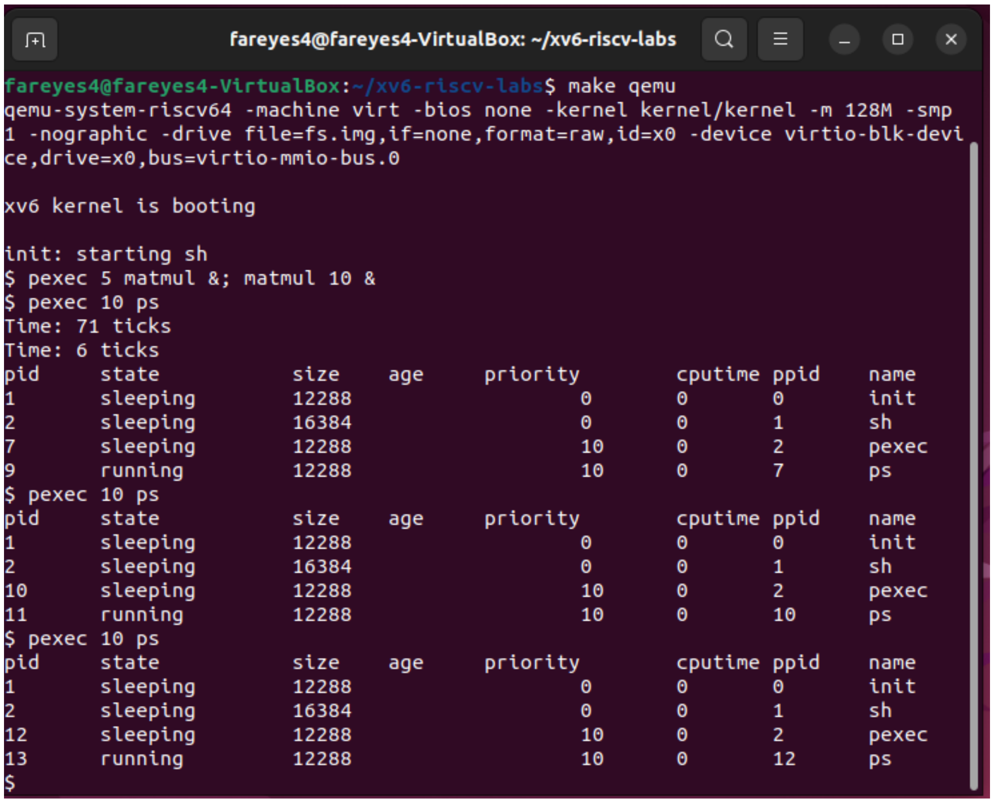
CS4375-13948 Fall 2023
Franz A. Reyes
Fareyes4@miners.utep.edu

Homework Report
Submitted on 10/18/2023

HW 3: Priority-based Scheduler for xv6

Task 1. Modify the provided ps command to print the priority of each process.

- I modified the user ps command by working the priority information. The modifications extended to the kernel-level getprocs() system call, where I included the priority field in the struct pstat defined in kernel/pstat.h. To achieve this, I modified the procinfo() helper function to populate the priority information when calling process details. This improved ps command now presents not only standard process data but also the priority of each process.
-



fareyes4@fareyes4-VirtualBox: ~/xv6-riscv-labs

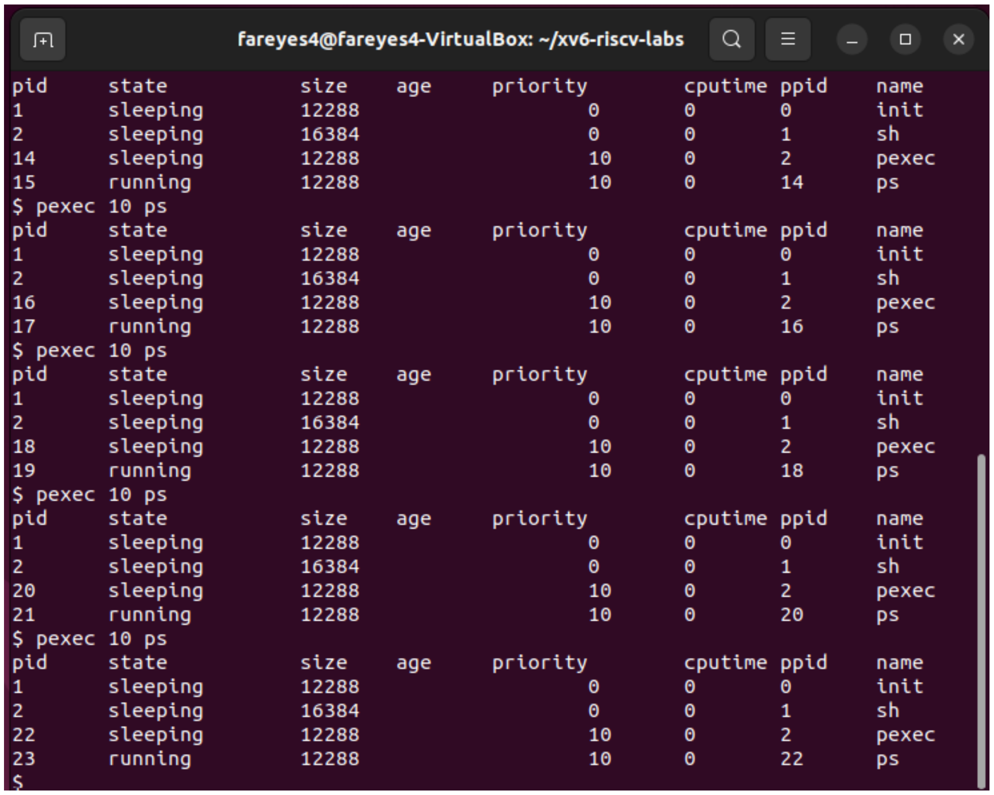
```
fareyes4@fareyes4-VirtualBox:~/xv6-riscv-labs$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
1 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-devi
ce,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ pexec 5 matmul &; matmul 10 &
$ pexec 10 ps
Time: 71 ticks
Time: 6 ticks
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
7       sleeping        12288                   10      0       2       pexec
9       running         12288                   10      0       7       ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
10      sleeping        12288                   10      0       2       pexec
11      running         12288                   10      0       10      ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
12      sleeping        12288                   10      0       2       pexec
13      running         12288                   10      0       12      ps
$
```

-

Task 2. Add a readytime field to struct proc, initialize it correctly, and modify ps to print a process’s age.

- To work a process's age within xv6, I introduced a "readytime" field in the process structure (struct proc) to mark when a process enters the RUNNABLE state. By calculating the time difference between the readytime and the current time, we obtain the process's age, representing the duration it has spent in the ready queue. This approach is crucial for averting process starvation. The modified ps command now not only displays conventional process data but also reveals the age of each process.

- This information is valuable for assessing how long processes have been waiting for execution. It helps evaluate process behavior and ensuring fairness within the system, as it allows processes with extended wait times to be considered for potential priority adjustments or scheduling decisions.
-



fareyes4@fareyes4-VirtualBox: ~/xv6-riscv-labs

```
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
14      sleeping        12288                   10      0       2       pexec
15      running         12288                   10      0       14      ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
16      sleeping        12288                   10      0       2       pexec
17      running         12288                   10      0       16      ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
18      sleeping        12288                   10      0       2       pexec
19      running         12288                   10      0       18      ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
20      sleeping        12288                   10      0       2       pexec
21      running         12288                   10      0       20      ps
$ pexec 10 ps
pid     state           size    age     priority        cputime ppid    name
1       sleeping        12288                   0       0       0       init
2       sleeping        16384                   0       0       1       sh
22      sleeping        12288                   10      0       2       pexec
23      running         12288                   10      0       22      ps
$
```

Task 3. Implement a priority-based scheduler.

- In the xv6 codebase, I made several key modifications to implement a priority-based scheduler. These changes included adding readytime and priority fields to struct proc in proc.h for age calculation and priority tracking. I also updated the scheduler in proc.c to prioritize processes based on their priority field, added readytime to struct pstat in pstat.h, and modified the ps program (ps.c) to display process priority and age. Running test programs, such as pexec, confirmed the scheduler's functionality, ensuring that higher priority processes were scheduled to run first.
- Task 3 showed me how to create schedulers that can be easily switched at compile time. By defining constants in param.h, I could switch between scheduling policies. I faced some difficulties in making this switch seamless, but I solved them by fine-tuning the scheduler code. Overall, the task highlighted the need for adaptable schedulers in operating systems.

Task 4. Add aging to your priority based scheduler.

- The aging policy I implemented in the priority-based scheduler helps a process's priority based on its wait time in the ready queue, preventing starvation. Running tests with aging confirmed its benefits by improving fairness among processes with different priorities.
- I encountered challenges related to time tracking but overcame them by carefully managing the readytime field, highlighting the importance of dividing the code into modules and understanding system performance in scheduling.

Extra Credit Task (10 points).

- The implementation of getpriority and setpriority commands involved creating system calls for accessing and adjusting a process's priority. These commands help tasks to dynamically change their priorities. By using setpriority to increase priority, jobs can receive faster CPU access, which also improves responsiveness for these tasks, allowing users to have more control over the scheduling.