CS3224 Midterm
Intro to Operating Systems

Prof. G. Sandoval

April 6, 2020

Question	Points	Score
1	40	
2	30	
3	10	
4	10	
5	10	
Extra credits	10	
Total	110	

Exam Rules:

- The exam must represent your work alone. You may not misrepresent someone else's work as yours. You may not submit work of which you are not the sole author.
- While working on the exam you may not communicate with any other student for any reason. This prohibition includes the use of electronic communication, such as email, texting and phone. As an exception to this rule, if you have questions during the exam, you may ask the instructors or TAs. You may not discuss the content of the exam with other students. You may **not** ask questions on piazza. Email TA's or the professor directly.
- Do not copy or distribute the exam document or your answers at any time. After you have finished and have pushed your exam to Github, delete the exam and your answers from your computer.
- If we suspect any academic dishonesty, the professor will make a Zoom appointment with you right after the exam. You will need to support your answers.

Upon completion of the exam, fill in this pledge of academic honesty in the exam.txt file:

I, <your name>, affirm that I have completed the exam completely on my own, without consulting with other classmates. I have followed the required rules. I understand that violating these rules would represent academic dishonesty.

Part 0:

Setup Same as the last homework assignment we will be using the Anubis autograder, and github for submitting work. You may push to github as often as you would like until the deadline. Use the Anubis website at nyu.cool to view the status of your submissions. As before, you will need to enter the commit hash to see your submission. After pushing to github use git rev-parse master to get your commit hash. Please use this link to get your repo:

https://classroom.github.com/a/8meyf1sx

There is a file called exam.txt where you should record all your written responses (questions 4 through 6) along with your honesty pledge.

Note:

- You will ***not*** need to create any new files for this assignment.
- Do ***not*** modify the Makefile.

1. (40 points**) XV6.**

Write a program that prints the first 10 lines of its input. If a filename is provided on the command line (i.e., `head FILE`) then `head` should open it, read and print the first 10 lines, and then close it. If no filename is provided, `head` should read from standard input.

```
$ head README.md
# VSCode Integration

## Compiling
If you are using the class vm, to compile, just run `make xv6.img` in the
project directory. That will build xv6.img. Assuimging that was
successful, you can then run xv6 by running `make qemu`. If you make
changes to any xv6, you will likely need to first clean out the "stale"
binaries before rebuilding xv6.img. You can clean your build environment
with `make clean`.

## Debugging
You will then want to navigate to
run `make clean xv6.img qemu-vscode` once the gdb server has started, it
will wait for connections. You can then
navigate to your debug console in VSCode and select gdb from the gear
icon. You should see an "Attach to QEMU" profile
available. When you run this, you should connect to the gdb server. Go
ahead and try to set up breakpoints and whatnot.
```

You should also be able to invoke it without a file, and have it read from standard input. For example, you can use a *pipe* to direct the output of another xv6 command into `head`:

```
$ grep the README.md | head
If you are using the class vm, to compile, just run `make xv6.img` in the
project directory. That will build xv6.img. Assuimging that was
successful, you can then run xv6 by running `make qemu`. If you make
changes to any xv6, you will likely need to first clean out the "stale"
binaries before rebuilding xv6.img. You can clean your build environment
with `make clean`.
You will then want to navigate to
run `make clean xv6.img qemu-vscode` once the gdb server has started, it
will wait for connections. You can then
```

```
navigate to your debug console in VSCode and select gdb from the gear
icon. You should see an "Attach to QEMU" profile
available. When you run this, you should connect to the gdb server. Go
ahead and try to set up breakpoints and whatnot.
## Dependencies (if you are not on the VM)
If you are compiling natively, will need to add a plugin to your vscode.
To install the necessary dependencies for debugging in vscode, press
`ctrl p` then enter `ext install webfreak.debug`.
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
xv6 borrows code from the following sources:
JOS (asm.h, elf.h, mmu.h, bootasm.S, ide.c, console.c, and others)
```

The above command searches for all instances of the word `the` in the file `README`, and then prints the first 10 matching lines.

Extra Credit: Extending `head` (5 points)

The traditional UNIX `head` utility can print out a configurable number of lines from the start of a file. Implement this behavior in your version of `head`. The number of lines to be printed should be specified via a command line argument as `head -NUM FILE`, for example `head -3 README` to print the first 3 lines of the file README. The expected output of that command is:

```
$ head -3 README.md
# VSCode Integration

## Compiling
```

If the number of lines is not given (i.e., if the first argument does not start with `-`), the number of lines to be printed should default to 10 as in the previous part.

****Hints**:**

a) You can convert a string to an integer with the `atoi` function.
b) You may want to use *pointer arithmetic* (discussed in class in Lecture 2) to get a string suitable for passing to `atoi`.

2. (30 Points) **System Call**

Implement a system call in XV6. The system call will be called mynetid. The system call will need an implementation and the only thing that the implementation needs to do is print your netid with a newline. You will also need to test your new system call. Your test should be in testmynetid.c. Simply calling the new system call and having it print your netid will be sufficient.

3. (10 points) **Scheduling**. Three batch jobs, A through E, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4 and 8 minutes. Their priorities are 3, 5, 2, 1 and 4 respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean turnaround time. Ignore any process switching overhead.

a) Round Robin
b) Priority Scheduling
c) First Come, first served (run in order A, B, C, D, E)
d) Shortest job first.

4. (10 points) **Assembly**. Take the following recursive assembly function. The mul instruction may be new to you. You can read about where it reads and stores values here.

```
mystery:
  push %ebp
  mov %esp, %ebp
  sub $0x04,  %esp

  mov 8(%ebp), %eax
  mov %eax, -4(%ebp)
  cmp $1, %eax
  jle .base

  dec %eax
  push %eax

  call mystery
  add $4, %esp
  mov -4(%ebp), %ebx
  mul %ebx

  jmp .end

.base:
  mov $1, %eax

.end:
  mov %ebp, %esp
  pop %ebp
  ret
```

a) What would mystery(3) be?
b) How many local variables are used in this function? (hint: how much stack space is used?)
c) Translate this function into an equivalent recursive python function.
d) This function has a specific name. What is that name?

5. (10 points) Read the following code snippet and then answer the questions below:

```
void main()
{
            fork();
            fork();
            exit();
}
```

a) How many child processes are created when this program is executed? Explain your answer.
b) Are there any zombies created? Explain your answer.

6. (5 points) How could we make the ideas on this course easier to understand ?

_____

_____

(there is no wrong answer here except a blank answer 😊😊)