CS4023 – Lab Exercise, Week 8

Linux Threads Communicating via Shared Memory

The goal of this exercise is understand the need of special mechanisms for synchronization when threads are sharing data.

Step 1. Open http://www.csis.ul.ie/coursemodule/CS4023/?wkid=8 and download the file **`example2.c`**.

Step 2. Open **`example2.c`** (with emacs, for example). This file contains the second solution to the *producer-consumer example* explained in Tutorial 2.

Here the *producer* and the *consumer* are two threads within the same process. The *producer* thread stores data in the shared array **`buffer`**, and the consumer reads from the array.

Note that that both the *producer* and the *consumer* threads run 1000000 iterations **`for`** loops before finally updating the **`count`** variable. The purpose of those loops is to slow down the update of **`count`** so that you can observe the effect of the lack of synchronization between the threads.

Step 3. Execute **`gcc example2.c`** at the command line to build the executable **`a.out`**. Then run the executable, i.e. execute **`./a.out`** at the command line.

Q1. How many items does the *producer* produce?

Q2. What is the size of the **`buffer`** array?

Q3. Change the size of the **`buffer`** array to 5 and repeat step 3. What happens now?

Step 4. Notice that the *consumer* thread is created only after the *producer* thread has completed its execution. Modify the code to let the two threads run in parallel and change the buffer size to 20. Rebuild the executable **`a.out`** and run it again.

Q4. What did you change in the code?

Q5. What is the result of executing the modified version?

Step 5. Now change the buffer size to 5 and rebuild the executable again. Run **`a.out`** at least 10 times, one after another.

Q6. What do you observe? Does the code seem to work correctly? If not, what could be the eventual problem?

IMPORTANT: In your own time write a report that describes your answers to **Q1-6**. This will become part of your end-of-semester project.