Project 2b: xv6 Scheduler

Objectives

There are two objectives to this assignment:

- To familiarize yourself with a real scheduler.
- To change that scheduler to a new algorithm.

Updates

Read these updates to keep up with any small fixes in the specification.

When a bad priority (not 1 or 2) is passed to your setpri() system call, return -1 (indicating an error). Otherwise return 0 (indicating success). Similarly, if a bad pointer is passed to your getpinfo() call, return -1; otherwise, if the call is successful, return 0.

In your Makefile, replace `CPUS := 2` with `CPUS := 1` . The old setting runs xv6 with two CPU cores, but you only need to do scheduling for a single core in this project.

Overview

In this project, you'll be putting a new scheduler into xv6. It is called a simple **priority-based scheduler** . The basic idea is simple: assign each running process a **priority,** which is an integer number, in this case either 1 (low priority) or 2 (high priority). At any given instance, the scheduler should run processes that have the high priority (2). If there are two or more processes that have the same high priority, the scheduler should round-robin between them. A low-priority (level 1) process does NOT run as long as there are high-priority jobs available to run. Note that this is a simpler scheduler than the one discussed in the book chapter on MLFQ.

Details

You will need to implement a couple of new system calls to implement this scheduler. The first is **int setpri(int num),** which sets the priority for the calling process. By default, each process should get a priority of 1; calling this routine makes it such that a process can raise or lower its priority.

The second is **int getpinfo(struct pstat *).** This routine returns some basic information about each running process, including ~~how many times it has been chosen to run~~ how long it has run at each priority (measured in clock ticks) and its process ID. You can use this system call to build a variant of the command line program **ps,** which can then be called to see what is going on.

If either of these calls are passed bad parameters, return -1 to indicate a failure. Otherwise, return 0 upon success.

One thing you'll have to do is make sure to initialize the priority of a process correctly. All processes, when created, should start at priority level 1. Only by calling **setpri()** can a process change its priority to 2.

Tips

Most of the code for the scheduler is quite localized and can be found in **proc.c.** The associated header file, **proc.h** is also quite useful to examine. To change the scheduler, not much needs to be done; study its control flow and then try some small changes.

You'll need to understand how to fill in the structure **pstat** in the kernel and pass the results to user space. The structure looks like what you see in here.

Random xv6 tip: To run the xv6 environment, use **make qemu-nox.** Doing so avoids the use of X windows and is generally fast and easy. However, quitting is not so easy; to quit, you have to know the shortcuts provided by the machine emulator, qemu. Type **control-a** followed by **x** to exit the emulation. There are a few other commands like this available; to see them, type **control-a** followed by an **h.**

The Code

The source code for xv6 (and associated README) can be found in **~cs537-1/ta/xv6/** . Everything you need to build and run and even debug the kernel is in there.

You may also find the following readings about xv6 useful, written by the same team that ported xv6 to x86: xv6 book

Particularly useful for this project: Chapter 5.

What To Turn In

Turn in your source code, as usual.