TRAVAUX PRATIQUES 4
Synchronisation de processus sous Linux

L'objectif de ce TP est de revenir sur la notion d'interblocage et la 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. Pour cela, vous pouvez utiliser un éditeur de texte comme `kwrite` en exécutant la commande ci-dessous dans le répertoire contenant `prog.c` : `$>kwrite prog.c &`

Rappel : Pour compiler vos programmes en langage C afin de les exécuter vous pouvez utiliser le compilateur gcc comme suit : `$> gcc -o exo1 exo1.c`

Exercice 1

Dans cet exercice, nous allons nous intéresser à la concurrence d'accès à la mémoire par des processus légers.
Soit le programme suivant :

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

int tab_heure[4];
void heure()
{
     while(1) {
           tab_heure[0] = (tab_heure[0] + 1)%60;

           if (tab_heure[0] == 0)
           {
              tab_heure[1] = (tab_heure[1] + 1)%60;
           }
           if (tab_heure[1] == 0)
           {
             tab_heure[2] = (tab_heure[2] + 1)%24;
           }
           if (tab_heure[2] == 0)
           {
             tab_heure[3] = tab_heure[3] + 1;
           }
           sleep(1);
     }
}

void main(void)
{
     pthread_t num_thread;

     tab_heure[0]=1;
```

```
      tab_heure[1]=4;
      tab_heure[2]=5;
      tab_heure[3]=2;

      if(pthread_create(&num_thread, NULL, (void *(*)())heure, NULL) == -1)
         perror("pb pthread_create\n");

      while(1)
      {
         printf("L'heure du système est : %d jour,%d heure,%d minutes,%d
      secondes\n", tab_heure[3], tab_heure[2], tab_heure[1], tab_heure[0]);

        sleep(5);
      }

 }
```

Question 1 – A votre avis, que pourrait il se produire sur l'interaction entre les deux processus du programme ci-dessus ?
Supposons par exemple que le tableau `tab_heure` contienne les valeurs suivantes :

```
      tab_heure[0] = 59     // seconde
      tab_heure[1] = 59     // minute
      tab_heure[2] = 2      // heure
      tab_heure[3] = 4      // jour
```

et que les deux threads père et fils soient lancés en même temps. L'heure affichée par le thread parent est elle toujours juste ?

Question 2 – Que doit-on ajouter à ce programme pour que le problème identifié ci-dessus ne se produise pas. Combien d'accès concurrent peut accepter le tableau `tab_heure` ?

Nous vous fournissons les fonctions nécessaires pour la synchronisation des processus :

- `init_verrou()` : cette fonction permet d'initialiser la structure de synchronisation,
- `prendre_verrou()` : cette fonction est à appeler à l'entrée de la section critique,
- `rendre_verrou()` : cette fonction est à appeler en sortie de la section critique.

Voici ci-dessous le code de ces trois fonctions :

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

      int semid;
      struct sembuf operation;

      void init_verrou(void)
      {
        semid = semget(12, 1, IPC_CREAT|IPC_EXCL|0600);
        semctl(semid, 0, SETVAL, 1);
      }

      void prendre_verrou(void)
      {
        /* opération P */
        operation.sem_num= 0;
        operation.sem_op = -1;
```

```
  operation.sem_flg= 0;

  semop(semid, &operation, 1);
}

void rendre_verrou(void)
{
  /* opération V */
  operation.sem_num= 0;
  operation.sem_op= 1;
  operation.sem_flg= 0;

  semop(semid, &operation, 1);
}
```

Question 3 – Rappelez ce que l'on appel une section critique.

Question 4 – Identifiez la section critique et utilisez ces trois fonctions pour intégrer une synchronisation correcte dans le programme donné ci-dessus comme demandé à la Question 2. Quel schéma de synchronisation faut il utiliser ?

Question 5 – Créez deux fichiers `ex_synchro.c` et `sem_func.c` contenant respectivement le programme fournit dans l'énoncé et les trois fonctions de synchronisation. Afin d'exécuter le programme, réalisez une compilation séparée pour générer le module objet associé aux différents fichiers fournis comme indiqué ci-dessous :

```
$> gcc –lpthread –c ex_synchro.c
$> gcc –c sem_func.c
$> gcc –lpthread –o ex_synchro ex_synchro.o sem_func.o
```

Les deux premières commandes permettent de générer respectivement les codes objets `ex_synchro.o` et `sum_func.o`, tandis que la dernière commande permet de réaliser l'édition des liens du programme et générer l'exécutable `ex_synchro`.

Exercice 2

Dans cet exercice, nous allons nous intéresser au problème d'interblocage qui peut arriver entre plusieurs processus utilisant les mêmes ressources.

On considère ici un système de réservation de places. Le système propose deux fonctions aux utilisateurs :

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

L'exécution de chacune de ces fonctions nécessite deux ressources :

1. accéder à une base de données,
2. obtenir un identifiant valide (nécessaire durant toute la transaction avec la base de données).

Ces deux étapes sont encadrées par un système de verrou basé sur l'objet Sémaphore pour contrôler la concurrence d'accès à ces deux ressources. Ainsi, deux sémaphores sont utilisées :

- **Sem1** : indiquant le nombre de connexion disponible à la base de données,
- **Sem2** : indiquant le nombre d'identifiants valides disponibles.

On considère ici que ces deux sémaphores sont initialisées avec un nombre de jeton égal à 1, autorisant un seul accès concurrent à l'objet qu'elle contrôle respectivement.
Chaque ressource est manipulées grâce à deux fonctions pour obtenir et relâcher la ressource, qui sont respectivement `connexion()` et `deconnexion()` pour **Sem1** et `prendre_num()` et `rendre_num()` pour **Sem2**.

Question 1 – Rappelez ce que représente la notion d'interblocage entre processus.

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

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

int nb_places=5;

void reservation(void)
{
     connexion();
     prendre_num();

     printf("Debut réservation place.\n");
     if(nb_places>0)
     {
           nb_places=nb_places-1;
           sleep(5);
           printf("Place réservée.\n");
     }
     else printf("Aucune place disponible!\n");

     printf("Fin réservation.\n");

     rendre_num();
     deconnexion();
}

void consultation(void)
{
     prendre_num();
     connexion();

     printf("Debut consultation places.\n");
     sleep(5);
     printf("Il reste %d places.\n", nb_places);

     printf("Fin consultation.\n");

     deconnexion();
     rendre_num();
}
```

```
void main(void)
{
  pthread_t num_thread;
  char c[2];

  init_verrous();

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

  while(1)
  {
    printf("Voulez vous réserver une place ? (o/n)\n");
     scanf("%c\n", &c);

     if(strcmp(&c, "o") == 0)
     {
           reservation();
     }
    sleep(2);
  }
}
```

Question 2 – Analysez le programme ci-dessous et montrez que ce programme risque de conduire à un interblocage.

Question 3 – Modifiez le programme ci-dessus afin de supprimer le risque d'interblocage.

Question 4 – Modifiez les codes sources fournis et exécutez le nouveau programme afin de vérifier si il se produit des interblocages.

Question 5 – On considère maintenant que le nombre d'identifiants valides en simultané est de 3. Indiquez comment modifier le code de la question précédente pour prendre en compte ce changement.