COMP8051 – Operating System Engineering

Completion Date: 6th March 2023

Value: 15 marks

On completion please zip up your files and upload to Canvas.

On your laptop you may need to install qemu-system-i386 e.g. on Ubuntu:

sudo apt-get install qemu-system-i386

Clone the xv6 operating system from here using git.

git clone git://github.com/mit-pdos/xv6-public.git

run **make** to build xv6

and

make qemu

to run xv6

Execute some shell commands to get familiar with the user part of the OS. The shell commands are separate programs e.g. ls.c, cat.c.

To get a feel for how programs look in xv6, and how various APIs should be called, you can look at the source code for other utilities: echo.c, cat.c, wc.c, ls.c.

Hints for the questions below:

In places where something asks for a file descriptor, you can use either an actual file descriptor (i.e., the return value of the open function), or one of the standard I/O descriptors: 0 is "standard input", 1 is "standard output", and 2 is "standard error". Writing to either 1 or 2 will result in something being printed to the screen.

The standard header files used by xv6 programs are "types.h" (to define some standard data types) and "user.h" (to declare some common functions). You can look at these files to see what code they contain and what functions they defined.

Beware that programs don't have access to the typical stdio library that you expect to find on most systems. You'll have many of the functions you expect but some of the behavior might be different. For example, *printf* accepts an initial parameter that is the output stream: 1 represents the standard output and 2 represents the standard error stream. There is no FILE* type and no *fopen*, *fclose*, *fgets*, etc. calls. Look through **usertests.c** for examples on how all of the system calls provided with xv6 are used.

Question 1: hello world in xv6

Write a program for xv6 that, prints "Hello world" to the xv6 console. This can be broken up into a few steps:

1. Create the file hello.c in the xv6 directory

2. Edit the file Makefile, find the section UPROGS (which contains a list of programs to be built), and add a line to tell it to build your hello.c code. When you're done that portion of the Makefile should look like:

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _hello\
```

3. Run **make** to build xv6, including your new program

4. Run **make qemu** to launch xv6, and then type hello in the QEMU window. You should see "Hello world" be printed out.

(1 mark)

Question 2: head command

Write a program that prints the first 10 lines of its input for the xv6 operating system. 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.

See how the program cat.c works e.g

cat README

grep the README | head

should show ten lines each line has the word the in it.

Many aspects of this are similar to the wc program: both can read from standard input if no arguments are passed or read from a file if one is given on the command line. Reading its code will help you if you get stuck.

(2 marks)

Question 3: Trace

Add a new system call called *trace*. Its syntax is

```
int trace(int)
```

When called with a non-zero parameter, e.g., *trace(1)*, system call tracing is turned on for that process. Each system call from that process will be printed to the console in a user-friendly format showing:

- the process ID
- the process name
- the system call number
- the system call name

Any other processes will not have their system calls printed unless they also call *trace(1)*.

Calling *trace(0)* turns tracing off for that process. System calls will no longer be printed to the console.

In all cases, the *trace* system call also returns the total number of system calls that the process has made. Hence, you can write code such as:

```
printf("total system calls so far = %d\n", trace(0));
```

The system call counting for each system call on a per-process basis. You will need to keep track of this in the process control block, the **proc** structure.

Write a test program (try.c) to test your trace system call.

(4 marks)

Question 4: system calls and interrupts

Read the xv6 book, vector.S, trapasm.S, trap.c, and syscall.c and briefly describe how xv6 manages interrupts and system calls.

(4 marks)

Question 5: find

Write a simple version of the Linux `find` program: find all the files in a directory tree whose name matches a string.

Some hints:

- Look at `ls.c` to see how to read directories.
- Don't try to find any files in the "." and ".." directories.
- Changes to the file system persist across runs of QEMU; to get a clean file system run `make qemu`.

Your solution is correct if produces the following output (when the file system contains a file `a/b`):

```
$ make qemu
...
init: starting sh
$ echo > b
$ mkdir a
$ echo > a/b
$ find . b
./b
./a/b
$
```

(4 marks)