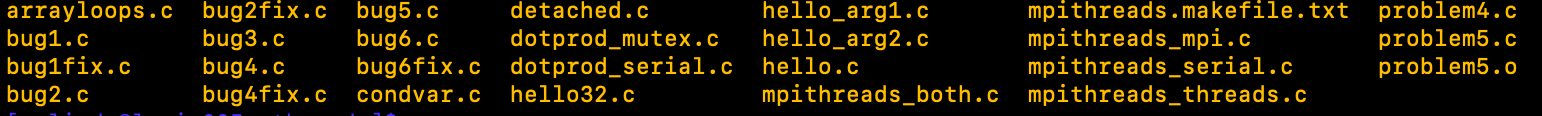
Oscar Galindo

CS 5334

Dr. Shirley Moore

Week 2 – Threading Exercises

* Exercise 1
  1. Logged in to bridges
  2. Downloaded all the files from the website and then used scp code/\* [galindo@bridges.psc.xsede.org](mailto:galindo@bridges.psc.xsede.org)
  3. Listed all the files



* 1. Loaded the gcc/9.2.0 compiler in bridges
  2. The output of problem 5, a threading exercise was varied, seemed consistent throughout the executions but, of course, the order changes sometimes.

A screen shot of a game

Description automatically generated with low confidence A screen shot of a game

Description automatically generated with low confidence

* 1. For problem 6 we have “compile and run the program. Notice the order in which thread output is displayed. Is it ever in the same order? How is this explained?”

Answer: No, the output is in different orders, and this is explained due to the scheduling policy of the OS, since the threads could be scheduled to run differently than the order at which they are created. At some, through pre-emption, jump and notice that thread 3 is ready to run instead of thread 1. Similarly, we could then jump to another thread and “let it run” because it is already passed 3 seconds, but the jumps between the threads are determined by the OS, hence the order is probably never the same.

* 1. For problem 7 the output looks as follows:

Text

Description automatically generated

There are multiple ways of solving this, the issue here is that the same variable we use in our loop to control the iterations is the variable that we then access when the threads run to actually printout the “id” of the thread. So, a quick solution is to pass the actual value instead of the address as we did before. Another solution would be to have a data structure at which we can contain the right “id”, or basically where the index is also the value in the entry of the data structure, and for which the address could still be passed, since no other thread would have access to that space of memory.

* 1. This is what happens after the bug5.c file is ran

Text

Description automatically generated

Honestly, I intuitively saw there was not a pthread\_exit at the end of main, so I placed it and saw that the code output this:

Text

Description automatically generated

So, this made me investigate further into what placing “pthread\_exit” at the end of main meant and I understood that the call makes main wait for the rest of the threads to end, without such call main basically returns, and the process is finished.

* 1. The output of the join program looks as follows:

Graphical user interface, text, application, chat or text message

Description automatically generated

This even seems to execute is a semi-logical order, except for when the main reports the creation of thread “3” the entire execution follows the sequence, create, start, calculate and report, join, and exit.

In contrast, when the detached threads are executed we see a much less “logical” transition of stages, as seen in the following example, the execution creates threads, starts them, then the main thread stops, and finally we get the result of the computations back from the threads. Is worth mentioning that under the detached construction of threads there is no need to actually exit in the threads, but rather it is only needed to end the execution. Main thread still need to exit.

Text

Description automatically generated

* 1. When the code for this problem is executed the following error is obtained:

Text

Description automatically generated’

In order to solve this, it is necessary that we explicitly define the correct size of the stack to be used for the threads.

* Exercise 2
  1. The issue with the code provided in bug6.c is that the summation is not locked, potentially either too many updates or too few updates might happen. The solution is to have a mutex before the critical section where we update the global variable of “sum”. The code for doing such change is as follows:

Text

Description automatically generated

And produces this output:

Graphical user interface, text, application, chat or text message

Description automatically generated

* 1. In this case the issue is that we focus the signal thread in a single or more than a single, although definitely not all the threads. Hence, when the iterations that must be reached by the increment-count threads are exhausted not all threads that observe the count are notified leaving some of those threads hanging. So, the change is simple and looks as follows:

Text

Description automatically generated