Multithreading with Posix Threads

Master M1 MOSIG, Grenoble University

2014

Abstract

During this lab we will:

- learn about POSIX PThread primitives (see this document's appendix for some technical details),
- write multithreaded algorithms and programs
- practice data sharing in process' memory space
- **Your report should reflect your work on Questions I to IV included.**

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.

We 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 : Swing low, sweet chariot
Processus 12303 Thread b5dccb90 : Swing low, sweet chariot
Processus 12303 Thread b5dccb90 : Swing 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 : Swing low, sweet chariot
Processus 12303 Thread b75cfb90 : Allons enfants de la patrie
Processus 12303 Thread b65cdb90 : Swing low, sweet chariot
Processus 12303 Thread b7dd0b90 : Allons enfants de la patrie
Processus 12303 Thread b5dccb90 : Swing 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 do happens when the* `usleep` *function is called? What about the state changes of a thread? What can you say about the order of the messages 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 about the* `main` *function? What happens if the developer forgets the last loop in the* `main` *function (the one with* `pthread_join` *calls) ?*

II. Parameter Passing

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

The goal here is to write a new version of the program which takes as an input the lyrics of the song and the number of times a supporter is going to repeat the song (we presume that an English supporter is eagerer than a French one...).

To do so, you need to pass two parameters to the threads but the `pthread_create` specification allows you to pass only one. In order to overcame this limitation, you need to create a structure containing two fields and pass a pointer to the structure.

Here is an example of an execution with 4 French supporters and 2 English supporters. The French sing 2 times while the 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 : Swing low, sweet chariot
Processus 32399 Thread b5612b90 : Swing low, sweet chariot
Processus 32399 Thread b6e15b90 : Allons enfants de la patrie
Processus 32399 Thread b5612b90 : Swing 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 : Swing low, sweet chariot
Processus 32399 Thread b5612b90 : Swing low, sweet chariot
Processus 32399 Thread b6614b90 : Allons enfants de la patrie
Processus 32399 Thread b5612b90 : Swing low, sweet chariot
Processus 32399 Thread b5612b90 : Swing low, sweet chariot
Processus 32399 Thread b5e13b90 : Swing low, sweet chariot
Processus 32399 Thread b5e13b90 : Swing low, sweet chariot
Processus 32399 Thread b5e13b90 : Swing low, sweet chariot
$
```

III. Return Value

The `pthread_join` function allows you to get the return value of the function executed by a thread. To understand this functionality, the goal here is to write a multi-threaded program calculating the sum of the integer elements of an array of size n.

Question III.1: *Write a sequential version of the program.*

Question III.2: *Write a multi-threaded version. 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 array he is in* *charge of. The main function will gather and compute the final results after the calls to* `pthread_join` *functions.*

Question III.3: *Compare the execution times of the sequential and the parallel version. You can try with 2, 4, 6, 8... threads. What can you say about the results?*

IV. Variable Sharing between Threads

Write a new version of the previous program in which all threads write their return value to a global buffer, the buffer being managed with the help of an index showing the next free slot. The return values should be written to the array in the order of their computation (i.e the first thread to finish writes in element 0, the second thread to finish writes in element 1, etc.). What do you observe in the buffer at the end of the program?

V. Search an element in a non-sorted vector

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

Question V.1: *Implement a sequential (i.e. non-multi-threaded) 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 implementation with* `time`*.*

Question V.2: *Write a multi-threaded 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 should the other threads do ?

Question V.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 the execution on one core)?*

Question V.4:

- *In theory how much can you speed up the program using n cores ? Time your program and calculate 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 theoretical speed-up and the practical speed-up. Why ?*

VI. Scalar product of two vectors

The goal in this exercise is to write a multi-threaded 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 VI.1: *Write a sequential version of the program.*

Question VI.2: *Write a multi-threaded version. 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 vectors he 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 identifier.
  - `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 the return value in `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)); /* To receive the return status */
    printf ("Arg = %d\n", *(int *)arg); /* Necessary cast to (int *) */
    *status = *(int *)arg * 2;
    pthread_exit (status);
```

```
}

int main()
{
    pthread_t pth;
    int err, arg = 3;
    int *res;
    err = pthread_create (&pth, NULL, routine, &arg);
    if (err != 0) fprintf(stderr,"Failed to create a thread: %d\n",err);
    pthread_join (pth, (void **)&res);
    printf ("Resultat: %d\n", *res);
    free (res);
    exit(0);
}
```