CS 1550

Lab 2 – xv6 Introduction

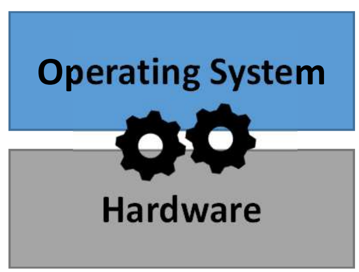
Setup and exercise

Teaching Assistant

Xiaoyu (Veronica) Liang

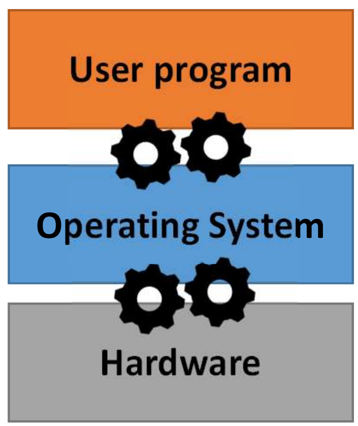
CS 1550 – Kernel Space vs User Space

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



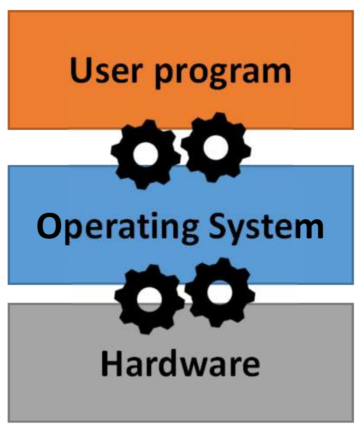
CS 1550 – Kernel Space vs User Space

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



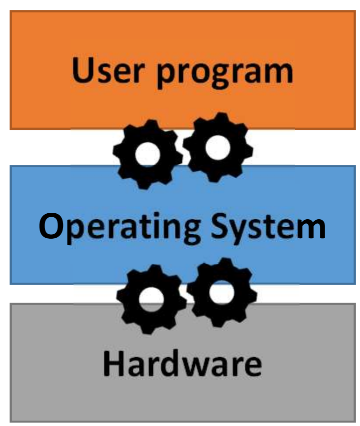
CS 1550 – Kernel Space vs User Space

- 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)

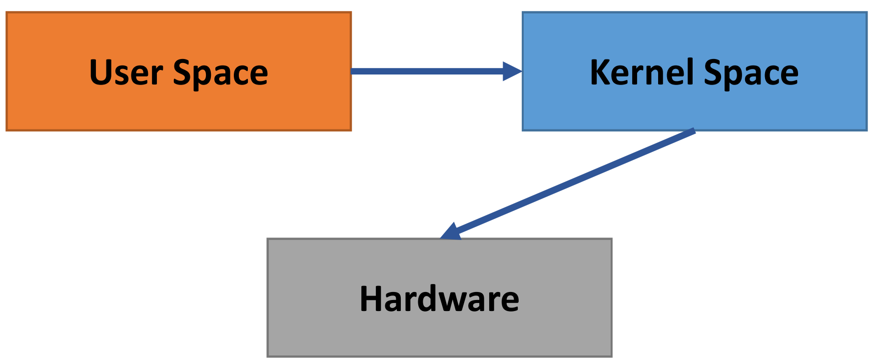


CS 1550 – Kernel Space vs User Space

- 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**

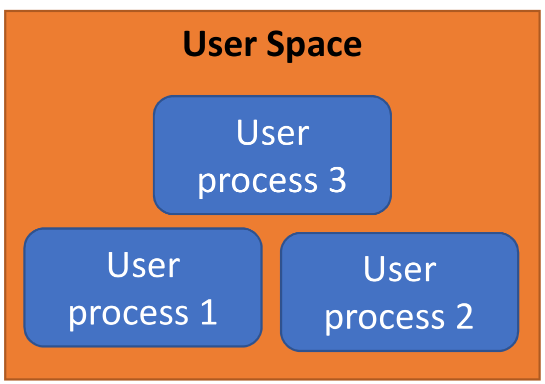
CS 1550 – Kernel Space vs User Space

- 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**



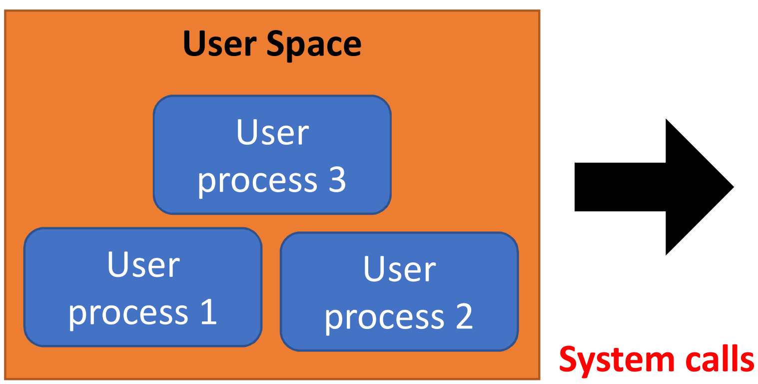
CS 1550 – Kernel Space vs User Space

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



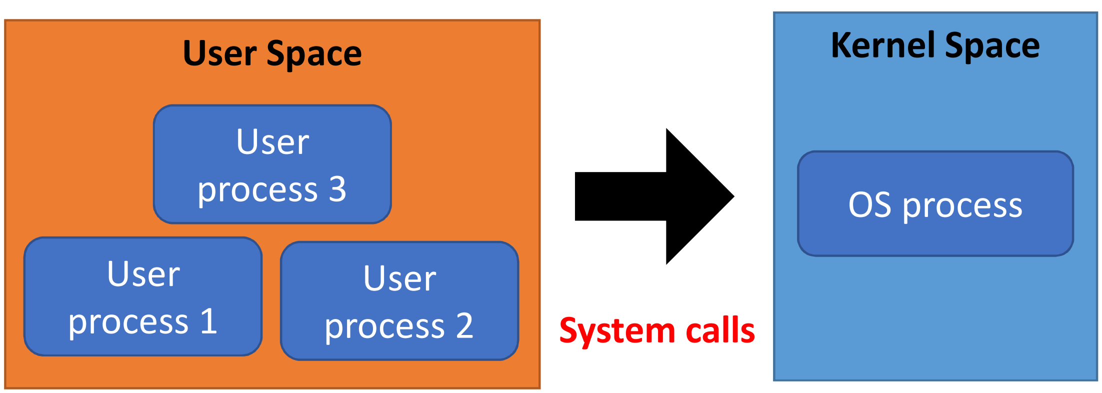
CS 1550 – Kernel Space vs User Space

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



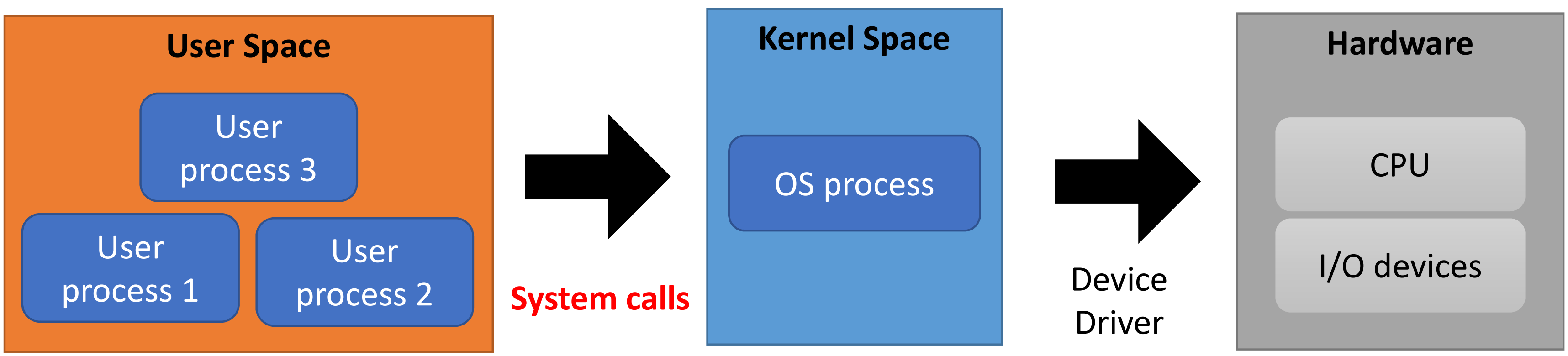
CS 1550 – Kernel Space vs User Space

- **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



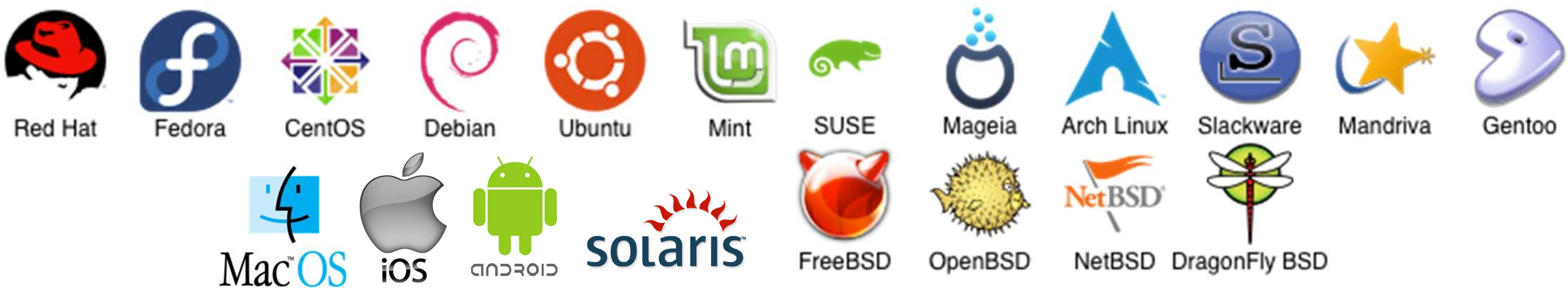
System Call
-
exercise

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

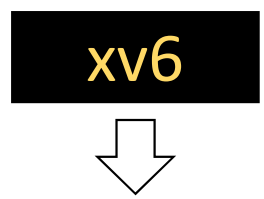


CS 1550 – xv6

- 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?



Run where?

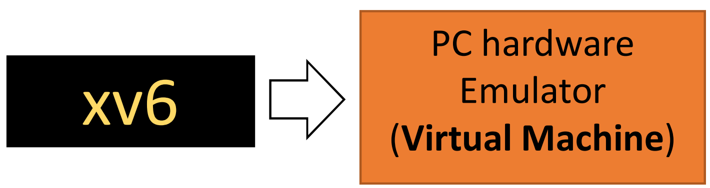
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server

xv6

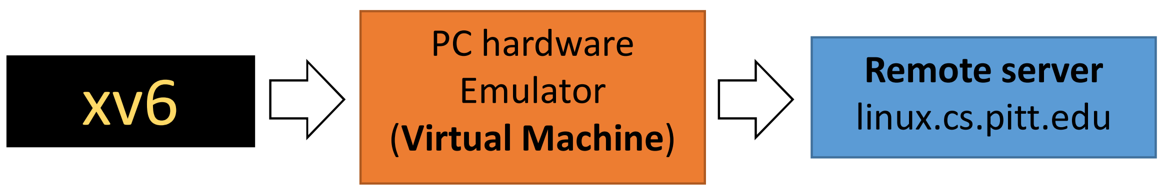
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



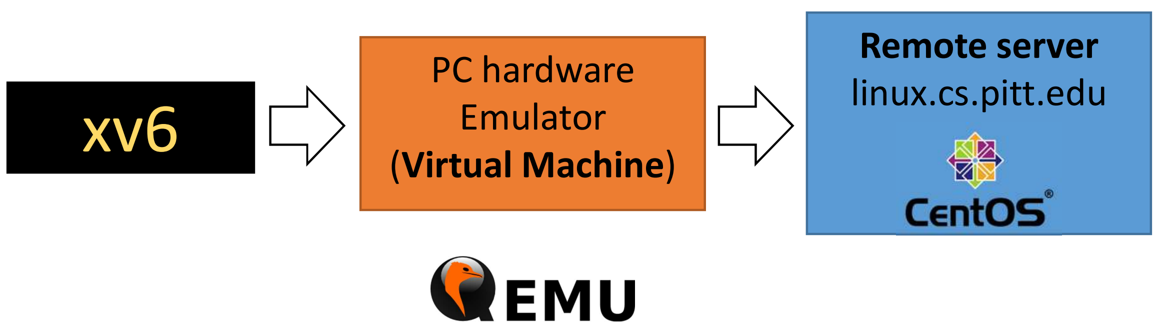
CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



CS 1550 – xv6

- Compile and Run xv6 in a cs pitt server



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

- 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

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

- Got 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**

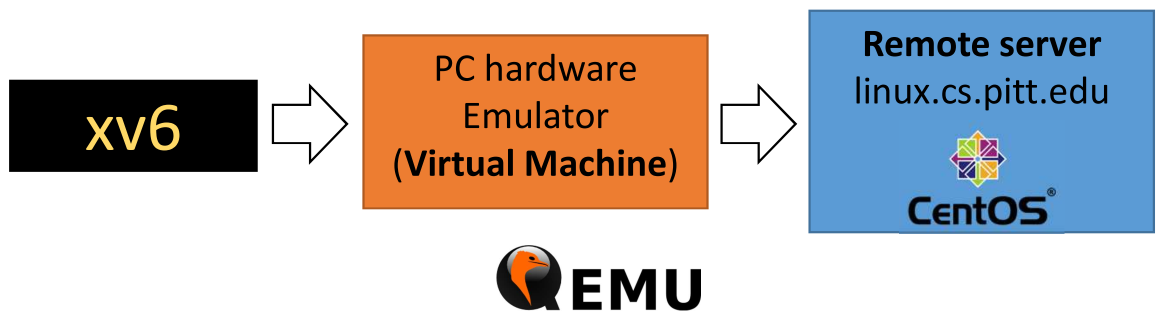
CS 1550 – xv6

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
$
```

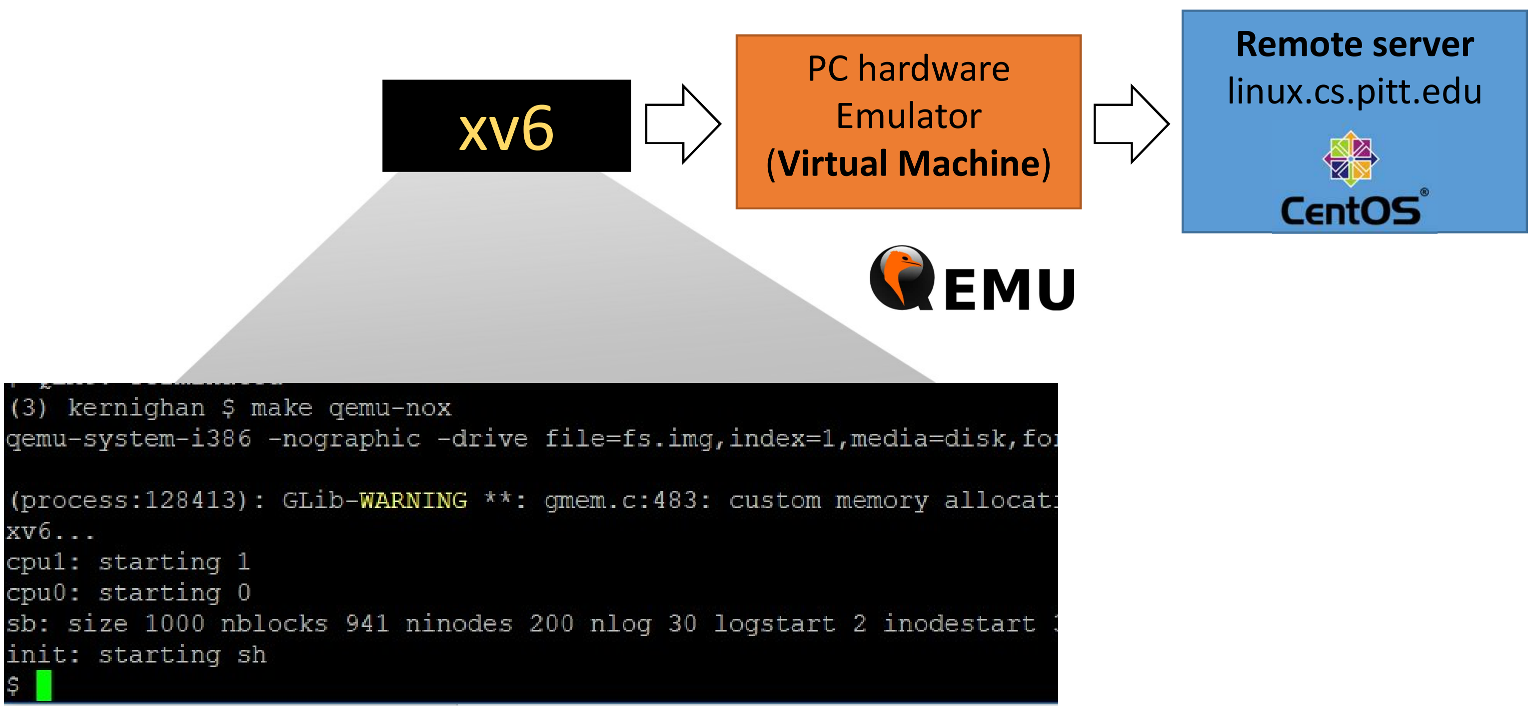
CS 1550 – xv6

- 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**

linux.cs.pitt.edu - PuTTY

```
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2327
cat            2 3 14508
echo           2 4 13364
forktest       2 5 8184
grep           2 6 16044
init           2 7 14252
kill           2 8 13396
ln             2 9 13336
ls             2 10 16192
mkdir          2 11 13424
rm             2 12 13400
sh             2 13 24844
stressfs       2 14 14352
usertests      2 15 67284
wc             2 16 15172
zombie         2 17 13060
console        3 18 0
$ 
```

CS 1550 – xv6 – Adding a custom Syscall

- First, we need to define our new call and its number at
  - **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
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#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

```
// 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
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#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,

```
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,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[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);

```c
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,
```

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**

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;
}
```

```
#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)

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

```
#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)
```

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

```c
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
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 an user program
  - Open makefile

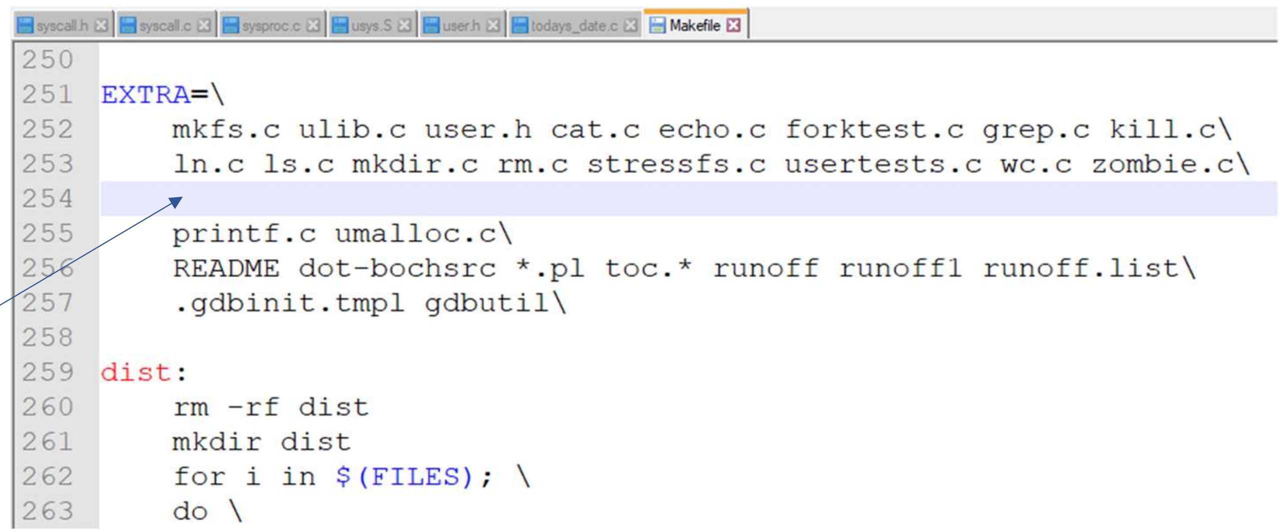
- Add
  - _todays_date\

```
.PRECIOUS: %.o

UPROGS=\
	_cat\
	_echo\
	_forktest\
	_grep\
	_init\
	_kill\
	_ln\
	_ls\
	_mkdir\
	_rm\
	_sh\
	_stressfs\
	_usertests\
	_wc\
	_zombie\

```

CS 1550 – xv6 – Adding a custom Syscall

- Adding an user program
  - Open makefile

- and also add
  - todays_date.c\



CS 1550 – xv6 exercise hints

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

CS 1550 – xv6 exercise hints

- Syscall calls will need variable to hold the counting values
  - Where to write this data structure?
    - Which file holds process 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

```c
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;
  }
}
```

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**

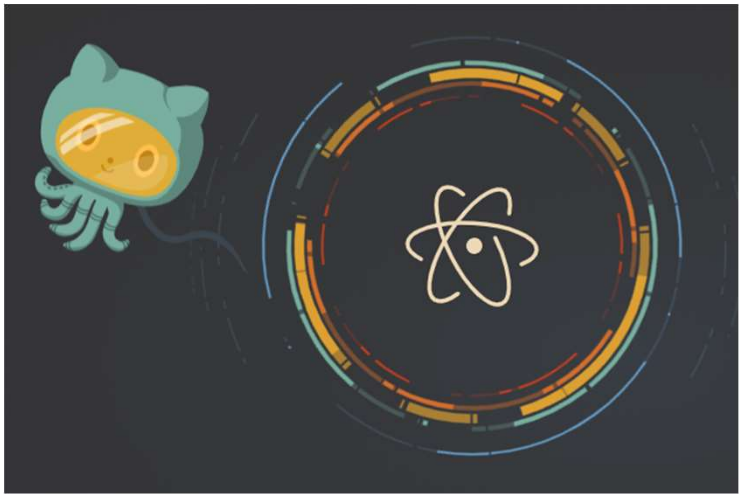
CS 1550 – xv6 exercise hints

- Adding an user program
  - Open makefile

- Add
  - _getcount\

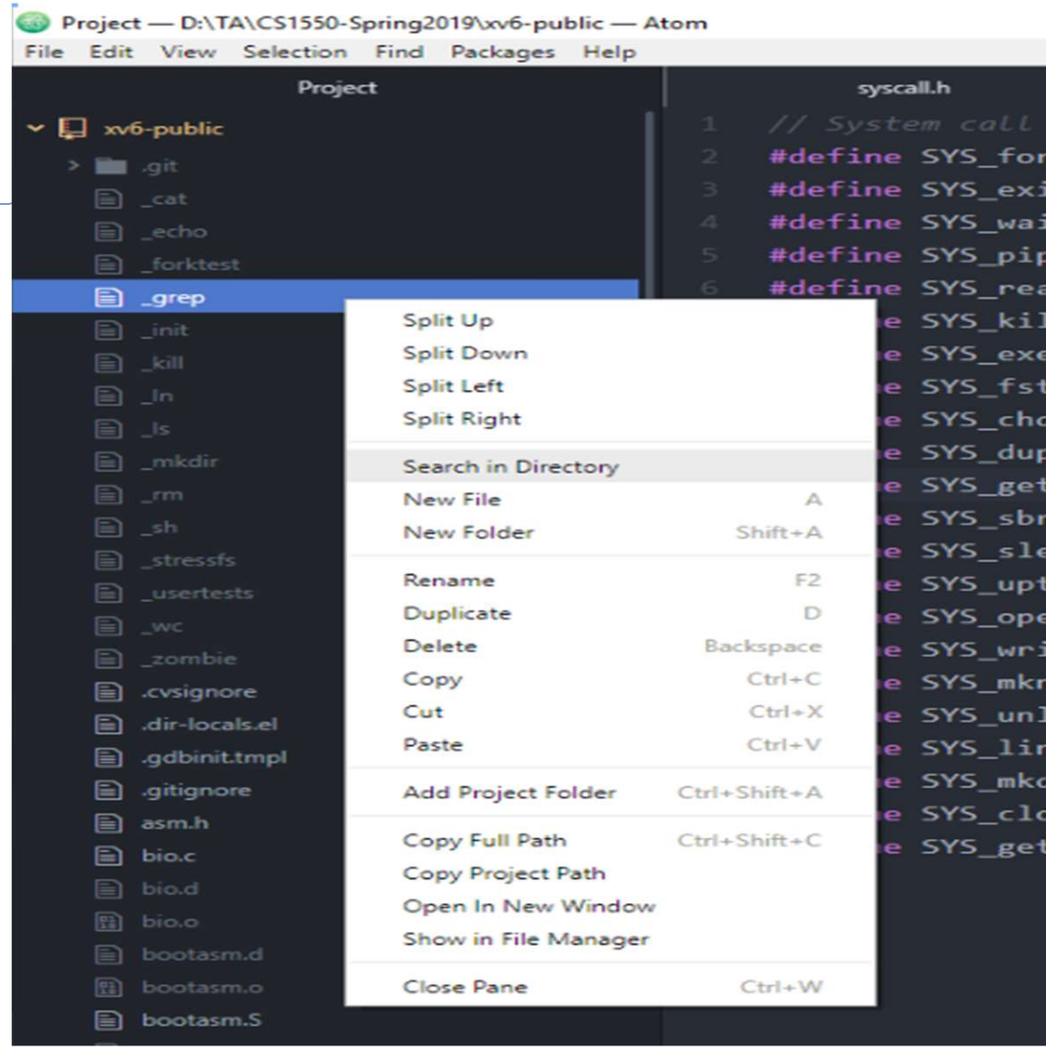
Atom io

- Good coding tool
- Easily search for variables names in any file inside a folder
- Lots of plugins



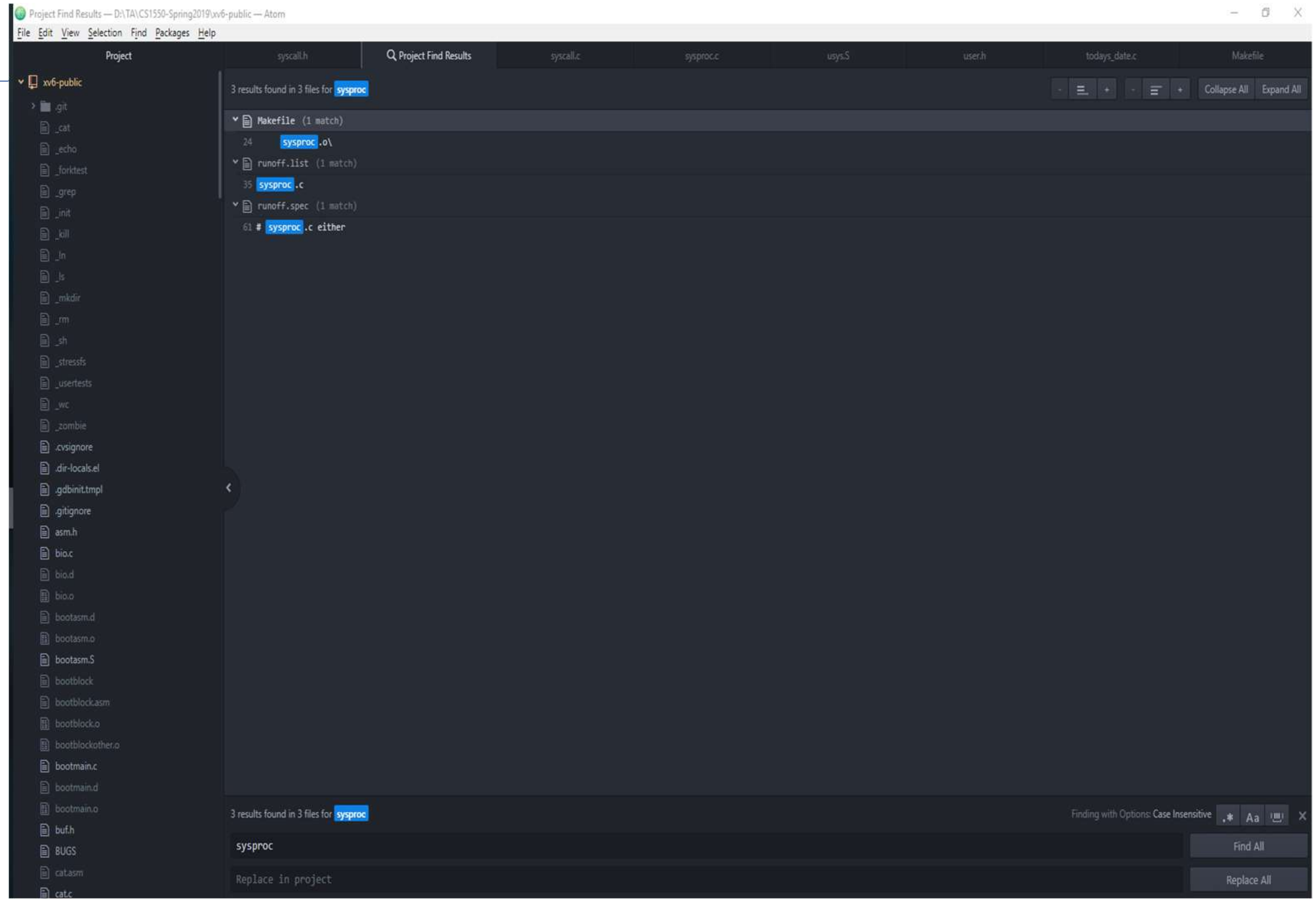
Atom io

- Search for variables in any file from a Directory



Atom io

- Search for variables in any file from a Directory



CS 1550 – Reminder

- **Lab 1**
  - **Due**: Friday, February 1st, 2019
  - Leave the files in your linux.cs.pitt.edu user account folders **for now**
- **Project 1**
  - **Due**: Monday, February 4th, 2019
- Slides Available:
  - http://people.cs.pitt.edu/~xil160/CS1550_Spring2019