Assignment 4: Enhancing xv6

Aum Alok Khatlawala
Naval Surange

Note: Search for “// Assignment 4” in the files to see where all the modifications were made.

Implementation of Specification 1:

1. **trace:** struct proc has a new variable called mask. The trace system call stores the mask variable based on the input given and the syscall function in syscall.c checks if the binary bit value of mask contains the ith bit as 1 where i is the system all index defined in the kernel definitions. If it is set, it prints out the pid of the process, name of syscall, the arguments the syscall takes and the return value of the syscall.The user program strace is used to trace the syscalls called by the process specified. The modified areas of code are well commented explaining the logic behind every line of code written and the modified areas of code are as follows:
   I. struct proc has a mask variable added (in kernel/proc.h).
   II. sys_trace implemented at the end of kernel/sysproc.c.
   III. new file strace.c created in user directory to deal with the user side of the syscall and if it is a legitimate command, this function calls the syscall from kernel space.
   IV. syscall.c has been modified to account for the strace syscall.
2. **sigalarm and sigreturn:** The purpose of these two syscalls is to alert a process about its CPU time usage. To do so, we update the number of ticks at every time interrupt and if the number of ticks crosses the number of ticks given as parameter to the sigalarm syscall, the process calls a user defined handler function to alert the process of its CPU time usage. The sigreturn syscall returns the process to its normal state (trapframe info and so on is restored) before the handler function was called. The changes were made in similar places to the first syscall and the code is very well commented to explain each change made.

Specification 2:

Most of the changes were made in the proc.c file and the code is well commented to explain the thought process involved in implementing each algorithm. The following paragraphs give a brief explanation of the idea involved in implementing each algorithm.

1. **FCFS:** The process which was created first should be allocated the CPU first. Therefore, multiple new variables like ctime (set in allocproc() at the time of process creation), end_time and so on were created in struct proc to enable coding FCFS algorithm. The scheduler essentially chooses the process with

the minimum creation time and gives it to the CPU. As FCFS is non preemptive, certain timer disabling changes were also made in trap.c.

2. **Lottery Based Scheduler:** The process with the most number of tickets has a higher chance of winning the lottery and must thus be given the CPU first. This type of scheduling is preemptive and the code keeps this in mind and as per the specification mentioned in the assignment pdf, the scheduler assigns a time slice to the process randomly depending on how many tickets it owns. Thus, the higher the number of tickets owned by a process, the higher probability it has to be run by the CPU. The syscall settickets has been implemented in the kernel space and linked to the user space and has been used to effectively test LBS.
3. **PBS:** The process with the highest priority is given the CPU time first. Based on the assignment pdf, static priority, dynamic priority and niceness were defined in the algorithm implementation in order to store priority. Static priority is initialised to 60 as mentioned and it can be changed by invoking the syscall set_priority and if the priority of the process changes, rescheduling needs to be done. Each time the scheduler selects a process, the process with the highest dynamic priority is chosen. Certain other tie scenarios are mentioned in the code. The scheduler can be tested by changing the priority of different processes using the set_priority system call.
4. **MLFQ:** A simplified preemptive MLFQ scheduler using round robin for processes at the lowest priority queue has been implemented by using the synopsis provided in the assignment pdf. The scheduler allows processes to move between different priority queues based on their CPU consumption. If a process uses too much CPU time, it is pushed to a lower priority queue and starvation is taken care of using the concept of aging. The time slices for different queues have also been taken care of in the code in proc.c.

Performance Analysis of the different Schedulers:
Round Robin:
rtime: 122, wtime: 22

FCFS:
rtime: 67, wtime: 56

LBS:
rtime: 133, wtime: 15

PBS:
rtime: 110, wtime: 27

MLFQ:
rtime: 123, wtime: 23

As expected, FCFS has the highest waiting time and significantly lesser running time. MLFQ, PBS and Round Robin are better schedulers and LBS is a pretty random, probability based scheduler.