LUCRAREA 3

Excluderea mutuală

Resursa este un element de care are nevoie un *task* pentru a putea fi rulat. Resursele sunt de natură materială (procesor, memoria interna, dispozitivele de I/E, ceasul de timp real etc.) sau logică (subrutine, fișiere, variabile etc.).

Resursa locală — resursă accesabilă doar de către un singur *task*.

Resursa comună — resursă accesabilă de către mai multe *task*-uri.

Resursa critică — resursă comună care admite ca doar o sesiune de lucru asupra ei să fie în derulare la un moment dat.

Resursa reentrantă — resursă comună care admite ca oricâte sesiuni de lucru asupra ei să fie în derulare la un moment.

Secțiune critică — o porțiune de program prin care se realizează o sesiune de lucru asupra unei resurse critice (cel puțin 2 *task*-uri necesită aceeași resursă critică în același interval de timp).

Excluderea mutuală — excluderea de către un *task* aflat într-o secțiune critică referitoare la o resursă, a posibilității altor *task*-uri de a pătrunde în secțiuni critice care au ca obiect aceeași resursă.

Există mai multe mecanisme de realizare a excluderii mutuale;

- excluderea mutuală prin dezactivarea întreruperilor;
- excluderea mutuală prin fanioane de excluziune;
- excluderea mutuală prin semafoare;
- excluderea mutuală prin blocuri resursă.

Excluderea mutuală prin semafoare

Semaforul este un ansamblu format dintr-o variabilă întreagă I și o coadă de așteptare C. Asupra semafoarelor sunt definite două funcții, P și V, care se mai numesc și primitive și se execută în condiții de indivizibilitate. Prin intermediul lor se poate acționa asupra variabilelor I și C.

Primitiva P(S) constă în următoarele operații:

- I = I - 1;
- dacă I $\geq$ 0, funcția se încheie și *task*-ul în care ea se execută își continuă rularea;
- dacă I < 0, *task*-ul în care funcția se execută se blochează și este înscris în coada de așteptare C, provocând un proces de comutare pentru introducerea în rulare a unui alt *task*.

Primitiva V(S) constă în următoarele operații:

- I = I + 1;
- dacă I > 0, funcția se încheie și *task*-ul în care ea se execută își continuă rularea;

— dacă I ≤ 0, se deblochează *task*-ul din capul cozii C, extrăgându-se din ea și introducându-se în rândul *task*-urilor rulabile.

Variabila I are următoarea semnificație:

— dacă I > 0, I reprezintă numărul de *task*-uri care pot executa funcția P(S) fără a se bloca;

— dacă I ≤ 0, | I | reprezintă numărul *task*-urilor blocate la semaforul S și aflate în coada de așteptare a acestuia, C.

Coada C, în cazul de față, este gestionată după principiul FIFO.

Pentru implementarea semafoarelor se introduce tipul de date, `SEMAPHORE`, cu următoarea structură:

```
typedef struct{
     uschar contor;
     uschar coada[MAX_TSK];
     uschar *pi;
     uschar *pe;
     uschar valinit;
}SEMAPHORE;
```

unde: `contor` — este contorul semaforului

`coada` — este coada de așteptare la semafor

`pi` — este *pointer*-ul de intrare; prin intermediul lui se introduc elementele în coadă

`pe` — este *pointer*-ul de ieșire; prin intermediul lui se extrag elementele din coadă

`valinit` — este valoarea inițială a contorului

Toate semafoarele sunt grupate într-un tablou:

```
#define MAX_SEM 10
SEMAPHORE _sem[MAX_SEM];
```

Un semafor este accesat prin indicele său din acest tablou. Funcțiile care vor opera asupra semafoarelor vor fi:

```
#define SEM_IDENT uschar
void _s_init(void);
void s_create(SEM_IDENT *semidadr, uschar initval);
#define _s_create(semid) s_create(&semid)
void _s_destroy(SEM_IDENT semid);
void _s_signal(SEM_IDENT semid);
void _s_wait(SEM_IDENT semid, usint timeout);
```

Funcția `_s_init()` inițializează tabloul de semafoare prin stabilirea valorii `NULL` a componentei `pi` a tuturor elementelor sale.

Funcția `s_create()`:

— identifică un element liber al tabloului de semafoare;

— atribuie componentelor `contor` și `valinit` valoarea `initval;`

— poziționează componentele `pi` și `pe` către primul element al cozii;

— returnează, prin parametrul `semidadr`, indicele elementului de tablou alocat.

Funcția `_s_wait()` implementează funcția P(S):

- se decrementează contorul;
- dacă acesta este mai mic decât 0:
    - se înscrie *task*-ul curent în coada de așteptare în poziția indicată de `pi`;
    - dacă `pi` indică ultimul element din coadă, se actualizează valoarea acestuia astfel încât să indice primul element din tablou; altfel, se incrementează `pi`;
    - se blochează *task*-ul curent (apelând funcția `sleep()` cu parametrul `time-out`).

Funcția `_s_signal()` implementează funcția V(S):

- se incrementează contorul;
- dacă acesta este mai mic sau egal cu 0:
    - se determină indicele *task*-ului aflat în capul cozii de așteptare (cel indicat de pointerul de ieșire `pe`);
    - dacă acesta se află pe ultima poziție din tablou, pointer-ul `pe` va indica primul element al tabloului; altfel, se incrementează `pe`;
    - se deblochează acest *task* (apelând funcția `wakeup()`).

Funcția `_s_destroy()`:

- dezalocă (dacă i se permite) elementul cu indicele specificat; adică `pi` se readuce la valoarea inițială NULL (elementul este pus la dispoziția funcției `s_create` pentru o nouă alocare);

Cerințele lucrării:

Să se implementeze funcțiile legate de gestionarea semafoarelor.