

Christian Revilla

Dr. Moore

Operating Systems

October 14th, 2023

HW3: Priority-based Scheduler for xv6

Task 1:

In this assignment, we were asked to implement a priority-based scheduler for our xv6 processor. I first began by creating a new homework3 branch which will be the main source branch for this assignment. After successfully cloning my previous homework2 branch to have my most up to date code I was ready to begin. The first step was to add a new field in struct proc. In my implementation struct proc now has a new attribute that is int priority. From here I went to *syscall.h* to define two new system calls that will be implemented, *getpriority* and *setpriority*.

The next step was to modify the *syscall.c* file to add the newly defined system calls to the system call array and globally define them as uint64 in the Kernel code. Added entry in *us.pl* for both system calls to be able to generate an assembly file version for them. From here headed to *sysproc.c* to define the new *setpriority* and *getpriority* to be able to compile the code. At this point all I did was copy the system wait function and change the name to the new system calls. In my implementation both set and get priority take in one argument. In set priority it takes in an int

that will be assigned to the priority of the call and in get priority we take an int PID so we can return the correct priority number associated with the correct call. Finally, the last thing there was left to do was to modify the fork system call so the child process inherits the parent's priority. The main difficulty I encountered in this task was understanding that modifying fork was needed to pass the parents priority to the child. Overall set and get priority are simple getters and setters so wasn't too difficult to implement them. Below is a screenshot of my test run.

```

carevilla@carevilla-virtual-machine: ~/Documents/os-code
carevilla@carevilla-virtual-machine: ~/D... x carevilla@carevilla-virtual-machine: ~/D... x v
init: starting sh
$ ps
pid    state    size    priority    ppid    name
1      sleeping 12288    0           0       init
2      sleeping 16384    0           1       sh
3      running  12288    0           2       ps
$ pexec 10 ps
pid    state    size    priority    ppid    name
1      sleeping 12288    0           0       init
2      sleeping 16384    0           1       sh
4      sleeping 12288    10          2       pexec
5      running  12288    10          4       ps
$ pexec 5 matmul 5&; matmul 10&;
$ pexec 10 ps
pid    state    size    priority    ppid    name
1      sleeping 12288    0           0       init
2      sleeping 16384    0           1       sh
12     sleeping 12288    10          2       pexec
13     running  12288    10          12      ps
8      sleeping 12288    5           1       pexec
10     runnable 12288    0           1       matmul
11     runnable 12288    5           8       matmul
$ Time: 344 ticks

```

Task 2:

For the task 2 portion of homework 3 we were assigned to add a *readytime* field to *struct proc* and initialize it to the current time each time the process became runnable. To achieve this, I first modified the *pstat* header file in the *kernel* and added the attribute *int readytime* to the *struct pstat*. This attribute will track the time since the process has switched to the *RUNNABLE* state. The next step after adding the attribute was to modify *procinfo* to store the newly added

readytime field to be accessible to user mode. The final step in this task was to modify the *ps.c* file in user space to print the current age of the process. To achieve this, I modified the print statement to add the age field (column) to the output. For processes that are in the RUNNABLE state I printed the age field and for the others I print a tab character (nothing). I did this with a switch statement. Per the homework instructions the age of the process is computed by determining the difference between the current time and the time the process entered the RUNNABLE state. After trial and error, I got the output to come out correctly as shown below.

Task 3:

Per the instructions I added constants to the *param.h* file. I defined two constants, the MAXPRIORITY which is the maximum priority a process can give itself. And the MAXEFFPRIORITY which is the maximum priority a process can reach as it waits for its turn to execute. I created a *highest_priority* function to return the process in the table with the highest priority. I used a compiler directive to control the priority code. If the PRIORITY flag is set by the compiler, then the scheduler will use the priority code to determine which process should execute next. For ties I simply select the last process with the maximum priority as the one to execute. The reason for that is that if I select the first, I could not reach the *shell* process. I also didn't implement a random selector. The figure below shows that the *matmul* process is interrupted by the *ps* process since the *ps* has a higher priority. We can also see that I typed *ps* on the shell prompt, but this process needs to wait until the *matmul* completes since it has a priority of 0. We can confirm it by the output of the *matmul* (the number of ticks it took to complete) that is printed before the *ps* output.

```

Oct 20 20:36
carevilla@carevilla-virtual-machine: ~/Documents/os-code
carevilla@carevilla-virtual-machine: ~
carevilla@carevilla-virtual-machine: ~/D...
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-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

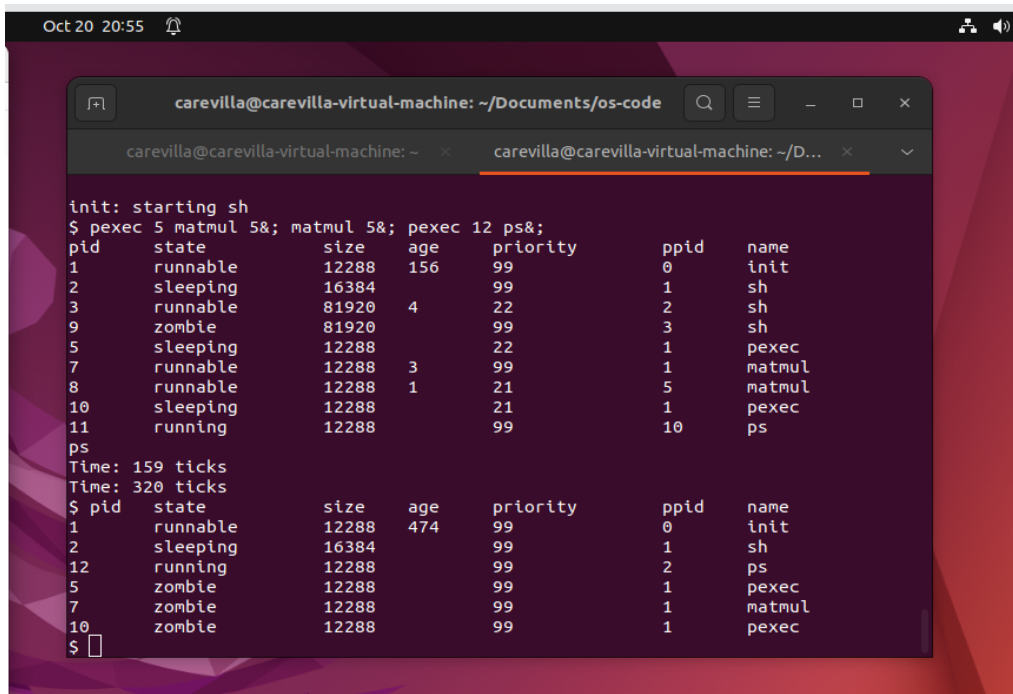
init: starting sh
$ pexec 10 matmul 10&; pexec 20 ps&;
$ pid      state      size    age      priority    ppid    name
1          runnable    12288   209      0           0       init
2          sleeping    16384           0           1       sh
8          runnable    12288           10          5       pexec
9          running     12288           20          7       ps
5          sleeping    12288           10          1       pexec
7          sleeping    12288           20          1       pexec

ps
Time: 318 ticks
pid      state      size    age      priority    ppid    name
1          runnable    12288   528      0           0       init
2          sleeping    16384           0           1       sh
10         running     12288           0           2       ps
5          zombie     12288           10          1       pexec
7          zombie     12288           20          1       pexec
$

```

Task 4:

I implemented the aging policy listed in the first page of the homework. That is, the effective priority is increased to a maximum of 99 the longer the process was last in the runnable state. This is achieved by computing the effective priority of the processes before comparing them. Effective priority overrides the initial priority eventually once it exceeds the initial priority. Unfortunately, as all the programs reach 99 for an effective priority and I use the selector for the last process, then that effectively makes the last process entered the one that executes next. This is visible on the screenshot below.



The screenshot shows a terminal window titled "carevilla@carevilla-virtual-machine: ~/Documents/os-code". The terminal displays the following commands and output:

```
init: starting sh
$ pexec 5 matmul 5&; matmul 5&; pexec 12 ps&;
```

pid	state	size	age	priority	ppid	name
1	runnable	12288	156	99	0	init
2	sleeping	16384		99	1	sh
3	runnable	81920	4	22	2	sh
9	zombie	81920		99	3	sh
5	sleeping	12288		22	1	pexec
7	runnable	12288	3	99	1	matmul
8	runnable	12288	1	21	5	matmul
10	sleeping	12288		21	1	pexec
11	running	12288		99	10	ps

```
ps
Time: 159 ticks
Time: 320 ticks
$ pid state size age priority ppid name
1 runnable 12288 474 99 0 init
2 sleeping 16384 99 1 sh
12 running 12288 99 2 ps
5 zombie 12288 99 1 pexec
7 zombie 12288 99 1 matmul
10 zombie 12288 99 1 pexec
```