107

Laboratory: xv6 Projects

This chapter presents some ideas for projects related to the xv6 kernel. The kernel is available from MIT and is quite fun to play with; doing these projects also make the in-class material more directly relevant to the projects. These projects (except perhaps the first couple) are usually done in pairs, making the hard task of staring at the kernel a little easier.

107.1 Intro Project

The introduction adds a simple system call to xv6. Many variants are possible, including a system call to count how many system calls have taken place (one counter per system call), or other information-gathering calls. Students learn about how a system call actually takes place.

107.2 Processes and Scheduling

Students build a more complicated scheduler than the default round robin. Many variants are possible, including a Lottery scheduler or multi-level feedback queue. Students learn how schedulers actually work, as well as how a context switch takes place. A small addendum is to also require students to figure out how to make processes return a proper error code when exiting, and to be able to access that error code through the `wait()` system call.

107.3 Intro to Virtual Memory

The basic idea is to add a new system call that, given a virtual address, returns the translated physical address (or reports that the address is not valid). This lets students see how the virtual memory system sets up page tables without doing too much hard work. Another variant explores how to transform xv6 so that a null-pointer dereference actually generates a fault.

107.4 Copy-on-write Mappings

This project adds the ability to perform a lightweight `fork()`, called `vfork()`, to xv6. This new call doesn't simply copy the mappings but rather sets up copy-on-write mappings to shared pages. Upon reference to such a page, the kernel must then create a real copy and update page tables accordingly.

107.5 Memory mappings

An alternate virtual memory project is to add some form of memory-mapped files. Probably the easiest thing to do is to perform a lazy page-in of code pages from an executable; a more full-blown approach is to build an `mmap()` system call and all of the requisite infrastructure needed to fault in pages from disk upon dereference.

107.6 Kernel Threads

This project explores how to add kernel threads to xv6. A `clone()` system call operates much like fork but uses the same address space. Students have to figure out how to implement such a call, and thus how to create a real kernel thread. Students also should build a little thread library on top of that, providing simple locks.

107.7 Advanced Kernel Threads

Students build a full-blown thread library on top of their kernel threads, adding different types of locks (spin locks, locks that sleep when the processor is not available) as well as condition variables.

Requisite kernel support is added as well.

107.8 Extent-based File System

This first file system project adds some simple features to the basic file system. For files of type EXTENT, students change the inode to store extents (i.e., pointer, length pairs) instead of just pointers. Serves as a relatively light introduction to the file system.

107.9 Fast File System

Students transform the basic xv6 file system into the Berkeley Fast File System (FFS). Students build a new `mkfs` tool, introduce block groups and a new block-allocation policy, and build the large-file exception. The basics of how file systems work are then understood at a much deeper level.

107.10 Journaling File System

Students add a rudimentary journaling layer to xv6. For each write to a file, the journaling FS batches up all dirtied blocks and writes a record of their pending update to an on-disk log; only then are the blocks modified in place. Students demonstrate the correctness of their system by introducing crash points and showing that the file system always recovers to a consistent state.

107.11 File System Checker

Students build a simple file system checker for the xv6 file system. Students learn about what makes a file system consistent and how exactly to check for it.