Labs 6: Race Conditions (v0.2)

CS211: Programming and Operating Systems

Weeks 10 and 11, 2020

Protocol on online labs

1. Access the lab through Blackboard... CS211... Virtual Classroom.
2. Turn video **off** at all times; turn on your mic only when asked. You may also be asked to share your screen. **Make sure you have no private material viable**.
3. When you enter add a “Chat” message (accessed through the “Collaborate panel” on the bottom left).
4. If you have a question, raise your hand (icon bottom centre).
5. Niall will then allocate you to one of the tutor Groups

Set-up

This lab builds on work from Lab 5, and also the Wednesday lecture of Week 9. Make sure you have studied both before starting.

In particular, it is important that you know how to use the `fork()` and `pipe()` system calls. Furthermore, since these are UNIX-related system calls, you'll need to complete these exercises using in suitable online compiler (such as `https://www.onlinegdb.com/online_c_compiler`). The one you used for Lab 5 should suffice.

In particular, `code::blocks`, with the *mingw* compiler, is not sufficient.

Code from Week 9

1. Verify that your chosen compiler can run `adder.c` from Week 9. You can download it from `http://www.maths.nuigalway.ie/~niall/CS211/Week09` Check that it produces the expected output.

2. Next, download, compile and run `adder_race_condition.c` (also at `http://www.maths.nuigalway.ie/~niall/CS211/Week09`) Read that code carefully, and make sure you understand it. Check that it does *not* produce the expected output.

3. A key line in `adder_race_condition.c`, which greatly increases the probability of a race condition is the `sleep(1)` instruction in the `child()` function (see Line 71). If that line is removed, does a race condition occur?

Too many children

You should have found that the race condition is unlikely when the `sleep()` call is removed from the `adder()` function. But we can still get a race condition if the are many child processes running. So,...

Modify the `adder_race_condition.c` program so that *K* children all try to add 4 numbers.

Changes that you need to make include:

1. `fork()` should be called exactly *K* times, and only by the parent (no child should call `fork()`).
2. `adder()` should be called by the parent *K* times.
3. Remove as much output from the code as possible, but most especially the two `printf` lines in the `child()` function. This is because some online compilers limit the run-time to 10 seconds, most of which is taken up with input and output
4. The `printf` line called by the children in main (i.e., Line 47 in the original version of the code) should be called only if $ans \neq 10$.

For what value of K is a race condition likely to occur?

Semaphore solution

Finally write a version of the code that uses a *semaphore* to avoid the race condition. One way to implement this is to create a pipe that is shared by all processes. Initialise it by writing one byte to the pipe. Specifically, you should

- Write a `Test()` function that checks if a resource is available by reading a byte from the pipe (which will cause the process that called `Test()` to wait if the pipe is empty).
- Write an `Increment()` function, which releases the resource by writing a byte to the pipe.
- Modify your `main()` function so that a lock is placed around a single line solving the race condition, by putting a call to `Test()` immediately before it, and to `Increment()` immediately after it. Correctly identifying the correct line is a crucial part of this assignment.
- Don't forget to initialise the semaphore: i.e., at the start, add a single byte to the pipe which indicates that the resource is available.

Your assignment

*Submit your final program by **5pm, 3 April**.*
As ever, it should include

- *your name,*
- *ID number, and*
- *email address.*

It should also include comments with the URL of the online compiler you used (or the name of the off-line compiler), and the value of K for which a race condition occurs when a semaphore is not used.