



# CS 1550

Lab 1 – xv6 Introduction

Setup and exercise

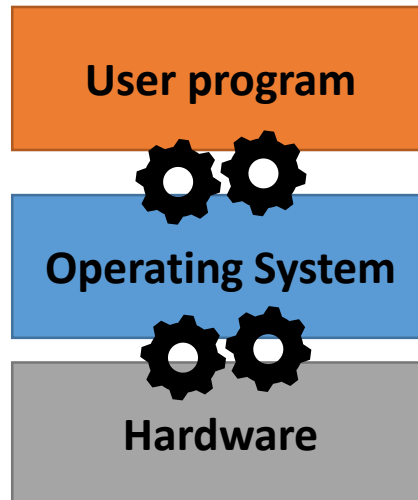
Teaching Assistant

Xiaoyu (Veronica) Liang

# CS 1550 – Kernel Space vs User Space

---

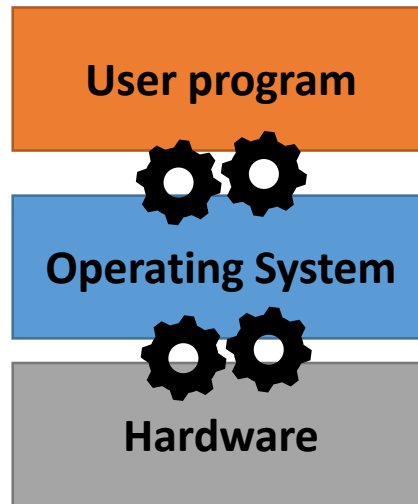
- Hardware Resources
  - CPU
  - Memory (Address space)
  - I/O devices (Disk, mouse, video card, sound, network, etc.)
  - Power and System Management



# CS 1550 – Kernel Space vs User Space

---

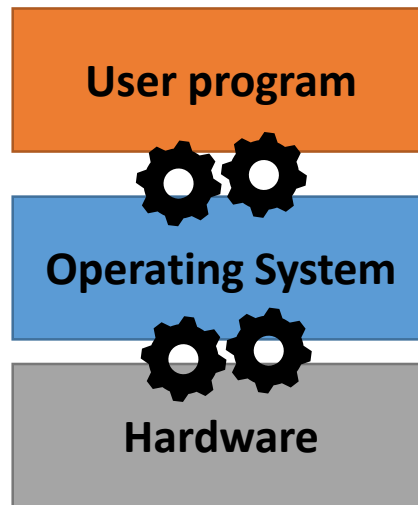
- Abstraction
  - Hides details of different hardware configurations
  - Applications do not need to be tailored for each possible device that might be present on a system
- Arbitration
  - Manages access to shared hardware resources
  - Enables multiple applications to share the same hardware simultaneously
- OS is **just** another **software**



# CS 1550 – Kernel Space vs User Space

---

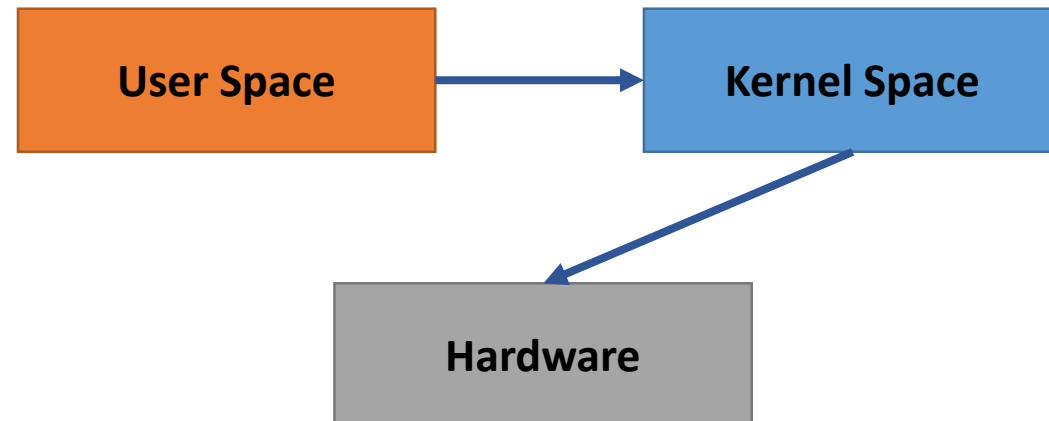
- OS is **just** another **software**
- User applications should not change the kernel(OS software)



# 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**

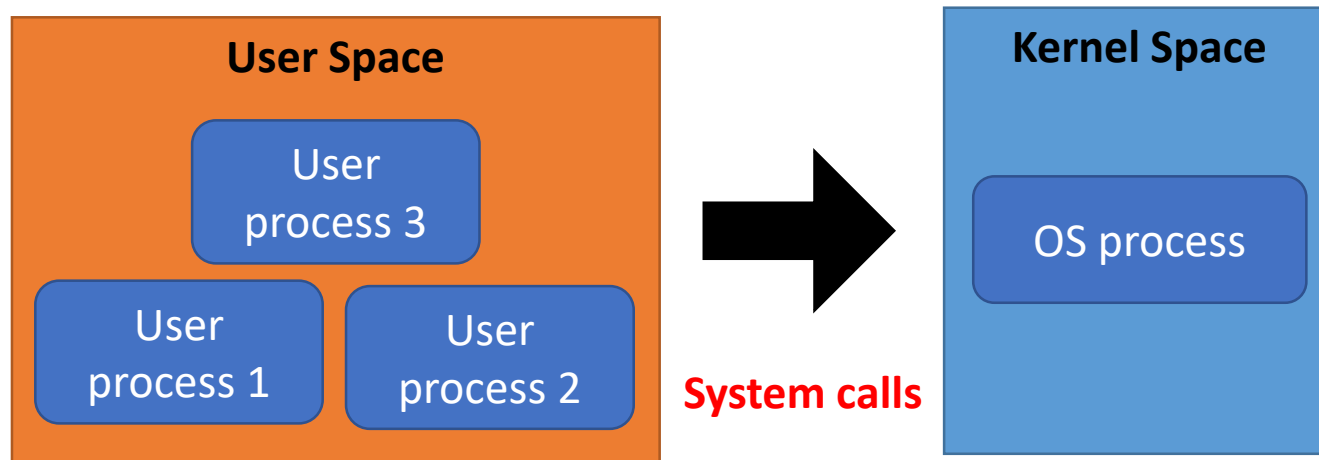


# CS 1550 – Kernel Space vs User Space

---

- **System Call**

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

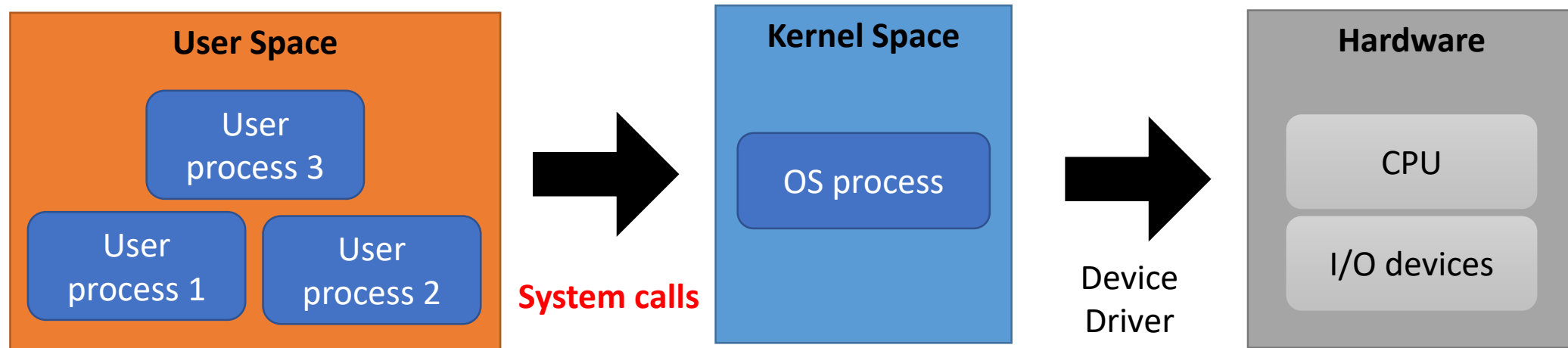


# CS 1550 – Kernel Space vs User Space

---

- **System Call (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**, developed in 2006.
- Provides basic services to running programs

# CS 1550 – xv6

---

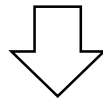
- Has a **subset of traditional** system calls
  - **fork()** Create process
  - **exit()** Terminate current process
  - **wait()** Wait for a child process
  - **kill(pid)** Terminate process pid
  - **getpid()** Return current process's id
  - **sleep(n)** Sleep for n time units
  - **exec(filename,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

# CS 1550 – xv6

---

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

xv6

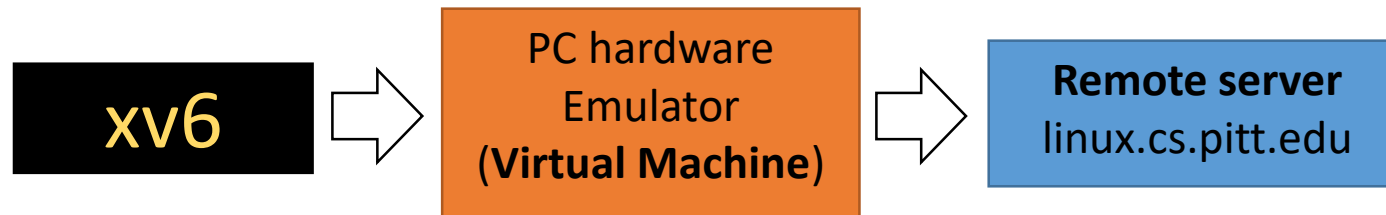


Run where?

# CS 1550 – xv6

---

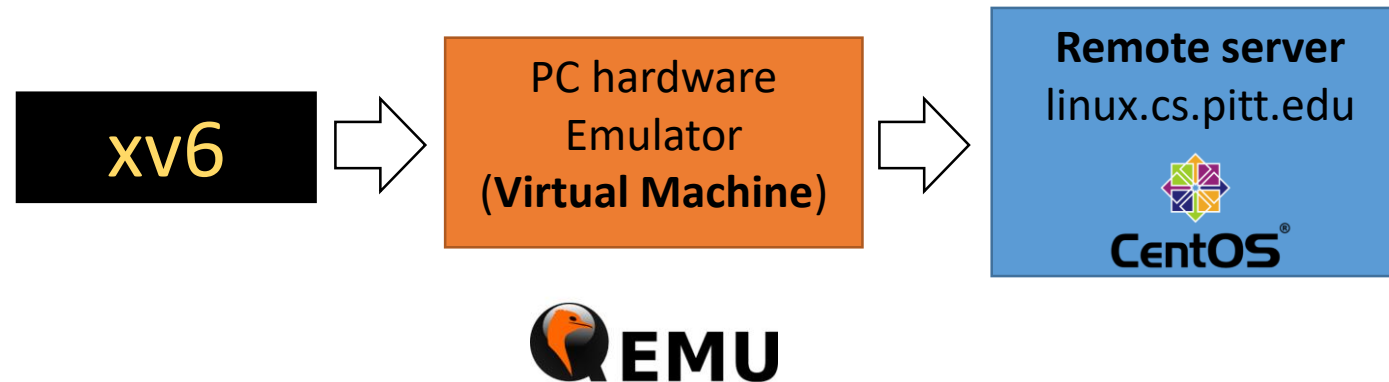
- Compile and Run xv6 in a cs pitt server



# 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 than 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**
- 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
  - 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 -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512

(process:118651): GLib-WARNING **: gmem.c:483: custom memory allocation vtable not supported
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$
```



# CS 1550 – xv6

- Once in xv6 you can call “ls”
- Will see the entire list of shell commands available to you.
- The list is very small.

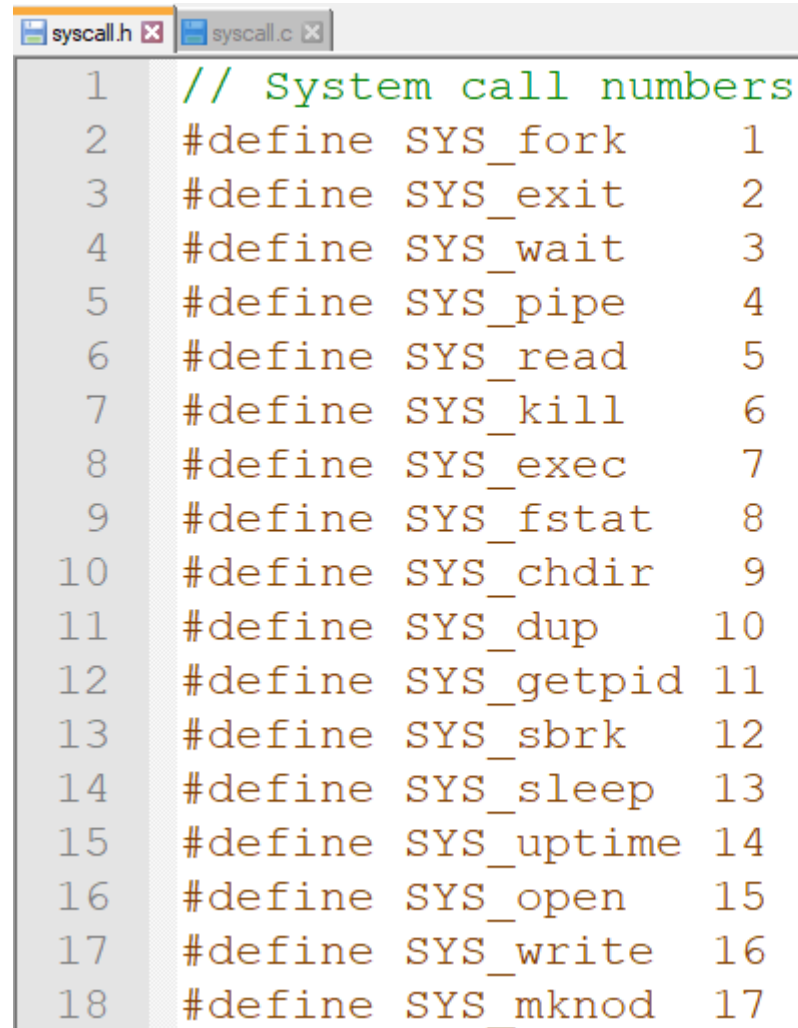
```
linux.cs.pitt.edu - PuTTY
sb: size 1000 nblocks 941 ninoc
t 58
init: starting sh
$ ls
.          1 1 512
..         1 1 512
README    2 2 2327
cat        2 3 14508
echo       2 4 13364
forktest   2 5 8184
grep       2 6 16044
init       2 7 14252
kill       2 8 13396
ln         2 9 13336
ls         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

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

# CS 1550 – xv6 – Adding a custom Syscall

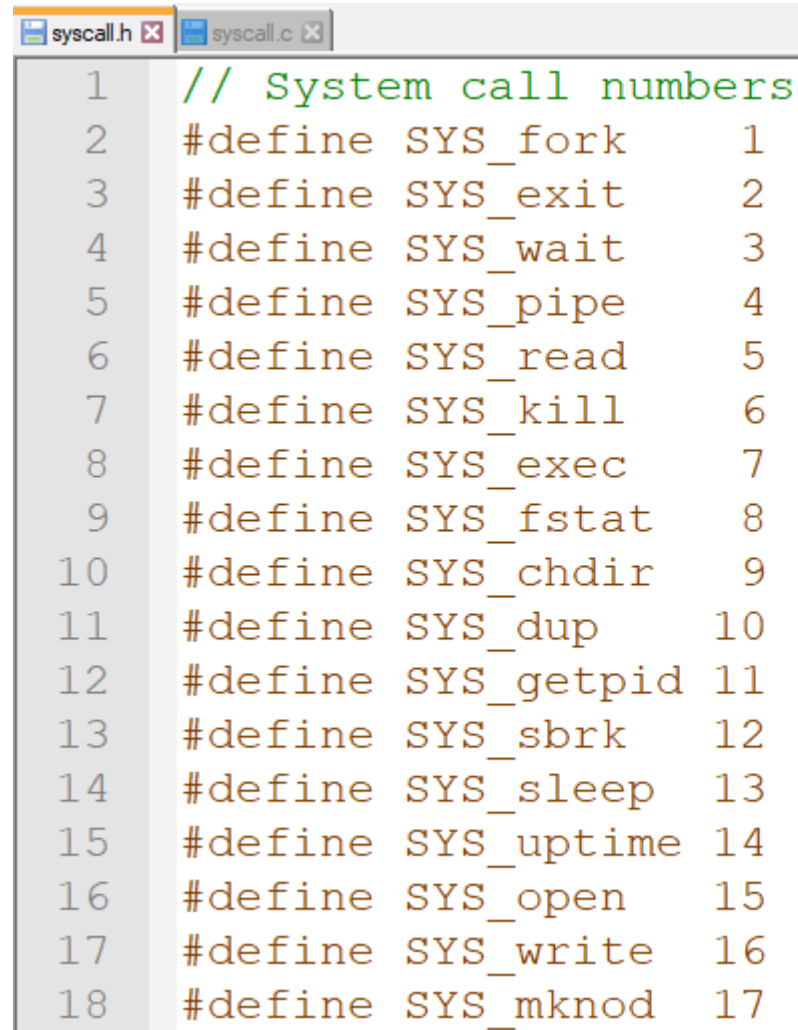
- First, we need to define our new call and its number at
  - **syscall.h**



```
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`



```
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);

```
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**

# 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 x syscall.c x sysproc.c x
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 }
```

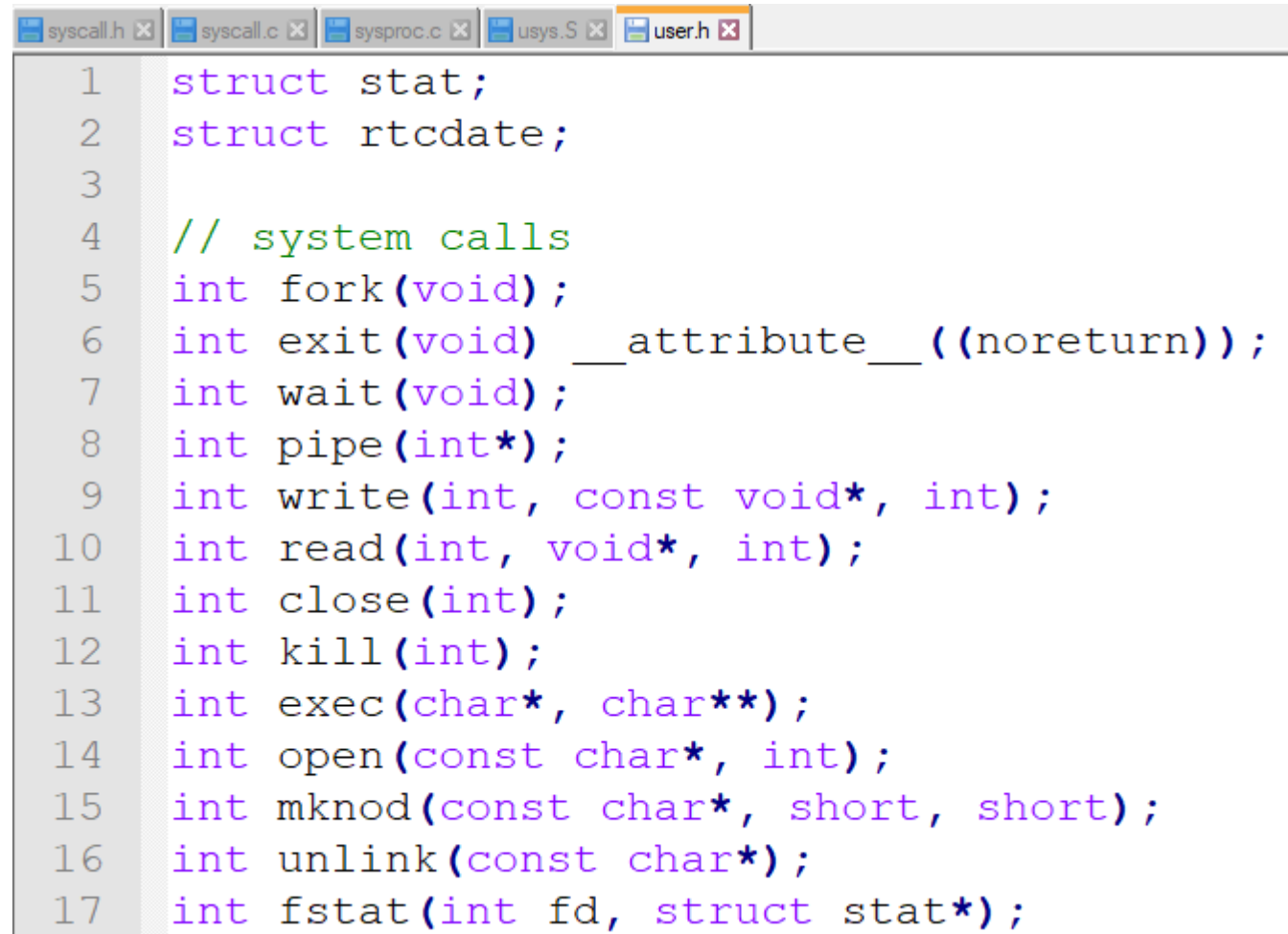
# CS 1550 – xv6 – Adding a custom Syscall

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

```
syscall.h x syscall.c x sysproc.c x usys.S x
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);



```
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 an user program
  - Open makefile
- Add
  - `_today's_date\`

```
166 .PRECIOUS: % .o
167
168 UPROGS=\
169     _cat\
170     _echo\
171     _forktest\
172     _grep\
173     _init\
174     _kill\
175     _ln\
176     _ls\
177     _mkdir\
178     _rm\
179     _sh\
180     _stressfs\
181     _usertests\
182     _wc\
183     _zombie\
184
```

# CS 1550 – xv6 – Adding a custom Syscall

- Adding an user program
  - Open makefile
- and also add
  - `today's_date.c\`

```
250
251 EXTRA=\
252 mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\
253 ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\
254
255 printf.c umalloc.c\
256 README dot-bochsrc *.pl toc.* runoff runoffl runoff.list\
257 .gdbinit.tmpl gdbutil\
258
259 dist:
260 rm -rf dist
261 mkdir dist
262 for i in $(FILES); \
263 do \
```

# 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

---

```
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 lab1 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 lab1 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, 09/20 @ 11:59pm

- Slides Available:

- [http://people.cs.pitt.edu/~xil160/CS1550\\_Fall2019/](http://people.cs.pitt.edu/~xil160/CS1550_Fall2019/)