OS Assignment 2
Priority Scheduler

Madhur Singhal
2015CS10235

March 2017

1 Introduction

Scheduling is one of the fundamental tasks of any operating system. At a given time many applications or processes might be in progress but only a certain number of physical cores are present on a system. Thus an OS decides which process to run next and for what duration to run it. Priority scheduling is one method used in modern operating systems to allocate CPU time between processes. The idea is that each process is assigned a number which represents it's priority and when the OS has to decide the next process to run it chooses the one with the highest priority. In this assignment we implement a priority scheduler and add a very basic mechanism of starvation avoidance to it.

2 Adding priority to processes

This is the first and easiest task. First we add a field called priority inside the proc data structore inside **proc.h**. After this we modify the **sys_ps** system call to print the priorities of the processes too. Lastly we added a **sys_setpriority** system call which takes the pid of a process as argument and changes the priority to the given value. The same procedure involving changes to **syscall.c**, **syscall.h**,**usys.S**, **sysproc.c** and **user.h** was used to add this system call.

3 Priority Scheduler Implementation

The schedule function in **proc.c** was modified to force it to select the process with highest priority as the next process. In case there are multiple processes with the highest priority they are executed in a round robin fashion. The round robin part was implemented by keeping a pointer to the last scheduled process and beginning the search from the next element next time.

4 Starvation Avoidance

Finally we implemented a **sys_getpriority** system call using the same procedure as the first part. We also implemented an extension to the priority scheduling procedure to deal with the starvation problem. The problem is that processes with low priority might never actually get CPU time to execute and may wait for a very long time. To solve this we utilized a simple algorithm which incremented a process' priority by one each time the number of switches reported by it exceeded a threshold. This makes the priority scheduler run like round robin in the worst case but improves fairness by ensuring all processes can excess resources.