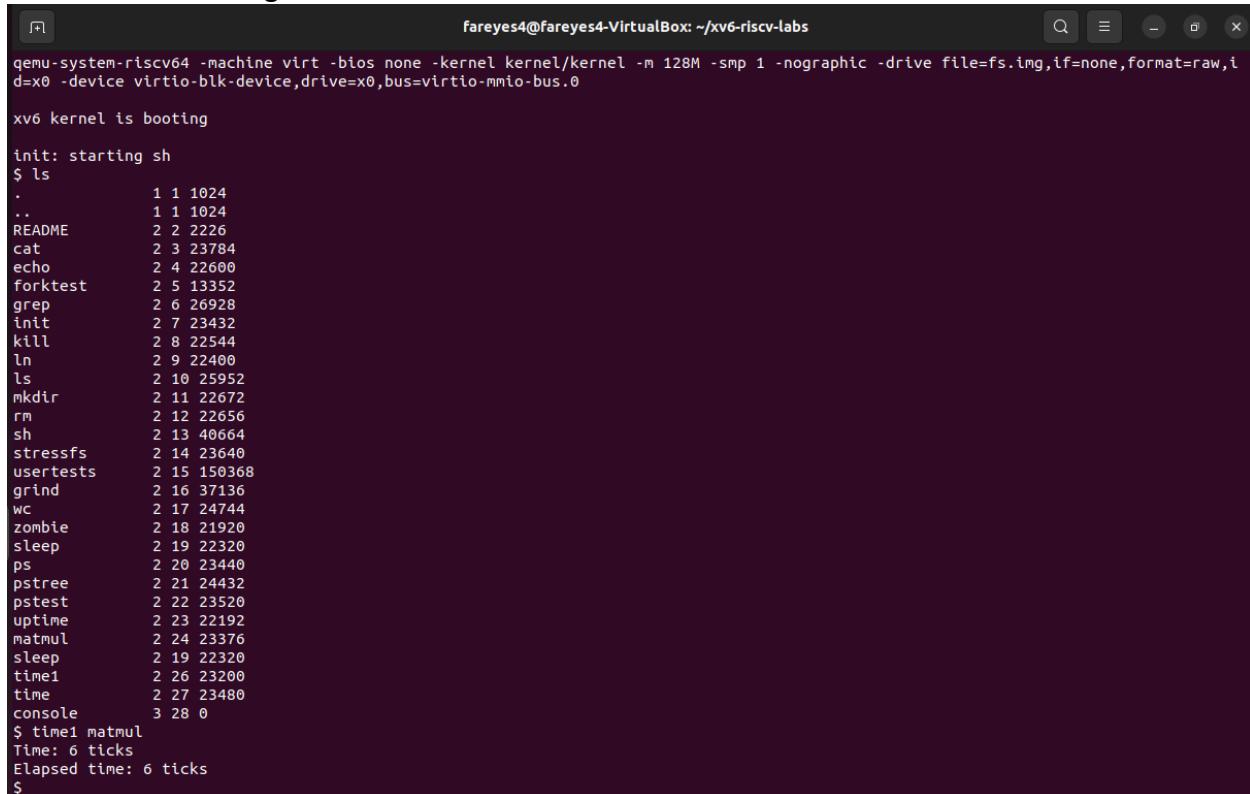


HW 2: Time Command

Task 1. Implement a time1 command that reports elapsed time.

Results from running the time1.c command:



```

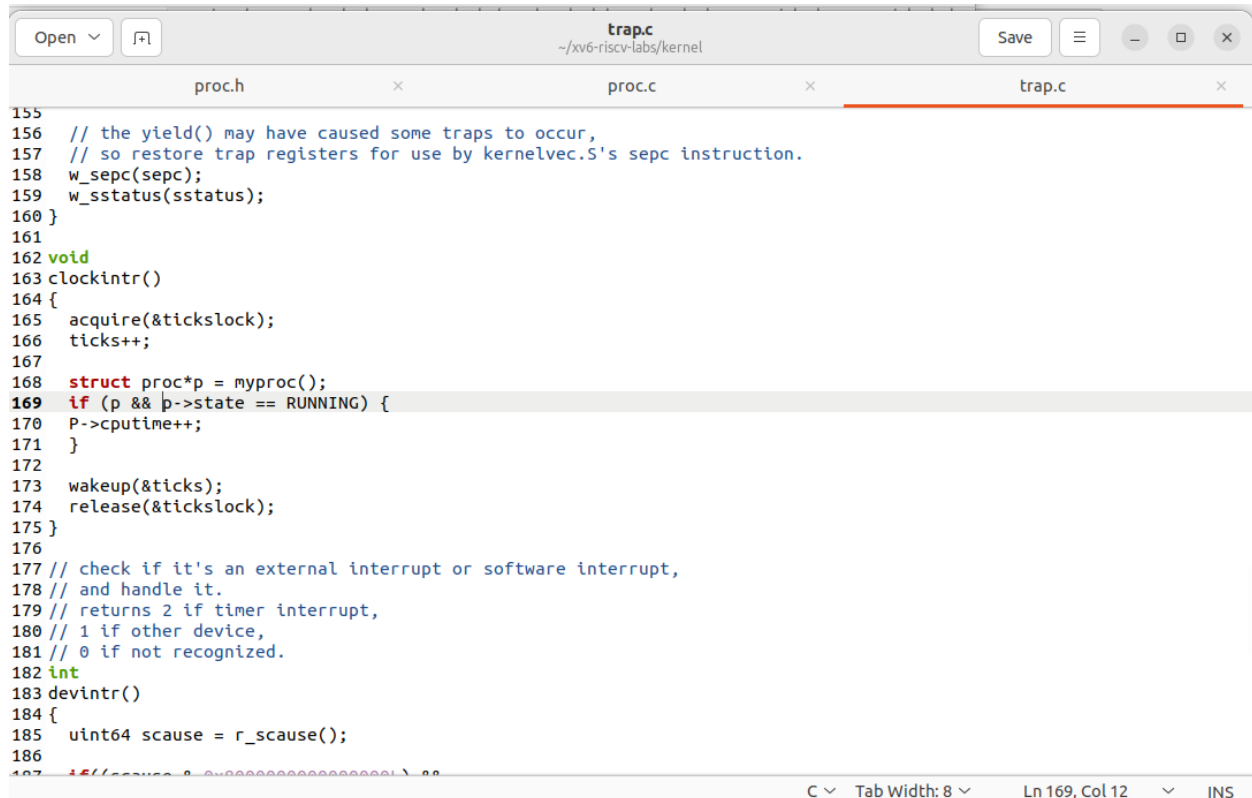
fareyes4@fareyes4-VirtualBox: ~/xv6-riscv-labs
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2226
cat        2 3 23784
echo       2 4 22600
forktest  2 5 13352
grep       2 6 26928
init       2 7 23432
kill       2 8 22544
ln         2 9 22400
ls         2 10 25952
mkdir      2 11 22672
rm         2 12 22656
sh         2 13 40664
stressfs   2 14 23640
usertests  2 15 150368
grind      2 16 37136
wc         2 17 24744
zombie     2 18 21920
sleep      2 19 22320
ps         2 20 23440
pstree     2 21 24432
pstest     2 22 23520
uptime     2 23 22192
matmul     2 24 23376
sleep      2 19 22320
time1      2 26 23200
time       2 27 23480
console    3 28 0
$ time1 matmul
Time: 6 ticks
Elapsed time: 6 ticks
$
  
```

- I was able to develop the time1.c file and add it to the user directory along with the matmul.c and sleep.c files and make the necessary changes to Makefile to run the time1.c command.
- What I learned from this task is the ability to system call to create a new process and also how to replace the current process. This task also helped me get a better understanding regarding measuring elapsed time and the new commands within the xv6 operating system.
- Some difficulties I encountered was some authentication failed errors as well as some command update errors that kept me busy for a while but I was able to find the solutions to those errors and work my way through the task.

Task 2. Keep track of how much cputime a process has used.



```
155
156 // the yield() may have caused some traps to occur,
157 // so restore trap registers for use by kernelvec.S's sepc instruction.
158 w_sepc(sepc);
159 w_sstatus(sstatus);
160 }
161
162 void
163 clockintr()
164 {
165     acquire(&tickslock);
166     ticks++;
167
168     struct proc*p = myproc();
169     if (p && p->state == RUNNING) {
170         p->cputime++;
171     }
172
173     wakeup(&ticks);
174     release(&tickslock);
175 }
176
177 // check if it's an external interrupt or software interrupt,
178 // and handle it.
179 // returns 2 if timer interrupt,
180 // 1 if other device,
181 // 0 if not recognized.
182 int
183 devintr()
184 {
185     uint64 scause = r_scause();
186
187     if (scause & 0xffffffff) {
```

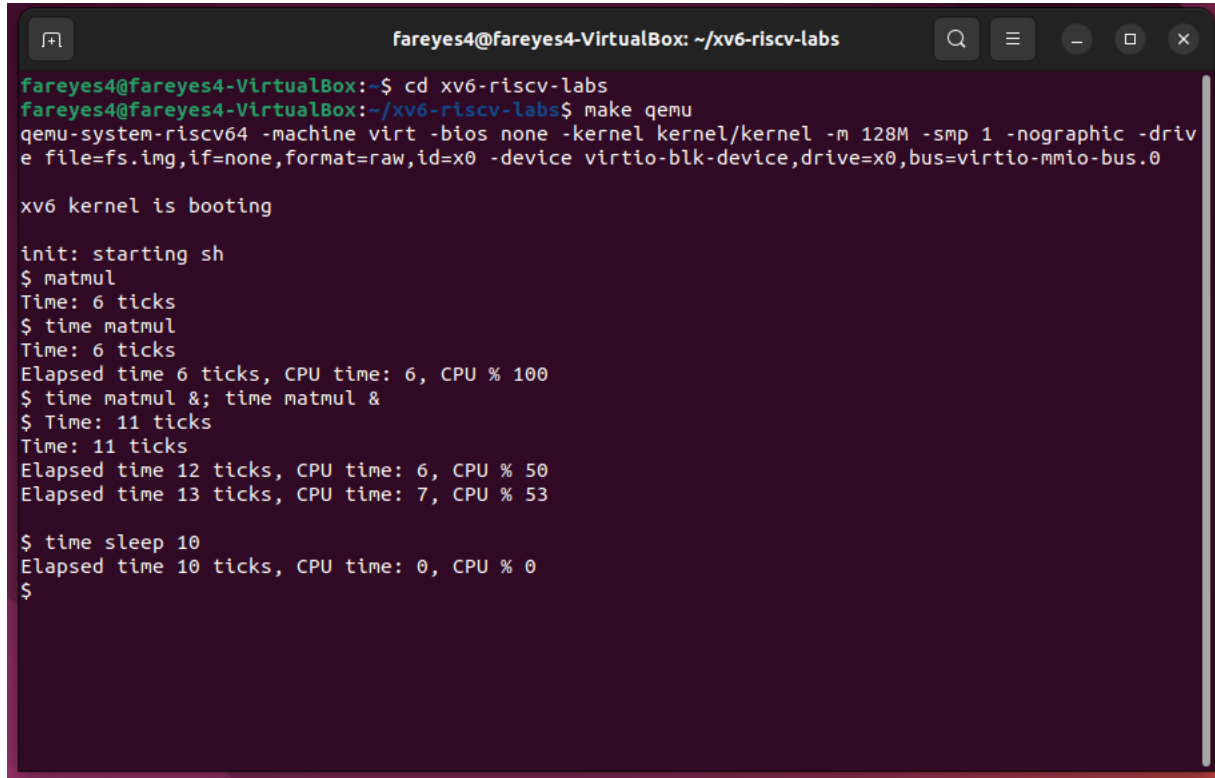
- I made the necessary changes to proc.h where I added a cputime field and to proc.c where I initialized that same field to zero. Then, I implemented the code in trap.c that increments the cputime field of the current running process.
- First, I learned about how cpu time works and how each process consumes it. I was also introduced to scheduling and how it works regarding proper resource consuming. It is also important to notice that this task helped me realize how important is to keep track of how much a cputime a process can take and how it can be improved.
- This task went very smooth because the instructions were very specific and clear and that made this task a little more achievable. Still, working the code in the trap.c file was very hard due to some syntax errors which were clarified after a few minutes.

Task 3. Implement a wait2() system call that waits for a child to exit and returns the child's status and rusage.

- pstat.h: Defined structure rusage for cpu time information in order to keep track of the cpu usage.
user.h: Declared system call wait2 and the cpu time structure for access.
usys.pl: Registered wait2 to register in the system call table and in the operating system.
syscall.h and syscall.c: makes sys_wait2 and wait2 implementation for functional purposes in kernel for it to respond.
sysproc.c: Implemented sys_wait2 system call handler to return status and the cpu time registered.
proc.c: Implemented wait2 function that acts by retrieving the child processes to get cpu time and status.
- I learned how to define from scratch and be able to use a new system call. I was also introduced to a new system call handler that performs actions based on response from other processes. I got a little more knowledge regarding struct proc and how it extends the process. Finally, I was able to get a little more understanding on synchronization.
- This was a very hard task for me since the instructions were a little tricky and I had to keep a very organized log on the changes I made to every single file. It was very difficult keeping track of any changes made to all the different files that were involved in this task. I had a few errors regarding access to shared data but after carefully reviewing the instructions I cleared that problem.

Task 4. Implement a time command that runs the command given to it as an argument and outputs elapsed time, CPU time, and %CPU used.

- The results running the time command:



```
fareyes4@fareyes4-VirtualBox: ~/xv6-riscv-labs
fareyes4@fareyes4-VirtualBox:~$ cd 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-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ matmul
Time: 6 ticks
$ time matmul
Time: 6 ticks
Elapsed time 6 ticks, CPU time: 6, CPU % 100
$ time matmul & time matmul &
$ Time: 11 ticks
Time: 11 ticks
Elapsed time 12 ticks, CPU time: 6, CPU % 50
Elapsed time 13 ticks, CPU time: 7, CPU % 53

$ time sleep 10
Elapsed time 10 ticks, CPU time: 0, CPU % 0
$
```

Extra Credit (5 points). Discuss limitations of our time command. (Hint: For one limitation, consider what would happen if the command that is being timed forks child processes).

- One limitation to our time command is that it does not provide an accurate representation of real time, it represent time by cpu ticks based off on cpu execution.
- All units such as cpu %, elapsed time and ticks are all not accurate at all and will be very different based on the operating system, hardware, etc.
- It is too simple, there are not many ways to improve the command and make it more specific, precise, or accurate. I don't think it is very practical.