MLFQ and Lottery Scheduling in xv6

Operating Systems Lab # 4

By:
Mr. Muhammad Abdullah (2015-EE-166)

Submitted to:
Mam Sana Younas

EE431L: Operating Systems
Spring 2019
Date Submitted: March 10, 2019

MLFQ and Lottery Scheduling in xv6

Objective

In this lab, a new scheduler is implemented in xv6, which is a mixture of two schedulers i.e. the Multi-Level Feedback Queue (MLFQ) Scheduler and the Lottery Scheduler.

The MLFQ scheduler basically consists of two levels or queues, the high queue and the low queue. Each new process is initially placed on the high queue where it runs for one time slice, after which it is moved down to the low queue. In the low queue, each process runs for two time slices instead of one.

Within each queue, lottery scheduler is implemented. When two or more processes are present on a single queue, then each process runs in proportion to the number of tickets associated with it. A winner is selected randomly via a lottery which determines the process to be run for that particular time slice.

Procedure

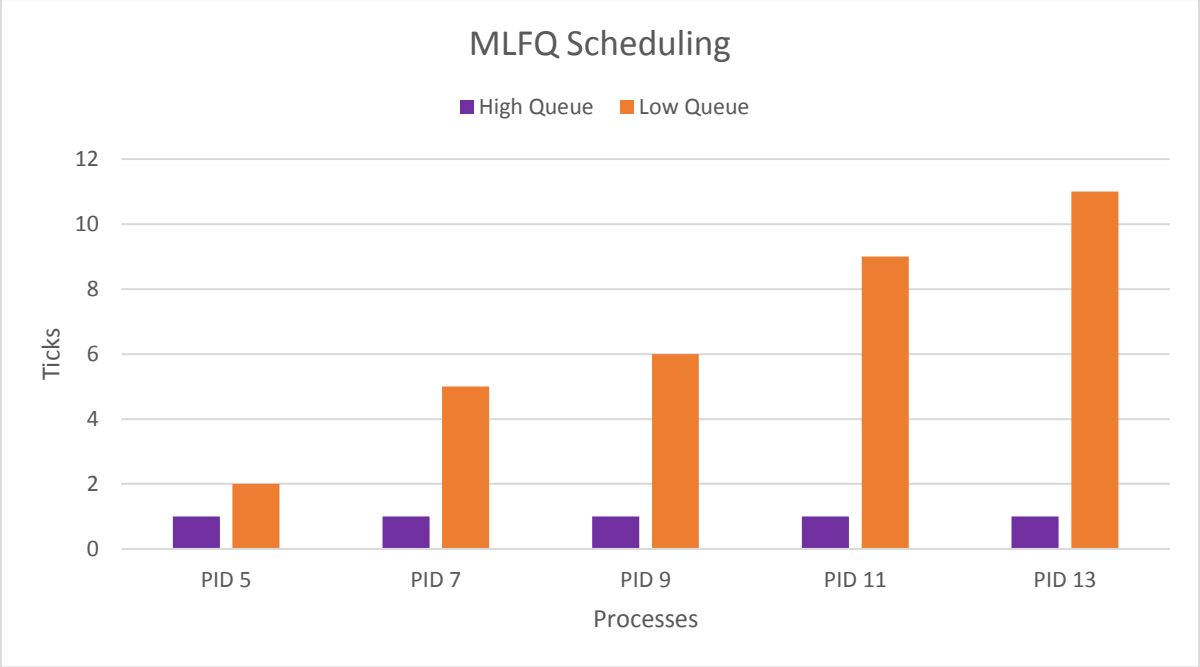
- Two new system calls **settickets** and **getpinfo** were implemented in **proc.c** to set the number of tickets for a process, and to get the information for currently running processes respectively.
- The file **pstat.h** was included to be utilized in **proc.c** as well as **main.c**
- Two new elements namely tickets and queues were added to the proc structure in **proc.h** to keep track of the number of tickets and current queue for each process.
- Changes were made in the **scheduler** function of **proc.c** to check the check and populate both the queues and then run the winning process from each queue after selecting it via lottery.
- The **yield** function of **proc.c** was also modified to run processes on high queue for single time slice and processes on low queue for two time slices.
- Finally, the **main.c** file was written to create multiple processes via **fork** and assign them different number of tickets and then check the cpu usage for each process through the number of ticks accumulated for each process.

Results

Five different processes were created via fork each containing an infinite while loop. Different number of tickets were assigned to each process. The number of ticks accumulated for each process were monitored after regular intervals via the procdump function.

PID	Tickets
5	100
7	200
9	300
11	400
13	500

The first graph depicts the MLFQ part of the scheduler. Five different processes are running each with different number of tickets. Each process runs for only one time slice in the high queue after which it is shifted to the low queue where it runs for the rest of its life, two time slices at once.



The second graph depicts the lottery part of the scheduler. Five different processes are running each with different number of tickets. Each process runs proportional to the number of tickets it has. More the number of tickets a process has, the more chances it has of winning the random lottery selection, and ultimate the more ticks it accumulates over time and hence the more cpu it consumes.