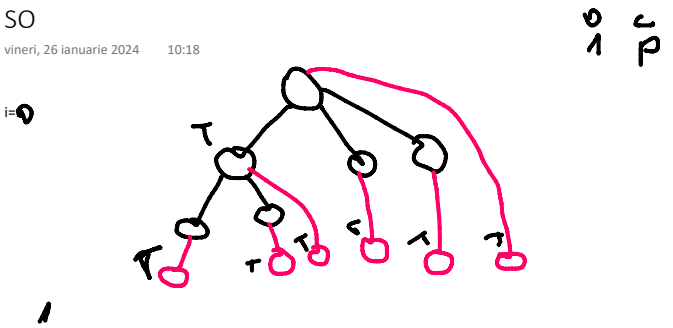
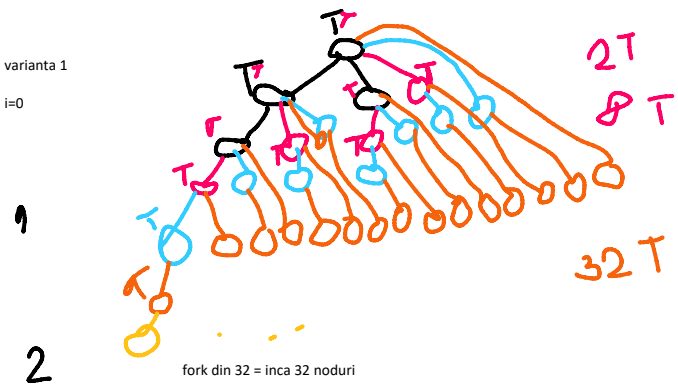
SO

vineri, 26 ianuarie 2024 10:18



varianta 1

i=0



fork din 32 = inca 32 noduri

64 noduri
32 thread uri

am presupus ca fork nu copiaza thread urile parintelui

Var 2 ex 2

```
nod=1
if(nod==1)
        if(!fork())
                nod=2

if(nod==1)
        if(!fork())
                nod=3
if(nod==1)
        if(!fork())
                nod=4


if(nod==2)
        if(!fork())
                nod=5

if(nod==3)
        if(!fork())
                nod=6
```

// A C program to demonstrate **Zombie Process.**	PĂRINTELE FACE SLEEP DECI NU MOARE, DAR NICI NU IL ASTEAPTA PE COPIL

```
// Child becomes Zombie as parent is sleeping
// when child process exits.
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        // Fork returns process id
        // in parent process
        pid_t child_pid = fork();
```

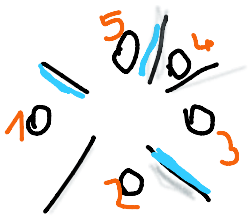
```
        // Parent process
        if (child_pid > 0)
                sleep(50);

        // Child process
        else
                exit(0);

        return 0;
}
```

// A C program to demonstrate **Orphan Process.**	părintele își dă print si moare, copilul ramane in aer

```
// Parent process finishes execution while the
// child process is running. The child process
// becomes orphan.
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        // Create a child process
        int pid = fork();

        if (pid > 0)
                printf("in parent process");

        // Note that pid is 0 in child process
        // and negative if fork() fails
        else if (pid == 0)
        {
                sleep(30);
                printf("in child process");
        }

        return 0;
}
```

v4 ex 3

```
do{

        if(i%2==1)
                wait(chopstick[(i+1) %n ]);
                wait(chopstick[i]);

                /* .. */
                signal(chopstick[(i+1) %n ]);
                signal(chopstick[ i ]);


        else
                wait(chopstick[i]);
                wait(chopstick[(i+1) %n ]);

                /*..... * /
                signal(chopstick[I]);
                signal(chopstick[(i+1) %n ]);

} while (true);
```



cei cu numar par vor astepta mereu sa se elibereze resursa de la un vecin cu numar impar(din dreapta)
resursa de la cei cu nr par din stanga va fi libera daca vecinul din stanga nu mananca, atunci o va putea lua si cel par

2 mananca <=> 2 hungry si 1 si 3 nu mananca (ca la monitor)

b) de ce solutia satisface cele 3 proprietati?
excl mutuala: cineva are resursa la un moment => celalat nu are aceeasi resursa

> 2 mananca <=> 2 hungry si 1 si 3 nu mananca (ca la monitor)

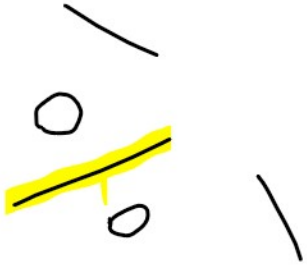
progres:

> Proprietatea de progres asigură că, dacă un filosof dorește să ridice betișoarele, va putea face acest lucru eventual. În codul furnizat, filosofii nu stau blocati într-un buclu infinit, ci ridică și eliberează betișoarele într-un mod ciclic, astfel încât progresul este asigurat.

timp finit de asteptare: nu se intra in deadlock pentru ca nu se ocupa toate resursele odata

> Timp finit de așteptare: Această proprietate înseamnă că un filosof nu va aștepta la nesfârșit pentru a obține accesul la betișoarele sale. În acest cod, filosofii

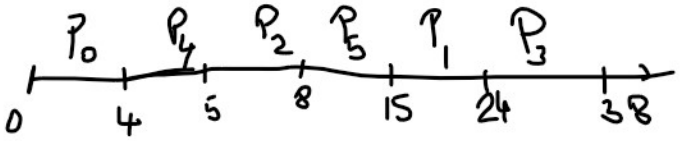
așteaptă o perioadă finită pentru a obține accesul la betișoare (prin funcția pthread_mutex_lock), și apoi își eliberează betișoarele. În consecință, timpul de așteptare este finit.



problema cu cpu scheduleing din notite

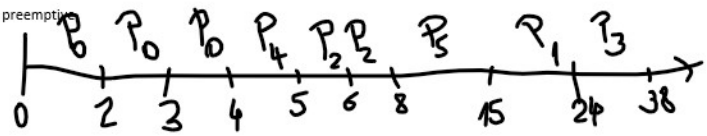
SJF non preemptive = nu intrerupem pe nimeni

Proces	Arrival	Burst	Priority
P_0	0	4	3
P_1	2	9	5
P_2	4	3	1
P_3	2	14	2
P_4	3	1	4
P_5	6	7	3



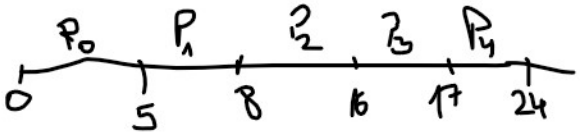
timpul mediu de asteptare
0 + 13 + 1 + 22 + 1 + 2 = 39 / 6 = 6.5

SJF preemptive

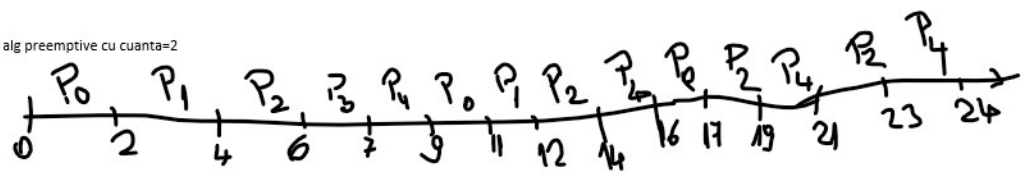


Proces	t	CPU
P_0	0	
P_1	0	
P_2	2	
P_3	4	
P_4	6	

non preemptive



alg preemptive cu cuanta=2

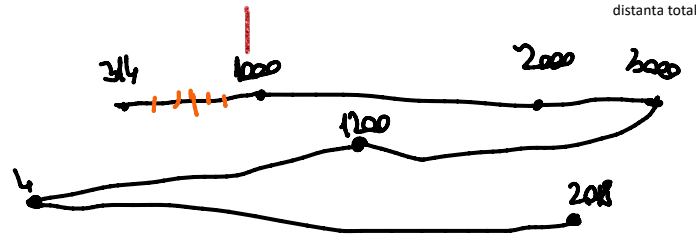


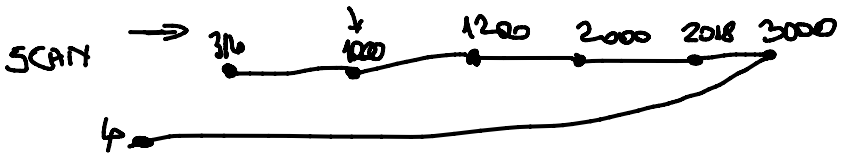
5. Fie un disk cu 5000 de cilindri si următoarea coadă de cereri I/O în asteptare 2000, 3000, 1200, 4, 2018. Fiecare intrare reprezintă un cilidru, iar capul de citire al disk-ului se află la poziția 1000 si a fost înainte la poziția 314.

   (a) (5p) Începând de la poziția curentă, care este ordinea si distanța totală parcursă de cap pentru a satisface toate cererile din coadă folosind algoritmul FCFS?

   (b) (5p) Dar folosind algoritmul SCAN?

distanta totala= 2000-1000 + 3000-2000 + 3000-1200 + 1200-4 + 2018-4 = 7010





distanta = 3000-1000 + 3000-4 = 4996

1200 - 1000 + 2000-1200 + 2018-2000+ 3000 - 2018 + 3000-4 = 4996