Politécnico Castelo Branco
Escola Superior de Tecnologia

Escola Superior de Tecnologia
Curso: Licenciatura em Engenharia Informática
2º Semestre do 2º Ano
Unidade Curricular: Sistemas Operativos

Ano Letivo 2024/2025

Ficha de Trabalho nº 4 – Sincronização de Processos com Semáforos (Linux)

Índice

1 Objectivos

- Construção de programas que envolvam a sincronização entre processos com recurso a semáforos, em ambiente Linux.
- Familiarização com a criação e utilização de conjuntos de semáforos em Linux.

2 Semáforos

Os semáforos são objectos *Inter-Process Communication* (IPC) utilizados para a sincronização entre processos. Os semáforos permitem resolver problemas de exclusão mútua. Nomeadamente, os semáforos resolvem conflitos de acesso concorrentes de processos distintos a um mesmo recurso.

2.1 Determinar o estado dos mecanismos IPC

Os sistemas Linux fornecem ao utilizador um conjunto de comandos que permite o acesso a informação relacionada com os três mecanismos IPC (semáforos, memória partilhada e filas de mensagens). Os comandos **ipcs** e **ipcrm** são bastante úteis aos programadores durante o desenvolvimento de aplicações.

O comando **ipcs -<tipo de recurso>** fornece informação actualizada sobre cada IPC implementado no sistema. O tipo de recurso pode ser especificado da seguinte forma:

- **ipcs -s** -> informações sobre os semáforos;
- **ipcs -m** -> informações relativas aos segmentos de memória partilhada;
- **ipcs -q** -> informações sobre as filas de mensagens;
- **ipcs -a** -> todos os recursos (opção por omissão, se nenhum parâmetro for especificado).

O comando **ipcrm** permite que recursos IPC que tenham acidentalmente ficado no sistema após a execução duma aplicação possam ser destruídos através da linha de comandos. Este comando obriga a que se especifique um parâmetro indicativo do tipo de recurso a ser destruído, assim como o identificador associado a esse recurso. A sintaxe de utilização do comando **ipcrm** é a seguinte:

- **ipcrm [-s|-m|-q] <id>**

2.2 Conceitos essenciais sobre semáforos

Em sistemas Linux, a criação de um conjunto de semáforos requer que: seja associado o **valor de uma chave** ao conjunto de semáforos a criar, seja indicada a **quantidade de semáforos** a criar e que sejam definidas as **permissões** associadas ao conjunto de semáforos a criar. O sistema, após a criação de um conjunto de semáforos, retorna um **identificador**

único para esse conjunto de semáforos. Cada semáforo dentro do conjunto de semáforos é identificado pelo índice correspondente à sua posição dentro do conjunto (a numeração dos índices começa em 0).

Cada semáforo tem associado um valor inteiro e este pode ser incrementado ou decrementado pelos vários processos que tenham acesso a esse semáforo. A utilização de semáforos por parte dos processos, em sistemas Linux, rege-se pelas seguintes regras:

- Sendo N o valor actual do semáforo e n o valor usado pelo processo na operação sobre o semáforo:

- Se $n > 0$ (operação de incremento)
    - O valor actual do semáforo é **incrementado** n unidades e o processo continua a sua execução;
- Se $n < 0$ (operação de decremento)
    - se $N + n \geq 0$, o valor actual do semáforo é **decrementado** $|n|$ unidades e o processo continua a sua execução;
    - se $N + n < 0$, o **processo bloqueia** a sua execução nesta operação e **espera até que** $N + n \geq 0$.

2.3 Criar um conjunto de semáforos

A função ***semget()*** é responsável por criar um conjunto de semáforos. A função tem os seguintes argumentos e retorno:

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

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

Os argumentos envolvidos na criação de um conjunto de semáforos (***semget()***) são os seguintes:

1. ***key*** – valor que identifica um conjunto de semáforos;
2. ***nsems*** – quantidade de semáforos a criar;
3. ***semflg*** – acessos desejados para o semáforo e a constante IPC_CREATE (criar a chave, se não existir). A utilização da constante IPC_EXCL irá provocar a falha no caso da existência de um conjunto associado à chave. O valor 0600 garante apenas ao utilizador as permissões de escrita e leitura de semáforos.

```
/* Mode bits (semflg) for `msgget', `semget', and `shmget'.  */
#define IPC_CREAT       01000           /* Create key if key does not exist. */
#define IPC_EXCL        02000           /* Fail if key exists.  */
#define IPC_NOWAIT      04000           /* Return error on wait.  */
```

A função ***semget()*** retorna o identificador do conjunto do semáforos (valor não negativo) em caso de sucesso. Em situação de erro retorna o valor -1.

Exemplo: Criar um conjunto de semáforos (**criar_sem.c**):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>

#define KEY 123
#define NUM_SEMS 4

int main(int argc, char *argv[])
{
        /* identificador do conjundo de semaforos */
        int semid;

        /* criar o grupo de semaforos (4 semaforos) */
        if((semid = semget(KEY, NUM_SEMS, IPC_CREAT|IPC_EXCL|0600)) == -1) {
                perror("Erro ao criar o semaforo");
                return 1;
        }

        printf("ID grupo semaforos: %d\n", semid) ;
        printf("Identificado pela chave unica : %d\n", KEY);

        return 0;
}
```

Teste o programa e utilize as ferramentas **ipcs** (para consultar o grupo de semáforo criado) e **ipcrm** (para remover o semáforo criado).

2.4 Operações sobre um semáforo

A função ***semop()*** permite realizar operações sobre um semáforo. As operações que podem ser realizadas sobre um semáforo são **incrementos** e **decrementos**. A função tem os seguintes argumentos e retorno:

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

int semop(int semid, struct sembuf *sops, unsigned nsops);
```

Os argumentos da função ***semop()*** são:

1. ***semid*** – identificador do conjunto de semáforos sobre o qual serão efectuadas operações;
2. ***sops*** – conjunto de operações a realizar sobre o conjunto de semáforos, podem realizar-se várias operações, neste caso definidas usando um *array*. As operações são especificadas utilizando a estrutura de dados ***sembuf*** (constituição desta estrutura apresentada em baixo). Aqui especifica-se o número do semáforo (***sem_num***), a operação a realizar (***sem_op***) e as opções (*flags*) das operações (***sem_flg***).
    a. A operação a realizar é baseada no valor inteiro indicado em ***sem_op***. Se o valor indicado for positivo, então este valor é adicionado ao semáforo e o processo continua a execução. Se o valor indicado for negativo, então neste caso **duas situações podem acontecer**:

- **Primeira**, se o valor actual do semáforo for maior ou igual ao valor indicado em ***sem_op***, então o processo pode prosseguir e o valor indicado em ***sem_op*** é subtraído ao valor actual do semáforo;
- **Segunda**, se o valor actual do semáforo for inferior ao valor indicado em ***sem_op***, então o processo fica bloqueado até que a situação descrita no ponto anterior se verifique.

b. As *flags* podem ser IPC_NOWAIT (para indicar que o semáforo não é bloqueante) e/ou **SEM_UNDO** (desfazer automaticamente quando o processo terminar).

3. ***nsops*** – o número de operações especificadas em ***sops***.

Para usar a estrutura ***sembuf***, necessária como segundo argumento da chamada à função ***semop()***, basta fazer a inclusão das bibliotecas indicadas em cima. A constituição desta estrutura é a seguinte:

```
struct sembuf {
  short int sem_num;            /* número do semáforo */
  short int sem_op;             /* operação a realizar no semáforo */
  short int sem_flg;            /* flag da operação */
};
```

A função ***semop()*** retorna 0 em caso de sucesso e -1 em caso de erro.

2.5 Remoção de um semáforo e outros comandos auxiliares

A função ***semctl()*** é responsável por realizar comandos auxiliares sobre semáforos. A função tem os seguintes argumentos e retorno:

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

int semctl(int semid, int semnum, int cmd);
```

Os argumentos da função ***semctl()*** são:

1. ***semid*** – identificador do conjunto de semáforos;
2. ***semnum*** – número do semáforo em que se pretende comandar;
3. ***cmd*** – comando a realizar sobre o semáforo; Os comandos que podemos realizar sobre um semáforo estão indicados na tabela seguinte:

```
/* Comandos (cmd) para semctl */
#define IPC_RMID        0                   /* Remove identifier.  */
#define IPC_SET         1                   /* Set `ipc_perm' options.  */
#define IPC_STAT        2                   /* Get `ipc_perm' options.  */
#define IPC_INFO        3                   /* See ipcs.  */

/* Comandos (cmd) do semctl:  */
#define GETPID              11              /* get sempid */
#define GETVAL              12              /* get semval */
#define GETALL              13              /* get all semval's */
#define GETNCNT             14              /* get semncnt */
#define GETZCNT             15              /* get semzcnt */
#define SETVAL              16              /* set semval */
#define SETALL              17              /* set all semval's */
```

A função *semctl()* pode retornar vários valores, dependendo do comando realizado. Em caso de erro é retornado o valor -1.

2.6 Exemplos

Exemplo de decremento de um semáforo (dec_op_sem.c):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>

#define KEY 123
#define NUM_SEMS 4

int main(int argc, char *argv[])
{
        struct sembuf sempar;
        int semid, semval;

        /* Obter o identificador de um conjunto de semaforos anteriormente criado */
        if((semid = semget(KEY, 0, 0)) == -1 ){
                perror ("Error semget()");
                return 1;
        }

        printf("ID grupo semaforos: %d\n", semid);
        printf("Identificado pela chave unica : %d\n", KEY);

        /* Operacao de decremento do segundo semaforo */
        sempar.sem_num = 1;           // vamos operar o segundo semaforo
        sempar.sem_op = -1;           // decrementa este valor ao valor do semaforo
        sempar.sem_flg = SEM_UNDO;    // desfaz apos o processo terminar
        if(semop(semid, &sempar, 1) == -1){
                perror("Error semop()");
                return -1;
        }

        printf("O valor do segundo semaforo foi decrementado!\n");

        return 0;
}
```

Exemplo de incremento de um semáforo (inc_op_sem.c):

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include <errno.h>

#define KEY 123
#define NUM_SEMS 4

int main(int argc, char *argv[])
{
        struct sembuf sempar;
        int semid, semval;

        /* Obter o identificador de um conjunto de semaforos anteriormente criado */
        if((semid = semget(KEY, 0, 0)) == -1 ){
                perror("Error semget()");
                return 1;
        }

        printf("ID grupo semaforos: %d\n", semid);
        printf("Identificado pela chave unica : %d\n", KEY);
```

```
        /* Operacao de incremento do segundo semaforo */
        sempar.sem_num = 1;           // vamos operar o segundo semaforo
        sempar.sem_op = 1;            // incrementa este valor ao valor do semaforo
        sempar.sem_flg = SEM_UNDO;    // desfaz apos o processo terminar
        if(semop(semid, &sempar, 1) == -1){
                perror("Error semop()");
                return -1;
        }

        printf("O valor do segundo semaforo foi incrementado!\n");

        sleep(1);

        return 0;
}
```

Teste os dois programas em execução simultânea, mas usando a seguinte ordem: primeiro coloque em execução **dec_op_sem** e em seguida **inc_op_sem**. Analise os resultados obtidos.

3 Exercícios

Construa dois programas, para execução simultânea, com o seguinte comportamento:

- O **programa1** deve esperar a passagem de 20 segundos. Em seguida deve permitir que o programa2 avance na sua execução e esperar a indicação deste para que possa voltar a avançar. Depois de receber essa indicação espera 5 segundos antes de terminar.
- O **programa2** deverá esperar a indicação do programa1 para avançar e passado 30 segundos deve dar a este a indicação de avançar. Após essa indicação deverá deixar passar 5 segundos e terminar.
- A evolução da execução dos dois programas deve ser percebida através da apresentação de mensagens indicativas das etapas de execução dos programas.

Nota: os recursos IPC criados devem no final ser destruídos pelos programas.