CS 1550

Week 2

Lab 1 – xv6 Introduction

Setup and exercise

Teaching Assistant

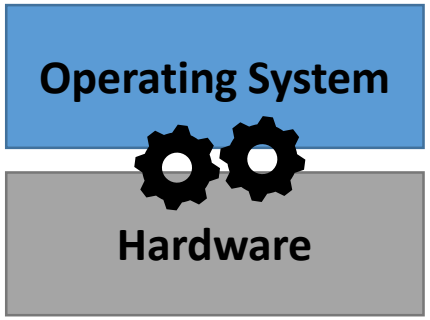
Henrique Pötter

Recitation TA – Office Hours

- Office Hours (SENSQ 6507)
  - Tuesday:
    - 10:00 am to 11:00 am / 6:00 pm to 7:00 pm
  - Wednesday:
    - 10:00 am to 12:00 pm
  - Friday:
    - 10:00 am to 12:00 pm
- Email
  - potter.hp@pitt.edu
- TA Website
  - http://people.cs.pitt.edu/~henriquepotter/

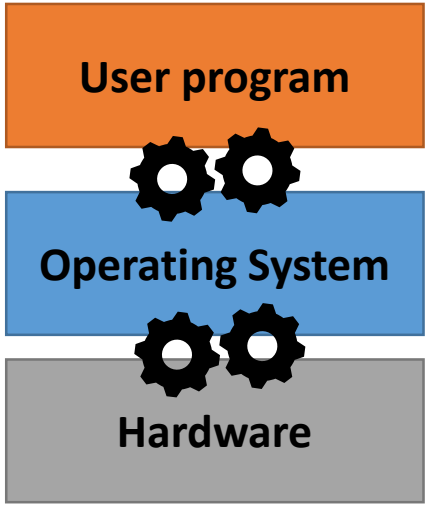
CS 1550 – Kernel Space vs User Space

- Hardware Resources
  - CPU
  - Memory (Address space)
  - I/O devices (Disk, mouse, display, sound, network, etc.)
  - Power and System Management



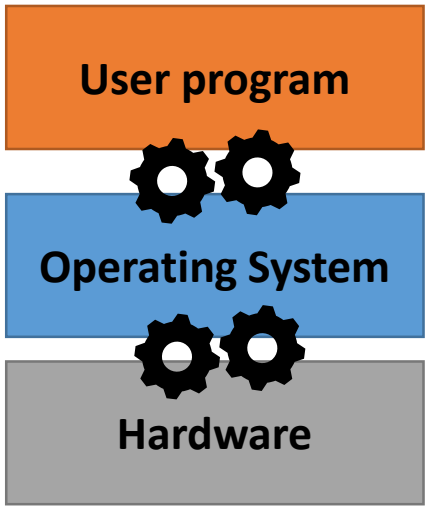
CS 1550 – Kernel Space vs User Space

- Hardware Resources
  - CPU
  - Memory (Address space)
  - I/O devices (Disk, mouse, display, sound, network, etc.)
  - Power and System Management



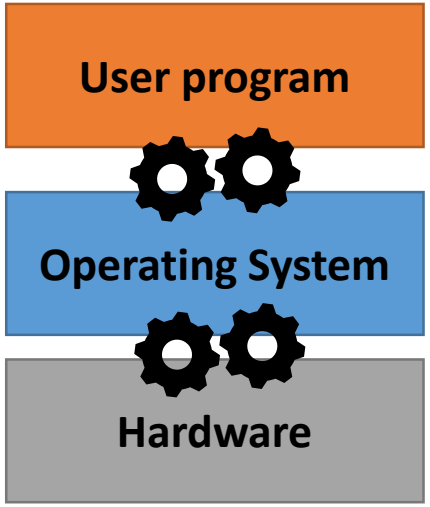
CS 1550 – Kernel Space vs User Space

- **Abstraction**
  - Hides details of different hardware configurations
  - Applications do not need to be tailored for each possible device that might be present on a system



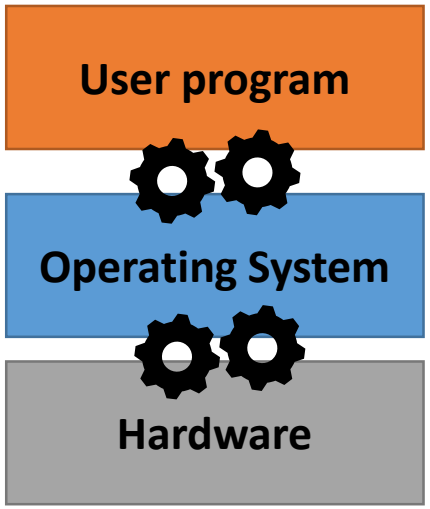
CS 1550 – Kernel Space vs User Space

- **Arbitration**
  - Manages access to shared hardware resources
  - Enables multiple applications to share the same hardware simultaneously



CS 1550 – Kernel Space vs User Space

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

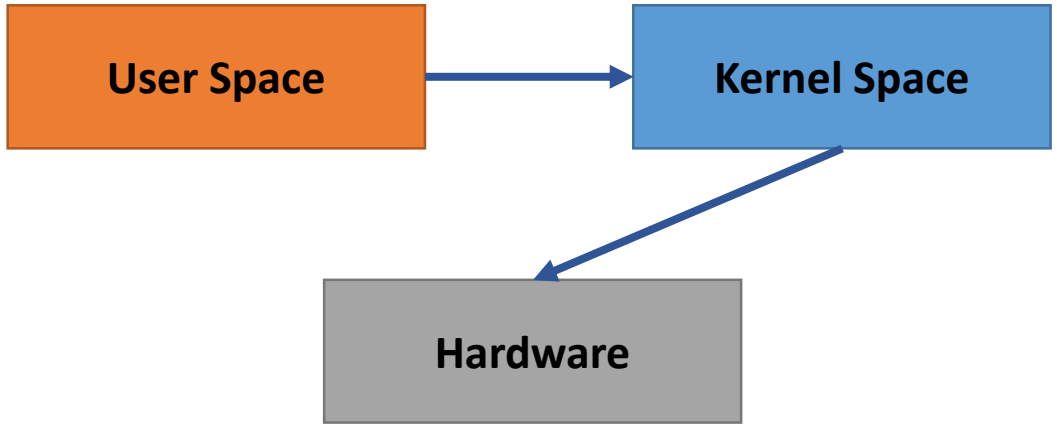


CS 1550 – Kernel Space vs User Space

- 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**

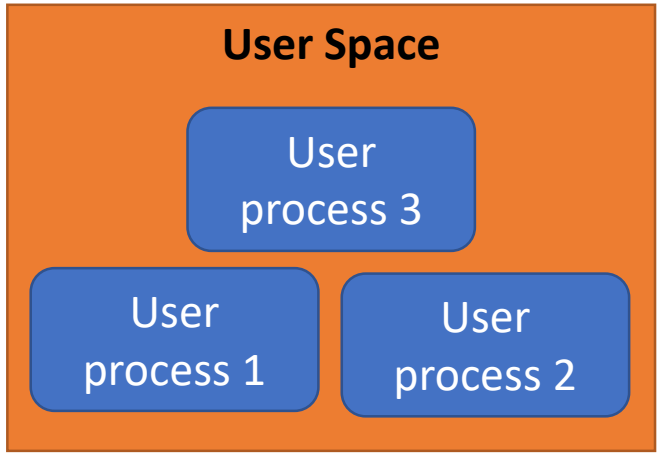
CS 1550 – Kernel Space vs User Space

- 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**



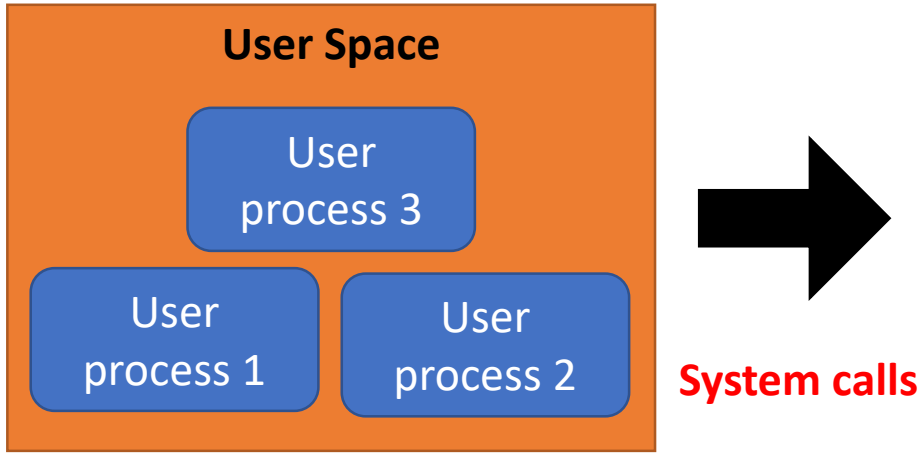
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes must execute system calls to access OS resources and hardware



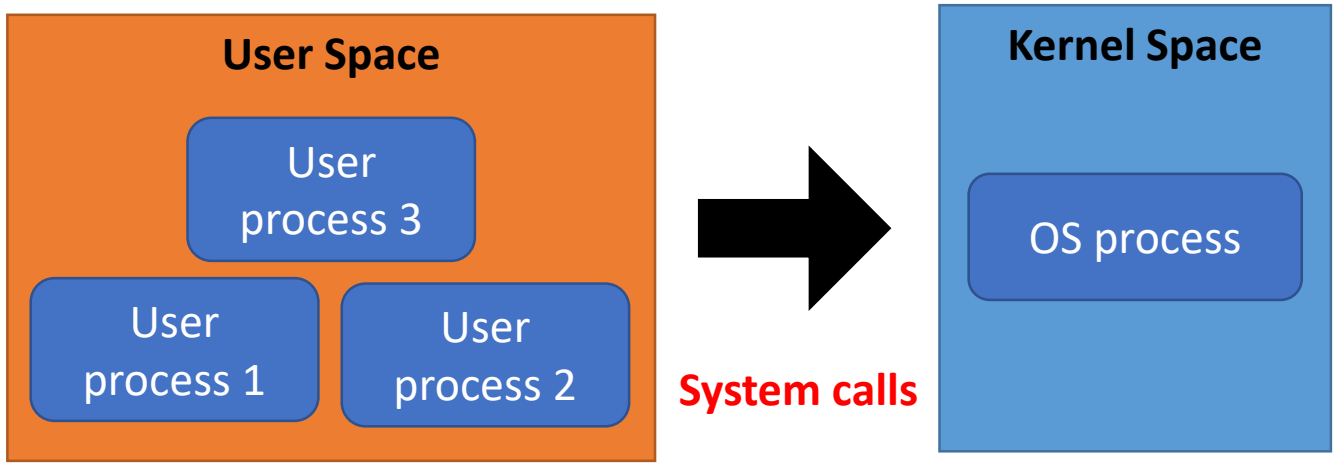
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes must execute system calls to access OS resources and hardware



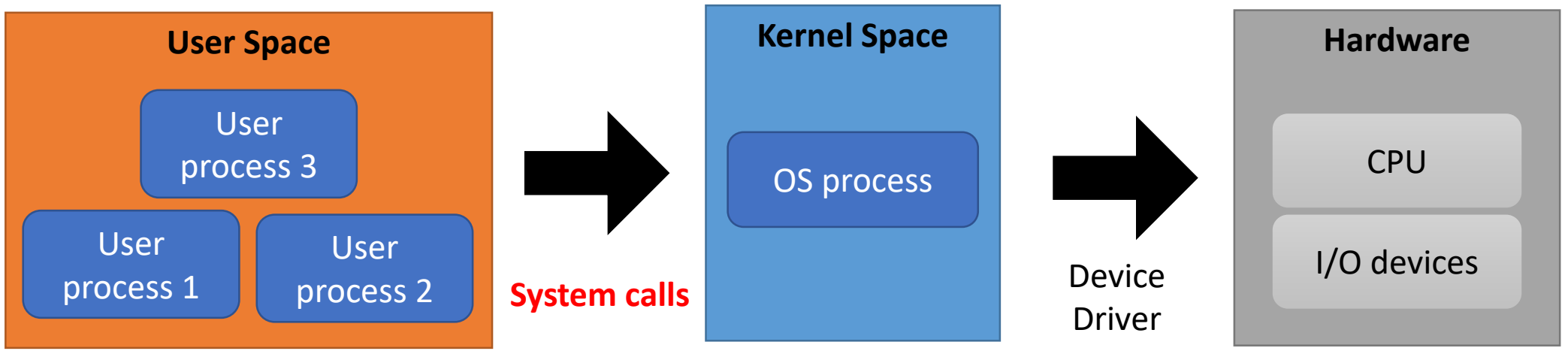
CS 1550 – Kernel Space vs User Space

- **System Call**
  - User processes must execute system calls to access OS resources and hardware



CS 1550 – Kernel Space vs User Space

- **System Call (OS function)**
  - User processes must execute system calls to access OS resources and hardware



System Call
-
exercise

CS 1550 – xv6

- Simple Unix-like **operating system** for teaching, developed in 2006.
  - Provides basic services for running programs
  - https://pdos.csail.mit.edu/6.828/2012/xv6.html?utm_source=dlvr.it&utm_medium=tumblr

xv6 OS

CS 1550 – xv6

- 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

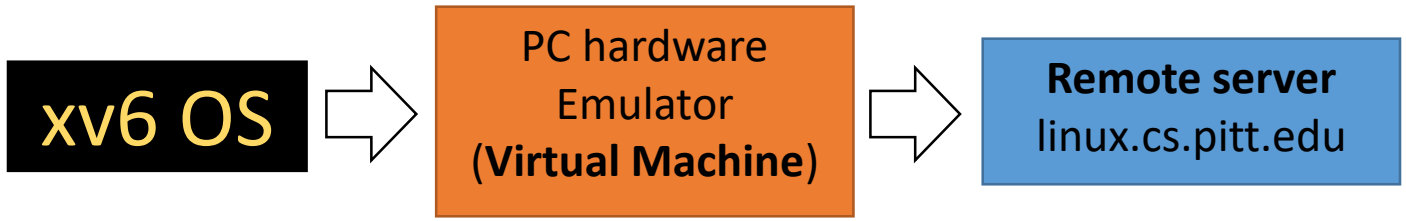
- Compile and Run xv6 in a cs pitt server

CS 1550 – xv6

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

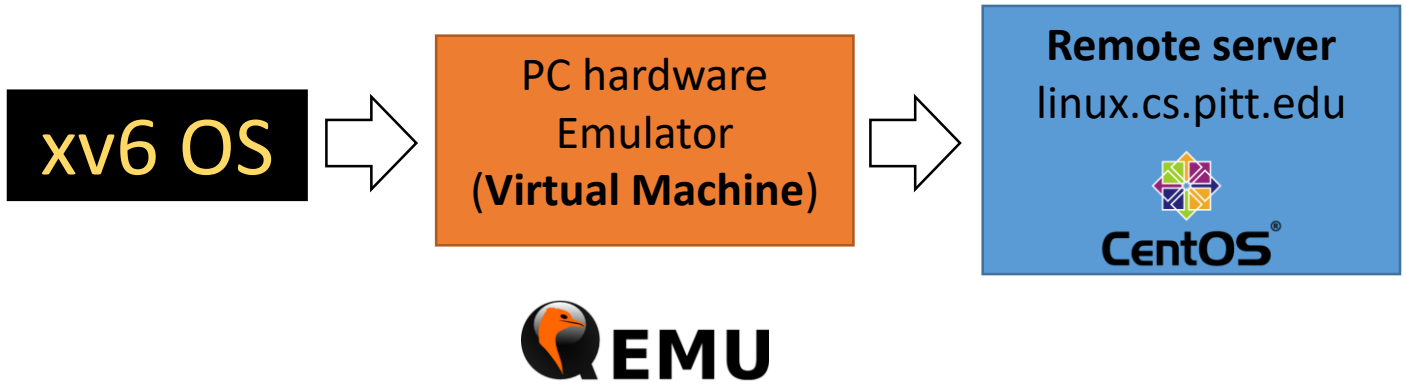
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



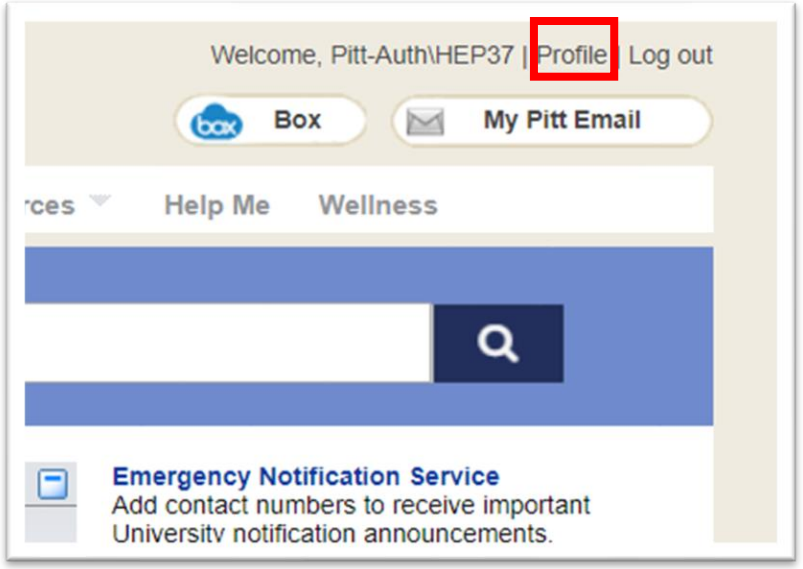
CS 1550 – xv6

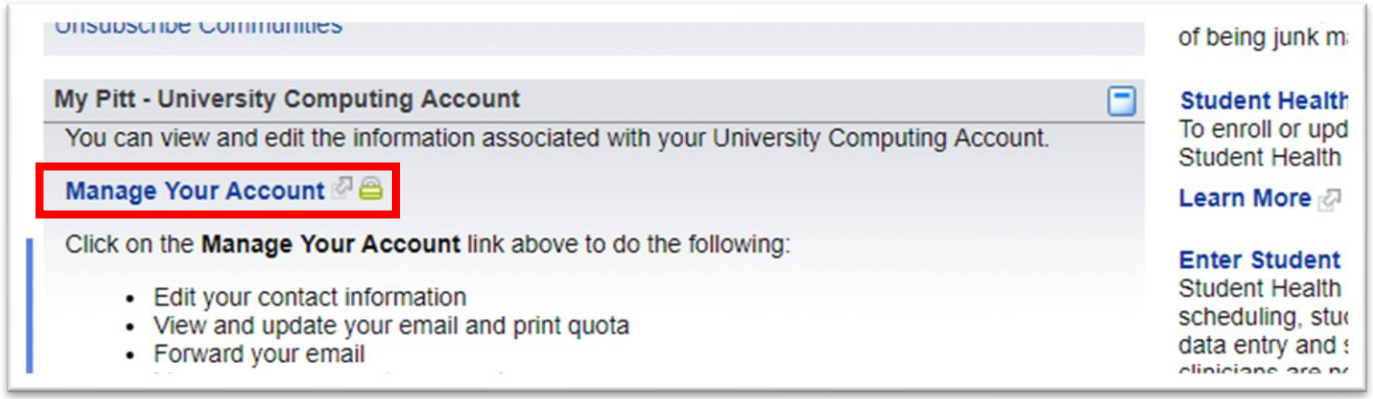
- Compile and Run xv6 in a cs pitt server



CS 1550 – Compile and Run xv6

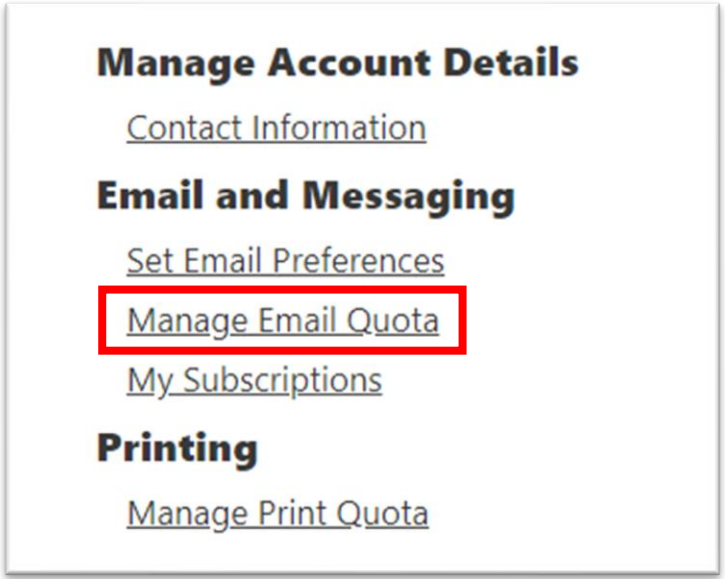
1. Extend disk Quota, if you have less then 500mb free space
   a) Log in to https://my.pitt.edu
   b) Click on "Profile" at the top-right corner of the screen
   c) Click on "Manage Your Account"
   d) Click on "EMAIL & MESSAGING" -> "UNIX QUOTA“
   e) Click on "Increase My UNIX Quota"

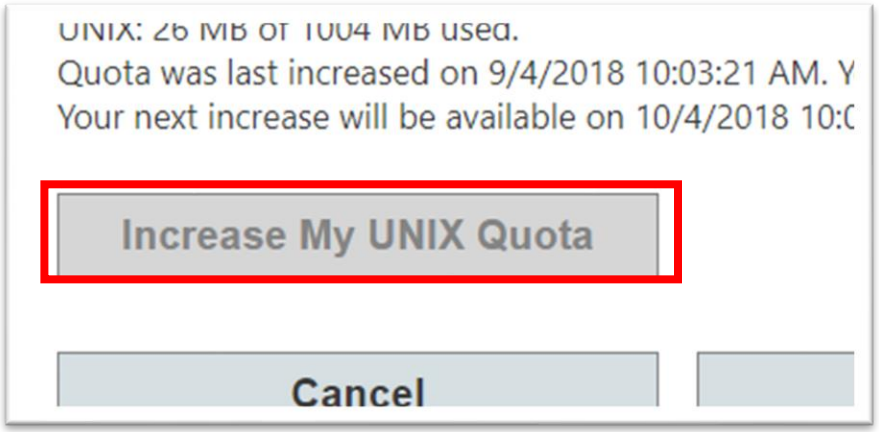




CS 1550 – Compile and Run xv6

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-right corner of the screen
   c) Click on "Manage Your Account"
   d) Click on "EMAIL & MESSAGING" -> "UNIX QUOTA“
   e) Click on "Increase My UNIX Quota"

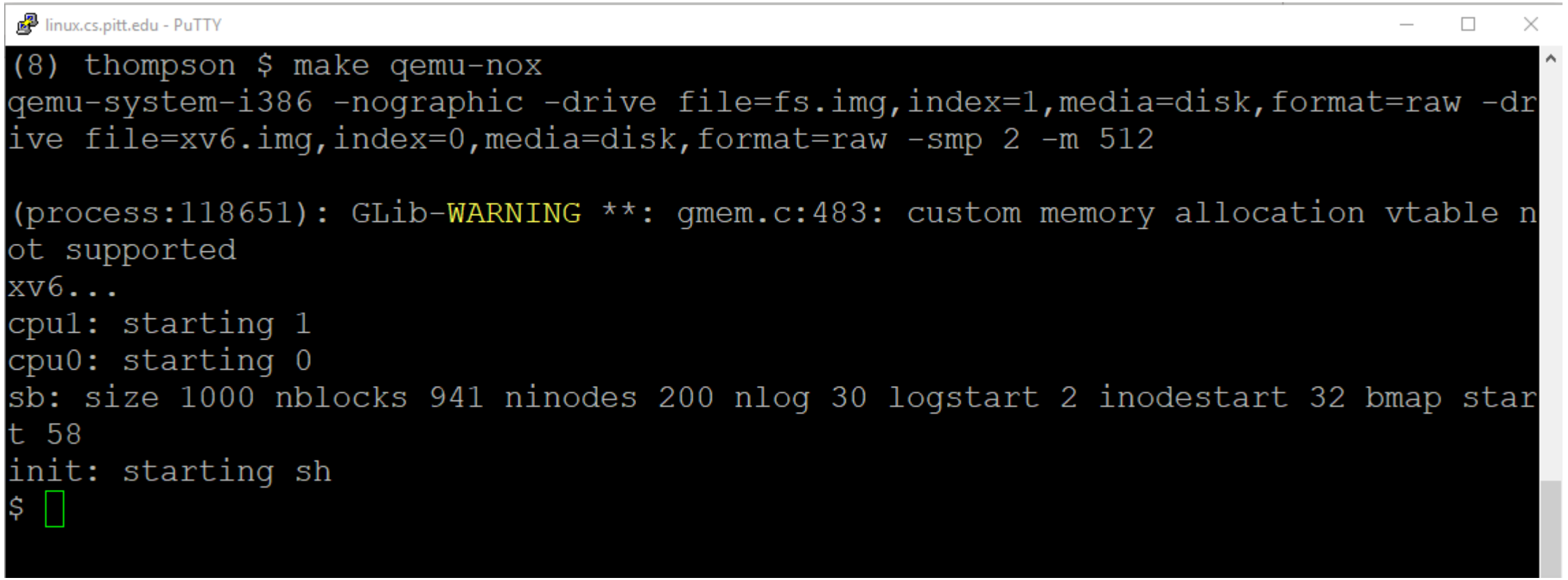




CS 1550 – xv6

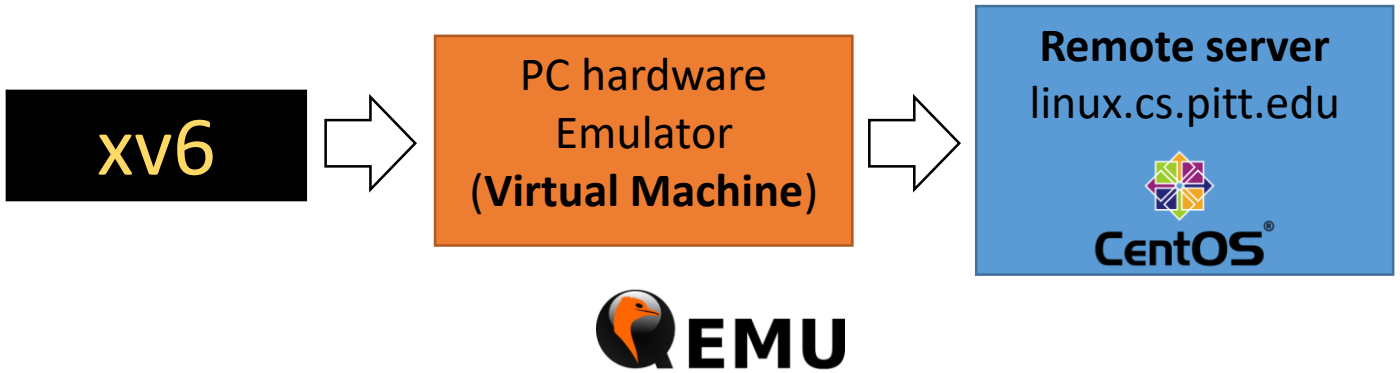
- Log in to **linux.cs.pitt.edu**
  - **ssh** user_name**@linux.cs.pitt.edu**
- Download the xv6 source code from github
  - **git clone git://github.com/mit-pdos/xv6-public.git**
- Get into the cloned xv6 source code folder
  - **cd xv6-public**
- Compile and run the code with
  - **make qemu-nox** → **Compiles** and **run xv6** with **qemu**
  - qemu-nox run the console version of the emulator

CS 1550 – xv6


linux.cs.pitt.edu - PuTTY
(8) thompson $ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -dr
ive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
(process:118651): GLib-WARNING **: gmem.c:483: custom memory allocation vtable n
ot supported
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$

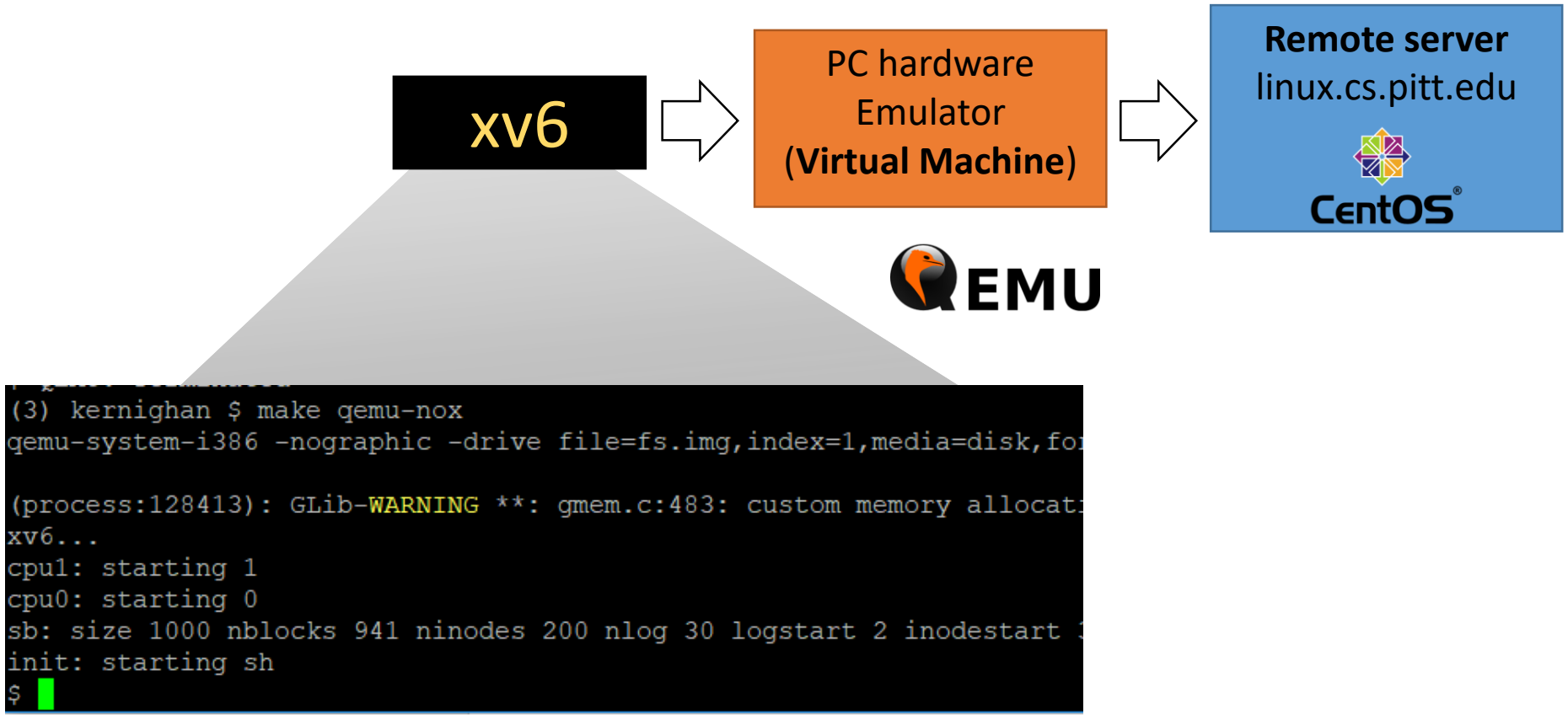
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



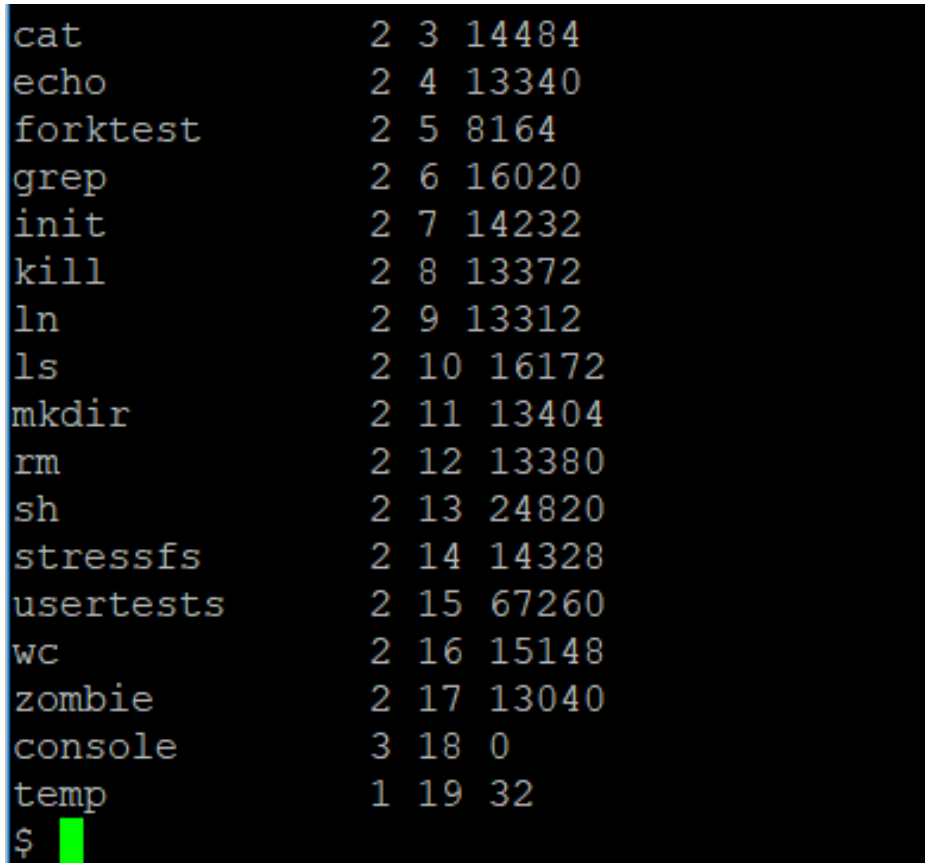
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



CS 1550 – xv6

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



CS 1550 – xv6 – Adding a custom Syscall

- Add a syscall “getday”
- Return the date we hardcoded in the source file

CS 1550 – xv6 – Adding a custom Syscall

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

syscall.h | syscall.c

```
1  // System call numbers
2  #define SYS_fork    1
3  #define SYS_exit    2
4  #define SYS_wait    3
5  #define SYS_pipe    4
6  #define SYS_read    5
7  #define SYS_kill    6
8  #define SYS_exec    7
9  #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid 11
13 #define SYS_sbrk   12
14 #define SYS_sleep  13
15 #define SYS_uptime 14
16 #define SYS_open   15
17 #define SYS_write  16
18 #define SYS_mknod  17
```

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

syscall.h | syscall.c

```
1  // System call numbers
2  #define SYS_fork    1
3  #define SYS_exit    2
4  #define SYS_wait    3
5  #define SYS_pipe    4
6  #define SYS_read    5
7  #define SYS_kill    6
8  #define SYS_exec    7
9  #define SYS_fstat   8
10 #define SYS_chdir   9
11 #define SYS_dup    10
12 #define SYS_getpid 11
13 #define SYS_sbrk   12
14 #define SYS_sleep  13
15 #define SYS_uptime 14
16 #define SYS_open   15
17 #define SYS_write  16
18 #define SYS_mknod  17
```

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
  - [SYS_getday] sys_getday,

- Add
  - extern int sys_getday(void);

syscall.h | syscall.c

```
94  extern int sys_link(void);
95  extern int sys_mkdir(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);
100 extern int sys_sbrk(void);
101 extern int sys_sleep(void);
102 extern int sys_unlink(void);
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106
107 static int (*syscalls[])(void) = {
108 [SYS_fork]    sys_fork,
109 [SYS_exit]    sys_exit,
110 [SYS_wait]    sys_wait,
```

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**

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

syscall.h | syscall.c | sysproc.c

```
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  }
```

CS 1550 – xv6 – Adding a custom Syscall

- Afterwards we define the interface for user programs to call
  - Open usys.S

- Add
  - SYSCALL(getday)

syscall.h | syscall.c | sysproc.c | usys.S

```
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)
```

CS 1550 – xv6 – Adding a custom Syscall

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

syscall.h | syscall.c | sysproc.c | usys.S | user.h

```
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, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
```

CS 1550 – xv6 – Adding a custom Syscall

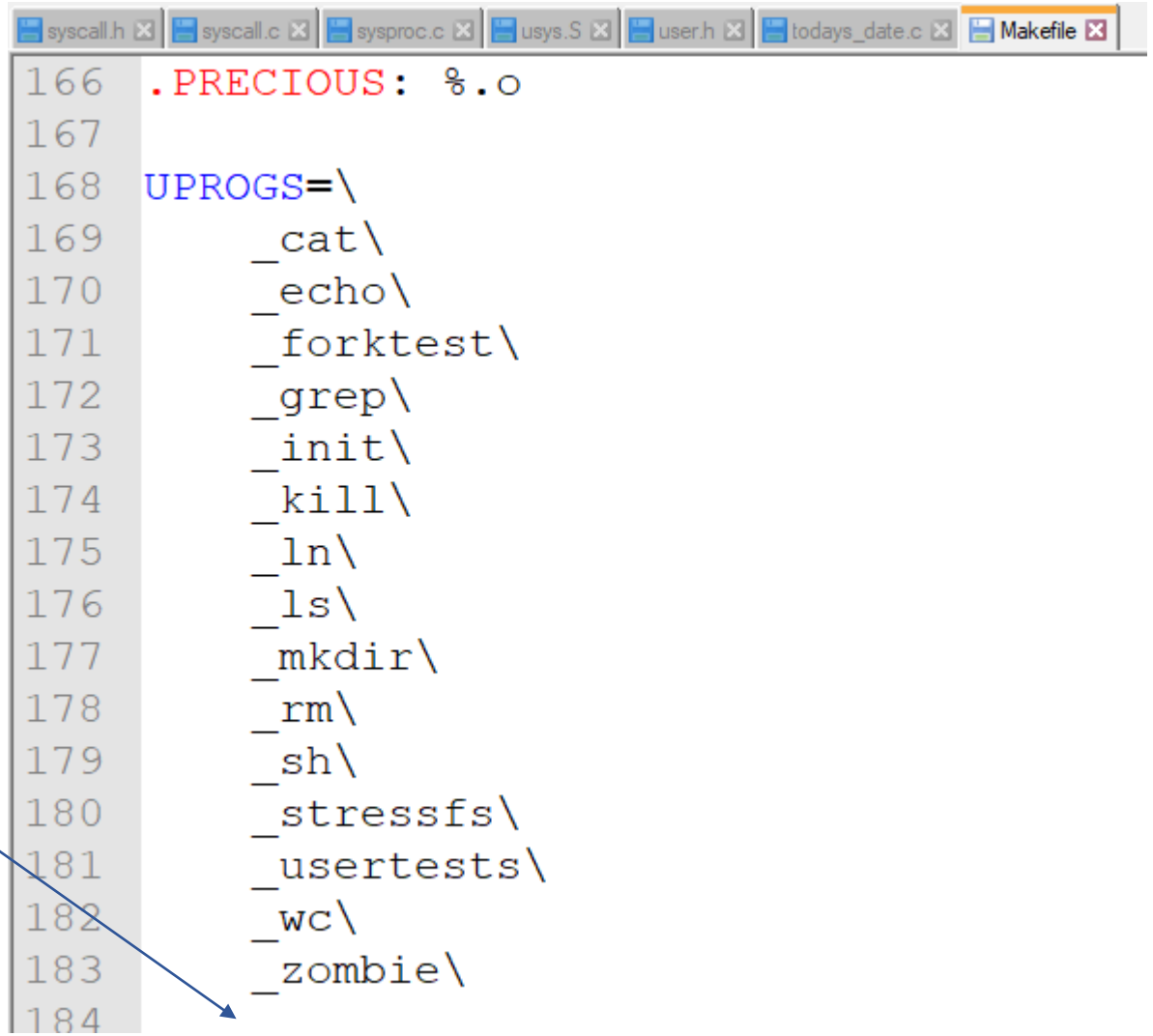
- Example user program
  - todays_date.c

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

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

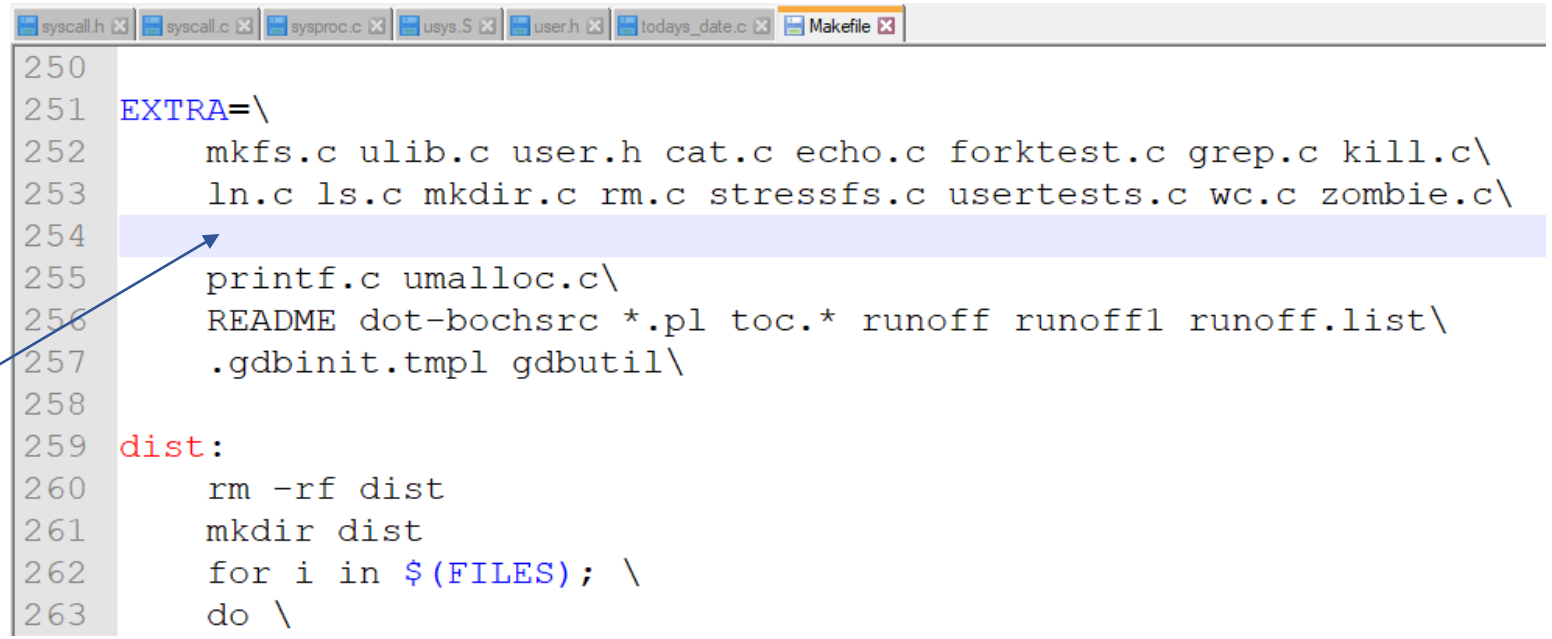
CS 1550 – xv6 – Adding a custom Syscall

- Adding a user program
  - Open makefile
- Add
  - _todays_date\



CS 1550 – xv6 – Adding a custom Syscall

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

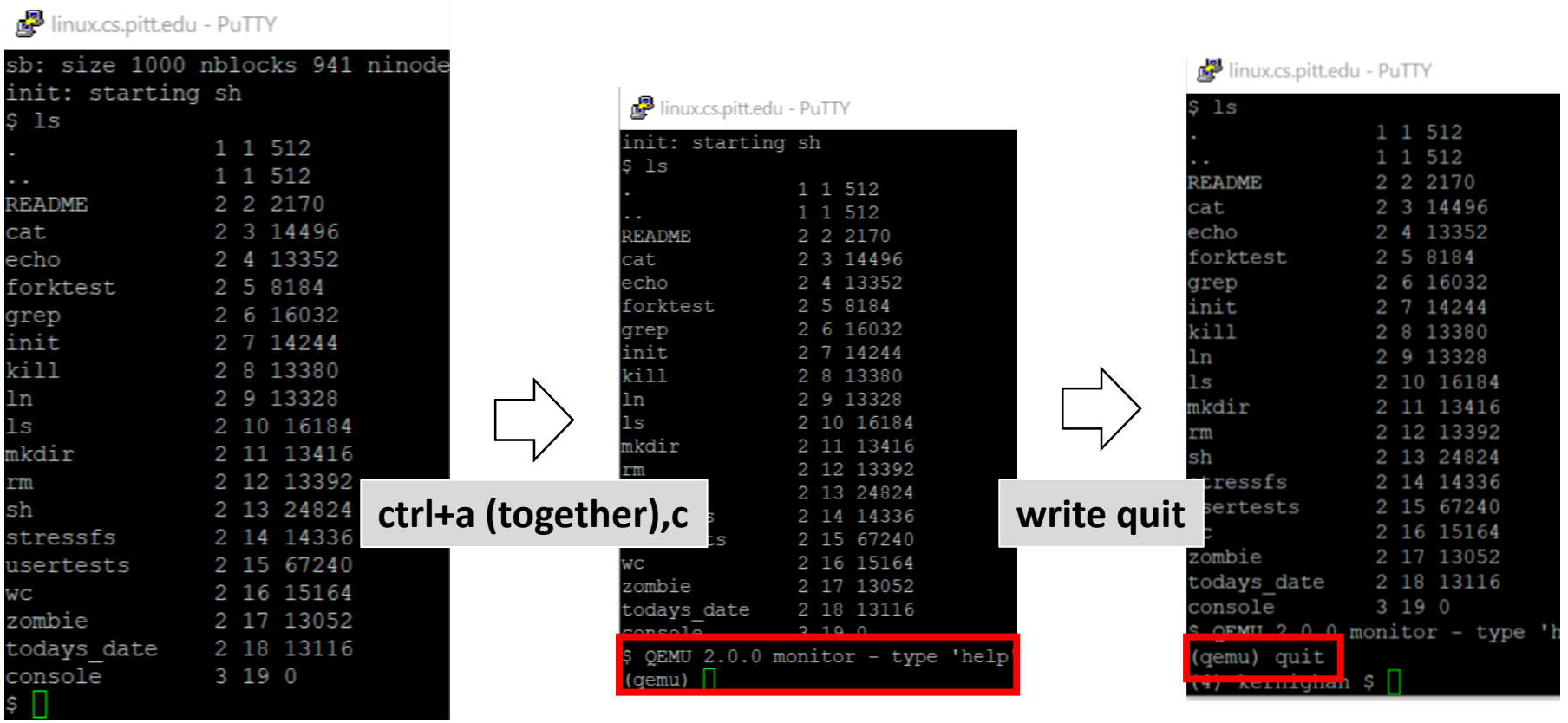


CS 1550 – xv6 – Adding a custom Syscall

- Adding a user program
  - Open makefile
- and add
  - todays_date.c\
- Finally execute
  - make qemu-nox
- Run the program by calling it directly within xv6
  - todays_date

CS 1550 – xv6 – Adding a custom Syscall

- To exit xv6 (after executing make qemu-nox)
  - Press **ctrl+a (together)** and then press **c** and write **quit**



CS 1550 – xv6 Lab 1

- We need to worry about two things:
  - How to count syscalls?
  - Implement the method to return the number of syscalls

CS 1550 – xv6 exercise hints

- We need to worry about two things:
  - How to count syscalls?
  - Implement the method to return the number of syscalls

CS 1550 – xv6 exercise hints

- Syscall calls will need a variable to hold the counting values
  - Where to write this data structure?
    - Which file holds processes metadata? proc.c
  - Which data structure?
    - Each syscall have an id, which could be used as?
    - Which basic data structure uses indices for element positions?
  - Important method can be found in syscall.c
    - syscall(void) -> Is called every time any syscall is called

CS 1550 – xv6 exercise hints

```
void
syscall(void)
{
  int num;
  struct proc *curproc = myproc();

  num = curproc->tf->eax;
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    curproc->tf->eax = syscalls[num]();
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

The system call numbers match the entries in the syscalls array, a table of function pointers

CS 1550 – xv6 exercise hints

- Implementing **getcount**
  - Specify the method and its **id** in **syscall.h**
  - Specify extern method and pointer
    - **syscall.c**
  - Where to implement int **sys_getcount(void)**?
    - **sysproc.c**
  - Add SYSCALL(getcount)
    - **usys.S**
  - getcount.c
  - Modify proc.c, proc.h according to your method of counting.
    - Declare counting array?
    - Initialize counting array?
  - Makefile

CS 1550 – xv6 exercise hints

- Submit to GradeScope the files that you have modified within the source code of xv6.
- You should modify the following files only:
  - syscall.h
  - syscall.c
  - user.h
  - usys.S
  - proc.h
  - proc.c
  - sysproc.c
  - Makefile

CS 1550 – Reminder

- **Lab 1**
  - **Due**: Friday, 01/28 @ 11:59pm
- Email
  - potter.hp@pitt.edu
- TA Website
  - http://people.cs.pitt.edu/~henriquepotter/