Master M1 MOSIG, UJF Grenoble
Multithreading with Posix Threads

Arnaud Legrand, Sascha Hunold, Benjamin Negrevergne

2010

Résumé

During this lab we will :
– Learn about POSIX PThread primitives,
– write multithreaded algorithms and programs
– practice data sharing in process's memory space
This subject is to be treated during this lab only. You are expected to finish the exercices before the next lab.

I. A first program...

This program simulates the behavior of rugby supporters attending a match. Each supporter is modeled by a thread. The program inputs two parameters : the number of supporters for team 1 and team 2.

Wi will study now the provided program `match.c`. After compiling, the program execution gives the following result.

```
$ 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...
$
```

Answer the following questions in order to understand the program's behavior.

Question I.1. *What is the goal of the* `tids` *variable in the* `main` *function ? How is the space for this variable allocated ? How is the variable initialized ? How and when is the memory space for this variable freed ?*

Question I.2. *Explain how the threads are created. Detail the* `pthread_create` *function.*

Question I.3. *What happens when the* `usleep` *function is called ? What are the state changes of a thread ? Observer and analyze the order of the mesages printed by the supporter threads.*

Question I.4. *Explain how the program* `match` *terminates. What do the threads do at the end of their execution ? What does the* `main` *function ? What would have happened if the developer has forgotten the last loop in the* `main` *function (the one with* `pthread_join` *calls) ?*

Question I.5. *Draw a picture of the memory space of the process (threads, variables...).*

II. Rugby championship semi-finals

We want to extend the `match` program so as to model two simultaneous matchs. Each match should be modeled by a distinct process (i.e two matchs : two processes). The matchs take place in parallel.

Question II.1. *Change the program so as to have two matchs in two parallel processes. The program must input at least* 4 *parameters.*

Question II.2. *Draw the processes' memory space.*

III. Parameter passing

The `pthread_create` function takes a single `void *` pointer argument. In the previous example, the argument were the lyrics sung by the supporters.

We would also like to pass as a parameter the number of times a supporter is going repeat the song. We presume that an English supporter is eagerer than a French one...

How can you pass more than one argument to the function executed by the threads ?

Here is an example of an execution with 4 French supporters and 2 English supporters. The French sing 2 times while English sing 5 times.

```
$ 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...
$
```

IV. Search an element in a non sorted vector

The goal is to design and implement a multirhreaded algorithm to search an element in a non sorted vector. We assume here that the elements are integers, but the algorithm principle is the same whatever the data type.

Question IV.1. *Implement a sequential (ie. non-multithreaded) algorithm.*
- *Create and initialize a vector with random values (cf.* `man 3 rand`*).*
- *Write a* `search(T, n, x)` *function that searches for a value x in the vector T. T having n elements. The function treats all elements* sequentially*, from the first to the last. The program prints the index of x in T if x was found, -1 if it was not.*
- *Validate and time your imlpementation with* `time`*.*

Question IV.2. *Write a multithreaded version of the program and of the* `search` *function in particular.*

Each thread will search for the value in a fraction of the vector. If one thread finds the value, what the other threads should do ?

Question IV.3. *If the value is not in the vector, if you have a dual-core machine and if the* `search` *function is executed by two threads, what could be the speed-up of the algorithm ? (how much faster is the algorithm on two cores compared to execution on one core) ?*

Question IV.4. – *In theory how much can you speed up the program using n cores ? Time you program and calculates your $speedup$ with the following formula :*

$$speedup = \frac{time_{sequential}}{time_{parallel}}$$

- *For what size do you observe the best $speedup$.*
- *There is a significant difference between the thoritical speedup and the pratical speed up. Why ?*

V. Scalar product of two vectors

The goal in this exercise is to write a multithreaded program to calculate the scalar product of two vectors. The size of the vectors is n. The values of the vectors' elements are read on the standard input. The formula for computation is the following

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

Question V.1. *Write a sequential version of the program.*

Question V.2. *Write multithtreaded program. The principle is close to the previous one, but each thread has to store in a local variable the product of the fraction of the vector which is in charge of. The main function will gather and compute the final results after the calls to* `pthread_join` *functions.*

1 The POSIX Threads Library (PThread)

POSIX (Portable Operating System Interface) defines a standard thread interface.The primitives can me consulted using man (`man pthread`). Here are some basic primitives for thread manipulation.

- **Thread creation**

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

  - `pthread_t` : thread type
  - `pthread_t *thread` : after a successful creation, the first argument contains the thread identificator
  - `const pthread_attr_t *attr` : We will ignore these attributes that may be used to configure the scheduling strategy and the thread priorities.
  - `void*(*start_routine)(void *)` : the third argument gives the function the thread should execute.
  - `void *arg` : the fourth argument is the argument to pass to the function to be executed by the thread.

- **Thread termination**

```
void pthread_exit (void *status);
```

  Terminates the thread and gives a return value in `status`.

- **Wait for a thread to terminate**

```
int pthread_join(pthread_t th, void ** status);
```

  Wait of the termination of the thread `th` and store teh return value in dans `status`. The thread should not be detached (see `pthread_detach`).

- **free the CPU**

```
int pthread_yield(void)
```

- **thread identification**

```
pthread_t pthread_self (void);
```

1.1 Example

```
#include <pthread.h>
#include <stdio.h>
void *routine (void *arg)
{
    int *status = malloc (sizeof(int)); /* pour renvoyer le code de retour */
    printf ("Arg = %d\n", *(int *)arg); /* casting vers (int *) necessaire */
    *status = *(int *)arg * 2;
    pthread_exit (status);
}
```

```
int main()
{
    pthread_t un_p;
    int erreur, argument = 3;
    int *resultat;
    erreur = pthread_create (&un_p, NULL, routine, &argument);
    if (erreur != 0) fprintf(stderr,"Echec creation de thread: %d\n",erreur);
    pthread_join (un_p, (void **)&resultat);
    printf ("Resultat: %d\n", *resultat);
    free (resultat);
    exit(0);
}
```