Semáforos em Unix

→ Criar um vetor de semáforos:

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

onde:

- `semid` identificador de acesso ao semáforo.
- `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;
- `flag` condicionam o modo de uso do semáforo: `IPC_CREAT | 0644`, cria novo com permissões `0644`; se forem `NULL`, para obter um vetor já existente.

Semáforos em Unix

→ Remover um vetor de semáforos:

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

com `comando = IPC_RMID`. Outros comandos permitem usar esta função com outro significado.

Operações sobre Semáforos em Unix

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

onde:

- `semid` identificador de acesso ao semáforo.
- `nsemops` número de estruturas na tabela.
- `semops`: tabela de estruturas que definem as operações sobre os semáforos;

```
struct sembuf {
  short semNum; /* num. sem. afectado por semOp */
  short semOp;  /* operação sobre semáforo      */
  short SemFlag;
}
```

- Valores `semOp<0` implicam espera do processo (se resultado igual a zero) e lock do semáforo; , valores `semOp>0` libertam o semáforo.

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

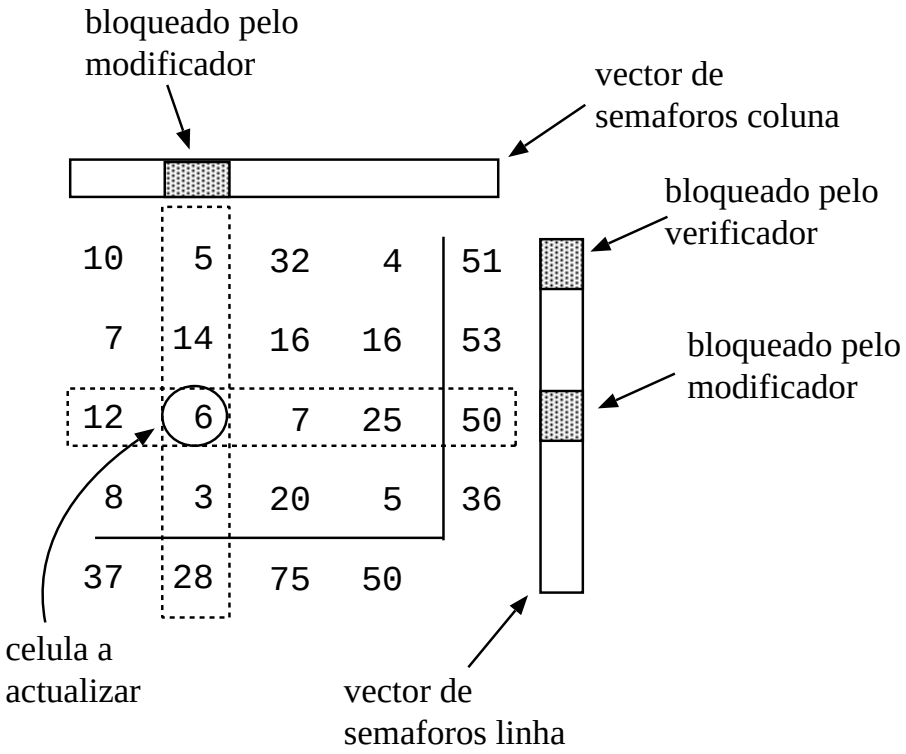
```
#define UP(sid,n) {                    \
            struct sembuf up={n,1,0};\
            semop(sid, &up, 1);        \
}
#define DOWN(sid,n) {                      \
            struct sembuf down={n,-1,0};\
            semop(sid, &down, 1);          \
}
#define INIT(sid,n) UP(sid,n)
```

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

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


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.

```
VERIFICADOR:
  para todas as linhas {
    entrar na zona crítica
      calcula soma de todas células na linha
      se (soma != total na folha)
         escreve mensagem
    sair da zona crítica
  }
  para todas as colunas {
    entrar na zona crítica
      calcula soma de todas células na coluna
      se (soma != total na folha)
         escreve mensagem
    sair da zona crítica
  }
```

```
MODIFICADOR:
  escolhe aleatoriamente
        1a 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.)

```
#include <stdio.h>                      #define INIT(sid,n) UP(sid,n)
#include <sys/types.h>
#include <sys/ipc.h>                    /* VARIAVEIS GLOBAIS */
#include <sys/shm.h>                    int totalLin, totalCol; /* totais na linha e coluna */
#include <sys/sem.h>                    int linSems, colSems;   /* id vetores de semaforos */

/* CONSTANTES */                        /* PROTOTIPOS DAS FUNCOES */
#define NLINS  8                        void gera_nova_entrada(int *);
#define NCOLS  8                        void escreve_e_verifica(int *);
#define SKEY 123                        int  inicia_sems(key_t, int);
#define SSIZE  NLINS*NCOLS*sizeof(int)

/* MACROS */
#define CELL(s,r,c)  (*((s)+((r)*NCOLS)+(c)))

/* operações sobre semáforos */
#define UP(sid,n) {                   \
            struct sembuf up={n,1,0};\
            semop(sid, &up, 1);      \
}
#define DOWN(sid,n) {                   \
            struct sembuf down={n,-1,0};\
            semop(sid, &down, 1);       \
}
```

Folha de Cálculo (cont.)

```
int main()
{
  int id, lin, col, *folha, i=0, j=0;

  setbuf(stdout,NULL);  /* evita buffering */
  totalLin= NLINS-1;
  totalCol= NCOLS-1;
  /* seg. mem. partilhada para a matriz */
  id= shmget(SKEY, SSIZE,IPC_CREAT|0600);
  folha= (int *) shmat(id,0,0);
  for (lin=0; lin < NLINS; lin++) /* celulas a zero */
    for (col=0; col<NCOLS; col++)
      CELL(folha, lin, col)=0;
                          /* cria e inicia vecs de sems */
  linSems= inicia_sems(SKEY, NLINS);
  colSems= inicia_sems(SKEY+1, NCOLS);
  if (fork()) {           /* pai escreve e verifica*/
    for (;;) escreve_e_verifica(folha);
  }
  else {                  /* filho gera valores */
     for (;;) gera_nova_entrada(folha);
     exit(0);
  }
}
```

```
                                        wait(0);
                                        semctl(linSems,0,IPC_RMID);/* liberta sems*/
                                        semctl(colSems,0,IPC_RMID);
                                        shmdt(folha);              /* liberta shm */
                                        shmctl(id,IPC_RMID, 0);
                                        exit(0);
```

Folha de Cálculo (cont.)

```
int inicia_sems(key_t k, int n)
{
  int semid, i;

  if ((semid=semget(k,n,0))!=-1)  /* se ja existem */
    semctl(semid,0,IPC_RMID);     /* liberta-os    */
                                  /* cria novos    */
  if ((semid=semget(k,n,IPC_CREAT|0600))!=-1)
    for (i=0; i<n; i++)   /* inicia sems do vetor */
      INIT(semid,i);
  return semid;
}
void gera_nova_entrada(int *s)
{
  int lin, col, old, new;

  /* escolhe aleatoriamente celula e novo valor */
  lin= rand() % (NLINS-1);
  col= rand() % (NCOLS-1);
  new= rand() % 1000;
                    /* tenta entrar na zona critica */
  DOWN(linSems, lin);
  DOWN(colSems, col);
}
```

```
old= CELL(s, lin, col); /* atualiza celula e totais */
CELL(s, lin, col)= new;
CELL(s, lin, totalCol) += (new-old);
CELL(s, totalLin, col) += (new-old);
UP(colSems, col); /* sai da zona critica */
UP(linSems, lin);
usleep(5000);
```

Folha de Cálculo (cont.)

```
void escreve_e_verifica(int *s)
{
  int lin, col, soma, totalErrs;
  static int ctr= 0;

  totalErrs=0;
  ctr++;
  for (lin=0; lin<NLINS; lin++) {
    soma= 0;
    DOWN(linSems, lin);
    for (col=0; col<NCOLS; col++) {
      if (col != totalCol)
        soma += CELL(s, lin, col);
      printf("%5d", CELL(s,lin,col));
    }
    if (lin!= totalLin)
      totalErrs += (soma != CELL(s, lin, totalCol));  }
    UP(linSems, lin);
    printf("\n");
  }
  for (col=0; col<totalCol; col++) {
    soma=0;
    DOWN(colSems,col);
    for (lin=0; lin<totalLin; lin++)
      soma += CELL(s,lin, col);
    totalErrs += (soma != CELL(s, totalLin, col));
    UP(colSems, col);
  }
  if (totalErrs)
    printf("\nFolhaCalculo n(%d) falhou\n",ctr);
  if ((ctr % 20) == 0)
    printf("\nFolhaCalculo n(%d) processada\n",ctr);
  printf("\n-------------------------------------\n");
  sleep(2);
```