Problemas sobre Programação Concorrente

Produtor / Consumidor

```
/*+-------------------------------------------------------------------+
  | Programa 1  : Produtor / Consumidor                               |
  | Descricao   : Algoritmo Produtor / Consumidor .                   |
  | O buffer para transmissao de dados situa-se numa zona de memoria  |
  | partilhada entre os dois processos.                               |
  | Esse buffer tem SIZE_OF_BUFFER posicoes.                          |
  -------------------------------------------------------------------+*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "my_sem.h"

#define SHMKEY IPC_PRIVATE

#define SIZE_OF_BUFFER 10
#define MUTEX 0
#define FULL 1
#define EMPTY 2

int semid;

int *pont_int;

int write_pos = 0;
int read_pos  = 0;

int shmid;

extern char *shmat();
int cleanup();
void produz(),coloca(),consome(),retira();

main()
{
int a,b,i;
char *adr;

for(i=0;i<20;i++)               /* redirecciona os signals */
        signal(i,cleanup);
                                /* cria memoria partilhada */
shmid = shmget(SHMKEY,SIZE_OF_BUFFER*sizeof(int),0777|IPC_CREAT);

adr = shmat(shmid,0,0);         /* attach memory */
pont_int = (int *)adr;

// Cria os 3 semaforos: MUTEX, FULL, EMPTY
semid=sem_create(3,0);
sem_setvalue(semid,MUTEX,1);
sem_setvalue(semid,FULL,0);
sem_setvalue(semid,EMPTY,SIZE_OF_BUFFER);

if(semid == -1) {
            printf("Erro a criar semaforos");
            cleanup();
            }

printf("-------------------- PRODUTOR / CONSUMIDOR ! -------------------\n");
if (fork() == 0){
```

```
        while(1){               /* proc. filho -> produtor */
                produz(&a);
                sem_wait(semid,EMPTY);
                sem_wait(semid,MUTEX);
                coloca(a);
                sem_signal(semid,MUTEX);
                sem_signal(semid,FULL);
                sleep(2);
                }
        }
while(1){                       /* proc. pai -> consumidor */
        sleep(1);
        sem_wait(semid,FULL);
        sem_wait(semid,MUTEX);
        retira(&b);
        sem_signal(semid,MUTEX);
        sem_signal(semid,EMPTY);
        consome(b);
        sleep(3);
        }
}
/*-------------------------------------------------------------------------*/
cleanup()
{
        sem_rm(semid);
        shmctl(shmid,IPC_RMID,0);       /* remove a shared memory */
        exit();
}
/*-------------------------------------------------------------------------*/
void produz(a)
int *a;
{
static int i = 0 ;

i++ ;
if(i>30) i=1 ;
printf("PRODUTOR : produziu o valor %d \n",i);
*a = i;
}
/*-------------------------------------------------------------------------*/
void coloca(a)
int a;
{
int k;
printf("PRODUTOR  : coloca valor %d na posicao %d \n",a,write_pos);
*(pont_int + write_pos) = a ;
write_pos++;
write_pos %= SIZE_OF_BUFFER ;
printf("BUFFER_PROD = ");
for(k=0;k<SIZE_OF_BUFFER;k++)
        printf("%d ",*(pont_int + k));
printf("\n");
}
/*-------------------------------------------------------------------------*/
void retira(b)
int *b;
{
int k;
printf("BUFFER_CONS = ");
for(k=0;k<SIZE_OF_BUFFER;k++)
        printf("%d ",*(pont_int + k));
printf("\n");

*b = *(pont_int + read_pos);
printf("CONSUMIDOR : retira valor %d da posicao %d \n",*b,read_pos);
read_pos ++;
read_pos %= SIZE_OF_BUFFER ;
}
/*-------------------------------------------------------------------------*/
void consome(b)
int b;
```

```
{
printf("CONSUMIDOR : consumiu o valor %d \n",b);
}
/*------------------------------------------------------------------------*/
```

Leitores / Escritores
(v1: leitores têm prioridade)

```
/*+-----------------------------------------------------------------------+
  | Programa V1 : Leitores / Escritores                                    |
  | Descricao  : Algoritmo Leitores / Escritores .                         |
  | Nesta solucao , os Leitores tem prioridade .                           |
  | A variavel "readcount" encontra-se em memoria partilhada pelos        |
  | processos, assim como o buffer de leitura/escrita .                    |
  +-----------------------------------------------------------------------+*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "my_sem.h"

#define SHMKEY IPC_PRIVATE
#define MUTEX 0
#define WRIT  1

int semid;
int shmid;

extern cleanup();
extern char *shmat();

char *str = "HELLO, READERS !... ";

main()
{
int i;
char *adr;
int  *readcount;
char *buffer;

for(i=0;i<20;i++)             /* redirecciona os signals */
       signal(i,cleanup);
                              /* cria regiao memoria partilhada */
shmid = shmget(SHMKEY,128,0777|IPC_CREAT);

adr = shmat(shmid,0,0);               /* attach memory */
readcount = (int *)adr;
buffer = (char *)(adr+1);

*readcount = 0;

/* cria 2 semaforos */
semid=sem_create(2,1);
//sem_setvalue(semid,MUTEX,1);
//sem_setvalue(semid,WRIT,1);
if(semid == -1){
         printf("Erro ao criar os semaforos\n");
         cleanup();
        }

printf("------------------- READERS & WRITERS -------------------\n");
printf("--------------- Leitores tem prioridade ! ---------------\n");

if (fork() == 0){
       fork();                        /* 4 processos Leitores */
```

```
        fork();
                while(1){
                sem_wait(semid,MUTEX);
                *readcount ++;
                if(*readcount == 1)
                        sem_wait(semid,WRIT);
         sem_signal(semid,MUTEX);

                printf("LEITOR %d : leu do buffer : %s\n",getpid(),buffer);

         sem_wait(semid,MUTEX);
                *readcount --;
                if(*readcount == 0)
             sem_signal(semid,WRIT);
            sem_signal(semid,MUTEX);
                sleep(1);
                }
        }
while(1){               /* processo Escritor */
        sem_wait(semid,WRIT);
        while(*str)
                *buffer++ = *str++;    /* escreve string */
        *buffer = '\0';
        printf("ESCRITOR : escreveu texto no buffer \n");
        sem_signal(semid,WRIT);
        sleep(3);
        }
}
/*-------------------------------------------------------------------------*/
cleanup()
{
    sem_rm(semid);
        semctl(shmid,IPC_RMID,0);      /* e a memoria partilhada */
        exit();
}
/*-------------------------------------------------------------------------*/
```

Leitores / Escritores (v2: escritores têm prioridade)

```
/*+-------------------------------------------------------------------------+
  | Programa V2 : Leitores / Escritores                                      |
  | Descricao  : Algoritmo Leitores / Escritores .                          |
  | Nesta solucao , os Escritores tem prioridade .                          |
  | As variaveis "readcount" e "writecount" encontram-se em memoria         |
  | partilhada pelos processos, assim como o buffer de leitura/escrita.     |
  +-------------------------------------------------------------------------+*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "my_sem.h"

#define SHMKEY IPC_PRIVATE
#define MUTEX1 0
#define MUTEX2 1
#define MUTEX3 2
#define WR     3
#define RD     4

int semid;
int shmid;

int cleanup();
```

```
extern char *shmat();

char *str = "HELLO, READERS !... ";

main()
{
int i;
int  *adr;
int  *readcount;
int  *writecount;
char *buffer;

for(i=0;i<20;i++)             /* redirecciona os signals */
       signal(i,cleanup);
                              /* cria regiao memoria partilhada */
shmid = shmget(SHMKEY,128,0777|IPC_CREAT);

adr = (int *)shmat(shmid,0,0);               /* attach memory */
readcount = (int *)adr;
writecount = (int *)(adr+1);
buffer = (char *)(adr+2);

*readcount  = 0;
*writecount = 0;

/* cria 5 semaforos */
semid= sem_create(5,1);
//sem_setvalue(semid,MUTEX1,1);
//sem_setvalue(semid,MUTEX2,1);
//sem_setvalue(semid,MUTEX3,1);
//sem_setvalue(semid,WR,1);
//sem_setvalue(semid,RD,1);
if(semid == -1){
         printf("Erro ao criar os semaforos\n");
         cleanup();
         }

printf("---------------------- READERS & WRITERS ----------------------\n");
printf("------------------ Escritores tem prioridade ! ---------------\n");

if (fork() == 0){
       fork();                        /* 4 processos Leitores */
       fork();
        while(1){
         sem_wait(semid,MUTEX3);
         sem_wait(semid,RD);
         sem_wait(semid,MUTEX1);
         *readcount ++;
          if(*readcount == 1)
                   sem_wait(semid,WR);
         sem_signal(semid,MUTEX1);
         sem_signal(semid,RD);
         sem_signal(semid,MUTEX3);

               printf("LEITOR %d : leu do buffer : %s\n",getpid(),buffer);

        sem_wait(semid,MUTEX1);
               *readcount --;
               if(*readcount == 0)
                       sem_signal(semid,WR);
               sem_signal(semid,MUTEX1);

               sleep(2);
               }
       }
fork();
while(1){              /*  2 processos Escritores */
       sem_wait(semid,MUTEX2);
       *writecount ++;
       if(*writecount == 1)
               sem_wait(semid,RD);
```

```
        sem_signal(semid,MUTEX2);

        sem_wait(semid,WR);
        while(*str)
                *buffer++ = *str++;    /* escreve string */
        *buffer = '\0';
        printf("ESCRITOR %d : escreveu texto no buffer \n",getpid());
        sem_signal(semid,WR);

        sem_wait(semid,MUTEX2);
        *writecount --;
        if(*writecount == 0)
                sem_signal(semid,RD);
        sem_signal(semid,MUTEX2);
        sleep(3);
        }
}
/*-------------------------------------------------------------------------*/
cleanup()
{
        sem_rm(semid);
        semctl(shmid,IPC_RMID,0);     /* e a memoria partilhada */
        exit();
}
/*-------------------------------------------------------------------------*/
```

Buffer Cleaner (v1) [shm+sem]

```
/*+-----------------------------------------------------------------------+
  | Programa  (V1) : Buffer Cleaner                                       |
  | Descricao   : Existem N processos que leem dados de um periferico     |
  | virtual e depositam esses dados num buffer comum.                     |
  | Existe ainda outro processo que espera que o buffer fique cheio       |
  | para o despejar, permitindo assim novo preenchimento.                 |
  | Neste caso N = 3 .                                                    |
  | Alem do semaforo MUTEX existem ainda os semaforos FULL e EMPTY.       |
  | O buffer de dados assim como o seu index ficam em shared memory.      |
  +-----------------------------------------------------------------------+*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "my_sem.h"

#define SHMKEY IPC_PRIVATE
#define SIZE_OF_BUFFER 10
#define MUTEX 0
#define FULL 1
#define EMPTY 2

int semid;
int shmid;
int *buffer;
int *indice;

extern char *shmat();
int cleanup();

main()
{
int   i,dado,k;
int   chr;
int   *adr;

for(i=0;i<20;i++)               /* redirecciona os signals */
        signal(i,cleanup);
```

```
                                /* cria regiao memoria partilhada */

shmid = shmget(SHMKEY,(SIZE_OF_BUFFER+1)*sizeof(int),0777|IPC_CREAT);

if(shmid == -1){
         printf("Erro ao criar a Shared Memory\n");
         cleanup();
         }

adr    = (int *)shmat(shmid,0,0);
indice = (int *)adr;
buffer = (int *)(adr + 1) ;

*indice = 0;

for(i=0;i<SIZE_OF_BUFFER;i++)
    *(buffer + i) = 0 ;


/* Cria Semaforos */
semid=sem_create(3,0);
sem_setvalue(semid,MUTEX,1);
// setvalue(FULL,0)
// setvalue(EMPTY,0)

if(semid == -1){
          printf("Nao conseguiu criar semaforos\n");
          cleanup();
          }


printf("------------------------ BUFFER CLEANER ! ------------------------\n");

if (fork() == 0){               /* processo cleaner */
        while(1){

                sem_wait(semid,FULL);  /* wait till is FULL */

                printf("*** Buffer = ");
                for (i=0;i < SIZE_OF_BUFFER;i++){
                        printf("%d ",*(buffer + i));
                        *(buffer + i) = 0;
                        *indice = 0;
                        }
                printf(" ****\n");

                sleep(1);
                printf("processo cleaner \n");
                sem_signal(semid,EMPTY);  /* say it is already EMPTY */
                }
        }

for(i=0;i<2;i++){               /* cria mais 2 processos */
        if(fork() == 0) break;
        }

printf("Processo com pid=%d\n", getpid());
fflush(stdout);
dado = i+1;

while(1){                       /*  3 processos trabalhadores */
        sleep(1);

        sem_wait(semid,MUTEX);

        k= *indice;
        *(buffer + k) = dado ;
        (*indice)  ++;
        printf("processo %d colocou valor %d \n",i+1,i+1);
```

```
        if((*indice) == SIZE_OF_BUFFER){
                sem_signal(semid, FULL);  // say its FULL
                sem_wait(semid, EMPTY); // wait for EMPTY
                }

        sem_signal(semid,MUTEX);
        }
}
/*-------------------------------------------------------------------------*/
cleanup()
{
        printf("CLEANUP\n");

        sem_rm(semid);                  /* remove os semaforos */
        shmctl(shmid,IPC_RMID,0);       /* remove shared memory */
        exit();
}
/*-------------------------------------------------------------------------*/
```

Buffer Cleaner (v2) [shm+msg para synch]

```
/*+-------------------------------------------------------------------------+
  | Programa    : Buffer Cleaner  (v2)                                      |
  | Descricao   : Existem N processos que leem dados de um periferico       |
  | virtual e depositam esses dados num buffer comum.                       |
  | Existe ainda outro processo que espera que o buffer fique cheio         |
  | para o despejar, permitindo assim novo preenchimento.                   |
  | Neste caso N = 3 .                                                      |
  | Os semaforos FULL e EMPTY sao substituidos por mensagens.               |
  | O buffer de dados assim como o seu index ficam em shared memory.        |
  +-------------------------------------------------------------------------+*/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/shm.h>
#include "my_sem.h"

#define SHMKEY 1000
#define MSGKEY 1000

#define FULL  1
#define EMPTY 2
#define SIZE_OF_BUFFER 10

#define MUTEX 0

struct mymsg{
        long    mtype;
        short   mshort;
        char    mdata[10];
        };

int semid;
int shmid;
int msgqid;
int *buffer;
int *indice;

struct mymsg msg_FULL,msg_EMPTY,msg;

extern char *shmat();
int cleanup();

main()
{
int   i,dado,k;
int   chr;
int   *adr;
```

```
for(i=0;i<20;i++)               /* redirecciona os signals */
        signal(i,cleanup);

                                /* cria regiao memoria partilhada */
shmid = shmget(SHMKEY,(SIZE_OF_BUFFER+1)*sizeof(int),0777|IPC_CREAT);

if(shmid == -1){
        printf("Nao conseguiu criar fila msg\n");
        cleanup();
        }

adr    = (int *)shmat(shmid,0,0);               /* attach memory */
indice = (int *)adr;
buffer = (int *)(adr + 1) ;

*indice = 0;
for(i=0;i<SIZE_OF_BUFFER;i++)           /* inicializa buffer */
        *(buffer + i) = 0 ;

msgqid = msgget(MSGKEY,0777|IPC_CREAT);     /* cria fila de mensagens */

msg_FULL.mtype  = 1 ;           /* define apenas os tipos das mensagens */
msg_EMPTY.mtype = 2 ;

if(msgqid == -1){
        printf("Nao conseguiu criar fila msg\n");
        cleanup();
        }

semid= sem_create(1,1);
// sem_setvalue(semid,MUTEX,1)
if(semid == -1){
         printf("Nao conseguiu criar o semaforo\n");
         cleanup();
         }

printf("------------------------ BUFFER CLEANER ! ------------------------\n");

if (fork() == 0){               /* processo cleaner */
        while(1){
                msgrcv(msgqid,&msg,256,FULL,0);  /* espera por FULL */

                printf("************************* Buffer = ");
                for (i=0;i < SIZE_OF_BUFFER;i++){
                        printf("%d ",*(buffer + i));
                        *(buffer + i) = 0;
                        *indice = 0;
                        }
                printf("\n");
                sleep(1);

                msgsnd(msgqid,&msg_EMPTY,sizeof(int),0);  /* envia EMPTY */
                }
        }

for(i=0;i<2;i++){               /* cria mais 2 processos */
        if(fork() == 0) break;
        }

while(1){                       /*  3 processos trabalhadores */
        sleep(1);
        dado = i+1;

        sem_wait(semid,MUTEX);
        k= *indice;
        *(buffer + k) = dado ;
        (*indice)  ++;
        printf("processo %d colocou valor %d \n",i+1,i+1);
        if((*indice) == SIZE_OF_BUFFER){
                msgsnd(msgqid,&msg_FULL,sizeof(int),0);  /* envia FULL */
```

```
                msgrcv(msgqid,&msg,256,EMPTY,0); /* espera por EMPTY */
                }
        sem_signal(semid,MUTEX);
        }
}
/*--------------------------------------------------------------------------*/
cleanup()
{
        sem_rm(semid);                  /* remove o semaforo */
        shmctl(shmid,IPC_RMID,0);       /* remove shared memory */
        msgctl(msgqid,IPC_RMID,0);      /* remove fila de mensagens */
        exit();
}
/*--------------------------------------------------------------------------*/
```

O Jantar dos Filósofos.

```
/*+-------------------------------------------------------------------------+
  | Programa    : O Jantar dos Filosofos                                    |
  | Descricao   : O problema do Jantar dos Filosofos.                       |
  +-------------------------------------------------------------------------+*/
#include <sys/types.h>
#include <sys/ipc.h>
#include "my_sem.h"
#include <sys/shm.h>
#include <errno.h>

#define MUTEX 5
// os outros 5 semaforos vao de 0 a 4
// os semaforos representam cada um dos recursos

int semid;

int cleanup();
void obtem_chopsticks();
void liberta_chopsticks();
//=========================================================00
main()
{
int i;
int num;

for(i=0;i<20;i++)               /* redirecciona os signals */
        signal(i,cleanup);

// cria 6 semaforos
semid=sem_create(6,1);
if(semid == -1){
         printf("Nao conseguiu criar os semaforos\n");
         cleanup();
         }

printf("--------------- O JANTAR DOS FILOSOFOS ! --------------- \n");

for(num=0;num<4;num++){
        if(fork()==0) break;
        }

while(1){
        printf("<<< Filosofo (%d) a Pensar\n",num);
        sleep(5);
        printf("--- Filosofo (%d) com Fome...\n",num);

        obtem_chopsticks(num);

        printf(">>> Filosofo (%d) a Comer...  ;-) !!!\n",num);
        sleep(4);
```

```
        liberta_chopsticks(num);
        }
}
/*-------------------------------------------------------------------------*/
void obtem_chopsticks(int id)
{
if(id % 2 == 0){ // even number: left, then right
       sem_wait(semid,(id+1)%5);
       sem_wait(semid,id);
       }
else{  // odd number: right, then left
     sem_wait(semid,id);
     sem_wait(semid,(id+1)%5);
     }
 printf("PHILOS(%d) obtem chopstick(%d) e chopstick(%d)\n",
                         id, id, (id+1)%5);
}
/*-------------------------------------------------------------------------*/
void liberta_chopsticks(int id)
{
     sem_signal(semid,id);
     sem_signal(semid,(id+1)%5);
 printf("PHILOS(%d) liberta os chopstick(%d) e chopstick(%d)\n",
                         id, id, (id+1)%5);
}
/*-------------------------------------------------------------------------*/
cleanup()
{
sem_rm(semid); // remove os semaforos
exit();
}
/*-------------------------------------------------------------------------*/
```