ENCE360 Lab 6: Threads

Objectives

This lab will introduce you to writing parallel programs using `pthreads`.

The aims of this lab are for you to understand:

1. The basics of how the `pthreads` library works.
2. How to convert simple serial code to work in paralle.

Preparation

Download and extract your Lab 6 files from `lab6Threads.zip` on Learn. This contains the files `thread.c` and `collatzSerial.c`.

You should compile your programs with the additional flag `-lpthread` at the very end to link the threading library.

Once you've completed the tasks below, make sure you also complete the quiz on the quiz server.

Program `thread.c`

This is a sample program to show how the memory model of threads works. First, compile this program with

```
gcc -o thread thread.c -lpthread
```

Run the program a few times, and make sure you can explain the results. Compare your results to what you'd expect from using processes instead of threads (if you still have the code, `fork.c` from lab 2 is a good comparison).

Parallelise `collatzSerial.c`

The Collatz conjecture is a mathematical problem that asks how many steps it takes for a given number to be reduced to 1, following a recursive algorithm. The details are not hugely important (though feel free to have a glance at the Wikipedia page), and you won't have to modify the function `collatzWalk()` at all, just `collatzSweep()` (i.e. the less mathy one). What is important to know is that the contents of the loop inside `main()` are *embarassingly parallel*, making them prime targets for large speedups via threading.

First, run `collatzSerial.c` so you can see what its output and execution duration is like.

Then, make a copy of `collatzSerial.c` called `collatzParallel.c`. Modify it so that all the elements in the `testValues` array are tested at the same time, with one thread each. You will need to change `collatzSweep()` as well as `main()`. Make sure you `join()` the threads only *after* they've all been spawned.

Once you've done this correctly, there should be an obvious speedup. You can measure this by using the `time` program, e.g.

```
time ./collatzSerial
time ./collatzParallel
```

By what factor is the execution sped up? Can you explain it?

Try changing the contents of `testValues` to be all identical. Does this change the speedup factor? What can we say about the order of the outputs?

Return from `collatzSweep()`

Make a copy of your `collatzParallel.c` called `collatzJoin.c`. Modify it so that instead of printing your results inside `collatzSweep()`, the results are returned via `pthread_exit()` to the spawning thread, and then printed there. This output will be different from above, and will be deterministic even if all the elements of `testValues` are the same. Can you explain why?