RIADNY TERMIN OS 2012

1 Večerajúci filozofi, trebalo "implementovať" funkcie take_forks a put_forks. Mal tam ničo písané k tomu ešte, že ako si to predstavuje.

```
#define N 5

semaphor sem[N]={1};
Take_fork(int i)
{
        if(i!=N ) // čísla v strede
        {
                sem[i-1].wait();
                sem.[i].wait();
        }
        else // číslo N v mojom pripade 5 aby najprv 0 čakala potom 4
        {
                sem[0].wait();
                sem.[i-1].wait();
        }
}
Put_forks(int i)
{
        sem[i%N].signal();
        sem[(i-1)%N].signal();
}
```

Filozofi Bankárovím algoritmom
Počiatočná Tabuľka

	1-vydl	2-vydl	3-vydl	4-vydl	5-vydl
1-fil.	0	1	0	0	1
2-fil.	1	0	1	0	0
3-fil.	0	1	0	1	0
4-fil.	0	0	1	0	1
5-fil.	1	0	0	1	0
Banka	1	1	1	1	1

Nebezpečný stav :

	1-vydl	2-vydl	3-vydl	4-vydl	5-vydl
1-fil.	0	0	0	0	1
2-fil.	1	0	0	0	0
3-fil.	0	1	0	0	0
4-fil.	0	0	1	0	0
5-fil.	0	0	0	1	0
Banka	0	0	0	0	0

2 Mame proces A, ktory 100ms pracuje a 5ms caka na I/O operaciu. Tento proces je nekonecna slucka. Potom je 5 procesov B -> B0-B4 a tie 1ms pocitaju a 5ms cakaju na I/O.Kazdy proces B ma v sebe slusku so 100 cyklami. A mali sme zistit za aky cas sa skonci prvy zo skupiny procesov B.

a)nepreemptivne planovanie(tzn. proces ma procesor k dispozicii az pokial nevykona celu svoju cinnost)FCFS(first come first served)

b)Round Robin. Casove kvantum bolo 20 ms.
Cas na vykonanie preruseni bol 0.1 ms.
Prvy proces ktory je spusteny je proces A.

Riesenie :

Treba si uvedomit, ze cas na prepinanie procesov sa nezaratava do casoveho kvanta, cize proces ma sam pre seba celych 20 ms. Dalsia vec s ktorou treba ratat je, ze ak sa proces zablokuje tym ze caka na I/O operaciu tak dojde k preruseniu a naplanuje sa dalsi proces v poradi. (Cize nie je to tak ze pri FCFS by stale bezal dookola len proces A ...lebo po 100ms dojde k preruseniu a prepnutiu procesov, takze sa dostanu k slovu aj procesy B)

a) Je to nepreemptivny algoritmus, tak proces ktory ma k dispozicii procesor pracuje kym sa nezablokuje.Prvy proces je na plane A. Prvy prisiel, prvy bude obsluzeny. Takze spusti sa na 100ms, potom dojde k preruseniu, lebo musi cakat na I/O. Takze sa spusti proces B0 (napr.) a ten bezi 1ms a potom aj ten sa zablokuje a caka na I/O ....a takto postupne sa vystrieda vsetkych 5 procesov B, a kazdy bezal 1 ms, dokopy 5ms, cize proces A uz ma nacitane data a moze pokracovat v cinnosti a znova sa to cele dookola zopakuje.Za ten cas co bezi proces A maju vsetky procesy B uz nachystane data z I/O operacii.

Uzitocny cas treba ratat nejako takto(stratil som v tomto priklade 2 body, tak predpokladam ze v a) aj v b) som ten cas nejak zle zratal, ale princip by mal byt takyto) :

Cas ktory procesor pracuje je : (100ms+5*1ms) --> toto je uzitocny cas v jednom "cykle" vystriedania sa procesov..kedze proces B ma 100 iteracii a v kazdej iteracii 1 ms pocita a potom caka 5 ms a az potom konci jedna iteracia, je treba pocitat 99* "cyklus" + ten sty krat je treba 100 ms pre proces A, 1 ms pre proces B a este 5 ms aby skoncil citanie a az potom konci cely proces. (dufam ze to je zrozumitelne, ak nie, tak sa pytaj)....takze konecne samotny vypocet :

uzitocny cas (100ms+5*1ms)*99 + 100+1+5 --->do uzitocneho casu sa rata len cas kedy je procesor vyuzity
---------- = ----------
celkovy cas (100ms + 5*1ms + 6*0.1ms)*99 + 100+1+5+0.6 ----->do celkoveho casu je uz treba pripocitat aj cas na vykonanie preruseni a to je 6*01, lebo musi sa vykonat prerusenie po procese A a 5* po procesoch B

b) V algoritme round robin sa postupuje tak ze je urcene casove kvantum, ktore ma kazdy proces k dispozicii a potom dojde k preruseniu a prepnutie dalsieho procesu v rade(procesy sa striedaju stale dookola, podla toho ako prisli), nezalezi na tom ci proces este moze pracovat, alebo caka, jednoducho sa mu vezme procesor, ak ubehne kvantum jemu pridelene. Procesy su teda zoradene v rade...predpokladajme ze takto : A, B0...B4 . Prvy sa teda spusti proces A na 20 ms. Dalsi na rade je proces B0, ten sa spusti na 1 ms a potom sa zablokuje, lebo caka na I/O, takze dojde k preruseniu a naplanuje sa dalsi proces v rade(dostane k dispozicii cele nove casove kvantum, ak sa pocas neho zablokuje, tak ide dalsi proces na rade ktory ma zasa cele nove

kvantum a tak dalej.....cize nie ze dokonci to nacate kvantum, tak to nie je). Takto sa postupne vystriedaju vsetky B procesy a zas je na rade proces A a cele sa to znova opakuje. Pokial prejde 20 ms procesu A, vsetky B procesy uz maju data nachystane a znova sa "historia "moze opakovat. A zasa naopak; ak proces A caka na I/O, tak sa mu data pripravia pocas 5 ms co bezia procesy B. cize tu netreba uvazovat ziadne specialne pripady...jednoducho to vsetko ide pekne po sebe.

Vypocet procesoroveho casu : Znova je to analogicky ako v ulohe a), je tu 99 cyklov "klasickych" + ten posledny ukoncovaci (Mozno tam som mal tu chybu, ze bolo treba ratat cas az do konca nacateho casoveho kvanta v ktorom este ten proces co ide skoncit cita data a zatedy pracuje iny proces....neviem)

$$\frac{\text{uzitocny cas } (20ms+5*1ms)*99 + 20+1+5}{\text{celkovy cas } (20ms + 5*1ms + 6*0.1ms)*99 + 20+1+5+0.6} = \frac{}{}$$

3 Príklad z fragmentom

Zadanie : Dany je fragment programu

Fragment programu:

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

je po skompilovani 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≥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	

Máme 4 rámce tieto rámce majú adressy 0x0A, 0x0E, 0x1A, 0x1E, a veľkosť jednej stránky 1024 slabík.Vypočítajte fyzické adresy prvokov A[64] a C[20] **po prvej interácií cyklu**, Vyber obete second change:

Riešenie :

Krok 1: zistíme si pracovnú množinu stránok (na ktoré stránky bude táto časť programu pristupovať). Keďže
máme veľkosť stránky 400 hexa vieme, že jednostlivé stránky budú od seba vzdialené 400 (hexa) tj. Ich čísla

budú 0, 400, 800, C00, 1000 (hexa) atď. Číslo stránky dostaneme tak, že vydelíme číslo fyzickej adresy
veľkosťou stránky (celočíselne). Takto dostaneme tieto stránky:
kód - 0
pole A - 6
pole B - 7
pole C - 8
konšt. - 9

Krok 2 : treba zostaviť reťazec odkazov na stránky (ako konkrétne budú za sebou nasledovať stránky počas vykonávania). Vždy platí, že najskôr sa načíta inštrukcia, a potom sa vykoná. Ak procesor načítava externú premennú tak pristupuje aj do inej časti pamäte ako ku kódu.
Takže reťazec odkazov pre daný kód vyzerá nasledovne:
0 9 0 9 (0 0 0 7 0 8 0 6 0 9 0)$_{1024}$ 0 0

Krok 3 : Second Change Tabulka

0	9	0	9	0	0	0	7	0	8	0	6	0	9	0
0_1	9_1	9_1	9_1	9_1	9_1	9_1	7_1	7_1	8_1	8_1	6_1	0_1	9_1	9_1
	0_1	0_1	0_1	0_1	0_1	0_1	9_1	9_1	7_1	7_1	8_0	6_1	0_1	0_1
							0_1	0_1	9_1	9_1	7_0	8_0	6_1	6_1
									0_1	0_1	9_0	7_0	8_0	8_0
*	*						*		*		*	*	*	

0	0	0	7	0	8	0	6	0	9	0
9_1	9_1	9_1	7_1	7_1	8_1	8_1	6_1	6_1	9_1	9_1
0_1	0_1	0_1	9_1	9_1	7_0	7_0	8_1	8_1	6_1	6_1
6_1	6_1	6_1	0_1	0_1	9_0	9_0	7_0	7_0	8_1	8_1
8_0	8_0	8_0	6_1	6_1	0_0	0_1	0_0	0_1	0_0	0_1
			*		*		*		*	

0	0	0	7	0	8	0	6	0	9	0
9_1	9_1	9_1	7_1	0_1	8_1	8_1	6_1	6_1	9_1	9_1
6_1	6_1	6_1	9_0	7_1	0_1	0_1	8_1	8_1	6_0	6_0
8_1	8_1	8_1	6_0	9_0	7_1	7_1	0_1	0_1	8_0	8_0
0_1	0_1	0_1	8_0	6_0	9_0	9_0	7_1	7_1	0_0	0_1
			*	*	*		*		*	

Rátame až kým na konci cyklu nebudú rovnaké čísla stránok a aj s rovnakými R-bytami
Tento príklad nieje dobrý lebo by som sa narátal ešte hodne …..

4.krok . Zostrojíme tabulku ktorá bude určovať v ktorom ráme sa nachádza ktorá stránka … nasledovne.
(Pozerali sme na tabulku second change....)

	0	9	0	9	0	0	0	7	0	8	0	6	0	9	0
0x0A	0	0	0	0	0	0	0	0	0	0	0	6	6	6	6
0x0E		9	9	9	9	9	9	9	9	9	9	9	0	0	0
0x1A								7	7	7	7	7	7	9	9
0x1E										8	8	8	8	8	8

5.krok Ideme počítať A[64] a C[20]...
Polu A zodpovedá číslo 6 ktoré sa nachádza v rámci 0x0A
Polu C zodpovedá číslo 8 ktoré sa nachádza v rámci 0x1E

6.krok asi len prírať ktomu 64 a 20 … takže …. asi

64 = 2^6 = 100 0000 = 0x0040
20 = 2^4 + 4 = 1 0100 = 0x0014

0x0A je iba cislo strankoveho ramu, pricom strankovy ram obsahuje 1024 bytov. Pamat je teda priradena
strankovym ramom nejako takto:

Strankovy ram: Pridelena pamat:
0x00 0x0000 - 0x03FF
0x01 0x0400 - 0x07FF
0x02 0x0800 - 0x0BFF
0x03 0x0C00 - 0x0FFF
…
0x0A 0x2800- 0x0BFF
...
0x1E 0x7800-0x7BFF
Typ VZOREC : ram * veľkosť stránky = pamäť
tj 0x0A * 0x400 = 0x2800
to znamená že adresa A[64] JE 0x0040 + 0x2800 = 0x2840
B[20] JE 0x0014 + 0x7800 = 0x7814

4 Synchronizácia správ pomocou semafóru.

```
semaphore sem1=0,sem2=0;
msgt buffer;


send (message m)
{
        buffer=m
        sem2.signal();
        sem1.wait()
}
receive(message *m)
{
        sem2.wait()
        *m=buffer
        sem1.signal()
}
```

5 Monitor. Trebalo dorobiť tri operácie tak, aby sa na konci vypísalo to, čo chcel. Chcel to cez podmienkové premenné (typ cond).

```
int v ;
cond cnt,cnt2;

P1(void)
{
        v=3;
                printf("%d",v);
                cnt.csignal();
}
P2(void)
{
        while (v!=3) cnt.cwait();
        v*=3;
        printf("%d",v);
        cnt2.csignal();
}
P3(void)
{
        while (v!=9) cnt2.cwait();
        v*=2;
                printf("%d",v);
}
```

6 Príklad z Fragmentom NRU (Not recently USED)

0x0040 (Rl)	<- ZERO Rl 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>=n choď na 0x004C
0x0044 (R3) <-B(Rl)	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) - (Rl)	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 (Rl) <-(Rl) + ONE	inkrementuj k
0x004B branch 0x0042	

Proces ma pridelené 4 rámy 0x8, 0x9, 0xA, 0xB.

Chceme vedieť stav po vykonaní inštrukcie na adresse 0x004A.

0x0000 -> 0x03FF	0	Program
0x0400 -> 0x07FF	1	
0x0800 -> 0x0BFF	2	
0x0C00 -> 0x0FFF	3	A
0x1000 -> 0x13FF	4	B
0x1400 -> 0x17FF	5	C
0x1800 -> 0x1BFF	6	D
0x1C00 -> 0x1FFF	7	ONE, ZERO, N

1.Krok : Zostrojiť REFERENCE STRING : 0 7 0 7 0 0 0 4 0 5 0 3 0 0 4 0 6 0 7 0 | 0 0

NRU - TEORIA

Každá stránka je označená dvoma stavovými bitmi: R – referenced a M – modified.

Bit R sa nastavuje na 1, ak v stanovenom doterajšom časovom intervale bola stránka adresovaná

V pravidelných intervaloch sa R bity stránok nulujú.

Bit M sa nastavuje na 1 vtedy, ak sa do stránky zapisovalo.

Podľa týchto R a M bitov sú Stránky rozdelené do 4 kategórií:

1. R = 0, M = 0 - stránka naposledy nereferencovaná, obsah nezmenený, možno obetovať,
2. R = 0, M = 1 - nereferencovaná, ale pri vyhodení treba jej obsah vrátiť na disk, lebo sa zmenil,
3. R = 1, M = 0 - referencovaná, asi sa často používa, ale vyhodila by sa ľahko – je nezmenená,
4. R = 1, M = 1 - používa sa asi často a má zmenený obsah, neoplatí sa ju obetovať.

Na obetovanie teda vyberame tu stranku, ktora je v najnizsej kategorii. Ak je takychto stranok viac, zvolime nahodne 1 z nich.

2.krok: NRU - Upravenie Reference string

Rozdelíme si REFERENCE STRING podľa inštrukcii a posledné 3 zmažeme lebo nie sú podstatné :

Vytvoríme tak Reference string nasledovný :

0 7 | 0 7 | 0 | 0 | 0 4 | 0 5 | 0 3 | 0 | 0 4 | 0 6 | 0 7

3.krok: NRU - Mazanie R bitu

R bity sa budu periodicky nulovat po vykonani 4 instrukcii // to by malo byť zadané

0 7 | 0 7 | 0 | 0 | 0 4 | 0 5 | 0 3 | 0 | 0 4 | 0 6 | 0 7

^ ^

tu tu

4.krok: Tabulka

(1) Vypadla stránka 7 lebo mala R bit na 0 a aj M bit na 0

(2) Vypadok stranky 5. pretože mala R bit 0 a M bit 0 (1.skupina)

(3) Vypadok stranky 3. pretože mala R bit 0 a M bit 1 (2.skupina)

				tj.070700			Vynul.			tj.04			tj.05			tj. 03 (1)			0 a Vynul.			tj. 04		
	0.inštruk.			1-4.Inštruk.			Po 4.Inš			Po 5.Inš			Po 6.Inš			Po 7.Inš			Po 8.Inš			Po 9.Inš		
	F	R	M	F	R	M	F	R	M	F	R	M	F	R	M	F	R	M	F	R	M	F	R	M
0	X	0	0	0x8	1	0	0x8	0	0	0x8	1	0	0x8	1	0	0x8	1	0	0x8	0	0	0x8	1	0
3	X	0	0	X	0	0	X	0	0	X	0	0	X	0	0	0x9	1	1	0x9	0	1	0x9	0	1
4	X	0	0	X	0	0	X	0	0	0xA	1	0	0xA	1	0	0xA	1	0	0xA	0	0	0xA	1	0
5	X	0	0	X	0	0	X	0	0	X	0	0	0xB	1	0	0xB	1	0	0xB	0	0	0xB	0	0
6	X	0	0	X	0	0	X	0	0	X	0	0	X	0	0	X	0	0	X	0	0	X	0	0
7	X	0	0	0x9	1	0	0x9	0	0	0x9	0	0	0x9	0	0	X	0	0	X	0	0	X	0	0

(1) Vypadla stránka 7 lebo mala R bit na 0 a aj M bit na 0

	tj. 06 (2)			tj. 07 (3)		
	Po 10 Inštrukcií			Po 11 Inštrukcií		
	F	R	M	F	R	M
0	0x8	1	0	0x8	1	0
3	0x9	0	1	X	0	0
4	0xA	1	0	0xA	1	0
5	X	0	0	X	0	0
6	0xB	1	1	0xB	1	1
7	X	0	0	0x9	1	1

(2) Vypadok stranky 5. pretože mala R bit 0 a M bit 0 (1.skupina)

(3) Vypadok stranky 3. pretože mala R bit 0 a M bit 1 (2.skupina)

7 Spocitaj aku vzdialenost (v cylindroch) prejde hlavicka disku, pri algoritme a.) SSTF b.) SCAN c.) C-SCAN. Rad aj aktualna pozicia hlavicky bola zadana. (6b)

FCFS - ide sa doradu ako je zadane
SSTF - ide sa vzdy k najblizsiemu(1-2-3-4-5)
SCAN - ide sa po okraj disku(0-199), tam sa zmeni smer, inde nie

C-SCAN - ide sa po okraj disku(0-199), tam sa zmeni smer, obsluhuje sa iba pri ceste jednym smerom

LOOK - ako SCAN, ale nejde sa po okraj disku, ale po najvzdialenejsiu poziadavku

C-LOOK - ako LOOK, ale obsluhuje sa len po ceste jednym smerom.

8 Priklad na vzajomne vylucovanie, riesene cez semafory. Procesov je n (ocislovane od 0 po n-1) viacero, telo procesu je reprezentovane funkcou proces(). Funkcia moje_cislo() vrati cislo volajuceho procesu. Uloha je napisat tela programov init() a resume(). Vo funkcii resume() si procesy odovzdavaju procesor. Ako prvy sa spusti proces s cislom 0.

Zadané :

```
#define pocet_procesov 5
void proces(int pid) {
init (moje_cislo());
while (1){
rob_daco_nepodstatne();
int k = CisloGenerator();
resume(k);
}
}
```

Riešenie :

```
semaphore sem[N] = {0} // vsetky semafory inicializovane na 0
void init(int pid) {
 if (pid != 0)sem[pid].wait();
}
void resume(int k) {
sem[k].signal();
sem[moje_cislo()].wait();
}
```

9 Napiste synchrnonne ssend a sreceive pomocou asynchrnonnych asend a areceive

```
ssend(sprava)
        {
                asend(sprava);//posle spravu
                areceive(token);//zablokuje sa az kym mu nedojde token
        }

srecive(*sprava)
        {
                areceive(sprava);//precita spravu
                asend(token);//po uspesnom precitani posle token aby sa ssend odblokoval
        }
```

10 inode... bolo 8 priamych, 1 nepriamy a 1 dvojito nepriamy a kazdy mal 32b ... a bola tabulka s polozkami a velkostou poloziek... a poctom volnych.. a ze o kolko sa zmeni pocet volnych ked sa prida 758kB subor... tam bolo dost o hubu jedine premienanie.... a malo to aj ulohu b) ze ked sa tam daju 2 subory maximalnej velkosti ze ako sa zmeni pocet volnych poloziek

I-uzly.
Disk ma 2,5 mil. blokov, volnych 5,9 mil. poloziek tabulky i-uzlov (resp. metadát/atributov).
Blok ma 512B (0,5 kB).
Adresa kazdeho bloku zabera 32b = 4B.
1 tabuľka i-uzlov sa zmestí do jednoho bloku na disku.
Struktura tabulky i-uzlov: 8 priamych, 1 nepriamy, 1 dvojito nepriamy blok.

A) Ako sa zmení počet voľných blokov, keď tam pridáme súbor o veľkosti 785 kB? (v preklade koľko a akých blokov ubudne)
B) Ako sa zmení počet voľných blokov, keď tam vložíme dva súbory maximálnej veľkosti? (vypočítať maximálnu veľkosť súboru a koľko zaberá)

Riešenie:

Rekapitulácia zadania:

- 1tabuľka i-uzla sa zmestí do jedného bloku
- tabuľka i-uzla má tieto ukazovatele/polozky: 8 priamych, 1 nepriamy a 1 dvojito-nepriamy blok

A)
Krok 1: Koľko blokov potrebujeme na tento súbor?
785kB (veľkosť) / 0,5kB (blok) = 1570 blokov.

Koľko odkazov môžeme mať najviac v nepriamych blokoch?
512B (blok) / 4B (adresa) = 128 odkazov.

Krok 2: Priradíme bloky na priame adresovanie: máme 8 priamych, teda využijeme 8 blokov (zostáva nám 1570-8=1562 blokov).

Krok 3: Jednoduché nepriame (single indirect): máme iba jeden, ktorý vie ukázať na 128 blokov. 1*128=128 blokov. (zostáva ním 1562-128=1434 blokov)

Krok 4: Dvojito nepriame (double indirect): tento mám tiež len jeden. V ňom môžeme mať maximálne 128 odkazov na bloky, ktoré sa odkazujú na iné odkazovacie bloky. Avšak nevyužijeme všetky odkazy z prvého bloku (lebo 128 (prvý blok) * 128 (každý druhý blok) = veľa, nepotrebujeme toľko na 1434 blokov).
Otázka teda je: koľko blokov odkazov po 128 potrebujeme na pokrytie 1434 blokov? Je to 1434 / 128 = 11,203125. Zaokrúhľujeme hore, lebo potrebujeme aj ten 12. blok i keď ho celý nevyužijeme.

Teda pre dvojito nepriamu máme 1*12*128=1536 (áno, toto číslo je menšie preto, lebo v poslednom dvanástom bloku nevyžívame všetkých 128 odkazov, ale iba 26 (=128*0,203125).

Zhrnutie úlohy A: Takže dostávame sa k výsledku. Pre i-uzol potrebujeme 1 blok. Pre samotný súbor potrebujeme 1570 blokov. Pre nepriame bloky potrebujeme: 1 (nepriamy) + 1*12 (dvojito nepriame) = 13 blokov.

Odpoveď: Po pridaní súboru veľkosti 785 kB **sa zníži počet voľných blokov pre uloženie tabuliek i-uzlov o 1** a počet ostatných blokov o 158**4** (1570 súbor + 1 nepriamy + 1**3** dvojito nepriame).
Prehľadnejšie: súbor zaberie 1585blokov (1-Inode + 8 dátových blokov + 1 nepriamy + 128 dátových blokov + 1 dvojito nepriamy + 12 nepriamych+ 1434 dátových blokov)

11 napisat v akych stavoch(a kolko) sa budu nachadzat dva nekonecne procesy p1 a p2 (tusim sa malo zacat p2) -p1 trva vypocet 2ms a v/v 12sek a p2 nema v/s 2 procesy P1,P2 bezia nekonecne. P2 zacina. Kvantum je 15ms, mal sa vyuzit round robin algoritmus. P2 nerobilo vstupno vystupne operacie, P1 robilo V/V kazde 2ms. Cas na zmenu kontextu je zanedbatelny. Popis stavy procesov a dlzky casov, ktory kolko bezi.

Stavy: Running

Runnable/Ready
Blocked/Waiting
Zombie
Riešenie:
P1 (výpočet 2ms, V/V oper. -12ms) , P2 (V/V-oper. - nemá, výpočet ???ms) ,
Časové kvantum 15ms

1.začína P2 - buď je taký krátky že sa mu odoberie časové kvantum 15ms. alebo ho celé využije.
P1-runable , P2 - running

2.nasleduje P1 - pracuje 2ms - odoberie sa mu zasové kvantum a v poradí pracujú jeho V/Voperácie.
P1 - running , P2 - runable

3.nasleduje P2 - (pre proces P1 ešte 12ms budú pracovať V/V operácie)
P1 - blocked , P2 – running

4.Nasleduje stale bezi p2, akurat dobehli VV operacie p1takze toto nizsie je stav ktory bude trvat 3ms, potom kvantum dostane opäť p1 a cyklus sa opakuje
P1 - Runable , P2 – Running.