Introduction to XV6

CS202 Lab

Presenter: Sina Davanian

Instructor: Prof. Heng Yin

Presentation outline

- About me (Sina)
- Logistics
- What is XV6
- Hello World in XV6
- System calls in XV6

About me

- I have two names, but almost everybody calls me Sina
- I am one of Prof. Yin's PHD students
- I work on system security
  - A cool way of saying the same thing is that I am trying to hack into OSes ☺
- I have one MS in security and privacy and one MS in Entrepreneurship and Management
- My hubbies are:
  - Playing pool
  - Dancing
  - And hanging out with friends

Logistics

- The demo will be uploaded as a recorded video
  - You just need to follow the same steps later to get the tasks running
- In addition to this presentation, a few other tutorials will be uploaded
- During the class, I suggest you just listen and ask questions (don’t try the commands)
- Start playing with XV6 ASAP!
  - It’s fun, you’ll enjoy it
- Don’t wait until last minute (for the entire quarter) to raise your questions

What is XV6?

- XV6 is a lightweight operating system
- It was designed in MIT for educational purposes
- It has an easy-to-understand structure

The bottom line is:

Understanding high level OS concepts using XV6 is straightforward

How can one run XV6?

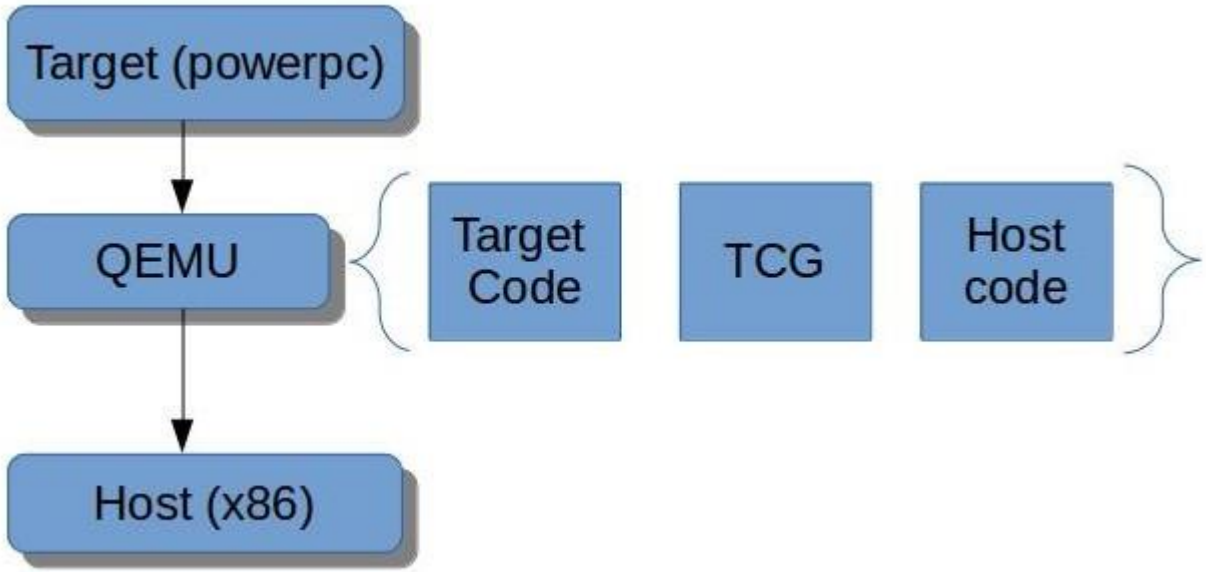
- XV6 can be installed as a standalone OS (not recommended)
- XV6 can run in an emulated environment (recommended)
- ***QEMU*** is the emulator we use to run XV6

How can one run XV6?

- QEMU is a user program/system emulator

Examples:

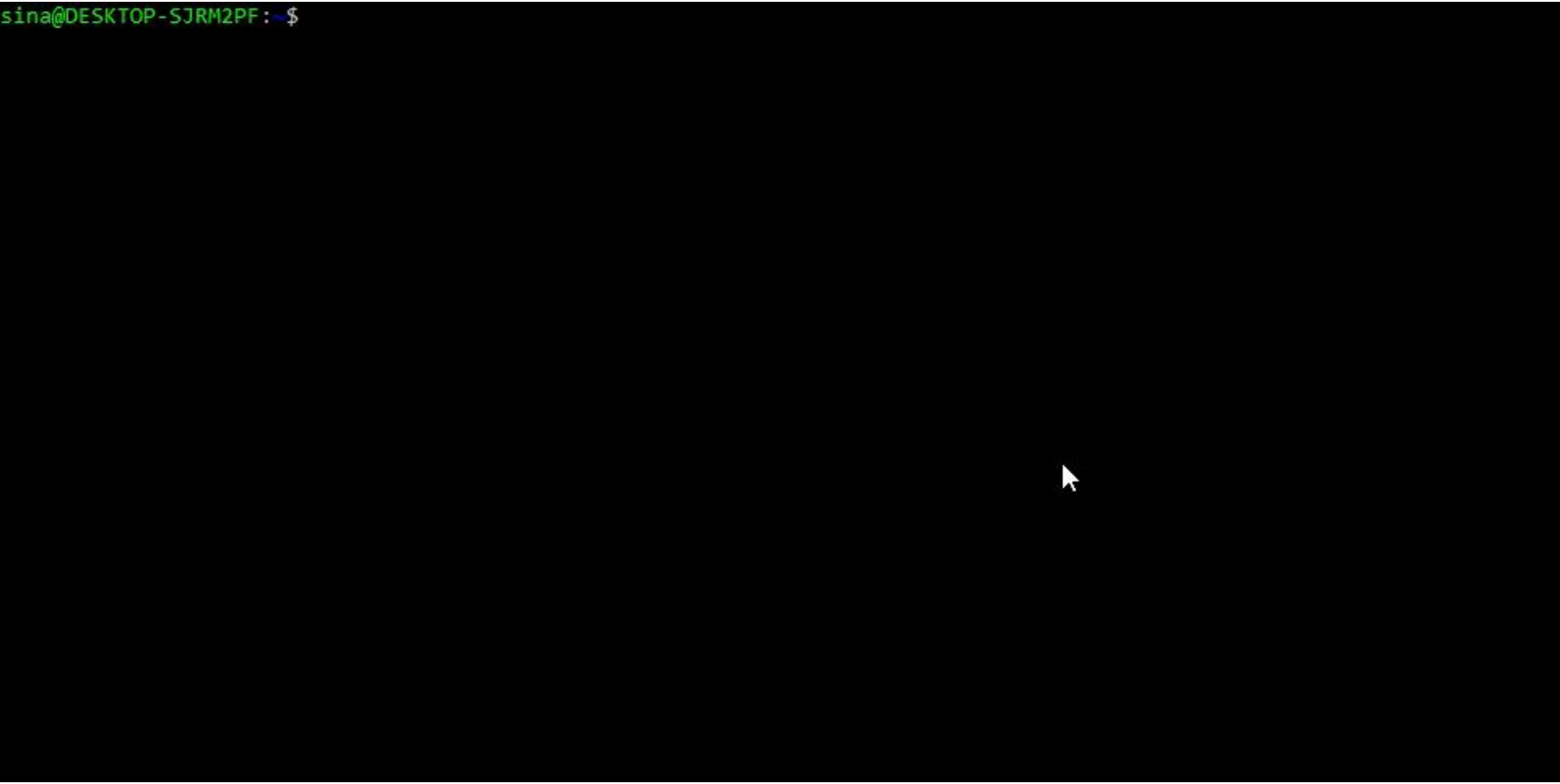
- Running Arm on your Home PC
- Running Windows on Linux
- Running XV6 on your Ubunto



Write a hello world program in XV6

High level steps

- Download XV6 source code
- Write a hello world program
- Include the program in the XV6 source code
  - A way of uploading the program to XV6
- Compile and run XV6 with QEMU
- Run the hello world program inside XV6


sina@DESKTOP-SJRM2PF:~$

Make a “hello world” system call

mkdir system call in XV6

Let’s see how system calls are already defined in xv6

Definition of a system call in the kernel land

- SYSCALL([NAME]) in usys.S:
  - SYSCALL(mkdir)
- This line translates to the following assembly:

```
.globl mkdir;
mkdir:
movl $SYS_mkdir, %eax;   //putting the system call number in eax
int $T_SYSCALL;          //calling the system call handler in interrupt mode
```


We have to define this constant

Definition of a system call in the kernel land

- $T_SYSCALL is a pointer to “syscall” function in syscall.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;
  }
}
```

syscalls[SYS_mkdir]

Definition of a system call in the kernel land

- To make the call to syscalls[SYS_mkdir] working, we need two things:
  - define SYS_mkdir

```
#define SYS_mkdir  20
```

in syscall.h

  - put the pointer to our system call in syscalls[SYS_mkdir] array element

```
106
107 static int (*syscalls[])(void) = {
108   [SYS_fork]    sys_fork,
109   [SYS_exit]    sys_exit,
110   [SYS_wait]    sys_wait,
111   [SYS_pipe]    sys_pipe,
112   [SYS_read]    sys_read,
113   [SYS_kill]    sys_kill,
114   [SYS_exec]    sys_exec,
115   [SYS_fstat]   sys_fstat,
116   [SYS_chdir]   sys_chdir,
117   [SYS_dup]     sys_dup,
118   [SYS_getpid]  sys_getpid,
119   [SYS_sbrk]    sys_sbrk,
120   [SYS_sleep]   sys_sleep,
121   [SYS_uptime]  sys_uptime,
122   [SYS_open]    sys_open,
123   [SYS_write]   sys_write,
124   [SYS_mknod]   sys_mknod,
125   [SYS_unlink]  sys_unlink,
126   [SYS_link]    sys_link,
127   [SYS_mkdir]   sys_mkdir,
128   [SYS_close]   sys_close,
129 };
130
```

in syscall.c

Definition of a system call in the kernel land

- So now syscalls[SYS_mkdir] points to *sys_mkdir*
- We need to define *sys_mkdir* function in the kernel land
- We put the function definition in “sysproc.c” file
- Since we define *sys_mkdir outside syscall.c, we need this line:*

```
95    extern int sys_mkdir(void);
```

in syscall.c

Definition of a system call in the kernel land

- *sys_mkdir skeleton in the kernel land (sysproc.c):*

```
int
sys_mkdir(void)
{
  char *path;
  struct inode *ip;

  begin_op();
  if(argstr(0, &path) < 0 || (ip = create(path, T_DIR, 0, 0)) == 0){
    end_op();
    return -1;
  }
  iunlockput(ip);
  end_op();
  return 0;
}
```

Define the system call in the kernel land

- What if we want to put the function definition in “proc.c” file because we can access many variables and kernel functions in proc.c? [you need to put your code there for your assignment ;-)]
  - We define a function in proc.c and call it in the sysproc.c definition

Defining “hello” system call in the kernel land

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
484
485
486
487
488 //BR
489 void
490 hello(void)
491 {
492     cprintf("\n\n Hello from the kernel space! \n\n");
493 }
494 //BR
```

Since hello is in a different file we need to mention this in “defs.h” header file

In “defs.h”

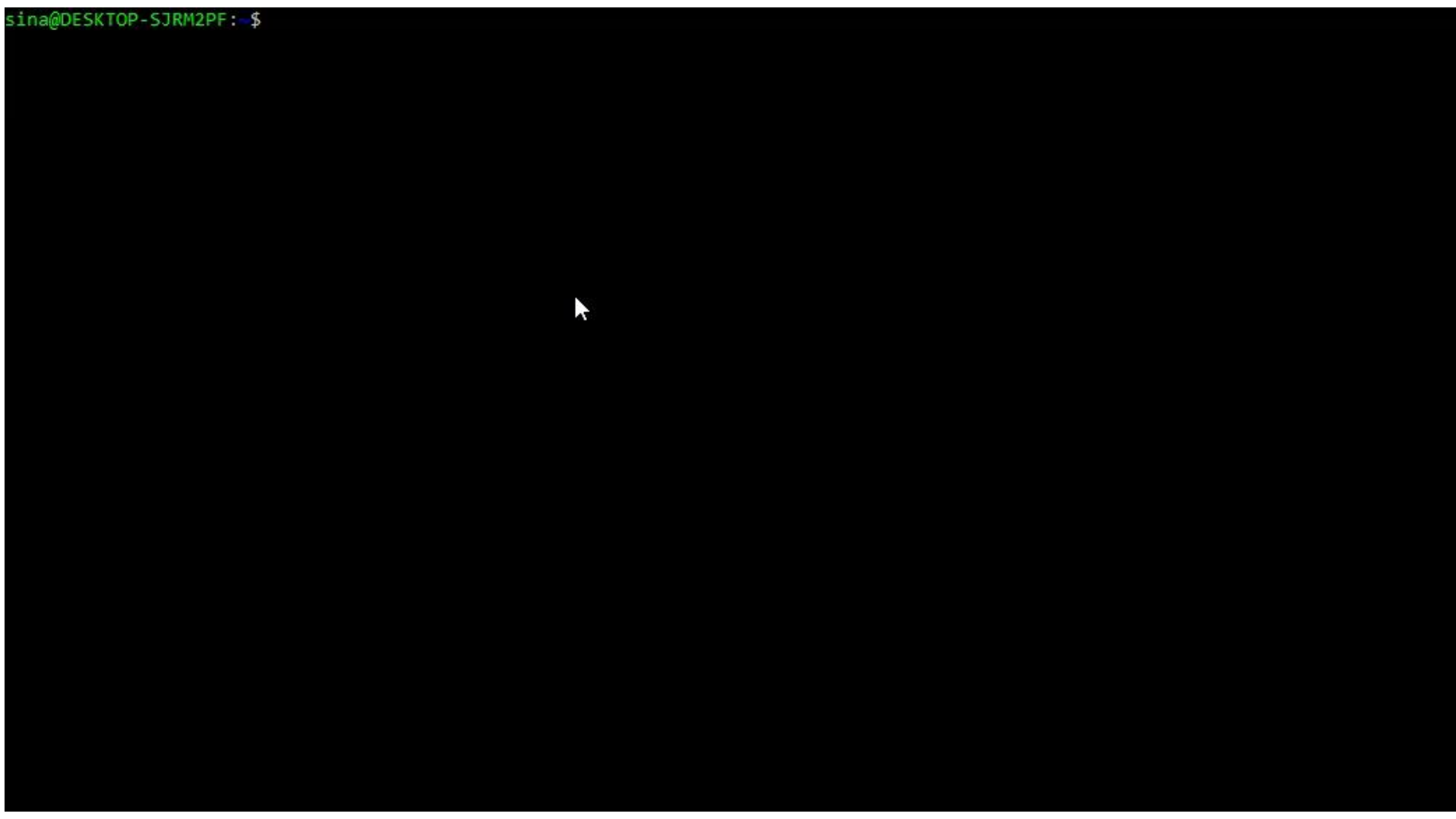
```
104 //PAGEBREAK: 16
105 // proc.c
106 void            exit(void);
107 int             fork(void);
108 int             growproc(int);
109 int             kill(int);
110 void            pinit(void);
111 void            procdump(void);
112 void            scheduler(void) __attribute__((noreturn));
113 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
```

Defining “hello” system call in the kernel land

- Finally…
- We need to mention that “hello” is a library function

In “user.h”

```
4   // system calls
5   int fork(void);
6   int exit(void) __attribute__
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, shor
16  int unlink(char*);
17  int fstat(int fd, struct sta
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
```


sina@DESKTOP-SJRM2PF:~$