G

Laboratory: Systems Projects

NOTE: Projects are slowing being added to `https://github.com/remzi-arpacidusseau/ostep-projects`, which includes project descriptions and a simple testing framework. Please be sure to check that out if interested.

This chapter presents some ideas for systems projects. We usually do about six or seven projects in a 15-week semester, meaning one every two weeks or so. The first few are usually done by a single student, and the last few in groups of size two.

Each semester, the projects follow this same outline; however, we vary the details to keep it interesting and make "sharing" of code across semesters more challenging (not that anyone would do that!). We also use the Moss tool [M94] to look for this kind of "sharing".

As for grading, we've tried a number of different approaches, each of which have their strengths and weaknesses. Demos are fun but time consuming. Automated test scripts are less time intensive but require a great deal of care to get them to carefully test interesting corner cases. Check the book web page for more details on these projects; if you'd like the automated test scripts, we'd be happy to share.

G.1 Intro Project

The first project is an introduction to systems programming. Typical assignments have been to write some variant of the `sort` utility, with different constraints. For example, sorting text data, sorting binary data, and other similar projects all make sense. To complete the project, one must get familiar with some system calls (and their return error codes), use a few simple data structures, and not much else.

G.2 UNIX Shell

In this project, students build a variant of a UNIX shell. Students learn about process management as well as how mysterious things like pipes and redirects actually work. Variants include unusual features, like a redirection symbol that also compresses the output via gzip. Another variant is a batch mode which allows the user to batch up a few requests and then execute them, perhaps using different scheduling disciplines.

G.3 Memory-allocation Library

This project explores how a chunk of memory is managed, by building an alternative memory-allocation library (like `malloc()` and `free()` but with different names). The project teaches students how to use `mmap()` to get a chunk of anonymous memory, and then about pointers in great detail in order to build a simple (or perhaps, more complex) free list to manage the space. Variants include: best/worst fit, buddy, and various other allocators.

G.4 Intro to Concurrency

This project introduces concurrent programming with POSIX threads. Build some simple thread-safe libraries: a list, hash table, and some more complicated data structures are good exercises in adding locks to real-world code. Measure the performance of coarse-grained versus fine-grained alternatives. Variants just focus on different (and perhaps more complex) data structures.

G.5 Concurrent Web Server

This project explores the use of concurrency in a real-world application. Students take a simple web server (or build one) and add a thread pool to it, in order to serve requests concurrently. The thread pool should be of a fixed size, and use a producer/consumer bounded buffer to pass requests from a main thread to the fixed pool of workers. Learn how threads, locks, and condition variables are used to build a real server. Variants include scheduling policies for the threads.

G.6 File System Checker

This project explores on-disk data structures and their consistency. Students build a simple file system checker. The `debugfs` tool can be used on Linux to make real file-system images; crawl through them and make sure all is well. To make it more difficult, also fix any problems that are found. Variants focus on different types of problems: pointers, link counts, use of indirect blocks, etc.

G.7 File System Defragmenter

This project explores on-disk data structures and their performance implications. The project should give some particular file-system images to students with known fragmentation problems; students should then crawl through the image, and look for files that are not laid out sequentially. Write out a new "defragmented" image that fixes this problem, perhaps reporting some statistics.

G.8 Concurrent File Server

This project combines concurrency and file systems and even a little bit of networking and distributed systems. Students build a simple concurrent file server. The protocol should look something like NFS, with lookups, reads, writes, and stats. Store files within a single disk image (designed as a file). Variants are manifold, with different suggested on-disk formats and network protocols.

References

[M94] "Moss: A System for Detecting Software Plagiarism"
Alex Aiken
Available: http://theory.stanford.edu/~aiken/moss/