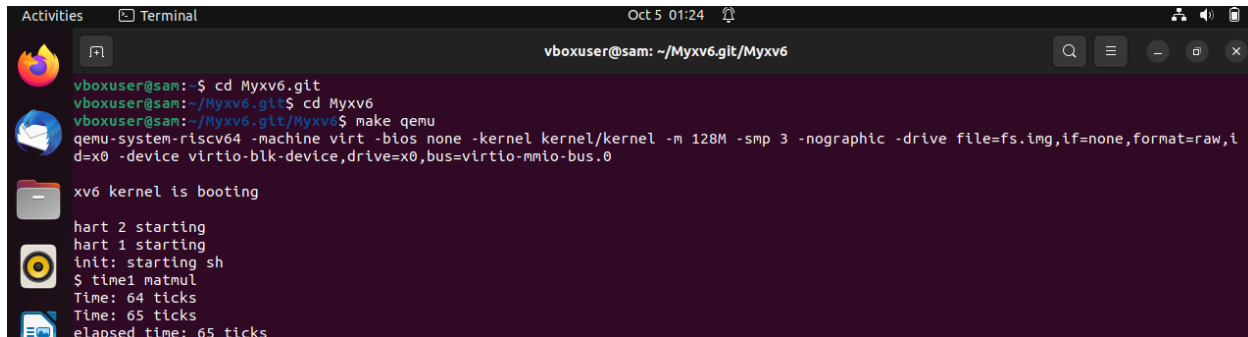


Time Command

Task 1. Implement a time1 command that reports elapsed time.
My time1.c code is added to my hw2 branch in my repository.
The command was added to UPROGS in the Makefile.

Results of time1 command:



```
Activities Terminal Oct 5 01:24
vboxuser@sam: ~/Myxv6.glt/Myxv6
vboxuser@sam:~$ cd Myxv6.glt
vboxuser@sam:~/Myxv6.glt$ cd Myxv6
vboxuser@sam:~/Myxv6.glt/Myxv6$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -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
hart 2 starting
hart 1 starting
init: starting sh
$ time1 matmul
Time: 64 ticks
Time: 65 ticks
elapsed time: 65 ticks
```

What I learned from this task:

By carrying out the task of implementing time1 command, I learned about different system calls that I used to create the time1 command like fork which creates a new child process that is a copy of a parent process. System call wait which is used by parent process to wait for the child process to finish executing and the exec the system calls which replaces the current process with a new program.

Difficulties I ran into: Using and understanding what other processes were doing while creating my time1.c to overcome this problem I had to read and test the system calls to better understand them.

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

Files changed:

- proc.h: I changed this header file because it has struct proc which is the representation of a process in xv6. In struct proc we are adding an integer cputime to store the CPU time of each of the running processes. Doing this change will let us keep track of the CPU time a process has taken.
- proc.c: In this file we are modifying the allocproc function. This function creates a new process. We are initializing our cputime field to zero for every new created process to make sure that the cputime tracking time starts from 0 when the process is created.
- trap.c: In this file we are modifying the kerneltrap() and usertrap() functions by adding a line of code that increments cputime. This is change is made in order to increment the cputime when a process is consuming CPU time.

What I learned from this task: I learned about kernel data structures and how to modify them, such as struct proc. I also gained an understanding of the purpose of various files and some of their functions, as well as how they operate.

Difficulties I ran into: I had trouble trying to find where to insert my code in some of the files and functions. This made me make some errors and the way I was able to overcome this problem was by comparing it to the other files in the user directory.

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

Files changed:

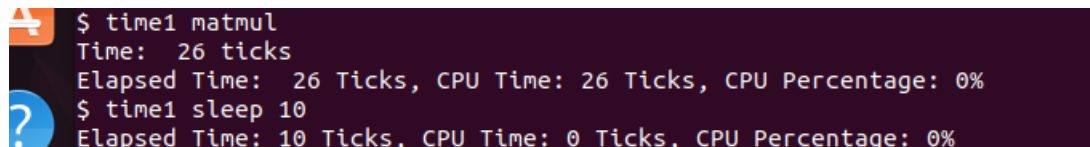
- pstat.h: Created a new header file in which we define the struct rusage, which is meant to store resource usage information for processes.
- user.h: We add struct rusage Declaration to the code in order to allow the code to use struct rusage.
- usys.pl: To specify the system call number for wait2() so that it can be used by user programs to the script.
- syscall.h and syscall.c: To make wait2() known to the kernel so that it can handle system calls.
- sysproc.c: Implementing the actual functionality of the wait2() system call.
- proc.c: To make the wait2() system call visible outside of the proc.c file

What I learned from this task: I learned how to add and modify additional features in the xv6 operating system, gaining valuable experience in kernel modification. I also grasped the concept of waiting for child processes to complete and collecting essential information. Additionally, I acquired the skills to track CPU time usage, which is crucial for understanding process performance.

Difficulties I ran into: I ran into a problem in which I got an error when running make qemu. The error was how procinfo wasn't being used and wasn't know. I had never changed any relating to that and to solve it I had to go back to the files I modified and fix some typos and code placement.

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.

Results of time command:



```
$ time1 matmul
Time: 26 ticks
Elapsed Time: 26 Ticks, CPU Time: 26 Ticks, CPU Percentage: 0%
$ time1 sleep 10
Elapsed Time: 10 Ticks, CPU Time: 0 Ticks, CPU Percentage: 0%
```

Some of the limitations of the time command that was made in this Homework would be that we are limiting ourselves to a single CPU usage. Most computers now can run programs with multi-CPU systems. Another limitation would be that our command can't measure CPU time and usage for programs that can be run in parallel exactions.