TP – Multitâche
Découverte des Mutex et des Semaphores

Exercice 1 Exclusion mutuelle : Mutex

Ecrire un programme qui permet à 5 threads d'écrire un message complet à l'écran (par exemple "Message du thread n°2") sans être dérangé par les autres threads.
L'affichage du message se fera lettre par lettre grâce à une temporisation de 0,5 seconde entre chaque affichage de lettre. Le but est d'illustrer l'utilisation d'un mutex.
Vous créerez vos threads dans la boucle principale et chaque thread doit pouvoir ensuite afficher son message sans être interrompu.

Vous pouvez vous aider du programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h> //Ne pas oublier d'inclure le fichier time.h
#include <pthread.h>
/* strcat example */
#include <string.h>

pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
#define NB_THREADS 10

typedef struct
{
  int n;
}data;

void printcharbychar(char *s, int n){
        int i=0;
        while(s[i]!='\0' ){
                printf("%c",s[i]);
                i++;
                sleep(1);
        }
        printf("%d \n",n);
        sleep(1);
}
```

```
void *mytask(void *p_data)

{

  char s1[50]="Message du thread numero ";

  data *info =p_data;

  pthread_mutex_lock( &mutex1 );
  printcharbychar(s1,info->n);
  pthread_mutex_unlock( &mutex1 );
}



int main (void)
{
  printf("main start\n");
  int i;
  pthread_t threads[NB_THREADS];
  data infos[NB_THREADS];

  for(i=0;i<NB_THREADS;i++){
        infos[i].n=i;
        pthread_create (&threads[i], NULL, mytask,&infos[i]);
  }

  for(i=0;i<NB_THREADS;i++){
        pthread_join (threads[i], NULL);

  }

   printf("main end\n");

}
```

Exercice 2 Exclusion mutuelle : Semaphore

Réaliser le même programme que l'exercice 1 mais avec un sémaphore.
Vous pouvez vous aider du bout de code suivant :

```
#include <semaphore.h>

#define SEMNAME "monsem"
sem_t *attente;

int main (void)
{


  printf("main start\n");
  int i,val;
  attente = sem_open(SEMNAME, O_CREAT|O_EXCL, 0777, 1);
   if(attente == SEM_FAILED)
   {
         perror("unable to create semaphore");
         sem_unlink(SEMNAME);
         exit( -1 );
   }


  //sem_wait(attente);
  //sem_post(attente);


    /* Close the semaphore */
  sem_close(attente);
  /* Remove the semaphore */
  sem_unlink(SEMNAME);

   printf("main end\n");

}
```

Exercice 3 Barrière de synchronisation

Une barrière de synchronisation permet de garantir qu'un certain nombre de tâches aient passé un point spécifique. Ainsi, chaque tâches qui arrivera sur cette barrière devra attendre jusqu'à ce que le nombre spécifié de tâches arrivées à cette barrière soient atteint.

Algorithme

Pour réaliser une barrière de synchronisation, il faut disposer de deux sémaphores et d'une variable :

- Un sémaphore MUTEX (initialisé à 1) protégeant la variable.
- Un sémaphore ATTENTE (initialisé à 0) permettant de mettre en attente les tâches.
- Une variable Nb_Att (initialisée à 0) permettant de compter le nombre de tâches déjà arrivées à la barrière de synchronisation.

Il faut encore définir la constante N qui indique le nombre de tâches devant arriver à la barrière avant de l'ouvrir.

La barrière de synchronisation utilise l'algorithme suivant :

```
Barriere :
        P(MUTEX)
        Nb_Att++
        SI Nb_Att==N ALORS
                POUR I DE 1 à N-1 FAIRE
                        V(ATTENTE)
                FIN POUR
                Nb_Att=0
                V(MUTEX)
        SINON
                V(MUTEX)
                P(ATTENTE)
        FIN SI
```

Exemple d'utilisation

Les barrières de synchronisation peuvent être utilisées pour

- Garantir qu'une ou plusieurs tâches ont effectué une opération particulière.
- Attendre la fin d'un ensemble de tâches.

- Ecrivez un programme qui lance 10 threads qui ne font rien pendant un temps aléatoire (max 5 sec)

et qui s'attendent grâce au mécanisme décrit ci-dessus.

Exercice 3 Problème du lecteur-rédacteur

Le problème

Supposons qu'une base de données ait des lecteurs et des rédacteurs, et qu'il faille programmer les lecteurs et les rédacteurs de cette base de données.

Les contraintes sont les suivantes :

- plusieurs lecteurs doivent pouvoir lire la base de données en même temps ;
- si un rédacteur est en train de modifier la base de données, aucun autre utilisateur (ni rédacteur, ni même lecteur) ne doit pouvoir y accéder.

Solutions

Il est assez simple de faire en sorte que le rédacteur soit mis en attente tant qu'il y a encore des lecteurs. Mais cette solution présente de gros problèmes, si le flux de lecteurs est régulier, le rédacteur pourrait avoir à patienter un temps infini.

Il existe donc une deuxième solution, qui consiste à mettre en attente tous les lecteurs ayant adressé leur demande d'accès après celle d'un rédacteur.

Solution avec utilisation des sémaphores et priorité aux lecteurs

La solution suivante permet de résoudre le problème des lecteurs et des rédacteurs en donnant priorité aux lecteurs. Cette solution nécessite trois sémaphores et une variable, à savoir :

- Un sémaphore M_Lect, initialisé à 1 qui permet de protéger la variable Lect. Il s'agit donc d'un mutex.
- Un sémaphore M_Red, initialisé à 1 qui permet de bloquer les tâches de rédaction. Il s'agit donc à nouveau d'un mutex.
- Un sémaphore Red, initialisé à 1 qui permet aussi de bloquer les tâches de rédaction.
- Une variable Lect qui compte le nombre de lecteurs.

Cette solution utilise les quatre méthodes suivantes :

Commencer une lecture

```
Commencer_Lire :
P(M_Lect)
Lect++
SI Lect==1 ALORS
        P(Red)
FIN SI
V(M_Lect)
```

Cette méthode incrémente le nombre de lecteurs. Ensuite, s'il s'agit du premier lecteur à essayer d'entrer, elle n'autorise l'entrée que s'il n'y a pas de rédaction en cours. Dans le cas contraire, la méthode doit attendre que la rédaction soit finie. L'utilisation de deux sémaphores pour les rédacteurs permet d'induire la priorité aux lecteurs.

Finir une lecture

```
Finir_Lire :
P(M_Lect)
Lect--
SI Lect==0 ALORS
        V(Red)
FIN SI
V(M_lect)
```

Cette méthode décrémente le nombre de lecteurs. Ensuite, s'il s'agit du dernier lecteur à sortir, elle autorise les rédacteurs à entrer (si nécessaire).

Commencer une écriture

Commencer_Ecrire
P(M_Red)
P(Red)
Finir une écriture
Finir_Ecrire
V(Red)
V(M_Red)

Ecrivez un programme qui lance 10 threads lecteurs qui ne font rien pendant un temps aléatoire (max 5 sec) et qui viennent lire la donnée (un char déclaré en variable globale) au bout d'un certain temps (affichage de la donnée lue). Le programme lancera aussi 2 autres threads rédacteurs qui iront écrire dans la donnée une information aléatoire (la lettre A,B,C,D,E ou F) au bout d'un temps aléatoire. Les threads tourneront en boucle.