Pontifícia Universidade Católica do Rio de Janeiro

Lab 6 - Semáforos

Sistemas Operacionais

Alunos Tharik Lourenço
Suemy Inagaki

Professor Luiz Fernando Seibel

Rio de Janeiro, 20 de maio de 2021

Conteúdo

1 Problema 1

Faça um programa que gere 3 processos para alterar um valor de uma variável na memória compartilhada inicializada com zero.

- Cada programa executa um loop de 500 execuções.
- O processo 1 soma 1 à variável, o processo 2 soma 2 à variável e o processo 3 soma 3.
- Execute o programa e verifique se houve problema de concorrência.

1.1 Código Fonte (sem semáforo)

```
/*
Tharik Lourenço
Suemy Inagaki
SO - Lab6 - Semáforos
*/


#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>


int main( void ){

    int segmento;
    long long *soma;
    segmento=shmget(IPC_PRIVATE, sizeof (int), IPC_CREAT |
    IPC_EXCL | S_IRUSR | S_IWUSR);
    if(segmento<0){
        printf("Shmget erro!\n");
        exit(1);
    }
    soma = (long long*)shmat(segmento,0,0);
    *soma = 0;
```

```
    for (int i = 1; i <= NUMPROC; ++i)
    {
        if (fork() == 0){
            printf("começo do processo filho %d\n",getpid());
            for(int j = 0; j < 5000000; j++){
                *soma+=i;
                //printf("%d\n", *soma);
            }
            exit(EXIT_SUCCESS);
        }
    }
    for (int i = 0; i < 3; ++i)
        wait(NULL);
    printf("processo pai vai terminar !!!\n");
    printf("soma = %lld\n", *soma);
    exit(0);
}
```

1.2 Código Fonte (com semáforo)

```
/*
Tharik Lourenço
Suemy Inagaki
SO - Lab6 - Semáforos
*/

#include <sys/sem.h>
#include <sys/shm.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <sys/stat.h>
#include <sys/wait.h>
union semun {
        int val;
        struct semid_ds *buf;
        ushort *array;
};
// inicializa o valor do semáforo
int setSemValue(int semId);
```

```
// remove o semáforo
void delSemValue(int semId);
// operação P
int semaforoP(int semId);
//operação V
int semaforoV(int semId);
int main (int argc, char * argv[]){
        int i;
        int segmento;
        segmento = shmget (IPC_PRIVATE, sizeof (int), IPC_CREAT |
        IPC_EXCL | S_IRUSR | S_IWUSR);
        if(segmento<0){
        printf("Shmget erro!\n");
        exit(1);
    }
        long long* soma = (long long*)shmat(segmento,0,0);
        *soma = 0;
        int semId;
        semId = semget (8752, 1, 0666 | IPC_CREAT);
        setSemValue(semId);
        for(int j = 1; j <= 3; j++){
                if(fork() == 0){
                        semaforoP(semId);
                        printf("começo do processo filho %d\n",getpid());
                        for(long k = 0; k < 5000000; k++){
                                *soma+=j;
                        }
                        semaforoV(semId);
                        exit(EXIT_SUCCESS);
                }
        }
        for (int j = 1; j <= 3; ++j)
                   wait(NULL);

        sleep(10);
        delSemValue(semId);
        printf("Soma final = %lld\n", *soma);

        return 0;
}
int setSemValue(int semId){
```

```
        union semun semUnion;
        semUnion.val = 1;
        return semctl(semId, 0, SETVAL, semUnion);
}
void delSemValue(int semId){
        union semun semUnion;
        semctl(semId, 0, IPC_RMID, semUnion);
}
int semaforoP(int semId){
        struct sembuf semB;
        semB.sem_num = 0;
        semB.sem_op = -1;
        semB.sem_flg = SEM_UNDO;
        semop(semId, &semB, 1);
        return 0;
}
int semaforoV(int semId){
        struct sembuf semB;
        semB.sem_num = 0;
        semB.sem_op = 1;
        semB.sem_flg = SEM_UNDO;
        semop(semId, &semB, 1);
        return 0;
}
```

1.3 Resultado da Execução

O programa que não utilizou semáforos apresentou concorrência e o resultado da soma que obtivemos não foi a esperada. (Esperávamos 30000000 e obtivemos 15126297).

Já o programa com semáforos, não houve concorrência e o resultado final foi exatamente o esperado.

Isso se deve ao fato de que os semáforos impedem que outros processos acessem a área crítica enquanto um processo estiver nela. Assim, a concorrência não acontece.

```
$ gcc processos.c -o processos
$ ./processos
come o do processo filho 733
come o do processo filho 734
come o do processo filho 735
processo pai vai terminar !!!
soma = 15126297
```

```
$ gcc semaforos.c -o semaforos
$ ./semaforos
come o do processo filho 746
come o do processo filho 748
come o do processo filho 747
Soma final = 30000000
```