Labs 6: Race Conditions

CS211: Programming and Operating Systems
Niall Madden (Niall.Madden@NUIGalway.ie)

Weeks 10 and 11, 2021

This lab builds on work from Lab 5, and also material covered in Week 9. It will help to review 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.

. . . .

The proposed deadline is ***5pm, Tuesday, 4 May****.*

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. The the `sleep(1)` instruction in the `SubProc()` function of `adder_race_condition.c` (see Line 72) greatly increases the probability of a race condition. If that line is removed, does a race condition occur?

Exercise 1: how 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 subprocesses running. So,...

Modify the `adder_race_condition.c` program so that K subprocesses 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 subprocess 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 `SubProc()` 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 subprocesses in the `main()` function (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?

Exercise 2: Semaphore solution

Next write a version of the program 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

The proposed deadline is ***5pm, Tuesday, 4 May****.*

You should submit two programmes:

1. one which solves Exercise 1, and leads to a Race Condition;
2. one that solves Exercise 2, by implementing a semaphore.

As ever, your programmes should include

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

Your programmes should also include comments detailing the compiler you used, and case of Exercise 1, the value of *K* for which a race condition occurs when a semaphore is not used.

If you collaborated with anyone on this assignment, you should give their name and email address.