

SYSTEMES D'EXPLOITATION

Série d'exercices N°2

Consignes aux étudiants :

- N'hésitez pas d'appeler votre professeur pour vous expliquer et interpréter le résultat.
- Un compte rendu est à rendre obligatoirement à la fin de chaque séance. Sa note fera partie de la note de l'évaluation pratique (TP).

Objectifs :

- Création, gestion et communication des processus.
 - Création, gestion et communication des threads.
-

I. Gestion des Processus

I.1 Communication entre Processus

Exercice 1

Écrire un programme où le père et le fils communiquent par l'intermédiaire d'un tube anonyme. Le père envoie au fils N nombres entiers. Le fils calcule la somme et l'envoie au père qui l'affiche à l'écran.

Exercice 2 :

Le but est de faire communiquer deux processus entre eux en utilisant les tubes anonymes. Écrire un programme dans lequel :

- Le processus père crée un processus fils et lui envoie un message.
- Le processus fils lit le message et ensuite répond au processus père.
- Le processus père attend la réponse de son fils avant de se terminer.

I.2 Synchronisation des processus.

Exercice 3 :

Ecrire un programme dont le fonctionnement est le suivant : il lit sur la ligne de commande un nombre N et crée N processus. il se met en attente (appel à *Pid_Fils* = *wait(&Etat)*) de ces N processus fils et visualise leur identité (*Pid_Fils* et valeur de *Etat*) au fur et à mesure de leurs terminaisons.

Pour attendre la fin de tous les fils, utiliser le fait que *wait* renvoie la valeur -1 quand il n'y a plus de processus fils à attendre.

Chacun des processus fils *Pi*:

- Visualise son pid (*getpid*) et celui de son père (*getppid*),
- Se met en attente pendant $2*i$ secondes (*sleep(2*i)*), visualise la fin de l'attente,
- Se termine par *exit(i)*.

II. Introduction à la gestion des threads

II.1 Rappel :

La fonction *pthread_create* crée un nouveau thread. Son prototype est le suivant :

```
int pthread_create ( pthread_t * thread,
                    pthread_attr_t * attributs,
                    void * (* fonction) (void * argument),
                    void * argument);
```

Voici les paramètres dont elle a besoin: Variable est thread

- Un pointeur vers une variable *pthread_t*, dans laquelle l'identifiant du nouveau thread sera stocké;
- Un pointeur vers un objet d'attribut de thread. Cet objet contrôle les détails de l'interaction du thread avec le reste du programme. Si vous passez *NULL* comme argument de thread, le thread est créé avec les attributs par défaut. Ceux-ci sont traités dans la Section 4.1.5, «Attributs de Threads»;

- Un pointeur vers la fonction de thread. Il s'agit d'un pointeur de fonction ordinaire de type: `void* (*) (void*)`;

- item Une valeur d'argument de thread de type `void*`. Quoi que vous passiez, l'argument est simplement transmis à la fonction de thread lorsque celui-ci commence à s'exécuter.

La fonction `pthread_create` renvoie zéro si elle réussit et une valeur non nulle sinon, correspondant à l'erreur survenue.

Un appel à `pthread_create` se termine immédiatement et le thread original continue à exécuter l'instruction suivant l'appel. Pendant ce temps, le nouveau thread débute l'exécution de la fonction de thread. Linux ordonnance les deux threads de manière asynchrone et votre programme ne doit pas faire d'hypothèse sur l'ordre d'exécution relatif des instructions dans les deux threads.

Exemple :

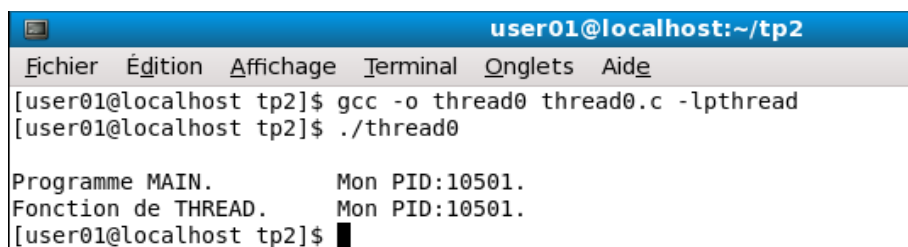
```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>

int main(){
    pthread_t  tid;

    /*Fonction qui va être exécutée par le thread*/
    void* fonction_thread(void* parm){
        printf("\nFonction de THREAD.\tMon PID:%d.",(int) getpid());
    }

    printf("\nProgramme MAIN.\t\tMon PID:%d.",(int) getpid());

    pthread_create(&tid, NULL, &fonction_thread, NULL);
    printf("\n");
    return 0;
}
```



```
user01@localhost:~/tp2
Fichier  Édition  Affichage  Terminal  Onglets  Aide
[user01@localhost tp2]$ gcc -o thread0 thread0.c -lpthread
[user01@localhost tp2]$ ./thread0

Programme MAIN.      Mon PID:10501.
Fonction de THREAD.  Mon PID:10501.
[user01@localhost tp2]$ █
```

Question : Compilez et exécutez ce programme en utilisant la commande suivante :

```
$ cc -o thread-create thread-create.c -lpthread
```

II.2 Thread et processus :

Exercice 4 : Donner les deux programmes suivants en respectant les étapes suivantes :

Dans le programme processus1.c	Dans le programme thread1.c
<ol style="list-style-type: none">1. Déclarer une variable avant la création des fils,2. Initialiser cette variable par une valeur,3. Faire modifier cette valeur par un processus fils,4. Afficher le contenu de cette variable par le processus Père,	<ol style="list-style-type: none">1. Déclarer une variable avant la création des threads,2. Initialiser cette variable par une valeur,3. Faire modifier cette valeur par un thread,4. Afficher le contenu de cette variable par un autre thread,

Interpréter les résultats obtenus.

Exercice 5 :

Écrire un programme qui crée un thread qui dort un nombre de secondes, passé en argument, pendant que le thread principal attend qu'il se termine.

Exercice 6 :

Écrire un programme qui crée un thread qui prend en paramètre un tableau d'entiers (initialisé par le thread principal) et l'affiche dans la console.

Exercice 7 :

Écrire un programme qui crée un thread qui alloue un tableau d'entiers, initialise les éléments par des entiers aléatoires entre 0 et 99. Le thread principal affichera le tableau d'entiers.