Série 2 : Les processus légers (threads POSIX)

Rappel 1: Les processus légers (les threads)

Une application quelconque peut être constituée de plusieurs « fils d'exécution » ou **processus légers** ou **threads**: le fil d'exécution principal (qu'on va appeler processus principal) de l'application est celui qui exécute la fonction main. Plusieurs threads secondaires peuvent être crées par le processus principal dans le but de réaliser certaines opérations ou taches de fond en parallèle avec la tache principale de l'application (exp. Un thread pour écouter le port de l'application et gérer les communications, un thread pour gerer l'affichage etc.).

- Les t**hreads** se distinguent des processus fork() par le fait qu'ils partagent le même espace mémoire avec le processus père. En effet, les données du processus ne sont pas dupliquées a la création d'un thread, seuls les informations et traitements spécifiques au nouveau thread le sont (d'ou le nom processus légers)

- Autre différence par rapport au processus fork(), les threads d'un même processus possèdent tous le même pid que le processus créateur, mais se distinguent par des **tid** différents (tid=identificateur de thread)

- La norme POSIX définie le standard pour les threads sous linux. La librairie **Pthread** est une implémentation des threads POSIX. Le tableau suivant résume les différentes fonctions pthreads pour la création et manipulation de threads POSIX.

Fichier entête a inclure	**#include <pthread.h>**
Compilation	**gcc prog.c -lpthread -o executable**
Commandes linux pour lister les threads d'un processus	**$ ps -T -p <pid>** **$ top -H -p <pid>**
Création d'un thread (si attr==NULL, création d'un thread avec les attributs par defaut joignable et politique d'ordonnancement)	int pthread_create(pthread_t * tid, pthread_attr_t * attr, void * (*start_routine)(void *), void * arg);
Recuperer le tid d'un thread	pthread_t pthread_self (void)
Attendre la fin d'un autre thread	int pthread_join(pthread_t tid, void **thread_return);
Détacher un processus le mettre dans l'état non joignable (empêche les appels pthread_join et ses ressources sont libérées a sa terminaison).	int pthread_detach(pthread_t tid);
Terminaison d'un thread	void pthread_exit(void *retval);
Annulation d'un thread par un autre thread.	int pthread cancel(pthread t thread);

Exercice 1 : Analyser le comportement du programme suivant

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/wait.h>

#include <sys/types.h>

void *thread_function(void *var) { //le code a executer par un thread

printf("Je suis le thread de tid :%p, mon pid est :%d\n", pthread_self ()  ,getpid()) ;

sleep(6);

printf("Au revoir du thread de tid :%p\n", pthread_self ()) ;

pthread_exit((void*)42);

}

int main(){

 int i;

pthread_t tid;

pthread_create(&tid, NULL, thread_function, NULL);

printf("Thread principal de (pid, tid) :(%d, %p), creation d'un thread de tid  %p \n",getpid(), pthread_self(),tid);

//pthread_join(tid, (void **)&i);

//printf("Thread principal, la valeur de retour du thread  %p est: %d\n",tid,i);

return 0; }
```

- Compiler le programme avec la ligne de commande :

gcc nomprog.c -o executable -lpthread

- Pourquoi ne vois t'ont pas s'afficher le message Au revoir du thread crée ?
- De-commenter les 2 dernières instructions de la fonction main et recompiler. Vérifier les affichages. Quel est le rôle de la fonction thread_join ?
- Que ce passera t il si on remplace l'appel a la fonction pthread_exit par un appel exit au niveau de la fonction thread_function ?
- Modifier la fonction thread_function pour calculer le carre d'un entier transmis a la fonction lors de la création du thread. La fonction doit alors retourner le résultat du calcul.

- Modifier le programme main pour créer n threads dans le but de calculer le carre d'un tableau de n valeurs entières. Le thread i va afficher « je suis le thread numéro i , mon tid i » , puis va calculer le carre de la valeur t[i] du tableau t et retourner le résultat. Le processus main attend la fin de tout les threads pour afficher le tableau resultat.

- Calculer le temps de calcul avec les n threads et comparer avec le temps séquentiel si le processus main avait exécute la fonction thread_function dans une boucle au lieu de créer des threads. Vous pouvez utiliser la commande time ./prog.

- Question sur l'ordonnancement des threads d'une application sur un CPU multi-coeurs : est ce qu'on gagne toujours du temps de calcul en créant un thread pour chaque tache d'une application ? Interet de l'utilisation de plusieurs threads dans une application ? Est ce que les threads d'une application sont ordonnances directement par le systeme ou alors ils s'executent dans le quantum de temps du processus principal ?

Exercice 2 :

Étant donné un tres grand nombre, n , on souhaite verifier si ce nombre est premier ou pas, on va alors essayer de trouver un diviseur par la méthode naïve consistant à calculer les restes n mod i pour i allant de 2 à $(n)^{1/2}$. Cette methode engendre beaucoup de calculs pour les tres grands nombre, on va alors tenter de la paralleliser a travers le lancement de plusieurs threads.

Cette stratégie est résumée dans le code suivant :

```
#include<stdio.h>
#include<unistd.h>
#include<math.h>
#define nb 18446743979220271

int main  (void)
{
 register unsigned long long sqrt_nb ;
 register unsigned long long i ;

 sqrt_nb = sqrt(nb) ;

 for(i=2;i<sqrt_nb; i++)
  if (!(nb % i))
  {
   printf(" %llu est un diviseur de %llu \n",i,nb) ;
   return 0 ;
  }
 return 1 ;
}
```

Puisque chaque opération modulaire est indépendante, on se propose de créer autant de processus légers p que d'unités de calcul sur votre machine afin de paralléliser ce travail. Donnez la définition d'un code C qui va :

- construire p threads j=0,…,p 1 ;
- le thread j va calculer les restes n mod i pour i allant de 2+j $(n)^{1/2}/p$ à 2+(j+1)$(n)^{1/2}/p$;
- quand un thread trouve un diviseur, il affiche son résultat et termine l'exécution de tous les threads auxilliaires.

Le processus principal affiche le resultat final : x est un nombre premier ou x n'est pas un nombre premier.

Mesurer le temps d'execution de la version multi-threadee et le comparer au programme main donne ci dessus.

Exercice 3:

Compiler et executer le code suivant:

```
#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <math.h>

void * fonction(void * var) {

printf("Je suis le thread de tid: %p \n",pthread_self());

sleep(5);

printf("Au revoir");

return NULL; }

int main(){

pthread_attr_t attr;

pthread_attr_init (&attr); //initialisation par les valeurs par défaut //afficher les valeurs par défaut des attributs

//l'attribut scope d'ordonnancement

int scope;

pthread_attr_getscope (&attr, &scope);

printf("Attribut scope par défaut est:%s",(scope==PTHREAD_SCOPE_SYSTEM? "Scope Systeme": (scope==PTHREAD_SCOPE_PROCESS ? "Scope processus":"OTHER")));
```

//1: modifier la valeur de scope pour le rendre scope processus

```
//priorite par défaut du processus

int policy; struct sched_param sched;

pthread_getschedparam (&attr, &policy, &sched);
```

```
 printf ("politique : %s, priorite  :%d\n", (policy == SCHED_OTHER ? "SCHED_OTHER" : (policy == SCHED_FIFO ? "SCHED_FIFO" : (policy == SCHED_RR ? "SCHED_RR" : "inconnu"))), sched.sched_priority);
```

//2: modifier la priorité du thread a créer

```
//afficher la valeur par défaut de de l'attribut detachstate

int detache;

pthread_attr_getdetachstate (&attr, &detache);

if(detache==PTHREAD_CREATE_DETACHED) printf("Detache thread attribut par défaut est Detache/n");

else if(detache==PTHREAD_CREATE_JOINABLE )

printf("Detache thread attribut par défaut est Joignable\n");
```

//3: modifier le detach state pour créer un thread injoignable ou detache avec la fonction //pthread_attr_setdetachstate.

```
//creer un nouveau thread avec les attributs crees et modifies en haut

pthread_t tid;

pthread_create(&tid,&attr,fonction,NULL);

pthread_attr_destroy (&attr);

printf("Thread principal, creation d'un thread de tid  %p \n",tid);

return 0;

}
```

Questions:

1-Quelle est la difference entre SCOPE SYSTEM et SCOPE PROCESSUS ? Modifier la valeure de l'attribut scope

2 - Modifier la priorité du thread a créer

3- Modifier la valeur de l'attribut detachstate pour creer un thread injoignable

4- Ajouter un appel a la fonction pthread_join pour attendre la fin du thread tid (qui est maintenant injoignable) afficher la valeure de retour de la fonction. Quelle est l'utilite de creer des threads detaches? Et joignables?