

# CS 202: Advanced Operating Systems

You can use sledge server for the course labs.

```
$ ssh -X username@sledge.cs.ucr.edu
```

**To download xv6:**

```
$ git clone https://github.com/mit-pdos/xv6-public.git
```

```
$ cd xv6-public
```

Note: whenever you want to discard all of your changes do this in xv6-public:

```
$ git checkout .
```

**Links:**

[xv6 book](#)

[xv6 indexed/cross referenced code](#)

**To run XV6:**

```
$ make qemu
```

If the mouse pointer gets stuck in the QEMU emulator window press: Ctrl + Alt + G

Hint: Always do “make clean” first

```
$ make clean && make qemu
```

Qemu by default emulates 2 CPUs, it is much easier for future labs to have it emulate a single core only:

```
$ vim Makefile
```

Change:

```
ifndef CPUS
```

```
CPUS := 2
```

```
endif
```

to:

```
CPUS := 1
```

## To create a system call:

Make a system call “sys\_hello” that call a kernel function that displays: “Hello from the kernel space!”

To do that open the following files and add the line(s) with **//BR** comment:

In “**usys.S**”

```
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(hello) //BR
33 |
```

In “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
19 #define SYS_unlink 18
20 #define SYS_link 19
21 #define SYS_mkdir 20
22 #define SYS_close 21
23 #define SYS_hello 22 //BR
24
```

In “syscall.c”

```
79  
80 extern int sys_chdir(void);  
81 extern int sys_close(void);  
82 extern int sys_dup(void);  
83 extern int sys_exec(void);  
84 extern int sys_exit(void);  
85 extern int sys_fork(void);  
86 extern int sys_fstat(void);  
87 extern int sys_getpid(void);  
88 extern int sys_kill(void);  
89 extern int sys_link(void);  
90 extern int sys_mkdir(void);  
91 extern int sys_mknod(void);  
92 extern int sys_open(void);  
93 extern int sys_pipe(void);  
94 extern int sys_read(void);  
95 extern int sys_sbrk(void);  
96 extern int sys_sleep(void);  
97 extern int sys_unlink(void);  
98 extern int sys_wait(void);  
99 extern int sys_write(void);  
100 extern int sys_uptime(void);  
101 extern int sys_hello(void); //BR  
102
```

And

```
102
103 static int (*syscalls[])(void) = {
104     [SYS_fork]    sys_fork,
105     [SYS_exit]    sys_exit,
106     [SYS_wait]    sys_wait,
107     [SYS_pipe]    sys_pipe,
108     [SYS_read]    sys_read,
109     [SYS_kill]    sys_kill,
110     [SYS_exec]    sys_exec,
111     [SYS_fstat]   sys_fstat,
112     [SYS_chdir]   sys_chdir,
113     [SYS_dup]     sys_dup,
114     [SYS_getpid]  sys_getpid,
115     [SYS_sbrk]    sys_sbrk,
116     [SYS_sleep]   sys_sleep,
117     [SYS_uptime]  sys_uptime,
118     [SYS_open]    sys_open,
119     [SYS_write]   sys_write,
120     [SYS_mknod]   sys_mknod,
121     [SYS_unlink]  sys_unlink,
122     [SYS_link]    sys_link,
123     [SYS_mkdir]   sys_mkdir,
124     [SYS_close]   sys_close,
125     [SYS_hello]   sys_hello, //BR
126 };
127
```

In "sysproc.c"

```
92
93 // BR
94 int
95 sys_hello(void)
96 {
97     hello();
98     return 0;
99 }
100 // BR
101
```

In "proc.c"

```
483         cprintf(" %p", pc[i]);
484     }
485     cprintf("\n");
486 }
487 }
488 //BR
489 void
490 hello(void)
491 {
492     cprintf("\n\n Hello from the kernel space! \n\n");
493 }
494 //BR
```



In "defs.h"

```
104 //PAGEBREAK: 16
105 // proc.c
106 void      exit(void);
107 int       fork(void);
108 int       growproc(int);
109 int       kill(int);
110 void      pinit(void);
111 void      procdump(void);
112 void      scheduler(void) __attribute__((noreturn));
113 void      sched(void);
114 void      sleep(void*, struct spinlock*);
115 void      userinit(void);
116 int       wait(void);
117 void      wakeup(void*);
118 void      yield(void);
119 void      hello(void); //BR
120
121 // swtch.S
122 void      swtch(struct context**, struct context*);
123
```

In "user.h"

```

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);
25 int uptime(void);
26 int hello(void); //BR
27
```

Create “test.c” file in the home directory of “xv6-public”

```
#include "types.h"

#include "user.h"

int main(int argc, char *argv[]) {

hello();

    exit();
}
```

Edit “Makefile” by appending “\_test\” to UPROGS

```
159
160 UPROGS=\
161     _cat\
162     _echo\
163     _forktest\
164     _grep\
165     _init\
166     _kill\
167     _ln\
168     _ls\
169     _mkdir\
170     _rm\
171     _sh\
172     _stressfs\
173     _usertests\
174     _wc\
175     _zombie\
176     _test\
177
```

Now type:

\$ make qemu

After xv6 boots: type”

\$ test

And you should see the message:



```
$ [broma002@sledge xv6-public]$ make qemu
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB) copied, 0.0377488 s, 136 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000161106 s, 3.2 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
334+1 records in
334+1 records out
171121 bytes (171 kB) copied, 0.000858728 s, 199 MB/s
qemu -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6...
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test
```

Shell

(broma002) sledge - KDE Terminal Emulator

(broma002) sledge - KDE Terminal Emulator

```
QEMU
SeaBIOS (version pre-0.6.3-20110315_112143-titi)

iPXE v1.0.0-591-g7aee315
iPXE (http://ipxe.org) 00:03.0 C900 PCI2.10 PnP PMM+1FFC8D60+1FF88D60 C900

Booting from Hard Disk...

cpu0: starting xv6

ioapicinit: id isn't equal to ioapicid; not a MP
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ test

Hello from the kernel space!
$ _
```

To run Qemu with GDB

You need to do this the first time only, in your home dir:

```
$ echo "add-auto-load-safe-path /home/csgrads/NetID/xv6-public/.gdbinit" >> .gdbinit
```

Then you need to open another terminal at the same xv6-public folder:

```
$ xfce4-terminal &
```

In the original terminal type:

```
$ make qemu-gdb
```

In second terminal type:

```
$ gdb
```

```
+ target remote localhost:25049
```

```
The target architecture is assumed to be i8086 [f000:fff0] 0xffff0: ljmp $0xf000,$0xe05b  
0x0000fff0 in ?? ()
```

```
+ symbol-file kernel
```

```
$ (gdb) continue
```

For GDB with GUI, type:

```
$ gdb -tui
```

**Note:** you may find the text editor geany to be useful:

```
$ cd xv6-public/
```

```
$ geany *.c *.h *.S &
```

You can right click a function or variable to find where it has been defined and used in the code.

For a quick vim tutorial type in terminal:

```
$ vimtutor
```

If you want to use vim instead of geany, use ctags to help you index the code:

<https://andrew.stwrt.ca/posts/vim-ctags/>

This grep command is helpful:

```
$ grep -rnw -color . -e "search string"
```

<https://www.cyberciti.biz/faq/howto-use-grep-command-in-linux-unix/>

hint: do “make clean” before running grep and be aware that grep results will conflict with the ctags generated file.

Windows users may find “MobaXterm” to be a good SSH and FTP client (the free version):

<https://mobaxterm.mobatek.net/download.html>

MAC users will need:

<https://www.xquartz.org/>

for GUI