Primitivas para Comunicação e Sincronização

#include <sys/types.h>
#include <sys/ipc.h>

	Message Queue	Semaphore	Shared Memory
Include File	<sys/msg.h>	<sys/sem.h>	<sys/shm.h>
System call to create or open	msgget	semget	shmget
System call for control oper.	msgctl	semctl	shmctl
System call for IPC operations	msgsnd msgrcv	semop	shmat shmdt

get operation:

Key - IPC_PRIVATE (cria canal IPC único).
outro valor. (uso global).

Flag - IPC_CREAT (cria se não existe, ou devolve valor existente).
IPC_CREAT | IPC_EXCL (cria se não existe, dá erro se já existe).
No flag (dá erro se não existe; OK se já existe).

9-bits menos significativos da flag - Privilégio.

Os identificadores **msqid**, **semid**, **shmid** têm significado em todo o sistema, enquanto que os file descriptores tinham apenas significado no contexto do processo.

Comandos Unix:

- **ipcs** report ipc status.
- **ipcrm** remove a message queue, semaphore set, or shared memory ID.

Filas de Mensagens

Criar uma fila de mensagens ou obter uma já existente:

```
int msgget(key_t key, int msgflag);
```

Devolve **msqid**, ou -1 em caso de erro.

Colocar uma mensagem numa fila de mensagens:

```
int  msgsnd(int msqid,struct msgbuf *ptr,
            int lenght, int flag);
```

Devolve 0 se tudo correu bem, ou -1 em caso de erro.
ptr é um ponteiro para o buffer com a mensagem. Deve ser uma estrutura do tipo:

```
struct msgbuf{
        long    mtype;          /* message type */
        char    mtext[1];       /* message data */
        };
```

O conteúdo da mensagem não é interpretado pelo sistema, apenas o tipo. Sendo assim, podemos definir estruturas diferentes, desde que o primeiro campo seja sempre um long (mtype).

```
typedef struct my_msgbuf{
        long    mtype;
        short   mshort;
        char    mdata[10];
        };
```

lenght é o tamanho do campo de dados da mensagem.
flag: IPC_NOWAIT (retorna imediatamente se a fila estiver cheia)
0

Retirar uma mensagem de uma fila de mensagens:

```
int  msgrcv(int msqid,struct msgbuf *ptr,
            int lenght, long mtype, int flag);
```

Devolve tamanho da mensagem se tudo correu bem, ou -1 em caso de erro.

mtype = 0 devolve a primeira mensagem da fila (ordem FIFO);

mtype > 0 devolve a primeira mensagem com um tipo igual a mtype;
mtype < 0 devolve a primeira mensagem com um tipo menor ou igual ao valor absoluto de mtype.

flag IPC_NOWAIT (retorna imediatamente se não houver nenhuma mensagem disponível).
0 (bloqueia até receber mensagem)
MSG_NOERROR (se este bit estiver activado não acusa erro quando o tamanho da mensagem é maior que o length indicado)

Operação de controle sobre uma fila de mensagens:

```
int msgctl(int msqid,int cmd,
           struct msqid_ds *buff);
```

cmd IPC_RMID (remove a fila de mensagens).

Exemplo 10

```
/*+-------------------------------------------------------------------------+
  | Exemplo 10  : demonstra uso de filas de mensagens                       |
  | Descricao   : Existem 3 processos organizados sob a forma de um         |
  |               ring e que passam mensagens nesse sentido.                  |
  +-------------------------------------------------------------------------+*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MSGKEY 10
#define SIZE_MSG 20

char conteudo[]="hot potato ...";

struct my_msg{
       long mtype;
       char data[SIZE_MSG];
       }msg[3],buffer;

int msgqid;

int ring(int);
int cleanup();
/*------------------------------ MAIN ----------------------------------*/
main()
{
int   i,status;

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

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

for(i=0;i<3;i++){
```

```
        msg[i].mtype  = i+1;
        strcpy(msg[i].data,conteudo);
        }

printf("------------------------ EXEMPLO 10 -------------------------\n");

for(i=1;i<3;i++){               /* cria 2 processos (i=1,i=2) */
        if(fork() == 0){
               ring(i);
               exit(0);
               }
        }

msgsnd(msgqid,&msg[0],SIZE_MSG,0);
ring(i);               /* processo pai, i=3 */

for(i=0;i<2;i++)
        wait(&status);

cleanup();
}
/*--------------------------------------------------------------------------*/
int ring(int id)
{
int  k;

printf("Eu sou o processo %d (pid=%d)\n",id,getpid());

for(k=0;k<10;k++){
        msgrcv(msgqid,&buffer,SIZE_MSG,id,0);
        printf("Processo[%d]: I got the %s\n",id,buffer.data);
        fflush(stdout);
        sleep(7);
        msgsnd(msgqid,&msg[id%3],SIZE_MSG,0);
        }
}
/*--------------------------------------------------------------------------*/
cleanup()
{
msgctl(msgqid,IPC_RMID,0);       /* remove fila de mensagens */
exit();
}
/*--------------------------------------------------------------------------*/
```

Semáforos

Criar um conjunto de semáforos:

```
int semget(key_t key, int nsems, int semflag);
```

Devolve **semid**, ou -1 em caso de erro.
nsems número de semáforos.

Operações (WAIT/SIGNAL) (DOWN/UP) sobre semáforos:

```
int semop(int semid,struct *opsptr,int nops);
```

opsptr é um ponteiro para um array que define as operações sobre os semáforos.
Cada elemento é uma estrutura do tipo:

```
struct sembuf{
        ushort      sem_num;
        short       sem_op;
        short       sem_flg;
        };
```

sem_op > 0 o valor de sem_op é adicionado ao valor corrente do semáforo. Os processos bloqueados que vejam os seus pedidos satisfeitos serão desbloqueados (SIGNAL).
sem_op = 0 bloqueia à espera do valor 0 no semáforo;
sem_op < 0 adiciona sem_op ao valor do semáforo. Se o resultado for zero liberta os processos à espera da condição zero. Se o valor do semáforo for menor em valor absoluto que o valor de sem_op, o processo fica bloqueado (WAIT).

sem_flg IPC_NOWAIT - não fica à espera se a operação não pode ser efectuada;
SEM_UNDO - liberta o semáforo caso o processo termine.

Operação de controle sobre um conjunto de semáforos:

```
int  semctl(int semid,int semnum, int cmd,
            union semun arg);
```

cmd IPC_RMID (remove este conjunto de semáforos).
GETVAL (obtém o valor de um semáforo).
SETVAL (atribuí um valor a um semáforo)

IPC_GETALL
IPC_SETALL

union semun{
int val; /* value for SETVAL */
struct semid_ds *buf; /* buffer for IPC_STAT, IPC_SET */
unsigned short int *array; /* array for GETALL, SETALL */
struct seminfo *__buf; /* buffer for IPC_INFO */
}**arg**;

Exemplo 11

```
/*+-------------------------------------------------------------------+
  | Exemplo 11  : demonstra uso de semaforos                          |
  | Descricao   : Existem 3 processos que enviam bursts de caracteres |
  |               para o ecran. Existe um semaforo que garante a      |
  |              exclusao mutua no acesso ao stdout.                  |
  +-------------------------------------------------------------------+*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define USE_LOCK 1

#define SEMKEY 10

union semun {
        int val;                    /* value for SETVAL */
        struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
        unsigned short int *array;  /* array for GETALL, SETALL */
         struct seminfo *__buf;      /* buffer for IPC_INFO */
};

int semid;

struct sembuf Wait_stdout,Signal_stdout;

int printa(int);
int cleanup();
/*------------------------------ MAIN ----------------------------------*/
main()
{
int   i,status;
union semun init;

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

semid = semget(SEMKEY,1,0777|IPC_CREAT);        /* cria 1 semaforo */

init.val = 1 ;
semctl(semid,0,SETVAL,init);            /* Inicializa Semaforo */

Wait_stdout.sem_op  = -1 ;
Wait_stdout.sem_flg = SEM_UNDO ;
Wait_stdout.sem_num = 0 ;

Signal_stdout.sem_op  = 1 ;
Signal_stdout.sem_flg = SEM_UNDO ;
Signal_stdout.sem_num = 0 ;
```

```
printf("------------------------ EXEMPLO 10 -------------------------\n");

for(i=1;i<3;i++){               /* cria 2 processos (i=1,i=2) */
        if(fork() == 0){
               printa(i);
               exit(0);
               }
        }

printa(i);             /* processo pai, i=3 */

for(i=0;i<2;i++)
       wait(&status);

cleanup();
}
/*-------------------------------------------------------------------------*/
int printa(int id)
{
int  k,i;

printf("Eu sou o processo %d (pid=%d)\n",id,getpid());

for(k=0;k<10;k++){
       sleep(5);

       #ifdef USE_LOCK
       semop(semid,&Wait_stdout,1);
       #endif

       for(i=0;i<350;i++)
              printf("%d",id);
       printf("\n");
       fflush(stdout);

       #ifdef USE_LOCK
       semop(semid,&Signal_stdout,1);
       #endif
       }
}
/*-------------------------------------------------------------------------*/
cleanup()
{
semctl(semid,1,IPC_RMID,0);       /* remove semaforo */
exit();
}
/*-------------------------------------------------------------------------*/
```

Memória Partilhada

Com estas chamadas que se seguem, os processos podem declarar regiões de memória partilhada como pertencentes ao seu espaço de endereçamento.

Criar uma região de memória partilhada:

```
int shmget(key_t key, int size, int shmflag);
```

Devolve **shmid**, ou -1 em caso de erro.
size tamanho da região.

Mapeamento de uma região de memória partilhada:

```
char *shmat(int shmid,char *shmaddr,
            int shmflag);
```

Devolve o endereço do segmento de memória partilhada.
Se shmaddr=0 --- o sistema escolhe o endereço para o segmento (é aconselhável).

Retirar uma região de memória partilhada do espaço end. processo:

```
char *shmdt(char *shmaddr);
```

Operação de controle sobre uma região memória partilhada:

```
int  shmctl(int shmid,int cmd,
            struct shmid_ds *buf);
```

cmd IPC_RMID (remove esta região de memória partilhada).

NOTA IMPORTANTE !!!

A coerência no acesso à região de memória partilhada deve ser da responsabilidade do programador (através do uso de semáforos).

Exemplo 12

```
/*+-------------------------------------------------------------------+
  | Programa    : ex12.c                                              |
  | Usage       : copy1 file1 file2                                   |
  | Descricao   : Algoritmo Produtor / Consumidor (c/ 1 buffer).      |
  |               Proc. pai lê do file1 para shared buffer.           |
  |               Proc. filho lê do shared buffer para file2          |
  +-------------------------------------------------------------------+*/
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SEMKEY IPC_PRIVATE
#define SHMKEY IPC_PRIVATE

#define SIZE_OF_BUFFER 300

union semun {
       int val;                    /* value for SETVAL*/
       struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
       unsigned short int *array;  /* array for GETALL, SETALL */
       struct seminfo *__buf;      /* buffer for IPC_INFO */
};

struct sembuf Wait_EMPTY,Signal_EMPTY;
struct sembuf Wait_FULL,Signal_FULL;

int semid;
int shmid;

extern char *shmat();
int cleanup();
/********************************** MAIN ****************************************/
main(int argc,char **argv)
{
int     i,status;
char    *adr;
int     *ptr_size;
ushort  init[2];
union semun aux;
int     fd1,fd2;
int     nwr=1;
int     nread=1;
int     done = 0;

if(argc != 3){
       printf("Usage: copy1 file1 file2\n");
       exit(-1);
       }
/* Abre file1 && file2 */
fd1=open(argv[1],O_RDONLY);
if(fd1 == -1){
       printf("Este ficheiro (%s) nao existe !!!\n",argv[1]);
       exit(-1);
       }
fd2=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0600);
if(fd2 == -1){
       printf("Nao consigo criar o ficheiro (%s)!!!\n",argv[2]);
       exit(-1);
```

```
        }

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

/* cria memoria partilhada:          */
/*  | int  | SIZE_OF_BUFFER * char | */
/*    size         buffer            */

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

adr       = shmat(shmid,0,0);         /* attach memory */
ptr_size  = (int *)adr;
adr      += sizeof(int);

semid = semget(SEMKEY,2,0777|IPC_CREAT);        /* cria 2 semaforos */

init[0] = 1 ;                          /* INIT EMPTY */
init[1] = 0 ;                             /* INIT FULL  */
aux.array = &init[0];
semctl(semid,2,SETALL,aux);             /* Inicializa Semaforos */

Wait_EMPTY.sem_op  = -1 ;
Wait_EMPTY.sem_flg = SEM_UNDO ;
Wait_EMPTY.sem_num = 0 ;
Signal_EMPTY.sem_op  = 1 ;
Signal_EMPTY.sem_flg = SEM_UNDO ;
Signal_EMPTY.sem_num = 0 ;
Wait_FULL.sem_op  = -1 ;
Wait_FULL.sem_flg = SEM_UNDO ;
Wait_FULL.sem_num = 1 ;
Signal_FULL.sem_op  = 1 ;
Signal_FULL.sem_flg = SEM_UNDO ;
Signal_FULL.sem_num = 1 ;

printf("--------- PROC_PAI  : READ FROM FILE1 TO SHARED BUFFER -------------------\n");
printf("--------- PROC_FILHO: READ FROM SHARED BUFFER AND WRITE TO FILE2 ---------\n");
if (fork() == 0){
        while(!done){               /* proc. filho */
               semop(semid,&Wait_FULL,1);
               if((*ptr_size) > 0){
                       write(fd2,adr,(*ptr_size));
                       printf("filho: copy %d bytes to file\n",*ptr_size);
                       fflush(stdout);
                       semop(semid,&Signal_EMPTY,1);
                       }
               else{
                       done = 1;
                       }
             }
       exit(1);
        }
while(!done){                       /* proc. pai */
        semop(semid,&Wait_EMPTY,1);
        nread=read(fd1,adr,SIZE_OF_BUFFER);
        *ptr_size = nread;
        semop(semid,&Signal_FULL,1);
        if(nread <= 0)
               done = 1;
        }
wait(&status);
cleanup();
}
/*---------------------------------------------------------------------*/
cleanup()
{
        semctl(semid,2,IPC_RMID,0);     /* remove os semaforos */
        shmctl(shmid,IPC_RMID,0);       /* remove a shared memory */
        exit();
}
/*---------------------------------------------------------------------*/
```

Módulo de Semáforos

```
/*-------------------------------------------------------------------*/
/* my_sem.h              */
/* Declaracao de funcoes */
extern int sem_create(int nsem, int init_val);
extern int sem_setvalue(int sem_id, int sem_num, int value);
extern int sem_wait(int sem_id, int sem_num);
extern int sem_signal(int sem_id, int sem_num);
extern int sem_op(int sem_id, int sem_num, int n);
extern int sem_rm(int sem_id);
/*-------------------------------------------------------------------*/

/*+-----------------------------------------------------------------+
  | Modulo de Semaforos: my_sem.c                                   |
  +-----------------------------------------------------------------+*/
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <assert.h>
#include "my_sem.h"

/*****************/
#if defined(__GNU_LIBRARY__) && !defined(_SEM_SEMUN_UNDEFINED)
/* union semun is defined by including <sys/sem.h> */
#else
/* according to X/OPEN we have to define it ourselves */
union semun {
  int val;                    /* value for SETVAL */
  struct semid_ds *buf;       /* buffer for IPC_STAT, IPC_SET */
  unsigned short int *array;  /* array for GETALL, SETALL */
  struct seminfo *__buf;      /* buffer for IPC_INFO */
};
#endif
/*****************/
/*-------------------------------------------------------------------*/
/* CREATE OR OPEN A SEMAPHORE                                        */
/*-------------------------------------------------------------------*/
int sem_create(int nsem, int init_val)
{
union semun aux;
int semid;
int x,i;

semid=semget(IPC_PRIVATE,nsem,0777|IPC_CREAT);

if(semid == -1){
         printf("NAO CONSEGUIU CRIAR O SEMAFORO\n");
         return(-1);
         }

for (i=0; i<nsem; i++){
    x=sem_setvalue(semid, i, init_val);
    if(x == -1){
         printf("Erro em sem_setvalue\n");
         return(-1);
         }
    }
return(semid);
}
/*-------------------------------------------------------------------*/
/* SET THE VALUE OF A SEMAPHORE                                      */
/*-------------------------------------------------------------------*/
int sem_setvalue(int sem_id, int sem_num, int value)
{
  union semun val;
```

```
  val.val = value;
  int ret;

  ret=semctl(sem_id, sem_num, SETVAL, val);
  if(ret == -1){
         printf("NAO CONSEGUIU INICIALIZAR\n");
         return(-1);
         }
}
/*-------------------------------------------------------------------------*/
/* WAIT ON THE SEMAPHORE                                                   */
/*-------------------------------------------------------------------------*/
int sem_wait(int sem_id,int sem_num)
{
struct sembuf x;
int ret;

x.sem_op  = -1 ;
x.sem_flg = 0 ;
x.sem_num = sem_num;

ret=semop(sem_id,&x,1);
if(ret == -1){
         printf("NAO CONSEGUIU FAZER WAIT NO SEMAFORO\n");
         return(-1);
         }
}

/*-------------------------------------------------------------------------*/
/* SIGNAL THE SEMAPHORE                                                    */
/*-------------------------------------------------------------------------*/
int sem_signal(int sem_id, int sem_num)
{
struct sembuf x;
int ret;

x.sem_op  = +1 ;
x.sem_flg = 0 ;
x.sem_num = sem_num ;

ret=semop(sem_id,&x,1);
if(ret == -1){
         printf("NAO CONSEGUIU FAZER SIGNAL NO SEMAFORO\n");
         return(-1);
         }
}
/*-------------------------------------------------------------------------*/
/* OPERATION ON A SEMAPHORE                                                */
/* if n < 0, wait                                                          */
/* if n > 0, signal                                                        */
/*-------------------------------------------------------------------------*/
int sem_op(int sem_id,int sem_num,int n)
{
struct sembuf x;
int ret;

x.sem_op  = n ;
x.sem_flg = 0 ;
x.sem_num = sem_num ;

ret=semop(sem_id,&x,1);
if(ret == -1){
         printf("NAO CONSEGUIU FAZER SEM_OP NO SEMAFORO\n");
         return(-1);
         }
}
/*-------------------------------------------------------------------------*/
/* REMOVE A SEMAPHORE                                                      */
/*-------------------------------------------------------------------------*/
int sem_rm(int sem_id)
{
```

```
int ret;

semctl(sem_id,0,IPC_RMID,0);
if(ret == -1){
         printf("NAO CONSEGUIU REMOVER O SEMAFORO\n");
         return(-1);
         }
}
/*-------------------------------------------------------------------------*/
```

EXERCÍCIO ex13.c (faz uso do Módulo de Semáforos)

```
/*+-------------------------------------------------------------------+
  | Programa    : ex13.c                                              |
  | Usage       : copy1 file1 file2                                   |
  | Descricao   : Algoritmo Produtor / Consumidor (c/ 1 buffer).      |
  |               Proc. pai lê do file1 para shared buffer.           |
  |               Proc. filho lê do shared buffer para file2          |
  | NOTA: este programa e' igual ao ex12.c. A unica diferenca reside na |
  |       no uso do modulo "my_sem". Gcc -o ex13 ex13.c my_sem.c      |
  +-------------------------------------------------------------------+*/
#include <stdio.h>
#include <fcntl.h>
#include "my_sem.h"
#include <sys/shm.h>

#define SHMKEY IPC_PRIVATE
#define SIZE_OF_BUFFER 300

#define FULL 0
#define EMPTY 1

int semid;
int shmid;

extern char *shmat();
int cleanup();
/********************************** MAIN ***************************************/
main(int argc,char **argv)
{
int     i,status;
char    *adr;
int     *ptr_size;
ushort  init[3];
int     fd1,fd2;
int     nwr=1;
int     nread=1;
int     done = 0;

if(argc != 3){
        printf("Usage: copy1 file1 file2\n");
        exit(-1);
        }

/* Abre file1 && file2 */

fd1=open(argv[1],O_RDONLY);
if(fd1 == -1){
        printf("Este ficheiro (%s) nao existe !!!\n",argv[1]);
        exit(-1);
        }

fd2=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0600);
if(fd2 == -1){
        printf("Nao consigo criar o ficheiro (%s)!!!\n",argv[2]);
        exit(-1);
        }
```

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

/* cria memoria partilhada:          */
/*  | int  | SIZE_OF_BUFFER * char | */
/*    size         buffer            */

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

adr       = shmat(shmid,0,0);         /* attach memory */
ptr_size  = (int *)adr;
adr      += sizeof(int);

/* cria e inicializa semaforos */

semid = sem_create(2,0);
sem_setvalue(semid,FULL,0); // FULL = 0
sem_setvalue(semid,EMPTY,1);  // EMPTY = 1

if(semid == -1){
            printf("Erro a criar semaforos");
            cleanup();
            }
printf("--------- PROC_PAI  : READ FROM FILE1 TO SHARED BUFFER -------------------\n");
printf("--------- PROC_FILHO: READ FROM SHARED BUFFER AND WRITE TO FILE2 ---------\n");
if (fork() == 0){
        /* processo filho */

        while(!done){
               sem_wait(semid,FULL);
               if((*ptr_size) > 0){
                       write(fd2,adr,(*ptr_size));
                       printf("filho: copy %d bytes to file\n",*ptr_size);
                       fflush(stdout);
                       sem_signal(semid,EMPTY);
                       }
               else{
                       done = 1;
                       }
             }
        exit(1);
        }

/* processo pai */

while(!done){
        sem_wait(semid,EMPTY);
        nread=read(fd1,adr,SIZE_OF_BUFFER);
            *ptr_size = nread;
            sem_signal(semid,FULL);
        if(nread <= 0)
                        done = 1;
        }
wait(&status);
cleanup();
}
/*--------------------------------------------------------------------*/
cleanup()
{
        sem_rm(semid);         /* remove os semaforos */
        shmctl(shmid,IPC_RMID,0);       /* remove a shared memory */
        exit();
}
/*--------------------------------------------------------------------*/
```