TP3 : Threads POSIX

RS : Réseaux et Systèmes

Deuxième année

★ Exercice 1: Création et attente de threads

▷ **Question 1:** Écrivez un programme ayant le comportement suivant :

1. Des threads sont créés (leur nombre étant passé en paramètre lors du lancement du programme) ;
2. Chaque thread affiche un message (par example « hello world ! ») ;
3. Le thread « principal » attend la terminaison des différents threads créés.

★ Exercice 2: Identification des threads

▷ **Question 1:** Modifiez le programme de la question précédente pour que chaque thread affiche :

— son PID (avec `getpid()`) ;

— La valeur opaque retournée par `pthread_self`, par exemple avec :
`printf("%p\n", (void *) pthread_self());`

Réponse

solution aux exercices 1+2

```c
/* à compiler avec gcc -o E12 -pthread E12.c */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>


void * helloworld(void * arg) {
    printf("Hello world! pid=%d pthread_self=%p\n",
            getpid(), (void *) pthread_self());
    return NULL;
}

int main(int argc, char** argv) {
    int i, nb;
    pthread_t * threads;
    nb = atoi(argv[1]);
    threads = malloc(nb * sizeof(pthread_t));
    for (i = 0; i < nb; i++) {
        pthread_create(
           &threads[i], NULL, helloworld, NULL);
    }
    printf("Début de l'attente\n");
    for (i = 0; i < nb; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Fin de l'attente\n");
    return 0;
}
```

```
$ gcc -Wall -o E12 -pthread E12.c
$ ./E12 5
Hello world! pid=22755
pthread_self=0x7fef3c0ab700
Hello world! pid=22755
pthread_self=0x7fef3c8ac700
Hello world! pid=22755
pthread_self=0x7fef3d0ad700
Hello world! pid=22755
pthread_self=0x7fef3b8aa700
Début de l'attente
Hello world! pid=22755
pthread_self=0x7fef3b0a9700
Fin de l'attente
```

Fin réponse

★ Exercice 3: Passage de paramètres et exclusion mutuelle

▷ **Question 1:** Modifiez le programme de la question précédente pour passer son numéro d'ordre à chaque thread. Chaque thread doit ensuite l'afficher. Vérifiez que le numéro d'ordre affiché par chaque thread est bien différent (corrigez votre programme le cas échéant).

Réponse

Le gros piège ici est que si on fait qqchose comme :

```c
for (i = 0; i < nb; i++) {
    pthread_create(
       &threads[i], NULL, helloworld, &i);
}
```

Alors la variable **i** dont on passe l'adresse va continuer à être modifiée par le thread principal. Du coup, on se retrouve avec une sortie comme :

```
$ ./E3 5
Hello world! i=3
Hello world! i=5
```

```
Hello world! i=4
Hello world! i=2
Début de l'attente
Hello world! i=2
Fin de l'attente
```

(Note : deux fois i=2, et i=5 alors qu'on devrait compter de 0 à 4 !)

La bonne solution est d'utiliser un tableau et d'y recopier l'argument pour que chaque thread ait bien sa zone mémoire à lui :

solution exercice 3.1

```c
/* à compiler avec gcc -o E12 -pthread E12.c */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>


void * helloworld(void * arg) {
    printf("Hello world! i=%d\n",
            * (int*)arg);
    return NULL;
}

int main(int argc, char** argv) {
    int i, nb;
    int * args;
    pthread_t * threads;
    nb = atoi(argv[1]);
    threads = malloc(nb * sizeof(pthread_t));
    args = malloc(nb * sizeof(int));
    for (i = 0; i < nb; i++) {
        args[i] = i;
        pthread_create(
            &threads[i], NULL, helloworld, &args[i]);
    }
    printf("Début de l'attente\n");
    for (i = 0; i < nb; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("Fin de l'attente\n");
    return 0;
}
```

```
$ ./E3 5
Hello world! i=1
Hello world! i=4
Hello world! i=2
Hello world! i=3
Début de l'attente
Hello world! i=0
Fin de l'attente
```

Fin réponse

▷ **Question 2:** Déclarez une variable globale `somme` initialisée à 0. Chaque thread doit, dans une boucle, ajouter 1 000 000 fois son numéro d'ordre à cette variable globale (on veut bien faire 1 000 000 additions par thread, pas juste une). Affichez la valeur obtenue après la terminaison de tous les threads.

▷ **Question 3:** Avec 5 threads (numérotés de 0 à 4), on devrait obtenir (0+1+2+3+4)*1 000 000 = 10 000 000. Corrigez votre programme s'il n'affiche pas systématiquement ce résultat.

Réponse

Le piège ici est évidemment qu'on modifie une variable globale depuis tous les threads, sans protéger son accès. Il faut utiliser un mutex.

solution exercice 3.3

```c
/* à compiler avec gcc -o E12 -pthread E12.c */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>

int somme = 0;                          ////
pthread_mutex_t mutex;                  ////

void * helloworld(void * arg) {
    int ordre = * (int*) arg;
    int i;
    for (i = 0; i < 1000000; i++) {
        pthread_mutex_lock(&mutex);         /////
        somme += ordre;
        pthread_mutex_unlock(&mutex);       /////
    }
    return NULL;
}

int main(int argc, char** argv) {
    int i, nb;
    int * args;
    pthread_t * threads;
    nb = atoi(argv[1]);
    threads = malloc(nb * sizeof(pthread_t));
    args = malloc(nb * sizeof(int));
    pthread_mutex_init(&mutex, NULL);               /////
```

```
    for (i = 0; i < nb; i++) {
        args[i] = i;
        pthread_create(
           &threads[i], NULL, helloworld, &args[i]);
    }
    for (i = 0; i < nb; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("somme=%d\n", somme);
    return 0;
}
```

Fin réponse

▷ **Question 4:** Modifiez votre programme pour ne plus utiliser de variables globales. Il faut donc passer les adresses des différentes variables nécessaires en argument aux threads, dans une structure.

Réponse

solution exercice 3.4

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>

struct threadargs {
    int i;
    int * somme;
    pthread_mutex_t * mutex;
};

void * helloworld(void * arg) {
    struct threadargs * ta = (struct threadargs *) arg;
    int i;
    for (i = 0; i < 1000000; i++) {
        pthread_mutex_lock(ta->mutex);
        *ta->somme += ta->i;
        pthread_mutex_unlock(ta->mutex);
    }
    return NULL;
}

int main(int argc, char** argv) {
    int i, nb;
    struct threadargs * args;
    int somme = 0;
    pthread_mutex_t mutex;
    pthread_t * threads;
    nb = atoi(argv[1]);
    threads = malloc(nb * sizeof(pthread_t));
    args = malloc(nb * sizeof(struct threadargs));
    pthread_mutex_init(&mutex, NULL);
    for (i = 0; i < nb; i++) {
        args[i].i = i;
        args[i].somme = &somme;
        args[i].mutex = &mutex;
        pthread_create(
           &threads[i], NULL, helloworld, &args[i]);
    }
    for (i = 0; i < nb; i++) {
        pthread_join(threads[i], NULL);
    }
    printf("somme=%d\n", somme);
    return 0;
}
```

Fin réponse