A decorative network diagram in the top-left corner, consisting of a complex web of interconnected nodes and lines, rendered in a light gray color. The nodes are represented by small circles, some of which are larger and have concentric circles inside, suggesting a hierarchical or multi-layered structure. The lines connecting them are thin and gray, creating a mesh-like pattern.

OS Task 3: Adding System Calls In XV6

Farah Kabesh - 900191706


Karim Sherif - 900191474

A decorative network diagram in the bottom-right corner, similar to the one in the top-left, featuring a complex web of interconnected nodes and lines in a light gray color. The nodes are small circles, some with concentric circles, connected by thin gray lines.

A decorative network diagram in the top-left corner, consisting of a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, while others are smaller. The lines are thin and gray, connecting the nodes in a non-linear fashion. The overall style is minimalist and technical.

Description of 3 System Calls & Prototypes

sys_cps
sys_addrsize
sys_datetime

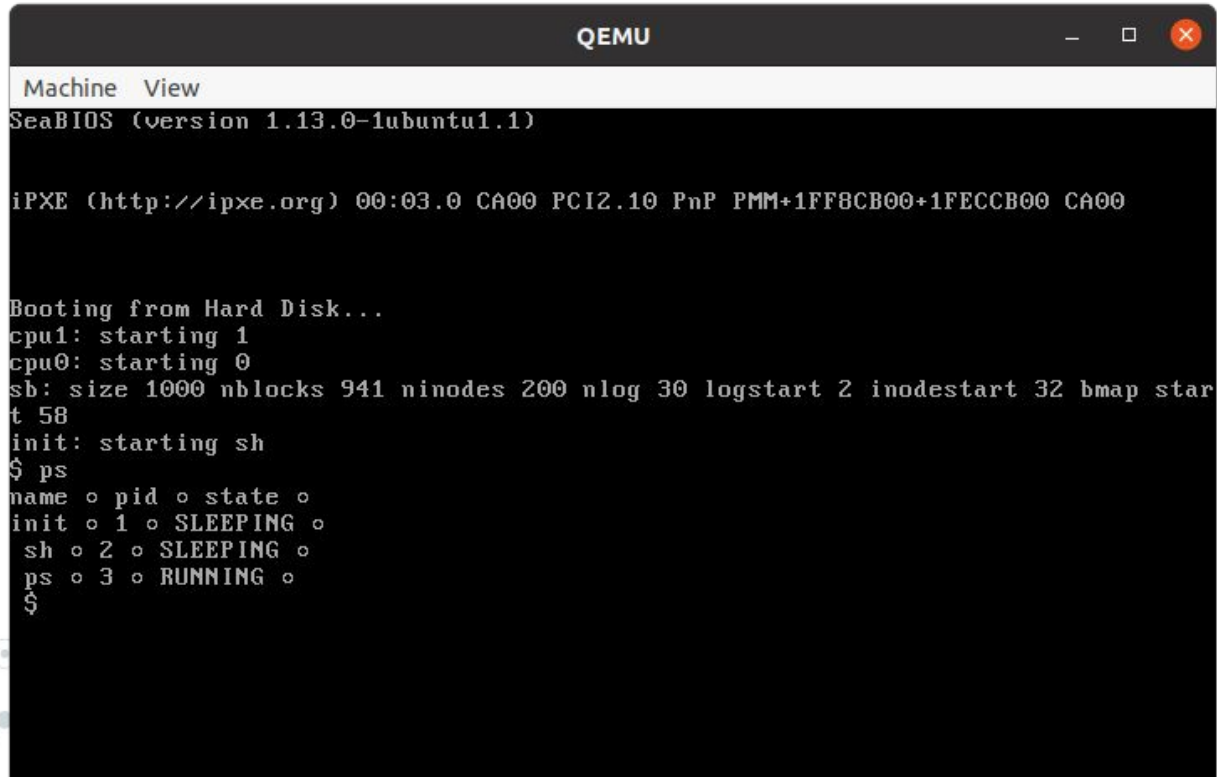
A decorative network diagram in the bottom-right corner, similar to the one in the top-left. It features a cluster of nodes connected by lines. The nodes vary in size and some have concentric circles, all rendered in a light gray color. The lines are thin and gray, creating a web-like structure.

sys_cps

- © Prototype: `int sys_cps(void)`
- © Purpose: gets process name, process ID and process state
- © Implementation:

```
536 int
537 cps(void)
538 {
539     struct proc *p;
540
541     //Enable interrupts on this processor
542     sti();
543
544     //Loop over process table looking for process with pid.
545     acquire(&ptable.lock);
546     cprintf("name \t pid \t state \t \n");
547     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
548         if(p->state == SLEEPING)
549             cprintf("%s \t %d \t SLEEPING \t \n ", p->name, p->pid);
550         else if(p->state == RUNNING)
551             cprintf("%s \t %d \t RUNNING \t \n ", p->name, p->pid);
552         else if(p->state == RUNNABLE)
553             cprintf("%s \t %d \t RUNNABLE \t \n ", p->name, p->pid);
554     }
555     release(&ptable.lock);
556     return 22;
557 }
```

Testing The System Call

A screenshot of a QEMU terminal window. The window has a title bar with the text 'QEMU' and standard window controls. Below the title bar is a menu bar with 'Machine' and 'View'. The terminal content shows the SeaBIOS boot process, including the iPXE boot loader, booting from a hard disk, and the initialization of the system. The 'ps' command is run, showing the status of the 'init', 'sh', and 'ps' processes.

```
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

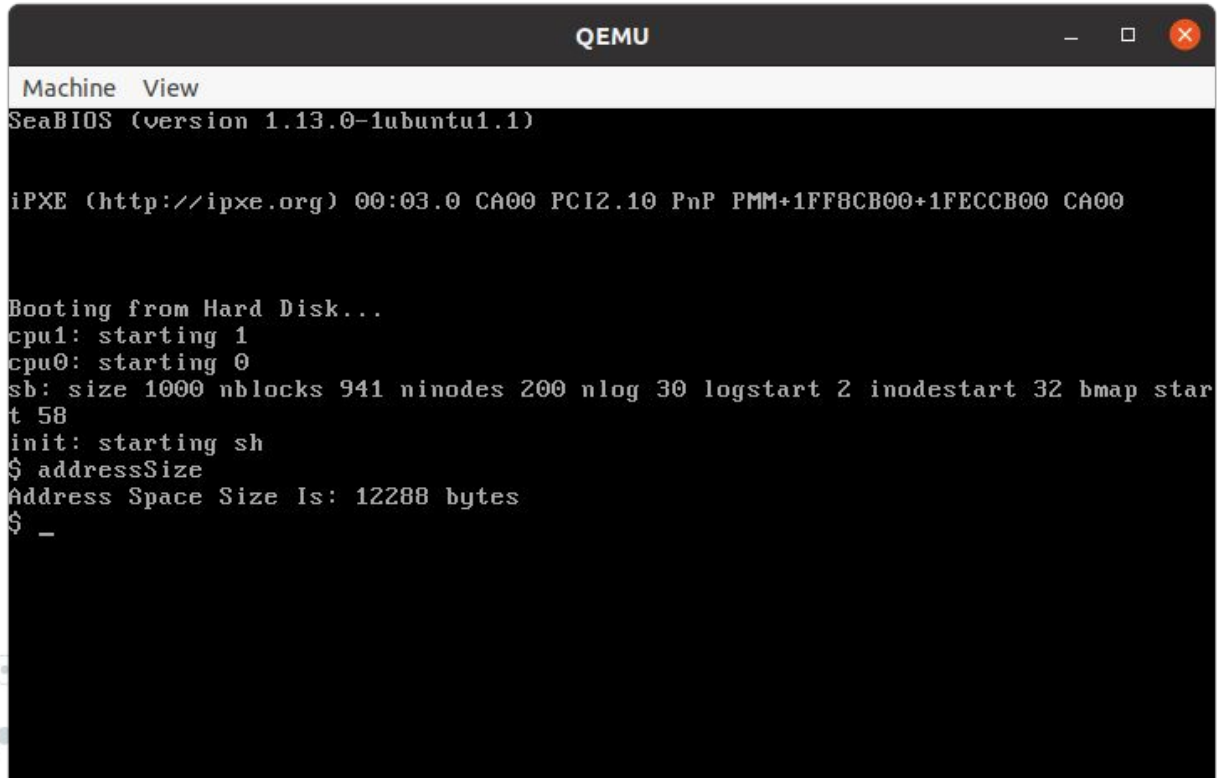
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ ps
name  o pid  o state  o
init  o  1  o SLEEPING o
sh    o  2  o SLEEPING o
ps    o  3  o RUNNING  o
$
```

sys_addrsize

- © Prototype: `int sys_addrsize(void)`
- © Purpose: gets address space size of process memory (currently running user program) in bytes
- © Implementation:

```
93 int
94 sys_addrsize(void)
95 {
96     return myproc()->sz;
97 }
98
```

Testing The System Call

A screenshot of a QEMU terminal window. The window has a title bar with the text 'QEMU' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'Machine' and 'View'. The main area is a black terminal with white text. The text shows the SeaBIOS boot process, including the iPXE boot loader, booting from a hard disk, and the initialization of the kernel and shell. The prompt '\$' is visible at the bottom.

```
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

Booting from Hard Disk...
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
$ addressSize
Address Space Size Is: 12288 bytes
$ _
```

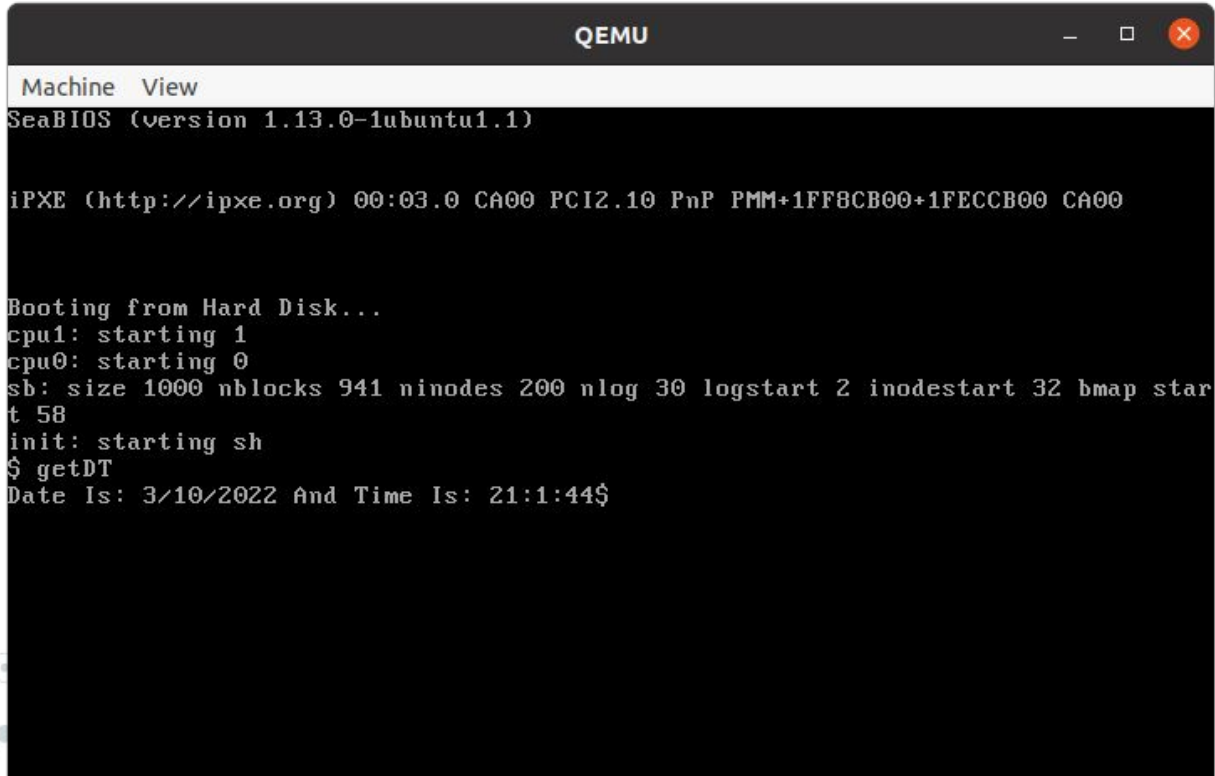
sys_datetime

- © Prototype: `int sys_datetime(void)`
- © Purpose: gets current date and time (UTC timezone)
- © Implementation:

```
98
99 int
100 sys_datetime(void)
101 {
102     struct rtcdate *d;
103     if(argptr(0, (void*) &d, sizeof(struct rtcdate)) < 0)
104         return -1;
105     cmostime(d);
106
107     return 0;
108 }
109
110 int sys_cps(void)
111 {
112     return cps();
113 }
```

- *cmostime()* helper function which is defined in *lapic.c* file to read real time clock
- *date.h* file contains definition of struct *rtcdate*

Testing The System Call



The screenshot shows a QEMU window titled "QEMU" with a menu bar containing "Machine" and "View". The main display area is black with white text showing the boot process of a virtual machine. The text includes the SeaBIOS version, iPXE boot loader information, and the start of the Linux kernel boot process, including CPU initialization and the start of the init system.

```
Machine View
SeaBIOS (version 1.13.0-1ubuntu1.1)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CB00+1FECCB00 CA00

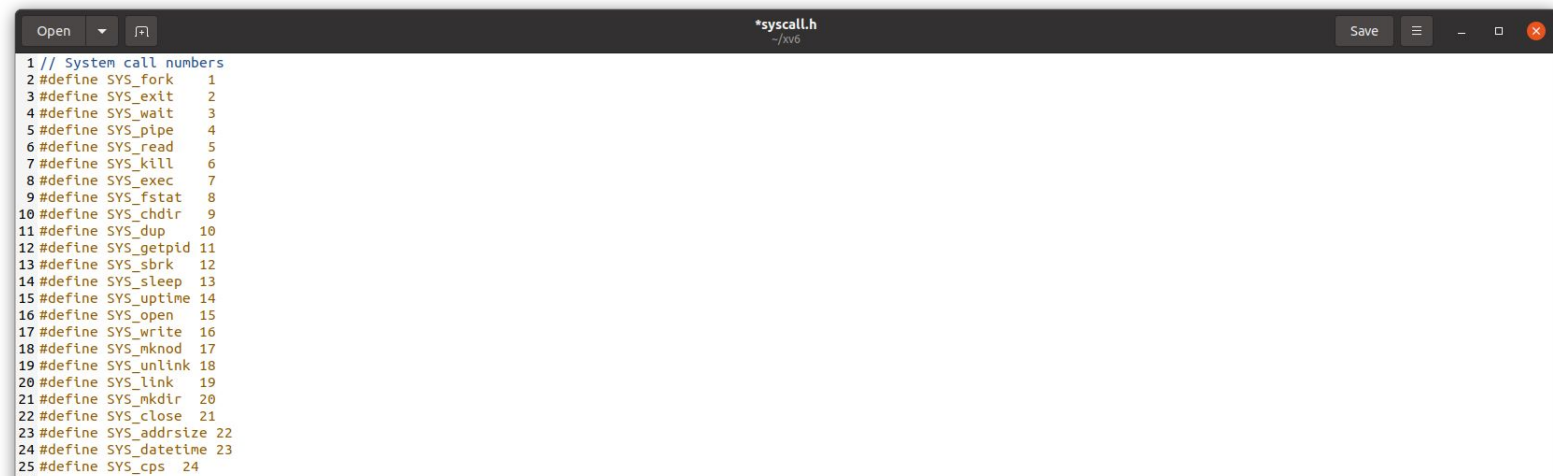
Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ getDT
Date Is: 3/10/2022 And Time Is: 21:1:44$
```


Reasons Why 3 Functions Should Be System Calls

- © The Kernel mode allows programs to access memory and hardware resources directly.
- © Therefore, when a program needs a memory or resource, it makes a system call and the mode changes from user mode to kernel mode.
- © System calls are handled via a software interrupt.
- © The interrupt handler runs the implementation of the system call in kernel mode, then switches back to user mode.

How System Calls Were Added In XV6

© Edit *syscall.h* file



```
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_addrsize 22
24 #define SYS_datetime 23
25 #define SYS_cps 24
```

How System Calls Were Added In XV6

- © Edit *syscall.c* file to include function prototypes and add them to array of syscalls

```
86 extern int sys_close(void);
87 extern int sys_dup(void);
88 extern int sys_exec(void);
89 extern int sys_exit(void);
90 extern int sys_fork(void);
91 extern int sys_fstat(void);
92 extern int sys_getpid(void);
93 extern int sys_kill(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 extern int sys_addrsize(void);
107 extern int sys_datetime(void);
108 extern int sys_cps(void);
109
```

```
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,
128 [SYS_mknod] sys_mknod,
129 [SYS_unlink] sys_unlink,
130 [SYS_link] sys_link,
131 [SYS_mkdir] sys_mkdir,
132 [SYS_close] sys_close,
133 [SYS_addrsize] sys_addrsize,
134 [SYS_datetime] sys_datetime,
135 [SYS_cps] sys_cps,
136 };
137
```

How System Calls Were Added In XV6

© Edit *sysproc.c* file to implement system call functions

```
92 `
93 int
94 sys_addrsize(void)
95 {
96     return myproc()->sz;
97 }
98
99 int
100 sys_datetime(void)
101 {
102     struct rtcdate *d;
103     if(argptr(0, (void*) &d, sizeof(struct rtcdate)) < 0)
104         return -1;
105     cmostime(d);
106
107     return 0;
108 }
109
110 int sys_cps(void)
111 {
112     return cps();
113 }
```

C Tab Width: 8 Ln 16, Col 4 INS

How System Calls Were Added In XV6

sys_cps

© Edit *defs.h* and *proc.c* files

```
105 // proc.c
106 int
107 void
108 int
109 int
110 int
111 struct cpu*
112 struct proc*
113 void
114 void
115 void
116 void
117 void
118 void
119 void
120 int
121 void
122 void
123 int
124

cpuid(void);
exit(void);
fork(void);
growproc(int);
kill(int);
mycpu(void);
myproc();
pinit(void);
procdump(void);
scheduler(void) __attribute__((noreturn));
sched(void);
setproc(struct proc*);
sleep(void*, struct spinlock*);
userinit(void);
wait(void);
wakeup(void*);
yield(void);
cps(void);
```

```
535
536 int
537 cps(void)
538 {
539     struct proc *p;
540
541     //Enable interrupts on this processor
542     sti();
543
544     //Loop over process table looking for process with pid.
545     acquire(&ptable.lock);
546     printf("name \t pid \t state \t \n");
547     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
548         if(p->state == SLEEPING)
549             printf("%s \t %d \t SLEEPING \t \n ", p->name, p->pid);
550         else if(p->state == RUNNING)
551             printf("%s \t %d \t RUNNING \t \n ", p->name, p->pid);
552         else if(p->state == RUNNABLE)
553             printf("%s \t %d \t RUNNABLE \t \n ", p->name, p->pid);
554     }
555     release(&ptable.lock);
556     return 22;
557 }
```

How System Calls Were Added In XV6

© Edit *usys.s* and *user.h* files

```
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
11SYSCALL(fork)
12SYSCALL(exit)
13SYSCALL(wait)
14SYSCALL(pipe)
15SYSCALL(read)
16SYSCALL(write)
17SYSCALL(close)
18SYSCALL(kill)
19SYSCALL(exec)
20SYSCALL(open)
21SYSCALL(mknod)
22SYSCALL(unlink)
23SYSCALL(fstat)
24SYSCALL(link)
25SYSCALL(mkdir)
26SYSCALL(chdir)
27SYSCALL(dup)
28SYSCALL(getpid)
29SYSCALL(sbrk)
30SYSCALL(sleep)
31SYSCALL(uptime)
32SYSCALL(addrsize)
33SYSCALL(datetime)
34SYSCALL(cps)
35
```

```
1struct stat;
2struct rtcdate;
3
4// system calls
5int fork(void);
6int exit(void) __attribute__((noreturn));
7int wait(void);
8int pipe(int*);
9int write(int, const void*, int);
10int read(int, void*, int);
11int close(int);
12int kill(int);
13int exec(char*, char**);
14int open(const char*, int);
15int mknod(const char*, short, short);
16int unlink(const char*);
17int fstat(int fd, struct stat*);
18int link(const char*, const char*);
19int mkdir(const char*);
20int chdir(const char*);
21int dup(int);
22int getpid(void);
23char* sbrk(int);
24int sleep(int);
25int uptime(void);
26int addrsz(void);
27int datetim(struct rtcdate*);
28int cps(void);
29
```

Description of 3 User Programs

- ◎ User Program Name: **ps.c**
- ◎ Calls system call *cps()* to test its functionality

```
1 #include "types.h"
2 #include "user.h"
3 #include "stat.h"
4
5
6
7 int main()
8 {
9     cps();
10    exit();
11 }
12
```

Description of 3 User Programs

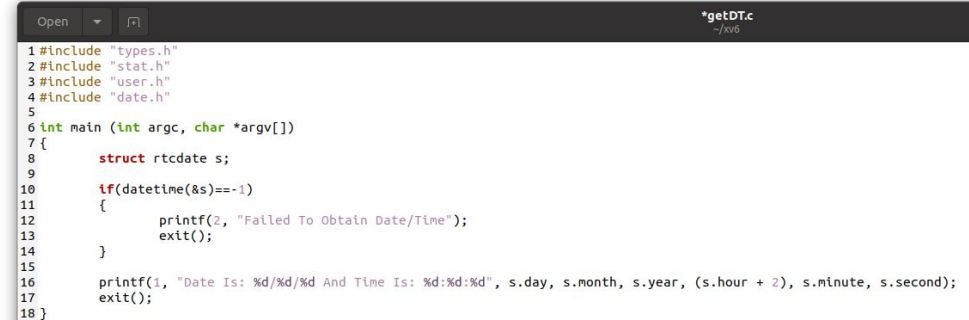
```
Open [v] [f]
```

```
1 #include "types.h"
2 #include "user.h"
3 #include "stat.h"
4
5 int main(void)
6 {
7     printf(1, "Address Space Size Is: %d bytes \n", addrspace());
8
9     exit();
10 }
11
```

- © User Program Name: **addressSize.c**
- © Calls system call *addrspace()* to test its functionality and prints address space size in bytes

Description of 3 User Programs

- © User Program Name: **getDT.c**
- © System call *datetime(&s)* used inside an if-statement to test its functionality. If equal to -1, then it'll print "Failed To Obtain Date/Time". Otherwise it'll print Date and Time in the specified format.
- © (s.hour + 2) in order to display time in Egypt, since it is automatically set to the UTC timezone .



```
1 #include "types.h"
2 #include "stat.h"
3 #include "user.h"
4 #include "date.h"
5
6 int main (int argc, char *argv[])
7 {
8     struct rtcdate s;
9
10    if(datetime(&s) == -1)
11    {
12        printf(2, "Failed To Obtain Date/Time");
13        exit();
14    }
15
16    printf(1, "Date Is: %d/%d/%d And Time Is: %d:%d:%d", s.day, s.month, s.year, (s.hour + 2), s.minute, s.second);
17    exit();
18 }
```