1 ère année Master : RSD **Année : 2021/2022**

Cours : Programmation réseaux et systèmes

Chapitre : Les threads.

Introduction

Le modèle de processus décrit au chapitre précédent est un programme qui s'exécute selon un chemin unique avec un seul compteur ordinal. On dit qu'il a un flot de contrôle unique ou un seul thread. De nombreux systèmes d'exploitation modernes offrent la possibilité d'associer à un même processus plusieurs chemins d'exécution ou multithread. Ils permettent ainsi l'exécution simultanée des parties d'un même processus.

Chaque partie correspond à un chemin d'exécution du processus.

Introduction

Le processus est vu comme étant un ensemble de ressources (code exécutable, segments de données, de fichiers, de périphériques, etc.) que ces parties appelées flots de contrôle ou processus légers (threads en anglais) partagent.

Chaque flot de contrôle (thread) a cependant, en plus des ressources communes, sa propre zone de données ou de variables locales, sa propre pile d'exécution, ses propres registres et son propre compteur ordinal.

Qu’est-ce qu’un thread ?

Un thread (ou fil d’exécution en français) est une parie du code d’un programme (une fonction), qui se déroule parallèlement à d’autre parties du programme. Un premier intérêt peut être d’effectuer un calcul qui dure un peu de temps (plusieurs secondes, minutes, ou heures) sans que l’interface soit bloquée (le programme continue à répondre aux signaux)

Qu’est-ce qu’un thread ?

L’utilisateur peut alors intervenir et interrompre le calcul sans taper un ctrl-C brutal. Un autre intérêt est d’effectuer un calcul parallèle sur les machines multi-processeur.Les fonctions liées aux thread sont dans la bibliothèque pthread.h, et il faut compiler avec la librairie lib pthread.a :

$ gcc -lpthread monprog.c -o monprog

Avantages

Comparativement aux processus à un flot de contrôle unique, un thread ou processus léger avec plusieurs flots de contrôle présente plusieurs avantages, notamment :

Réactivité. Le processus léger continue à s'exécuter même si certaines de ses parties sont bloquées.

Partage de ressources.

Économie d'espace mémoire et de temps.

Threads utilisateur et noyau

La majorité des systèmes permettent le multiflot (multithreading).

Ils sont offerts soit au niveau utilisateur , soit au niveau noyau.

Les threads utilisateur sont supportés au-dessus du noyau et sont implantés par une bibliothèque de threads au niveau utilisateur . Ils sont portables sur différentes plateformes.

Ils sont gérés par une application où le blocage du thread peut bloquer le processus complet. Le changement de contexte est rapide.

.

Threads utilisateur et noyau

Les threads noyau sont directement supportés par le noyau du système d'exploitation. Le système d'exploitation se charge de leur gestion et le changement de contexte est lent.

les threads combinés sont implantés par le système d'exploitation (utilisateur et système). Les threads utilisateur sont associés à des threads. système. La plupart des tâches gestion s'effectuent sous le mode utilisateur

Les opérations concernant les threads sont notamment la création, la terminaison, la suspension et la relance.

Création d'un thread et attente de terminaison

Pour créer un thread, il faut créer une fonction qui va s'exécuter dans le thread, qui a pour prototype :

```
void *ma_fonction_thread(void *arg);
```

Dans cette fonction, on met le code qui doit être exécuté dans le thread. On crée ensuite le thread par un appel à la fonction pthread_create, et on lui passe en argument la fonction ma_fonction_thread dans un pointeurs de fonction (et son argument arg). La fonction pthread_create a pour prototype :

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

Création d’un thread et attente de terminaison

Le premier argument est un passage par adresse de l’identifiant du thread (de type pthread_t). La fonction pthread_create nous retourne ainsi l’identifiant du thread, qui l’on utilise ensuite pour désigner le thread. Le deuxième argument attributs désigne les attributs du thread, et on peut mettre NULL pour avoir les attributs par défaut. Le troisième argument est un pointeur sur la fonction à exécuter dans le thread (par exemple ma_fonction_thread, et le quatrième argument est l’argument de la fonction de thread.

Création d’un thread et attente de terminaison

Le processus qui exécute le main (l’équivalent du processus père) est aussi un thread et s’appelle le thread principal. Le thread principal peut attendre l’exécution d’un autre thread par la fonction pthread_join (similaire à la fonction wait dans le fork.

Cette fonction permet aussi de récupérer la valeur retournée par la fonction ma_fonction_thread du thread.

Le prototype de la fonction pthread_join est le suivant :

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

Création d’un thread et attente de terminaison

Le premier paramètre est l’dentifiant du thread (que l’on obtient dans pthread_create), et le second paramètre est un passage par adresse d'un pointeur qui permet de récupérer la valeur retournée par ma_fonction_thread.

Terminaison de threads

void pthread_exit(void *valeur_de_retour);

pthread_exit() permet à un processus léger de terminer son exécution, en retournant l'état de terminaison.

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
void * fonction_thread(void *arg)
{
printf("Nous sommes dans le thread.\n");
pthread_exit(NULL);
}
int main(void)
{
pthread_t thread1;
printf("Avant la création du thread.\n");
if(pthread_create(&thread1, NULL, fonction_thread, NULL) == -1)
 {  perror("pthread_create");
return EXIT_FAILURE;
}

printf("Après la création du thread ");
exit(0);
}
```