CS4375-13948  Fall 2023
**Edmundo Pariente**
**eipariente@miners.utep.edu**

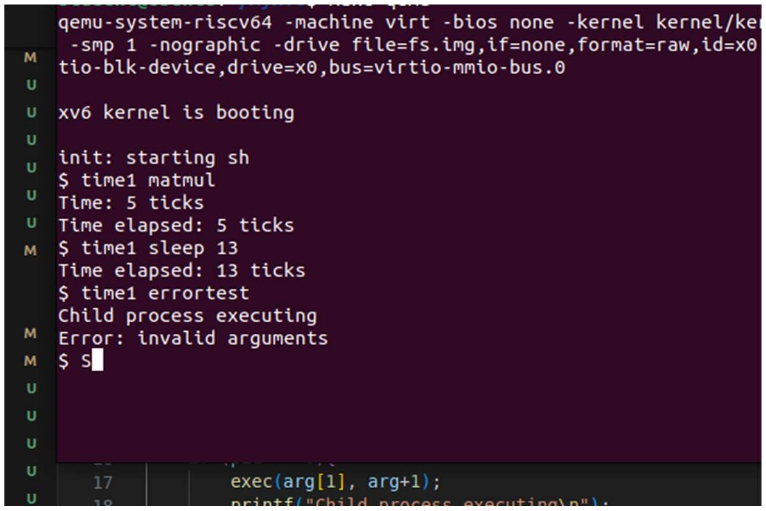
Homework Report
Submitted on **10/4/2023**

# HW 2: Time Command

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

This task helped to understand better the fork() function and how it handles the creation/execution of a child process with the ID that it returns.

One of the things that I really struggled with was unrelated because the Makefile was giving me errors to compile, I had to get rid of the "int" return type normally used in C for the main, that seemed to solve the problem.



```
qemu-system-riscv64 -machine virt -bios none -kernel kernel/ke
 -smp 1 -nographic -drive file=fs.img,if=none,format=raw,id=x0
tio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ time1 matmul
Time: 5 ticks
Time elapsed: 5 ticks
$ time1 sleep 13
Time elapsed: 13 ticks
$ time1 errortest
Child process executing
Error: invalid arguments
$ S
```

```
        exec(arg[1], arg+1);
```

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

I changed proc.h file in the kernel folder to include an integer variable called cputime inside the struct "proc". Inside the the c file of proc, I initialized the value of the cputime field to 0 whenever the method allocproc() is called.

Finally, in the trap.c file I included the expression p->cputime += 1 in both the the usertrap() and kerneltrap() methods but only inside the if statements that get triggered when there's a timer interrupt. This would increase the count by 1 every time this happens.

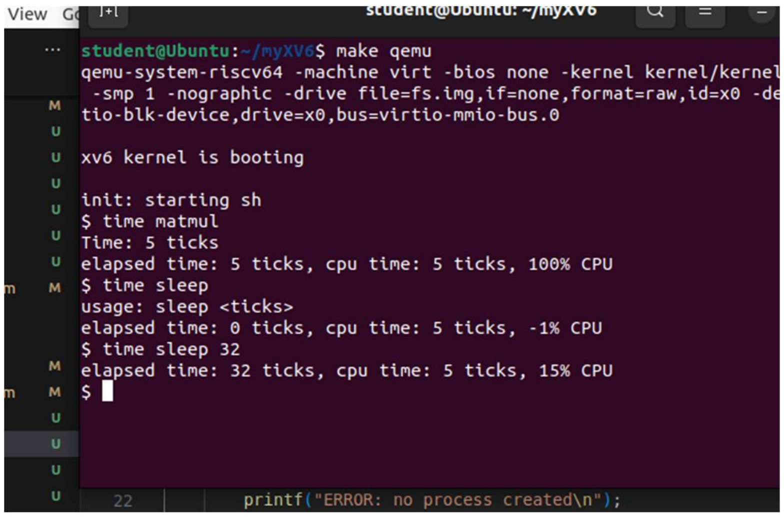
**Task 3. Implement a wait2() system call that waits for a child to exit and returns the child's status and rusage.**

I followed the instructions for this task, and I ended up doing changes in a lot of files, I had to create a file for the rusage struct to keep track of CPU time inside the kernel folder, called "pstat.h". Then I added the system call for the wait2() method in the user.h file so that it can have a definition and be visible to the rest of the files. I added the wait2() entry in usys.pl, I'm not exactly sure what that did.

I added the wait2 system call to the syscall.h and syscall.c files and added extra functionality to the regular sys_wait and wait() methods to handle the extra argument in sysproc.c and proc.c files.

This task was incredibly difficult, mainly because it involved working with too many files that I struggled to keep track of and remember what each of them was doing, it definitely helped me learn that xv6 and operating systems in general have a good delegation system among their modules.

**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.**



```
student@Ubuntu:~/myXV6$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel
 -smp 1 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -de
tio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

init: starting sh
$ time matmul
Time: 5 ticks
elapsed time: 5 ticks, cpu time: 5 ticks, 100% CPU
$ time sleep
usage: sleep <ticks>
elapsed time: 0 ticks, cpu time: 5 ticks, -1% CPU
$ time sleep 32
elapsed time: 32 ticks, cpu time: 5 ticks, 15% CPU
$ 
```

```
        printf("ERROR: no process created\n");
```

**Extra Credit (5 points). Discuss limitations of our time command.**

To my understanding the time command we did only works with 1 CPU, incrementing the number of CPUs could potentially change the time elapsed in all commands and therefore we would be getting erroneous information when reporting CPU usage as it only deals with 1 CPU at a time.