Punerea unor monitoare pe obiecte

Alte sisteme de operare folosesc pentru excluderea zonei critice noțiunea de **monitor**.

Un exemplu de monitor in Java, folosind cuvântul cheie **synchronized**:

```
synchronized (ob) {
....
}
```

sau

```
public synchronized met(...){
...
}
```

Observatie: Atat obiectul **ob**, cat si functia **met** trebuie sa fie statice.

Threaduri (Fire de execuție)

- în cadrul unui proces se pot executa mai multe threaduri, care ruleaza în paralel.

Într-un sistem de operare, procesele pot fi: - monothread (UNIX)
- multithread (Windows)

În UNIX s-a implementat o bibiliotecă, numită **pthread**, care permite folosirea mai multor threaduri într-un proces.

Threadurile partajează variabilele globale și fac parte din același proces.

Problema celor 5 (N) filosofi

Noțiunea de deadlock

Acesta se produce, atunci când anumite obiecte pot lua anumite resurse pentru a le utiliza în mod exclusiv. Resursele respective sunt nepartajabile (trebuiesc eliberate de procesul care le deține, pentru a putea fi folosite mai departe de un alt proces).

ex.: $P_1 \rightarrow r_1$ (a se citi: „procesul P_1 achiziționează resursa r_1)
$P_2 \rightarrow r_2$
$P_1 \rightarrow r_2$ (P_1 va aștepta după P_2)
$P_2 \rightarrow r1$ (P_2 va aștepta după P_1)

și astfel s-a ajuns într-o situație de deadlock.

Sistemul de operare detectează astfel de situații și va alege unul din procese ca *victimă* (apelul sistem al acestui proces iese afară cu cod de eroare, semnificând că nu a reușit să achiziționeze resursa respectivă).
În UNIX codul de retur va fi **-1**, iar **errno** ia valoarea **EDEADLOCK.**

Observatie: Exista situații de așteptare (diferite de deadlock). În acestea sistemul de operare nu intervine.
De ex.: cu semafoare (care nu sunt privite ca resurse).

Un deadlock poate fi:

- **direct** (ca în exemplu);
- **indirect** (circular: P_1 așteapta după P_2, P_2 după P_3, ...; în această situație, un proces este ales victimă și astfel se „sparge” cercul);

Observatie: Pentru evitarea deadlockurilor se dorește eliberarea resurselor folosite cât mai repede posibil.

Criteriile alegerii procesului victimă de către sistemul de operare:

- în funcție de resursele consumate de fiecare proces în parte. (cel cu mai puține resurse va fi ales);
- ia în considerare și timpul CPU consumat (procesul victimă va fi cel cu durata de timp mai scăzută);

Problema celor 5 (N) filosofi

NU

```
#define N 5
void philosopher(int i){
        while(TRUE){
                think();
                take_fork();            /* apeluri blocante, va rămâne în așteptare dacă nu e liberă*/
                take_fork((i+1)%N);
                eat();
                put_fork(i);
                put_fork((i+1)%N);
        }
}
```

DA

```
#define N 5
#define LEFT (i+N-1)%N          /*indicele vecinului din stânga*/
#define RIGHT  (i+1)%N          /*indicele vecinului din dreapta*/
#define THINKING        0
#deifine HUNGRY         1
#define EATING          2

typedef int semaphore;
int state[N];                   /*variabile partajate cu valorile 0,1,2*/
semaphore mutex=1;              /*pentru excluderea zonei critice*/
semaphore s[N];                 /*vector de N semafoare inițializate cu 0*/

void philosopher(int i){
        while(TRUE){
                think();
                take_forks();           /*se iau ambele furculițe, dacă nu se trece mai departe*/
                eat();
                put_forks();
        }
}

void take_forks(int i){
        down(&mutex);
        state[i]=HUNGRY;
        test(i);
        up(&mutex);
        down(&s[i]);            /*semaforul rămâne pe 0, dacă nu a reușit să le ia*/
}

void put_forks(int i){
        down(&mutex);
        state[i]=THINKING;
        test(LEFT);
        test(RIGHT);
        up(&mutex);
```

```
}
```

Problema scriitorii-cititori

Primul proces	Al doilea proces
write(a);	read(a);
write(b);	read(b);

- zone critice (vezi cursul 4)

Problema frizerului somnoros

Într-o frizerie se află unul sau mai mulți frizeri şi clienți, simbolizând două tipuri de procese.
Regulamentul: - dacă nu e niciun client, frizerul doarme (dacă nu doarme, tunde);
- clienții dacă nu găsesc frizer liber (dormind), se aşază să aştepte pe scaun;
- avem **n** clienți şi **m** scaune;
- clienții dacă nu găsesc scaune libere, ies în afara frizeriei.

```
#define CHAIRS 5
typedef int semaphore;
semaphore customers=0;
semaphore barbers=0;
semaphore mutex=1;
int waiting=0;                  /*variabila indică numărul de clienți aflați în aşteptare pe scaun*/

void barber(){
        while(TRUE){
                down(&customers);
                down(&mutex);
                waiting--;
                up(&barbers);
                up(&mutex);
                cut_hair();
        }
}

void customer(){
        down(&mutex);
        if(waiting<CHAIRS){
                waiting++;
                up(&customers);
                up(&mutex);
                down(&barbers);
                get_haircut();
        }
        else
                up(&mutex);
}
```

Planificarea proceselor

Având o listă de procese, atunci când un proces *running* este întrerupt, urmează să continuăm cu un alt proces.

Criteriile de alegere a următorului proces:

- (vechi) FIFO;
- (modern) **priorităţile** proceselor;

Prioritatea este specificată de utilizator şi ajustată în timp de sistemul de operare.

UNIX: (comenzi din shell)

$com **-** procesul va fi lansat cu prioritate standard
$nice com - procesul va fi lansat cu prioritate mai scăzută
$nice –nr com - în plus, se specifică în **nr** numărul de unităţi cu care este scăzută prioritatea (altfel se scade un număr standard de unităţi). Din **root** se poate specifica o valoare negativă, pentru a lansa un proces cu prioritate mai ridicată decât cea standard.

Pe parcursul desfăşurării procesului, în funcţie de timpul CPU consumat şi resursele folosite, sistemul de operare poate hotărî modificarea (ajustarea) priorităţii.

- În UNIX se foloseşte în acest scop algoritmul Round Robin : lista de procese este partiţionată în funcţie de prioritate.
- Alte sisteme de operare folosesc *tehnici de prognoză* (prognoza se face pe baza execuţiilor anterioare ale procesului)