Assignment 3

Priority-based Scheduler

Important Notes

1. This is an **individual** project. Do your own work!
2. The project is based on the Linux operating system.
3. Make sure your program works on linprog.cs.fsu.edu because that is where it will be graded.

Acknowledgment

This project is designed by Prof Zhi Wang in FSU Tallahassee which being derived from 600.318/418: Operating Systems at John Hopkins University.

Overview

In this assignment, you will learn how to implement a priority-based scheduler in Xv6. To get started, you need to clone xv6 from git://github.com/mit-pdos/xv6-public.git, which is a simple Unix-like teaching operating system from MIT.
In particular, you'll replace xv6's current *round-robin* scheduler with a ***priority-based*** scheduler.

Words of Wisdom: First, please start early! Second, please make minimal changes to xv6; you do not want to make it hard for us to grade!

Part 1: Priority-based Scheduler for Xv6 (50%)

In the first part, you will replace the round-robin scheduler for xv6 with a priority-based scheduler. The valid priority for a process is in the range of 0 to 200, inclusive. The smaller value represents the higher priority. For example, a process with a priority of 0 has the highest priority, while a process with a priority of 200 has the lowest priority. The default priority for a process is 50. A priority-based scheduler always selects the process with the highest priority for execution. If there are multiple processes with the same highest priority, the scheduler uses round-robin to execute them in turn to avoid starvation. For example, if process **A**, **B**, **C**, **D**, **E** have the priority of **30**, **30**, **30**, **40**, **50**, respectively, the scheduler should execute **A**, **B**, and **C** first in a round-robin fashion, then execute **D**, and execute **E** at last.

For this part, you will need to modify **proc.h** and **proc.c**. The change to **proc.h** is simple: just add an integer field called priority to struct proc. The changes to **proc.c** are more complicated. You first need to add a line of code in the **allocproc** function to set the default priority for a process to 50. Xv6's scheduler is implemented in the **scheduler** function in **proc.c**. The scheduler function is called by the **mpmain** function in **main.c** as the last step of initialization. This function will never return. It loops forever to schedule the next available process for execution. If you are curious about how it works, read ☞ Xv6 book/commentary, Chapter 5. In this part, you need to replace the scheduler function with your implementation of a priority-based scheduler. The major difference between your scheduler and the original one lies in how the next process is selected. Your scheduler loops through all the processes to find a process with the highest priority (instead of locating the next runnable process). If there are multiple processes with the same priority, it schedules them in turn (round-robin). One way to do that is to save the last scheduled process and start from it to loop through all the processes.

A major issue of the priority scheduling is starvation in which a low priority process never gets CPU time due to the existence of runnable higher priority processes. A solution to this problem is called aging. You will also implement aging for your scheduler. Specifically, if the process uses up its CPU time, you are going to decrease its priority by 2 (i.e., add 2 to its priority since lower numbers represent higher priority); if a process is waken up from waiting, increase its priority by 2. Keep in mind that you should always keep the priority in its valid range (0 to 200). In this part, you need to add some code to function **wakeup1** in **proc.c**, and function trap in **trap.c**.

Part 2: Add a Syscall to Set Priority (50%)

The first part adds support of the priority-based scheduling. However, all the processes still have the same priority (**50**, the default priority). In the second part, you will add a new syscall (**setpriority**) for the process to change its priority. The syscall changes the current process's priority and returns the old priority. If the new priority is lower than the old priority (i.e., the value of new priority is larger), the syscall will call **yield** to reschedule.

In this part, you will need to change **user.h**, **usys.S**, **syscall.h**, **syscall.c**, and **sysproc.c**. Review the ***pstree*** project to refresh the steps to add a new syscall. Here is a summary of what to do in each file:

- **syscall**.h: add a new definition for **SYS_setpriority**.
- **user.h**: declare the function for user-space applications to access the syscall by adding:

```
int setpriority(int);
```

- **usys.S**: implement the **setpriority** function by making a syscall to the kernel.
- **syscall.c**: add the handler for **SYS_setpriority** to the **syscall**s table using this declaration:

```
extern int sys_setpriority(void);
```

- **sysproc.c**: implement the syscall handler **sys_setpriority**. In this function, you need to check that the new priority is valid (in the range of [0, 200]), update the process's priority. If the new priority is larger than the old priority, call **yield** to reschedule. You can use the **proc** pointer to access the process control block of the current process.

Delverables

Submit modified xv6 project folder that implements the priority-based scheduler as a gzip compressed tarball. Make sure that this project includes a testing user program source that can validates the correctness of the implemented priority-based scheduler. In addition, make sure clean all object files by "make clean" before creating this tarball. The name of your attachment should be

cop4610-project3-yourname.tar.gz

with yourname is replaced by your CS account name in linprog.cs.fsu.edu.

Your submission will be graded by compiling and running it. We will first check the presence of your test program source that validates the priority-based scheduler with your test program. Second, we will validate your scheduler with our validation program for the correctness of your scheduler.

- Please make sure your source code can compile. Absolutely no credit if it does not compile.
- Please don't include the binary files. Do a **make clean** before submission. You'll make grading harder for us if you do.
- Please don't leave out any files! You'll make grading harder for us if you do.
- Please don't modify any files you don't need to! You'll make grading harder for us if you do.
- Please don't send us the meta-information from your revision control system! You'll make grading harder for us.