Sistemas Operativos
2021 / 2022

Licenciatura em Engenharia Informática

Lab. 05 – Sincronização de processos Unix - Semáforos

Nesta aula pretende-se que os alunos fiquem com uma noção prática da sincronização de processos em Linux recorrendo à utilização de semáforos.

Ex. 1 – Utilização de semáforos

O seguinte programa implementa um semáforo partilhado por dois processos de modo a gerir o acesso exclusivo à função *do_job(char *)*.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <semaphore.h>
5  #include <fcntl.h>
6
7
8  void do_job(char *owner)
9  {
10     printf("%s locks mutex..\n", owner);
11     sleep(2);
12     printf("%s releases mutex..\n", owner);
13 }
14
15 int main()
16 {
17     sem_unlink("mymutex");
18     sem_t *mutex = sem_open("mymutex", O_CREAT, 0644, 1);
19
20     if (fork() == 0) {
21         printf("Child process!\n");
22         sem_wait(mutex);
23             do_job("Child");
24         sem_post(mutex);
25     }
26     else {
27         printf("Parent process!\n");
28         sem_wait(mutex);
29             do_job("Parent");
30         sem_post(mutex);
31     }
```

```
32
33      sem_close(mutex);
34      return (EXIT_SUCCESS);
35  }
```

Coloque o código num ficheiro de nome *ex1.c* e compile com o *gcc* usando a biblioteca *pthreads* (Posix threads):

```
$ gcc -o ex1 ex1.c -pthread
```

Execute o programa e verifique que o output é semelhante ao seguinte:

```
Parent process!
Parent locks mutex..
Child process!
Parent releases mutex..
Child locks mutex..
Child releases mutex..
```

De uma forma geral, a função *sem_open()* (linha 16) permite-nos criar um semáforo com o nome "mymutex", semáforo esse que é utilizado para controlar o acesso à função *do_job()* tanto no processo pai como no processo filho (*sem_wait()* nas linhas 22 e 29 para esperar enquanto o semáforo não for liberto, e *sem_post()* nas linhas 24 e 31 para libertar o semáforo após a execução da função).

a) Veja as entradas de manual das funções *sem_open*, *sem_close*, *sem_wait* e *sem_post* e tome especial atenção aos argumentos de cada, especialmente da função *sem_open()*.

b) Explique porque razão o processo filho só obtém o *mutex* depois do processo pai soltar o *mutex*.

c) Utilizando a função *sleep(int secs)*, modifique o código de modo a que o processo filho seja sempre o primeiro a obter o *lock* do semáforo.

d) Modifique o programa original (sem o *sleep* anterior) de modo a que o processo pai nunca termine sem receber o evento *exit()* do processo filho. Deverá usar as funções *wait* e *exit* usadas no laboratório anterior.

Ex. 2 – Ping-Pong

Considere o seguinte programa:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <semaphore.h>
5  #include <fcntl.h>
6
7
8  int main()
9  {
```

```
10      if (fork() == 0) {
11          while (1) {
12              printf("Pong\n");
13          }
14      } else {
15          while (1) {
16              printf("Ping\n");
17          }
18      }
19
20      return (EXIT_SUCCESS);
21  }
```

Coloque o código num ficheiro de nome *ex2.c*, compile e execute-o. Verifique no output que em várias situações verá a palavra “Ping” ou a palavra “Pong” de seguida.

Modifique o necessário de modo a que o output seja alternado, isto é, o processo pai deverá escrever “Ping” na consola e o processo filho deverá responder com “Pong”. O “Ping” deverá iniciar primeiro.

Sugestão: use dois semáforos (com nomes do tipo *sem_ping* e *sem_pong*) para controlar a ordem de execução dos processos. Por exemplo, o processo pai deverá aguardar pelo semáforo *sem_ping* e quando escrever “Ping” deverá libertar o semáforo *sem_pong*. O processo filho deverá fazer o procedimento inverso. Conseguirá controlar a ordem de execução inicializando um dos semáforos a 1 (um) e o outro a 0 (zero),

Coloque um sleep(1) a seguir a cada um dos *printf* de modo a conseguir visualizar melhor o output.

Ex. 3 – Ping-Pong a 3

Considere o seguinte programa:

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <unistd.h>
4   #include <semaphore.h>
5   #include <fcntl.h>
6
7
8   int main()
9   {
10      // Pong 1
11      if (fork() == 0) {
12          while (1) {
13              printf("Pong 1\n");
14          }
15      }
16
17      // Pong 2
18      if (fork() == 0) {
19          while (1) {
20              printf("Pong 2\n");
```

```
21          }
22      }
23
24      // Ping
25      while (1) {
26          printf("Ping\n");
27      }
28
29      return (EXIT_SUCCESS);
    }
```

Usando os semáforos *sem_ping*, *sem_pong1* e *sem_pong2* altere o necessário de forma a que a ordem de escrita seja sempre “Ping”, “Pong 1”, “Pong 2”, “Ping”. Comece sempre pelo “Ping”.

(fim de enunciado)