Exercícios
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.

Exercício 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.

Exercício 2: produtor-consumidor

Considere o pseudo-código seguinte que implementa um algoritmo que resolve o problema do produtor/consumidor utilizando semáforos. O que se pretende aqui é ter um *buffer* que guarda valores inteiros, no qual o produtor vai colocando valores. Cada consumidor vai por sua vez retirando valores do *buffer*, um de cada vez, à medida que estes vão ficando disponíveis.

Deve ter em mente que não se poderá efectuar consumo quando o *buffer* está vazio e para além disso deve garantir-se que há exclusão mútua no acesso ao recurso crítico (neste caso o *buffer*).

a) Explique o porquê da utilização neste exemplo de semáforos e mutexes.

b) Suponha que em vez de múltiplos consumidores tinha apenas um. Reescreva o pseudo-código seguinte de forma a contemplar todas as alterações necessárias.

```
program prod_cons_sem;
   var  in, out: integer;
   buf: array[0..n-1] of integer;
   empty, full, mutex: semaphore;

procedure producer;
   var item: integer;
   generate item;
   while true do
   begin
      ACQUIRE(empty);
      ACQUIRE(mutex);
         buf[in] := item;
         in := (in + 1) mod n;
      RELEASE(mutex);
      RELEASE(full);
   end;

procedure consumer;
   var item: integer;
   while true do
   begin
      ACQUIRE(full);
      ACQUIRE(mutex);
         item := buff[out];
         out := (out + 1) mod n;
      RELEASE(mutex);
      RELEASE(empty);
      use item;
   end;


// main
begin
   in := 0;
   out := 0;
   full := semaphore(0);
   empty := semaphore(n);
   mutex := semaphore(1);

   cobegin
      producer;
      consumer;
   coend;
end.
```

Exercício 3: produtor-consumidor

Tendo em conta o pseudo-código anterior, implemente o algoritmo do produtor/consumidor em C utilizando semáforos.

Notas

- Para compilar o código com suporte à biblioteca *pthreads* no Netbeans, deverá incluir a opção "-pthread" em *Run → Set Project Configuration → Customize → C Compiler → Additional Options.*
- Como os semáforos são criados em áreas de memória fora dos processos principais, no caso de um programa terminar incorrectamente, poderá ser necessário "reiniciar" o semáforo usando o comando *sem_unlink("nome_sem")*.