Lab Worksheet 04 - Threads

Objectives

1. Make your first multi-threaded programs.
2. Implement basic synchronization among threads.

Remark

- When asked to synchronize threads, please refrain from using tools that are outside the pthread library; for example unnamed semaphores.
- Remember that compiling code that uses pthreads requires to link with the pthread library. You need to add option `-lpthread` at the end of your compilation command.

  eg. `$gcc -o test -Wall my-code-with-pthreads.c -lpthread`

Exercise 1: First steps with threads

Write a program where the main thread creates N secondary threads, and passes their order of creation i (0 <= i <= N-1) as an argument of `pthread_create`.
Each secondary thread will execute a function `thread_control` which displays (a) the order of creation of the thread and (b) its identifier (`tid`), and then terminates and returns its order of creation multiplied by 2.
The main thread waits for the termination of every secondary thread and displays the termination value that was returned. When all secondary threads have terminated, the main thread completes its run.

Exercise 2: Mutual exclusion

Modify the program written for exercise 1 so that each secondary thread now executes a function `rand_thread` which generates a random value between 0 and 10 as follows:

```
random_val = rand()%10;
```

Every secondary thread displays the random value it generated (`random_val`), adds it to a global variable initialized to zero by the main thread, and then terminates by calling:

```
pthread_exit(0);
```

After waiting for the termination of all secondary threads, the main thread displays the total sum of all the values generated by the secondary threads.

Exercise 3: Synchronization & detached threads

Modify the program written for exercise 2 so that the final display of the running total is performed by an extra secondary thread instead of the main thread. This extra secondary thread executes a function `print_thread` that blocks until all the other secondary threads have finished adding up the random values they generate. All secondary threads that perform `rand_thread` set their state to detached from the start. The last `rand_thread` call unblocks the print thread.

Exercise 4: Synchronization by broadcast

Download and fill in the code of program that forces a rendez-vous point (RVP) between N secondary threads. When a thread reaches the RVP, it remains blocked until all others reach the RVP too.

If N is 3, the point is to get your program to display:

```
Before barrier
Before barrier
Before barrier
After barrier
After barrier
After barrier
```

Exercise 5: Reusable synchronization by broadcast

The previous exercise can be solved with a "disposable" barrier. Now let's try the same thing, but with threads that loop on the RVP.

Download and fill in the code of program that forces an iterable rendez-vous point (RVP) between N secondary threads. When a thread reaches the i^{th} RVP, it remains blocked until all others reach the i^{th} RVP too.

If N is 3 and threads iterate twice, the point is to get your program to display something like so:

```
START RVP ITERATIONS
before barrier 0
before barrier 0
before barrier 0
after barrier 0
before barrier 1
after barrier 0
before barrier 1
after barrier 0
before barrier 1
after barrier 1
after barrier 1
after barrier 1
END PROG
```

Your solution can only be correct if no thread will display:

- *before barrier i* once any other thread has displayed *before barrier i+1*
- *before barrier i* once any other thread has displayed *after barrier i*