**CPSC 261: Lab 4**

**Lab 4 - Performance measurement and improvement**

* [Goals](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#goals)
* [Introduction](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#introduction)
* [Overview](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#overview)
* [The Details](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#details)
* [Provided Materials](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#provided)
* [Handing It In](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#handing)

**Goals**

The goals of this lab are to:

* Explore the performance of the fread and fwrite C library calls that you are now familiar with from the previous two labs, as well as the performance of the underlying system calls on which they depend.
* Understand how to improve the execution performance of an assembly language program

**Introduction**

In Labs 2 and 3 you used the fread and fwrite C library function calls to read and write data from files. In this lab you will be writing programs to measure the speed at which these functions can deliver data to and from files. These C library function calls depend on two other functions: read() and write() which are implemented by the underlying Linux operating system. We want to understand the performance of these two layers of the file system (the C library layer and the underlying operating system layer).

You have also seen in class how we were able to perform a number of improvements to the performance of the sum() function. You will be duplicating this effort on a slightly more complex assembly language function.

**Overview**

This lab has two main parts:

* First you must write four programs that read and write data to and from files. These will be called:
  + timeread
  + timewrite
  + timefread
  + timefwrite

After writing them and making sure that they work, you will use them to measure the speed at which they can be used to read and write data to and from files.

* Next, you must take an existing function assembly language function called issawtooth and make it run faster.

**The Details**

* [Step 1](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#step1) Write the timeread and timewrite programs
* [Step 2](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#step2) Write the timefread and timefwrite programs
* [Step 3](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#step3) Run your timing programs to measure their performance
* [Step 4](http://www.ugrad.cs.ubc.ca/~cs261/2013w2/labs/lab4.html#step4) Make issawtooth run as fast as possible
* Step 1 - Write the timeread and timewrite programs

You have been given a program named timenothing in a source file named timenothing.c. This program opens 2 Linux *special* files for reading and writing, but doesn't do any I/O to them. The two files are:

/dev/null

This file is the Unix analogue of a black hole. You can write as much data as you want to this file, and it just disappears from view. The file never gets any bigger or fills up. It is always 0 bytes long.

/dev/zero

This file is a never-ending source of bytes, all of them zeros. You can read from it forever and never reach the end of the file. However every byte that you read will be the null byte - with integer value 0.

Copy timenothing.c to timeread.c and timewrite.c and modify them to call the read() and write() functions respectively on the previously described special files: read from /dev/zero and write to /dev/null. Your program should read() (resp. write()) a total of limit / size times, each time reading or writing size bytes. Both the limit and size variables are defined in the main function of the timenothing program. When you run your program, the first argument to the program will be taken as a new value for the size variable, and the second argument will be taken as a new value for the limit variable.

The documentation for [read()](http://linux.die.net/man/2/read) and [write()](http://linux.die.net/man/2/write) is available on the web.

The special files /dev/null and /dev/zero are already opened for you by the main function of the timenothing program, and their file handles are available in the variables null and zero. All you need to do is add the calls to read() and write() in the body of the loop.

* Step 2 - Write the timefread and timefwrite programs

Make new program files timefread.c and timefwrite.c which are just like your timeread and timewrite programs, except that they call the fread() and fwrite() functions instead of read() and write().

The variables fnull and fzero contain file pointers (of type FILE \*) suitable for passing to fread() and fwrite().

Make sure that your programs work.

* Step 3 - Run your timing programs to measure their performance

Run all 4 of your timeread, timewrite, timefread, and timefwrite programs reading the same total number of bytes using 3 different values of the size parameter:

* + 1
  + 4096
  + 32768

You may want to modify the limit parameter if the program completes too quickly or runs for too long. However, once you have selected a good limit, you should use that same limit for all of your experiments. Make sure that you record the limit that you are using in your README.txt file. Carefully record how long each program takes to run for each size in your README.txt file. Run each program 3 times for each size to account for variations caused by external effects (other load on the computer, for example).

Document your observations about the performance of these programs in your README.txt file. What variations in performance do you see? Are any of these variations surprising to you? Can you explain what is causing the timing that you see?

* Step 4 - Make issawtooth run as fast as possible

You have been given a timeissawtooth.c file that runs an issawtooth function a bunch of times. You have also been given an implementation of the issawtooth function in an assembly language file named issawtooth.s. issawtooth.s is a slightly cleaned up version of the output of the C compiler compiling issawtooth.c with optimization level 3 (-O3).

Just to provide a bit of information about this issawtooth function, it's job it to ensure that the array of longs that it is passed has a *sawtooth* pattern. That is, the values in the array alternately increase and decrease by an arbitrary (> 0) amount.

Your task is to apply all the techniques that you know of to improve the performance of issawtooth. The techniques that we've talked about in class include:

* + Reducing the number of memory accesses
  + Reducing the number of branch instructions per iteration of any loops by rewriting the loop
  + Reducing the number of branch instructions per iteration by unrolling the loop and copying the body of the loop
  + Rearranging instructions or register usage to reduce the dependencies between instructions

Keep track of each thing that you try and keep track of whether it increases the performance (or decreases it!) and by how much. Keep your log in your README.txt file. You should be able to decrease the time taken by the program by about 50% if you are clever enough.

**Provided Materials**

You should use subversion to check out the lab4 directory from your repository, in the same way that you did for the previous labs.

Your lab4 directory should contain the following files:

* Makefile

The provided Makefile contains instructions for compiling and linking your programs. You should not need to, and should not, change the Makefile that we have provided for you.

* timenothing.c

This program contains an empty timing loop that can serve as the foundation for your programs that time data reading and writing.

* timeissawtooth.c

This program contains a main program that calls your issawtooth function and times its execution.

* issawtooth.s

The issawtooth function defined in assembler language.

* issawtooth.c

The issawtooth function defined in C in case it is helpful for you to read it.

You will need to create all of the other files.