Instituto Superior de Engenharia do Porto
Departamento de Engenharia Informática

SISTEMAS OPERATIVOS I

Textos de Apoio às Aulas Práticas

Programação Concorrente em Linux: Memória Partilhada e Semáforos

Versão 1.03

Maio de 2006

Luís Lino Ferreira
Berta Batista
Maria João Viamonte
Nuno Malheiro

Sugestões e participações de erros para: llf@dei.isep.ipp.pt

Agradecimentos

Agradecemos aos colegas Jorge Pinto Leite e António Costa a sua participação na revisão deste documento.

ÍNDICE

1 Introdução

Normalmente um Sistema Operativo (SO) protege a área de memória ocupada por cada processo de modo a que outros processos não sejam capazes de acidentalmente aceder e alterar os seus dados de outros processos. No entanto, um dos mecanismos mais simples para a comunicação entre processos é a utilização de memória partilhada. Este tipo de mecanismo permite que uma parte da memória seja utilizada simultaneamente por dois ou mais processos trocando informações ao escreverem ou lerem dessa área. Por outro lado, o acesso a estas zonas de memória apenas deve ser feito por vários processo de forma sincronizada, evitando situações em que simultaneamente um processo está a ler e outro processo está a escrever sobre os mesmos dados. Para tal podem ser utilizados mecanismos de sincronização baseados em semáforos.

Estes apontamentos descrevem a utilização de duas bibliotecas, uma para utilização de memória partilhada (shmlib.c) e outra para a utilização de semáforos (semlib.c), que permitem de forma simples mas limitada a utilização dos mecanismos de memória partilhada e de semáforos existentes no SO Linux.

2 Livraria de Memória Partilhada

A livraria de memória partilhada (shmlib.c) é constituída apenas por duas funções: `ShmDef()` e `ShmDel()`. A primeira permite obter um zona de memória partilhada enquanto que segunda permite libertar uma zona de memória partilhada, os seu protótipos aparecem a seguir:

```
void *ShmDef(int size, int *descr);
int  ShmDel(int descr);
```

A função `ShmDef()` permite obter uma zona memória partilhada privada (ao processo evocador e aos respectivos filhos). Têm como parâmetros **`size`** e **`descr`**. O parâmetro **`size`** define o comprimento da zona de memória partilhada em Bytes, cujo valor deverá ser superior a zero. Note-

se que para definir uma zona de memória partilhada para armazenar dados de tipos diferentes: `struct`, `double` ou `int`, é necessário utilizar a macro do pré-processador `sizeof()`, tal como apresentado na Figura 1. O parâmetro `descr` é um descritor que identifica, perante o kernel do SO, a zona de memória partilhada que foi criada, note-se que este parâmetro deve ser passado por referência. Em caso de sucesso, a função devolve um ponteiro para a zona de memória partilhada que foi criada, no caso contrário é devolvido um ponteiro para `NULL`. No entanto, deve ser feito um *cast* do ponteiro para o tipo a utilizar. A Figura 1 apresenta um exemplo da utilização da função `ShmDef()` em que é reservada uma zona de memória partilhada com tamanho suficiente para armazenar um vector com 20 inteiros. O acesso ao vector pode ser feito através da aritmética de ponteiros[1] ou, tal como demonstrado no código da figura, através dos mecanismos de manipulção de vectores disponibilizados pela linguagem C.

```
…
 int *shm;   // Apontador para a zona de memória partilhada
 int shmd;  // Variável que vai armazenar o descritor da zona de memória
partilhada

…
 shm = (int *) ShmDef(sizeof(int)*20, &shmd);
…
 shm[0]  = 20;
 shm[19] = 222;
…
```

Figura 1 – Utilização da função ShmDef

A função `ShmDel()`, permite marcar a zona de memória partilhada como podendo ser apagada, o que apenas acontece quando todos os processos que a estão a utilizar terminarem. A função têm como parâmetro o descritor da zona de memória partilhada obtido através da função `ShmDef()`. Devolve 0 em caso de sucesso e -1 em caso de erro.

A Figura 2, mostra um exemplo da utilização de uma zona de memória partilhada com comprimento suficiente para armazenar 20 inteiros[2]. Neste exemplo, o processo pai escreve os números de 1 até 20 nessa zona de memória que depois são lidos pelo processo filho e impressos no ecrã.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
 int     *apt;
 int     f;
 int     pid;
 int     shmd; //descritor da zona de memória partilhada

 apt = (int *) ShmDef(sizeof(int)*20, &shmd);
```

[1] A aritmética de ponteiros não é muito aconselhável devido aos erros que potencialmente poderão ser cometidos.

[2] Na realidade o espaço reservado depende do tamanho da página que está a ser utilizado pelo processador. P.e. ao reservar-se um espaço de 20 Bytes, o SO reserva normalmente em Linux um espaço de memória com 4096Bytes.

```
 if (apt == NULL)
 {
     perror("Erro na obtenção de memória partilhada\n");
     exit(-1);
 }


 pid = fork();
 if (pid > 0) // Processo Pai
 {
     for(f = 0; f < 20; f ++)
     {
          apt[f] = f;
     }
     printf("Processo pai concluído com sucesso!!!");
     ShmDel(shmd); // Liberta a zona de memória partilhada
 }
 else // Processo Filho
 {
     sleep(4);
     for(f = 0; f < 20; f++)
     {
          printf("1:f=%d val=%d\n", f, apt[f]);
     }
     exit(0);
 } // Fim do processo filho
} //Fim do main
```

Figura 2 – Exemplo de memória partilhada

Note-se que no código da Figura 2 foram omitidas a maior parte das verificações de erros de modo a simplificar o exemplo.

3 Livraria de Semáforos

O ficheiro semlib.c contém um conjunto de funções que, de forma simples, permite operar com os mecanismos de semáforos existentes em Linux. A livraria contém funções para definir e atribuir um valor inicial a um semáforo, para implementar as funcionalidades de *Wait* e de S*ignal* (tal como definidas nos apontamentos da aulas teóricas de SOP1) e para apagar os semáforos. De seguida são apresentados os protótipos das funções da livraria.

```
int SemDef(   int inicial_value);
int SemSignal(int descr_sem);
int SemWait(  int descr_sem);
int SemDel(   int descr_sem);
```

A função `SemDef()` permite definir um semáforo no kernel do SO, privado ao processo evocador e respectivos filhos. Ao memo tempo, através do parâmetro **`incial_value`** define-se o valor inicial do semáforo. Esta função retorna um inteiro (o descritor do semáforo) que permite identificar o semáforo criado nas operações que decorram a seguir. Em caso de erro é devolvido o valor -1.

A função `SemSignal()` implementa a primitiva de *Signal* que permite incrementar o valor do semáforo. O parâmetro **`descr_sem`** é o descritor do semáforo, devolvido pela função `SemDef()`. Em caso de sucesso a função devolve o valor 0 e em caso de erro o valor -1.

A função `SemWait()` implementa a primitiva de *Wait* que permite decrementar o valor do semáforo, caso o valor do semáforo seja igual ou inferior a zero o processo evocador é bloqueado

até que outro processo evoque a primitiva de `SemSignal()`, nessa altura o primeiro processo que ficou bloqueado passa para o estado de ***Ready to Run***. Em caso de sucesso a função devolve o valor 0 e em caso de erro o valor -1.

Para libertar os recursos ocupados pelo semáforo no sistema operativo deve-se utilizar a função `SemDel()`, passando como parâmetro o descritor do semáforo. Tal como nos casos anteriores, em caso de sucesso a função devolve o valor 0 e em caso de erro o valor -1.

O programa apresentado na Figura 3, demonstra para o caso da Figura 2, como é que se podem utilizar semáforos de modo a garantir a exclusão mútua no acesso à zona de memória partilhada.

```
#include <stdio.h>
#include <stdlib.h>


main()
{
 int     *apt;
 int     f;
 int     pid;
 int     s;       // Descritor do semáforo
 int     shmd;    // Descritor da memória partilhada
 int     erro;

 apt = (int *) ShmDef(sizeof(int)*20, &shmd);
 if (apt == NULL)
 {
     perror("Erro na obtenção de memoria partilhada\n");
     exit(-1);
 }

 // Define um semáforo com valor incial igual a 1
 s = SemDef(1);
 if (s == -1)
 {
     perror("Erro na criação do semáforo");
     exit(-1);
 }

 pid = fork();
 if (pid > 0)
 {
     //Código do processo pai
     SemWait(s);
     printf("Pai: a escrever na memória partilhada!!\n");
     //Inicio da secção crítica
     for(f=0; f < 20; f++)
     {
          apt[f] = f;
     }
     sleep(3);
     printf("Pai: a sair da secção crítica\n");
     //Fim da secção crítica
     SemSignal(s);

     //Coloca o pai 'a espera do filho
     wait();
```

```
    //Remove a zona de memória partilhada
    erro = ShmDel(shmd);

    if (erro == -1)
    {
        printf("Pai: Erro em ShmDel!!!");
        exit(-1);
    }
    //Remove o semáforo
    erro = SemDel(s);
    if (erro == -1)
    {
        printf("Pai: Erro em SemDel!!!");
        exit(-1);
    }
    printf("Pai:Processo pai a terminar!!!\n");
 }
 else
 {
    // Código do filho
    sleep(3);
    printf("Filho: 'a espera para entrar na secção crítica!!\n");

    //Inicio da secção crítica
    SemWait(s);
    printf("Filho: a executar a secção crítica\n");
    for(f=0; f < 20; f++)
    {
        printf("Filho: f=%d val=%d\n", f, apt[f]);
        sleep(1);
    }
    printf("Filho: a sair da secção crítica\n");
    SemSignal(s);
    //Fim da secção crítica

    exit(0);
 }
} // Fim do main
```

Figura 3 – Exemplo de utilização da livraria de semáforos

A instrução `sleep(3)` colocada no início do código do processo filho coloca o filho no estado de **Waiting** durante 3 s, permitindo desta forma que o código do processo pai corra.

Note-se que no código da Figura 3 foram omitidas a maior parte das verificações de erros de modo a simplificar o exemplo.

4 Comandos da Shell

A Shell do Linux oferece um conjunto de comandos para a gestão de zonas de memória partilhada e de semáforos, nomeadamente os comandos `ipcs` e `ipcrm`. Estes comandos são particularmente úteis quando os programas realizados durante as aulas práticas funcionam mal dado que permitem saber que recursos estão ser ocupados e liberta-los. Note-se que os recursos do sistema são limitados.

O comando `ipcs` imprime uma lista com informação sobre as zonas de memória partilhada, semáforos e filas de mensagens. A Figura 4 apresenta um exemplo da saída do comando `ipcs`. A tabela apresenta para os zonas de memória partilhada os campos: `key`, `shmid`, `owner`, `perms`,

`bytes`, `nattch` e `status`. Para os semáforos são apresentados os campos `key`, `semid`, `owner`, `perms`, `nsems` e `status`. Deste conjunto de campos é de destacar os campos `shmid` e semid, que representam o identificador da zona de memória partilhada e do semáforo. Este identificador deve ser utilizado em operações de manipulação do segmento de memória partilhada ou do semáforo, tal como apagar o semáforo.

```
~/so1/proc> ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00001389 4098       i020847    666        10         0
0x00000000 23555      i010526    666        4          0
0x00000000 28676      llf        600        20         0

------ Semaphore Arrays --------
key        semid      owner      perms      nsems      status
0x00000000 4608       i010526    666        1
0x00000000 5121       llf        777        1

------ Message Queues --------
key        msqid      owner      perms      used-bytes  messagesipcs
```

Figura 4 – Exemplo da execução do comando ipcs

De modo a não consumir inutilmente os recursos do SO, deve-se em primeiro lugar apagar os semáforos e as zonas de memória partilhada no próprio programa que as utiliza, caso esta opção falhe, pode-se apagar o recurso recorrendo ao comando *ipcrm*. Este comando aceita como opções *sem* ou *shm*, respectivamente para apagar semáforos ou memória partilhada. A seguir à opção deve aparecer o identificador do recurso (o valor contido no campo *shmid* e *semid*, que foi devolvido pelo comando *ipcs*). A figura seguinte apresenta o resultado da utilização do comando ipcrm de modo a apagar a zona de memória partilhada com o identificador 28676.

```
~> ipcrm shm 28676
resource(s) deleted
```

Figura 5 – Utilização do comando ipcrm

5 Bibliografia

Orlando Sousa, “Processos”, Apontamentos das aulas práticas de SOP2, Instituto Superior de Engenharia do Porto, 2004.

W. Richard Stevens, “Advanced Programming in the UNIX Environment”, Addison Wesley, 1994

John Shapley Gray, “Interprocess Communications in UNIX – The Nooks & Crannies, 2nd Edition”, Prentice Hall, 1998