TP3

FERREIRA MATHIEU

Compte rendu TP3

Question 1 - Signalisation :

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

void a();
void b();

/* Global variables */
int x = 0;
sem_t *sem_s1;


/* Thread function */
void *p1(void *arg)
{
    a();
    /* send signal to the created thread */
    sem_post(sem_s1);

}
void *p2(void *arg)
{
    /* wait for signal from main thread */
    sem_wait(sem_s1);
    b();
}

void a(){
    sleep(10);
    printf("X = %d\n", x);
    x=55;
}

void b(){
  printf("X = %d\n", x);
```

```
}


void main () {
    sem_s1 = sem_open("sem_s1", O_CREAT, 0644, 0);


    pthread_t thread1, thread2;
    /* semaphore sync should be initialized by 0 */
    if (sem_s1 == SEM_FAILED) {
        perror("Could not initialize mylock semaphore");
        exit(2);
    }
    if (pthread_create(&thread1, NULL, p1, NULL) < 0) {
        perror("Error: thread cannot be created");
        exit(1);
    }
    if (pthread_create(&thread2, NULL, p2, NULL) < 0) {
        perror("Error: thread cannot be created");
        exit(1);
    }
    /* wait for created thread to terminate */
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    /* destroy semaphore sync */
    sem_close(sem_s1);

    exit(0);
}
```

Pour cette question il fallait rajouter l’include “#include <fcntl.h>” ainsi que rajouter void a() et void b().

Au niveau des p1 et p2 on rajoute :

```
sem_post(sem_s1);
...
sem_wait(sem_s1);
```

Dans le main il manquait la ligne :

```
sem_s1 = sem_open("sem_s1", O_CREAT, 0644, 0);
```

Et il fallait modifier :

```
    if (sem_s1 == SEM_FAILED) {
        ...
    }
    if (pthread_create(&thread1, NULL, p1, NULL) < 0) {
        ...
    }
    if (pthread_create(&thread2, NULL, p2, NULL) < 0) {
        ...
    }
```

Ainsi que changer le destroy en :

```
sem_close(sem_s1);
```

Et lorsque l’on teste le code on a bien pour résultat :

```
X = 0
X = 55
```

Question 2 - Rendez-vous :

Pour la question 2 il faut changer le a() et b() en :

```
void a1();
void a2();
void b1();
void b2();
```

Rajouter la variable globale :

```
sem_t *sem_s1, *sem_s2;
```

Transformer les p1 et p2 en :

```
/* Thread function */
void *p1(void *arg)
{
    a1();
    /* send signal to the created thread */
    sem_post(sem_s1);
    sem_wait(sem_s2);
    a2();

}
void *p2(void *arg)
{
    b1();
    /* wait for signal from main thread */
    sem_post(sem_s2);
    sem_wait(sem_s1);
    b2();
}
```

Modifier les testes par :

```
void a1() {
    printf("a1()\n");
}
void a2() {
    printf("a2()\n");
}
void b1() {
    printf("b1()\n");
}
void b2() {
    printf("b2()\n");
}
```

Au niveau du main on doit rajouter :

```
void main () {
    ...
    sem_s2 = sem_open("sem_s2", O_CREAT, 0644, 0);
    ...
    if (sem_s2 == SEM_FAILED) {
        perror("Could not initialize mylock semaphore");
        exit(2);
```

```
    }
    ...
    sem_close(sem_s2);
    ...
}
```

On a pour résultat :

```
a1()
b1()
b2()
a2()
```

On voit bien que a1() à lieu avant b2() et b1() avant a2().

Question 3 - Exclusion mutuelle :

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

/* Global variables */
int x = 0;
sem_t m;

/* Thread function */
void *thread(void *arg)
{
    int i = *((int*) arg);
    /* Wait until the semaphore is available */
    sem_wait(&m);
    /* critical section */
    x = x + 1;
    /* Release the semaphore */
    sem_post(&m);
    printf("x:%d, i:%d\n",x,i);
    pthread_exit(NULL);
}

void main ()
```

```
{
    pthread_t tid[10];
    int i, args[10];
    /* Semaphore m should be initialized by 1 */
    if (sem_init(&m, 0, 1) == -1) {
        perror("Could not initialize mylock semaphore");
        exit(2);
    }
    /* Create TEN threads */
    for (i=0; i<10; i++)
    {
        args[i] = i;
        if (pthread_create(&tid[i], NULL, thread, &args[i]) < 0) {
            perror("Error: thread cannot be created");
            exit(1);
        }
    }
    /* Wait for all created thread to terminate */
    for (i=0; i<10; i++) pthread_join(tid[i], NULL);
    printf("Final value of x is %d\n", x);
    /* Destroy the semaphore */
    sem_destroy(&m);
    exit(0);
}
```

Pour cette question il fallait donc rajouter les éléments de *thread :

```
    int i = *((int*) arg);
    /* Wait until the semaphore is available */
    sem_wait(&m);
    ...
    /* Release the semaphore */
    sem_post(&m);
    ...
    pthread_exit(NULL);
```

Au niveau du main :

Il faut ajouter une initialisation :

```
int i, args[10];
```

Ainsi que :

```
for (i=0; i<10; i++)
{
    args[i] = i;
    ...
}
```

Et pour finir un destroy comme cela :

```
...
/* Destroy the semaphore */
sem_destroy(&m);
...
```

Et ce programme nous donne bien le résultat ci-dessous :

```
x:1, i:1
x:4, i:3
x:2, i:0
x:3, i:2
x:5, i:4
x:6, i:5
x:7, i:6
x:8, i:7
x:9, i:8
x:10, i:9
Final value of x is 10
```

Question 4 :

Tout d’abord il faut initialiser les variable globale

```
#define SIZE_TAMPON 5 //taille du tampon

sem_t *semC; // semaphore du consommateur
sem_t *semP; // semaphore du producteur

volatile char buffer[SIZE_TAMPON]; // initialisation du tampon

char caractere = 'a'; // premier caractère à afficher
```

Ensuite on définie la producer

```
void* producer(void* args) {
    int ip = 0; //place dans le buffer
    while (1) {

        sleep(1);
        // Ajout au tampon
        sem_wait(semP);
        buffer[ip] = caractere; //ajout du caractere à son emplacement dans le buffer
        sem_post(semC);

        printf("On a produit [%d] :\n", ip);
        printf("\tcaractere produit : %c\n",caractere);


        caractere++; // on passe au caractere suivant
        ip++; // on agmente le compteur de un

        // si jamais on est à la fin du tampon
        if (ip == SIZE_TAMPON) {
            ip = 0;
        }
    }
}
```

Puis le consumer :

```
void* consumer(void* args) {
    int ic = 0;  //place dans le buffer
    while (1) {
        int mess; // init du message qu'on recevra

        // Retirer du tampon
        sem_wait(semC);
        mess = buffer[ic]; // place message dans le buffer
        sem_post(semP);

        printf("On a recupere[%d] :\n", ic);
        printf("\tcaractere recupere : %c\n", mess);

        ic++; // augmentation du compteur

        // si jamais on est au bout du compteur
        if (ic == SIZE_TAMPON) {
            ic = 0;
        }
```

```
        sleep(1);
    }
}
```

Et enfin le main :

```
int main(int argc, char* argv[]) {
    pthread_t prod_th, cons_th; // on initialise les threads

    // initialisation des semaphores
    semP = sem_open("semP", O_CREAT, 0644, 0);
    semC = sem_open("semC", O_CREAT, 0644, 0);

    // creation des threads
    pthread_create(&prod_th, NULL, producer, NULL);
    pthread_create(&cons_th, NULL, consumer, NULL);

    // join des threads
    pthread_join(prod_th, NULL);
    pthread_join(cons_th, NULL);

    // on destroy les semaphores
    sem_destroy(semP);
    sem_destroy(semC);

    return 0;
}
```

On obitient bien le résultat souhaité :

```
On a produit [0] :
        caractere produit : m
On a recupere[0] :
        caractere recupere : m
On a produit [1] :
        caractere produit : n
On a recupere[1] :
        caractere recupere : n
On a recupere[2] :
        caractere recupere : o
On a produit [2] :
        caractere produit : o
On a produit [3] :
        caractere produit : p
On a recupere[3] :
```

```
        caractere recupere : p
On a produit [4] :
        caractere produit : q
On a recupere[4] :
        caractere recupere : q
On a produit [0] :
        caractere produit : r
On a recupere[0] :
        caractere recupere : r
```

Question 5 :

Je n’ai pas reussi à faire cette question