Machine Problem 0 - xv6 Setup

CSIE3310 - Operating Systems
National Taiwan University

Total Points:	100
Release Date:	February 15
Due Date:	February 28, 23:59:00
TA e-mail:	`ntuos@googlegroups.com`
TA hours:	Tue. & Thu. 13:00-14:00 before due date, CSIE Building R430 (Please knock the door)

Contents

1 Summary

xv6 is an example kernel created by MIT for pedagogical purposes. We will study xv6 to get a picture of the main concepts of operating systems. The reference book of xv6 is xv6: a simple, Unix-like teaching operating system. You will learn to set up the environment for xv6 and develop an xv6 version `tree` command in this MP.

2 Docker Installation

If you have installed Docker in an Unix environment, you can directly go to the next step. If your operating system is Windows, we strongly sugggest you to install WSL2 and run Docker in WSL2.

2.1 Why Docker?

To run an arbitrary OS on your machine, you need virtualization. Virtualization is the process of running a virtual instance of a computer system in a layer abstracted from the actual hardware. A virtual machine (VM) is the emulated equivalent of a computer system that runs on top of another system. However, VM runs a guest OS with virtual access to host resources through a hypervisor. It generally incurs lots of overhead beyond what is being consumed by your application logic. By contrast, container runs a discrete process, taking no more memory than any other executable, making it lightweight. Docker is a platform for you to build and run with containers. We also leverage the advantage of virtualization to standardize the environment of your homework, making the problems independent from both architectures and operating systems of your machines.

2.2 Supported Platforms

Docker is available on Windows, Mac, and various Linux platforms. You can find your preferred operating system below and follow the instructions of the installation guide.

- Docker Desktop for Windows
- Docker Desktop for Mac
- Docker Engine for Ubuntu
- Docker Engine for Debian
- Docker Engine for CentOS
- Docker Engine for Fedora

We suggest you to install Docker on Linux platforms because Docker commands we provide in MPs are based on Linux platforms. We do not guarantee to answer Docker issues on other platforms.

2.3 Installation Testing

To check whether you have installed Docker successfully, run the `hello-world` image.

```
$ docker run hello-world
```

You should see something like this

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

3 Launching xv6

3.1 Launching Docker Image of MP0

1. Download the `MP0.zip` from NTUCOOL, unzip it, and enter it.

```
$ unzip MP0.zip
$ cd mp0
```

2. Pull Docker image from Docker Hub

```
$ docker pull ntuos/mp0
```

Note that the image only supports the following CPU architecture

- `x86_64` or `amd64`
- `arm64`

You can check the architecture of your CPU by running the following command

```
$ arch
```

If you are not able to use the image we provided, please send an email to the TA and let us know the output of your `arch` command.

3. Use `docker run` to start the process in a container and allocate a TTY for the container process

```
$ docker run -it -v $(pwd)/xv6:/home/os_mp0/xv6 ntuos/mp0
```

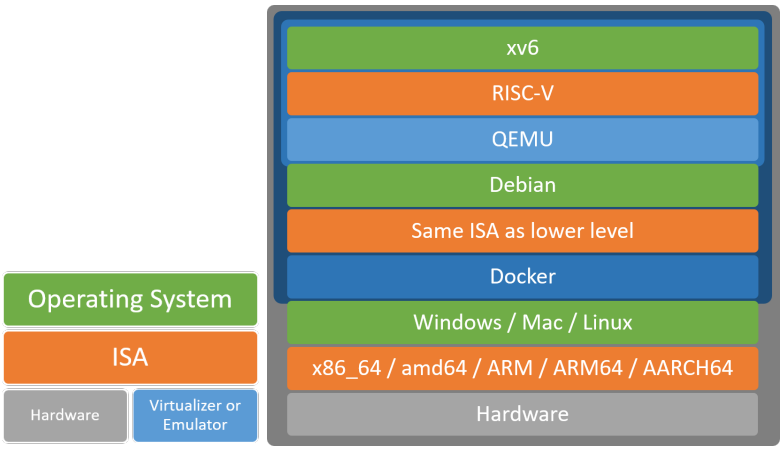
This command will make the Docker container interactive and pretend to be a shell. You may notice that we mount a volume with `-v`. Volumes are often a better choice than persisting data in a container's writable layer because a volume does not increase the size of the containers using it, and the volume's contents exist outside the life-cycle of a given container. We adopt it here to give you the flexibility to do coding outside the container with your favorite editor rather than coding with vim inside. Of course, you can use vim if you really love it.

4. Please check the environment in the Docker container

```
$ cat /etc/os-release
```

3.2 QEMU Introduction

xv6 is implemented in ANSI C for a multi-core RISC-V system, but most of our machines are not RISC-V systems. Therefore, we need QEMU to help us launch xv6 on non-RSIC-V architecture. QEMU is an emulator and virtualizer that can perform hardware virtualization. An instruction set architecture (ISA) is an abstract model of a computer, also referred to as computer architecture. The ISA serves as the interface between software and hardware.



3.3 Launching QEMU

If you successfully run the Docker container, you should be in the `xv6` directory. In the `xv6` directory, build and run xv6 on QEMU by running `make qemu`. The behavior of `make qemu` is defined by the `Makefile` we provided for you.

```
$ make qemu
...

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$
```

If you type `ls` at the prompt, you should see an output similar to the following

```
$ ls
.              1 1 1024
..             1 1 1024
README         2 2 2059
cat            2 3 24096
echo           2 4 22920
grep           2 5 27400
init           2 6 23656
kill           2 7 22880
ln             2 8 22720
ls             2 9 26288
mkdir          2 10 23016
rm             2 11 23008
sh             2 12 41832
stressfs       2 13 23880
grind          2 14 37984
wc             2 15 25208
zombie         2 16 22256
testgen        2 17 25664
console        3 18 0
```

These are the files that `mkfs` includes in the initial file system; most are executable programs. Now you have successfully set up xv6. You are welcome to play around in preparation for future MPs. Here are some useful commands for you

- `Ctrl-p`: Print information about each process. If you try it now, you will see two lines: one for `init` and the other for `sh`.
- `Ctrl-a x`: Quit QEMU. Note that you should hold `Ctrl` and press `a`, release both `Ctrl` and `a`, and then press `x` to quit.

4 Example MP - xv6 Version Tree Command

Below we provide an easy example MP. If you find it hard, you should think twice before taking this course or work harder. We assume

- You are good at coding with C.
- You are familiar with System Programming.
- You are willing to learn the competency of looking up reference book and tracing source code.

The ability to google the solution on Stack Overflow will not make you escape from this. If you feel disinclined for learning them, you may also feel terrible while doing MPs.

4.1 `tree`

`tree` is a command used to list contents of directories in a tree-like format. In this MP, we will write a program to mimic the behavior of the tree command.

4.2 `tree` Executable

To develop a custom xv6 command, you need to modify `Makefile` so that the system will build the executable of `tree` after you run `make qemu`.

4.3 Output Format

Assume that there is a directory `os2022` with the following structure

- three sub-directories: `d1`, `d2`, and `d3`.
- `d2` has two sub-directories `a`, `b`, and one file `c`.
- `d3` has one sub-directory `a` and one file `b`

```
os2022/
|
+-- d1
|
+-- d2
|   |
|   +-- a
|   |
|   +-- b
|   |
|   +-- c
|
+-- d3
    |
    +-- a
    |
    +-- b
```

To clarify the output format, we need to define some terminologies first.

- `root_directory`: the directory passed to the `tree` command. E.g., if the command is

  ```
  $ tree os2022/
  ```

  the `root_directory` is `os2022`.

- `parent_directory`: a directory above another directory/file. The table below shows the `parent_directory` of each directory/file.

directory/file	`parent_directory`
`os2022`	N/A
`os2022/d1`, `os2022/d2`, `os2022/d3`	`os2022/`
`os2022/d2/a`, `os2022/d2/b`, `os2022/d2/c`	`os2022/d2`
`os2022/d3/a`, `os2022/d3/b`	`os2022/d3`

- `level`: the `level` of the `root_directory` is 0. For other files and directories, the `level` is equal to `level(parent_directory)`+1. The `level` of each directory/file is shown in the table below.

directory/file	`level`
`os2022`	0
`os2022/d1`, `os2022/d2`, `os2022/d3`	1
`os2022/d2/a`, `os2022/d2/b`, `os2022/d2/c`, `os2022/d3/a`, `os2022/d3/b`	2

- `ancestor`: Recursively find the `parent_directory` until we reach the `root_directory`. All traversed directories are the `ancestor`(s). We refer to a specific ancestor by saying `the ancestor at level k`.
- `index`: the index of a directory/file is the rank when we `ls <parent_directory>`, note that `.` and `..` should be ignored. E.g., if the output of `ls os2022/` is:

```
$ ls os2022/
.               1 20 80
..              1 1 1024
d1              1 21 32
d2              1 22 80
d3              1 23 32
```

  the `index` of `os2022/d1`, `os2022/d2`, `os2022/d3` are 0, 1, 2, respectively. The directory/file with the largest `index` is called `the last directory/file of <parent_directory>`. E.g., `os2022/d3` is `the last directory/file of os2022/`.

Now, we can define the output format. You must traverse all files and directories from the `root_directory` and follow the output format to print the tree structures.

1. First, you must print the directory passed to the tree command. If the path passed to the tree command does not exist or is not a directory, you should output the path followed by `[error opening dir]`, separated by a space. Note that if the path passed to the tree command is not a valid directory, you only have to print the first line and send the `file_num` and `dir_num` to the parent process through the pipe. Both `file_num` and `dir_num` should be set to 0.

command	first line
`tree os2022/`	`os2022/`
`tree os2022`	`os2022`
`tree os2002`	`os2002 [error opening dir]`
`os2022/d2/c`	`os2022/d2/c [error opening dir]`

2. The child process should traverse the file system in depth-first order. And the order it traverses within a directory should be identical to the order of `ls`. If we get some output from the `ls` command as shown below, the child process should start by checking `file_b` instead of `dir_a`.

```
$ ls some_dir
file_b
dir_a
```

3. Once a directory/file is traversed, the child process should print the partial tree structure. The partial tree structure is composed of two lines. The first line should contain multiple `|` (ASCII-124) separated with three spaces. The number of `|` can be up to its `level` minus 1. However, if `the ancestor at level k` is `the last sub-directory/file of the ancestor at level (k-1)`, the `(k-1)`-th `|` should be replaced with a space, where `k` > 0. We provide two examples to illustrate this rule

   (a) Assume that the child process has printed the following output:

```
os2022
|
+-- d1
|
+-- d2
|   |
|   +-- a
|   |
|   +-- b
```

   The current traversed file is `os2022/d2/c`, its `ancestor`(s) are `os2022` and `d2`. The `level` is 2, the first line can have up to two `|`

```
|   |
x012x (three spaces)
```

Since `the ancestor at level 1` is not the `last directory/file of the ancestor at level 0`, we do not have to replace any `|` with space.

(b) Assume that the child process has printed the following output

```
os2022
|
+-- d1
|
+-- d2
|   |
|   +-- a
|   |
|   +-- b
|   |
|   +-- c
|
+-- d3
```

The current traversed file is `os2022/d3/a`, its `ancestor`(s) are `os2022` and `os2022/d3`. The `level` is 2. The first line can have up to two `|`. However, `the ancestor at level 1` is `the last directory/file of the ancestor at level 0`. We have to replace the 0th `|` with space. So the output should be

```
    |
0123x (4 spaces)
```

You might have noticed that if the current traversed file is at `level` N, it will not have the `the ancestor at level N`, so the (N-1)-th `|` will never be replaced. That is, the last `|` will always be there. Note that there should not be any space after the last `|`.

4. The second line is similar to the first line. The only difference is that you have to replace the last `|` with `+-- <basename>`. `<basename>` is the string after the last `/` of the current traversed file path. We continue the examples

(a) The first line is

```
|   |
x012x (three spaces)
```

All we have to do is replace the last `|` with `+-- c`

```
|   |
|   +-- c
x012xxx3x (four spaces)
```

(b) The first line is

```
    |
0123x (four spaces)
```

By replacing the last `|` with `+-- a`, we get

```
    |
    +-- a
0123xxx4x (five spaces)
```

4.4 Steps

1. The process `tree` reads one argument from the command line.

```
$ tree <root_directory>
```

2. The process forks a child.
3. The child process has to traverse the files and directories under `<root_directory>`, output in tree-like format, and analyze the total number of traversed files and directories. Note that the testcases only contain files and directories (no devices).
4. The child process sends two integers, `file_num` and `dir_num`, to the parent process through a pipe.
   - `file_num`: the number of traversed files
   - `dir_num`: the number of traversed directories
5. The parent process reads the integers from the pipe and prints

   ```
   <dir_num> directories, <file_num> files
   ```

   `<file_num>` is the number of traversed files, and `<dir_num>` is the number of traversed directories. **The `<root_directory>` should not be count**. You should separate the child process and the parent process outputs with a blank line. You can print the blank line in either the child process or the parent process. However, if you do not use a pipe to receive `dir_num` and `file_num` and print out the result in the parent process, you will be regarded as **CHEATING** and get 0 points in this MP.

4.5 Testcase Specifications

- The `level` will be smaller than 5, the total number of files and directories will not be larger than 20.
- You only have to consider files and directories. We will not generate a testcase that contains a device.

4.6 Sample Outputs

```
# Assume that os2202 does not exist
$ tree os2202
os2202 [error opening dir]

0 directories, 0 files
```

```
$ testgen
$ tree os2022/
os2022/
|
+-- d1
|
+-- d2
|   |
|   +-- a
|   |
|   +-- b
|   |
|   +-- c
|
+-- d3
    |
    +-- a
    |
    +-- b

6 directories, 2 files
```

4.7 Public Testcases

You can get 40 points (100 points in total) if you pass all public testcases. You can judge the code by running the following command in the docker container (not in xv6; this should run in the same place as `make qemu`). Note that you should only modify `xv6/Makefile` and `xv6/user/tree.c`. We do not guarantee that you can get the public points from us if you modify other files to pass all testcases during local testing. We will run `make qemu` to compile your code, but we will not run `make grade` to grade your code.

```
$ make grade
```

If you successfully pass all the public testcases, the output should be the same as below

```
== Test tree command with public testcase 0 (5%) ==
tree command with public testcase 0: OK (10.9s)
== Test tree command with public testcase 1 (5%) ==
tree command with public testcase 1: OK (0.7s)
== Test tree command with public testcase 2 (5%) ==
tree command with public testcase 2: OK (0.7s)
== Test tree command with public testcase 3 (5%) ==
tree command with public testcase 3: OK (0.9s)
== Test tree command with public testcase 4 (10%) ==
tree command with public testcase 4: OK (1.1s)
== Test tree command with public testcase 5 (10%) ==
tree command with public testcase 5: OK (0.9s)
Score: 40/40
```

If you want to know the details about the testcases, please check `xv6/grade-mp0` and `xv6/user/testgen.c`

Hint: remember to close the file descriptor. Check `xv6/user/grind.c` and `xv6/user/ls.c` for function usage.

5 Submission

Please compress your xv6 source code as `<whatever>.zip` and upload to NTUCOOL. The filename does not matter since NTUCOOL will rename you submissions. Never compress files we do not request, such as `.o`, `.d`, `.asm` files. You can run `make clean` in the container before you compress. Make sure your xv6 can be compiled by `make qemu`.

5.1 Folder Structure after Unzip

We will unzip your submission by running `unzip *.zip`. Your folder structure **AFTER UNZIP** should be

```
<student_id>
|
+-- xv6
    |
    +-- user
    |   |
    |   +-- tree.c
    |
    +-- Makefile
```

Note that **the English letters in the `<student_id>` must be lowercase**. E.g., it should be `d08922025` instead of `D08922025`.

5.2 Grading Policy

- You will get 0 points if we cannot compile your submission.
- You will be deducted 10 points if the folder structure is wrong. Using uppercase in the `<student_id>` is also a type of wrong folder structure.
- If your submission is late for n days, your score will be $raw\_score \times (100\% - 20\% \times n)$. Note that you will not get any points if $n >= 5$

6 References

[1] xv6, a simple Unix-like teaching operating system. https://pdos.csail.mit.edu/6.828/2012/xv6.html

[2] Docker: Empowering App Development for Developers. https://docs.docker.com/

[3] RISC-V: The Free and Open RISC Instruction Set Architecture. https://riscv.org/

[4] QEMU, the FAST! processor emulator. https://www.qemu.org/