Gloria Nduka
April 2021
Project 3

The files I edited in the xv6-public-master folder are bolded below.

trap.c
I increment the amount of ticks by 1. Instead of yielding the CPU after every tick, I yield the CPU for 1 tick for the process in q0, yield the CPU for 2 ticks for process in q1, and yield the CPU for 8 ticks for process in q2 all on lines 108-116.

proc.c
This is where I did a majority of the project implementation.
I initialized the counters to be -1 in lines 27-29.

Allocproc(): In this function I am allocating the first unused process in the ptable.proc. I am initializing all the proc variables equal to 0. I then remove it from the corresponding queue and add the process to the end of q0. This is all on lines 82-195.

Scheduler(): In this function is where I implement the scheduler. I changed the default round-robin scheduling queue to a 3 level queue each with different priorities. Q0 is the highest priority then q1, then q2. In q0, a process would run for 1 tick then get moved to q1 and run for 2 ticks then get moved to q2 and run for 8 ticks. How I implemented it , I first found the first runnable process in the first possible queue; first checking q0 if it's not empty, q1, then q2. I then switch to user mode and execute the process with the function switchuvm(p). After the process gives up the CPU, I then switch back to kernel mode with the function switchkvm(). I then update the variables and information about that process such as the start time, duration, and the priority it has. After that I copy the process to the lower priority queue (if in q0 or q1) by adding it to the end of the lower priority queue then deleting it from the higher priority queue it was in. If the process is in Q2, I move it to the end of Q2 since there isn't a queue below it. The reason why I move it to the end of its own queue in Q2 is to ensure that round-robin scheduling is happening between the processes that have the same priority. I then call the boost() function right after. This function is implemented in lines 429-582.

Boost(): In this function I am implementing a priority boosting mechanism. This function increases the priority of a process that has been idle for a while to avoid starvation. After a runnable process has been idle and waiting in the lowest priority queue for 50 ticks or more, I move the process to the end of the highest priority queue and delete it from that lowest priority queue then shift all the cells to the left. The wait_time variable of all the processes in the lowest priority queue that is runnable is increased by the duration. This is all on lines 396-418.

getpinfo(): This function prints out the scheduling information about the process to the terminal. This function takes in a process ID as input, and it prints out the ticks at which it was scheduled, the duration it runs for before giving up the CPU, and the priority it has each time it is scheduled. This function returns a -1 if there is an error and returns 0 when it successfully outputs the information. This is all on lines 584-608.

proc.h

I declared the queues array q0[], q1[], q2[] and the counters c0, c1, c2 on top of proc.h. I also added times[3], ticks[3], wait_time, ticks_run and priority to the proc struct for helping me keep track of the scheduling information. The variable ticks_amt is used to determine the number of timer ticks the process has run for. The times and ticks array are to help with debugging. needed to schedule. The wait_time variable helps with figuring out when a boost can happen. The priority variable is the priority of the process and helps know when to yield the CPU. All this is in lines 62-75. I also added the function prototype int getpinfo(int pid) LINE 26.

pstat.h

Defined NSCHEDSTATS TO BE 1500. NSCHEDSTATS is the number of sched_stat_t slots per process. The scheduler fills in the slots as it schedules processes to record information about scheduling. I then added a sched_stat_t struct which has the variables start_tick, duration, and priority which is used for when each instance the process is scheduled. This holds the information that is returned to the user then getpinfo syscall is called.

syscall.c

Included extern int sys_getpinfo(void); line 106
In *syscall[] library/array [SYS_getpinfo] sys_getpinfo, LINE 130. Added getpinfo to syscall table.

syscall.h

I assigned getpinfo a unique number.
Line 23
#define SYS_getpinfo 22
This file contains symbolic definitions of system call numbers. Define a unique number for the system call.

sysproc.c

I implemented int sys_getpinfo(void) in lines 93-102. I validated the user input arguments and returns getpinfo function with id passed in as parameter.

user.h

I included the system call function prototypes. I added int getpinfo(int) on line 26.

usys.S
I included SYSCALL(getpinfo) on lines 32.

Makefile
I changed the number of CPUS to 1 because only 1 CPU is needed (line 220) to make testing less complicated. I also added the 3 tests in the UPROGS variable and the EXTRA variable so it can compile with all the other files when running make.

Test1.c, Test2.c,Test3.c

User level programs that tested priority of processes in the queues, boosting of processes, and cheating the CPU. Test 1 I included both IO intensive and CPU intensive tests. Test 2 I implemented a workload that shows priority boosting. Test 3 I cheated the CPU. I explain more in workload.pdf

Pr.pl, sign.pl
Didn't modify but claimed I modify since I accidentally copied old pr.pl and sign.pl files into new directory that I was pushing to github,

Vectors.pl
"Had undefined reference to vectors" so

```
sudo chmod +x *.pl
make clean
make qemu (or make qemu-nox)
```

modified this file