CS 1550

Lab 2 – xv6 Introduction

Setup and exercise

Teaching Assistant

Maher Khan

(Slides credited to Henrique Potter)

Recitation TA – Office Hours

- Tuesday
  - 2:00pm - 5:00pm
- Friday
  - 1:00pm – 4:00pm

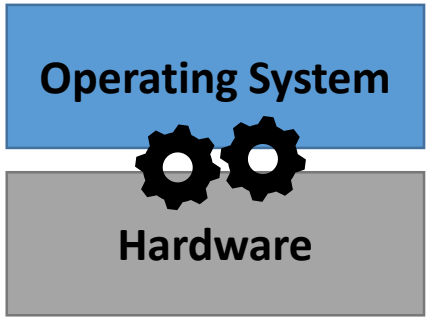
- Office: SenSq 5802

Slides on GitHub!

- https://github.com/maher460/Pitt_CS1550_recitation_materials

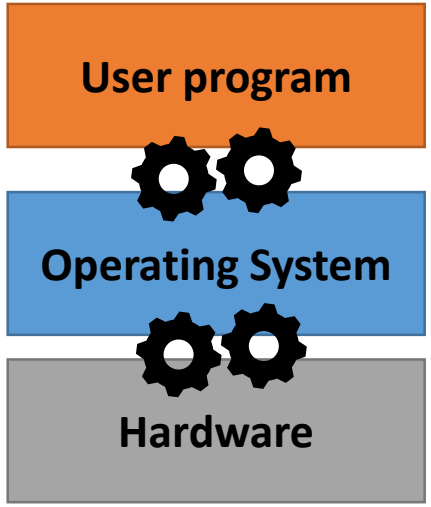
CS 1550 – Kernel Space vs User Space

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



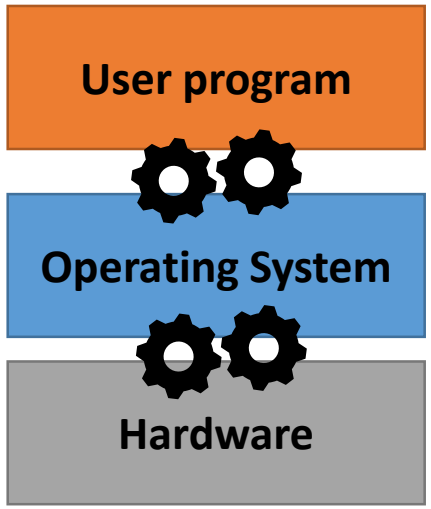
CS 1550 – Kernel Space vs User Space

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



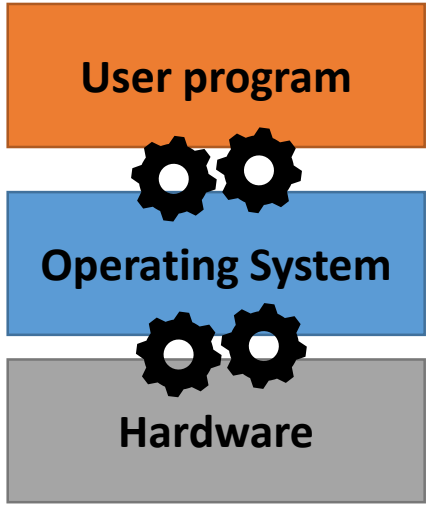
CS 1550 – Kernel Space vs User Space

- 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)

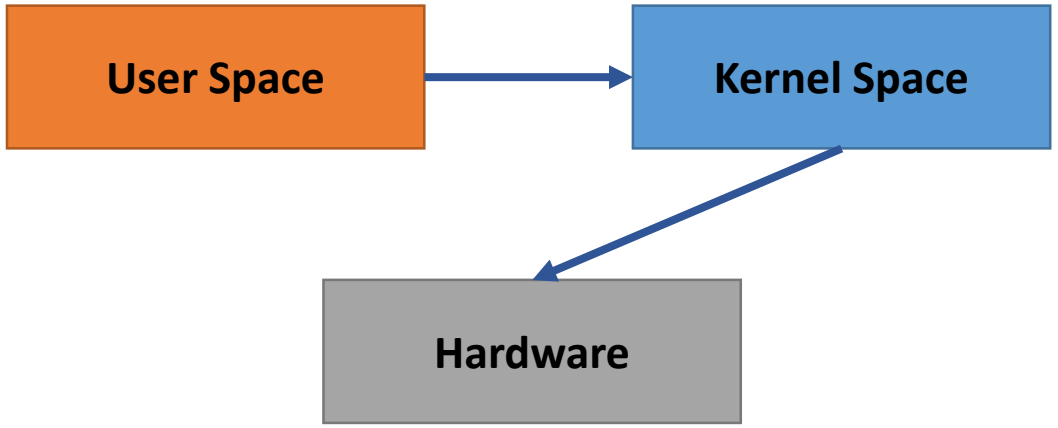


CS 1550 – Kernel Space vs User Space

- 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**

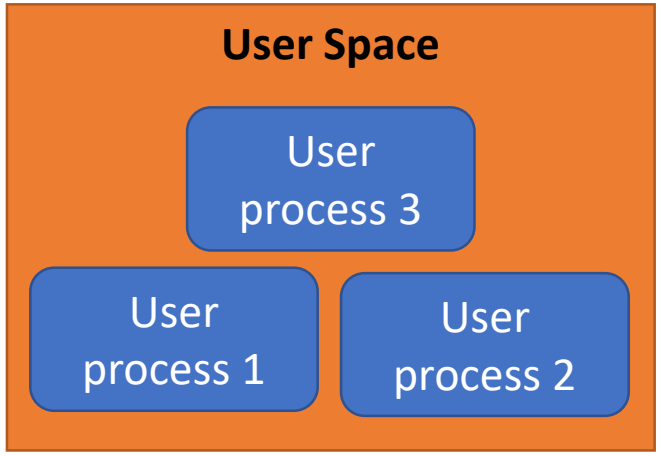
CS 1550 – Kernel Space vs User Space

- 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**



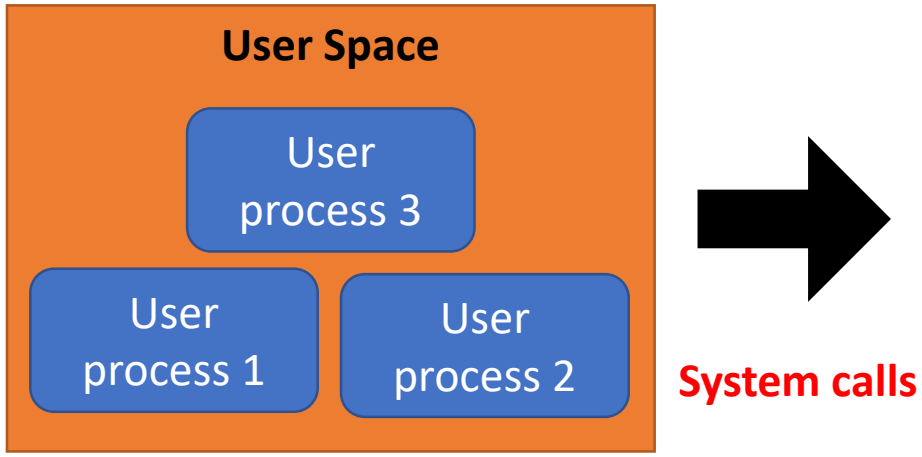
CS 1550 – Kernel Space vs User Space

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



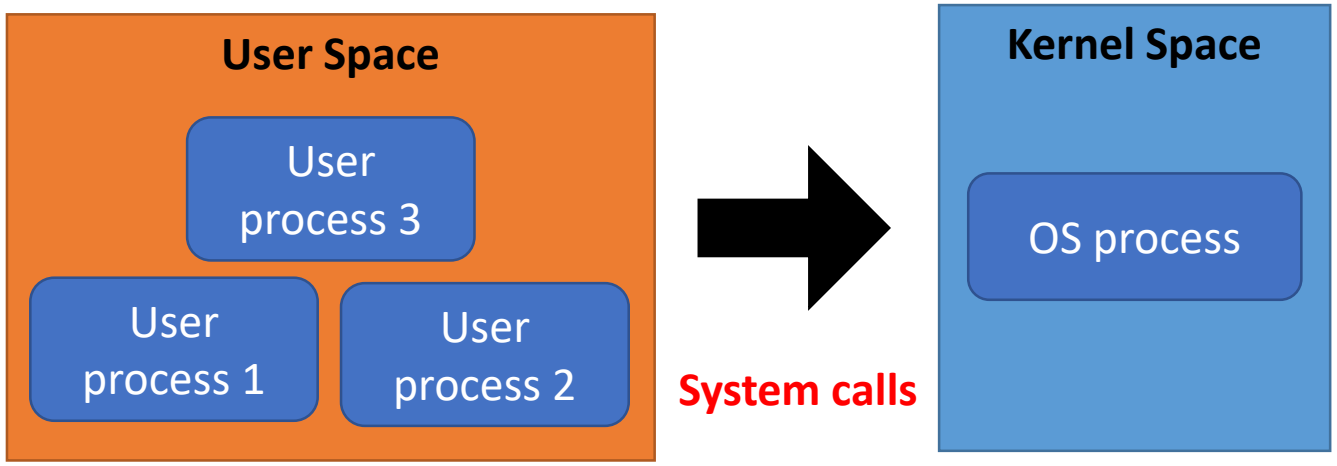
CS 1550 – Kernel Space vs User Space

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



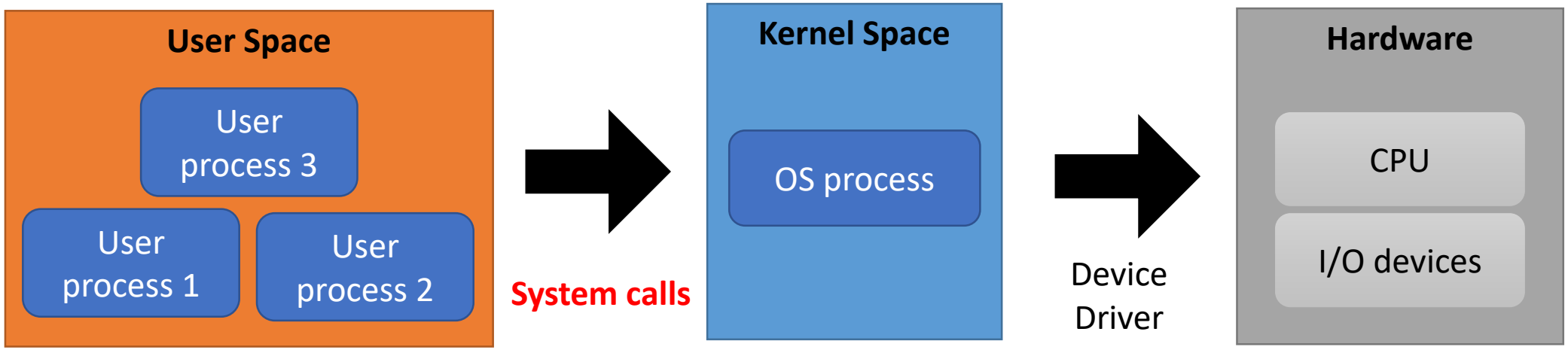
CS 1550 – Kernel Space vs User Space

- **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

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?

CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

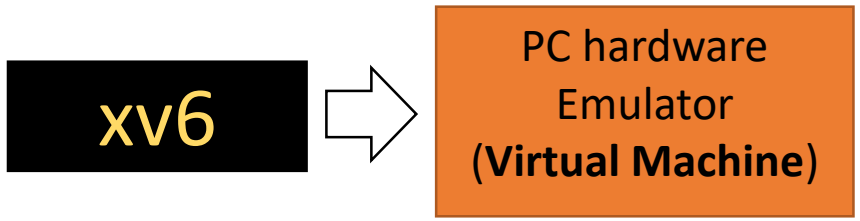
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

xv6

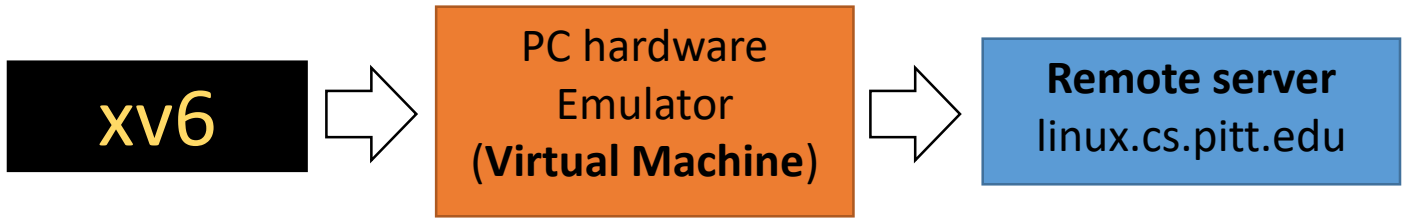
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



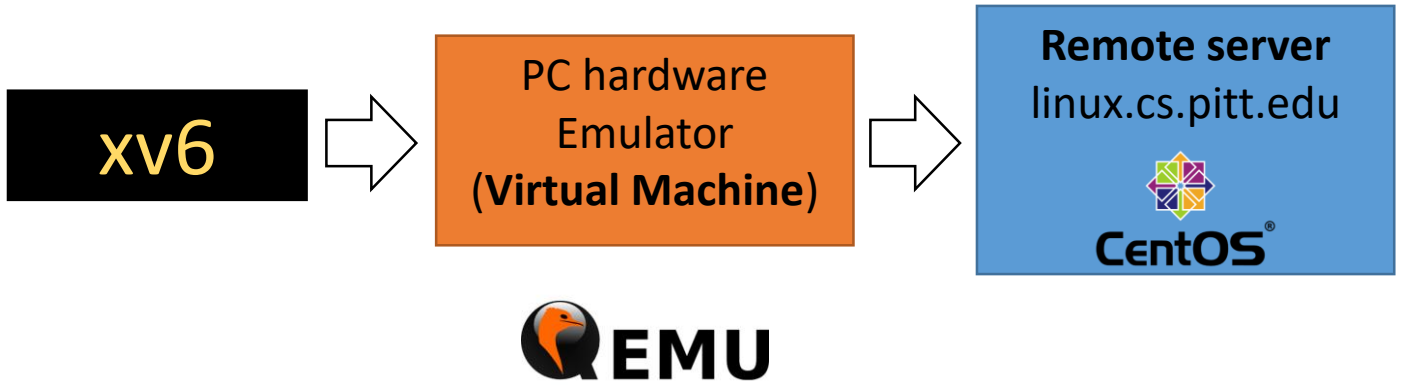
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



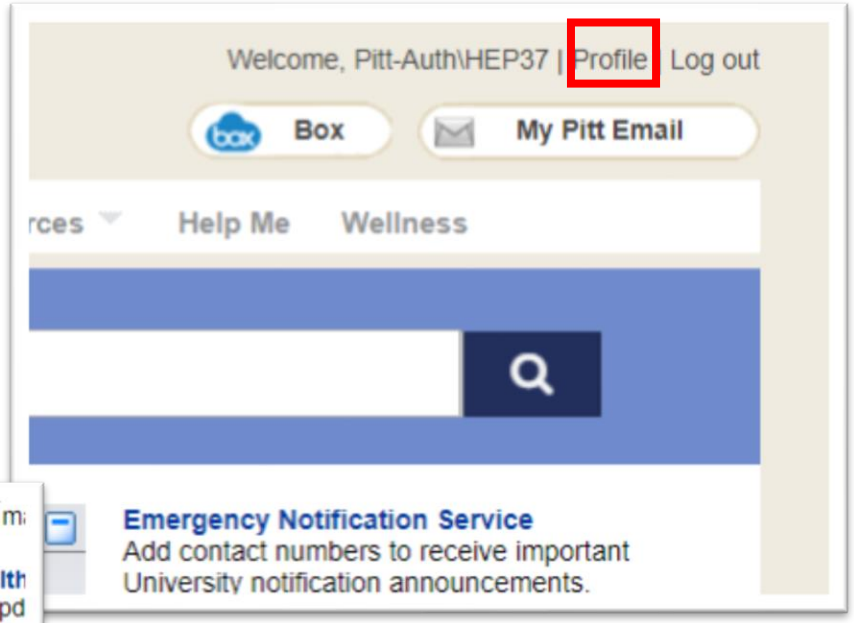
CS 1550 – xv6

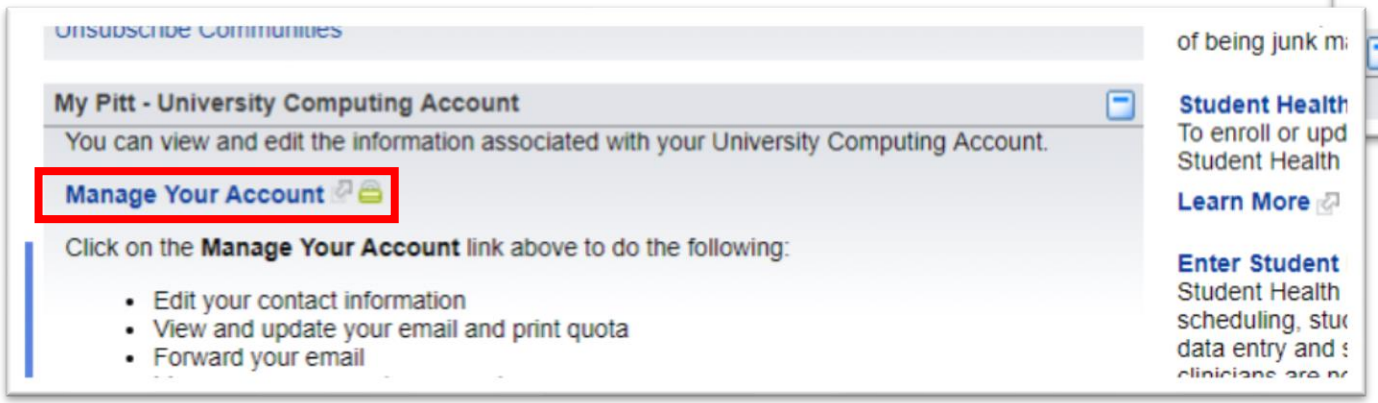
- Compile and Run xv6 in a cs pitt server



CS 1550 – Compile and Run xv6

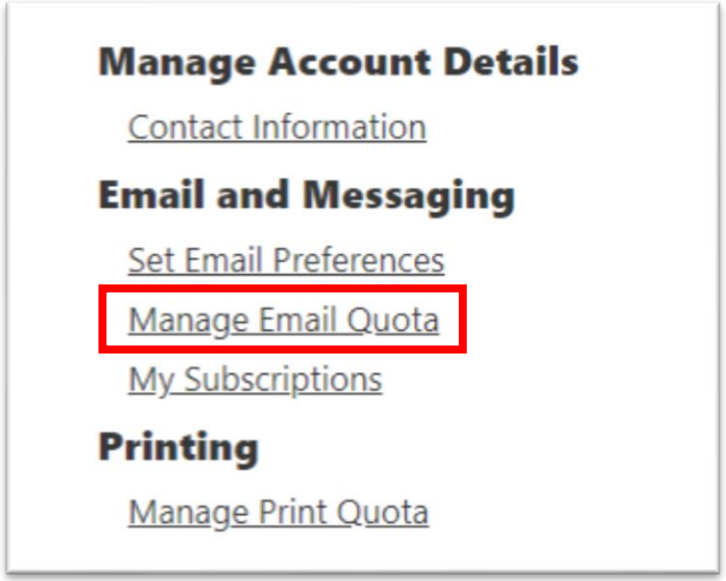
1. Extend disk Quota, if you have less then 500mb free space
   a) Log in to https://my.pitt.edu
   b) Click on "Profile" at the top of the screen
   c) Click on "Manage Your Account"
   d) Click on "Manage Email Quota"
   e) Click on "Increase My UNIX Quota"

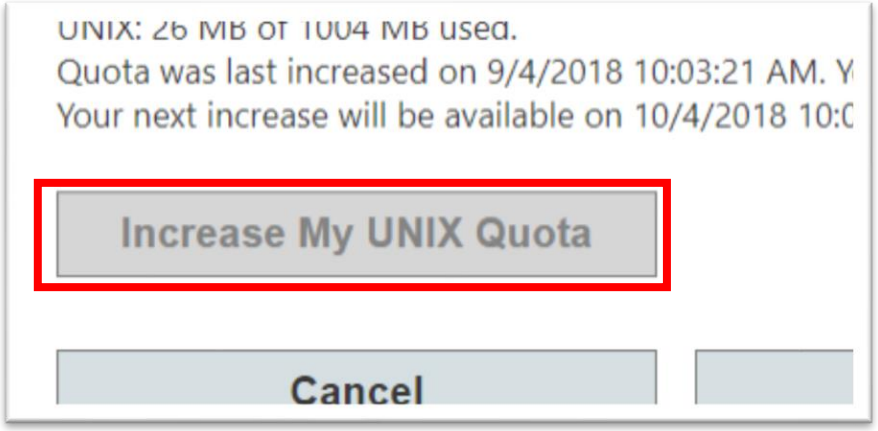




CS 1550 – Compile and Run xv6

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"





CS 1550 – xv6

- Connect to Pulse (campus VPN) first if connected to non-Pitt wifi.
- Log in to linux.cs.pitt.edu
  - **ssh** user_name**@linux.cs.pitt.edu**

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

CS 1550 – xv6

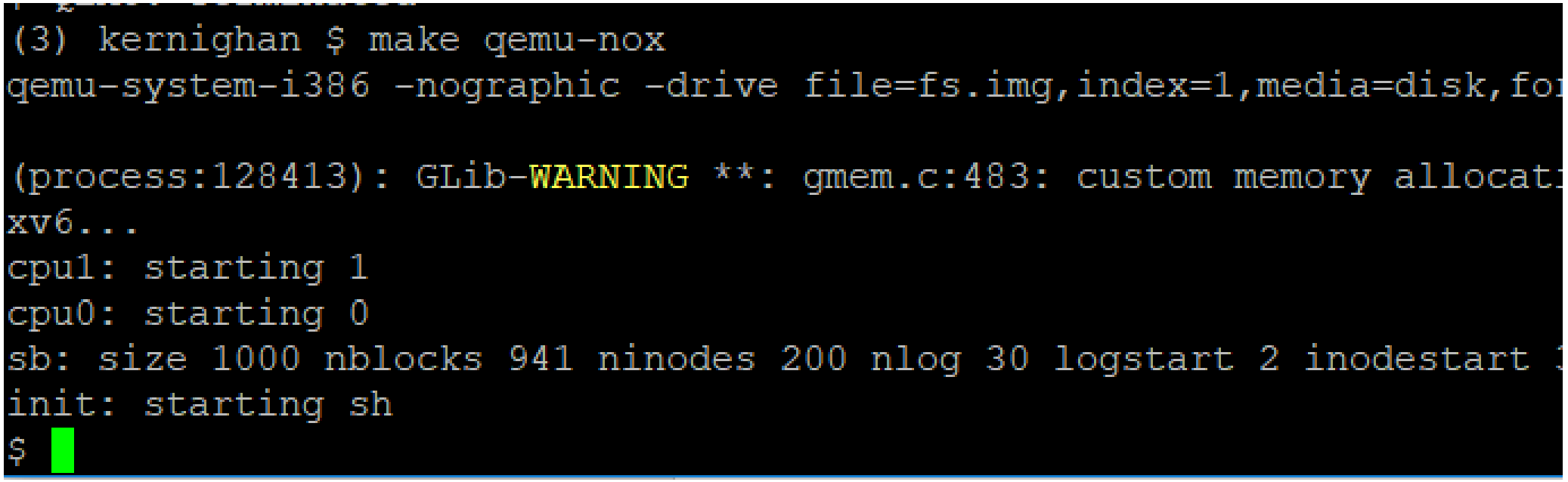
- Go into the private folder
  - **cd private**
- Download the xv6 source code from github
  - **git clone git://github.com/mit-pdos/xv6-public.git**

CS 1550 – xv6

- Go 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**

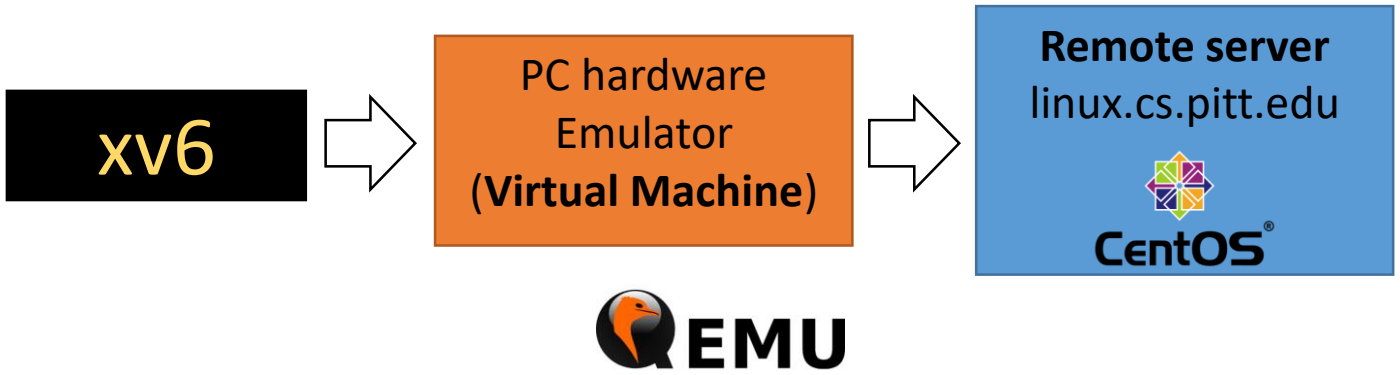
Compiles and **run xv6** with **qemu**

```
(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 allocati
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart
init: starting sh
$
```

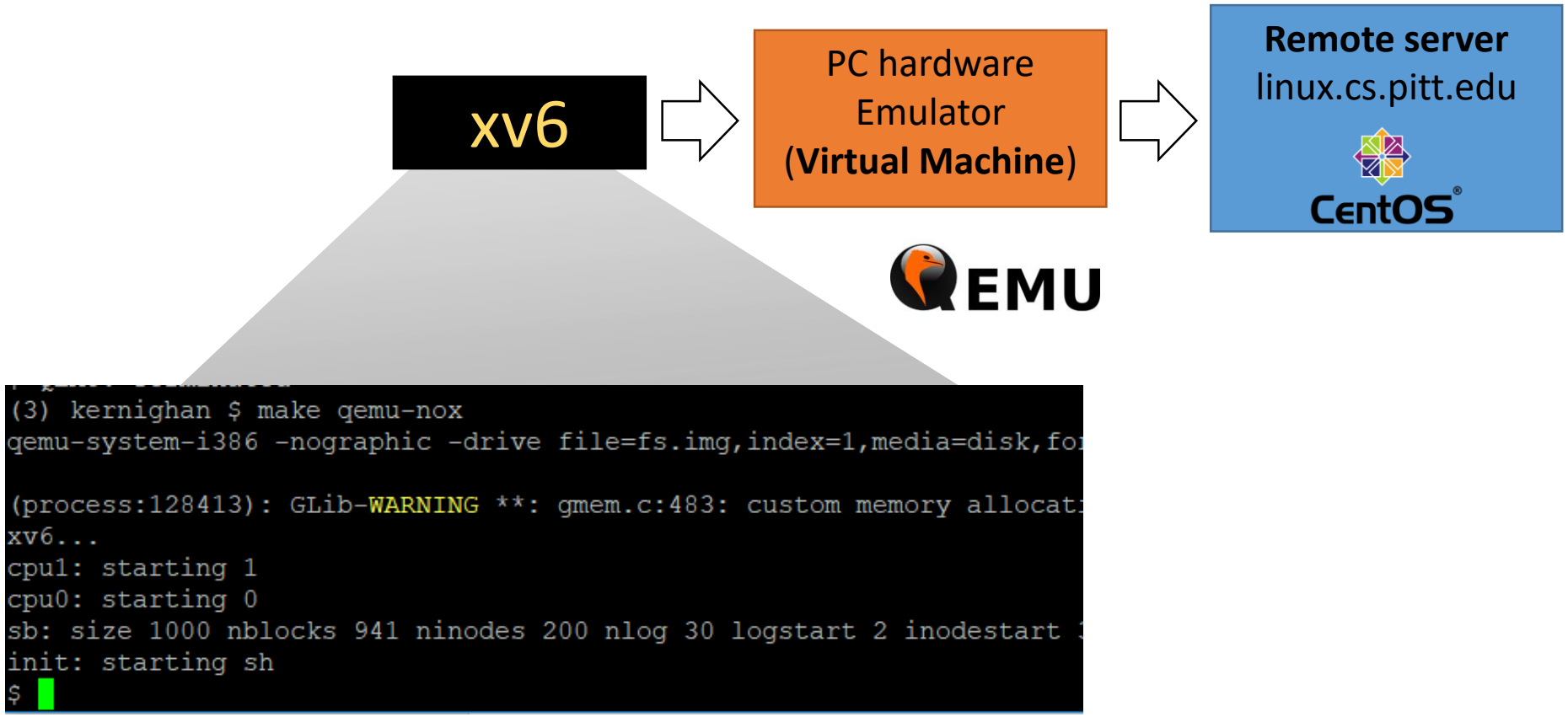
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



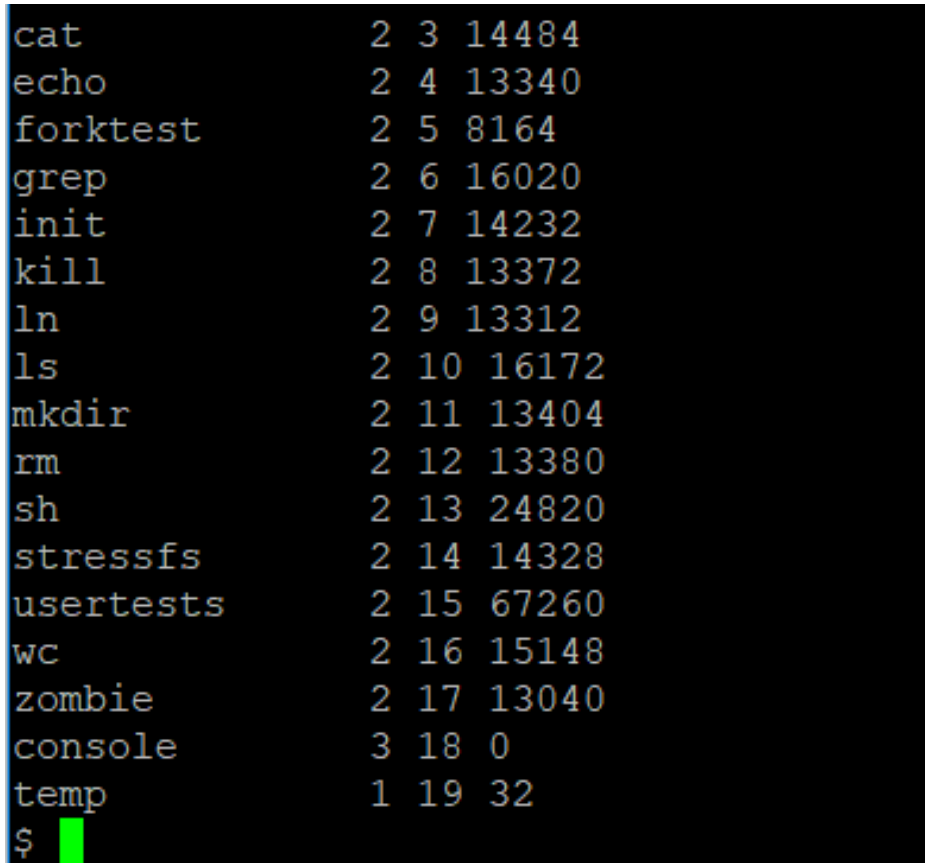
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



CS 1550 – xv6

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



Exiting xv6

- To exit xv6:
  - **Control+a**
  - Then, **x**

My xv6 Github

- Before you start changing the code, it’s better to have the code accessible through your own private GitHub repository.
- Go to www.github.com
- Create an account if you don’t have one already, otherwise login.
- Create a new repository

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner / Repository name *


maher460 / my_xv6

Great repository names are short and memorable. Need inspiration? How about **stunning-memory**.

Description (optional)

Public
Anyone can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None | Add a license: None

Create repository

My xv6 Github

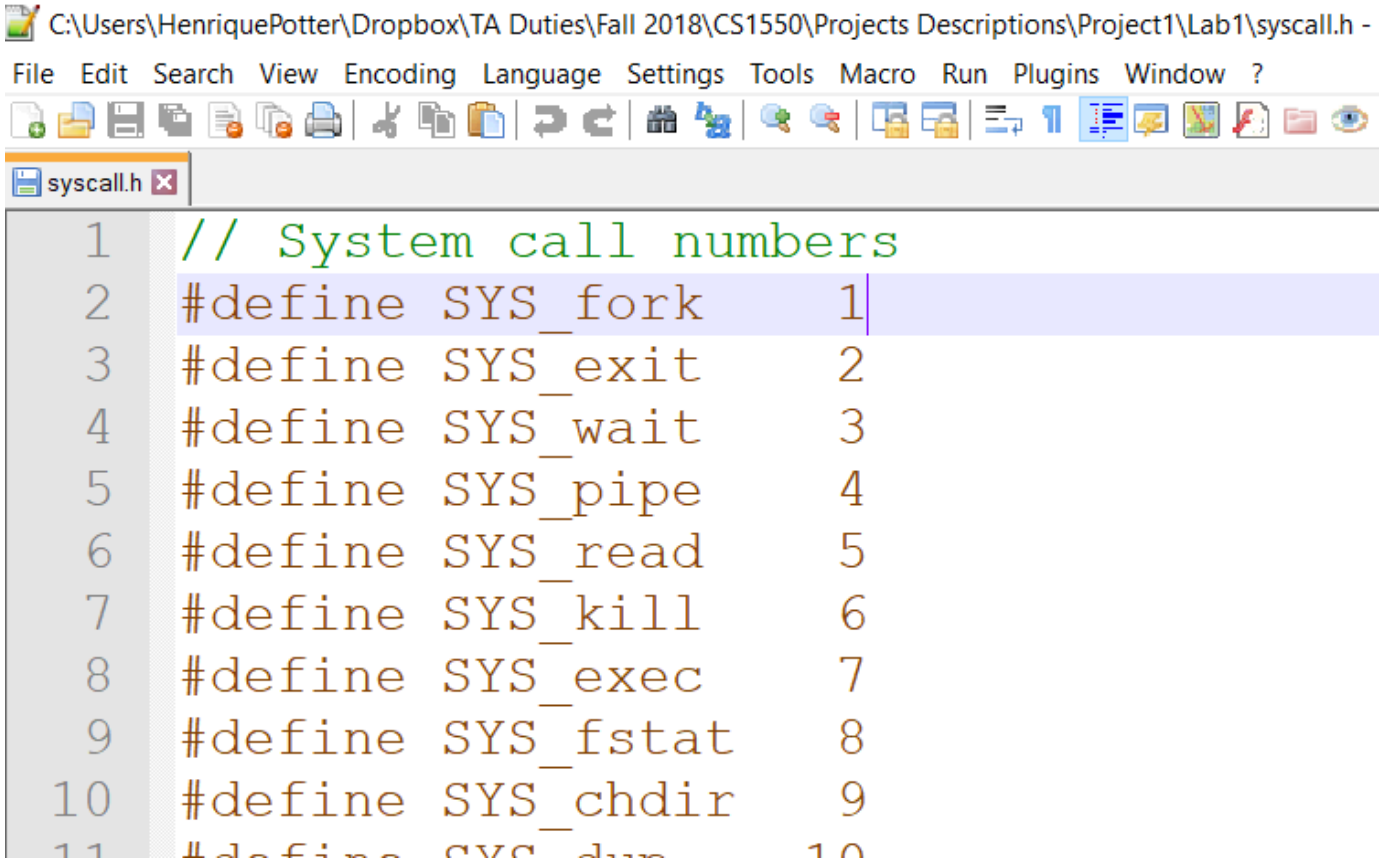
- Inside the xv6-public folder in linux.cs.pitt.edu machine:
  - Set the git remote URL to your newly created private repo
    - **git remote set-url origin https://github.com/maher460/my_xv6.git**
  - Next you will have to remove a tag from your local git in order to avoid a pointer conflict:
    - **git tag –d master**
  - Now, we are ready to push all the code to the private repo!
    - **git push –u origin master**
  - Check your private GitHub repo to make sure all the code is there.

My xv6 GitHub

- You want to clone the GitHub repo to your own local machine
- In your Terminal/PowerShell (in a new window/tab to keep the ssh of linux.cs.pitt.edu still running):
  - **git clone https://github.com/maher460/my_xv6.git**
- Now, you can use your favourite IDE to code and upload all the changes to your private git:
  - **git add –A**
  - **git commit –m “**comment about the changes you made**”**
  - **git push**
- Pull all the uploaded changes in the linux.cs.pitt.edu:
  - **git pull**

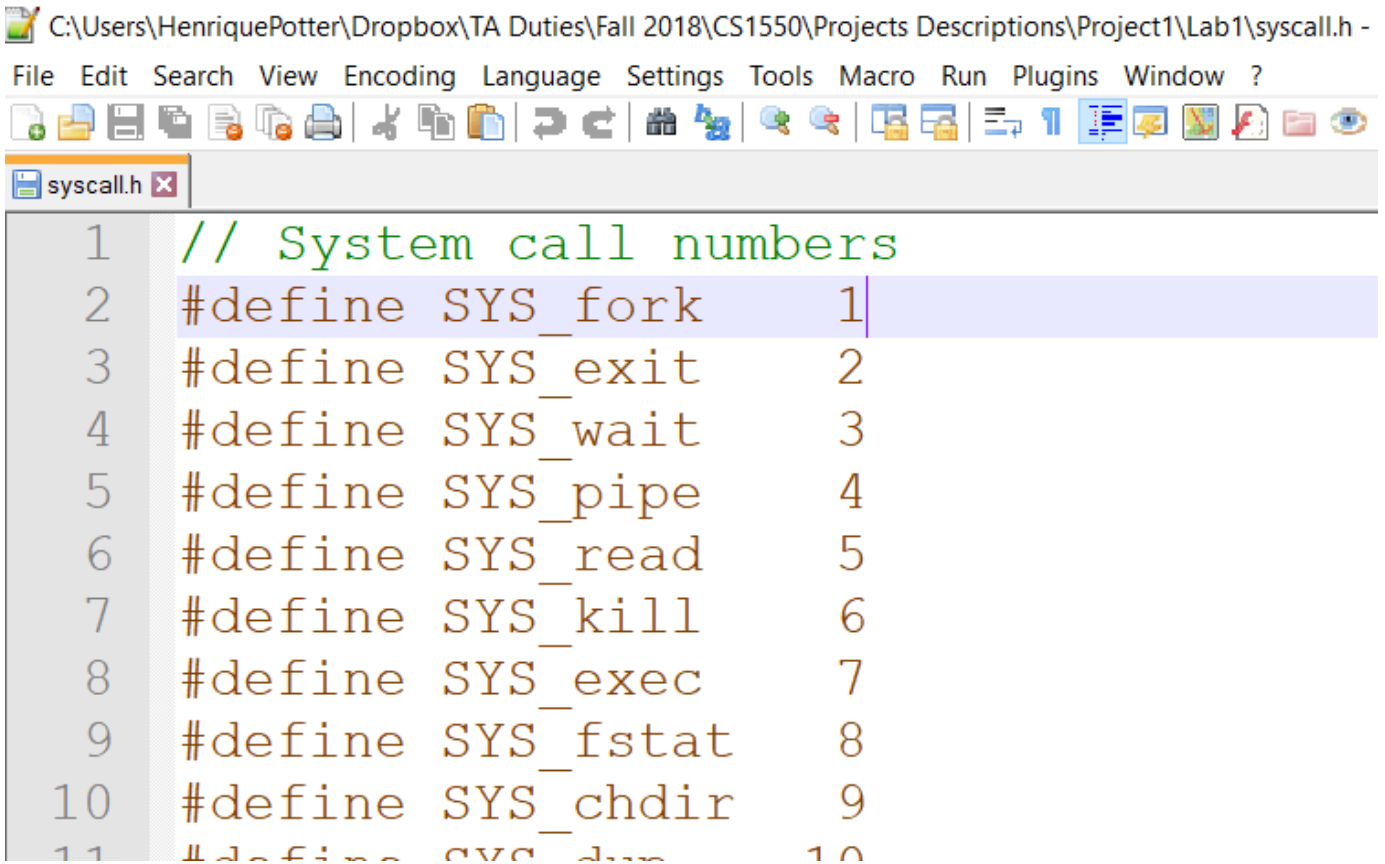
CS 1550 – xv6 – Adding a custom Syscall

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



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



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
  - [SYS_getday]   sys_getday,

```
110
111 static int (*syscalls[])(void) = {
112 [SYS_fork]    sys_fork,
113 [SYS_exit]    sys_exit,
114 [SYS_wait]    sys_wait,
115 [SYS_pipe]    sys_pipe,
116 [SYS_read]    sys_read,
117 [SYS_kill]    sys_kill,
118 [SYS_exec]    sys_exec,
119 [SYS_fstat]   sys_fstat,
120 [SYS_chdir]   sys_chdir,
121 [SYS_dup]     sys_dup,
122 [SYS_getpid]  sys_getpid,
123 [SYS_sbrk]    sys_sbrk,
124 [SYS_sleep]   sys_sleep,
125 [SYS_uptime]  sys_uptime,
126 [SYS_open]    sys_open,
127 [SYS_write]   sys_write,
```

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);

```
96  extern int sys_mknod(void);
97  extern int sys_open(void);
98  extern int sys_pipe(void);
99  extern int sys_read(void);
00  extern int sys_sbrk(void);
01  extern int sys_sleep(void);
02  extern int sys_unlink(void);
03  extern int sys_wait(void);
04  extern int sys_write(void);
05  extern int sys_uptime(void);
06
07  static int (*syscalls[])(void) = {
08      [SYS_fork]    sys_fork,
09      [SYS_exit]    sys_exit,
10      [SYS_wait]    sys_wait,
11      [SYS_pipe]    sys_pipe,
```

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**

```
3  #include "defs.h"
4  #include "date.h"
5  #include "param.h"
6  #include "memlayout.h"
7  #include "mmu.h"
8  #include "proc.h"
9
10 int
11 sys_fork(void)
12 {
13   return fork();
14 }
15
16 int
17 sys_exit(void)
18 {
19   exit();
20   return 0;  // not reached
21 }
22
```

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**

- Add the following to sysproc.c:

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

```
#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)

```
1  #include "syscall.h"
2  #include "traps.h"
3
4  #define SYSCALL(name) \
5    .globl name; \
6    name: \
7      movl $SYS_ ## name, %eax; \
8      int $T_SYSCALL; \
9      ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
```

CS 1550 – xv6 – Adding a custom Syscall

- Finally we open
  - user.h
- Add
  - int getday(void);

```
1  struct stat;
2  struct rtcdate;
3
4  // system calls
5  int fork(void);
6  int exit(void) __attribute__((noreturn));
7  int wait(void);
8  int pipe(int*);
9  int write(int, void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(char*, int);
15 int mknod(char*, short, short);
16 int unlink(char*);
17 int fstat(int fd, struct stat*);
18 int link(char*, char*);
19 int mkdir(char*);
20 int chdir(char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
```

CS 1550 – xv6 – Adding a custom Syscall

- Example user program
  - todays_date.c

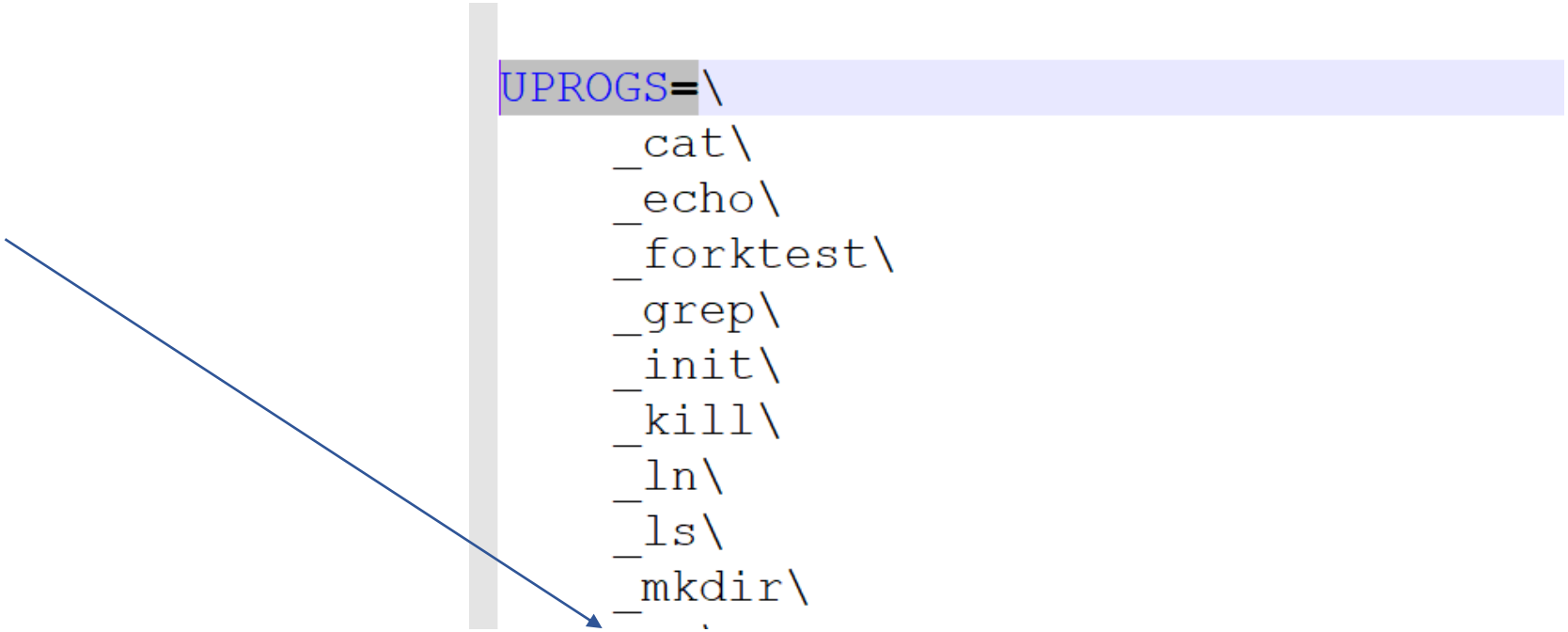
```
#include "types.h"
#include "stat.h"
#include "user.h"

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\



CS 1550 – xv6 – Adding a custom Syscall

- Adding an user program
  - Open makefile
- and also add
  - todays_date.c\

```
EXTRA=\
    mkfs.c ulib.c user.h cat.c e
    kill.c\
    ln.c ls.c mkdir.c rm.c stres
    zombie.c\
    
    printf.c umalloc.c\
    README dot-bochsrc *.pl toc.
    .gdbinit.tmpl gdbutil\

dist:
    rm -rf dist
    mkdir dist
    for i in $(FILES); \
```

CS 1550 – xv6 – Done!

Now, you can fire up your ssh to linux.cs.pitt.edu, then:

- **git pull**
- **make qemu-nox**
- **todays_date** (run this inside the xv6)

- You get this slide and code at :
  - https://github.com/maher460/Pitt_CS1550_recitation_materials

CS 1550 – Project

- **Due**: Wednesday, September 19, 2018
- **Late**: Friday, September 21, 2018
  - 10% reduction per late day

CS 1550

Lab 2 – xv6 Introduction

Setup and exercise

Teaching Assistant

Maher Khan

(Slides credited to Henrique Potter)