Semáforos em Unix

→ Criar um conjunto (vetor) de semáforos:

```
semid = semget ( chave, nsems, flag);
```

onde:

- `semid` identificador de acesso ao vetor de semáforos.
- `chave` identificador global que identifica este vetor de semáforos; se a chave for conhecida por outro processo, este pode com a execução de um novo `semget()` e mesma chave, ganhar acesso aos mesmos semáforos.
- `nsems` número de semáforos a criar no vetor (tamanho do vetor);
- `flag` condicionam o modo de uso dos semáforos: `IPC_CREAT | 0644`, cria novos com permissões `0644`; se for `NULL`, obtém um vetor já existente.

O vetor começa na posição zero (0).

Semáforos em Unix

→ Remover um vetor de semáforos:

```
semctl ( semid, semnum, comando);
```

com `comando = IPC_RMID`. Neste caso, o argumento `semnum` é ignorado.

`semctl` pode ser usado com outros comandos diferentes de IPC_RMID.

Operações sobre Semáforos em Unix

```
status = semop (semid, semops, nsemops);
```

onde:

- `semid` identificador de acesso ao vetor de semáforos.
- `semops`: vetor de estruturas que definem as operações sobre os semáforos;
- `nsemops` número de semáforos que vão ser afetados pelas operações definidas no vetor de estruturas.

  Cada entrada do vetor de estruturas é do tipo:

```
1 struct sembuf {
2   short semNum; /* num. sem. afetado por semOp */
3   short semOp;  /* operação sobre semáforo      */
4   short SemFlag;
5 }
```

- Valores `semOp<0` implicam bloqueio do processo se houver mais algum processo usando a região crítica associada àquele semáforo OU acesso à região crítica; valores `semOp>0` libertam o semáforo.

→ Semáforos binários (tomam apenas valores 0 e 1):

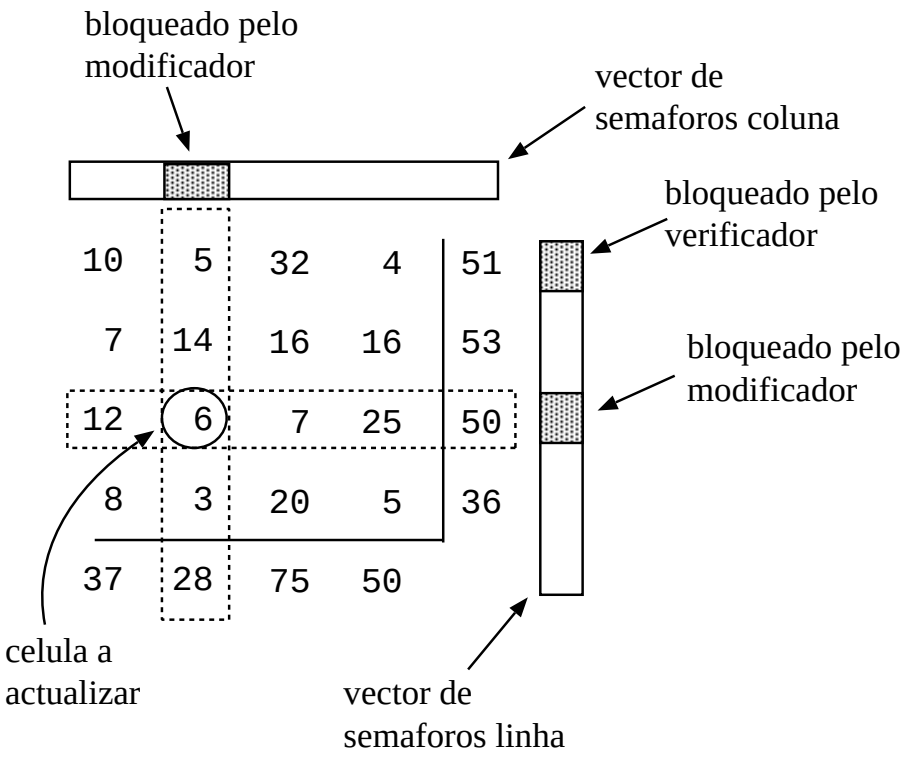
```
1 #define UP(sid,n) {                    \
2             struct sembuf up={n,1,0};\
3             semop(sid, &up, 1);       \
4 }
5 #define DOWN(sid,n) {                     \
6             struct sembuf down={n,-1,0};\
7             semop(sid, &down, 1);        \
8 }
9 #define INIT(sid,n) UP(sid,n)
```

Exemplo com “shm” e “sem”: Folha de Cálculo

Consideremos uma matriz de N linhas e M colunas, em que cada elemento da última linha é a soma dos elementos da mesma coluna e cada elemento da última coluna é a soma dos elementos da mesma linha.

Suponhamos que temos dois processos:

- um *modificador*: periodicamente modifica, de forma aleatória, o valor de uma célula da matriz e atualiza os totais na linha e coluna que contêm a célula.
- um *verificador*: periodicamente escreve a matriz, e verifica se os totais estão corretos.


bloqueado pelo
modificador
vector de
semaforos coluna
bloqueado pelo
verificador
bloqueado pelo
modificador
10 5 32 4 51
7 14 16 16 53
12 6 7 25 50
8 3 20 5 36
37 28 75 50
celula a
actualizar
vector de
semaforos linha

Folha de Cálculo (cont.)

Potenciais dificuldades: modificador e verificador podem estar a usar valores da mesma linha ou coluna e ocasionar problemas de concorrência.

```
1  VERIFICADOR:
2    para todas as linhas {
3      entrar na zona crítica
4        calcula soma de todas células na linha
5        se (soma != total na folha)
6           escreve mensagem
7      sair da zona crítica
8    }
9    para todas as colunas {
10     entrar na zona crítica
11       calcula soma de todas células na coluna
12       se (soma != total na folha)
13          escreve mensagem
14     sair da zona crítica
15   }
```

```
MODIFICADOR:
  escolhe aleatoriamente
        1 célula e 1 valor
  entrar na zona crítica
    mod célula com novo valor
    atualiza total na linha
    atualiza total na coluna
  sair da zona crítica
```

Folha de Cálculo (cont.)

```
1  #include <stdio.h>
2  #include <sys/types.h>
3  #include <sys/ipc.h>                    /* VARIAVEIS GLOBAIS */
4  #include <sys/shm.h>                    int totalLin, totalCol;
5  #include <sys/sem.h>                    int linSems, colSems;
6
7  /* CONSTANTES */                        /* PROTOTIPOS DAS FUNCOES */
8  #define NLINS  8                        void gera_nova_entrada(int *);
9  #define NCOLS  8                        void escreve_e_verifica(int *);
10 #define SKEY 123                        int  inicia_sems(key_t, int);
11 #define SSIZE  NLINS*NCOLS*sizeof(int)
12
13 /* MACROS */
14 #define CELL(s,r,c)  (*((s)+((r)*NCOLS)+(c)))
15
16 /* operações sobre semáforos */
17 #define UP(sid,n) {                  \     #define INIT(sid,n) UP(sid,n)
18            struct sembuf up={n,1,0};\
19            semop(sid, &up, 1);      \
20 }
21 #define DOWN(sid,n) {                \
22            struct sembuf down={n,-1,0};\
23            semop(sid, &down, 1);    \
24 }
```

Folha de Cálculo (cont.)

```
1  int main()
2  {
3    int id, lin, col, *folha, i=0, j=0;
4
5    setbuf(stdout,NULL);  /* evita buffering */
6    totalLin= NLINS-1;
7    totalCol= NCOLS-1;
8    /* seg. mem. partilhada para a matriz */
9    id = shmget(SKEY, SSIZE, IPC_CREAT|0600);
10   folha = (int *) shmat(id,0,0);
11   for (lin=0; lin < NLINS; lin++) /* celulas a zero */
12     for (col=0; col<NCOLS; col++)
13       CELL(folha, lin, col)=0;
14                          /* cria e inicia vecs de sems */
15   linSems= inicia_sems(SKEY, NLINS);
16   colSems= inicia_sems(SKEY+1, NCOLS);
17   if (fork()) {          /* pai escreve e verifica*/
18      escreve_e_verifica(folha);
19   }
20   else {                 /* filho gera valores */
21      gera_nova_entrada(folha);
22      exit(0);
23   }
```

Folha de Cálculo (cont.)

```
1    wait(0);
2    semctl(linSems,0,IPC_RMID);/* liberta sems*/
3    semctl(colSems,0,IPC_RMID);
4    shmdt(folha);               /* liberta shm */
5    shmctl(id,IPC_RMID, 0);
6    exit(0);
7  } // fim do main
```

Folha de Cálculo (cont.)

```
1  int inicia_sems(key_t k, int n)
2  {
3    int semid, i;
4
5    if ((semid=semget(k,n,0))!=-1)  /* se ja existem */
6      semctl(semid,0,IPC_RMID);     /* liberta-os    */
7                                    /* cria novos    */
8    if ((semid=semget(k,n,IPC_CREAT|0600))!=-1)
9      for (i=0; i<n; i++)   /* inicia sems do vetor */
10       INIT(semid,i);
11   return semid;
12 }
```

Folha de Cálculo (cont.)

```
1  void gera_nova_entrada(int *s)
2  {
3    int lin, col, old, new;
4
5    /* escolhe aleatoriamente celula e novo valor */
6    lin= rand() % (NLINS-1);
7    col= rand() % (NCOLS-1);
8    new= rand() % 1000;
9    /* tenta entrar na zona critica */
10   DOWN(linSems, lin);
11     DOWN(colSems, col);
12      old= CELL(s, lin, col); /* atualiza celula e totais */
13      CELL(s, lin, col)= new;
14      CELL(s, lin, totalCol) += (new-old);
15      CELL(s, totalLin, col) += (new-old);
16      UP(colSems, col); /* sai da zona critica */
17   UP(linSems, lin);
18 }
```

Folha de Cálculo (cont.)

```
1  void escreve_e_verifica(int *s)
2  {
3    int lin, col, soma, totalErrs;
4
5    totalErrs=0;
6    for (lin=0; lin<NLINS; lin++) {
7      soma= 0;
8      DOWN(linSems, lin);
9      for (col=0; col<NCOLS; col++) {
10       if (col != totalCol)
11         soma += CELL(s, lin, col);
12       printf("%5d", CELL(s,lin,col));
13     }
14     if (lin!= totalLin)
15       totalErrs += (soma != CELL(s, lin, totalCol));
16     UP(linSems, lin);
17     printf("\n");
18   }
```

Folha de Cálculo (cont.)

```
1   for (col=0; col<totalCol; col++) {
2     soma=0;
3     DOWN(colSems,col);
4     for (lin=0; lin<totalLin; lin++)
5       soma += CELL(s,lin, col);
6     totalErrs += (soma != CELL(s, totalLin, col));
7     UP(colSems, col);
8   }
9   if (totalErrs)
10    printf("\nFolhaCalculo falhou\n");
11 }
```