OS Project 2

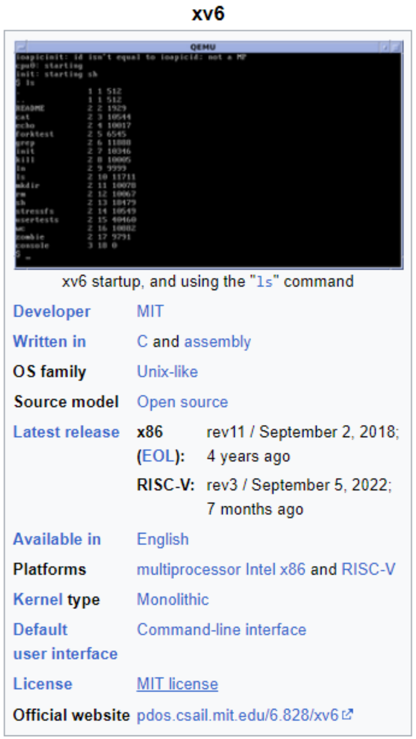
System Call on xv6 and User Defined Shell

Prof. Yongtae Kim

Computer Science and Engineering
Kyungpook National University

What is xv6 Operating System?

- **xv6 was developed at MIT for OS education**
  - It is run on multiprocessor Intel x86 and RISC-V systems
  - It contains important Unix concepts and constructs
  - It is an open-source project and can be compiled using the GNU C compiler
  - It is normally run using the QEMU emulator, which is a free and open-source hardware emulator



xv6

```
QEMU
ioapicinit: id isn't equal to ioapicid: not a MP
cpu0: starting
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 1929
cat            2 3 10544
echo           2 4 10017
forktest       2 5 6545
grep           2 6 11888
init           2 7 10346
kill           2 8 10005
ln             2 9 9999
ls             2 10 11711
mkdir          2 11 10078
rm             2 12 10067
sh             2 13 18479
stressfs       2 14 10549
usertests      2 15 40460
wc             2 16 10882
zombie         2 17 9791
console        3 18 0
$ _
```

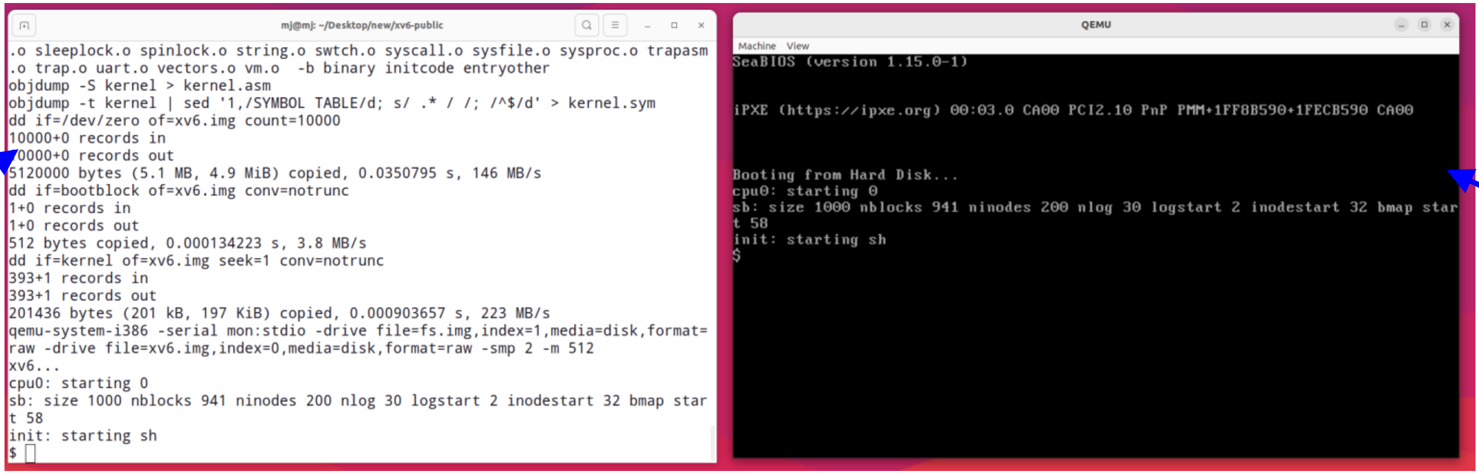
xv6 startup, and using the "ls" command

Developer	MIT
Written in	C and assembly
OS family	Unix-like
Source model	Open source
Latest release	x86 (EOL): rev11 / September 2, 2018; 4 years ago RISC-V: rev3 / September 5, 2022; 7 months ago
Available in	English
Platforms	multiprocessor Intel x86 and RISC-V
Kernel type	Monolithic
Default user interface	Command-line interface
License	MIT license
Official website	pdos.csail.mit.edu/6.828/xv6

- **Before you start (important!)**
  - xv6 runs on only x86 or RISC-V processor architecture
  - You must use a x86-based computer, not on apple silicon (M1 or M2) Mac
  - You are recommended to use VirtualBox with fresh installed Ubuntu 22.04
  - You can use the VirtualBox used in OS Project 1 (new kernel equipped Ubuntu) and may see some glitches

Install QEMU and build xv6

- **Install packages QEMU**
  - `$ sudo apt-get update`
  - `$ sudo apt-get install git build-essential qemu-kvm gdb vim -y`
- **Download and build xv6**
  - `$ git clone https://github.com/mit-pdos/xv6-public.git`
  - `$ cd xv6-public`
  - `$ make qemu` // build and run xv6 on QEMU command
  - You will see a new pop-up window for QEMU and you can type a xv6 command and see the results on either the original Linux terminal or QEMU terminal



Linux Terminal

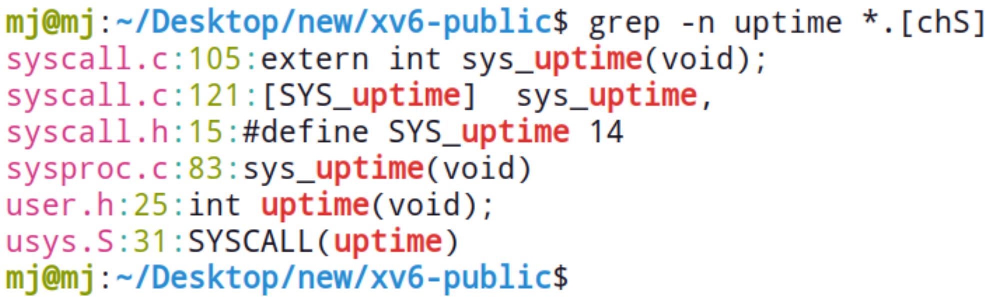
```
.o sleeplock.o spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o trapasm
.o trap.o uart.o vectors.o vm.o  -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0350795 s, 146 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000134223 s, 3.8 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
393+1 records in
393+1 records out
201436 bytes (201 kB, 197 KiB) copied, 0.000903657 s, 223 MB/s
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=
raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$
```

QEMU Terminal

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00



Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$
```

System Call (1)

- **Making a new system call**
  - New system call is to get the current UTC time and return it to the user program
  - **`$ grep -n uptime *.[chS]`**

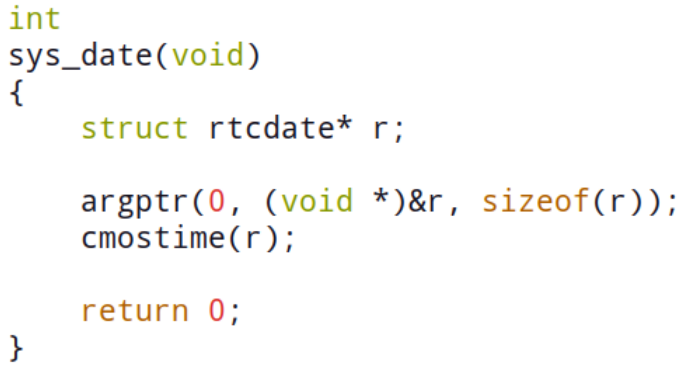


```
mj@mj:~/Desktop/new/xv6-public$ grep -n uptime *.[chS]
syscall.c:105:extern int sys_uptime(void);
syscall.c:121:[SYS_uptime]  sys_uptime,
syscall.h:15:#define SYS_uptime 14
sysproc.c:83:sys_uptime(void)
user.h:25:int uptime(void);
usys.S:31:SYSCALL(uptime)
mj@mj:~/Desktop/new/xv6-public$
```

  - You will need to modify syscall.c, syscall.h, sysproc.c, user.h, usys.S for new system call
  - In addition, you will need to write date.c for the new system call testing

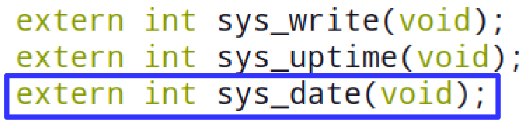
System Call (2)

- **Making date system call**
  - Writing data system call code in sysproc.c

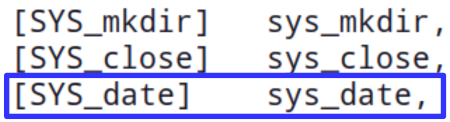


```
int
sys_date(void)
{
    struct rtcdate* r;

    argptr(0, (void *)&r, sizeof(r));
    cmostime(r);

    return 0;
}
```

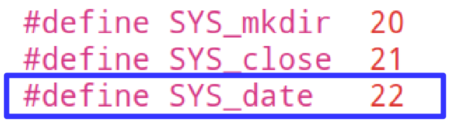
  - Adding the date system call in syscall.c and syscall.h



```
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_date(void);
```



```
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_date]    sys_date,
```

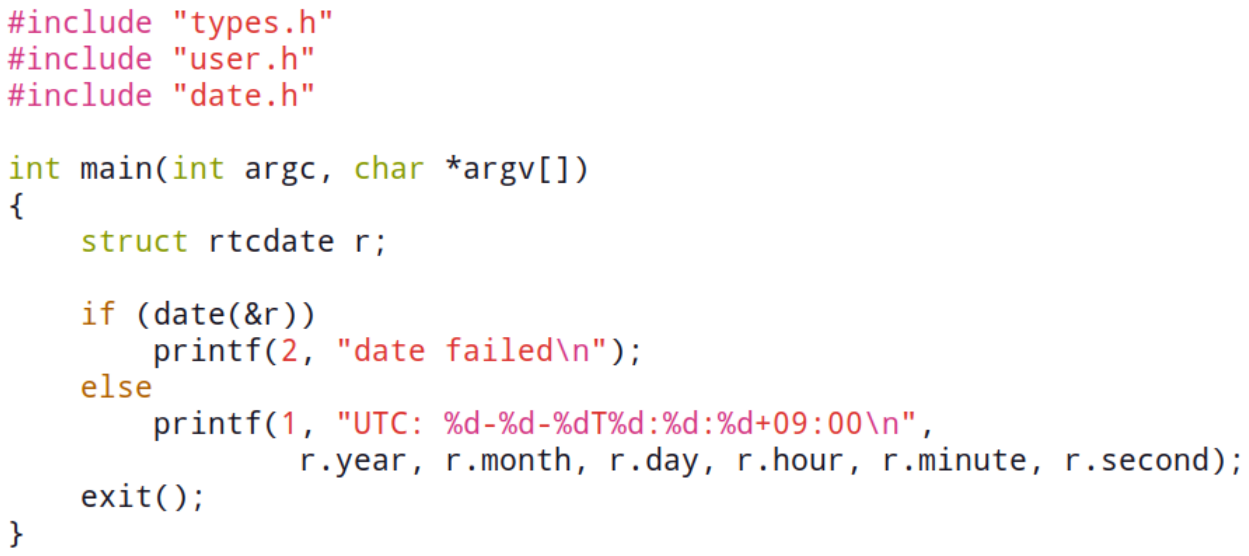


```
#define SYS_mkdir   20
#define SYS_close   21
#define SYS_date    22
```

System Call (3)

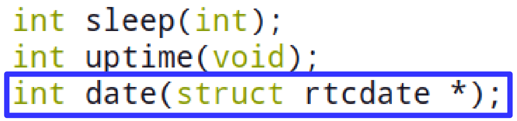
Preparing the date system call testing

- Writing date.c

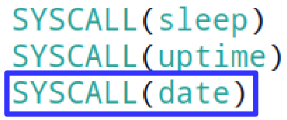


```
#include "types.h"
#include "user.h"
#include "date.h"

int main(int argc, char *argv[])
{
    struct rtcdate r;

    if (date(&r))
        printf(2, "date failed\n");
    else
        printf(1, "UTC: %d-%d-%dT%d:%d:%d+09:00\n",
                r.year, r.month, r.day, r.hour, r.minute, r.second);
    exit();
}
```

- Adding the date system call interface to user.h and usys.S



```
int sleep(int);
int uptime(void);
int date(struct rtcdate *);
```



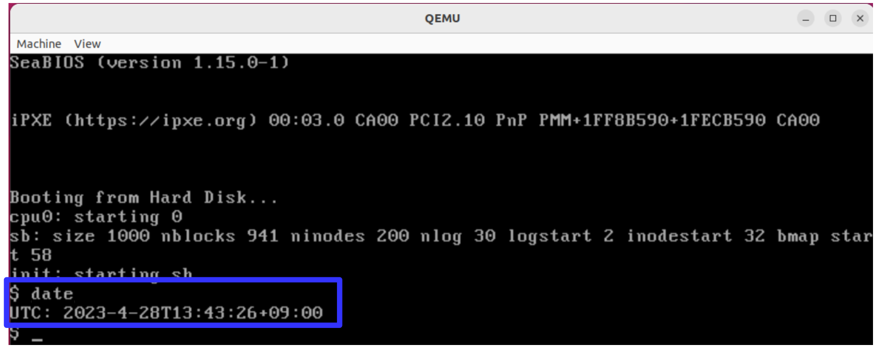
```
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(date)
```

System Call (4)

Compiling xv6 with new system call

- Modifying Makefile

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



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00



Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ date
UTC: 2023-4-28T13:43:26+09:00
$ _
```

Build xv6 and test

- `$ make qemu`
- Type `$date` on qemu terminal (or Linux terminal) and see the time

Task #1: Make your own system call on xv6

System Call Tracing

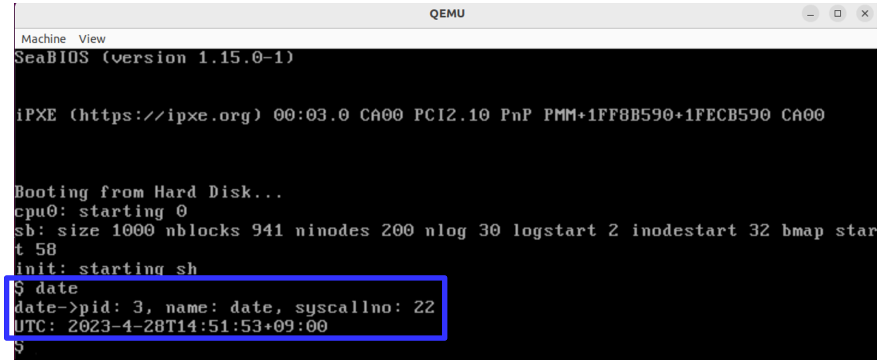
- **Modifying the xv6 kernel to trace each system call invocation**
  - Printing out the system call name & no, process id & name of each invocation
  - To do this, syscall.c should be modified

Initial code

```
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;
  }
}
```

```
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]();
    switch(num)
    {
        // your code here

        case 22: cprintf("date->pid: %d, name: %s, syscallno: %d\n",
                         curproc->pid, curproc->name, num);
                break;
    }
  } else {
    cprintf("%d %s: unknown sys call %d\n",
            curproc->pid, curproc->name, num);
    curproc->tf->eax = -1;
  }
}
```

  - `$ make qemu`

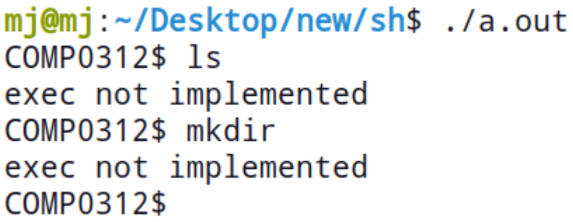


```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00



Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ date
date->pid: 3, name: date, syscallno: 22
UTC: 2023-4-28T14:51:53+09:00
$
```

- **Task #2: Make it trace all of the xv6's system calls**

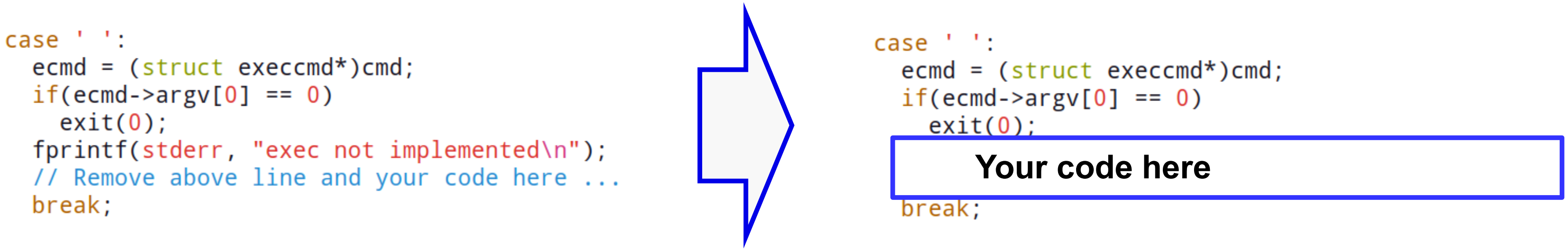
User Defined Shell on Linux (1)

- **Download the example shell code**
  - Note that this shell runs on Linux, not on xv6
  - `$ gcc sh.c` // you will see a warning, which can be ignored
  - `$ ./a.out`



```
mj@mj:~/Desktop/new/sh$ ./a.out
COMP0312$ ls
exec not implemented
COMP0312$ mkdir
exec not implemented
COMP0312$
```

- **Take #3-1: Make the shell to execute commands by filling codes**
  - Hint: use **`execvp`** system call to implement the command execution

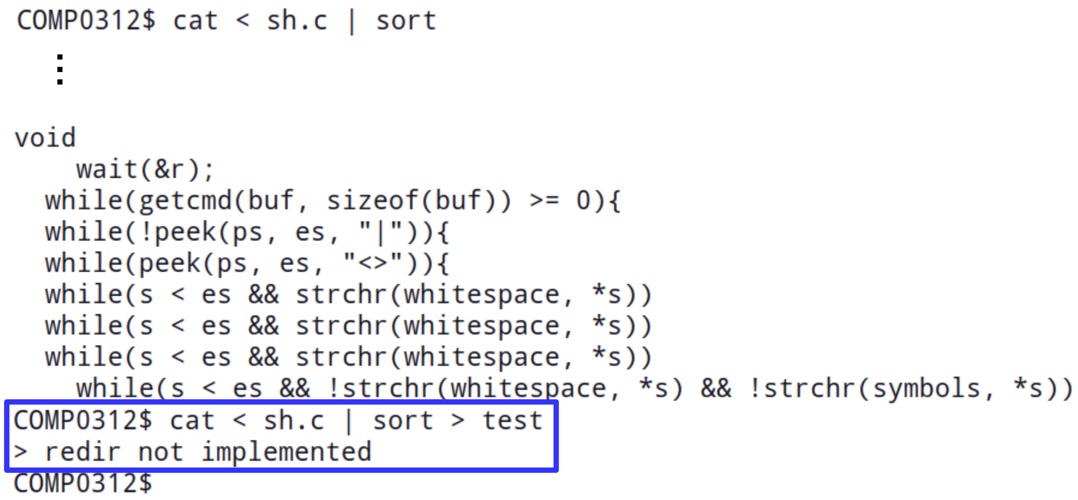


```
case ' ':
  ecmd = (struct execcmd*)cmd;
  if(ecmd->argv[0] == 0)
    exit(0);
  fprintf(stderr, "exec not implemented\n");
  // Remove above line and your code here ...
  break;
```

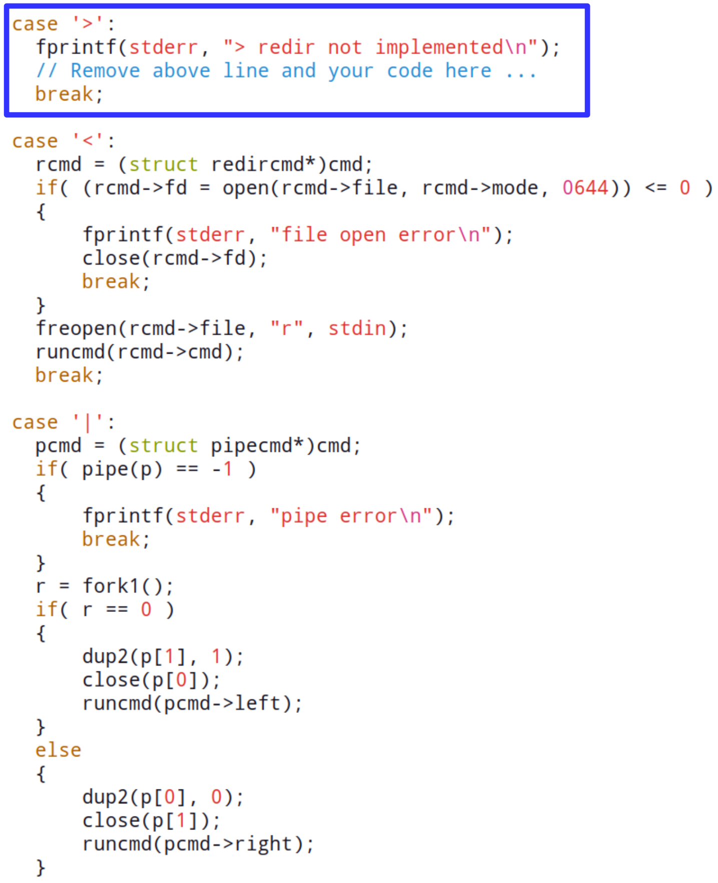
```
case ' ':
  ecmd = (struct execcmd*)cmd;
  if(ecmd->argv[0] == 0)
    exit(0);
      Your code here
  break;
```

User Defined Shell on Linux (2)

- **I/O direction and pipe**
  - < redirection and pipe (|) were already implemented but > redirection wasn’t
  - `$ ./a.out` // to do this, Task #3-1 must be implemented



```
COMP0312$ cat < sh.c | sort
  ⋮

void
  wait(&r);
 while(getcmd(buf, sizeof(buf)) >= 0){
 while(!peek(ps, es, "|")){
 while(peek(ps, es, "<>")){
 while(s < es && strchr(whitespace, *s))
 while(s < es && strchr(whitespace, *s))
 while(s < es && strchr(whitespace, *s))
  while(s < es && !strchr(whitespace, *s) && !strchr(symbols, *s))
COMP0312$ cat < sh.c | sort > test
> redir not implemented
COMP0312$
```



```
case '>':
  fprintf(stderr, "> redir not implemented\n");
  // Remove above line and your code here ...
  break;

case '<':
  rcmd = (struct redircmd*)cmd;
  if( (rcmd->fd = open(rcmd->file, rcmd->mode, 0644)) <= 0 )
  {
      fprintf(stderr, "file open error\n");
      close(rcmd->fd);
      break;
  }
  freopen(rcmd->file, "r", stdin);
  runcmd(rcmd->cmd);
  break;

case '|':
  pcmd = (struct pipecmd*)cmd;
  if( pipe(p) == -1 )
  {
      fprintf(stderr, "pipe error\n");
      break;
  }
  r = fork1();
  if( r == 0 )
  {
      dup2(p[1], 1);
      close(p[0]);
      runcmd(pcmd->left);
  }
  else
  {
      dup2(p[0], 0);
      close(p[1]);
      runcmd(pcmd->right);
  }
```

- **Task #3-2: Make > redirection possible**
  - Hint: < redirection code

OS Project 2

- ## What to Do
  - **Task #1**: Write your own system call on xv6
    - Your system call (proj2call) simply prints the following message to kernel: **COMP0312_OS_PROJ2_yourStudentID_yourName: Hello xv6**
  - **Task #2**: Write codes to trace all xv6's system call after booting
    - Tracing xv6's system call no. 1 (fork) ~ no. 23 (proj2call)
  - **Task #3**: Implement the user defined shell
    - Making the shell to execute commands and support > redirection
- ## Submission Due
  - Due: 5/21, Sunday 23:59
  - No late submission is allowed
- ## What to Submit (single tarball .tgz or .tar)
  - System call files: syscall.c, syscall.h, sysproc.c, user.h, usys.S, Makefile
  - User defined shell file: sh.c
- ## Grading
  - Total: 100 pts