Operating Systems 2021 Spring

MP0 - Set up xv6

xv6 is an example kernel created by MIT for pedagogical purpose. We will study **xv6** to get the picture of the main concepts of operating systems.

Reference book of **xv6** is *xv6: a simple, Unix-like teaching operating system*.

In this MP, you will learn how to set up the environment for **xv6**.

1. Install Docker

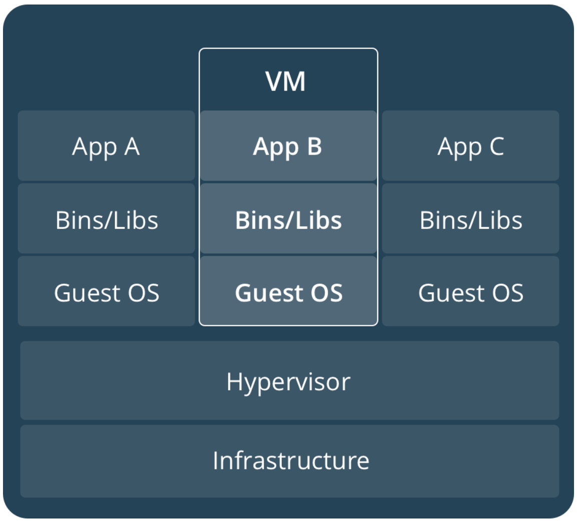
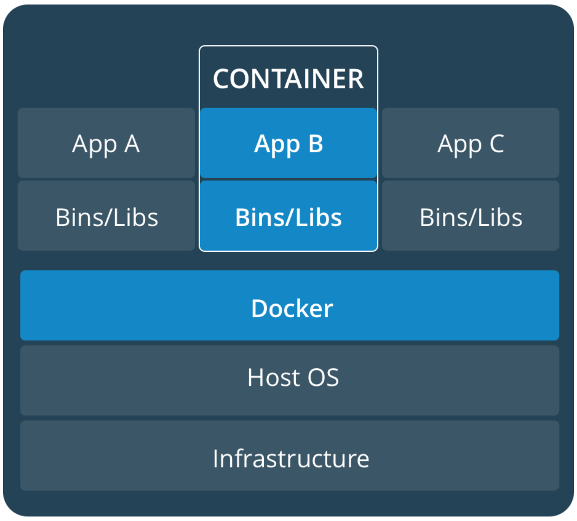
Why Docker

In order to run another 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 *full-blown* **guest OS** with virtual access to **host** resources through a **hypervisor**. In general, it 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. And **Docker** is a platform for you to build and run with containers.

Virtual Machine	Container
 VM App A App B App C Bins/Libs Bins/Libs Bins/Libs Guest OS Guest OS Guest OS Hypervisor Infrastructure	 CONTAINER App A App B App C Bins/Libs Bins/Libs Bins/Libs Docker Host OS Infrastructure

We also leverage the advantage of virtualization to *standardize* the environment of your homework, making the problems independent from both architectures and OSes of your machines.

Supported Platforms

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

Docker Desktop

Platform	x86_64 / amd64
Docker Desktop for Windows	✓
Docker Desktop for Mac	✓

Docker Engine

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64
Ubuntu	✓	✓	✓
Debian	✓	✓	✓
CentOS	✓		✓
Fedora	✓		✓

Yet, we suggest you install Docker on **Linux** platforms, because Docker commands we provide in this and future MPs will be based on Linux platforms. We do not guarantee to answer Docker's issues on other platforms. In addition, containers run *natively* on Linux.

Test Installation

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

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

You should see some informational messages.

2. Run Docker Container

1. Clone `mp0` from our GitHub repository and enter it.

   ```
   $ git clone [repository_path]
   $ cd mp0
   ```

   *Temporary Link: mp0.zip

2. Download tar archive os_mp0.tar (1.5G)

   —Update—

   Another approach
   Get the image from Docker Hub

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

   Then go to step 4.

   —Update—

   TA has set up the environment for the MP0 and packed it as a Docker image. TA will upload Docker image in each MP in similar way so that you can start your homework with ease by loading the image.

3. Load Docker image from tar archive.

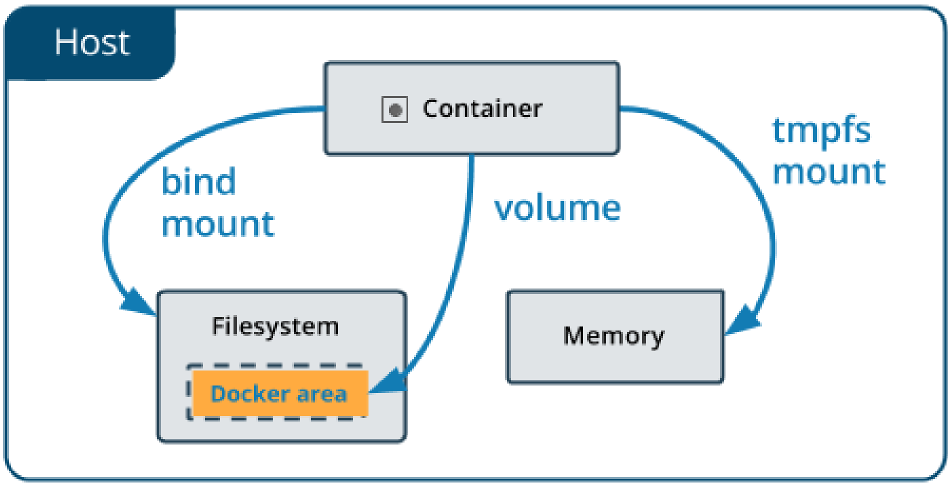
   ```
   $ docker load --input os_mp0.tar
   ```

4. **(Update)** Use `docker run` to start the process in a container and allocate a TTY for the container process.

   ```
   // Download image from Google Drive
   $ docker run -it -v $(pwd)/xv6:/home/os_mp0/xv6 os_mp0
   // Download image with docker pull
   $ docker run -it -v $(pwd)/xv6:/home/os_mp0/xv6 ntuos/mp0
   ```

   This will make Docker container be an interactive process 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 lifecycle of a given container.



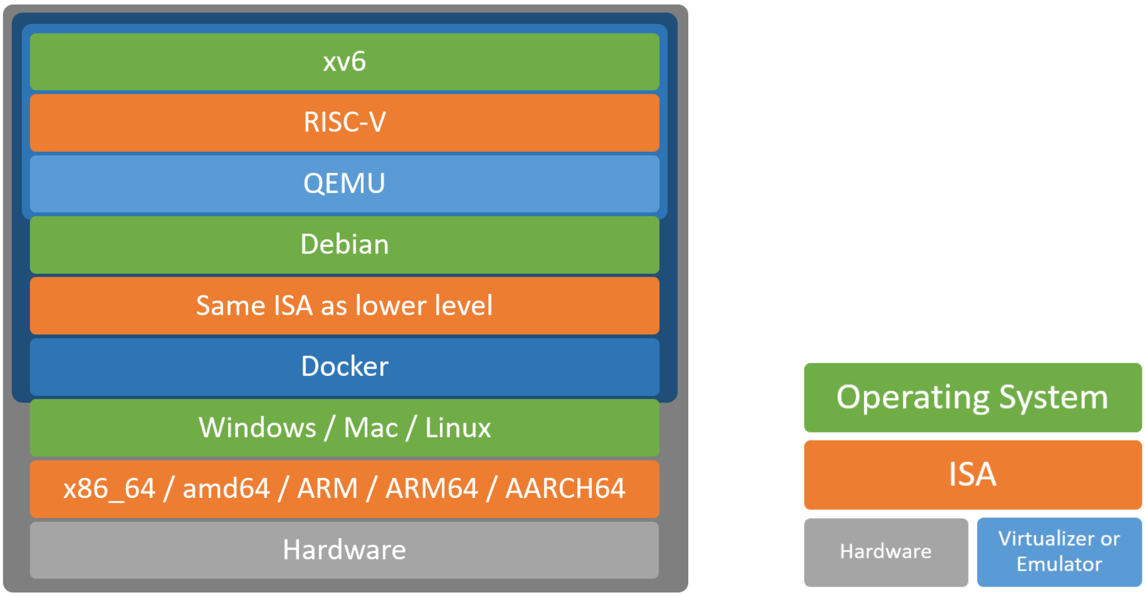
   But we adopt it here is simply to give you the flexibility that you can do coding outside the container with your favorite editor rather than coding with `vim` inside. Of course, if you are a `vim` lover, you get the privilege to be free of stepping in and out the container while debugging.

5. You should be able to check the environment in Docker container.

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

3. Launch xv6

xv6 is implemented in **ANSI C** for a multi-core **RISC-V** system, but most of our machines are not RISC-V system. Therefore, we need **QEMU** to help us launching **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. Learn more

Suppose you are in Docker container. You should see you are under `xv6` directory. In `xv6` directory, build and run **xv6** on QEMU by entering **`make qemu`** with `Makefile` we provided for you.

```
$ make qemu
riscv64-linux-gnu-gcc    -c -o kernel/entry.o kernel/entry.S
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -DSOL_UTIL -MD -mcmodel=medany
...
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_zombie user/zombie.o user/ul
riscv64-linux-gnu-objdump -S user/_zombie > user/zombie.asm
riscv64-linux-gnu-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/zombie.
mkfs/mkfs fs.img README  user/_cat user/_echo user/_grep user/_init user/_kill user/_ln user/_ls use
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
balloc: first 430 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive

xv6 kernel is booting

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

If you type `ls` at the prompt, you should see 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 27408
init           2 6 23656
kill           2 7 22888
ln             2 8 22720
ls             2 9 26288
mkdir          2 10 23016
rm             2 11 23008
sh             2 12 41840
stressfs       2 13 23880
grind          2 14 37984
wc             2 15 25208
zombie         2 16 22264
console        3 17 0
```

These are the files that `mkfs` includes in the initial file system; most are programs you can run. You just ran one of them: `ls`.

Congratulation! You have accomplished setting up **xv6**. You are welcome to play around in preparation for future MPs. Here are useful commands for you:

- **Ctrl-p**
  Print information about each process. If you try it now, you'll see two lines: one for `init`, and the other for `sh`.

- **Ctrl-a x**
  Quit QEMU.

4. Exercise

Below we provide an *easy* example MP. If you find it hard to you, you should think twice before taking this course, or should 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 of google-ing 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.

Example MP

Description

Write a program named **detective** that uses UNIX system calls to find the files in a directory tree with a specific name, and communicate the result between two processes over a pipe.

Explanation

In 1887, Europe was ringing with a name - Sherlock Holmes. His clients vary from the most powerful monarchs and governments of Europe, to wealthy aristocrats and industrialists, to impoverished pawnbrokers and governesses.

Holmes works as a detective for twenty-three years, with Watson assisting him for seventeen of those years. John Watson, Holmes’s friend, accompanies Holmes during his investigations and often shares quarters with him at the address of 221B Baker Street, London, where many of the stories begin.

“HOLMES PULLED OUT HIS WATCH.”

Now, you are disguising Conan Doyle by creating the legendary private detective in an operating system way. Here is the revised version for you to mimic how Holmes and Watson receive a commission from the client and perform the prelude to their forensics.

1. The detective accepts the commission from the client.
2. Holmes appoints the investigation job to Watson.

3. Watson looks for clues around the crime scene.
4. Watson reports the observation to Holmes.
5. Holmes smokes cigars (In fact he smokes pipes, but we try not confusing you here) and starts his deduction.

Your fiction in **xv6** world is:

1. The process `detective` reads one argument from command line.

   ```
   $ detective [commission]
   ```

2. The process forks a child.

3. The child finds the files below current working directory with a specific name (fully match), and prints **all** matches files.

4. The child writes a character on the pipe to the parent. The character `y` and `n` imply whether the child obtains files or not, respectively:

   ```
   /* Example 1 */
   $ detective hello
   // fork child
   <pid> as Watson: ./hello           // printed by child
   <pid> as Watson: ./some_dir/hello  // printed by child
   // child writes 'y' on the pipe
   /* Example 2 */
   $ detective world
   // fork child
   // child finds no matched file
   // child writes 'n' on the pipe
   ```

   where `<pid>` is child's process ID. Then the child exits.

   Note: The child should traverse the file system in a depth-first order. And the order it traverses within a directory should be identical to the order of `ls`. That is, if we get some output from `ls` command like:

   ```
   $ ls some_dir
   file_b
   dir_a
   ```

   The child should start from checking `file_b` instead of `dir_a`.

5. The parent reads the character from the pipe and print:

   ```
   /* Upon receiving 'y' */
   <pid> as Holmes: This is the evidence  // printed by parent
   /* Upon receiving 'n' */
   <pid> as Holmes: This is the alibi     // printed by parent
   ```

   where `<pid>` is parent's process ID and alibi stands for 不在場證明 in Mandarin. Then the parent exits.

Hint

- Read Chapter 1 of the **xv6** book.
- Use `fork` to create a child.
- Look at `user/ls.c` to see how to read directories.
- Use recursion to allow find to descend into sub-directories, and don't recurse into `.` and `..`.
- **(Update)** You can assume `[commission]` will never be the symbolic link.
- You'll need to manipulate C strings, such as using `strcmp()`.
- Use `pipe` to create a pipe.
- Use `read` to read from the pipe, and `write` to write to the pipe.
- Use `getpid` to find the process ID of the calling process.
- Make sure `main` calls `exit()` in order to exit your program.
- User programs on **xv6** have a limited set of library functions available to them. You can see the list in `user/user.h`. The source (other than for system calls) is in `user/ulib.c`, `user/printf.c` and `user/umalloc.c`.

Sample Execution

Your solution should be in the file `user/detective.c`.

Once you've done, add your **detective** program to `UPROGS` in `Makefile` and compile the kernel with `make qemu`.

Run the program from the **xv6** shell and it should produce the following output:

```
$ make qemu
...
init: starting sh
$ detective cat
4 as Watson: ./cat
3 as Holmes: This is the evidence
$ detective dog
5 as Holmes: This is the alibi
$
```

Submission

Push your **xv6** source code to GitHub. Never push any other we do not request, such as `.o`, `.d`, `.asm` files. You can run `make clean` in container before you push. Make sure your **xv6** can be compiled.

Your final repository should look like following hierarchy:

```
Repository
└── xv6
    ├── user
    │   ├── detective.c
    │   └── ...
    ├── Makefile
    └── ...
```

- You will get 0 if **xv6** cannot be compiled.
- We might give discount on your grade if your format is wrong, such as pushing `.o` files.

Reference

- xv6, a simple Unix-like teaching operating system
  https://pdos.csail.mit.edu/6.828/2012/xv6.html
- Docker: Empowering App Development for Developers
  https://docs.docker.com/
- RISC-V: The Free and Open RISC Instruction Set Architecture
  https://riscv.org/
- QEMU, the FAST! processor emulator
  https://www.qemu.org/

This page is maintained by KaoShengChieh.