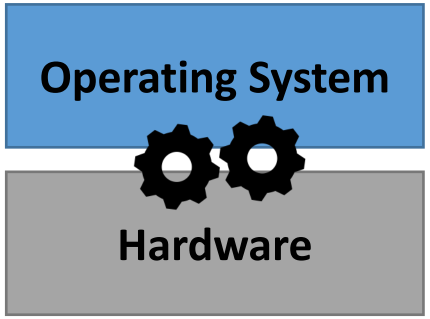
CS 1550

Lab 1 – xv6 Introduction

Setup and exercise

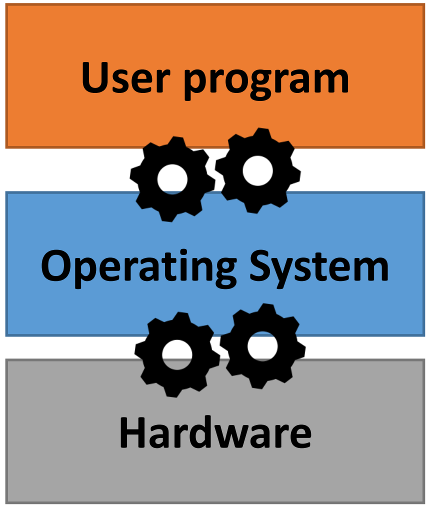
CS 1550 – Kernel Space vs User Space

- OS manages hardware, services and user processes
  - CPU
  - Memory (Address space)
  - I/O devices (Disk, mouse, video card, sound, network, etc.)



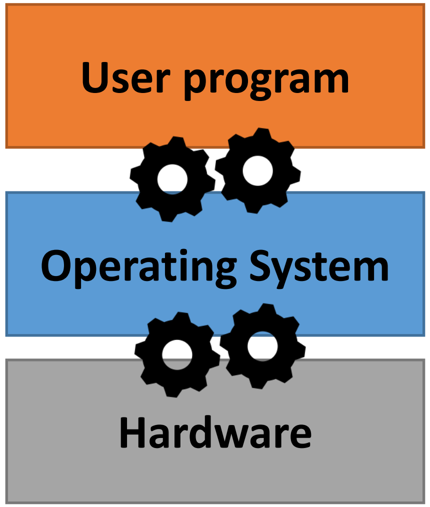
CS 1550 – Kernel Space vs User Space

- OS manages hardware, services and user processes
  - CPU
  - Memory (Address space)
  - I/O devices (Disk, mouse, video card, sound, network, etc.)



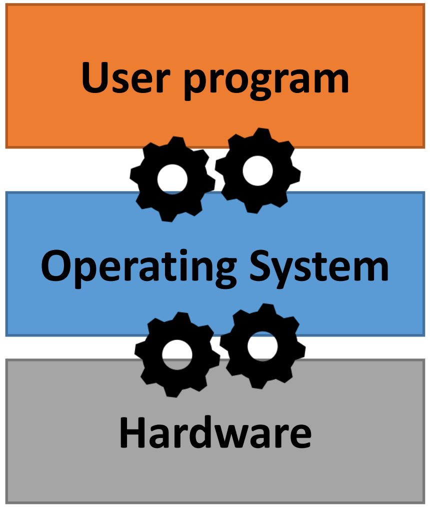
CS 1550 – Kernel Space vs User Space

- OS **is just** another **software**



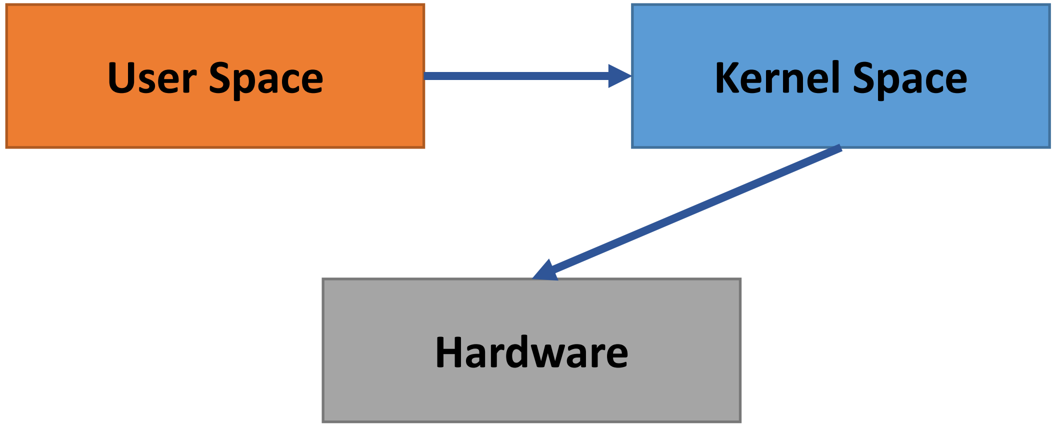
CS 1550 – Kernel Space vs User Space

- OS **is just** another **software**
- User applications should not change the kernel(OS software)



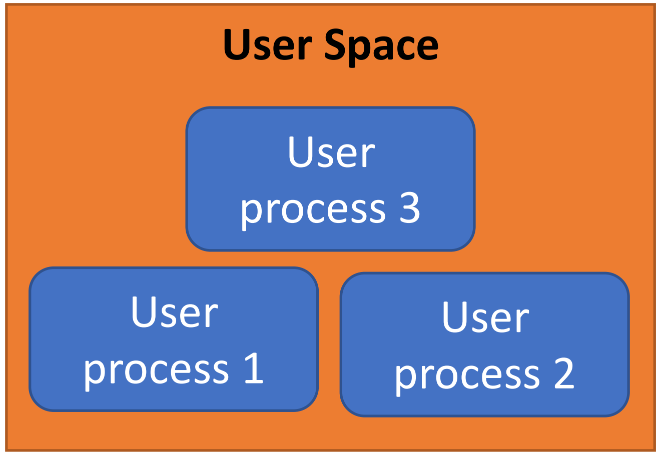
CS 1550 – Kernel Space vs User Space

- User space
  - **Less privileged memory space** where user processes execute
- Kernel space
  - **Privileged memory space** where the OS main process resides
  - **No User application should be able to change**



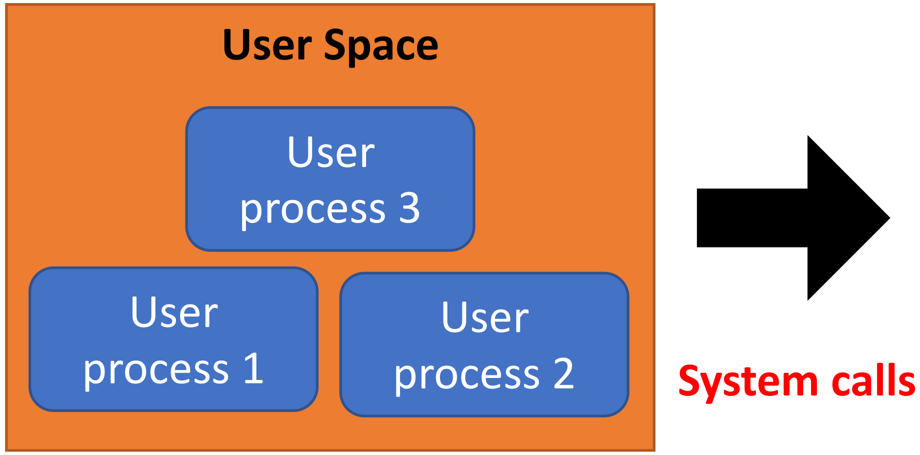
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes have to do system calls to access the OS resources and Hardware



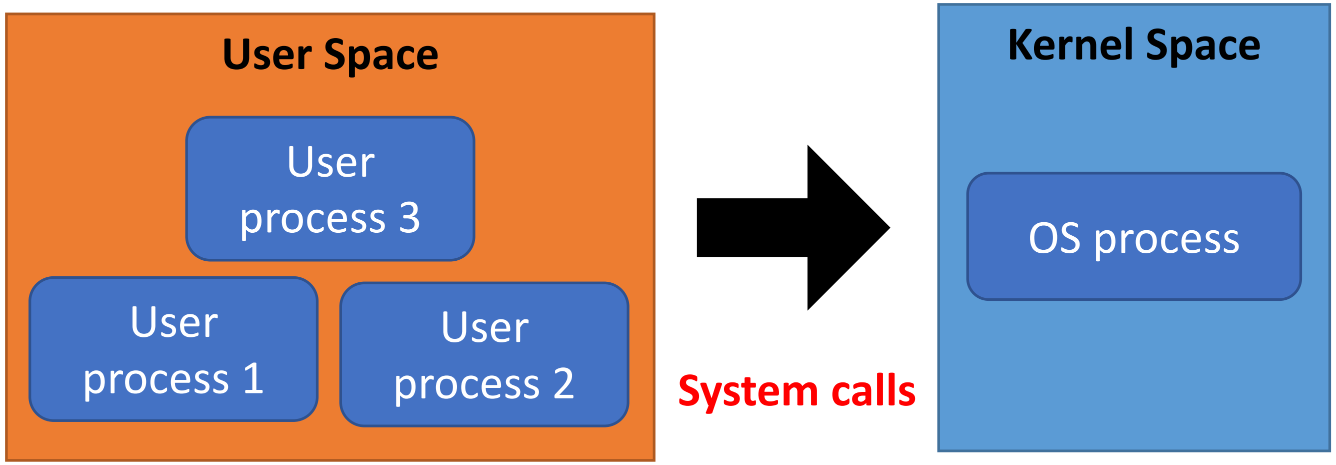
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes have to do system calls to access the OS resources and Hardware



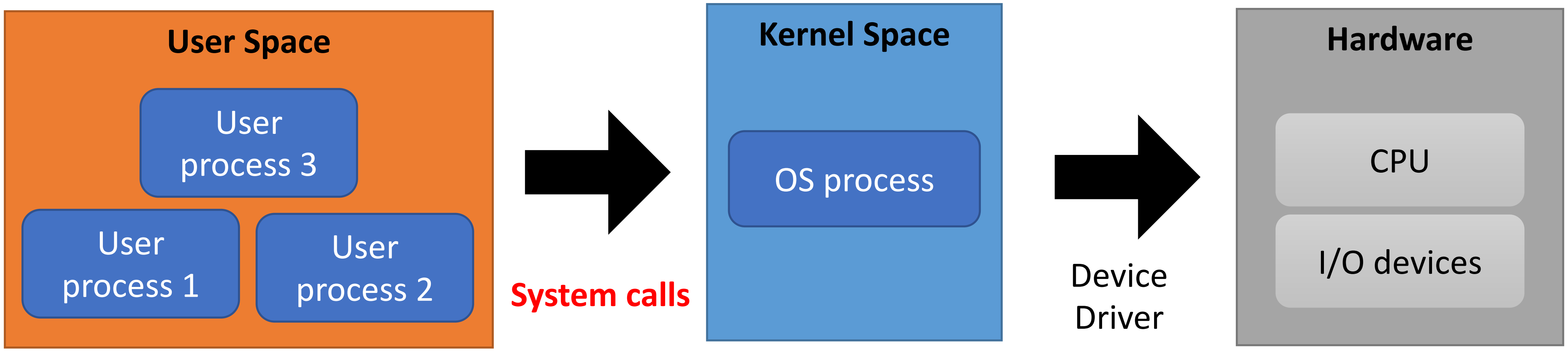
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes have to do system calls to access the OS resources and Hardware



CS 1550 – Kernel Space vs User Space

- **System Call (OS function)**
  - User processes have to do system calls to access the OS resources and Hardware



System Call
-
exercise

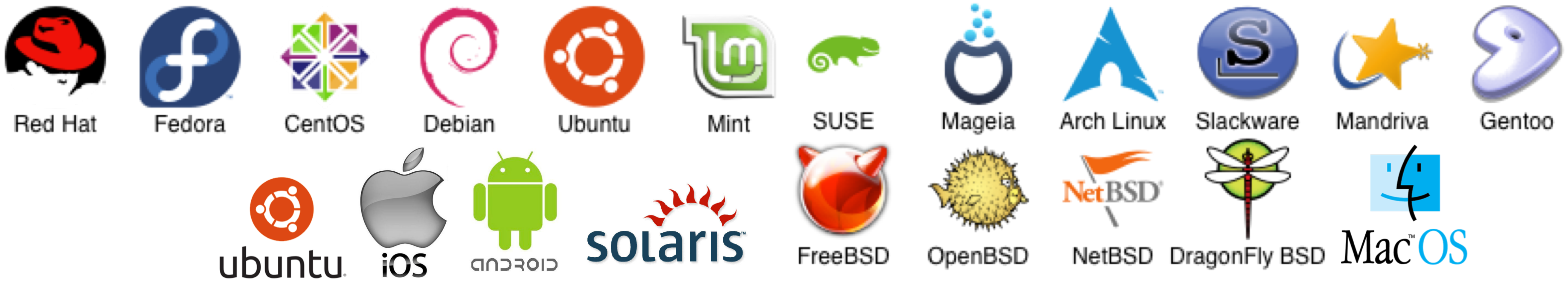
CS 1550 – xv6

- Simple Unix-like teaching **operating system** from MIT
    - Provides basic services to running programs

xv6

CS 1550 – Unix is everywhere

- Most operating systems are based on Linux

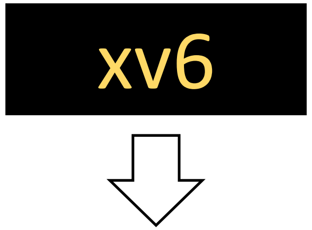


CS 1550 – xv6

- Simple Unix-like teaching operating system from MIT
  - Has a **subset of traditional** system calls
    - **fork**() Create process
    - **exit**() Terminate current process
    - **wait**() Wait for a child process
    - **kill**(pid) Terminate process pid
    - **getpid**() Return current process's id sleep(n)
    - **Sleep** for n time units exec(filename, *argv)
    - **Load** a file and execute it sbrk(n)
    - ….

CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server
    - Since it is an OS how can we run it?



xv6

Run where?

CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

xv6

Run where?

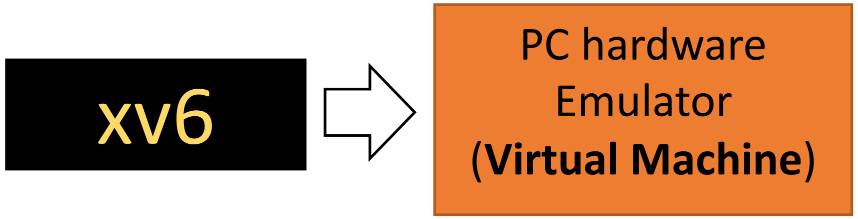
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

xv6

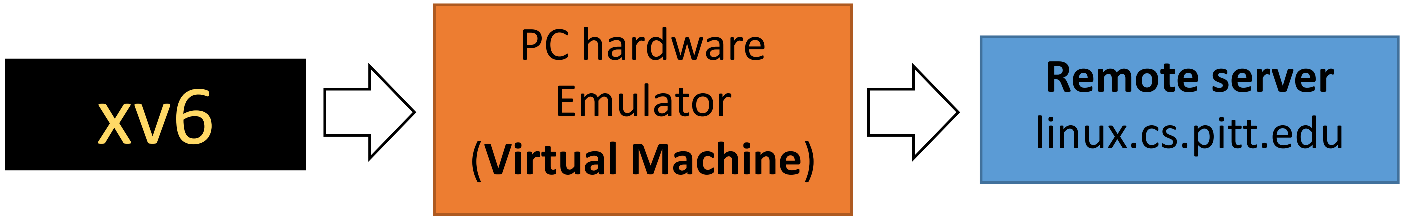
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



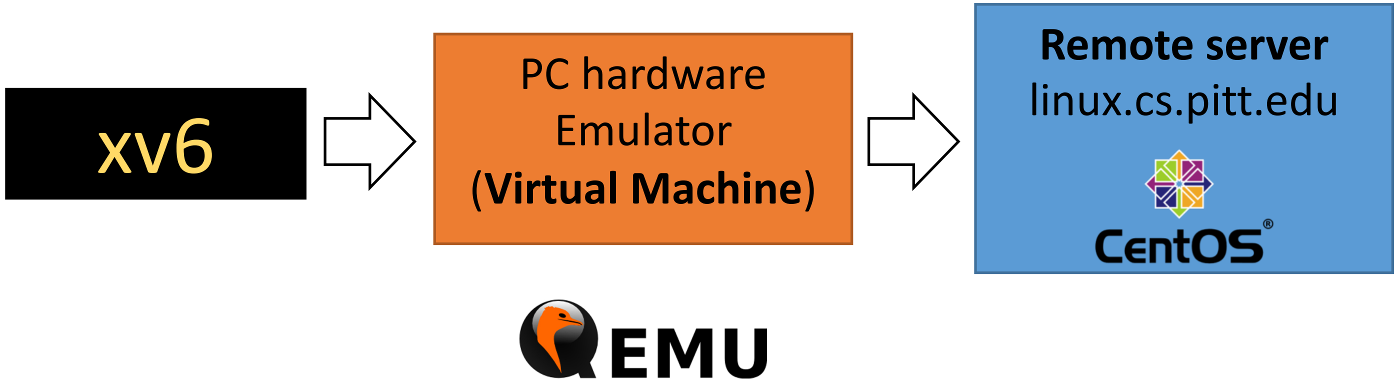
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



xv6

PC hardware Emulator **(Virtual Machine)**

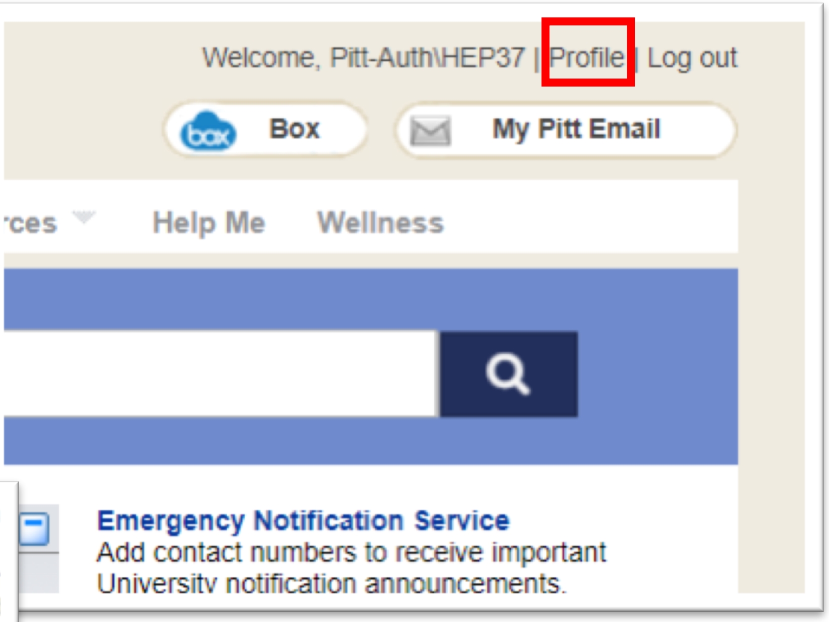
Remote server linux.cs.pitt.edu

CentOS

QEMU

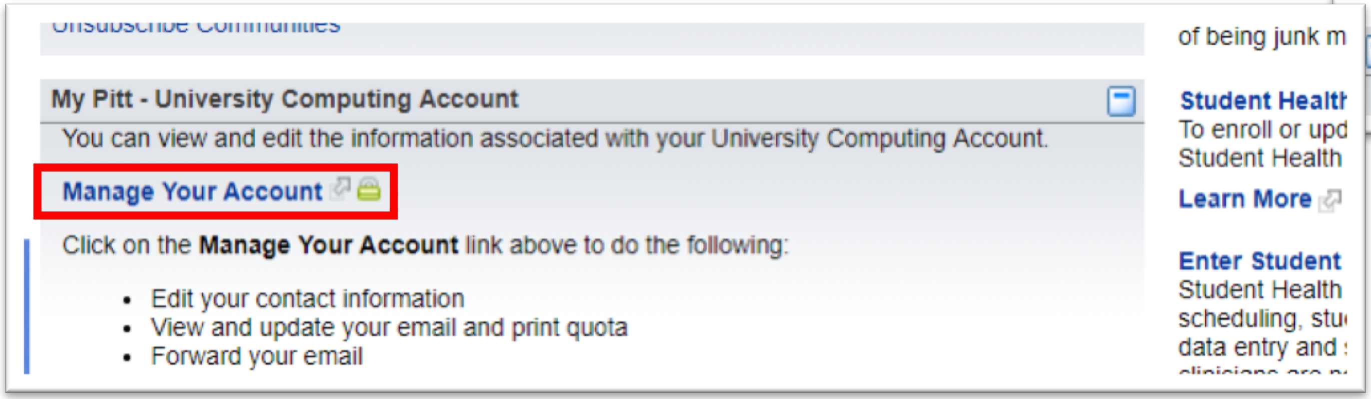
CS 1550 – Compile and Run xv6

1. Extend disk Quota, if you have less then 500mb free space
   a) Log in to https://my.pitt.edu
   b) Click on "Profile" at the top of the screen
   c) Click on "Manage Your Account"
   d) Click on "Manage Email Quota"
   e) Click on "Increase My UNIX Quota"



Welcome, Pitt-Auth\HEP37 | Profile | Log out

Emergency Notification Service
Add contact numbers to receive important University notification announcements.



Unsubscribe Communities

My Pitt - University Computing Account

You can view and edit the information associated with your University Computing Account.

Manage Your Account

Click on the **Manage Your Account** link above to do the following:

- Edit your contact information
- View and update your email and print quota
- Forward your email

of being junk m

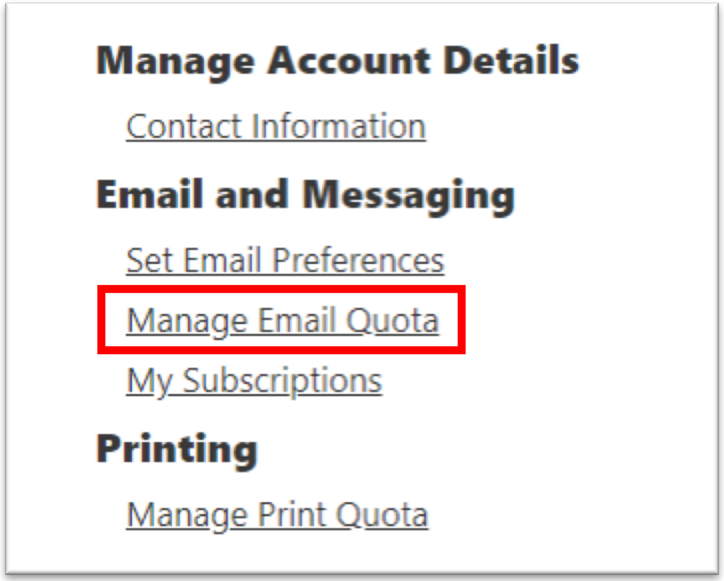
Student Healtl
To enroll or upd
Student Health

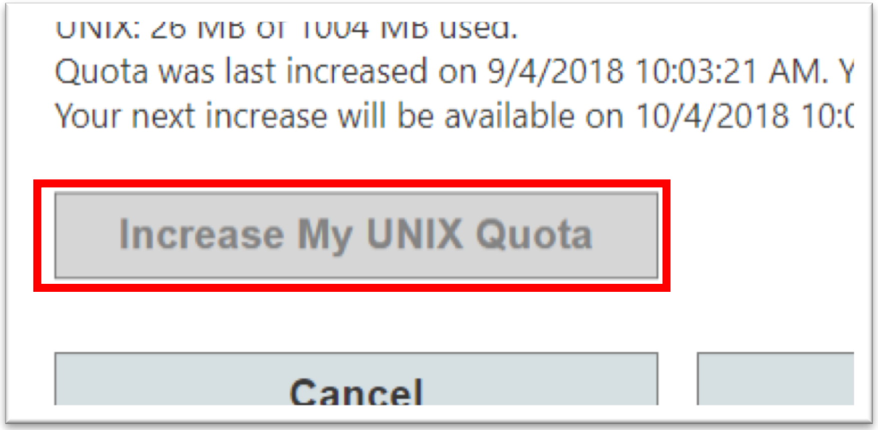
Learn More

Enter Student
Student Health
scheduling, stu
data entry and

CS 1550 – Compile and Run xv6

1. Extend disk Quota, if you have less then 500mb free space
   a) Log in to https://my.pitt.edu
   b) Click on "Profile" at the top of the screen
   c) Click on "Manage Your Account"
   d) Click on "Manage Email Quota"
   e) Click on "Increase My UNIX Quota"




UNIX: 26 MB of 1004 MB used.
Quota was last increased on 9/4/2018 10:03:21 AM. Y
Your next increase will be available on 10/4/2018 10:(

CS 1550 – xv6

- Log in to linux.cs.pitt.edu
    - **ssh** user_name**@linux.cs.pitt.edu**

- Use Terminal(MacOS/Ubuntu)
- Use Putty/Powershell (Windows)

CS 1550 – xv6

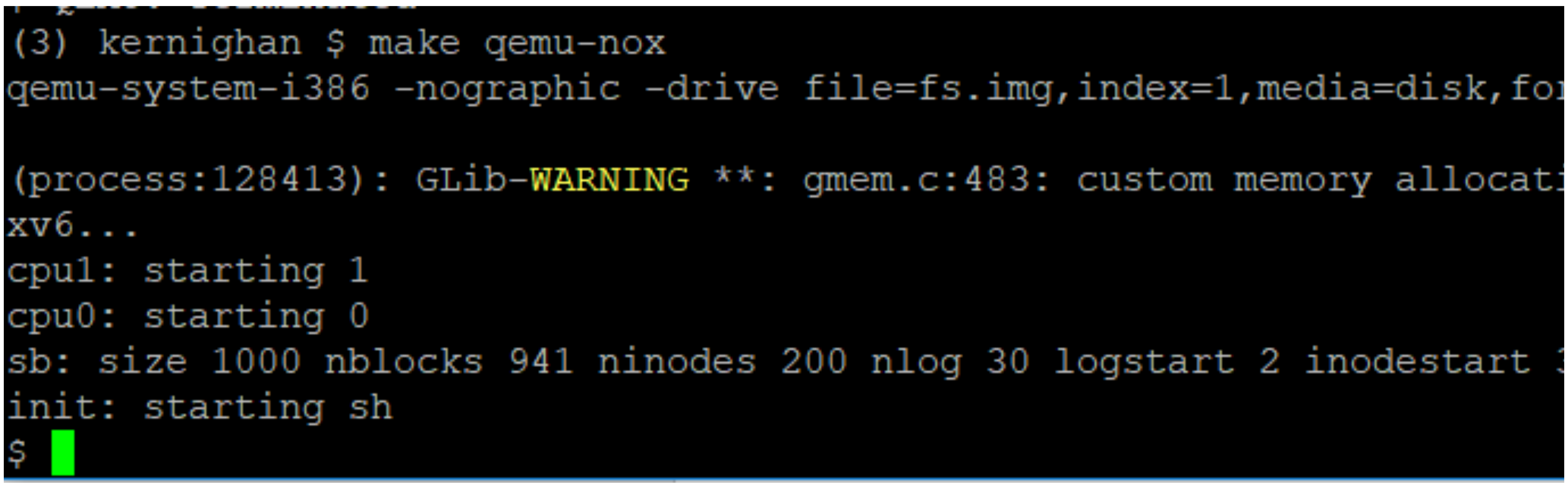
- Download the xv6 source code from github
  - **git clone git://github.com/mit-pdos/xv6-public.git**

CS 1550 – xv6

- Got into the cloned xv6 source code folder
  - **cd xv6-public**

- Compile and run the code with
  - **make qemu-nox**

CS 1550 – xv6

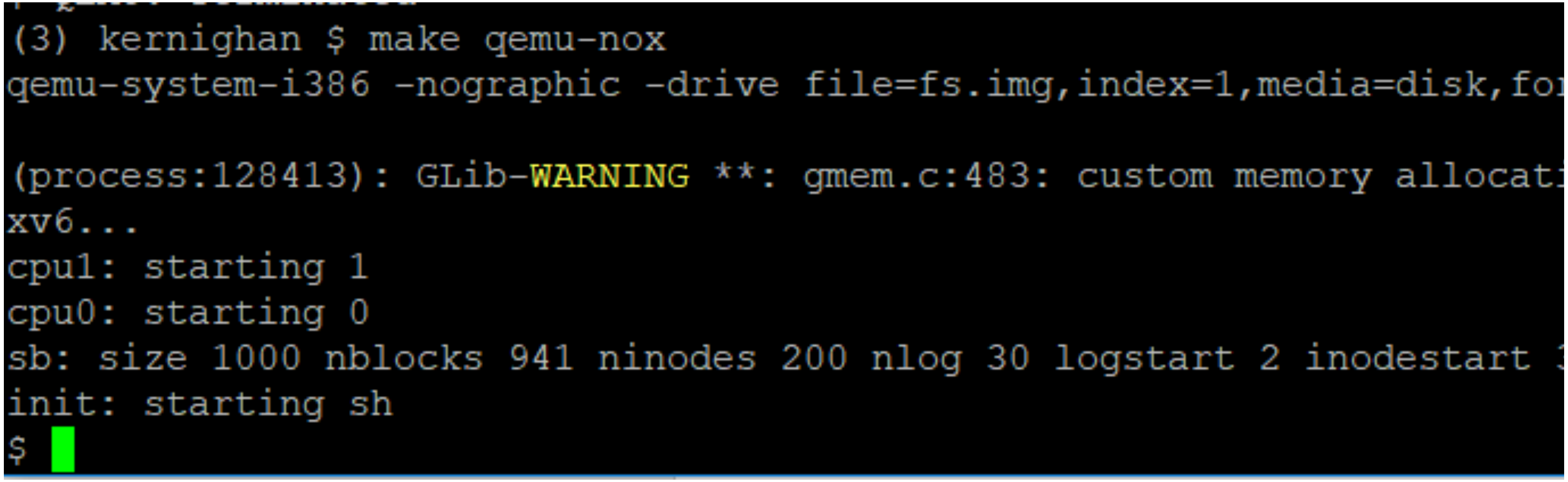


```
(3) kernighan $ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,fo

(process:128413): GLib-WARNING **: gmem.c:483: custom memory allocat
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart
init: starting sh
$ 
```

CS 1550 – xv6

- Compile and run the code with
  - **make qemu-nox**

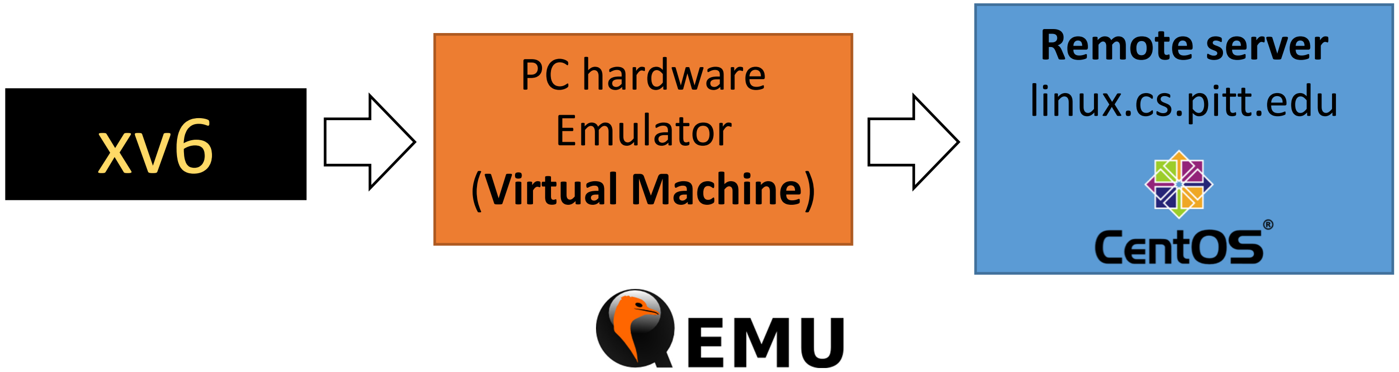
Compiles and **run xv6** with **qemu**



```
(3) kernighan $ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,fo

(process:128413): GLib-WARNING **: gmem.c:483: custom memory allocat
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart
init: starting sh
$ 
```

CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



xv6

PC hardware Emulator **(Virtual Machine)**

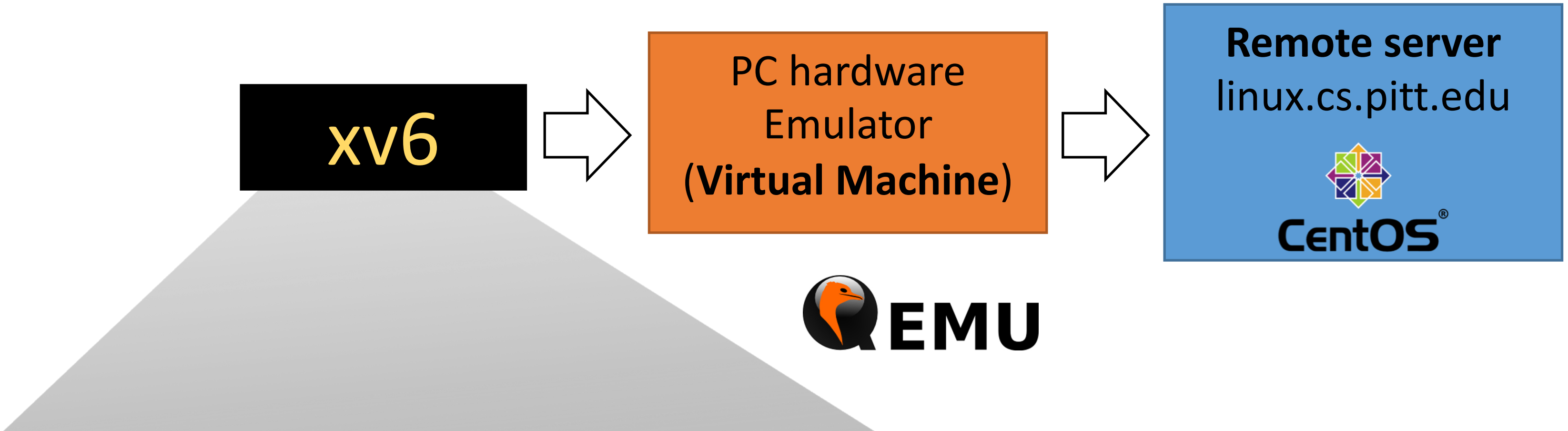
QEMU

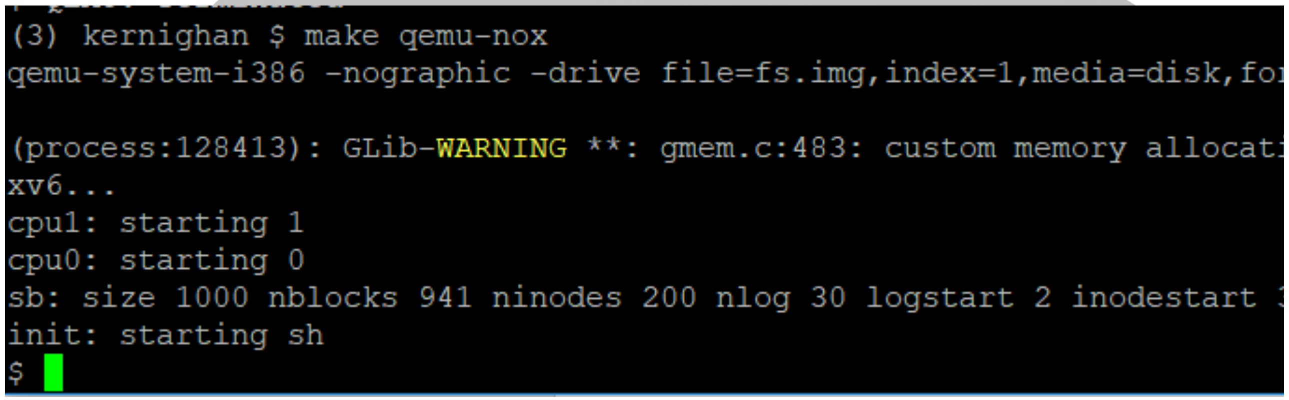
Remote server
linux.cs.pitt.edu

CentOS

CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

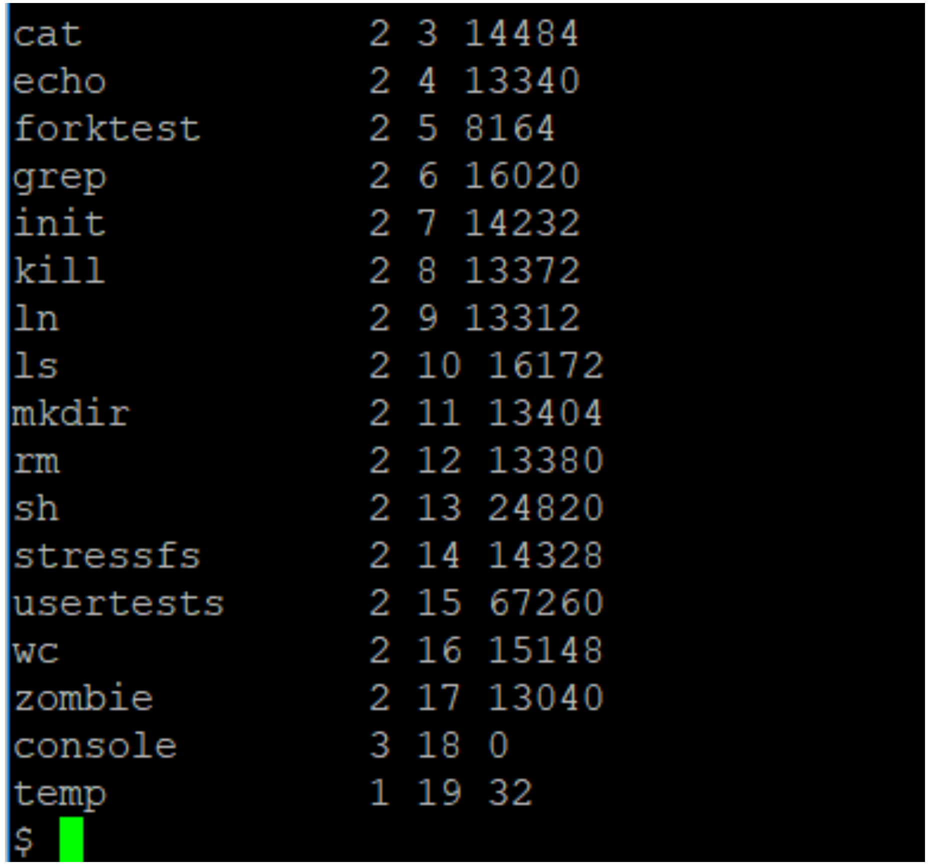




```
(3) kernighan $ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,fo

(process:128413): GLib-WARNING **: gmem.c:483: custom memory allocat
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart
init: starting sh
$
```

CS 1550 – xv6

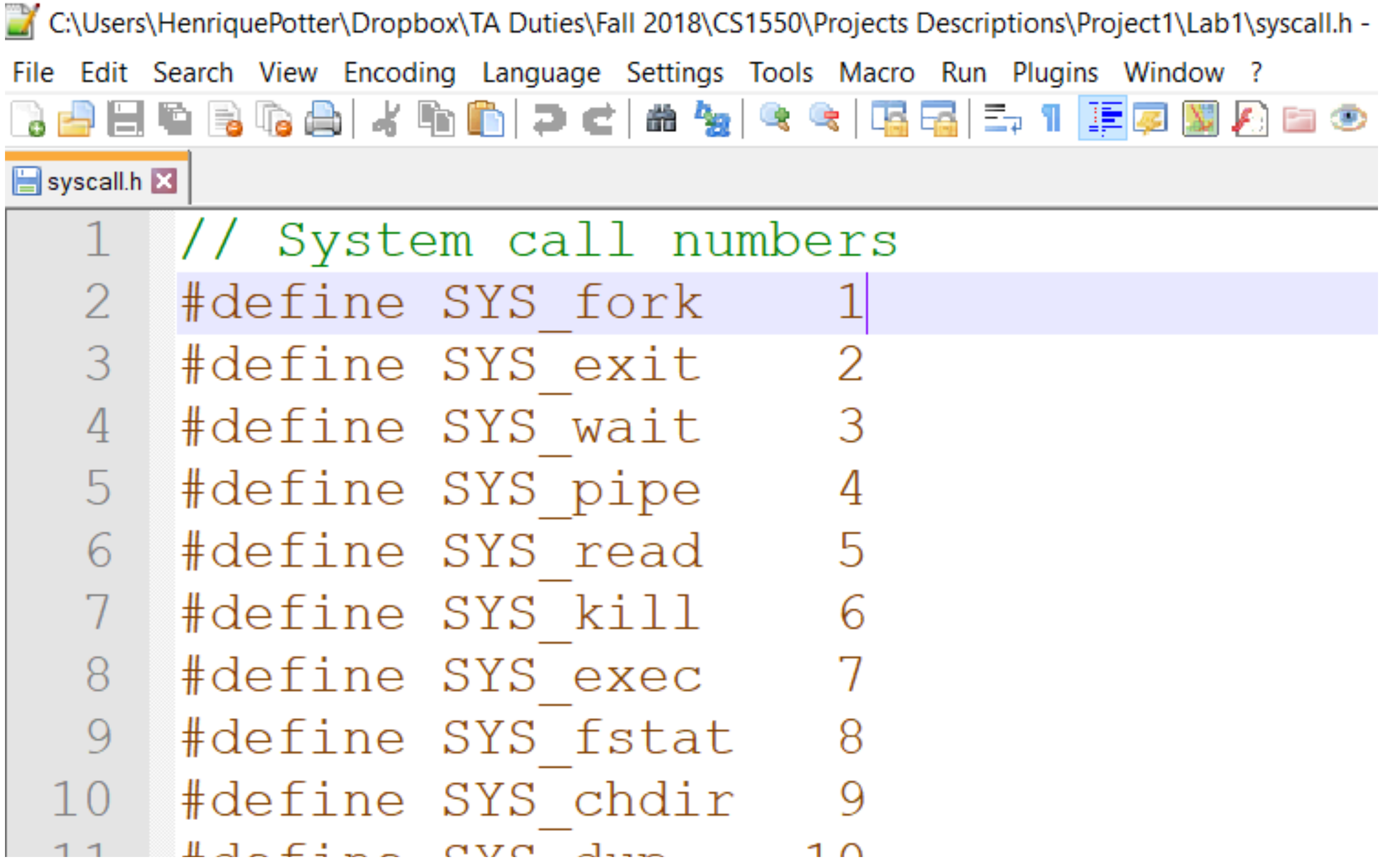
- Once in xv6 you can call **ls**



```
cat              2 3 14484
echo             2 4 13340
forktest         2 5 8164
grep             2 6 16020
init             2 7 14232
kill             2 8 13372
ln               2 9 13312
ls               2 10 16172
mkdir            2 11 13404
rm               2 12 13380
sh               2 13 24820
stressfs         2 14 14328
usertests        2 15 67260
wc               2 16 15148
zombie           2 17 13040
console          3 18 0
temp             1 19 32
$ 
```

CS 1550 – xv6 – Adding a custom Syscall

- First we need to define our new call and its number at
  - **syscall.h**



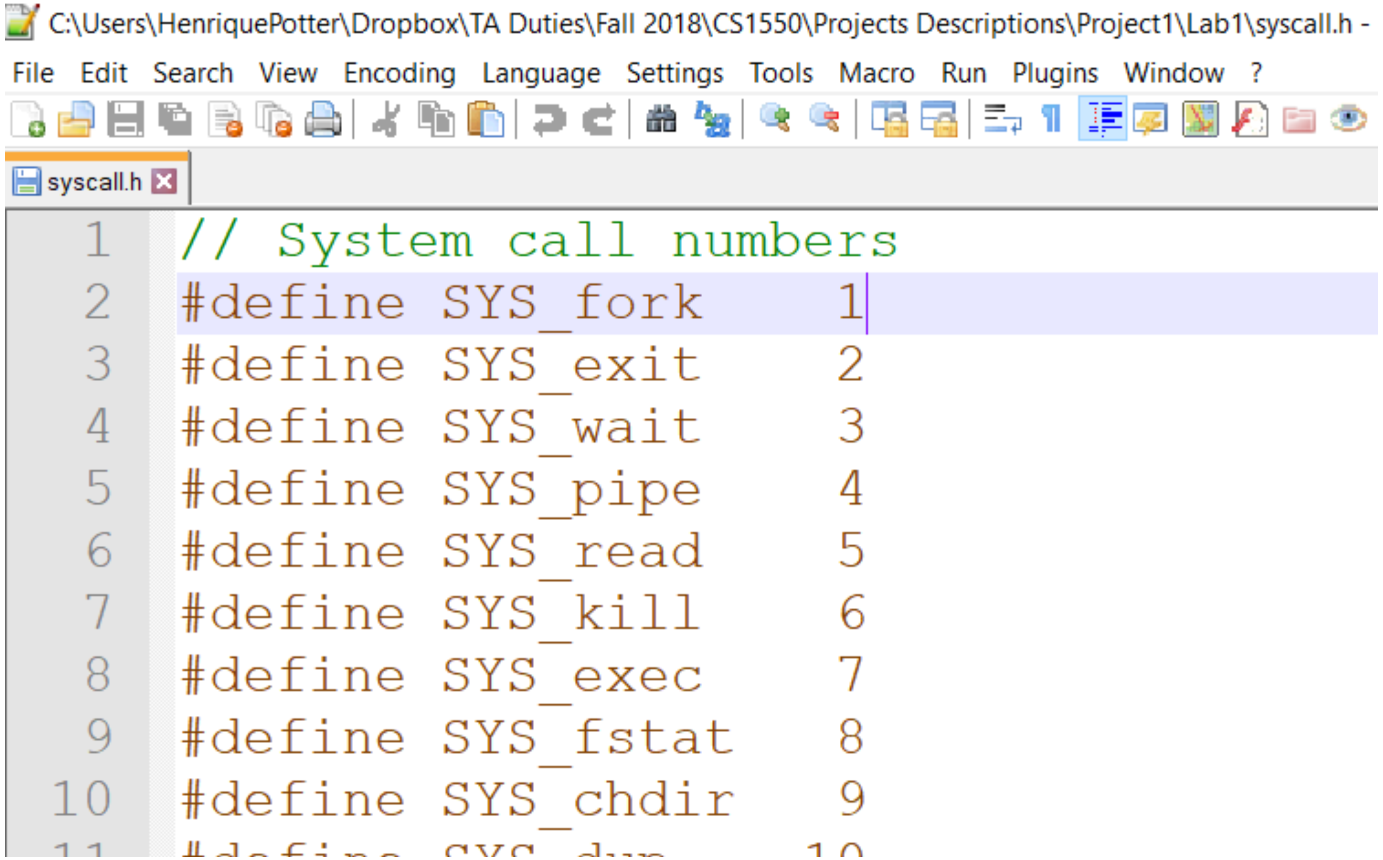
C:\Users\HenriquePotter\Dropbox\TA Duties\Fall 2018\CS1550\Projects Descriptions\Project1\Lab1\syscall.h -
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
```

CS 1550 – xv6 – Adding a custom Syscall

- First we need to define our new call and its number at
  - **syscall.h**

- Add
  - #define SYS_getday 22



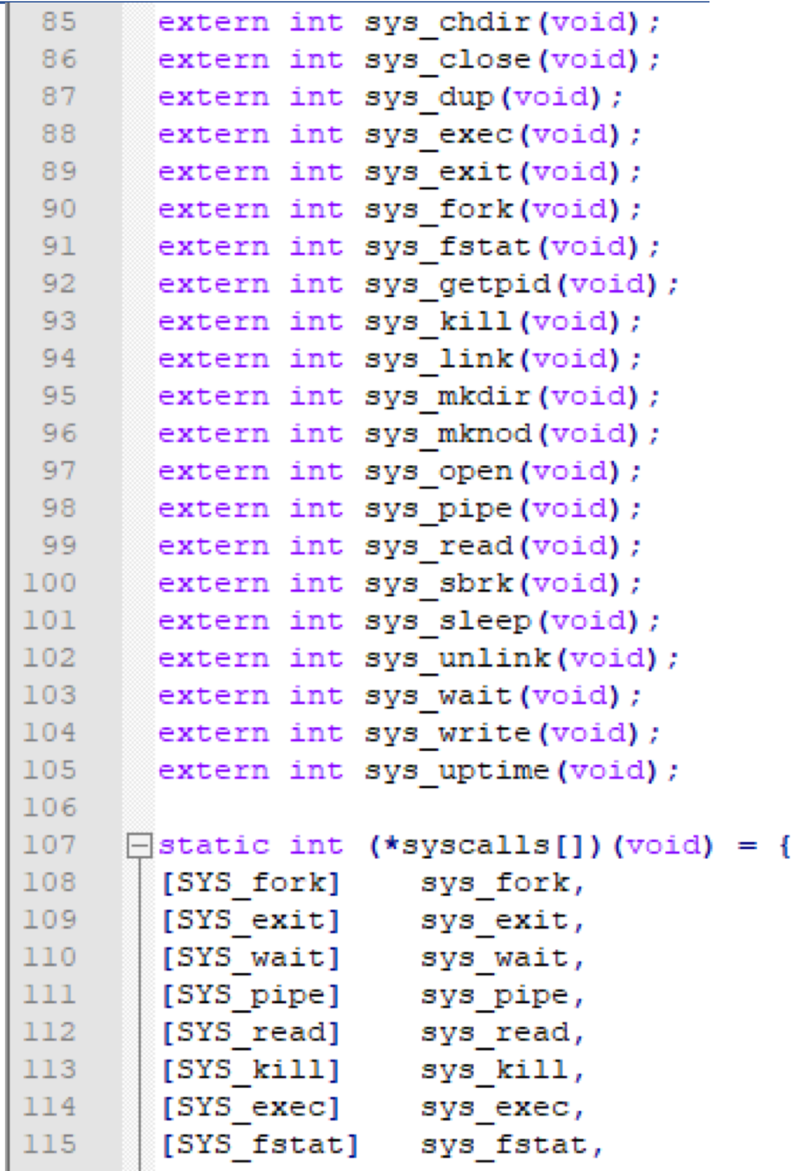
C:\Users\HenriquePotter\Dropbox\TA Duties\Fall 2018\CS1550\Projects Descriptions\Project1\Lab1\syscall.h -
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
syscall.h

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
```

CS 1550 – xv6 – Adding a custom Syscall

- Next we need to map the new call in the array pointer of system calls
  - **syscall.c**

- Add
  - extern int sys_getday(void);
  - [SYS_getday]   sys_getday,



```c
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
```

CS 1550 – xv6 – Adding a custom Syscall

- Then we need to implement the actual method

- In xv6 this is organized in two files.
  - sysfile.c -> file related system calls
  - sysproc.c -> all the other syscalls

CS 1550 – xv6 – Adding a custom Syscall

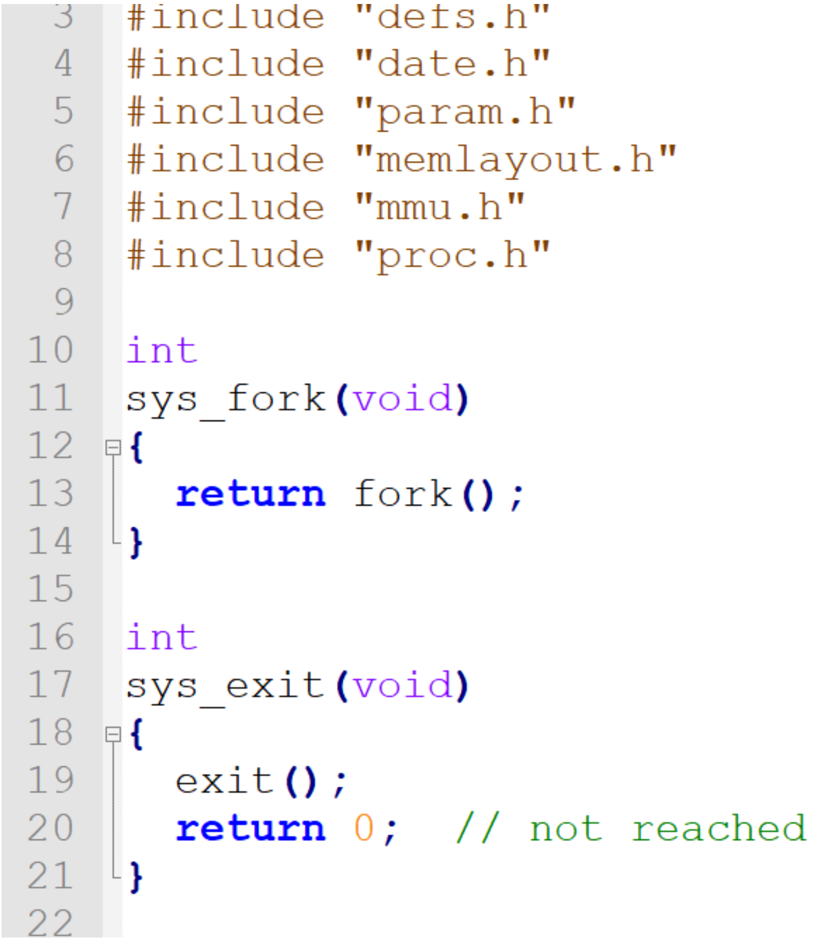
- Then we need to implement the actual method

- In xv6 this is organized in two files.
  - sysfile.c -> file related system calls
  - **sysproc.c -> all the other syscalls**

CS 1550 – xv6 – Adding a custom Syscall

- Then we need to implement the actual method

- In xv6 this is organized in two files.
  - sysfile.c -> file related system calls
  - **sysproc.c -> all the other syscalls**



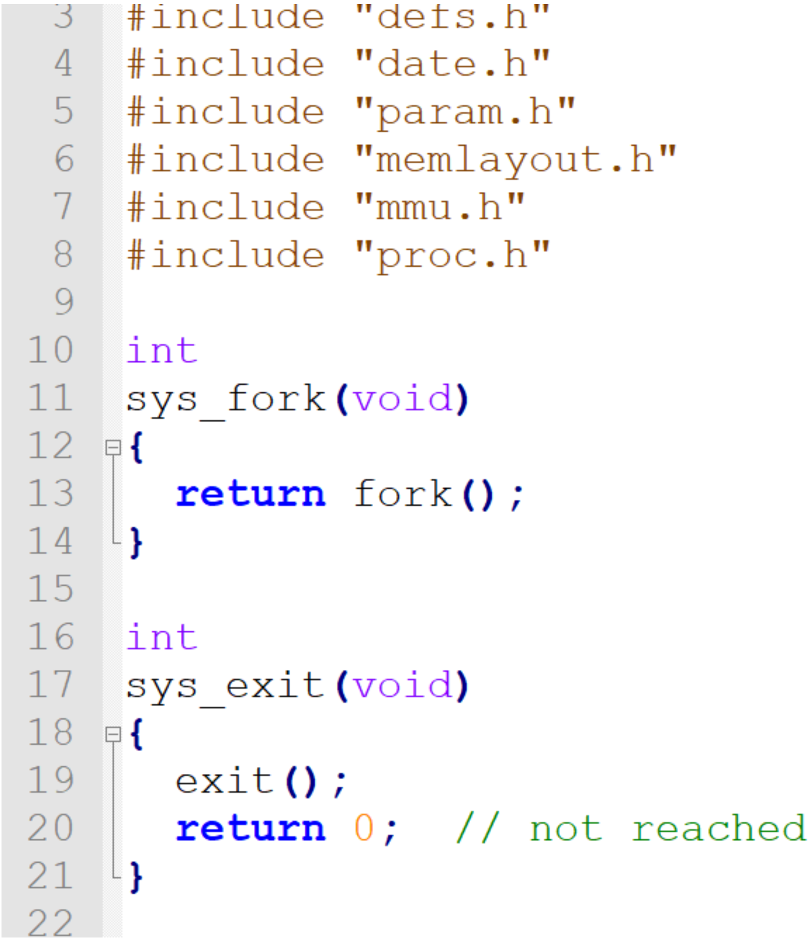
```
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"

int
sys_fork(void)
{
  return fork();
}

int
sys_exit(void)
{
  exit();
  return 0;  // not reached
}

```

CS 1550 – xv6 – Adding a custom Syscall

- Then we need to implement the actual method

- In xv6 this is organized in two files.
  - sysfile.c -> file related system calls
  - **sysproc.c -> all the other syscalls**

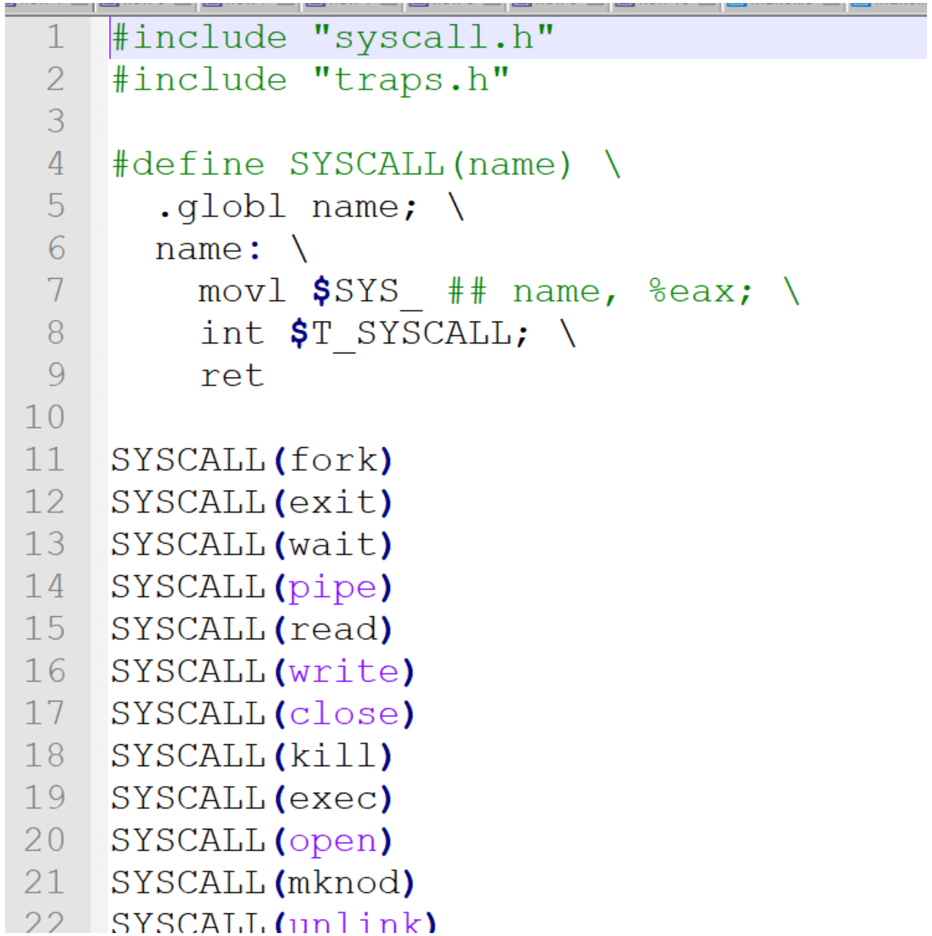
```
int
sys_getday(void)
{
   return 6;
}
```



```c
#include "defs.h"
#include "date.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "proc.h"

int
sys_fork(void)
{
  return fork();
}

int
sys_exit(void)
{
  exit();
  return 0;  // not reached
}

```

CS 1550 – xv6 – Adding a custom Syscall

- Afterwards we define the interface for user programs to call
  - Open usys.S
- Add
  - SYSCALL(getday)

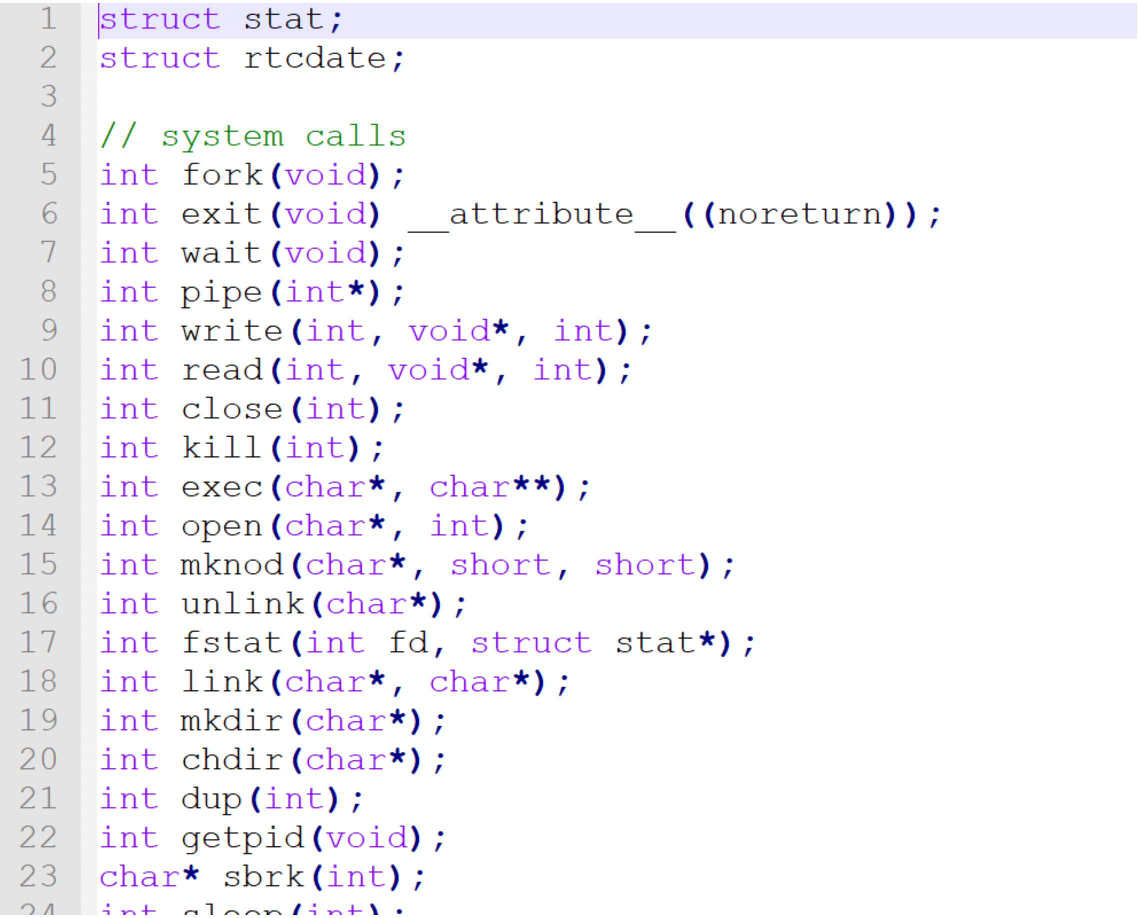


```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
```

CS 1550 – xv6 – Adding a custom Syscall

- Finally we open
  - user.h

- Add
  - int getday(void);



```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(char*, int);
int mknod(char*, short, short);
int unlink(char*);
int fstat(int fd, struct stat*);
int link(char*, char*);
int mkdir(char*);
int chdir(char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
```

CS 1550 – xv6 – Adding a custom Syscall

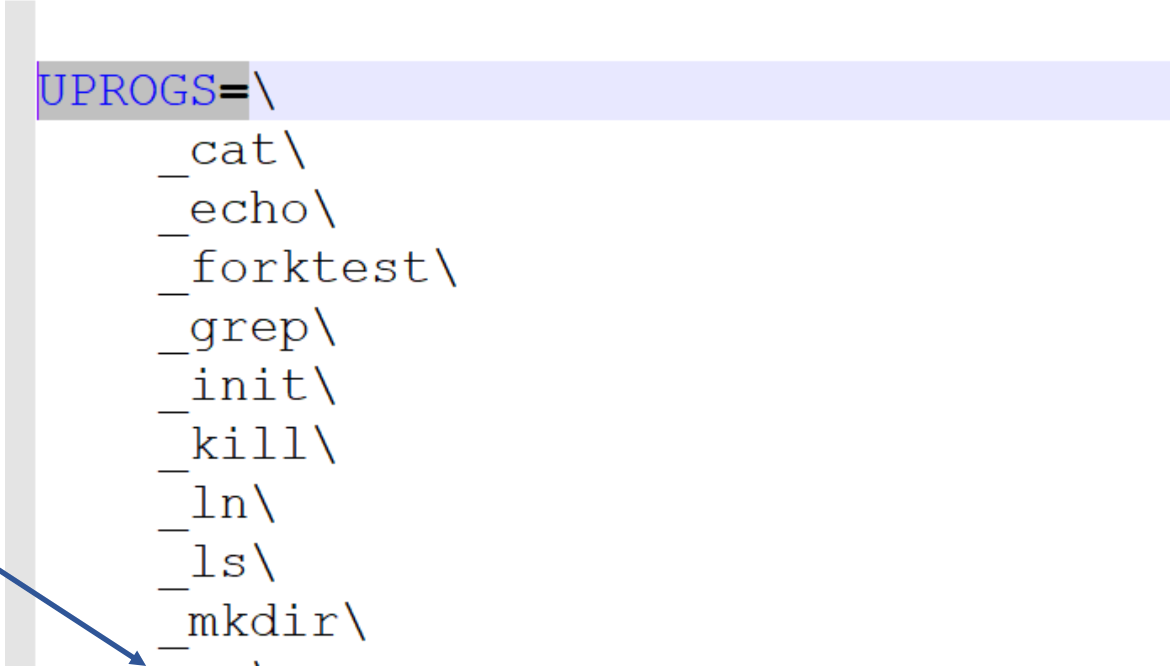
- Example user program
  - todays_date.c

```
#include "types.h"
#include "stat.h"
#include "user.h"

int main(void) {
        printf(1, "Today is %d\n", getday());
        exit();
}
```

CS 1550 – xv6 – Adding a custom Syscall

- Adding an user program
  - Open Makefile

- Add
  - _todays_date\



```
UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
```