TRAVAUX PRATIQUES
Synchronisation entre processus sous Linux
CORRECTION

L'objectif de ce TP est de revenir sur la notion de synchronisation entre processus sous Linux pour l'accès à des ressources partagées.

Dans ce TP vous aurez à écrire des programmes en langage C pour utiliser les outils fournit par les systèmes d'exploitation de type Linux.

Rappel : Pour compiler vos programmes en langage C vous pouvez utiliser le compilateur gcc comme suit : `$> gcc -o prog prog.c sem_func.c -lpthread`

Exercice 1

Dans cet exercice, nous allons nous intéresser au problème que peut poser l'accès concurrent à une ressource partagée par plusieurs processus désirant utiliser la mêmes ressource.
On considère ici un système de réservation de places. Le système propose deux fonctions aux utilisateurs :

- la consultation du nombre de places disponibles, réalisée par l'appel à la fonction `consultation()`,
- et la réservation d'une place, réalisée par l'appel à la fonction `reservation()`.

Nous utiliserons un système de verrou basé sur l'objet Sémaphore géré par le système d'exploitation GNU/Linux pour contrôler la concurrence d'accès à une ressource. On considère ici que le sémaphore sera initialisé avec un nombre de jeton égal à 1, autorisant un seul accès concurrent à l'objet contrôlé.
Le verrou est manipulé à l'aide de 3 fonctions suivantes (définies dans fichiers `sem_func.c` et `sem_func.h`) :

- `init_verrou()` : initialise un verrou de type sémaphore avec un jeton, retourne 0 si l'initialisation est réussie ou -1 sinon.
- `prendre_verrou()` : permet de demander au système d'exploitation un jeton associé au sémaphore si disponible, et bloque le l'exécution du processus dans le cas contraire en attente que le jeton soit relâché par le processus qui le détient. Lorsqu'un processus est bloqué, celui-ci est mis en sommeil et ajouté dans la file d'attente associé au sémaphore par le système d'exploitation.
- `prendre_verrou()` : permet de rendre un jeton associé au sémaphore, cela permet de réveiller un processus en sommeil situé dans la file d'attente associé au sémaphore.

Soit le programme ci-dessous implémentant le système de réservation de places :

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

```
#include <string.h>
#include "sem_func.h"

int nb_places=5;

void reservation(int *p)
{
   int j;
   for(j=0; j<10; j++)
   {
      printf("\n[Proc. Reservation-%d] Debut réservation place.\n", *p);
      if(nb_places>0)
      {
         sleep(2);
         nb_places=nb_places-1;
         printf("[Proc. Reservation-%d] Place réservée.\n", *p);
       }
       else printf("[Proc. Reservation-%d] Aucune place disponible!\n",
*p);

       printf("[Proc. Reservation-%d] Fin réservation.\n", *p);
       fflush(stdout);
       sleep(2);
   }
}


void consultation(void)
{
   int i;
   for(i=0; i<10; i++)

   {
      printf("\n[Proc. Consultation] Debut consultation places.\n");
      printf("[Proc. Consultation] Il reste %d places.\n", nb_places);
      printf("[Proc. Consultation] Fin consultation.\n");
      fflush(stdout);
      sleep(2);
   }
}


void main(void)
{
   int k, val, val2, ret;
   pthread_t num_thread[3];

   ret=init_verrou();
   if(ret==-1) exit(-1);

   if(pthread_create(&num_thread[0], NULL, (void *(*)())consultation,
NULL) == -1)    perror("pb pthread_create\n");

   val=1;
 if(pthread_create(&num_thread[1], NULL, (void *(*)())reservation, &val)
== -1)    perror("pb pthread_create\n");

   for(k=0; k<3; k++)
   {
      if(num_thread[k]>0) pthread_join(num_thread[k], NULL);

   }
```

```
    exit(0);
}
```

Question 1 – Analysez le programme ci-dessus et exécutez ce programme.

Question 2 – Modifiez le programme ci-dessus afin d'ajouter un nouveau fil d'exécution (thread) qui exécutera le code de la fonction `reservation()`.
Exécutez le programme modifié. Qu'observez-vous ?

Correction :
Il y a maintenant deux processus modifiant la variable `nb_places` *et dans certaines exécutions la concurrence d'accès sur cette variable conduit à obtenir un état incohérent. On voit que la variable* `nb_places` *peut contenir une valeur négative, ce qui ne devrait pas arriver pour ce système de réservation de places considéré ici.*

Question 3 – Dans le programme ci-dessus, indiquez quelle(s) est ou sont la ou les ressources partagée(s) et quelle(s) est ou sont la ou les sections critiques.

Correction :
Dans ce programme, il y a une seule ressource partagée qui est la variable `nb_places` *utilisées par les processus consultation et réservation.*
Les sections critiques concernent le corps des fonction `reservation()` *et* `consultation()` *qui manipulent la variable partagée.*

Question 4 – Proposez une modification du programme afin de résoudre le problème et garantir une exécution correcte.

Correction :
Pour cela, il faut encadrer les sections critiques par l'appel aux fonctions `prendre_verrou()` *et* `rendre_verrou()`. *Cela permettra de réaliser une sychronisation par exclusion mutuelle entre les processus afin de n'autoriser qu'au plus un processus à être dans une section critique simultanément.*

Question 5 – Le code source fourni pour cet exercice utilise l'implémentation System V des Sémaphores. Modifiez le code source fournis afin d'utiliser l'implémentation POSIX des Sémaphores afin de modifier le code des fonctions : `init_verrou()`, `prendre_verrou()`, et `rendre_verrou()`.