // čakanie na pravú
        vezmi_vidličku((osoba+1)%N);     // čakanie na ľavú
        daj_do_úst();                     // kritickej oblasti
        poloz_vidličku(osoba);           // uvoľnenie vidličiek
        poloz_vidličku((osoba+1)%N);
    }
}

semafor sem[N]; // pole semaforov inicializovaných na 0
int stav[N];    // pole premenných inicializovaných na NEJEM
semafor mutex;  // binárny semafor inicializovaný na 1

void zober_vidličky(int i) // i-ta osoba vezme obidve
{
    down(mutex);
    // nasledujúcu sekvenciu dvoch príkazov
    // nemôžu vykonať súčasne dve alebo viac osôb
    stav[i] = ŽIADAM; // chcem vidličky
    testuj(i);        // budem jesť?
    up(mutex);
    down(sem[i]);
    // ak medzitým nebolo vykonané up() v procedúre
    // testuj(), tak zaspím
}
```



```

void polož_vidličky(int i) // vráti obe vidličky
{
    down(mutex);
    // nasledujúcu sekvenciu troch príkazov
    // nemôžu vykonať súčasne dve alebo viac osôb
    stav[i] = NEJEM;
    testuj(lavý(i)); // ponúkni suseda vľavo
    testuj(pravý(i)); // ponúkni suseda vpravo
    up(mutex);
}

void testuj(int i) // i-ta osoba zistí, či bude jesť
{
    if(stav[i] == ŽIADAM    && // ja chcem jesť
        stav[lavý(i)] != JEM && // ľavý kolega neje
        stav[pravý(i)] != JEM) // pravý kolega neje -
    { stav[i] = JEM; up(sem[i]); } } // - teda ja budem

```

### Riešenie podľa učebnice – William Stallings: Operating Systems -

Toto riešenie ukázané pre päť filozofov je postavené na zjednodušujúcom pravidle, že najviac štyria sú pripustení ku stolu a piaty musí počkať. Tým sa odstraňuje zablokovanie a aj vyhladovanie.

```

Semafor vidlicka[5]; // pole 5 semaforov, každý inicializovaný na 1
Semafor miestnost; // semafor inicializovaný na 4

```

```

void filozof(int osoba)
{
    while(1) // dlhy zivot filozofa
    {
        myslim();
        down(miestnost); // vpustenie najviac styroch dovnutra
        down(vidlicka[osoba]); // beriem vidlicky
        down(vidlicka[(osoba+1)%5]);
        jem();
        up(vidlicka[osoba]); // odkladam vidlicky
        up(vidlicka[(osoba+1)%5]);
        up(miestnost);
    }
}

```

## 11) Problém spiaceho holiča

### Zadanie:

V holičstve je  $N$  holičov a v čakárni je  $M$  stoličiek. Zákazníci prichádzajú, ak nájdú voľnú stoličku tak zostanú čakať, ak nenájdu tak odídu. Ktorý holič je voľný, vezme ďalšieho zákazníka. Ak žiadny zákazník nečaká, holič môže spať.

### Riešenie podľa klasickej učebnice „Tannenbaum“:

Formalizácia problému: v systéme môže paralelne bežať ľubovoľné množstvo procesov vykonávajúcich činnosť holiča a procesov vykonávajúcich činnosť zákazníka. Pomocou dvoch semaforov sa zákazníci a holiči vzájomne púšťajú na rad, resp. do práce. Pomocou semafora **mutex** je zabezpečené, aby naraz mohol meniť hodnoty semaforov len jeden proces, holič alebo zákazník. Činnosť každého procesu typu holič znižuje počet zákazníkov v čakárni, činnosť procesu typu zákazník pridáva osoby do čakárne.

```
semafor mutex;    // binárny semafor, inicializovaný na 1
semafor holiči;   // semafor s hodnotou počtu voľných holičov
semafor čakajúci; // semafor s hodnotou počtu čakajúcich
int rad;          // zaznamenávame počet čakajúcich

void činnosť_holiča(void) {
// takýchto procesov môže súčasne bežať viac
while(1) {
    down(čakajúci); // ak nie je klient tak zaspím
    down(mutex);   // vstup do kritickej oblasti
    rad--;          // vezmem klienta z radu a -
    up(holiči);     // - pustím ho cez semafor(oholím)
    up(mutex);     // výstup z kritickej oblasti
}
}

void činnosť_zákazníka(void)
// takýchto procesov môže súčasne bežať viac
{
while(1)
{
    down(mutex);   // vstup do kritickej oblasti
    if(rad < MAX)  // ak ešte je voľná stolička
    {
        rad++;     // tak vjdem do čakárne
        up(čakajúci); // voľní holiči sa môžu zobudiť
        up(mutex); // výstup z kritickej oblasti
        down(holiči); // požiadam o holiča, čakám
        strihám_sa(); // deje sa moja obsluha
    }
    else up(mutex);
    // ak je preplnená čakáreň, tak nič
}
}
```

## 12) Vytváranie vody

### Zadanie:

Matka Príroda má synchronizačné problémy s chemickou reakciou, pri ktorej sa vytvára voda. Vtip spočíva v súčasnom zoskupení dvoch atómov vodíka (H) a jedného atómu kyslíka (O). Atómy sú vlákna. Každý H atóm vyvoláva procedúru **HReady**, keď je pripravený na reakciu a každý atóm kyslíka vyvolá **OReady**, keď je pripravený. Je potrebné, aby ste vytvorili kód procedúr **HReady** a **OReady**. Tieto procedúry musia čakať, kým nie sú prítomné aspoň dva H atómy a jeden O atóm a jedna z týchto procedúr musí zavolať procedúru **MakeWater** (ktorá len vygeneruje ladiaci výpis, že voda bola vytvorená). Po zavolaní **MakeWater**, dve inštancie **HReady** a jedna **OReady** by sa mali ukončiť. Napíšte kód pre **HReady** a **OReady** s použitím buď semaforov alebo zámkov a podmienených synchronizačných premenných. Vaše riešenie musí eliminovať starváciu a zaneprázdnené čakanie (busy-waiting).

### Riešenie:

Neustále budú paralelne spúšťané nové vlákna z dvoch druhov (**H**, **O - ready**).

## 13) Veľryby

### Zadanie:

Bezohľadné komerčné záujmy nebezpečne znížili svetovú veľrybiu populáciu, takže veľryby majú synchronizačné problémy pri hľadaní partnera. Zvláštnosť spočíva v tom, že pre zrodenie potomstva sú potrebné až tri veľryby – samec, samica a “dohadzovač”. Vašou úlohou je naprogramovať tri procedúry: **Male()**, **Female()** a **Matchmaker()**. Každá veľryba je reprezentovaná osobitným vláknom. Samec volá procedúru **Male()**, ktorá čaká, kým sa neobjaví samica a “dohadzovač”; podobne, samica musí čakať, kým sa neobjaví samec a “dohadzovač”. Keď sú všetky veľryby prítomné, všetky uvedené procedúry sa ukončia.

### Riešenie:

Neustále je paralelne spúšťané ďalšie vlákno niektorého z uvedených troch typov. Samec a samica si vzájomne púšťajú binárne semafore (dvojica semaforov usporiadaná pre riadenie typu “rendzevous”). Potom ešte musí byť jedna podmienka, ktorú nastavuje dohadzovač.

## 14) Zapisovatelia a čitatelia

### Zadanie:

Viacere zapisujúce a viacere čítajúce procesy súčasne prístupujú do jedného informačného záznamu, pričom tento záznam je kritická oblasť: môže byť mnohými procesmi súčasne čítaný, ale len jeden proces ho môže v jednom okamihu prepisovať, vtedy k záznamu nemôžu prístupovať ani čítajúce a ani iné zapisujúce procesy. Príkladom z praxe je systém rezervácie cestovných lístkov alebo vstupeniek, kde mnohí klienti doňho prístupujú paralelne v rovnakom čase.

### Riešenie:

V nasledujúcom zdrojovom texte je ukázané základné principiálne riešenie s použitím semaforov. Vstup zapisujúcich procesov do kritickej oblasti je riadený semaforom zápis, pričom naraz môže zapisovať len jeden proces, alebo naraz skupina procesov môže len čítať. V okamihu keď skupina procesov číta, môže sa k nej pridať ďalší proces čitateľ. Ak vstupujúci čitateľ zistí, že je prvý (teda počet čitateľov je 1), tak potom musí zistiť, či v tej chvíli neprebíha zapisovanie a vtedy čaká. Ak nie je prvý kto číta, nepotrebuje nič zisťovať. Pri skončení svojej činnosti čitateľ zisťuje, či náhodou nebol posledný a vtedy „zhasne svetlo“ – uvoľní semafor zápis. Semafor mutex stráži kritickú oblasť, v ktorej sa pracuje s premennou čitateľ.

```
void zapisovatel(void)
// takýchto procesov môže súčasne bežať viac
{
    while(1)
    {
        down(zápis); // nie je vnútri žiadny čitateľ
        zapisuj();   // teda môžem zapisovať
        up(zápis);   // skončil som
    }
}

void citatel(void)
// takýchto procesov môže súčasne bežať viac
{
    while(1)
    {
        down(mutex); // vchádzam medzi čitateľov
        citatel++;    // vstupujem do skupiny čitateľov
        if(citatel==1) // ešte nikto nečíta, som prvý
            down(zápis); // píše niekto? vtedy čakám
        up(mutex);    // už som vošiel medzi čitateľov
        čítam_si_čítam();
        down(mutex);  // vychádzam zo skupiny čitateľov
        citatel--;     // bude ich menej
        if(citatel==0) // bol som posledný čitateľ
            up(zápis); // nikto nečíta, chcete zapisovať?
        up(mutex);    // vyšiel som zo skupiny čitateľov
    }
}
```

## 15) Vstup riadku

### Zadanie:

Doplňte synchronizáciu do volajúceho procesu, V/V procedúry, ovládača a modulu obsluhy prerušenia pre vstup riadku. Uveďte začiatočné hodnoty použitých semaforov. Pre semaforey použite napr. názvy 'ziadostoriadok', 'klavesstlaceny', 'masriadok'.

```
int pocet; char riadok[80];

VstupRiadku(riadok, pocet);

/* Tu sa pozaduje riadok */

SYSCALL VstupRiadku(...) {
    vytvor novu poziadavku na riadok;
    pridaj poziadavku do raduPoziadaviek;
    return OK;
}

void TermDriver(){
    while (1) {
        vyber poziadavku z raduPoziadaviek;
        while (1) {
            pozaduj znak z klavesnice;
            if (stav_zariadenia == chybne) {
                zaznamenaj informáciu o chybe;
            }
            spracuj znak;
            if (znak == EOL) break;
        }
        zrus V/V poziadavku;
    }
}

InterruptHandler(){
    /* klavesa stlacena */
}
```

### Riešenie:

VstupRiadku je paralelne bežiacie vlákno, ktoré vkladá údajové položky (akési riadky) do radu požiadaviek a TermDriver je vlákno, ktoré z radu vyberá. Vkladanie a výber je kritická oblasť, ktorú zamykáme jedným binárnym semaforom. Ďalšie dva semaforey nech kontrolujú preplnenie/vyprázdnenie radu požiadaviek, teda voláme operácie up/down na príslušný semafor pri vkladaní/vyberaní.

## 16) Semaforey I.

### Zadanie:

Napíšte všetky možné hodnoty premennej `x` po ukončení nasledovného programu. Odpoveď zdôvodnite.

```
int x=0; sem_hndl s1 = -1, s2 = -1;
/* pre istotu priradme nepoužitelnú hodnotu, semafor musíme
najskeor inicializovat !!! */

void Process1()
{ Wait(s2); Wait(s1); x *= 2; Signal(s1); }

void Process2()
{ Wait(s1); x *= x; Signal(s1); }

void Process3()
{ Wait(s1); x += 3; Signal(s2); Signal(s1); }

main()
{
    if (Semaphore(&s1,1) == SYSERR ||      /* init na 1 */
        Semaphore(&s2,0) == SYSERR)      /* init na 0 */
        abort("Semaphore initialization failed");

    CreateProcess(Process1);
    CreateProcess(Process2);
    CreateProcess(Process3);
    Suspend(myself); /* main čaká na dokončenie detí */
}
```

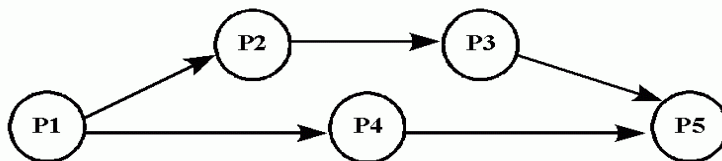
### Riešenie:

Semaforey `s1`, `s2` sú inicializované na hodnoty 1 a 0, to vidno v časti `main`. Pozeráme sa len na tie tri telá procesov `void Process()`, ostatné je v programe len na ozdobu. **Wait / signal** sú operácie na semafore **down / up**. Na začiatku sa dvojka a trojka paralelne rozbehnú a niektorá z nich prejde cez semafor, tá aktuálne oneskorená musí čakať a jednotka tiež čaká. Premenná `X` sa buď najprv umocní, alebo najprv vynásobí dvoma. Teda dve rôzne histórie. Proces 1 sa vôbec pohne až po vykonaní operácie `signal(s2)` tretieho. A pozor, tie procesy nie sú cykly, vykonajú sa len raz za život ☺

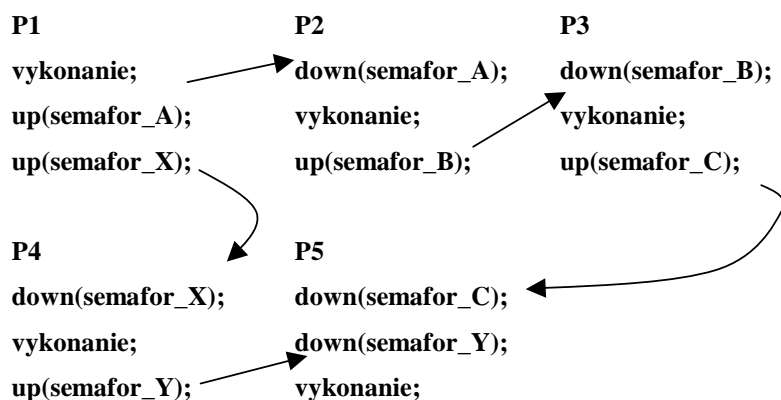
## 17) Semaforey II.

### Zadanie:

Napíšte s použitím semaforov programovú štruktúru piatich procesov, ktorých vykonanie bude usporiadané v čase podľa nasledujúceho orientovaného grafu. Každý nasledujúci proces sa spustí až vtedy, keď sa jeho predchodca skončil.

**Riešenie:**

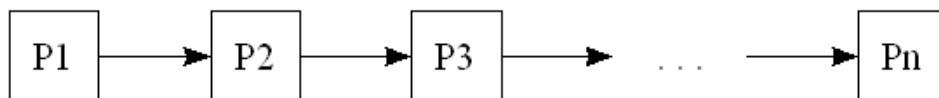
Procesy P1, P2, P3 a P5 budú vykonané sekvenčne, proces P4 bude vykonaný súbežne s procesmi P2 a P3. Pre každú hranu v grafe zavedieme jeden dvojstavový semafor. Semaforey označíme A,B,C pre hornú vetvu grafu a X,Y pre dolnú vetvu grafu. Počiatočná hodnota všetkých semaforov bude 0. Štruktúra jednotlivých procesov bude nasledovná:



Proces P5 sa môže začať vykonávať až po ukončení oboch vetiev procesov, preto čaká na dvoch semaforoch.

**18) Semaforey III.****Zadanie:**

Zabezpečte pomocou semaforov lineárne usporiadanie N procesov. Aký najmenší počet semaforov je na to potrebný? Ako budú vyzeráť skelety procesov?

**Riešenie:**

Podľa predošlého príkladu ihneď vezmeme N-1 semaforov, vložíme ich medzi procesy a máme riešenie. Otázka je, či je to najmenší možný počet semaforov. V tejto pokročilej hodine si nejasne spomínam, že možno po každom asi treťom? procese by bolo možné

použiť zase ten istý semafor, pretože každý tretí čaká na každý druhý...? Na skúške ma poteší aj to prvé riešenie.

## 19) Semaforey IV.

### Zadanie:

Počiatočná hodnota semaforu je 0. Akú maximálnu hodnotu môže semafor nadobudnúť počas vykonávania nasledujúcich troch paralelne bežiacich procesov? Aké rozličné hodnoty môže nadobudnúť semafor v okamihu po skončení procesov 1 a 2 ?

```
Proces1() { for(int i=0;i<2;i++) up(); }
Proces2() { for(int i=0;i<3;i++) up(); }
Proces3() { for(int i=0;i<4;i++) down(); }
```

### Riešenie:

Semafor je v systéme len jeden. Pri takej situácii plánovania, keď prvé dva procesy prebehnú až do svojho konca a tretí proces sa ešte nespustil, bude semafor inkrementovaný dohromady 5-krát a tak dosiahne maximálnu možnú hodnotu 5. Ak sa medzitým spustil tretí proces, tak mohol stihnúť znížiť hodnotu semafora nanajvyš 4-krát, takže možné hodnoty semafora po skončení prvých dvoch procesov sú 5, 4, 3, 2, 1. Pozor, obvyklou chybou je, keď prehliadneme koľkokrát sa jednotlivé cykly **for** naozaj spúšťajú.

## 20) Postupnosti znakov I.

### Zadanie:

Nasledujúce dva paralelne vykonávané procesy vytvárajú na spoločnom výstupe postupnosť znakov X,Y. Koľko najviac rovnakých znakov môže byť vytlačených po sebe? Semaforey majú počiatočnú hodnotu 0.

```
Proces1()
{
    for(int i=0;i<9;i++)
    {
        up(semafor_a);
        printf("X");
        down(semafor_b);
    }
}

Proces2()
{
    for(int i=0;i<9;i++)
    {
        up(semafor_b);
        printf("Y");
        down(semafor_a);
    }
}
```



**Riešenie:**

Predpokladáme, že poradie prepínania procesov je ľubovoľné a synchronizáciu riadia iba uvedené dva semaforey a, b. Predpokladajme, že jeden proces môže vykonať trikrát po sebe príkaz výstupu `printf()` bez toho, aby sa tento striedal s výstupom druhého procesu. Takýto pokus o riešenie vyzerá nasledovne:

Proces 2: <code>up(b)</code>	<code>b-&gt;1</code>	(inkrementácia stavu semafora b)
<code>printf(Y)</code>		
<code>down(a)</code>	<code>a=0(-1)</code>	(stop, zaspím)
Proces 1: <code>up(a)</code>	<code>a-&gt;0</code>	(zobudenie P2)
<code>printf(X)</code>		
<code>down(b)</code>	<code>b-&gt;0</code>	
<code>up(a)</code>	<code>a-&gt;1</code>	
<code>printf(X)</code>		
<code>down(b)</code>	<code>b=0(-1)</code>	(stop, zaspím)
Proces 2: <code>up(b)</code>	<code>b-&gt;0</code>	(zobudenie P1)
Proces 1: <code>up(a)</code>	<code>a-&gt;2</code>	
<code>printf(X)</code>		
<code>down(b)</code>	<code>b=0(-1)</code>	(zaspatie P1)
Proces 2: <code>printf(Y)</code>		(iným nemožno pokračovať)

Nasledujúca úvaha ukazuje, že jeden proces môže vykonať štyrikrát po sebe príkaz výstupu `printf()` bez toho, aby sa tento príkaz striedal s výstupom druhého procesu:

Proces 2: <code>up(b)</code>	<code>b-&gt;1</code>
<code>printf(Y)</code>	
<code>down(a)</code>	<code>a=0(-1)</code>
Proces 1: <code>up(a)</code>	<code>a-&gt;0</code>
Proces 2: <code>up(b)</code>	<code>b-&gt;2</code>
<code>printf(Y)</code>	
<code>down(a)</code>	<code>a=0(-1)</code>
Proces 1: <code>printf(X)</code>	
<code>down(b)</code>	<code>b-&gt;1</code>
<code>up(a)</code>	<code>a-&gt;0</code>
<code>printf(X)</code>	
<code>down(b)</code>	<code>b-&gt;0</code>
<code>up(a)</code>	<code>a-&gt;1</code>
<code>printf(X)</code>	
<code>down(b)</code>	<code>b=0(-1)</code>
Proces 2: <code>up(b)</code>	<code>b-&gt;0</code>
Proces 1: <code>up(a)</code>	<code>a-&gt;2</code>
<code>printf(X)</code>	
<code>down(b)</code>	<code>b=0(-1)</code>
Proces 2: <code>printf(Y)</code>	

Poklesom do záporného čísla (-1) je možné vyjadriť stav, keď je na semafore zastavený jeden proces a plánovač ho nemôže spúšťať. Návratom na 0 je zastavený proces uvoľnený, plánovač ho môže niekedy spustiť. Samotný semafor formálne nadobúda len hodnotu zdola ohraňujú nulu, záporným číslom môžeme vyjadriť počet procesov na semafore zastavených.

Pri napísaní takéhoto riešenia je otázkou, či sme našli naozaj to konečné riešenie a ako by sme dokázali že nejedná o iné správnejšie. To je už vlastne určitý matematický problém, ktorý sa prejaví pri väčšom počte procesov a zložitejšej komunikácii.

## 21) Postupnosti znakov II.

### Zadanie:

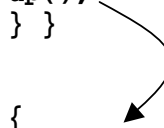
Áká bude postupnosť vytlačených znakov A, B po skončení nasledujúcich dvoch paralelne bežiacich procesov? Počiatočná hodnota semaforu je 0.

```

Proces_1() {
    for(int i=0;i<2;i++) {
        printf("A");
        up();
    }
}

Proces_2() {
    for(int i=0;i<3;i++) {
        down();
        printf("B");
    }
}

```

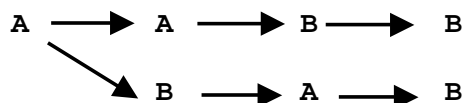


### Riešenie:

Predpokladáme, že naplánované poradie a prerušovanie procesov je vopred neznáme. Semafor je v systéme len jeden. Analýzou vzťahov medzi obidvoma procesmi možno odvodiť tieto tri tvrdenia:

- druhý proces nemôže nikdy na začiatku prejsť cez operáciu **down()**, kým prvý proces nevytlačí svoje písmeno a nevykoná operáciu **up()**,
- prvý proces môže vykonať dva svoje cykly po sebe, potom aj tak skončí,
- druhý proces nikdy nevykoná svoj tretí cyklus, pretože semafor je inkrementovaný prvým procesom iba dvakrát.

Generovanie výstupu vyjadrené vo forme rozhodovacieho stromu:



**22) Postupnosti čísel I.****Zadanie:**

Aké výstupy (postupnosti čísel) môže generovať program pozostávajúci z uvedených procesov? Zdôvodnite!

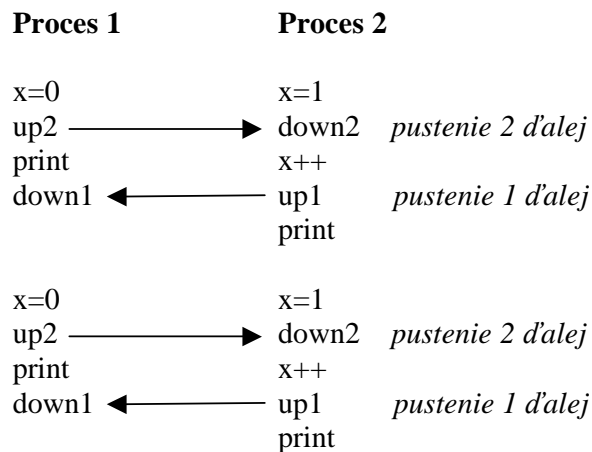
```
Semaphore sem1, sem2; int x;
```

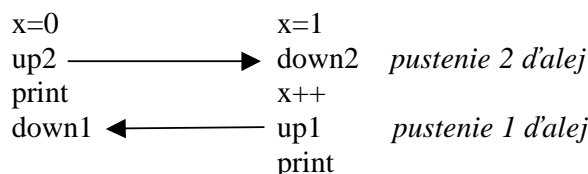
```
Process p1(){
    while (1)
    {
        x = 0;
        up(sem2);
        printf("%d", x);
        down(sem1);
    }
}
```

```
Process p2(){
    while (1)
    {
        x = 1;
        down(sem2);
        x++;
        up(sem1);
        printf("%d", x);
    }
}
```

**Riešenie:**

Obidva procesy bežia v nekonečných slučkách. Vzhľadom na synchronizáciu bude postupnosť na výstupe pozostávať z týchto pod-postupností: **00, 01, 02, 11, 12**. To sú možné priebehy, ktoré sa opakovane budú vyskytovať počas behu. Môže tam byť **21**? Dá sa to napísať na papier ako história takto:





atakďalej, keď je to takto rozpísané možno to považovať za polovičné riešenie príkladu, nanajvýš sa pomýlime vo výroku aká postupnosť môže/nemôže nastať. A ešte, v príklade nie je spomenutá počiatočná hodnota semaforov. Mala by byť. Alebo si povieme v zdôvodnení, že predpokladáme nulovú hodnotu. Alebo akú? Zadanie nám nič povinného v tomto neurčuje ☺

## 23) Postupnosti znakov II.

### Zadanie:

Aká bude postupnosť znakov X a Y po skončení dvoch paralelne vykonávaných procesov 1 a 2, pri použití dvoch semaforov a,b? Semaforey majú na počiatku hodnotu 0.

```
Proces_1()
{
    int I;
    for(I=0;I<3;I++)
        {up(semafor_a); printf("X"); down(semafor_b);}
}

Proces_2()
{
    int I;
    for(I=0;I<3;I++)
        {up(semafor_b); printf("Y"); down(semafor_a);}
}
```

### Riešenie:

Na rozdiel od predošlého príkladu je počet cyklov v procesoch konečný, otočia sa trikrát. A vzájomne sa blokujú a púšťajú ďalej, nemôže jeden príliš predbehnúť druhého. Nakresliť, považovať.

## 24) Postupnosti čísel II.

### Zadanie:

Ake výstupy (postupnosti čísel) môže generovať program pozostávajúci z uvedených procesov?

```
Semaphore sem1 = 0, sem2 = 0; // tu vidno počiatočné hodnoty
int x;
```

```
Process p1(){
    while (1) {
        x = 0;
        Signal(sem2);
        printf("%d", x);
        Wait(sem1);
        printf("%d", x);
    }
}
```

```
Process p2(){
    while (1) {
        x = 1;
        Wait(sem2);
        x++;
        Signal(sem1);
    }
}
```

**Riešenie:**

Je to ďalšia verzia toho pred-predošlého príkladu. Cykly bežia donekonečna, dva semafore ich zväzujú do povoleného rytmu.

**25) Postupnosť čísel III.****Zadanie:**

Ake výstupy (postupnosti čísel) môže generovať program z dvoch procesov?

```
Semaphore sem1 = 0, sem2 = 0;
int x;
```

```
Process p1(){
    while (1) {
        x = 0;
        Signal(sem2);
        printf("%d", x);
        Wait(sem1);
        printf("%d", x);
    }
}
Process p2(){
    while (1) {
        x = 1;
        Wait(sem2);
        x--;
        Signal(sem1);
    }
}
```

**Riešenie:**

Znovu iná verzia. Otázkou na zamyslenie je, či to možno v príkladoch ľubovoľne kombinovať a meniť, teda či nevyrobíme príklad, ktorý by nemal riešenie alebo by ho mal veľmi komplikované. A ešte je zaujímavá otázka, ako by sa dalo nájsť riešenie pomocou simulácie na počítači.

**26) Postupnosť čísel IV.****Zadanie:**

Uveďte všetky možné výstupy programu pozostávajúceho z uvedených paralelných procesov. Odpoveď zdôvodnite.

```
Semaphore sem = 0;
int x;

Process p1(){
    x = -1;
    Signal(sem);
    printf("%d", x);
}

Process p2(){
    x = 1;
    Wait(sem);
    printf("%d", x);
}
```

**Riešenie:**

Žiadne opakujúce sa cykly a iba jeden jediný semafor. To tuším bolo na opravnej písomke a dúfali sme, že tým mnohých študentov zachrátime...

**27) Postupnosť čísel V.****Zadanie:**

Uveďte všetky možné výstupy programu pozostávajúceho z uvedených paralelných procesov. Odpoveď zdôvodnite.

```
Semaphore sem = 0;
int x;

Process p1(){
    x = -1;
    Signal(sem);
    printf("%d", x);
}
```

```
Process p2(){
    x = 1;
    Wait(sem);
    printf("%d", x);
}
```

**Riešenie:** ako predošlé

## 28) Semaforey V.

### Zadanie:

Počiatočná hodnota semaforu je 0.

1. Ako maximálnu hodnotu môže semafor nadobudnúť počas vykonávania nasledujúcich troch paralelne bežiacich procesov ?
2. Nakreslite v časovom diagrame jeden príklad histórie stavu semaforu od počiatku až po moment kedy nadobudne svoju maximálnu hodnotu.
3. Ake rozličné hodnoty môže nadobudnúť semafor v okamihu po skončení procesov **Proces1** a **Proces2** ?

```
Proces1() {
    for(int i=0;i<2;i++)
        signal_semafor(semafor_a); // semafor up
}

Proces2() {
    for(int i=0;i<3;i++)
        signal_semafor(semafor_a); // semafor up
}

Proces3() {
    for(int i=0;i<4;i++)
        wait_semafor(semafor_a); // semafor down
}
```

### Riešenie:

Maximálna hodnota je 5. Históriu nakreslíme ako lomenú čiaru v grafe, kde os x = časové cykly a os y = stav semaforu. Napríklad rovnomerne stúpajúce schodíky. Odpoveď na tretiu otázku je, že hodnoty 5,4,3,2,1 pretože procesy prvý a druhý dvíhali semafor dokopy päťkrát a tretí proces buď vôbec ešte nebežal, alebo bežal až max štyrikrát, ako chcel.

## 29) Postupnosti znakov III.

### Zadanie:

Počiatočná hodnota semaforu je 0.

Napíšte všetky možné postupnosti na výstupe systému, ktorý sa skladá z týchto dvoch paralelne bežiacich procesov:

```
Proces1() {
    for(int i=0; i < 2 ;i++)
    {
        printf("A");
        signal_semafor(semafor_a); // semafor up
    }
}

Proces2() {
    for(int i=0; i < 3 ;i++)
    {
        wait_semafor(semafor_a); // semafor down
        printf("B");
    }
}
```

### Riešenie:

Semafor iba jeden, prvý proces vykoná dva cykly a druhý tri... nakresliť.

## 30) Postupnosti čísel VI.

### Zadanie:

Počiatočná hodnota každého semaforu je 0.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Aké rozličné hodnoty môžu nadobudnúť obidva semafore v okamihu po skončení procesu **vlakno\_B** ?

```
int x=y=z=0; // spolocne zdielane premenne
```

```
vlakno_A() {
    for(int i=0; i < 2 ;i++)
    {
        <<x = i;>> // atomicka operacia, vlákno B nemôže narušiť
        signal_semafor(semafor_prvy); // semafor up
    }
}

vlakno_B() {
    for(int i=0; i < 3 ;i++) {
        <<y = i;>> // atomicka operacia, vlákno A nemôže narušiť
        signal_semafor(semafor_druhy); // semafor up
    }
}
```

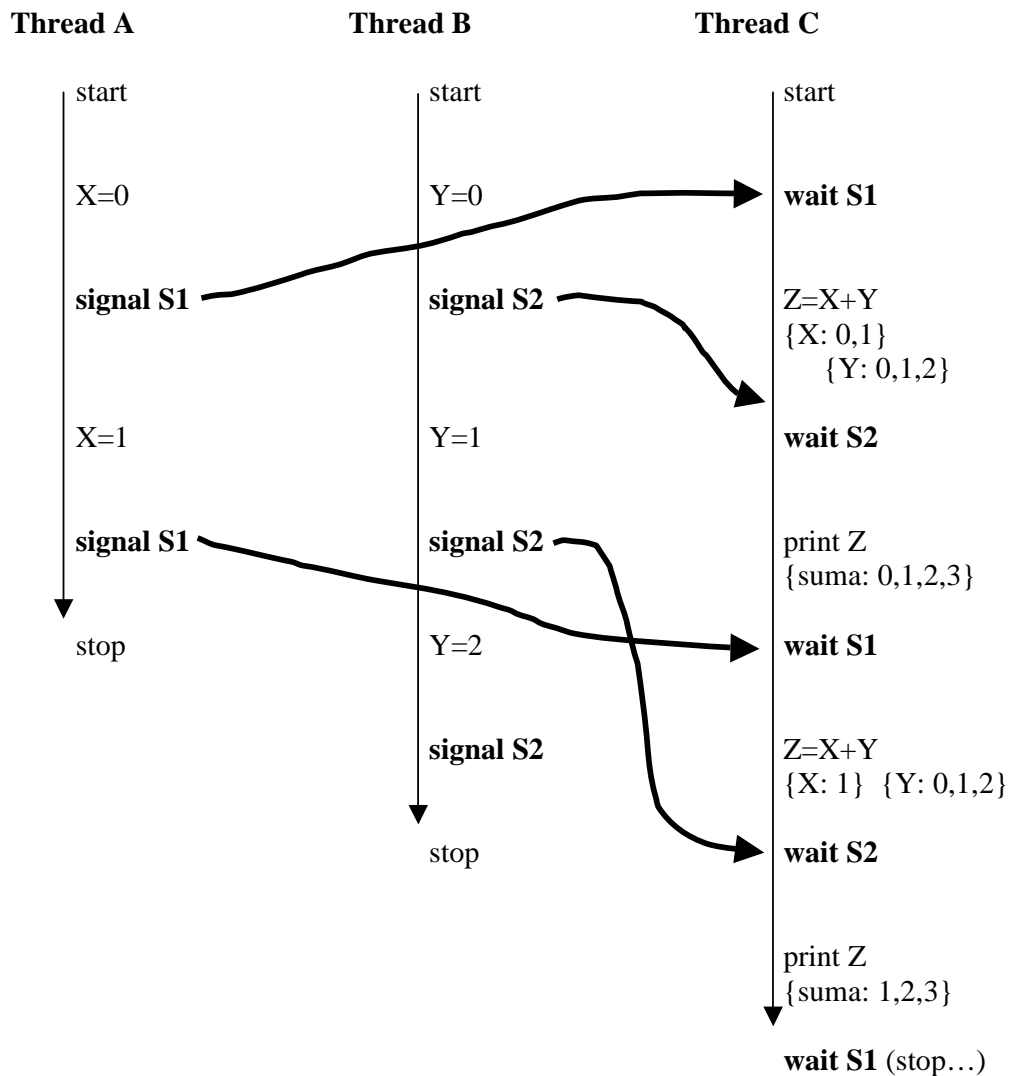


```

vlakno_C() {
    for(int i=0; i < 3 ;i++) {
        wait_semafor(semafor_prvy);    // semafor down
        // atomicka operacia, vlákna A,B nemôžu narušiť
        <<z = x + y;>>
        wait_semafor(semafor_druhy);    // semafor down
        printf("%d",z);
    }
}

```

Riešenie:



Šípky zľava-doprava znamenajú, že vľavo bežiaci proces dáva signál na pustenie vpravo bežiaceho procesu. Zároveň, vľavo bežiaci proces môže ľubovoľne ďaleko dopredu predbehnúť vpravo bežiaci proces. Toto je riešenie pre skupinu A, skupina B mala iný počet cyklov a namiesto súčtu súčin.

Stav semaforov po skončení vlákna B: ľubovoľný od 0 až po maximálny počet volaní signal, pretože vlákno A a vlákno B sa pri svojej činnosti neovplyvňujú, vlákno C môže ale ešte nemusí stihnúť znížiť hodnoty semaforov.

### 31) Postupnosti čísel VII.

#### Zadanie:

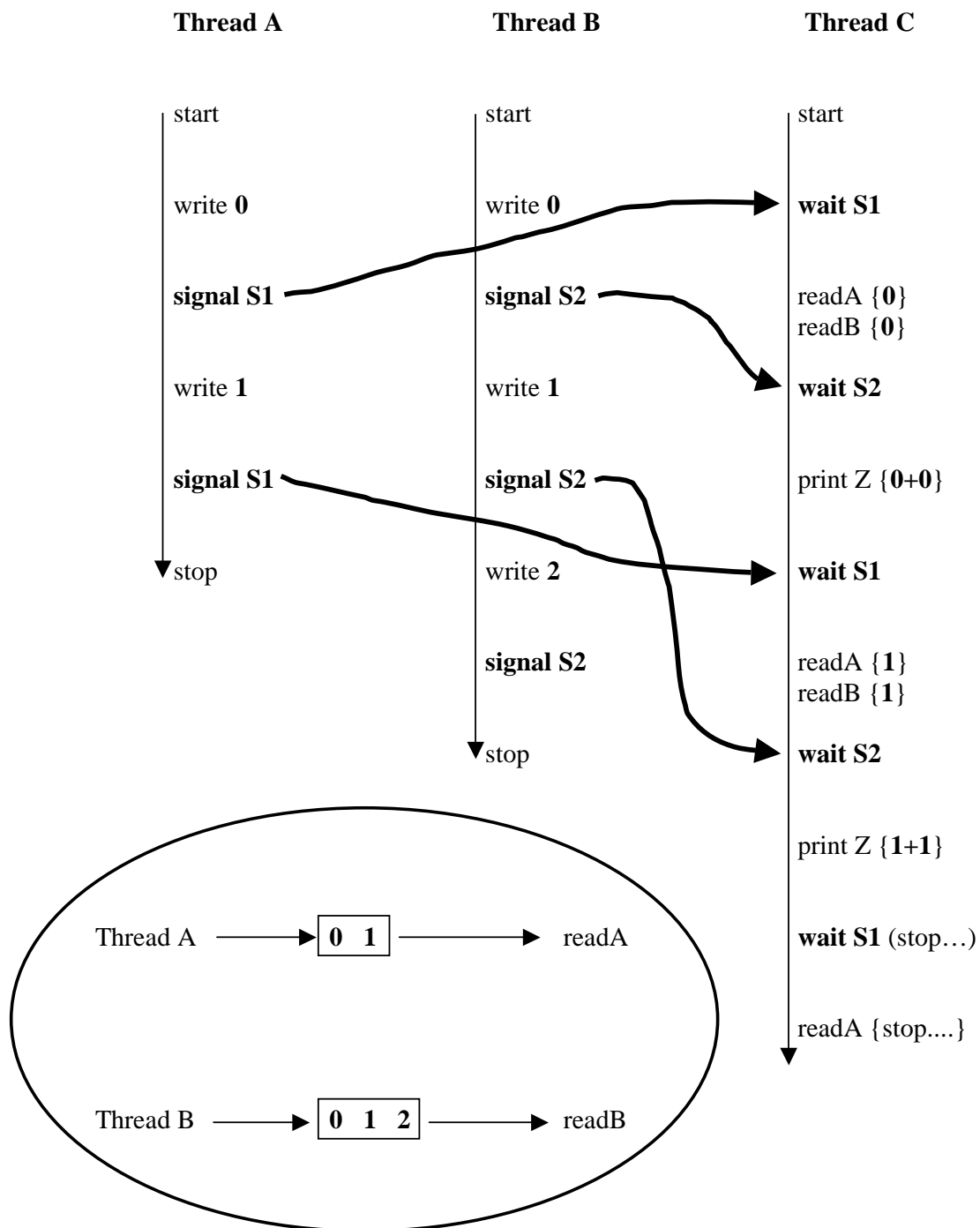
Počiatočná hodnota každého semaforu je 0. Procesy komunikujú za pomoci dvoch rúr.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Čo sa zmení, ak z programu vymažeme synchronizáciu pomocou semaforov? Zdôvodnite.

```
Proces_A() {
    for(int i=0; i < 2 ;i++)
    {
        vloz_do_rury_prvej(i);
        signal_semafor(semafor_prvy); // semafor up
    }
}

Proces_B() {
    for(int i=0; i < 3 ;i++)
    {
        vloz_do_rury_druhej(i);
        signal_semafor(semafor_druhy); // semafor up
    }
}

Proces_C() {
    for(int i=0; i < 3 ;i++)
    {
        wait_semafor(semafor_prvy); // semafor down
        a=vyber_z_rury_prvej();
        b=vyber_z_rury_druhej();
        wait_semafor(semafor_druhy); // semafor down
        printf("%d",a + b);
    }
}
```

**Riešenie:**

Čísla sa ukladajú do rúr ako do radov FIFO a vyberajú sa postupne. Prirodzené rúry by mali byť pri čítaní blokujúce (proces pri prázdnej rúre musí zostať čakať, lebo inak chyba), teda odstránenie semaforov nezmení výstup, nanajvýš tretie vlákno/proces pobeží ešte krokom readA, na ktorom sa zastaví pri prázdnej rúre. Obsah čo sa posúva cez rúry je naznačený v elipse.

### 32) Postupnosti čísel VIII.

**Zadanie:**

Počiatková hodnota každého semaforu je 0.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Aké rozličné hodnoty môžu nadobudnúť obidva semafore v okamihu po skončení procesu **vlakno\_B** ?

```
int x=y=z=0; // spolocne zdielane premenne

vlakno_A() {
    for(int i=0; i < 2 ;i++)
    {
        <<x = i;>> // atomicka operacia, premenna je uzamknuta
        signal_semafor(semafor_prvy); // semafor up
    }
}

vlakno_B() {
    for(int i=0; i < 3 ;i++)
    {
        <<y = i;>> // atomicka operacia, premenna je uzamknuta
        signal_semafor(semafor_druhy); // semafor up
    }
}

vlakno_C() {
    for(int i=0; i < 3 ;i++)
    {
        wait_semafor(semafor_prvy); // semafor down
        // atomicka operacia, premenne su uzamknute
        <<z = x * y;>>
        wait_semafor(semafor_druhy); // semafor down
        printf("%d",z);
    }
}
```

**Riešenie:** Je to iná verzia predošlého príkladu.

### 33) Postupnosť čísel IX.

**Zadanie:**

Počiatková hodnota každého semaforu je 0. Procesy komunikujú za pomoci dvoch rúr.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Čo sa zmení, ak z programu vymažeme synchronizáciu pomocou semaforov? Zdôvodnite.

```
Proces_A() {
    for(int i=0; i < 2 ;i++)
    {
        vloz_do_rury_prvej(i);
        signal_semafor(semafor_prvy); // semafor up
    }
}

Proces_B() {
    for(int i=0; i < 2 ;i++)
    {
        vloz_do_rury_druhej(i);
        signal_semafor(semafor_druhy); // semafor up
    }
}

Proces_C() {
    for(int i=0; i < 3 ;i++)
    {
        wait_semafor(semafor_prvy); // semafor down
        a=vyber_z_rury_prvej();
        b=vyber_z_rury_druhej();
        wait_semafor(semafor_druhy); // semafor down
        printf("%d",a * b);
    }
}
```

**Riešenie:** Znovu iná verzia.

### 34) Implementácia rúry.

**Zadanie:**

Nižšie je uvedený zdrojový text pre základné operácie s rúrou (pipe).

Rúra ma kapacitu desať znakov a nikdy sa nepoužije na prenos väčšieho počtu znakov, to neriešime.

1. Je táto implementácia v poriadku aj pre prípad, že rúru bude naraz používať viacero procesov? Zdôvodnite.
2. Navrhnite také riešenie pomocou semaforov, ktoré odstráni problémy, čo ste v kóde našli.
3. Ako sa bude správať rúra pri komunikácii, ak do nižšie uvedených funkcií použijeme namiesto semafora vzájomné vylučovanie pomocou zámku: operácie `lock()` a `unlock()` (zamkni a odomkni)?

```
pole R[10], int i,j;
```

```
vloz_do_rury (char znak)
{ R[i]=znak; i++; }
```

```
vyber_z_rury (char *znak)
{ while(j==i); *znak= R[j]; j++; }
```

**Riešenie:**

Pod rúrou budeme chápať ten kúsok programu, čo je tam napísaný. Pre viac procesov naraz vkladajúcich alebo vyberajúcich to bude fungovať len vtedy, keď to vloženie/vybratie bude celé atomická operácia. Keď jeden číta a druhý vkladá, tak to fungovať bude ... to asi zaručí konštrukcia `while(j==i);`

Jeden semafor môže strážiť vstup do kritickej oblasti, v ktorej sa bude manipulovať s druhým semaforom, ktorého hodnota bude na začiatku nastavená na maximálnu kapacitu rúry. Namiesto i, j postačí potom už len jeden index.

Pri použití zámku tiež zabránime kolízii, ale treba kontrolovať preplnenie rúry.

**35) Semaforey VI.****Zadanie:**

Nižšie je uvedený zdrojový text troch procesov, ktoré sú vzájomne synchronizované štyrmi semafori.

1. Aký výstup sa môže objaviť z tretieho procesu? Uveďte všetky možnosti a zdôvodnite.
2. Ako sa zmení situácia a výstup, ak nepoužijeme semaforey B,D ?
3. Čo sa zmení, ak prehodíme poradie príkazov podčiarknutých hviezdičkami?

```
int x=0;y;
```

```
proces1()
{ for ( i=0; i<2; i++ )
  { x++; signal(A); wait(B); }
}
```

```
proces2()
{ for ( i=0; i<2; i++ )
  { wait(A); y=x+1; signal(B); signal(C); wait(D); }
  } //*****
```

```
proces3()
{ for ( i=0; i<2; i++ )
  { wait(C); xprint(y); signal(D); }
}
```

**Riešenie:**

- 1: vytlačí sa postupnosť 2,3. Nič iné, všetko je perfektne zosynchronizované.
- 2: vzniknú rôzne kombinácie na výstupe, slušné je spomenúť aspoň jednu...
- 3: nič sa nezmení (uff...)

**36) Postupnosti čísel X.****Zadanie:**

Počiatočná hodnota každého semaforu je 0.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Aké rozličné hodnoty môžu nadobudnúť obidva semafore v okamihu po skončení procesu **vlakno\_A** ?

```
int x=y=z=0; // spolocne zdielane premenne

vlakno_A() {
    for(int i=0; i < 2 ;i++)
    {
        signal_semafor(semafor_prvy);    // semafor up
        <<x = i;>>    // atomicka operacia, premenna je uzamknuta
    }
}

vlakno_B() {
    for(int i=0; i < 2 ;i++)
    {
        <<y = i;>> // atomicka operacia, premenna je uzamknuta
        signal_semafor(semafor_druhy);    // semafor up
    }
}

vlakno_C() {
    for(int i=0; i < 2 ;i++)
    {
        // atomicka operacia, premenne su uzamknute
        <<z = x + y;>>

        wait_semafor(semafor_prvy);        // semafor down
        wait_semafor(semafor_druhy);        // semafor down
        printf("%d",z);
    }
}
```

**Riešenie:** je to ďalšia verzia predošlých príkladov...

### 37) Postupnosti čísel XI.

**Zadanie:**

Počiatočná hodnota každého semaforu je 0. Procesy komunikujú za pomoci dvoch rúr.

1. Aké rozličné postupnosti čísel môžu byť vytlačené na výstupe procesu C?
2. Čo sa zmení, ak z programu vymažeme synchronizáciu pomocou semaforov? Zdôvodnite.

```
Proces_A() {
    for(int i=0; i < 2 ;i++)
    {
        vloz_do_rury_prvej(i);
        signal_semafor(semafor_prvy); // semafor up
    }
}

Proces_B() {
    for(int i=0; i < 2 ;i++)
    {
        vloz_do_rury_druhej(i);
        signal_semafor(semafor_druhy); // semafor up
    }
}

Proces_C() {
    for(int i=0; i < 3 ;i++)
    {
        wait_semafor(semafor_prvy); // semafor down
        a=vyber_z_rury_prvej();
        b=vyber_z_rury_druhej();
        wait_semafor(semafor_druhy); // semafor down
        printf("%d",a * b);
    }
}
```

**Riešenie:** znovu ďalšia verzia z predošlých príkladov

### 38) Semaforey VII.

**Zadanie:**

Počiatočná hodnota pre `semafor_1` je 1, pre `semafor_2` je 0. Počiatočná hodnota zdieľanej premennej `x` = 0.

1. Aké sú možné hodnoty premennej `x` po skončení programu?
2. Aká bude postupnosť činnosti týchto dvoch paralelných procesov? Graficky znázornite.



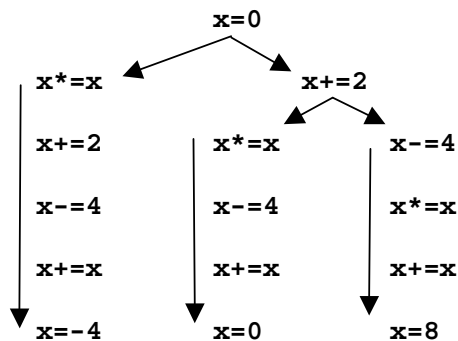
```

Proces_A() {
    P(semafor_1); // semafor down
    x*=x;
    V(semafor_1); // semafor up
    P(semafor_2);
    P(semafor_1);
    x+=x;
    V(semafor_1);
}
Proces_B() {
    P(semafor_1);
    x+=2;
    V(semafor_1);
    P(semafor_1);
    x+=4;
    V(semafor_1);
    V(semafor_2);
}

```

**Riešenie:**

Je možné si vytvoriť takýto stromček možností, takže ak som to správne odpísal z papiera vznikne toto:

**39) Postupnosti znakov IV.****Zadanie:**

Počiatočná hodnota semaforu je 0.

Napište všetky možné postupnosti na výstupe systému, ktorý sa skladá z týchto dvoch paralelne bežiacich procesov:

```

Proces1() {
    for(int i=0; i < 2 ;i++)
    {
        printf("A");
        signal_semafor(semafor_a); // semafor up
    }
}

```

```

Proces2() {
    for(int i=0; i < 3 ;i++)
    {
        printf("B");
        wait_semafor(semafor_a); // semafor down
    }
}

```

**Riešenie:**

Je tu len jeden jediný semafor, druhý proces pomocou neho vždy čaká na prvý. Vo svojom poslednom cykle však druhý proces zostane čakať a neskončí, pretože prvý proces už semafor do tretice nezdvihne.

**40) Implementácia semafora v jazyku C.****Zadanie:**

Napíšte v jazyku C implementáciu operácií semafora `init()`, `P()`, `V()` s použitím funkcií `send()`, `receive()`.

**Riešenie:**

Nič iné len toto (prečo?):

```

Init(x) {int i; for(i=0;i<x;i++) send('@'); }
P()      {(void)receive();}
V()      {send('*');}

```

**41) Semafor a fork dohromady I.****Zadanie:**

```

for (i = 0; i < 2 ; i++) {
    x = fork();
    if (som_rodic) {
        semafor_down(); // B
        xprint("R");    // C
        exit();         // D
    }
    else {
        xprint("P");    // E
        semafor_up();  // F
    }
}

```

Tento proces sa rozmnožuje a v systéme je len jeden semafor, inicializovaný na hodnotu nula. Procesy sú plánované algoritmom Round Robin, ktorý keď dosiahne momentálny

koniec zoznamu tak v ďalšom kroku naplánuje prvú položku. V jednom časovom kvante sa dokáže vykonať práve jedna operácia.

Odpovede na nasledujúce dve otázky vysvetlite pomocou vývoja stavu zoznamu procesov po každom kvante (použite uvedené písmenové označenie operácií):

Ako bude vyzerat' postupnosť operácií v prvých dvanástich kvantách?  
Ako bude vyzerat' postupnosť písmen na výstupe xprint?

### Riešenie:

1.	2.	3.	<i>pribúdajúce položky v zozname procesov</i>
A	E		Round Robin zlava doprava
B	F		...a znovu zlava doprava v zozname procesov
C	A	E	...pribudol tretí proces v zozname
D	B	F	
-	C	A	...to je 12.kvantum, už dosť!

(udalosti teda prebehnú AEBFCAEDBFCA, ale oindexujte si ich pre tri rôzne procesy<sub>1,2,3</sub>)

Ešte je ďalšia verzia riešenia, keď novovytvorený proces je zaradený do Round Robin až v ďalšom cykle:

1.	2.	3.	<i>pribúdajúce položky v zozname procesov</i>
A	nový		
B	E		
sleep	F		
C	A	nový	
D	B	E	
-	sleep	F	
-	C	A	...to je 12.kvantum, už dosť!

Postupnosť písmen je pre obidve verzie riešenia **“PRPR”**

**42) Semafor a fork dohromady II.****Zadanie:**

Rovnaký text príkladu ako predošlý, iný kód:

```
for (i = 0; i < 2 ; i++) {
    x = fork();
    if (som_rodic) {
        semafor_down();
        xprint("R");
    }
    else {
        xprint("P");
        semafor_up();
        exit();
    }
}
```

**Riešenie:**

Postupovať ako v predošlom prípade (tentoraz sa časť sekvencie jedenkrát pekne zopakuje), postupnosť vytlačených písmen je pre obidve verzie riešenia znova rovnaká **"PRPR"**.

**43) Semafor a fork dohromady III.****Zadanie:**

V systéme je statická tabuľka procesov s kapacitou 5 záznamov. Je tam len jeden semafor, inicializovaný na hodnotu nula. Na začiatku spustíme jeden proces s nasledovným obsahom:

```
while (1) {
    x = fork();
    if (som_rodic) semafor_up();
    else semafor_down();
}
```

Jedno časové kvantum zodpovedá vykonaniu jednej operácie. Odpovede na nasledujúce otázky vysvetlite pomocou znázornenia vhodnou označenou schémou:

Koľko časových kvánt pobeží celý tento systém procesov, kým nenastane problém? napíšte stav zoznamu procesov v systéme po každom časovom kvante

Aký bude stav semaforu v okamihu vzniku problému?

**Riešenie:**

Stav semafora v okamihu vzniku problému bude 0, verzie postupu sú znovu dve presne tak ako v predošlých príkladoch – novo zaradený proces naplánujeme až v ďalšom cykle Round Robin, alebo ho naplánujeme ihneď – pre túto verziu je schéma:

1.	2.	3.	4.	5.	<i>pribúdajúce položky v zozname procesov</i>
A	C				
B	A	C			
A	B	A	C	C	
B	A	problém...			

Potomok začína ako C ale v ďalšom cykle už je sám rodičom a vykonáva A, B.

A čo je to ten problém: zoznam procesov v tabuľke má obmedzenú kapacitu ☺

#### 44) Semafor a fork dohromady IV.

##### Zadanie:

Iná verzia kódu, ostatné ako v predošlom príklade:

```
while (1) {
    semafor_up();           // A
    x = fork();              // B
    if (som_rodic) semafor_down(); // C
}
```

##### Riešenie:

Stav semafora v okamihu vzniku problému bude 4, verzie postupu sú znovu dve presne tak ako v predošlých príkladoch – novo zaradený proces naplánujeme až v ďalšom cykle Round Robin, alebo ho naplánujeme ihneď – v tomto prípade sa dožijeme 18, respektíve 19 kvánt po vznik problému.

## Procesy

### 1) Bankárov algoritmus – jeden typ prostriedkov

#### Zadanie:

Máme 5 zákazníkov (5 procesov) P[0]-P[4], pre jednoduchosť 1 typ prideleného prostriedku (peniaze...), aktuálny stav je daný nasledovnou tabuľkou:

	Allocation	Max.	(Need)	Available
P[0]	0	7	7	3
P[1]	2	3	1	
P[2]	3	9	6	
P[3]	2	2	0	
P[4]	0	4	4	

V stĺpci 'Allocation' sú sumy peňazí, ktoré už boli jednotlivým zákazníkom požičané (pridelené), v stĺpci 'Max.' sú maximálne (celkové) sumy, ktoré budú zákazníci požadovať, stĺpec 'Need' uvádza, koľko ešte budú potrebovať a 'Available' je aktuálna suma, ktorú má bankár v banke.

- Ide o bezpečný stav? (Dajte príklad, v akom poradí môžu byť zákazníci - procesy vybavené.)
- Nech zákazník (proces) P[1] žiada o sumu 1 peňažná jednotka. Môže byť jeho požiadavka uspokojená okamžite?
- To isté ako b), ale tentoraz proces P[4] žiada sumu 3.

#### Riešenie:

##### a)

Stav je bezpečný, ak existuje taká postupnosť stavov, pri ktorej budú postupne splnené/obslúžené požiadavky všetkých zákazníkov. Takáto postupnosť vznikne napr. pri nasledovnom poradí vybavovania zákazníkov:

- P[3] - už má koľko žiadal, v konečnom čase pôžičku vráti, po vrátení bude v banke 5
- P[4] - po vrátení pôžičky bude v banke 5
- P[1] - po vrátení pôžičky bude v banke 7
- P[0] - po vrátení pôžičky bude v banke 7
- P[2] - po vrátení pôžičky bude v banke 10

Treba si uvedomiť, že pri tomto algoritme musí byť u každého zákazníka vopred známe, koľko bude požadovať maximálne a toto maximum sa počas hry nemení. V reálnom svete sa to obvykle mení a potom sa musí urobiť takzvané “navýšenie rozpočtu” ☺

**b)**

Ak by bankár vyhovel požiadavke zákazníka P[1], vznikol by stav daný nasledovnou tabuľkou:

	Allocation	Max.	(Need)	Available
P[0]	0	7	7	2
P[1]	3	3	0	
P[2]	3	9	6	
P[3]	2	2	0	
P[4]	0	4	4	

a úlohou bankára je zistiť, či je tento stav bezpečný. Možné poradie vybavovania zákazníkov je napr.:

P[1], P[3], P[4], P[2], P[0].

Nový stav bude bezpečný, bankár teda môže vyhovieť požiadavke okamžite.

**c)**

Ak by bankár vyhovel požiadavke zákazníka P[4], vznikol by stav daný nasledovnou tabuľkou:

	Allocation	Max.	(Need)	Available
P[0]	0	7	7	0
P[1]	2	3	1	
P[2]	3	9	6	
P[3]	2	2	0	
P[4]	3	4	1	

a úlohou bankára je opäť zistiť, či je tento stav bezpečný. Možné poradie vybavovania zákazníkov je napr.:

P[3], P[4], P[1], P[2], P[0].

Nový stav bude bezpečný, bankár teda môže vyhovieť požiadavke okamžite. Bezpečnosť stavu spočíva v tom, že P[3] už viac nič na svoju úspešnú existenciu nepotrebuje požičať, takže si spoločne počkáme až nám to vráti a potom vybavujeme ďalších. To je tiež rozdiel od reality: predpokladá sa, že kto si požičal, tak v rozumnom čase vráti. Inak by algoritmus nefungoval.

**2) Bankárov algoritmus – viac typov prostriedkov****Zadanie:**

Máme 5 zákazníkov (procesov) P[0] - P[4], 3 meny (3 typy prostriedkov) A, B, C, aktuálny stav je daný nasledovnou tabuľkou:

	Allocation	Max.	(Need)	Available
	A B C	A B C	A B C	A B C
P[0]	0 1 0	7 5 3	7 4 3	3 3 2
P[1]	2 0 0	3 2 2	1 2 2	
P[2]	3 0 2	9 0 2	6 0 0	
P[3]	2 1 1	2 2 2	0 1 1	
P[4]	0 0 2	4 3 3	4 3 1	

- Ide o bezpečný stav? (Dajte príklad v akom poradí môžu byť procesy vybavené.)
- Nech proces P[1] žiada prostriedky (1,0,2). Môže byť jeho požiadavka uspokojená okamžite?
- To isté ako b), ale tentoraz proces P[4] žiada (3,3,0).

**Riešenie:**

Proces musí mať súčasne všetky prostriedky (všetkých typov), ktoré požadoval, aby ich mohol použiť a v konečnom čase vrátiť. Riešenie je analogické ako v predošlom príklade, akurát je o stupeň zložitejšie, treba viac kombinovať. Pomôckou môže byť riešenie parciálne – vyhladáme všetky možné postupnosti len pre jeden typ prostriedku a potom z nich vyberáme tie, pri ktorých existuje vyhovujúca postupnosť pre druhý typ. A tak ďalej. Čiže vyriešenie úlohy pre jeden typ prostriedku je už vlastne tretina bodov za príklad.

**a)**

Poradie vybavovania procesov môže byť napr:

- P[1] - po vrátení pôžičky bude stav v banke (5, 3, 2)
- P[4] - po vrátení pôžičky bude stav v banke (5, 3, 4)
- P[3] - po vrátení pôžičky bude stav v banke (7, 4, 5)
- P[0] - po vrátení pôžičky bude stav v banke (7, 5, 5)
- P[2] - po vrátení pôžičky bude stav v banke (10, 5, 7)

**b)**

Ak by bankár vyhovel požiadavke procesu P[1], vznikol by nasledovný stav:

	Allocation	Max.	(Need)	Available
	A B C	A B C	A B C	A B C
P[0]	0 1 0	7 5 3	7 4 3	2 3 0
P[1]	3 0 2	3 2 2	0 2 0	
P[2]	3 0 2	9 0 2	6 0 0	



P[3] 2 1 1      2 2 2      0 1 1  
 P[4] 0 0 2      4 3 3      4 3 1

Tento stav je bezpečný, procesy môžu byť vybavené v poradí napr:  
 P[1], P[4], P[3], P[2], P[0].

c)

Ak by bankár vyhovel požiadavke procesu P[4], vznikol by nasledovný stav:

	Allocation	Max.	(Need)	Available
	A B C	A B C	A B C	A B C
P[0]	0 1 0	7 5 3	7 4 3	0 0 2
P[1]	2 0 0	3 2 2	1 2 2	
P[2]	3 0 2	9 0 2	6 0 0	
P[3]	2 1 1	2 2 2	0 1 1	
P[4]	3 3 2	4 3 3	1 0 1	

S kapitálom (0, 0, 2) nemôže bankár uspokojiť zostávajúcu potrebu žiadneho z procesov (mohol by vybaviť nejakú čiastkovú požiadavku, avšak žiadny z procesov nemôže získať všetky prostriedky, ktoré potrebuje). Tento stav nie je bezpečný, viedol by k uviaznutiu. Bankár teda nemôže vyhovieť požiadavke procesu P[4].

### 3) Plánovanie procesov s algoritmom Round Robin I.

#### Zadanie:

V systéme s preemptívnym plánovaním procesov, ktorý používa na plánovanie algoritmus **Round Robin**, sa naraz začnú vykonávať dva procesy A, B. Proces A potrebuje na svoje dokončenie 1 sekundu času procesora, proces B potrebuje 2 sekundy. K preplánovaniu dochádza každých 100 ms. Réžia na preplánovanie je 5 ms a je súčasťou kvanta. V systéme je len jeden procesor. V akom čase sa skončí vykonávanie každého z týchto procesov, keď sa obidva začnú vykonávať v čase 0 a okrem nich v systéme nie je žiadny iný proces? Uvažujte všetky možnosti. Postup pridelovania procesora procesom v čase graficky znázornite.

#### Riešenie:

Jedno časové kvantum trvá 100 ms a pozostáva z 5 ms rézie a 95 ms užitočného času pre činnosť procesu. Proces A potrebuje na svoje dokončenie 1000/95 časových kvánt a proces B potrebuje 2000/95 časových kvánt, čo je po zaokrúhlení na celé kvantá 11, resp. 22. Plánovanie typu Round Robin znamená, že sa procesy jednoducho striedajú a počas behu budú ich kvantá v nasledujúcom poradí:

**A B A B A B A B A B A B A B B B B B B B B B**

Proces A skončil po svojom 11. kvante, čo je celkovo po 21. kvante od začiatku. Proces B skončil po 33. kvante od začiatku. Ak máme uvažovať všetky možnosti, tak v úlohe nie je stanovené, ktorý proces bol naplánovaný ako prvý a druhá možnosť je potom:

**B A B A B A B A B A B A B A B B B B B B B B**

#### 4) Planovanie s nastavenými prioritami procesov A,B,C

##### Zadanie:

Statické priority procesov nech su nastavené takto: proces A má prioritu 1 (najnižšiu), proces B má prioritu 2 a proces C má prioritu 3 (najvyššiu). Nakreslite priebeh plánovania procesov pre dvanásť časových kvánt, pomocou uvedeného algoritmu plánovania podľa priority.

##### Riešenie:

Každému procesu pridáme plánovacie číslo, ktoré nastavíme podľa jeho priority. Ak použijeme algoritmus spomenutý vyššie v texte, tak pri každom časovom kvante dekrementujeme toto číslo a v ďalšom kvante spúšťame vždy ten proces, ktorý má číslo najvyššie. Ak sú čísla zhodné, rozhoduje algoritmus *Round Robin*. Keď čísla všetkých procesov dosiahnu hodnotu 0, tak ich znovu nastavíme na počiatočnú hodnotu a cyklus opakujeme. Opis algoritmu na papieri:

- 1) Podčiarkni najvyššie číslo v stĺpci, ak sú rovnaké tak z nich ďalšie v rade.
- 2) Zmenši podčiarknuté číslo o jednotku.
- 3) Ak je celý stĺpec vynulovaný, nastav čísla znovu na počiatočné hodnoty.

	1. cyklus						2. cyklus					
<b>A</b>	1	1	1	1	1	<u>1</u>	1	1	1	1	1	<u>1</u>
<b>B</b>	2	<u>2</u>	1	<u>1</u>	0	0	2	<u>2</u>	1	<u>1</u>	0	0
<b>C</b>	<u>3</u>	2	<u>2</u>	1	<u>1</u>	0	<u>3</u>	2	<u>2</u>	1	<u>1</u>	0

#### 5) Usporiadanie plánovaných úloh

##### Zadanie:

Naplánujte usporiadanie úloh po ich príchode, pričom zhromaždené úlohy vyžadujú 10, 6, 2, 4 a 8 minút čistého času na svoje vykonanie. Nie je žiadny paralelný multitasking, každá úloha sa vykoná bez prerušenia a až potom ďalšia. Aká bude priemerná odozva, ak použijeme algoritmus

- a) FCFS (First Come First Serve, v poradí príchodov)
- b) SJF (Shortest Job First, najkratšia úloha ako prvá)

##### Riešenie:

- a) úlohy sa končia v minúte 10. 16. 18. 22. 30. a priemer je 19,2
- b) priemer vychádza na 14 minút ukončenia každej úlohy od začiatku spoločného plánu.

## 6) Bankárov algoritmus – jeden typ prostriedkov II.

### Zadanie:

V systéme s celkovým počtom 17 prostriedkov sa vykonáva päť procesov, ktoré tieto prostriedky používajú. Procesy vopred deklarovali maximálny počet požadovaných prostriedkov. Aktuálny stav pridelenia je nasledujúci:

	Pridelených	Max. požadovaných
P1	3	8
P2	0	2
P3	3	11
P4	4	8
P5	3	4

Procesy postupne žiadajú o prostriedky nasledovne:

1. proces P4 žiada 2 prostriedky
2. proces P3 žiada 2 prostriedky
3. proces P5 žiada 1 prostriedok

Pomocou bankárovho algoritmu určte, v akom poradí môžu byť jednotlivé požiadavky vybavené. Uvažujte, že procesy prostriedky aj vracajú, a to v súlade s pravidlami bankárovho algoritmu. Svoju odpoveď samozrejme zdôvodnite.

### Riešenie:

Pridelených položiek je momentálne 13, deklarovaných je 33 čo je teda viac ako 17 = kapacita banky. P5 treba uprednostniť pred P3 (ak správne odpisujem z poznámok...), aby bol stále bezpečný stav.

## 7) Producent a konzument I.

### Zadanie:

Napíšte procedúry pre producenta a konzumenta správ, pričom v komunikačnom kanáli môže byť na ceste naraz najviac 6 správ. V komunikačnom kanáli v opačnom smere sa posielajú potvrdenia.

### Riešenie:

Nuž tak jeden z nich bude pravidelne vykonávať **send()** a druhý **receive()**. Opačným smerom to zavedieme tiež na posielanie potvrdení, pričom konzument na začiatku vyšle šesťkrát potvrdenie (alebo vstupenku do komunikácie). Pre producenta tieto potvrdenia čakajú v rade v komunikačnom kanáli na prečítanie a nikdy nepošle viac správ, než dostal potvrdení. *Nasledujúce príklady sú postavené presne naopak: procedúry sú už napísané a treba z nich vyčítať, aké vlastnosti bude mať komunikácia.*

## 8) Producent a konzument II.

### Zadanie:

Producent pri asynchrónnej komunikácii číta postupnosť znakov ABCDE. Uveďte všetky možnosti postupností znakov prijatých konzumentom

- a) po odoslaní znakov AB
- b) po odoslaní znakov ABCDE

```
#define COUNT 4

process producer(){
    int c, ack, n = 0;
    for(;;) {
        c = getchar();
        send(consumer, c, (n == 0) ? 1 : 0);
        if (c == EOF) break;
        if (n++ == COUNT) {
            receive(consumer, &ack);
            n = 0;
        }
    }
    receive(consumer, &ack);
}

process consumer ()
{
    int c, ack;
    for(;;) {
        receive(producer, &c, &ack);
        if (ack)      send(producer, ack);
        if (c == EOF) break;
        putchar(c);
    }
}
```

### Riešenie:

Rozpísať to do grafu komunikácie, ako postupujú udalosti v oboch programoch za sebou a kedy sú zviazané cez send/receive. Producent posiela naslepo “céčka”, teda znaky, až dovtedy, kým nenapočíta  $n = 4$ . Vtedy sa zastaví na receive, kde čaká kým nedostane nejaké potvrdenie a potom zase naslepo vyšle sériu “céčok”. Možnosti prijatých postupností: rozumie sa od najkratšej postupnosti (ešte neprijal nič) až po najdlhšiu možnú.

## 9) Producent a konzument III.

### Zadanie:

Producent pri asynchrónnej komunikácii číta postupnosť znakov ABCDE. Uveďte všetky možnosti postupností znakov prijatých konzumentom

- c) po odoslaní znakov ABC
- d) po odoslaní znakov ABCDE

```
#define COUNT 4

process producer(){
    int c, ack, n = 0;
    for(;;) {
        c = getchar();
        send(consumer, c, (n == 0) ? 1 : 0);
        if (c == EOF) break;
        if (++n == COUNT) {
            receive(consumer, &ack);
            n = 0;
        }
    }
    receive(consumer, &ack);
}

process consumer(){
    int c, ack;
    for(;;) {
        receive(producer, &c, &ack);
        if(ack) send(producer, ack);
        if(c == EOF) break;
        putchar(c);
    }
}
```

### Riešenie:

Zmenila sa pozícia operátora ++, viď manuál jazyka C.

## 10) Producent a konzument IV.

### Zadanie:

Napíšte procedúry pre producenta a konzumenta správ, ktorí si posielajú správy cez jeden obojsmerný komunikačný kanál s vlastnosťou rúry, pomocou jednoduchého volania funkcií v jazyku C. Funkcie deklarujte v tvare a formáte, aký potrebujete. V komunikačnom kanáli môžu byť na ceste ku konzumentovi najviac dve správy, na ceste k producentovi sa posielajú potvrdenia.

**Riešenie:**

Rúra je virtuálny súbor, do ktorého sa zapisuje na koniec a z ktorého sa číta od začiatku. Akonáhle sa čítanie dostane na aktuálny koniec súboru, musí sa prerušiť a čakať až tam niekto niečo pripíše. Tým možno synchronizovať udalosti. Do jednej rúry píšeme užitočné správy, do druhej rúry potvrdzovacie hocičo (napríklad znaky).

**11) Producent a konzument V.****Zadanie:**

Producent pri asynchrónnej komunikácii číta postupnosť znakov 1 2 3 4 5 6.

Uveďte všetky možnosti postupností znakov prijatých konzumentom

a) po odoslani znakov 1 2 3

b) po odoslani znakov 1 2 3 4 5 6

```
#define PREDSTIH 5
```

```
process producer ()
{
    int c, ack, n = 0;
    for(;;) {
        c = getchar();
        send(consumer, c, (n == 0) ? 1 : 0);
        if (c == EOF)
            break;
        if (++n == PREDSTIH) {
            receive(consumer, &ack);
            n = 0;
        }
    }
    receive(consumer, &ack);
}
```

```
process consumer ()
{
    int c, ack;
    for(;;) {
        receive(producer, &c, &ack);
        if (ack)
            send(producer, ack);
        if (c == EOF)
            break;
        putchar(c);
    }
}
```

**Riešenie:**

Iná verzia predošlých úloh.

## 12) Producent a konzument VI.

### Zadanie:

Aký počet znakov môže byť prijatý konzumentom po odoslaní ôsmich znakov producentom? Uveďte všetky možnosti a odpoved'zdôvodnite.

```
#define PREDSTIH 3

process producer ()
{
    int c, ack, n = 0;
    for(;;) {
        c = getchar();
        send(consumer, c, (n == 0) ? 1 : 0);
        if (c == EOF)
            break;
        if (++n == PREDSTIH) {
            receive(consumer, &ack);
            n = 0;
        }
    }
    receive(consumer, &ack);
}

process consumer ()
{
    int c, ack;
    for(;;) {
        receive(producer, &c, &ack);
        if (ack)
            send(producer, ack);
        if (c == EOF)
            break;
        putchar(c);
    }
}
```

### Riešenie:

Iná verzia predošlých úloh.

## 13) Bankárov algoritmus – viac typov prostriedkov II.

### Zadanie:

V systéme s troma druhmi prostriedkov sa vykonáva päť procesov, ktoré prostriedky používajú. Procesy vopred deklarovali maximálny počet požadovaných prostriedkov. Aktuálny stav pridelenia je nasledujúci:

	Pridelených	Max. požadovaných	Celkovo v systéme
	A B C	A B C	A B C
P[0]	3 0 1	8 3 2	17 9 6
P[1]	0 3 1	1 8 3	
P[2]	3 2 0	11 4 1	
P[3]	4 0 2	8 0 4	
P[4]	3 1 0	4 1 1	

- a) Pomocou bankárovho algoritmu určte, či je stav bezpečný alebo nie.  
 b) Môže byť ihneď vybavená požiadavka procesu P2 na pridelenie prostriedkov (3 1 0)?

**Riešenie:**

Iná verzia predošlých úloh.

**14) Bankárov algoritmus – viac typov prostriedkov III.****Zadanie:**

V systéme so štyrmi druhmi prostriedkov sa vykonáva päť procesov, ktoré prostriedky používajú. Procesy vopred deklarovali maximálny počet požadovaných prostriedkov. Aktuálny stav pridelenia je nasledujúci:

	Pridelených	Max. požadovaných	Celkovo v systéme
	A B C D	A B C D	A B C D
P[0]	0 0 1 2	0 0 1 2	3 14 12 12
P[1]	1 0 0 0	1 7 5 0	
P[2]	1 3 5 4	2 3 5 6	
P[3]	0 6 3 2	0 6 5 2	
P[4]	0 0 1 4	0 6 5 6	

- a) Pomocou bankárovho algoritmu určte, či je stav bezpečný alebo nie.  
 b) Môže byť ihneď vybavená požiadavka procesu P2 na pridelenie prostriedkov (1 0 0 2)?

**Riešenie:**

Iná verzia predošlých úloh.



**15) Bankárov algoritmus – viac typov prostriedkov III.****Zadanie:**

V systéme je kapacita 20 prideliteľných prostriedkov typu X a 10 prideliteľných prostriedkov typu Y. Momentálny stav pridelenia prostriedkov je vyznačený v nasledujúcej tabuľke a pridelovanie je kontrolované pomocou Bankárovho algoritmu.

proces	počet pridelených prostriedkov X	maximálne požadovaných prostriedkov X	počet pridelených prostriedkov Y	maximálne požadovaných prostriedkov Y
A	9	10	5	5
B	5	5	1	2
C	0	5	1	1
D	3	7	3	3

Aké požiadavky a ktorých procesov prichádzajú do úvahy na obslúženie v nasledujúcom jednom kroku, aby nenastal nebezpečný stav?

Akou postupnosťou vybavovania požiadaviek je možné bezpečne ukončiť celú činnosť procesov? Po obdržaní všetkých maximálne požadovaných prostriedkov každý proces skončí

**Riešenie:**

Možno začať len pridelením X prostriedku pre proces A. Potom po navrátení jeho požičaných prostriedkov už je viac možností.

**16) Producent a konzument VII.****Zadanie:**

Správy sa vysielajú neblokujúcim volaním **send()** a prijímajú blokujúcim volaním **receive()**. Aké rôzne postupnosti bude produkovať na výstupe proces **Konzument2** ?

```
Producent1() {
    for(int i=0;i<2;i++) send('A');
}
```

```
Producent2() {  
    for(int i=0;i<1;i++) send('B');  
}  
  
Konzument1() {char x;  
    for(int i=0;i<3;i++) { receive(&x); printf("%c", x); }  
}  
  
Konzument2() {char x;  
    for(int i=0;i<3;i++) { receive(&x); printf("%c", x); }  
}
```

**Riešenie:**

Neblokujúce – môžeme ho ihneď vykonať v akom počte chceme, blokujúce – čakáme až, naozaj niečo dostaneme.

Možné vyslané postupnosti: AAB, ABA, BAA

Možné prijaté postupnosti: nič, A, B, AB, BA, AA, AAB, ABA, BAA  
(máme na mysli všetky okamihy histórie prenosu, od začiatku až po úplné dokončenie)

**17) Plánovanie procesov s algoritmom Round Robin II.****Zadanie:**

V systéme s preemptívnym plánovaním procesov, ktorý používa na plánovanie algoritmus **Round Robin**, sa naraz začnú vykonávať dva procesy A, B. Proces A potrebuje na svoje dokončenie 1 sekundu času procesora, proces B potrebuje 2 sekundy. Počas behu každý z procesov čaká celkovo 1 sekundu na vstupno/výstupné operácie. K preplánovaniu dochádza každých 100 ms. Réžia na preplánovanie je 5 ms a je súčasťou kvanta. V systéme je len jeden procesor. Okrem procesov A a B v systéme nie je žiadny iný proces. Ako prvý sa začne vykonávať proces A.

Koľkokrát najmenej a koľkokrát najviac počas svojej existencie zmení proces A svoj stav na **BEZIACI** a **PRIPRAVENÝ**? Riešenie zdôvodnite.

**Riešenie:**

Je potrebné nakresliť si plánovanie – prepínanie medzi procesmi – pre extrémne prípady. Jeden extrém je, keď naraz celú sekundu proces A čaká na vstupno/výstupnú operáciu, vtedy môže nerušene bežať v každom kvante proces B. Sekundové čakanie sa môže udiť na začiatku alebo na konci behu každého procesu. Z toho sú štyri základné možnosti priebehu histórie.



```

Producent1() {
    for(int i=0;i<2;i++) write('A');
}

Producent2() {
    for(int i=0;i<1;i++) write('B');
}

Konzument1() {char x; read(&x); }
}

Konzument2() {char x;
    for(int i=0;i<3;i++) { read(&x); printf("%c", x); }
}

```

**Riešenie:**

Rúra je FIFO, do ktorého sa postupne v rôznej kombinácii (tri možnosti) môžu umiestniť položky A,B. Konzument1 môže jeden z vložených znakov stihnúť “ukradnúť” pred nosom druhého konzumenta. Prvý, druhý, alebo posledný.

**20) Rozmnožovací proces II.****Zadanie:**

V systéme beží na začiatku jeden proces, ktorý sa postupne rozmnoží. Každý proces vždy v jednom kvante stihne vykonať práve jeden príkaz (teda jeden riadok) zo svojho zdrojového textu. Procesy nečakajú a sú plánované metódou Round Robin. Premenná **x** je spoločne zdieľaná. Obsah procesu je nasledujúci:

**riadok      zdrojový text**

```

1      int x=0;
2      main() { while(1) { if(x > 1) break;
3                                x++;
4                                fork();
5                                print(x); } }

```

1. Určte postupnosť čísel vypísaných na spoločnom výstupe.
2. Znázornite časovú následnosť vykonávania riadkov všetkých vzniknutých procesov.
3. Uveďte, po ktorých časových kvantách sa zmení počet procesov v systéme a koľko ich vtedy je.

**Riešenie:**

Ako predošlý príklad 17, nám vyšlo pri písaní do vhodnej tabuľky že prvých 5 kvánt beží len prvý proces, ďalších 5 kvánt sa prepínajú medzi sebou dva, potom jedno kvantum vznik tretieho, potom šesť kvánt sa prepínajú štyri procesy a potom každé ďalšie kvantum

je o jeden menej. Round Robin chápeme tak, že sa chodí dookola radu, na ktorého koniec vždy pridáme ďalší novovzniknutý proces.

## 21) Producent a konzument VIII.

### Zadanie:

Správy sa vysielajú neblokujúcim volaním `send()` a prijímajú blokujúcim volaním `receive()`. Môže byť výstupom konzumenta reťazec „OS=...“ ? Ak áno, uveďte postupnosť operácií, ktorá na to vedie. Graficky znázorníte prepínanie procesov, ktoré vedie k tomuto výstupu.

```
Producent1() {  
    char *t = "KLMNOP";  
    for(int i=0; i<6; i++) send(*t++);  
}
```

```
Producent2() {  
    char *t = "PQRSTU";  
    for(int i=6, t+=i; i>0; i--) send(*--t);  
}
```

```
Producent3() {  
    char *t = "...==...";  
    for(int i=6, t+=i; i>0; i--) send(*--t);  
}
```

```
KonzumentA() {char x;  
    for(int i=0; i<25; i++) { receive(&x); }  
}
```

```
KonzumentB() {char x;  
    for(int i=0; i<6; i++) { receive(&x); printf("%c", x); }  
}
```

### Riešenie:

Môže byť, vieme nakresliť takú postupnosť odosielania správ, po ktorej sa prečíta OS=...

## 22) Plánovanie procesov s algoritmom Round Robin II.

### Zadanie:

V systéme bežia tri procesy. Veľkosť kvanta je 12 milisekúnd, režia sú 2 milisekundy bez prepnutia kontextu procesu a 4 milisekundy v prípade, že sa prepína aj kontext. Procesy A,B,C potrebujú pre svoje dokončenie A = 100 milisekúnd, B = 50 milisekúnd, C = 30 milisekúnd. Proces A je spustený v prvom kvante, proces B v piatom kvante a proces C v šiestom kvante od začiatku práce systému.

Aké sú celkové režijné náklady pre vykonanie procesu A, ak sa s ostatnými strieda pri algoritme plánovania Round Robin? Pribeh procesov graficky znázornite.

### Riešenie:

Proces A podľa nášho riešenia bežal 12 kvánt (prvé štyri kvantá osamote, potom prestriedaný druhými) a náklady by mali byť 38 milisekúnd.

## 23) Komunikujúce procesy

### Zadanie:

<pre> Proces1 for(i=0; i&lt;3; i++) {     send('P');    // 1a     semafor_up(); // 1b     send('R');    // 1c }         </pre>	<pre> Proces2 for(k=0; k&lt;4 ; k++) {     semafor_down();// 2x     z=receive();    // 2y     printf("%c",z); // 2z }         </pre>
--	--

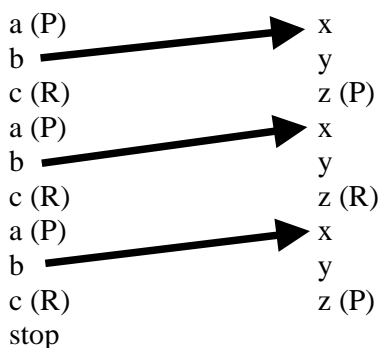
Dva paralelne bežiacie procesy zdieľajú jeden semafor, inicializovaný na hodnotu nula. Používajú jeden komunikačný kanál na vysielanie (send) a blokujúce prijímanie (receive), po jednom znaku.

- aká postupnosť znakov bude v komunikačnom kanáli na konci činnosti tejto dvojice procesov?
- aká postupnosť znakov sa vytlačí na výstupe?
- Aký bude stav semafora na konci činnosti dvojice procesov?

### Riešenie:

RPR ... zostane v kanáli  
 PRP ... bude prijaté a vytlačené  
 Stav semaforu na konci bude 0

Grafické vyjadrenie riešenia, postupnosť zhora nadol, šípka znamená riadenie semaforom:



**24) Komunikujúce procesy****Zadanie:**

```

while (1) {
    i = fork();                // A
    send('x');                 // B
    if (i==som_rodic)
    {
        send('y');            // C
        exit();               // D
    }
    else
        (void)receive();      // E
}

```

V systéme je jeden komunikačný kanál na vysielanie (send) a blokujúce prijímanie (receive), po jednom znaku. Plánovanie procesov prebieha algoritmom Round Robin. Jedno časové kvantum činnosti procesu je označené písmenom A-E

Aká postupnosť sa vykoná počas prvých deviatich kvánt?

Aká postupnosť znakov bude v komunikačnom kanáli

po deviatom kvante činnosti?

Koľko procesov bude v systéme aktívnych po deviatom kvante činnosti?

**Riešenie:**

Verzia 1:

1.	2.	3.	zoznam procesov
A			
B	B		...prepínanie zľava doprava a potom odznova
C	E		
D	A		
-	B	B	

Verzia 2:

1.	2.	3.	zoznam procesov
A	B		
B	E		
C	A	B	
D	B		

V oboch verziách: dva aktívne procesy zostanú po 9. kvante, v komunikačnom kanáli zostane „trčať“ postupnosť „XXYX“ a jeden „X“ bol prijatý a vytlačený.

## Správa pamäti

### 1) Fragment programu I.

#### Zadanie:

Fragment programu:

```
for(i=0; i < n; i++) A[i] = B[i] + C[i];
```

je po skompilovaní na počítači s registrami procesora R1, ..., R8 umiestnený vo virtuálnom adresovom priestore nasledovne (Nech  $n = 1024$ ):

Adresa	Inštrukcia	Komentár
0x0040	(R1) <- ZERO	R1 bude register pre i
0x0041	(R2) <- n	R2 bude register pre n
0x0042	compare R1,R2	porovnanie hodnôt i a n
0x0043	branch if gr or eq 0x0049	ak $i \geq n$ choď na 0x0049
0x0044	(R3) <- B(R1)	do R3 i-ty prvok poľa B
0x0045	(R3) <- (R3) + C(R1)	pričítaj C[i]
0x0046	A(R1) <- (R3)	súčet daj do A[i]
0x0047	(R1) <- (R1) + ONE	inkrementuj i
0x0048	branch 0x0042	choď na 0x0042
...		
...		
0x1800 .. 0x1BFF	storage for A	
0x1C00 .. 0x1FFF	storage for B	
0x2000 .. 0x23FF	storage for C	
0x2400	storage for ONE	
0x2401	storage for ZERO	
0x2402	storage for n	

Veľkosť stránky pamäti je 1k. Proces má pridelené 4 stránkové rámy. Koľko výpadkov stránok nastane počas behu fragmentu s použitím algoritmu výberu obete

- a) LRU                      b) FIFO                      c) Optimálny

Pred začatím vykonávania uvedeného fragmentu sa nepracuje so stránkami, ktoré tento fragment používa.

#### Riešenie:

1. adresový priestor 0x0040 až 0x2402 je potrebné rozdeliť na intervaly po jedno kilo (efektívne 1024), čo je dve na desiatu. Tým získame postupnosť stránok v



- pamäti a na každej stránke (stránky si očísľujeme) budeme vedieť, či je tam program, premenné, alebo tie údajové polia
2. pozrieme sa ako beží program: prečítanie inštrukcie je zrejme jasne idúce z prvej stránky, prečítaná inštrukcia vykoná čítanie **ZERO** ktoré leží zrejme na poslednej stránke, potom sa zase číta inštrukcia z prvej stránky a potom z poslednej stránky premenná **n**. Tretia inštrukcia nemá prístup do pamäti, je to iba činnosť s registrami v procesore. Atakďalej. Výsledkom ktorý potrebujeme je postupnosť čítaní stránok pri činnosti programu – postupnosť čísiel stránok.
  3. Kreslíme si tabuľku – v prvom momente sú štyri stránkové rámy v pamäti prázdne, prvá sa tam nasťahuje stránka s programom, potom stránka s premennými, atď - stačí cvičiť už len tú postupnosť čísiel a postupovať podľa určeného algoritmu na výber obete, spočítavať výpadky. (nasledujúci príklad 18 je vlastne len o tomto treťom kroku, bez prvých dvoch)

## 2) Multiprogramovanie s oblast'ami pevnej dĺžky, bez výmen - swapovania

### Zadanie:

Počítačový systém má v hlavnej pamäti priestor pre štyri programy. Tieto programy v priemere 50% času čakajú na V/V operácie. Aká časť času procesora (CPU) je v priemere nevyužitá?

### Riešenie:

Počas vykonávania každého programu sa strieda samotný výpočet (procesor vykonáva inštrukcie programu) s čakaním na V/V. Systém je bez swapovania, programy sú umiestnené v pamäti od začiatku do konca svojho behu, t.j. aj počas V/V operácii. Keby bol v pamäti len jeden program, bolo by nevyužitie v priemere 50% jeho času. V prípade štyroch programov je to pravdepodobnosť, že všetky štyri programy čakajú na V/V operácie =  $p^n$  (p na n-tú, pn) =  $0.5^4 = 1 - 0.9375 = 6.25\%$ .

## 3) Multiprogramovanie s oblast'ami pevnej dĺžky, bez výmen - swapovania

### Zadanie:

Nech počítač má 2MB pamäte, operačný systém zaberá 512kB a aj každý bežiaci program zaberá 512 kB. Ak všetky programy trávia 60% času čakaním na V/V, o koľko percent by sme zvýšili priepustnosť (využitie CPU) pridaním 1MB pamäte?

Koľko musíme pridať pamäte, aby sme zvýšili priepustnosť aspoň na 99% ?

### Riešenie:

Postup podobne ako v príklade 1: v prípade 2MB môžu byť v pamäti 3 programy, nevyužitý čas bude  $(0.6)^3 = 21.6\%$ . Ak pridáme 1 MB, môžeme do pamäti umiestniť 5 programov a nevyužitý čas bude  $(0.6)^5 = 7.77\%$ , t.j. dosiahli sme zvýšenie o 13.8%.

#### 4) Multiprogramovanie so swapovaním

**Zadanie:**

Niektoré systémy so swapovaním sa snažia eliminovať externú fragmentáciu pomocou kondenzácie (kompakcie) pamäti. Nech systém s 1MB používateľskej pamäti robí kondenzácie raz za sekundu. Nech prenos (copy) 1 bytu trvá 0.5 mikrosek. a priemerná dĺžka voľného úseku je 0.4 krát veľkosť priemerného segmentu. Aká časť celkového času CPU je použitá na kondenzácie?

Ako často treba robiť kondenzáciu, aby sa nespotrebovalo viac ako 10% času CPU ?

**Riešenie:**

Najskôr zistíme, koľko bytov pamäti (v priemere) sa preniesie pri každej kondenzácii: V pamäti sa striedajú voľné a obsadené úseky. Pri kondenzácii sa preniesú všetky obsadené úseky (segmenty). Ich celkovú dĺžku  $d$  zistíme nasledovne:  $d + 0.4 \cdot d = 1 \text{ MB}$ , t.j.  $d = 1/1.4 \text{ MB}$ . Prenos tejto pamäti bude trvať  $d \cdot 0.5 \text{ mikrosek.} = 1/2.8 \text{ sekundy} = 0.357 \text{ sec.}$  Teda ak sa kondenzácia robí raz za sekundu, zaberá 36% času procesora !!!

#### 5) Virtuálna pamäť so stránkovaním I.

**Zadanie:**

Nech vykonanie jednej inštrukcie trvá 1 mikrosek., ale v prípade odvolávky sa na neprítomnú stránku pamäti (page fault) ďalších  $n$  mikrosek. Aký je efektívny (priemerný) čas vykonávania jednej inštrukcie, ak sa page fault vyskytuje priemerne každých  $k$  inštrukcií?

**Riešenie:**

$k$  inštrukcií trvá  $k+n$  mikrosek, t.j. priemerný čas vykonania jednej inštrukcie je  $(k+n)/k = 1+n/k$  mikrosek.

#### 6) Virtuálna pamäť so stránkovaním II.

**Zadanie:**

Logický adresový priestor každého procesu má 8 stránok po 1024 slov a mapuje sa do fyzickej pamäti s 32 stránkovými rámami.

a) Koľko bitov má logická adresa?

b) Koľko bitov má fyzická adresa?

**Riešenie:**

Adresa = adresa stránky + offset v stránke. Pre adresovanie v rámci stránky príp. stránkového rámu o veľkosti 1024 slov potrebujeme offset dĺžky 10 bitov.

a) V logickom adresovom priestore potrebujeme na adresáciu 8 stránok 3 bity, t.j. logická adresa má dĺžku 13 bitov.

b) Vo fyzickom adresovom priestore potrebujeme na adresáciu 32 stránkových rámov 5 bitov, t.j. fyzická adresa má dĺžku 15 bitov.

Pozn. Väčšinou je logický adresový priestor väčší (alebo aspoň rovnaký) ako fyzická pamäť. V tomto prípade nie. Dá sa o tom diskutovať, prečo.

**7) Oneskorenie spôsobené kopírovaním tabuľky stránok****Zadanie:**

Počítač má 32-bitový adresový priestor a 8kB stránky. Tabuľka stránok pre práve bežiaci proces je v hardvéri a každá jej položka má 32 bitov. Keď proces štartuje (alebo sa prepína), tabuľka stránok sa kopíruje z pamäti do hardvéru rýchlosťou 1 položka/100 nsec. Ak každý proces beží 100 msec (vrátane naplňovania tabuľky), akú časť času CPU bude zaberat' naplňovanie?

**Riešenie:**

Stránke veľkosti 8kB zodpovedá offset dĺžky 13 bitov. Zostávajúcich 19 bitov adresy slúži na adresovanie v rámci tabuľky stránok, ktorej dĺžka je  $2^{19}$  položiek. Naplnenie tejto tabuľky trvá  $(2^{19}) \cdot (10^{-7}) \text{ sek.} = 52,43 \text{ msec}$  (po zaokruhlení). Ak každý proces beží 100 msec vrátane naplňovania tabuľky stránok, týmto naplňaním sa spotrebuje 52% času procesora !!!

**8) Dvojúrovňové tabuľky stránok****Zadanie:**

Počítač s 32-bitovými adresami používa dvojúrovňové tabuľky stránok (ako napr. 80386):

**9 bitov    11 bitov    offset**

1. Aké veľké sú stránky a koľko ich je vo virtuálnom adresovom priestore jedného procesu?
2. Akú časť adresového priestoru je možné adresovať pomocou jednej položky stránkového adresára, aká je s tým spojená réžia?
3. Aká by bola réžia na celý adresný priestor a pri jednoúrovňovej tabuľke stránok?

**Riešenie:**

V tomto prípade je adresa rozdelená na 3 časti: najvyššia časť (9 bitov) predstavuje adresu do adresára tabuliek stránok. Adresár tabuliek stránok je tabuľka prvej úrovne, ktorá obsahuje adresy stránkových rámov, v ktorých sa nachádzajú tabuľky stránok druhej úrovne. Stredná časť adresy slúži na adresovanie v rámci tabuľky druhej úrovne. Tabuľky druhej úrovne obsahujú adresy stránkových rámov fyzickej pamäti v ktorých sa nachádzajú jednotlivé stránky virtuálneho adresového priestoru procesu (ako pri jednoúrovňovom stránkovaní). Offset je adresa v rámci jednej stránky (ako obvykle).

V príklade sa offset skladá z  $32-9-11=12$  bitov, čiže veľkosť stránky je 4kB. Adresár tabuliek stránok obsahuje  $2^9 = 512$  položiek (t.j. adresy 512 tabuliek druhej úrovne), každá tabuľka druhej úrovne adresuje  $2^{11}=2048$  stránok. Celkový adresový priestor procesu pozostáva z  $2^{20}$  stránok.

**9) Page faults I.****Zadanie:**

Zistilo sa, že počet inštrukcií programu medzi dvoma výpadkami stránky (page faults) je priamo úmerný počtu pridelených stránkových rámov (t.j. väčšia časť vo fyzickej pamäti = väčšie intervaly medzi výpadkami). Nech inštrukcia trvá normálne 1 mikrosek. a s výpadkom stránky 2001 mikrosek. Program trval 60 sec. a mal 15000 výpadkov. Ako dlho by trval výpočet s dvojnásobným počtom stránkových rámov?

**Riešenie:**

15000 výpadkov trvá  $2001 \cdot 15000$  mikrosek = asi 30 sec. Ďalších 30 sec (z celkových 60) sa vykonávali inštrukcie dĺžky 1 mikrosek, bolo ich teda 30 mil. Dvojnásobný počet stránkových rámov znamená zdvojnásobenie počtu inštrukcií vykonaných medzi dvoma výpadkami, teda frekvencia výskytu výpadku stránky bude polovičná, t.j. počas behu programu bude 7500 výpadkov. To trvá (aj s vykonaním inštrukcií, ktoré vyvolali výpadok)  $7500 \cdot 2001$  mikrosek, t.j. približne 15 sec., teda celkový čas výpočtu klesol zo 60 sec na približne 45 sec, t.j. na 75% pôvodnej hodnoty.

**10) Asociatívna pamäť (T.L.B.)****Zadanie:**

V počítači majú logické adresové priestory procesov 1024 stránok. Tabuľky stránok sú držané v pamäti. Čítanie slova z tabuľky trvá 500 nsec. Na zníženie tejto režie má počítač asociatívnu pamäť, ktorá drží 32 párov (virtuálna stránka, fyzický stránkový rám) a dokáže vyhľadať položku za 100 nsec. Aká úspešnosť asociatívnej pamäti (hit rate) je potrebná na redukciu priemernej doby vyhľadávania na 200 nsec?

**Riešenie:**

Na zrýchlenie prekladu lineárnej adresy stránky na fyzickú adresu stránkového rámu sa takmer u všetkých procesorov používa asociatívna pamäť. Asociatívna pamäť obsahuje položky v tvare (stránka, stránkový rám), pričom kľúčom pre vyhľadávanie je stránka. Adresovací hardvér hľadá adresu stránky vo všetkých položkách paralelne, čo je veľmi rýchle. Iba ak nenájde správnu položku, musí zisťovať adresu stránkového rámu pomocou tabuľky stránok, čo trvá oveľa dlhšie.

Hit rate = pravdepodobnosť, že adresa sa nachádza na takej stránke, ktorá je v asociatívnej pamäti. Nech AP je počet (úspešných) vyhľadání adresy cez asociatívnu pamäť, TS je počet vyhľadání skomplikovaných okľukou cez prácu s tabuľkou stránok. Priemerná doba jedného vyhľadávania bude (v nsec)

$$(100 \cdot AP + 500 \cdot TS) / (AP + TS) = 100 \cdot AP / (AP + TS) + 500 \cdot TS / (AP + TS)$$

Pri dostatočne veľkom počte prístupov bude

$$AP / (AP + TS) = \text{hit\_rate}$$

$$TS / (AP + TS) = 1 - \text{hit\_rate}$$

teda hľadajú úspešnosť asociatívnej pamäti získame riešením rovnice:

$$200 = 100 \cdot \text{hit\_rate} + 500 \cdot (1 - \text{hit\_rate})$$

Ak však berieme do úvahy, že hodnota 500 je len čítanie slova z tabuľky a umiestnenie do asociatívnej pamäti a potom musí ešte raz nastať prečítanie z takto občerstvenej asociatívnej pamäti, tak je potrebné počítať s hodnotou 600.

$$200 = 100 \cdot \text{hit\_rate} + 600 \cdot (1 - \text{hit\_rate})$$

**11) Pridelovanie pamäti pomocou „Buddy“ algoritmu I.****Zadanie:**

Pridelovanie pamäti sa deje pomocou deliaceho “Buddy” algoritmu. Pamäť má veľkosť 1024. Päť procesov obsadí pamäť postupne s požiadavkami na veľkosť priestoru 52, 100, 198, 40 a 132. Koľko pamäti zostáva voľnej a v akých veľkých úsekoch?

**Riešenie:**

Nakreslíme si salámu a delíme ju hladným klientom zásadne len na polovice. Prvý chcel 52, takže delíme 1024 na 512, potom na 256, potom na 128 a potom na 64. To už je vhodné pre toho prvého, ktorý chcel 52. Druhý chcel 100, pre neho ešte rozkrájame ten zvyšný kus 256. Atakďalej, treba to nakresliť.

## 12) Veľkosť tabuľky stránok

### Zadanie:

Pamäť je organizovaná stránovaním. Veľkosť stránky je 4 KB. Virtuálna pamäť má veľkosť 4 GB, fyzická pamäť má veľkosť 256 MB. Akú veľkosť potrebuje mať tabuľka stránok? Aká je pravdepodobnosť chyby stránkovania pri náhodne zvolenej virtuálnej adrese?

### Riešenie:

Virtuálna pamäť je veľká 4GB/4KB stránok. Fyzická pamäť je veľká 256/4 stránok. Tabuľka stránok nám hovorí, kde je ktorá stránka zo všetkých, čo dokážeme adresovať. Má teda toľko položiek, koľko stránok je virtuálnych. Chyba stránkovania nastane, ak si zvolíme stránku a nie je vo fyzickej pamäti. Pravdepodobnosť trafenía sa – ak niet iných zadáných predpokladov – je daná pomerom fyzickej kapacity pamäti voči všetkým stránkam, ktoré virtuálne používame. Ak teda napríklad by fyzická pamäť obsahovala len štvrtinu zo všetkých stránok, pravdepodobnosť netrafenia sa by bola 75% (0,75).

## 13) Schéma výpočtu adresy

### Zadanie:

Nakreslite schému výpočtu adresy pri adresovaní s použitím segmentácie a aj stránkovania.

### Riešenie:

Číslo segmentu krát jeho veľkosť + číslo stránky krát jej veľkosť + posunutie

## 14) Fragment programu II.

### Zadanie:

Fragment programu:

```
for(k = 0; k < n; k++)  
{  
    A[k] = B[k] + C[k];  
    D[k] = B[n-k] + A[k];  
}
```

je po skompilovaní na počítači s registrami procesora R1,... R8 umiestnený vo virtuálnom adresovom priestore nasledovne (všetky čísla vrátane prvkov polí su veľkosti 1 slabika):

Adresa	Inštrukcia	Komentár
0x0040	(R1) <- ZERO	R1 bude register pre k
0x0041	(R2) <- n	R2 bude register pre n
0x0042	compare R1,R2	porovnanie hodnôt k a n
0x0043	branch if gr or eq 0x004C	ak $k \geq n$ choď na 0x004C
0x0044	(R3) <- B(R1)	do R3 k-ty prvok pola B
0x0045	(R3) <- (R3) + C(R1)	pričítaj C[k]
0x0046	A(R1) <- (R3)	súčet daj do A[k]
0x0047	(R4) <- (R2) - (R1)	v R4 bude n-k
0x0048	(R5) <- B(R4) + (R3)	v R5 bude B[n-k] + A[k]
0x0049	D(R1) <- (R5)	súčet do D[k]
0x004A	(R1) <- (R1) + ONE	inkrementuj k
0x004B	branch 0x0042	
	...	
	...	
0x0C00..0x0FFF	storage for A	
0x1000..0x13FF	storage for B	
0x1400..0x17FF	storage for C	
0x1800..0x1BFF	storage for D	
0x1C00	storage for ONE	
0x1C01	storage for ZERO	
0x1C02	storage for n	

Nech  $n = 1$ .

Veľkosť stránky pamäti je 1k. Proces ma pridelené stránkové rámy číslo 0x8, 0x9, 0xA, 0xB. Pred začatím vykonávania uvedeného fragmentu su všetky tieto rámy voľné a postupne sa obsadzujú v poradí čísel (0x8, 0x9,...). Každý riadok tabuľky stránok obsahuje nasledovné údaje: R (referenced) bit, M (modified) bit, present/absent bit, číslo stránkového rámu. Nech R bity sa nulujú periodicky vždy po vykonaní štyroch inštrukcií a je použitý algoritmus výberu obete NRU (not recently used - dlho nepoužitá stránka). Napíšte, čo budú obsahovať riadky tabuľky stránok zodpovedajúce stránkam, ktoré sa používajú v tomto fragmente. Zaujímá nás stav po vykonaní inštrukcie na adrese 0x004A.

#### Riešenie:

Ako predošlý príklad 1, fyzická pamäť má takisto 4 stránkové rámy, akurát sa tu aj nazývajú adresami. Na rozdiel od predošlého príkladu však riešime len jeden beh programového cyklu, lebo skončíme na uvedenej inštrukcii, kde prezentujeme stav tabuľky. Tabuľku vyplníme podľa uvedeného návodu.

## 15) Pridelovanie pamäti pomocou „Buddy“ algoritmu II.

#### Zadanie:

Pridelovanie pamäti sa deje pomocou deliaceho „Buddy“ algoritmu a pridávajú sa len celé súvislé úseky. Pamäť má veľkosť 1024 položiek a požiadavky prichádzajú v poradí 20, 8,

500, 260 a 132 položiek. Sú všetky tieto požiadavky obslužené? Koľko pamäti zostane k dispozícii na alokáciu a v akých veľkých úsekoch? Koľko úsekov pamäti zostane nevyužitých kvôli fragmentácii?

**Riešenie:**

postup delenia:

1024

512 512

512 256 256

512 256 128 128

512 256 128 64 64

512 256 128 64 **32** 32                    alokácia 20

512 256 128 64 **32** 16 16

512 256 128 64 **32** 16 **8** 8            alokácia 8

**512** 256 128 64 **32** 16 **8** 8            alokácia 500

**512** 256 128 64 **32** 16 **8** 8            alokácia 260 nie je obslužená

**512** **256** 128 64 **32** 16 **8** 8            alokácia 132

k dispozícii sú voľné úseky 128, 64, 16, 8

požiadavka na 260 nebola obslužená

vnútorná fragmentácia:

$$(32 - 20) + (8 - 8) + (512 - 500) + (256 - 132) = 148$$

**16) Fragment programu III.****Zadanie:**

Fragment programu:

```
for(k = 0; k < n; k++)
{
    A[k] = B[k] + C[k];
    D[k] = B[n-k] + A[k];
}
```

je po skompilovaní na počítači s registrami procesora R1,... R8 umiestnený vo virtuálnom adresovom priestore nasledovne (všetky čísla vrátane prvkov polí su veľkosti 1 slabika):

Adresa	Inštrukcia	Komentár
0x0020	(R1) <- ZERO	R1 bude register pre k
0x0021	(R2) <- n	R2 bude register pre n
0x0022	compare R1,R2	porovnanie hodnôt k a n
0x0023	branch if gr or eq 0x002C	ak k >= n choď na 0x002C
0x0024	(R3) <- B(R1)	do R3 k-ty prvok pola B
0x0025	(R3) <- (R3) + C(R1)	pričítaj C[k]
0x0026	A(R1) <- (R3)	súčet daj do A[k]
0x0027	(R4) <- (R2) - (R1)	v R4 bude n-k
0x0028	(R5) <- B(R4) + (R3)	v R5 bude B[n-k] + A[k]



0x0029	D(R1) <- (R5)	súčet do D[k]
0x002A	(R1) <- (R1) + ONE	inkrementuj k
0x002B	branch 0x0022	
	...	
	...	
0x0A00..0x0BFF	storage for A	
0x0C00..0x0DFF	storage for B	
0x0E00..0x0FFF	storage for C	
0x1000..0x11FF	storage for D	
0x1200	storage for ONE	
0x1201	storage for ZERO	
0x1202	storage for n	

Nech  $n = 1$ .

Veľkosť stránky pamäti je 512. Proces ma pridelené stránkové rámy číslo 0x7, 0x8, 0x9, 0xA. Pred začatím vykonávania uvedeného fragmentu su všetky tieto rámy voľné a postupne sa obsadzujú v poradí čísel (0x7, 0x8,...). Každý riadok tabuľky stránok obsahuje nasledovné údaje: R (referenced) bit, M (modified) bit, present/absent bit, číslo stránkového rámu. Nech R bity sa nulujú periodicky vždy po vykonaní štyroch inštrukcií a je použitý algoritmus výberu obete NRU (not recently used - dlho nepoužitá stránka). Napíšte, čo budú obsahovať riadky tabuľky stránok zodpovedajúce stránkam, ktoré sa používajú v tomto fragmente. Zaujíma nás stav po vykonaní inštrukcie na adrese 0x002A.

#### Riešenie:

Ako príklad 14, len trochu iný program a situácia. Znovu si to kreslíme len pre jeden cyklus programu a potom skončíme.

### 17) Fragment programu IV.

#### Zadanie:

Fragment programu:

```
for(k = 0; k < n; k++) {
    A[k] = A[k] + B[k] + C[k];
}
```

je po skompilovaní na počítači s registrami procesora R1,... R8 umiestnený vo virtuálnom adresovom priestore nasledovne (všetky čísla vrátane prvkov polí su veľkosti 1 slabika):

Adresa	Inštrukcia	Komentár
0x0030	(R1) <- ZERO	R1 bude register pre k
0x0031	(R2) <- n	R2 bude register pre n
0x0032	compare R1,R2	porovnanie hodnôt k a n

0x0033	branch if gr or eq 0x003A	ak $k \geq n$ chod' na 0x003A
0x0034	(R3) <- A(R1)	do R3 k-ty prvok pola A
0x0035	(R3) <- (R3) + B(R1)	pričítaj B[k]
0x0036	(R3) <- (R3) + C(R1)	pričítaj C[k]
0x0037	A(R1) <- (R3)	súčet do A[k]
0x0038	(R1) <- (R1) + ONE	inkrementuj k
0x0039	branch 0x0032	chod' na 0x0032
	...	
	...	
0x1A00..0x1BFF	storage for A	
0x1C00..0x1DFF	storage for B	
0x1E00..0x1FFF	storage for C	
0x2000	storage for ONE	
0x2001	storage for ZERO	
0x2002	storage for n	

Nech  $n = 2$ .

Veľkosť stránky pamäti je 512 slabík. Proces ma pridelené stránkove rámy číslo 0x0A, 0x0E, 0x0F. Pred začatím vykonávania uvedeného fragmentu su všetky tieto rámy voľné a postupne sa obsadzujú v poradí čísel (0xA, 0xE,...). Pre výber obete sa používa algoritmus FIFO.

Po skončení behu fragmentu vypočítajte

- fyzické adresy, na ktorých sa budú nachádzať nasledovné premenne (prvky polí): A[100], B[200].
- fyzickú adresu, ktorá je na virtuálnej adrese 0x1A2C.

Uveďte čitateľne celý postup riešenia !

#### Riešenie:

- ak som to odpísal zo správneho šalátu... tak je odpoveď takáto: stránka s prvým polom bude na 0x1E64, stránka s druhým polom nebude v danom okamihu vo fyzickej pamäti. Je vidieť, že fragment beží dva cykly a fyzická pamäť má iba tri stránky. Sledujeme teda históriu ich obsadenosti a čím sú obsadené.
- Program a dáta existujú vo virtuálnom svete, virtuálna adresa 0x1A2C ukazuje do toho pola A, jeho stránka kde je vo fyzickej pamäti – tam niekde je aj momentálna fyzická adresa tej virtuálnej polohy.

## 18) Page faults II.

#### Zadanie:

Postupnosť 16 referencií stránok v pamäti je nasledujúca:

**5, 3, 1, 4, 5, 2, 6, 3, 2, 2, 5, 6, 2, 6.**

Koľko výpadkov stránky budeme pozorovať, keď má pracovná pamäť kapacitu 3 stránkové rámy, pri algoritme výberu obetovaných stránok typu **FIFO** a pri algoritme **LRU**?

#### Riešenie:

Riešenie je ako príklad 1, avšak prvé dva kroky nemusíme robiť, máme už pripravenú postupnosť volaní stránok.

Postupnosť referencií stránok je postupnosť požiadaviek na čítanie alebo zápis, kedy musí byť stránka prístupná v pracovnej pamäti. V prvom riadku sú výpadky stránok (jej neprítomnosť v pracovnej pamäti) označené hviezdíčkou. V druhom riadku je zľava doprava časová postupnosť referencií, v ďalších troch hrubo vyznačených riadkoch je vývoj obsahu pracovnej pamäte (ktoré stránky sú v nej prítomné) a v ostatných riadkoch je vývoj obsahu priestoru na odkladanie stránok mimo pracovnej pamäte. Pri algoritme **FIFO** si pomáhame formálnym posúvaním čísiel smerom dolu, pri algoritme **LRU** zaradíme číslo práve referencovanej stránky vždy navrch, ako poslednej referencovanej. Z uvedených dvoch schém je zrejmé, že pri tejto postupnosti referencií algoritmus FIFO spôsobuje o jeden výpadok stránky viac. Tam, kde sa postup algoritmov odlišuje, sú čísla podčiarknuté.

riešenie pre algoritmus **FIFO**:

```
výpadky stránok:      * * * * * * * *      * * *
postupnosť referencií: 5,3,1,4,5,2,6,3,2,2,5,6,2,6
stránky v pamäti:     5 3 1 4 5 2 6 3 3 3 5 5 2 6
                      5 3 1 4 5 2 6 6 6 3 3 5 2
                      5 3 1 4 5 2 2 2 6 6 3 5
stránky odložené:     5 3 1 4 5 5 5 2 2 6 3
                      3 1 4 4 4 4 4 4 4
                      3 1 1 1 1 1 1 1
```

riešenie pre algoritmus **LRU**:

```
výpadky stránok:      * * * * * * * *      * *
postupnosť referencií: 5,3,1,4,5,2,6,3,2,2,5,6,2,6
stránky v pamäti:     5 3 1 4 5 2 6 3 2 2 5 6 2 6
                      5 3 1 4 5 2 6 3 3 2 5 6 2
                      5 3 1 4 5 2 6 6 3 2 5 5
stránky odložené:     5 3 1 4 5 5 5 6 3 3 3
                      3 1 4 4 4 4 4 4 4
                      3 1 1 1 1 1 1 1
```

## 19) Fragmentácia

#### Zadanie:

Veľkosť stránky je 512 bajtov. V systéme sa vykonáva alokácia len v súvislých intervaloch adries. Proces vykoná nasledujúcu sekvenciu operácií:

```
A = malloc(1000);  
B = malloc(500);  
free(A);  
A = malloc(2000);  
C = malloc(1000);  
D = malloc(500);  
free(C);  
C = malloc(2000);
```

Na alokáciu pamäte sa používa algoritmus First Fit. Koľko bajtov zostane nevyužitých v rámci vnútornej fragmentácie a koľko v rámci vonkajšej fragmentácie stránok? Predpokladajte, že sa pamäť alokuje od adresy 0. Odpoveď zdôvodnite.

**Riešenie:**

Nie je tu spomenutý žiadny Buddy algoritmus. First Fit znamená, že sa pohybujeme od začiatku pamäťového priestoru a vezmeme prvý úsek, do ktorého sa vôjdeme a alokujeme (odrezávame zo salámy) postupne po sebe 1000, 500, 2000, 1000, 500, 2000. Medzitým sa uvoľnia dve medzery o veľkosti 1000 a tie zostanú nepoužiteľné, keďže potom je požiadavka na pridelenie úseku 2000, teda súvislého úseku a ten už nie je.

Salámu krájame najmenej po celých stránkach, takže neúplne zaplnené stránky to je interná fragmentácia. Externá fragmentácia to sú celé uvoľnené stránky.

**20) Page faults III.****Zadanie:**

Postupnosť 16 referencií stránok v pamäti je nasledujúca:

**5,7,1,4,5,2,6,3,2,2,5,6,2,8,8,5.**

Koľko výpadkov stránky budeme pozorovať, keď pamäť má kapacitu 3 stránky, pri algoritme výberu stránok typu

- a) FIFO
- b) LRU
- c) optimálny

Aká je tu potrebná veľkosť priestoru na odkladanie stránok?

**Riešenie:**

Kreslíme si tabuľku s tromi riadkami (tri stránky fyzickej pamäti) a do nej zapisujeme priebeh histórie volania a výmeny stránok. Ako predošlý príklad 18, iná verzia. Otázka o veľkosti priestoru na odkladanie stránok – koľko najviac stránok nám počas behu vypadne.

**21) Page faults IV.****Zadanie:**

Postupnosť 8 referencií stránok v pamäti je nasledujúca:

**5, 7, 1, 4, 4, 2, 7, 5**

Koľko výpadkov stránky budeme pozorovať, keď pamäť má kapacitu 2 stránky, pri algoritme výberu stránok typu FIFO? Aká je tu potrebná veľkosť priestoru na odkladanie stránok?

**Riešenie:**

Ďalšia verzia predošlého príkladu.

**22) Page faults V.****Zadanie:**

Postupnosť 16 referencií stránok v pamäti je nasledujúca:

**5, 3, 1, 4, 5, 2, 6, 3, 2, 2, 5, 6, 2, 5.**

Koľko výpadkov stránky budeme pozorovať, keď pamäť má kapacitu 3 stránky, pri algoritme výberu stránok FIFO a pri optimálnom algoritme výberu stránok? Aká je tu potrebná veľkosť priestoru na odkladanie stránok?

**Riešenie:**

Optimálny algoritmus je taký, keď **dopredu vieme celú** túto postupnosť (čo v praxi nie je dosť dobre možné) a výmenu stránok tomu prispôbujeme v každom kroku, aby bola čo najvhodnejšia.

## 23) Buddy alokácia

### Zadanie:

V systéme je veľkosť jednej stránky pamäte 16 a pre alokáciu je k dispozícii 8 stránok. Samotné stránky sú nedeliteľné a prideliť ich alokačný mechanizmus typu Buddy (metóda rekurzívneho delenia).

```
x = 1; while(1) { if(malloc(x) == NULL) break; x = x + 7; }
```

Odpovede na nasledujúce otázky vysvetlite a znázorníte schémou (kreslením obsahu ôsmich stránok):

Koľko alokačných operácií sa vykoná úspešne?

Aká bude na záver súhrnná veľkosť internej fragmentácie?

Aká bude na záver súhrnná veľkosť externej fragmentácie?

### Riešenie:

Úspešné alokačné požiadavky budú len tieto: 1, 8, 15, 22, 29.

Interná fragmentácia 37 (súčet 5 koncových kúskov), externá 16 (jedna stránka vcelku)

## 24) Buddy alokácia ešte inak

### Zadanie:

V systéme je veľkosť jednej stránky pamäte 16 a pre alokáciu je k dispozícii 8 stránok. Samotné stránky sú nedeliteľné a prideliť ich alokačný mechanizmus typu Buddy (metóda rekurzívneho delenia).

```
x = 1; while(1) { if(malloc(x) == NULL) break; x = x + 8; }
```

Odpovede na nasledujúce otázky vysvetlite a znázorníte schémou (kreslením obsahu ôsmich stránok):

Koľko alokačných operácií sa vykoná úspešne?

Aká bude na záver súhrnná veľkosť internej fragmentácie?

Aká bude na záver súhrnná veľkosť externej fragmentácie?

### Riešenie:

Úspešné alokačné požiadavky budú len tieto: 1, 9, 17, 25.

Interná fragmentácia 44, externá 2 x 16

## 25) Model virtuálnej pamäti

### Zadanie:

Fragment programu a jeho dáta sú vo virtuálnom adresovom priestore procesu umiestnené podľa doleuvedenej schémy. Veľkosť stránky je 1k a proces má vo fyzickej

pamäti pridelené len štyri stránkové rámy označené 0xA, 0xB, 0xC, 0xD. Na začiatku sú tieto rámy neobsadené a obsadzujú sa postupne v poradí A,B,C,D s využitím algoritmu výberu obete stránky typu NRU (Not Recently Used, dlho nepoužívaná stránka). Jeden záznam v tabuľke stránok pozostáva zo štyroch položiek: číslo stránky, bit M, bit R (pre uvedený algoritmus) a bit Absent (stránka nie je v pracovnej pamäti). Periodické nulovanie bitu R nastáva po dokončení každej tretej inštrukcie programu. R1 až R5 sú registre v procesore.

Stránka obsahujúca konštantu programu ZERO je nastavená ako rezidentná. Znázornite riešenie pre odpovede na nasledujúce otázky:

Aká bude postupnosť referencií stránok počas vykonávania prvých desiatich inštrukcií programu a koľko výpadkov stránky vtedy nastáva?

Aký bude stav kompletného obsahu tabuľky stránok po vykonaní prvých piatich inštrukcií programu?

Aký bude stav kompletného obsahu tabuľky stránok po vykonaní prvých desiatich inštrukcií programu?

0x0c00 ...0x0cff	pole A	(stranka 1)
0x1000 ...0x10ff	pole B	(stranka 2)
0x1400 ...0x14ff	pole C	atd...
0x1800 ...0x18ff	pole D	
0x1c00 ...0x1c01	cislo ZERO	program odtialto:
0x2000	(R1)<-ZERO	prenos
0x2001	(R2)<-A(R1)	prenos (R1: index v poli)
0x2002	(R3)<-B(R1)	prenos
0x2003	B(R1)<-(R2)	prenos
0x2004	(R4)<-C(R1)	prenos
0x2005	(R2)<-D(R1)	prenos
0x2006	(R5)<-(R2)+(R4)	sucet
0x2007	(R5)<-(R5)+(R3)	sucet
0x2008	(R2)<-(R5)+B(R1)	sucet
0x2009	branch 0x2abc	skok

### Riešenie:

Stránka rezidentná sa nevyhadzuje a natvrdo tam zostáva! Po každej tretej inštrukcii treba bity R vynulovať. Kresliť schému ako v predošlých podobných príkladoch a pomôcka grafická: bity R,M si do tabuľky značiť k číslu stránky ako maličké indexy<sub>R,M</sub> ak sú nastavené na 1, inak ak sú nula tak nenapísať.

Postupnosť referencií (čísla stránok od 1 zhora nadol po 6) vychádza takto:

6 5 6 1 6 2 (nuluj) 6 2<sub>M</sub> 6 3 6 4 (nuluj) 6 6 6 2 (nuluj) 6 - stránka 6 obsahuje inštrukciu, jej čítaním sa vždy začne

**26) Model virtuálnej pamäti ešte jeden****Zadanie:**

Fragment programu a jeho dáta sú vo virtuálnom adresovom priestore procesu umiestnené podľa doleuvedenej schémy. Veľkosť stránky je 1k a proces má vo fyzickej pamäti pridelené len štyri stránkové rámy označené 0xA, 0xB, 0xC, 0xD. Na začiatku sú tieto rámy neobsadené a obsadzujú sa postupne v poradí A,B,C,D s využitím algoritmu výberu obete stránky typu NRU (Not Recently Used, dlho nepoužívaná stránka). Jeden záznam v tabuľke stránok pozostáva zo štyroch položiek: číslo stránky, bit M, bit R (pre uvedený algoritmus) a bit Absent (stránka nie je v pracovnej pamäti). Periodické nulovanie bitu R nastáva po dokončení každej tretej inštrukcie programu. R1 až R5 sú registre v procesore.

Stránka obsahujúca konštantu programu ONE je nastavená ako rezidentná. Znázornite riešenie pre odpovede na nasledujúce otázky:

Aká bude postupnosť referencií stránok počas vykonávania prvých desiatich inštrukcií programu a koľko výpadkov stránky vtedy nastáva?

Aký bude stav kompletného obsahu tabuľky stránok po vykonaní prvých piatich inštrukcií programu?

Aký bude stav kompletného obsahu tabuľky stránok po vykonaní prvých desiatich inštrukcií programu?

program:		
0x2000	(R1)<-ONE	prenos
0x2001	(R2)<-K(R1)	prenos
0x2002	(R3)<-L(R1)	prenos
0x2003	L(R1)<-(R2)	prenos
0x2004	(R4)<-M(R1)	prenos
0x2005	(R2)<-N(R1)	prenos
0x2006	(R5)<-(R2)+(R4)	sucet
0x2007	(R5)<-(R5)+(R3)	sucet
0x2008	(R2)<-(R5)+L(R1)	sucet
0x2009	branch 0x2abc	skok
0x2c00 ...0x2cff	pole K	(stránka 2)
0x3000 ...0x30ff	pole L	
0x3400 ...0x34ff	pole M	
0x3800 ...0x38ff	pole N	
0x3c00 ...0x3c01	cislo ONE	(stránka 6)

**Riešenie:**

Postup ako predošlý príklad, len je to inak otočené.

Postupnosť referencií (čísla stránok od 1 zhora nadol po 6) vychádza takto:

1 6 1 2 1 3 (nuluj) 1 3 1 4 1 5 (nuluj) 1 1 1 3 (nuluj) 1



**27) Model virtuálnej pamäti a ešte jeden****Zadanie:**

číslo stránky (virtuálna pamäť)	bit R	bit M	číslo stránkového rámu
5	0	1	3
2	0	0	2
3	0	0	1
6	0	0	4

Uvedený obsah tabuľky stránok je pre systém s kapacitou štyri stránky vo fyzickej pamäti. Algoritmus výberu obetovanej stránky je typu NRU (*Not Recently Used*, dlho nepoužívaná stránka), využívajúci bit R (referencovaná stránka) a bit M (modifikovaná stránka).

Od okamihu daného stavu tabuľky stránok sa vykoná nasledujúca postupnosť referencií virtuálnych stránok, obsahy stránok sú iba čítané a po štvrtom referencovaní nastane nulovanie bitu R:

2      3      4      5      3      2      6

Aká bude postupnosť volania stránkových rámov fyzickej pamäti?

Aký bude obsah tabuľky stránok na konci vykonávania postupnosti?

**Riešenie:**

Postupnosť volania bude:

2      1      4      3      1      2      4

Obsah tabuľky bude:

	R	M	
5	0	1	3
2	1	0	2
3	1	0	1
6	1	0	4

## Súborový systém

### 1) Vlastnosti súborového systému na diskoch

#### Zadanie:

Ako dlho bude trvať prečítanie 64 kB programu z disku s priemernou dobou vyhľadávania stopy 30 msec, časom rotácie 20 msec a veľkosťou stopy 32 kB

- a) pre veľkosť stránky 2kB
- b) pre veľkosť stránky 4kB

Stránky sú náhodne roztrúsené po disku.

#### Riešenie:

Prečítanie jednej stránky sa skladá z troch činností:

- vyhľadanie stopy = 30 msec
- otočenie disku, kým bude pod hlavičkou začiatok stránky = 10 msec
- prenos stránky

- a)  $(2\text{kB}/32\text{kB}) * 20\text{msec} = 5/4 \text{ msec}$
- b)  $(4\text{kB}/32\text{kB}) * 20\text{msec} = 5/2 \text{ msec}$

Prečítanie celého programu:

- a) 32 stránok, každá  $30+10+5/4 \text{ msec} = 40 \times 33 \text{ msec}$
- b) 16 stránok, každá  $30+10+5/2 \text{ msec} = 40 \times 17 \text{ msec}$

### 2) Súbor na disku I.

#### Zadanie:

Malý (veľmi malý) disk má jeden povrch, 40 stôp a 9 sektorov na stopu. Na tomto disku je uložený súbor, pričom priradenie fyzických sektorov logickým blokom je opísané reláciou

$T = \{ \langle 0,137 \rangle, \langle 1,45 \rangle, \langle 2,277 \rangle, \langle 3,211 \rangle, \langle 4,69 \rangle, \langle 5,109 \rangle, \langle 6,185 \rangle, \langle 7,226 \rangle \}$ .

Pre operáciu sekvenčného prečítania celého súboru z disku uveďte, v akom poradí sa budú čítať jednotlivé fyzické sektory, ak sa pre výber požiadavky používa politika „Shortest Seek First“ a na začiatku je čítacia hlavička nad stopou 21. Stopy, sektory a bloky sú číslované od 0.

**Riešenie:**

Bloky súboru 0,1,2,3,4,... sú na povrchu disku rozptýlené na pozíciách 137, 45, 277, 211,... pričom napríklad pozícia 137 je na 15. stope (každá stopa obsahuje 9 sektorov – pozícií pre bloky súboru). S hlavičkou ideme zo stopy 21 na najbližšiu kde sú sektory súboru a potom na ďalšiu najbližšiu, atď. Tie dvojice čísiel si teda treba na začiatku prepočítat' na ktorej stope ktorá leží.

**3) Evidencia voľných blokov****Zadanie:**

Disk s veľkosťou 10 MB má veľkosť bloku 1 KB. Na čísla blokov sú použité 2 B. Určte počet blokov potrebných na evidovanie voľných blokov pri prázdnom a pri plnom disku, keď sa na evidenciu voľných blokov používa

- a) bitová mapa
- b) zreťazené bloky indexov

Výpočet vysvetlite.

**Riešenie:**

Bitová mapa je reťazec bitov, kde 1/0 označujeme obsadenosť bloku, každý bit pre jeden blok. Pri plnom disku formálne nič, pri prázdnom je to počet bitov ako je počet blokov.

Zreťazený blok indexov je 1KB blok disku, do ktorého sú natlačené čísla blokov o veľkosti 2B, pričom posledné číslo nech teda ukazuje na ďalší blok s indexami. Pri plnom disku formálne zase nič nie je zapísané, pri celkom prázdnom tam budú nejaké bloky naplnené indexami všetkých blokov okrem seba.

**4) Súbor na disku II.****Zadanie:**

Disk má 1 povrch, 5 stôp a 10 sektorov na stopu. Na disku je uložený súbor, pričom priradenie fyzických sektorov logickým (blokom) je opísané reláciou

$$T = \{ \langle 0,13 \rangle, \langle 1,2 \rangle, \langle 2,18 \rangle, \langle 3,28 \rangle, \langle 4,15 \rangle, \langle 5,16 \rangle, \langle 6,39 \rangle, \langle 7,40 \rangle \}.$$

Používateľ požiada operačný systém o načítanie celého súboru do pamäti. Napíšte v akom poradí sa budú čítať jednotlivé fyzické sektory, ak sa pre výber požiadavky používa politika "Elevator" (výťah) a na začiatku je čítacia hlavička nad stopou 3. Stopy, sektory a bloky sú číslované od 0.

**Riešenie:**

Ako príklad súbor na disku I.

## 5) Stromový adresár a obsah i-uzlov

### Zadanie:

V súborovom systéme založenom na i-uzloch sa nachádzajú súbory `/os/foo/goal/exam.ps` a `/os/exam.tex`. Znázornite

- odpovedajúci stromový adresár
- obsah i-uzlov a adresárov potrebných na sprístupnenie súborov `exam.ps` a `exam.tex`

### Riešenie:

Kreslíme klasický obrázok adresárov a i-uzlov podľa horeuvedenej špecifikácie.

## 6) I-uzly a súbory I.

### Zadanie:

Súborový systém v operačnom systéme UNIX má i-uzol so štruktúrou päť priamych, dva nepriame, jeden dvojito nepriamy a jeden trojito nepriamy ukazovateľ. Veľkosť logického bloku je 256 slov, na reprezentáciu každého ukazovateľa sú potrebné 4 slová. Znázornite štruktúru i-uzla a umiestnite doňho informácie z nasledovnej tabuľky. Tabuľka obsahuje reláciu zobrazenia čísla logického bloku súboru na číslo logického sektora disku. Tabuľka obsahuje len sedem údajov, ostatné údaje o súbore nás v tomto príklade nezaujímajú.

Číslo logického bloku súboru	1	3	128	256	512	513	514
Číslo logického sektora disku	65523	256	1245	15265	15266	15267	15268

### Riešenie:

Kreslíme klasický obrázok adresárov a i-uzlov podľa horeuvedenej špecifikácie.

## 7) I-uzly a súbory II.

### Zadanie:

Predpokladajte súborový systém operačného systému UNIX, ktorého i-uzol obsahuje 2 priame, 1 nepriamy, 1 dvojito nepriamy a 1 trojito nepriamy ukazovateľ. Veľkosť fyzického sektora je 32 slabík. Na reprezentáciu ukazovateľa su potrebné 4 slabiky.

Znázorníte obsah i-uzla a príslušných použitých nepriamych blokov súboru, ktorého funkcia zobrazujúca číslo logického bloku súboru na číslo logického sektoru disku je popísaná reláciou

$$T = \{ \langle 0, 50 \rangle, \langle 1, 51 \rangle, \langle 5, 112 \rangle, \langle 6, 23 \rangle, \langle 22, 11 \rangle, \langle 500, 33 \rangle \}.$$
**Riešenie:**

Nakreslíme schému i-uzla podľa horeuvedenej špecifikácie a na pozície 50, 51, 112, 23... vyznačíme polohu blokov súboru. Predpokladá sa, že kreslič sa zaobíde bez nakreslenia 500 krabičiek, podarí sa mu to naznačiť tak schematicky zjednodušene ☺

**8) I-uzly a súbory III.****Zadanie:**

Predpokladajte súborový systém operačného systému UNIX, ktorého i-uzol obsahuje 1 priamy, 1 nepriamy, 1 dvojito nepriamy a 1 trojito nepriamy ukazovateľ. Veľkosť fyzického sektora je 16 slabík. Na reprezentáciu ukazovateľa su potrebné 4 slabiky.

Znázorníte obsah i-uzla a príslušných použitých nepriamych blokov súboru, ktorého funkcia zobrazujúca číslo logického bloku súboru na číslo logického sektoru disku je popísaná reláciou

$$T = \{ \langle 0, 25 \rangle, \langle 13, 50 \rangle, \langle 18, 51 \rangle, \langle 19, 112 \rangle, \langle 37, 23 \rangle, \langle 84, 11 \rangle \}.$$
**Riešenie:**

Ako predošlý príklad

**9) Priemerná veľkosť súboru****Zadanie:**

Na disku je 5000000 blokov. Na tomto disku je vytvorený súborový systém založený na i-uzloch, pričom tabuľka i-uzlov má po vytvorení pevnú veľkosť. Pri vytváraní súborového systému sa predpokladalo, že priemerná veľkosť súboru bude 8 blokov. Koľko miesta na disku určeného na dátové bloky by pribudlo, keby predpokladaná priemerná veľkosť súboru bola 16 blokov ?

Veľkosť i-uzla je 1/8 veľkosti bloku. Uvažujte, že na disku je len tabuľka i-uzlov a dátové bloky. Riešenie zdôvodnite.

**Riešenie:**

Je to optimalizačná úloha. Prvý interval blokov na disku je postupnosť blokov nesúca i-uzly, druhý interval je postupnosť samotných priamych blokov súborov (nepriamo ignorujeme – len dátové bloky). Jeden blok v tabuľke i-uzlov znamená, že kapacitne nesie

v sebe osem i-uzlov, teda obsluhuje osem súborov s priemernou veľkosťou 8 krát 8 blokov. Takže jeden blok v prvom intervale blokov kapacitne obsluží 64 blokov v druhom intervale blokov. Ak sa súbory zväčšia na dvojnásobok, tak tabuľka i-uzlov sa zmenší, pretože sa na disk vŕjde menej (väčších) súborov. O to viac bude blokov na súbory...

## 10) I-uzly a súbory IV.

### Zadanie:

Štruktúra i-uzla ukazuje na tri priame bloky, dva jednoducho nepriame a jeden dvojito nepriamy blok. Veľkosť bloku je 8 bajtov. Číslo bloku má 1 bajt.

1. Aká je maximálna veľkosť súboru v bajtoch ?
2. Nakreslite polohu všetkých blokov súboru na logickom disku, keď bol súbor vytvorený na prázdnom disku takto:

```
f = open("file", O_WRONLY);  
lseek(f, 150, SEEK_SET);  
write(f, "1234567890", 10);
```

### Riešenie:

Ak je potrebný 1 bajt na záznam čísla bloku a blok je veľký 8 bajtov, tak sa doňho vŕjde 8 indexov pre adresovanie blokov. Keď podľa čísiel príkladu upravíme schému na Obr. 4.4, tak i-uzol bude ukazovať:

- na **3 bloky** súboru priamo,
- na **16 blokov** súboru nepriamo, cez dva bloky indexov  $2 \cdot 8 = 16$ ,
- na **64 blokov** súboru dvojito nepriamo  $1 \cdot 8 \cdot 8 = 64$ .

Najväčší súbor zaberie  $3 + 16 + 64 = 83$  blokov údajov  $8 \cdot 83 = 664$  bajtov.

Plná štruktúra súboru obsahuje  $2 + 1 + 8 = 11$  blokov indexov (2 jednoducho nepriame a 1 dvojito nepriamy, ukazujúci na 8 blokov priamych indexov).

Súbor s maximálnou veľkosťou zaberie dohromady  $83 + 11 + 1$  **i-uzol** = 95 blokov.

## 11) I-uzly a súbory V.

### Zadanie:

Nech i-uzol adresuje 5 priamych blokov, dva dvojito nepriame a jeden trojito nepriamy. Veľkosť bloku je 64 kilobajtov. Na disku založíme nový adresár a doňho vložíme súbor o veľkosti 512 kilobajtov. Koľko blokov na disku bude obsadených pre tento nový adresár a súbor v ňom? Výpočet sprevádzajte schémou vzťahov medzi blokmi a i-uzlami.

### Riešenie:

Kreslíme klasickú schému i-uzlov a adresárov, pričom adresár je súbor a teda potrebuje i-uzol. A na začiatku máme koreňový adresár, ktorý už existuje a tiež je súbor.

## 12) I-uzly a súbory VI.

### Zadanie:

Štruktúra i-uzla obsahuje ukazovatele na tri priame bloky, dva jednoducho nepriame a jeden dvojito nepriamy blok. Veľkosť bloku je 8 bajtov. Číslo bloku má 1 bajt. Povolené meno suboru je jednopísmenové. V bežiacom systéme súborov založíme svoj adresár "T" a doňho vložíme súbor "X" o veľkosti 40 bajtov.

1. Graficky znázorníte, ako sú umiestnené štruktúry súborového systému v jednotlivých blokoch na disku. V znázornení uveďte všetky relevantné údaje.
2. Koľko kópií súboru "X" môžeme v adresári "T" vytvoriť, ak má disk veľkosť 50 blokov?
3. Po vytvorení všetkých možných kópií súboru "X" bude jeden náhodne zvolený blok na disku uvedenej veľkosti prepísaný chybne tak, že všetky bity budú mať hodnotu 1. Aká je pravdepodobnosť, že sa to bude dať zistiť pomocou štandardnej utility FSCK? Nebude sa ale vykonávať vzájomné porovnávanie obsahu súborov.

### Riešenie:

Najviac 6 súborov sa tam vôjde a pravdepodobnosť 0,32 alebo 0,34 podľa chaotických poznámok na mojom stole. Jednopísmenové mená súborov – súbor adresár tie mená obsahuje spolu s číslami blokov takže to v tých jeho blokoch je jednoducho rozmiestnené. Ak náhodne zmeníme bit obsahu súboru, je to vec používateľa súboru či verí jeho obsahu. Ak náhodne zmeníme bit v i-uzle alebo v nepriamom bloku, zrejme vznikne organizačný chaos a nesedí schéma i-uzla a adresára a systém to detekuje. Pravdepodobnosť trafenia sa na obsah súboru je pomer obsahu súborov a obsahu metadát.

## 13) I-uzly a súbory VII.

### Zadanie:

Spôsob organizácie jednoduchého súborového systému je nasledovný:

- používajú sa i-uzly, ktoré ukazujú na dva priame bloky a jeden nepriamy blok
- názvy súborov su jednoznakové

Vlastnosti logického disku su nasledovné:

- disk je veľký 70 blokov
- jeden blok je veľký 4 bajty (znaky), úplná kapacita disku je teda 280 bajtov.

1. Aký najväčší súbor je možné uložiť na disk? (počet bajtov)
2. Ak celkom zaplníme disk rovnako veľkými súbormi, kde každý ma veľkosť 12 znakov, aká je užitočná dátová kapacita disku pre používateľa? Podadresáre nevytvárame. Graficky to znázorníte a riešenie zdôvodnite.

**Riešenie:**

Maximálny súbor má veľkosť 24 bajtov. Rovnako veľkých súborov by sa vošlo 14, ale ak má byť aj koreňový adresár tak sa vôjde 12. Odpísal som to správne?

**14) I-uzly a súbory VIII.****Zadanie:**

Štruktúra i-uzla ukazuje na tri priame bloky a jeden jednoducho nepriamy blok. Veľkosť bloku je 8 bajtov. Číslo bloku má 1 bajt.

1. Na disku založíme koreňový adresár a doňho vložíme súbor s názvom „file“ o veľkosti 40 bajtov. Nakreslite štruktúru i-uzlov, blokov, ich obsahov a ich prepojení pre túto situáciu. Koľko blokov na disku bude obsadených pre ten adresár a súbor v ňom?
2. Nakreslite polohu všetkých blokov súboru na logickom disku, keď bol súbor vytvorený na prázdnom disku takto:  

```
f = open("file", O_WRONLY);  
lseek(f, 4, SEEK_SET);  
write(f, "1234567890", 10);
```
2. Aký veľký má byť logický disk, aby v ňom bolo možné uložiť systém súborov s horeuvedenou štruktúrou i-uzla, obsahujúci desať súborov maximálnej veľkosti? Mená súborov nech sú menej než osemznakové.
4. Uvažujme situáciu ako v otázke C. Na disku vznikne chyba, pričom jeden náhodne zvolený blok bude prepísaný tak, že všetky bity budú mať opačnú hodnotu. Aká je pravdepodobnosť, že sa to bude dať zistiť pomocou štandardnej utility pre kontrolu konzistencie systému súborov? Pravdepodobnosť je percento tých blokov z celku, ktorých prepísanie je zistiteľné utilitou.

**Riešenie:**

Obsadených má byť 9 blokov pre tú prvú situáciu. Operácia lseek posunie začiatok zápisu do súboru o štyri znaky, súbor je teda väčší než to čo sa doňho potom zapisuje. Súbor maximálnej veľkosti je taký, ktorý využíva celú štruktúru stromčeka i-uzla. Plus ešte miesto zaberá adresár. Ak náhodne zmeníme bit obsahu súboru, je to vec používateľa súboru či verí jeho obsahu. Ak náhodne zmeníme bit v i-uzle alebo v nepriamom bloku, zrejme vznikne organizačný chaos a nesedí schéma i-uzla a adresára a systém to detekuje. Pravdepodobnosť trafenia sa na obsah súboru je pomer obsahu súborov a obsahu metadát.



## 15) I-uzly a súbory IX.

### Zadanie:

Štruktúra i-uzla ukazuje na dva priame bloky, jeden jednoducho nepriamy a jeden dvojito nepriamy blok. Veľkosť bloku je 8 bajtov. Číslo bloku má 1 bajt.

```
f = open("file", O_WRONLY);
lseek(f, 9, SEEK_SET);
write(f, "12345678", 9);
lseek(f, 30, SEEK_SET);
write(f, "12345678", 9);
lseek(f, 270, SEEK_SET);
write(f, "12345678", 9);
```

1. Koľko dátových a koľko nedátových blokov bude použitých? Graficky znázornite.
2. Koľko čítaní a zápisov sektorov na disku sa vykoná počas vykonávania vyššie uvedeného úseku programu ? Veľkosť sektora je 4 bajty. Odpoveď zdôvodnite

### Riešenie:

Obmenený predošlý príklad. Čítanie je, keď sa na nejaký seek nastavujeme.

## 16) I-uzly a súbory X.

### Zadanie:

Nech i-uzol adresuje 5 blokov súboru priamo a ostatné bloky súboru cez dva bloky nepriamych indexov. Veľkosť bloku je 64 kilobajtov. Najvyššie číslo bloku na logickom disku zaberie 16 bajtov. Na disku založíme nový adresár a doňho vložíme súbor X s veľkosťou 512 kilobajtov. Koľko blokov na disku bude obsadených pre ten nový adresár a pre súbor X?

### Riešenie:

Adresár je špeciálny súbor a obsadí **jeden blok** pre svoj i-uzol a **jeden blok** pre svoj obsah.

Súbor X obsadí  $512 \text{ kB} / 64 \text{ kB} =$  **osem blokov** pre svoje údaje a **jeden blok** pre svoj i-uzol.

Schému i-uzla je potrebné upraviť podľa tohoto príkladu a možno skonštatovať, že z ôsmich blokov obsahujúcich údaje súboru bude 5 blokov adresovaných priamo z i-uzla a ďalšie tri bloky budú adresované ešte cez **jeden blok** nepriamych indexov.

Blok nepriamych indexov pojme  $64 \text{ kB} / 16 \text{ B} = 4000$  indexov, takže môže adresovať až taký počet blokov, nie iba tri.

Dohromady bude teda obsadených **12 blokov**.

**17) I-uzly a súbory XI.****Zadanie:**

Na disku je v jedinom adresári uložených 5 súborov rovnakej veľkosti: 17 znakov. Týchto päť súborov bolo skomprimovaných do jedného súboru, ktorého veľkosť obsahu vyšla na 80 znakov.

Použitý súborový systém je založený na princípe i-uzlov, pričom jeden blok má kapacitu jedného i-uzla, alebo štyroch ukazovateľov, alebo 8 znakov súboru, alebo štyroch položiek adresára. Samotný i-uzol má dva priame ukazovatele, jeden nepriamy a jeden dvojito nepriamy.

Odpovede na nasledujúce otázky vysvetlite a znázorníte schémou (kreslením obsahu blokov na disku):

Ako vyzerá hore opísaná štruktúra i-uzla?

Koľko blokov a akých blokov sa ušetrí náhradou piatich súborov za jeden skomprimovaný?

Nech je priemerná veľkosť súborov každého druhu v systéme 20 znakov. Nech je veľkosť vyčleneného priestoru na disku 60 blokov, pre bloky súborov aj i-uzly. Koľko blokov v uvedenom priestore disku optimálne vyhradíme pre i-uzly?

**Riešenie:**

Jeden 17-znakový súbor obsadí jeden i-uzol (teda jeden blok), jeden nepriamy blok a 3 dátové bloky.

Päť 17-znakových súborov obsadí dokopy 5 i-uzlov, 5 nepriamych blokov a 15 dátových blokov.

Osemdesiatznakový súbor (to akože skomprimovaných tých 5) obsadí 1 i-uzol, 3 nepriame bloky a 10 dátových blokov.

Ušetrili sme 11 blokov (25 mínus 14). Ale ešte nezabudnúť na adresár, tam ubudne jeden dátový blok. Takže sa ušetrí 12 blokov dokopy.

20-znakový súbor potrebuje jeden i-uzol a štyri bloky ukazovateľov a údajov. Pomer je 1 : 4, rozdelenie vychádza 12 : 48 čo je dokopy 60. Teda 12 i-uzlov.

**18) I-uzly a súbory XII.****Zadanie:**

Na disku je v jedinom adresári uložených 5 súborov rovnakej veľkosti: 17 znakov. Týchto päť súborov bolo skomprimovaných do jedného súboru, ktorého veľkosť obsahu vyšla na 80 znakov.

Použitý súborový systém je založený na princípe i-uzlov, pričom jeden blok má kapacitu jedného i-uzla, alebo štyroch ukazovateľov, alebo 8 znakov súboru, alebo štyroch položiek adresára. Samotný i-uzol má jeden priamy ukazovateľ, dva nepriame a jeden dvojito nepriamy.

Odpovede na nasledujúce otázky vysvetlite a znázornite schémou (kreslením obsahu blokov na disku):

Ako vyzerá hore opísaná štruktúra i-uzla?

Koľko blokov a akých blokov sa ušetrí náhradou piatich súborov za jeden skomprimovaný?

Nech je priemerná veľkosť súborov každého druhu v systéme 20 znakov. Nech je veľkosť vyčleneného priestoru na disku 60 blokov, pre bloky súborov aj i-uzly. Koľko blokov v uvedenom priestore disku optimálne vyhradíme pre i-uzly ?

#### Riešenie:

Ušetria sa dva bloky z adresárového súboru a desať blokov súborov.  
Tretia otázka rovnaká číselná odpoveď ako predošlý príklad.

### 19) I-uzly a súbory XIII.

#### Zadanie:

```
for (i=0; i<10; i++) fprintf(subor, "abcdef");
```

Uvedený proces zapisuje znaky do súboru. Použitý súborový systém je založený na princípe i-uzlov, pričom jeden blok má kapacitu jedného i-uzla, alebo štyroch ukazovateľov, alebo 8 znakov súboru. Samotný i-uzol má štyri priame ukazovatele na bloky dát a jeden dvojito nepriamy ukazovateľ. Odpovede na nasledujúce otázky vysvetlite a znázornite schémou (kreslením obsahu blokov na disku):

Ako vyzerá hore opísaná štruktúra i-uzla?

Koľko blokov, akých blokov a čím bude naplnených po vykonaní uvedeného procesu?

Koľko cyklov sa môže vykonať uvedená operácia `fprintf()` celkovo dohromady bez vzniku chybného stavu?

#### Riešenie:

Stručne: jeden blok sa naplní pre i-uzol, 2 bloky pre ukazovatele a 8 blokov pre dáta.

Cyklov zápisu sa vykoná v poriadku 26. Kapacita súboru v danom súborovom systéme je 160 znakov.

## 20) Štandardný výstup programu

### Zadanie:

Majme 7 súborov s nasledovným jednobajtovým obsahom:

```
a.txt  O
b.txt  )
c.txt  -
d.txt  =
e.txt  :
f.txt  $
g.txt  „
```

Aký bude štandardný výstup nasledovného programu po jeho vykonaní? Predpokladajte, že pri volaní `open()` nenastane chyba a všetky súbory sa úspešne otvoria. V prípade, že si myslíte, že program skončí chybou, zdôvodnite.

<pre>int main(void) {     int a,b,c,d,e,f,g;     char x;      a=open("a.txt", O_RDONLY);     b=open("b.txt", O_RDONLY);     c=open("c.txt", O_RDONLY);     d=open("d.txt", O_RDONLY);     e=open("e.txt", O_RDONLY);     f=open("f.txt", O_RDONLY);     g=open("g.txt", O_RDONLY);      close(0);     dup(a);     read(0,&amp;x,1);     close(0);     write(1,&amp;x,1);     dup(f);     read(0,&amp;x,1);     write(1,&amp;x,1);     close(0);     dup(d);     read(0,&amp;x,1);</pre>	<pre>close(d); close(0); write(1,&amp;x,1); dup(e); close(e); read(0,&amp;x,1); write(1,&amp;x,1); close(10); close(0); dup(c); read(0,&amp;x,1); write(1,&amp;x,1); close(c); close(a); close(11); close(8); close(9); close(0); dup(b); read(0,&amp;x,1); close(b); write(1,&amp;x,1); exit(0); }</pre>
---	---

### Riešenie:

O S = : - )

Je potrebné si nakresliť tabuľku otvorených vstupno/výstupných súborov, prvá položka tabuľky má index 0, z ktorej sa vždy číta a druhá má index 1 do ktorej sa vždy píše. Ďalších sedem položiek sú otvorené súbory. Položky tabuľky sa zatvárajú, teda uvoľňujú a pri `dup()` sa na prvé voľné miesto v tabuľke zhora duplikuje daná položka na čítanie.

