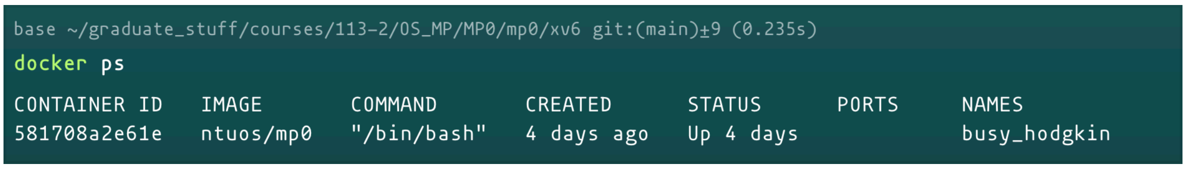
MP0

Baic info

Starting / exiting the container

In the xv6 directory, use the following command to check if a container is running:

```
docker ps
```



```
base ~/graduate_stuff/courses/113-2/OS_MP/MP0/mp0/xv6 git:(main)±9 (0.235s)
docker ps
CONTAINER ID   IMAGE       COMMAND       CREATED       STATUS      PORTS     NAMES
581708a2e61e   ntuos/mp0   "/bin/bash"   4 days ago    Up 4 days             busy_hodgkin
```

- If the docker is running

Ex:

```
docker exec -it busy_hodgkin /bin/bash
```

- -it: Interactive + TTY mode (keep the terminal open)
- busy_hodgkin: name of the container (Can ise the container ID got from "docker ps"
- /bin/bash: run bash shell in the container

Note:
1. docker run: create new container
2. docker exec: enter an already running container

- If the docker is not running

Start then exec:

```
docker start busy_hodgkin
docker exec -it busy_hodgkin /bin/bash
```

- Exit but keep the container running

Exit the container shell but keep running in the background:

```
exit
```

Compile and run

After finished editing the mp0.c file, run:

```
make clean
make qemu
```

The first command is to remove the old compiled files.
The second command is to compile all user programs (including mp0.c)

After running `make qemu`, check if mp0 is shown by running:

```
ls
```

Then we can try running:

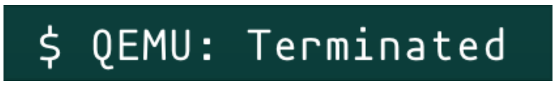
```
mp0 /some/path key
```

Recompile after mp0.c is modified

After modifying the mp0.c file, use:

```
Ctrl  + a
```
then
```
x
```

to exit qemu, and will show as the following image:



```
$ QEMU: Terminated
```

Then run:

```
make clean
make qemu
```

C: struct

stat

```
struct stat {
    int dev;    // Device number
    uint ino;   // Inode number
    short type; // File type (T_DIR for directory,
T_FILE for file)
    short nlink;// Number of hard links
    uint size;  // Size in bytes
};
```

dirent

```
struct dirent {
    ushort inum;    // Inode number (0 if unused)
    char name[DIRSIZ]; // Name of file/directory
};
```

- dirent = directory entry

System call

open()

```
open(path, mode)
```
>> opens a file or directory
>> returns a int which is **fd** (file descriptor)

- mode:
  - 0: Read-only
  - 1: Write-only
  - 2: Read-write

Function

fstat

`fstat(fd, &st)`
>> fills the st struct with the metadata of fd

- we use **&**st (a ptr to st) to directly modify the original structure (instead of copying the structure to the function << convention of C when passing a struct to a function)

<u>syntax</u>

```
int fstat(int fd, struct stat *st);
```
- fd: file descriptor
- st: ptr to a sturct stat where the metadata is stored

>> return:
- 0: success
- -1: fail