# Comp111 Assignment 5

Benchmarking

## Overview

An operating system *benchmark* is a program that determines the speed of some operation within the operating system. There are several kinds of benchmarks, depending upon what one is measuring. In this assignment we will measure both the speed of disk I/O and the speed of cache hits of various kinds.

## Objective

Your objective is to write several benchmark programs that measure the following attributes (if possible):

a. (`a5a.c`) The speed with which one can read from the disk containing your home directory and class directory (it's the same for everyone). Please print the seconds needed to read 8192 bytes.

b. (`a5b.c`) The speed with which one can read from the local directory `/tmp` on your local machine (writable to everyone). Please print the seconds needed to read 8192 bytes.

c. (`a5c.c`) The speed with which one can read from the disk page cache. Please print the seconds needed to read 8192 bytes.

d. (`a5d.c`) The speed with which one can write to the disk containing your home directory (and/or class directory). Please print the seconds needed to write 8192 bytes.

e. (`a5e.c`) The speed with which one can write to the local directory `/tmp` on your local machine. Please print the seconds needed to write 8192 bytes.

f. (`a5f.c`) The speed with which one can write to the page disk cache. Please print the seconds needed to write 8192 bytes.

g. (`a5g.c`) The speed with which 8192 bytes can be fetched into the memory cache. Please print the seconds needed to fetch 8192 bytes.

In all cases, individual actions take a very short time and one must repeat an action multiple times to measure the time for one action.

Note that as usual, I have been a bit slimey and some of these are easy to measure, while some are exceedingly difficult to benchmark. Figuring out which ones is up to you!

A solution to this assignment is comprised of several programs `a5a.c-a5g.c` that construct experiments and then measure them. For each kind of measurement, your program should create a situation to measure, do the measurement, and then print the results. It should execute on `comp111-01` to `comp111-06`. It should only create files in the current directory of the invoked process or in `/tmp`. Please also ***clean up after each experiment***; leave no temporary files behind, especially in `/tmp`.

It is rather important to do your experimenting on `comp111-01.cs.tufts.edu` to `comp111-06.cs.tufts.edu`, because you are going to be taking up a substantive amount of CPU time and local resources to do this. Keeping CPU time from influencing your measurements is a significant part of the problem.

There is one file to help you: `/comp/111/assignments/a5/testfile.dat`:

```
-rw-r--r-- 1 couch faculty 1000000000 Dec  5  2010 testfile.dat
```

This is a really big file that you can use to test reads. However, I have been incredibly slimey and there is a "catch" to using this file. This is up to you to figure out.

## Hints

- As usual, you will need to repeat things that take a short amount of time many times in order to calculate their true time.
- Note that in this assignment, you will need real wall-clock time rather than CPU time. These are real-time measurements and have nothing to do with how much cpu time (user or system) is used.
- Your approach should work on `comp111-01` through `comp111-06`.
- Your home directory is a network disk. `/tmp` is always a local disk on a workstation, but is a network disk on `comp111-01` to `-06`. Local and network disk should differ in time needed to read a block.
- When you read a block from disk the first time, you read it from the disk itself. The second time, it is read from the page cache, so measuring the speed of the cache just requires repeating the operation and ignoring the first read.
- The speed of disk I/O can only be measured if the file to be written or read is not already in cache. Otherwise we are measuring the speed of the cache, and not the speed of the disk! Thus it becomes important to read a file that is not in cache, which means in turn that you must construct that file, wait for it to flush out of cache, and *then* measure how fast you can read it.

- I strongly suggest that you use raw I/O, i.e., `read()` and `write()` (instead of formatted I/O including `fread()`, `fprintf()`, `fwrite()`, and `fscanf()`) to read from and write to disk. Otherwise, you will see a buffering delay that is not what you should be measuring.
- To measure the size of the physical cache, construct an experiment whose runtime varies based upon whether there are cache hits or misses. Then look for statistical variation in the results. The slope of the plot of size versus runtime will change slope when you move outside of cache.
- Use `sleep()` to time out the cache as necessary. You may take up to *one minute* of real time to make each measurement. We are going to grade your measurements offline and while we sleep. If you take more than a minute, we will kill your solution and not issue points to it.

## Getting started

There are several starting files that simply document the objective of each program. These are located in `/comp/111/assignments/a5`. To get started:

```
mkdir a5
cd a5
cp /comp/111/assignments/a5/*.c /comp/111/assignments/a5/Makefile ./
make
```

This will give you starting programs that don't do anything interesting but do document what they're supposed to print at the end. Vary from this format at your own risk. It is provided so the assignment will be easy to grade.

## Submitting completed assignments

To submit this assignment, if your programs are "`a5a.c`" to "`a5g.c`", ssh to the server and type:

```
provide comp111 a5 a5a.c a5b.c a5c.c ...
```

where `...` is replaced with the other files you are submitting for grading. Each file should compile into a program on its own.

You will see your results with the command "`progress comp111`".