EXERCICES DIRIGES – TRAVAUX PRATIQUES
Mécanismes de base sous Linux
Processus LEGERS

PROCESSUS LEGERS

Dans cet exercice, nous allons nous intéresser aux *processus légers* (ou *thread* en anglais). Contrairement à un processus lourd, les processus légers créés par le même processus parent partagent le même espace mémoire que celui-ci, même s'ils peuvent exécuter un segment de code différent.
Ce type de processus permet d'avoir une empreinte mémoire plus faible que l'utilisation de processus lourds (segment de données identique). De plus, comme les threads partagent le même espace mémoire il est plus facile de les faire communiquer.
En revanche, il est primordial de contrôler l'accès à la mémoire sans quoi certaines exécutions peuvent devenir incohérentes, à cause de l'accès concurrent par plusieurs processus à une même partie de la mémoire.

Comme vu en cours, la création d'un thread fils est réalisée en utilisant primitive `pthread_create()` suivante :

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*routine)(void *), void *arg);
```

Cette fonction retourne en premier paramètre le numéro du thread fils créé et prend respectivement en deuxième, troisième et dernier paramètre les attribus d'exécution particulier, l'adresse de la fonction à exécuter par le thread et un pointeur vers les paramètres de la fonction à exécuter.
Un code de retour de `0` est retourné en cas de succès lors de la création, sinon une erreur s'est produite.

D'autre part, la terminaison d'un thread est réalisée à l'aide la primitive `pthread_exit()` (au lieu de exit pour un processus lourd) :

```
int pthread_exit(void *value);
```

Cette fonction retourne un code de terminaison `value` au le thread principal.

Enfin, pour prendre en compte la terminaison d'un thread la primitive `pthread_join()` est à utiliser en remplacement de la primitive `wait()` :

```
int pthread_join(pthread_t thread, void **values);
```

Cette fonction prend deux paramètres, le premier indique le numéro du thread pour lequel on désire prendre en compte la terminaison et le second est un pointeur vers un tableau à deux dimensions permettant d'obtenir le code de retour des thread fils.

Elle retourne `0` en cas de succès, sinon une erreur s'est produite. Comme pour la primitive `wait()`, la primitive `pthread_join()` est bloquante.

Soit le programme `exo3.c` suivant :

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

int i;

int main(void)
{
     int pid;

     i = 0;
     pid = fork();
     if (pid == 0)
     {
           i = i + 10;
           printf ("hello,  fils %d\n", i);
           i = i + 20;
           printf ("hello, fils %d\n", i);
     }
     else
     {
           i = i + 1000;
           printf ("hello, père %d\n", i);
           i = i + 2000;
           printf ("hello, père %d\n", i);
           wait();
     }
     exit(0);
}
```

Question 1 – Tapez le programme `exo3.c` ci-dessus puis compilez et exécutez le programme exécutable obtenu. Quelles traces génère l'exécution de ce programme ?

Question 2 – Peut-on obtenir une trace différente lors d'une nouvelle exécution du programme ? Testez en relançant plusieurs fois ce programme.

Le programme `exo3.c` est modifié comme suit :

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

int i;
```

```
void addition()
{
     i = i + 10;
     printf ("hello, thread fils %d\n", i);
     i = i + 20;
     printf ("hello, thread fils %d\n", i);


     pthread_exit(0);

}

int main(void)
{
     pthread_t num_thread;

     i = 0;
     if(pthread_create(&num_thread, NULL, (void *(*)())addition, NULL) ==
     -1)
          perror ("pb pthread_create\n");
     i = i + 1000;
     printf ("hello, thread principal %d\n", i);
     i = i + 2000;
     printf ("hello, thread principal %d\n", i);
     pthread_join(num_thread, NULL);
     exit(0);
}
```

Question 3 – Modifiez le fichier `exo3.c` comme ci-dessus, puis compilez en utilisant l'option `-lpthread` (comme ci-dessous) et exécutez le programme exécutable `exo3bis` obtenu.

```
$> gcc -lpthread -o exo3 exo3.c
```

Quelles traces génère l'exécution de ce programme ?

Question 4 – Peut-on obtenir une trace différente lors d'une nouvelle exécution du programme ? Testez en relançant plusieurs fois ce programme.