TP N° 4 : synchronisation des processus par des sémaphores

Sémaphores

Les threads peuvent communiquer entre eux par les variables globales. Ils peuvent se synchroniser par attente active, sémaphores, verrous et variables conditionnelles. On s'intéresse dans ce tp par les sémaphores.

Pour utiliser les sémaphores il faut, premièrement, inclure le fichier de déclarations standard :

#include <semaphore.h>

Deuxièmement, déclarer un descripteur de sémaphore global aux fonctions exécutées par les threads qui l'utiliseront :

sem_t sema;

Troisièmement, il faut initialiser les descripteurs comme suit :

sem_init(&sema,0, 1);

Ou le troisième argument correspond à la valeur initiale du compteur du sémaphore (le deuxième argument mis à 0 signifie que le sémaphore peut-être partagé par tous les threads du processus).

Finalement, les opérations P et V sont implémentées par les fonctions sem_wait() respectivement Sem_post(). Donc, une section critique peut être implémentée comme suit :

```
sem_wait(&sema); /* P(sema) */
/* section critique */
sem_post(&sema); /* V(sema) */
```

Voila les primitifs sémaphores :

`int sem_init(sem_t *semaphore, int pshared, unsigned int valeur)`
Création d'un sémaphore et préparation d'une valeur initiale.

`int sem_wait(sem_t * semaphore) ;` Opération P sur un sémaphore.

`int sem_post(sem_t * semaphore) ;` Opération V sur un sémaphore.

`int sem_getvalue(sem_t * semaphore, int * sval) ;` Récupérer le compteur d'un sémaphore.

`int sem_destroy(sem_t * semaphore) ;` Destruction d'un sémaphore.

La fonction sched_yield( )

Un processus peut relâcher volontairement le processeur sans le bloquer en utilisant la function « **sched_yield** ». Le processus est alors placé à la fin de la queue et un nouveau processus est mis en marche.

Exemple :

```
#include <semaphore.h>

int main()
{
 sem_t semaphore;
 int compteur = 0;
 sem_init( &semaphore, 0, 1 ); /* On crée la sémaphore avec la valeur 1 */
 sem_wait( &semaphore ); /* On « prend » la sémaphore */
 compteur++;
 sem_post( &semaphore ); /* On « libère » la sémaphore */
 sem_destroy( &semaphore );
}
```

Le sémaphore binaire Mutex (Mutuel Exclusion)

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

Initialise le mutex avec les attributs attr (NULL pour les options par défaut) et renvoie un code d'erreur.

- **int pthread_mutex_destroy**(pthread_mutex_t *mutex); Détruit le mutex.
- **int pthread_mutex_lock**(pthread_mutex_t *mutex); WAIT sur le mutex.
- **int pthread_mutex_unlock**(pthread_mutex_t *mutex); POST sur le mutex.

Exercice N°1

Vous avez le programme suivant :

```
#include <pthread.h>
#include <stdio.h>
int a =0;
void* increment(void *arg);
void* decrement(void *arg);
int main ( )
```

```
{
     pthread_t threadA;
     pthread_t threadB;
     printf("programme principale : a = %d\n", a);
     pthread_create(&threadA, NULL, increment,NULL);
      pthread_create(&threadB,NULL,decrement,NULL);
      pthread_join(threadA,NULL);
      pthread_join(threadB,NULL);
      printf("programme principale, fin threads : a =%d
\n",a);
  return 0;
}

void* decrement(void *arg)
{

        a = a - 1 ;
        printf("thread B decrement : a=%d\n", a);

        return (NULL);
}

void* increment (void *arg)
{
        a = a + 1;
        printf("thread A increment : a = %d\n", a);

        return (NULL);
}
```

Q1 : Compiler et exécuter le programme. Analyser puis expliquer le résultat d'exécution.

Q2 : C'est quoi la cause de ce problème ? Comment peut-on le résoudre ?

Q3 : Proposer une solution pour résoudre le problème de l'incohérence des données rencontré dans ce programme en utilisant un sémaphore d'exclusion mutuelle pour protéger la variable partagée.

Exercice N°2

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

/*void *T1(void *arg) {

printf("2");
printf("0");
printf("2");
printf("1\n");
pthread_exit(NULL);
}*/
void *T2(void *arg) {

printf("S");
printf("Y");
printf("S");
pthread_exit(NULL);
}

void *T3(void *arg) {

printf("T");
printf("E");
printf("M");
printf("E\n");

pthread_exit(NULL);
}

int main(){


pthread_t tid1,tid2,tid3;
pthread_create(&tid1,NULL,T1, NULL);
pthread_create(&tid2,NULL,T2, NULL);
pthread_create(&tid3,NULL,T3, NULL);
pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
pthread_join(tid3,NULL);


exit(0);
}
```

Q1 :compilez ce programme.

Q2 : synchronisez les trois threads de ce programme en utilisant les sémaphores pour avoir l'affichage suivant : SYSTEME 2021

Exercice N°3

```
#include<stdio.h>

#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
sem_t sem1, sem2;

void *T1(void *arg) {
printf("S");
printf("S");
printf("E");
printf("E\n");
pthread_exit(NULL);
}
void *T2(void *arg) {
printf("Y");
printf("T");
printf("M");

pthread_exit(NULL);
}
int main(){
pthread_t tid1, tid2;
pthread_create(&tid1,NULL,T1, NULL);
pthread_create(&tid2,NULL,T2, NULL);
pthread_join(tid1,NULL);
pthread_join(tid2,NULL);
exit(0);
}
```

Q1 : Initialiser les sémaphores sem1, sem2 dans la fonction main() et insérer dans les fonctions T1 et T2 des instructions sem wait(...), sem post(...) pour que le programme modifié affiche obligatoirement **SYSTEME.**

Exercice N°4

création de deux threads

On crée trois tâches (threads) pour exécuter chacune des trois fonctions : une affichera des étoiles '*' des dièses '#'et arobase @ .

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
//Fonctions correspondant au corps d'un thread(tache)
void *etoile(void *inutilise);
void *diese(void *inutilise);

void *arobase(void *inutilise);



//Remarque:le prototype d'une tâche doit être :
void *(*start_routine)(void *)

int main(void)
{
pthread_t thrEtoile, thrDiese,thrarobase;
//les ID des de 3 thread
setbuf(stdout, NULL);
//pas de tampon sur stdout
printf("Je vais créer et lancer 3 threads");
pthread_create(&thrEtoile, NULL, etoile, NULL);
pthread_create(&thrDiese, NULL, diese, NULL);

pthread_create(&thrarobase, NULL, arobase, NULL);

//printf("J'attends la fin des 3 threads\n");
pthread_join(thrEtoile, NULL);
pthread_join(thrDiese, NULL);
pthread_join(thrarobase, NULL);

printf("\nLes 3 threads se sont termines\n");
printf("Fin du thread principal\n");
pthread_exit(NULL);
return EXIT_SUCCESS;
}

void *etoile( void *inutilise)
{ int i;
char c1 = '*';
for(i=1;i<=200;i++)
{
write(1, &c1, 1);
// écrit un caractère sur stdout(descripteur 1)
}

return NULL;
}
```

```
void *diese(void *inutilise)
int i;
char c1 = '#';
for(i=1;i<=200;i++)
{
write(1, &c1, 1);
}
return NULL;
}

void *arobase(void *inutilise)
int i;
char c1 = '@';
for(i=1;i<=200;i++)
{
write(1, &c1, 1);
}
return NULL;
}
```

Q1 : compilez ce programme. Quel est le problème rencontré ?

synchronisez les trois threads de ce programme en utilisant les sémaphores pour avoir l'affichage suivant :

-Q2) *#@*#@*#@

-Q3)***###@@@***###@@@

-Q4)**#@**#@**#@**