Lab 8

16 March 2017

In this lab, you'll see a bit of the experimental side of CS; in particular, measuring empirically the amount of time taken by different algorithms and data structures. Before you come to lab, you should read the instructions below, take notes in your notebook, and write a short program that can time things.

Timing things

If you were measuring how long something took in real life, you'd look at the clock before it started, then again afterwards, and subtract the minutes and seconds to find the result. That's essentially what we'll do here, except that the clock we look at is the computer system's internal clock, and we'll be counting in seconds and nanoseconds.

To get started, we'll write a program that just times a simple loop. This program will have four steps: check the time once, do something, check the time again, and then compare the times.

Create the program file `timing.cpp` with the usual includes and setup, but also

```
#include <ctime>
```

Inside the `main` function, type in the following two lines:

```
  timespec first_check;
  clock_gettime (CLOCK_MONOTONIC, &first_check);
```

This is a very C-ish idiom for making a system call: declare a variable (which will hold a value; in this case a clock reading), then call a function that takes the address of that variable, and afterwards the variable will have been updated with the new value. The only other relevant item here is the indication to use a monotonic clock—this is an indication to the system that we're using the clock as a stopwatch, rather than caring about the actual clock time per se.

I'll refer to that pair of lines (declaring a `timespec` variable and then calling `clock_gettime` with it) as "checking the time"; use a fresh variable each time so that you can compare them later. For now, add a second time check to the program.

Between the two time checks, add a loop that runs 1000 times. It doesn't even have to do anything; we're just killing time here.

Finally, after the second time check, you want to print the elapsed time between the checks. Before proceeding, consider for a moment how you would compute the amount of time between 3:46 and 5:10. Write out that problem, and the computation you performed to answer it, in your notebook. Take notes on the algorithm you used.

Now, I'll tell you that similar to a wall clock time (with one "field" marking hours and another marking seconds), `timespec` is a struct that contains a field `tv_sec` marking seconds and a field `tv_nsec` marking nanoseconds.[1] (There are 10^9 nanoseconds in a second.) Similar to the wall clock example, the nanoseconds "wrap around" to zero when the seconds increment; so even if you want a number of nanoseconds, you can't ignore the seconds. In your notebook, write out an expression (using `first_check.tv_sec` and `first_check.tv_nsec` and their counterparts from your second time check) that computes the total number of nanoseconds elapsed between the two time checks.

Type that into your program, and print out this number to `cout`. Compile and run your program.

See what happens if you vary the number of iterations in your "killing time" loop.

Modify your program to define a function `elapsed` that computes and returns the number of nanoseconds between one given `timespec` and another given `timespec`; use that function in the body of the code to print the number of nanoseconds between the two times.

[1]So, if you had a `timespec` value called `first_check`, you would access the number of seconds as `first_check.tv_sec` . Remember, structs are basically the same thing as classes in C++.