UFR IM²AG

UNIVERSITÉ Grenoble Alpes

Master M1 Informatique, CSE
TD & TP Threads
Année 2017-2018

Résumé

Cette fiche ne sera traitée que cette semaine. Vous traiterez en dehors de la séance les exercices et questions non résolues. Les objectifs de cette séance de TD/TP sont les suivants :
— Appréhender l'interface de programmation POSIX Threads
— Ecrire les premiers algorithmes et programmes multithreadés
— Partager des données entre threads dans l'espace mémoire du processus

Les exercices portant la mention "Travail personnel" figurant en fin defeuille de TD sont optionnels : le temps imparti à une séance de TD ne permettra sans doute pas de les traiter durant celle-ci et les évaluations de l'UE ne porteront pas sur les notions qu'ils permettent d'aborder. Il sont donnés à titre de travail personnel. Si vous ne parvenez pas à le traiter seul n'hésitez pas à demander un peu d'aide à vos enseignants.

1 Un premier programme pour se mettre en condition...

Dans cette partie, nous allons étudier le programme `match.c` fourni en annexe de ce texte. Ce programme simule le comportement positif des supporters pendant un match de rugby. Le comportement d'un supporter est modélisé par un thread. Il y a deux paramètres à ce programe : le nombre de supporters des équipes 1 et 2. Après compilation de ce programme, l'exécution donne le résultat suivant :

```
$ gcc -o match match.c -lpthread
$ match
$ match 3 2
Processus 12303 Thread b7dd0b90 : Allons enfants de la patrie...
Processus 12303 Thread b75cfb90 : Allons enfants de la patrie...
Processus 12303 Thread b6dceb90 : Allons enfants de la patrie...
Processus 12303 Thread b65cdb90 : Sweep low, sweet chariot
Processus 12303 Thread b5dccb90 : Sweep low, sweet chariot
Processus 12303 Thread b5dccb90 : Sweep low, sweet chariot
Processus 12303 Thread b6dceb90 : Allons enfants de la patrie...
Processus 12303 Thread b7dd0b90 : Allons enfants de la patrie...
Processus 12303 Thread b65cdb90 : Sweep low, sweet chariot
Processus 12303 Thread b75cfb90 : Allons enfants de la patrie...
Processus 12303 Thread b65cdb90 : Sweep low, sweet chariot
Processus 12303 Thread b7dd0b90 : Allons enfants de la patrie...
Processus 12303 Thread b5dccb90 : Sweep low, sweet chariot
Processus 12303 Thread b6dceb90 : Allons enfants de la patrie...
Processus 12303 Thread b75cfb90 : Allons enfants de la patrie...
$
```

Les quelques questions qui suivent ont pour but de vous aider dans la lecture de ce premier programme et dans la compréhension de son exécution.

Question .1. *A quoi sert la variable* `tids` *de la fonction* `main` *? Comment est alloué l'espace mémoire pour cette variable ? Comment cette variable est-elle initialisée ? Comment et quand l'espace mémoire alloué à cette variable est-il libéré ?*

Question .2. *Expliquez comment sont créés les threads dans ce programme (détaillez bien le fonctionnement de la fonction* `pthread_create`*).*

Question .3. *Que se passe-t-il avec la fonction* `usleep` *? Quels sont les changements d'état du thread quand cette fonction est appelée ? Analysez l'ordre et les messages affichés par les threads supporter ?*

Question .4. *Expliquez comment se termine le programme* `match`*. Que font les threads* `supporter` *lorsqu'ils se terminent ? Que fait la fonction* `main` *? Que se passerait-il si la dernière boucle de la fonction* `main` *(celle avec les appels à* `pthread_join`*) avait été oubliée par le programmeur ?*

Question .5. *Représentez sur un schéma le processus (avec son pid)), les threads (avec leurs tid)...*

2 Passage de plusieurs paramètres

Comme nous venons de le voir sur les deux exemples précédents, la fonction `pthread_create` ne permet le passage que d'un seul paramètre à la fonction exécutée par le thread. Sur l'exemple précédent, il s'agissait juste d'une chaîne de caractères correspondant au chant du supporter.

Nous souhaiterions maintenant passer au thread supporter le nombre de fois où il va interpréter ce chant. On supposera, par exemple, qu'un supporter Anglais est beaucoup plus actif et chante plus qu'un supporter Français...

Pour passer plusieurs paramètres à une fonction, il est nécéssaire de définir une structure avec comme champs les différents paramètres. Vous passerez l'adresse de cette structure à la fonction `pthread_create`.

Voici ce que pourrait donner l'exécution d'un match avec 4 supporters Français et 2 supporters Anglais. Les supporters français chantent 2 fois et les anglais 5.

```
$ gcc -o matchp matchp.c -lpthread
$ matchp
$ match 4 2 2 5
Processus 32399 Thread b7e17b90 : Allons enfants de la patrie...
Processus 32399 Thread b7616b90 : Allons enfants de la patrie...
Processus 32399 Thread b6e15b90 : Allons enfants de la patrie...
Processus 32399 Thread b6614b90 : Allons enfants de la patrie...
Processus 32399 Thread b5e13b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5612b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b6e15b90 : Allons enfants de la patrie...
Processus 32399 Thread b5612b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b7e17b90 : Allons enfants de la patrie...
Processus 32399 Thread b7616b90 : Allons enfants de la patrie...
Processus 32399 Thread b5e13b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5612b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b6614b90 : Allons enfants de la patrie...
Processus 32399 Thread b5612b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5612b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5e13b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5e13b90 : Sweep Low, sweet chariot...
Processus 32399 Thread b5e13b90 : Sweep Low, sweet chariot...
$
```

3 Recherche d'un élément dans un vecteur non trié

L'objectif de cet exercice est de concevoir un algorithme concurrent implémenté par un programme mulithreadé pour rechercher un élément dans un vecteur non trié. On supposera que les éléments sont de type entier, mais le principe doit s'appliquer à tous types de données.

Question .6. *Ecrire la version séquentielle du programme :*

- *une première fonction initialise le vecteur avec des éléments dont les valeurs sont lues à partir d'un fichier sur l'entrée standard ;*
- *la fonction* `search` *recherche la valeur x dans le vecteur T. Le vecteur T a une taille n. Cette fonction parcourt* séquentiellement *le vecteur depuis le premier élément* 0 *jusqu'au dernier ;*
- *le programme affiche ensuite si la valeur a été trouvée dans le vecteur.*

Question .7. *Ecrire la version multithreadée du programme et en particulier de la fonction* `search` *: chaque thread recherchera la valeur dans une portion du vecteur qui lui aura été affectée. Un thread doit pouvoir arrêter sa recherche si un autre thread a trouvé la valeur x recherchée dans le vecteur T.*

Question .8. *Lorsque la valeur n'est pas présente dans le vecteur, si votre machine dispose de deux cœurs et que la fonction* `search` *est exécutuée par deux threads, quel serait le facteur d'accéleration du programme (rapport entre le temps d'exécution sur un et sur deux cœurs) ?*

Question .9. *Cela change-t-il quelque chose si la valeur est présente dans le vecteur ?*

4 Travail personnel : produit scalaire de deux vecteurs

L'objectif de cet exercice est d'écrire un programme multithreadé qui calcule le produit scalaire de deux vecteurs. Les deux vecteurs sont de taille n. Les valeurs de ces vecteurs sont lues sur l'entrée standard. Le produit scalaire s'exprime de la manière suivante :

$$v1 * v2 = \sum_{i=0}^{n-1} v1[i] * v2[i]$$

Question .10. *Ecrire la version séquentielle du programme qui calcule le produit scalaire de deux vecteurs.*

Question .11. *Ecrire la version multithreadée du programme qui calcule le produit scalaire de deux vecteurs : Le principe est un peu le même que pour la recherche dans un vecteur. La différence, c'est que chaque thread va cumuler dans une variable locale le produit scalaire de la portion sur laquelle il travaillait. La valeur sera retournée au programme principal par la fonction* `pthread_exit` *au niveau du thread de calcul et par la fonction* `pthread_join` *pour le thread principal.*

Annexe A

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#define NB_SONG 3

void *supporter (void *arg){
  char      *str = (char *) arg ;
  int       i ;
  int       pid ;
  pthread_t tid ;
  pid = getpid () ;
  tid = pthread_self () ;
  srand ((int) tid) ;

  for (i = 0; i < NB_SONG; i++){
    printf ("Processus %d Thread %x : %s \n", pid, (unsigned int) tid, str) ;
    usleep (rand() / RAND_MAX * 1000000.) ;
  }
  return (void *) tid ;
}

int main (int argc, char **argv){

  int team1 ;
  int team2 ;
  int i ;
  int nb_threads = 0 ;
  pthread_t *tids ;

  if (argc != 3){
    fprintf(stderr, "usage : %s team1 team2\n", argv[0]) ;
    exit (-1) ;
  }

  team1 = atoi (argv[1]) ;
  team2  = atoi (argv[2]) ;
  nb_threads = team1 + team2;
  tids = malloc (nb_threads*sizeof(pthread_t)) ;

  /* Create the threads for team1 */
  for (i = 0 ; i < team1; i++){
    pthread_create (&tids[i], NULL, supporter, "Allons enfants de la patrie") ;
  }
  /* Create the other threads (ie. team2) */
  for ( ; i < nb_threads; i++){
    pthread_create (&tids[i], NULL, supporter, "Swing low, sweet chariot") ;
  }

  /* Wait until every thread ened */
  for (i = 0; i < nb_threads; i++){
    pthread_join (tids[i], NULL) ;
  }

  free (tids) ;
  return EXIT_SUCCESS;
}
```

Annexe B : POSIX

POSIX veut dire "Portable Operating System Interface". Comme son nom l'indique, c'est un standard qui aide la portabilité des programmes (code source) entre différentes systèmes d'exploitation. En d'autres termes, un programme qui est écrit en suivant les interfaces POSIX pourra être compilé et exécuté sur tous les systèmes qui supportent le standard.

Pour avoir de nombreuses informations sur le fonctionnement et l'interface POSIX pour la manipulation de threads, vous pouvez faire `man pthreads`. Dans la liste des fonctions de manipulation de threads. dans ce TD-TP nous utilisons :

Extrait des fonctions `pthread`

```
#include <pthread.h>

/** The pthread_create() function starts a new thread in the calling process.
 *  The new thread starts execution by invoking start_routine();
 *  arg is passed as the sole argument of start_routine().
 */
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);

/** The  pthread_join()  function  waits  for  the  thread  specified  by
 *  thread  to terminate. If that thread has already terminated,
 *  then pthread_join()  returns immediately. If retval is not NULL,
 *  then pthread_join() copies the exit status of the target thread
 *  (i.e., the value that the target thread supplied to
 *  pthread_exit(3))  into  the  location pointed to by *retval.
 */
int pthread_join(pthread_t thread, void **retval);

/** The  pthread_exit() function terminates the calling thread and returns
 *  a value via retval  that (if the thread is joinable) is available to
 *  another thread in the same process that calls pthread_join(3).
 */
void pthread_exit(void *retval);
```

Annexe C : Richard Stallman

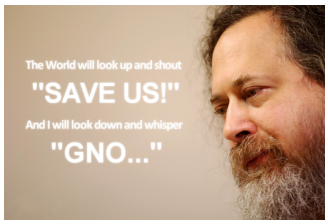

FIGURE 1 – R.Stallman. (www.unixmen.com/richard-matthew-stallman-father-free-software-foundation)

Nous vous laissons découvrir qui est R.Stallman et nous contenterons à vous dire qu'il a travaillé sur la définition de POSIX. Vous pouvez d'ailleurs aller lire son explication du nom POSIX à `https://stallman.org/articles/posix.html`