

Project-1 Report

Successfully ran make clean

```
asha3011@DESKTOP-VJINRSQ x + v
asha3011@DESKTOP-VJINRSQ:~$ cd xv6asha/
asha3011@DESKTOP-VJINRSQ:~/xv6asha$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _hello _hello_kernel _sleep
asha3011@DESKTOP-VJINRSQ:~/xv6asha$ |
```

Successfully ran make

```
asha3011@DESKTOP-VJINRSQ x + v
ck-protector -fno-pie -no-pie -c -o vm.o vm.c
gcc -m32 -gdwarf-2 -Wa,-divide -c -o entry.o entry.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32
ck-protector -fno-pie -no-pie -fno-pic -nostdinc -I. -c entryother.S
ld -m elf_i386 -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o
objcopy -S -O binary -j .text bootblockother.o entryother
objdump -S bootblockother.o > entryother.asm
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32
ck-protector -fno-pie -no-pie -nostdinc -I. -c initcode.S
ld -m elf_i386 -N -e start -Ttext 0 -o initcode.out initcode.o
objcopy -S -O binary initcode.out initcode
objdump -S initcode.o > initcode.asm
ld -m elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o
.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o swtch
.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/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0127788 s, 401 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 7.644e-05 s, 6.7 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
393+1 records in
393+1 records out
201596 bytes (202 kB, 197 KiB) copied, 0.00123363 s, 163 MB/s
asha3011@DESKTOP-VJINRSQ:~/xv6asha$ |
```

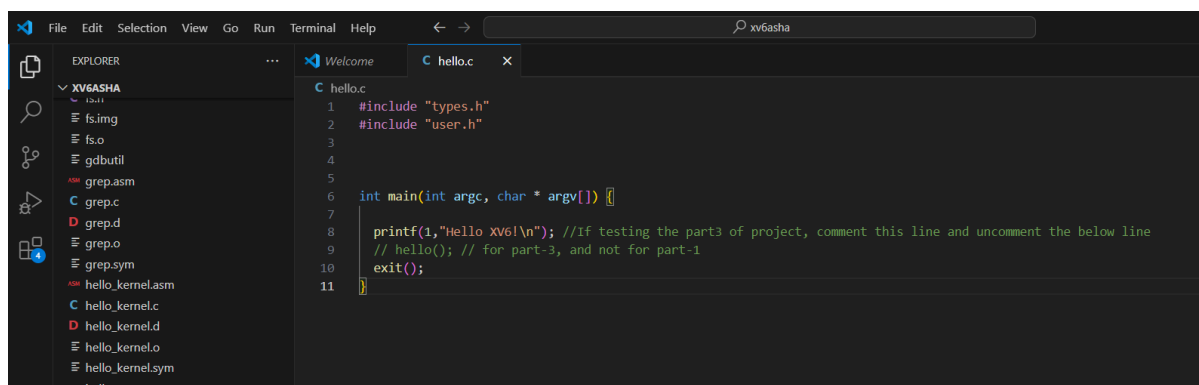
Successfully ran make qemu-nox

```
asha3011@DESKTOP-VJINRSQ x + v
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ |
```

Part-1 -> hello.c



```
File Edit Selection View Go Run Terminal Help
XV6ASHA
  fs.img
  fs.o
  gdbutil
  grep.asm
  grep.c
  grep.d
  grep.o
  grep.sym
  hello_kernel.asm
  hello_kernel.c
  hello_kernel.d
  hello_kernel.o
  hello_kernel.sym
  hello.asm

C hello.c
1 #include "types.h"
2 #include "user.h"
3
4
5
6 int main(int argc, char * argv[]) {
7
8     printf(1, "Hello XV6!\n"); //If testing the part3 of project, comment this line and uncomment the below line
9     // hello(); // for part-3, and not for part-1
10    exit();
11 }
```

Executed hello command

```
asha3011@DESKTOP-VJINRSQ x + v
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ hello
Hello XV6!
$ |
```

part-2 => ls.c

```
Welcome C ls.c X
C ls.c
1  #include "types.h"
2  #include "stat.h"
3  #include "user.h"
4  #include "fs.h"
5
6  int a_flag = 0;
7  int d_flag = 0;
8
9  char*
10 fmtname(char *path)
11 {
12     static char buf[DIRSIZ+1];
13     char *p;
14
15     // Find first character after last slash.
16     for(p=path+strlen(path); p >= path && *p != '/'; p--)
17         ;
18     p++;
19     int len = strlen(p);
20     if (d_flag == T_DIR) {
21         p[len] = '/';
22         p[++len] = '\0';
23     }
24
25     // Return blank-padded name.
26     if(strlen(p) >= DIRSIZ)
27         return p;
28     memmove(buf, p, strlen(p));
29     memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
30     return buf;
31 }
32
```

Written conditions to handle printing of an extra "/" at the end of a folder name, hiding "." files by default, and including "." files when -a flag is entered

```
Welcome C ls.c X
C ls.c
35 {
51
52 switch(st.type){
53     case T_FILE:
54         d_flag = st.type;
55
56         if(a_flag == 1)
57             printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
58         else if(a_flag == 0 && path[0] != '.')
59             printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
60         break;
61
62     case T_DIR:
63         if(strlen(path) + 1 + DIRSIZ + 1 > sizeof(buf)){
64             printf(1, "ls: path too long\n");
65             break;
66         }
67         strcpy(buf, path);
68         p = buf+strlen(buf);
69         *p++ = '/';
70         while(read(fd, &de, sizeof(de)) == sizeof(de)){
71             if(de.inum == 0)
72                 continue;
73             memmove(p, de.name, DIRSIZ);
74             p[DIRSIZ] = 0;
75             if(stat(buf, &st) < 0){
76                 printf(1, "ls: cannot stat %s\n", buf);
77                 continue;
78             }
79             d_flag = st.type;
80             char* f_buffer = fmtname(buf);
81
82             if(a_flag == 1)
83                 printf(1, "%s %d %d %d\n", f_buffer, st.type, st.ino, st.size);
84             else if(a_flag == 0 && f_buffer[0] != '.')
85                 printf(1, "%s %d %d %d\n", f_buffer, st.type, st.ino, st.size);
86 }
```

```
Welcome C ls.c X
C ls.c
35 {
52     switch(st.type){
62         case T_DIR:
70             while(1){
87                 }
88             break;
89     }
90     close(fd);
91 }
92
93 int
94 main(int argc, char *argv[])
95 {
96     int i;
97     if(argc > 1 && argv[1][0] == '-' && argv[1][1] == 'a' && argv[1][2] == '\0')
98     {
99         a_flag = 1;
100         ++ argv;
101         -- argc;
102     }
103
104     if(argc < 2){
105         ls(".");
106         exit();
107     }
108     for(i=1; i<argc; i++)
109         ls(argv[i]);
110     exit();
111 }
112
```

After modifications, ls, by default, hides "." files.

```
asha3011@DESKTOP-VJINRSQ x + v
iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP P

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logs
init: starting sh
$ ls
README          2  2  2286
cat              2  3  15500
echo            2  4  14376
forktest        2  5   8828
grep            2  6  18344
init            2  7  15000
kill            2  8  14464
ln              2  9  14364
ls              2 10  17540
mkdir           2 11  14484
rm              2 12  14468
sh              2 13  28524
stressfs        2 14  15396
usertests       2 15  62900
wc              2 16  15920
zombie          2 17  14048
hello           2 18  14160
hello_kernel    2 19  14024
sleep           2 20  14524
console         3 21   0
$ |
```

and shows "." files when -a flag is set

```
asha3011@DESKTOP-VJINRSQ  ×  +  ∨  
wc                2 16 15920  
zombie            2 17 14048  
hello             2 18 14160  
hello_kernel      2 19 14024  
sleep             2 20 14524  
console           3 21 0  
$ ls -a  
./                1 1 512  
../               1 1 512  
README            2 2 2286  
cat               2 3 15500  
echo              2 4 14376  
forktest          2 5 8828  
grep              2 6 18344  
init              2 7 15000  
kill              2 8 14464  
ln                2 9 14364  
ls                2 10 17540  
mkdir             2 11 14484  
rm                2 12 14468  
sh                2 13 28524  
stressfs          2 14 15396  
usertests         2 15 62900  
wc                2 16 15920  
zombie            2 17 14048  
hello             2 18 14160  
hello_kernel      2 19 14024  
sleep             2 20 14524  
console           3 21 0  
$ |
```

Created a random folder named "abc" in the directory. and executed ls to show that an extra forward slash("/") is appended for names of folders.

```
asha3011@DESKTOP-VJINRSQ x
+ v

wc          2 16 15920
zombie     2 17 14048
hello      2 18 14160
hello_kernel 2 19 14024
sleep      2 20 14524
console    3 21 0
$ mkdir abc/
$ ls
README     2 2 2286
cat        2 3 15500
echo       2 4 14376
forktest   2 5 8828
grep       2 6 18344
init       2 7 15000
kill       2 8 14464
ln         2 9 14364
ls         2 10 17540
mkdir      2 11 14484
rm         2 12 14468
sh         2 13 28524
stressfs   2 14 15396
usertests  2 15 62900
wc         2 16 15920
zombie     2 17 14048
hello      2 18 14160
hello_kernel 2 19 14024
sleep      2 20 14524
console    3 21 0
abc/       1 22 32
$ |
```

Part-3 => hello system call

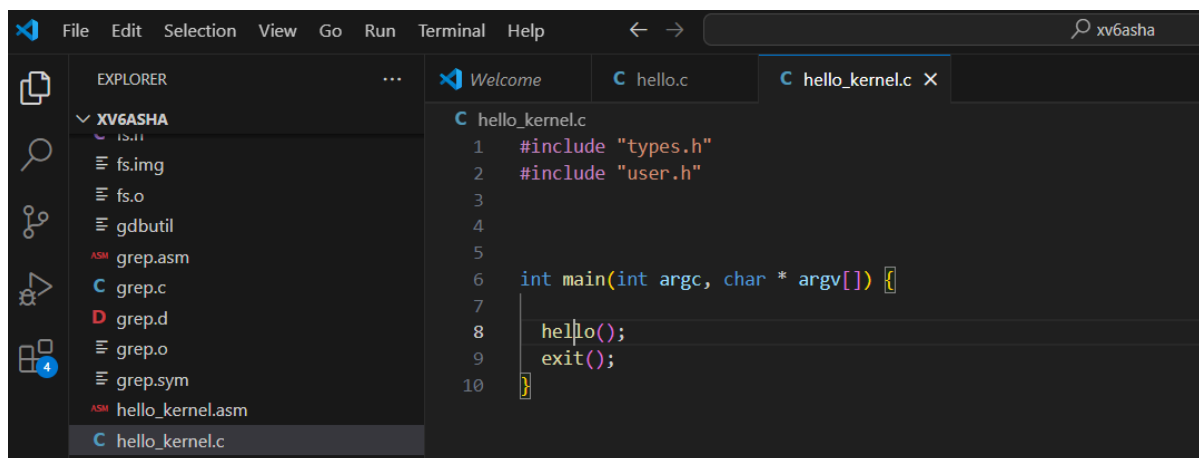
Created a system call for hello.

Used the same file hello.c, where a call to hello system call is made.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
  xv6asha
    fs.img
    fs.o
    gdbutil
    grep.asm
    C: hello.c
    D: grep.d
    E: grep.o
    E: grep.sym
    C: hello_kernel.asm
    C: hello_kernel.c
    D: hello_kernel.d
    E: hello_kernel.o

C: hello.c
1 #include "types.h"
2 #include "user.h"
3
4
5
6 int main(int argc, char * argv[]) {
7
8     // printf(1, "Hello xv6!\n"); //If testing the part3 of project, comment this line and uncomment the below line
9     hello(); // for part-3, and not for part-1
10    exit();
11 }
```

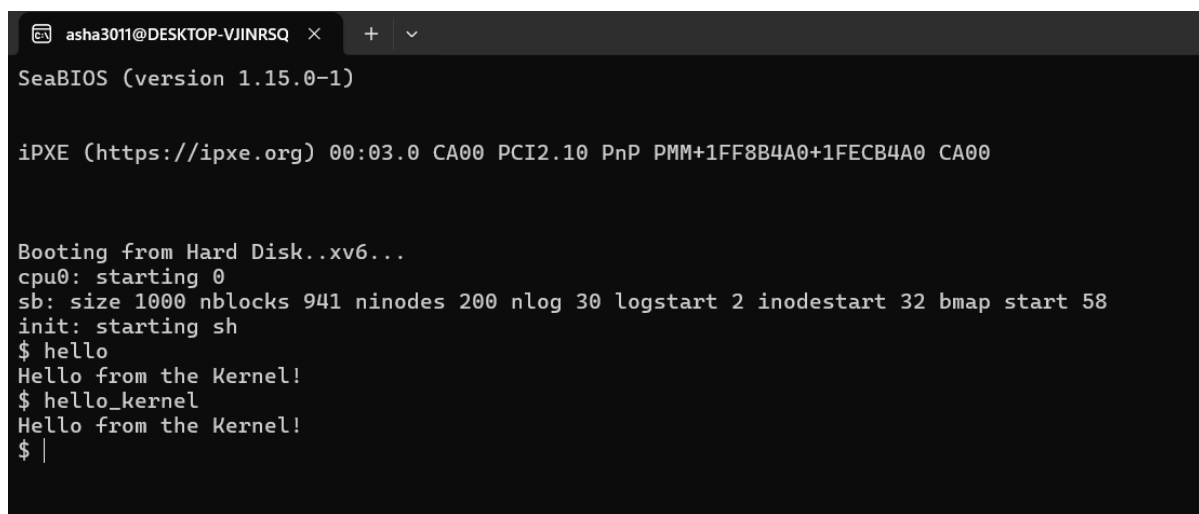

Created a separate file for part-3 demonstration which makes a call to the created hello system call.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the file structure of the 'XV6ASHA' project, including files like fs.img, fs.o, gdbutil, grep.asm, grep.c, grep.d, grep.o, grep.sym, hello_kernel.asm, and hello_kernel.c. The main editor area shows the 'hello_kernel.c' file with the following code:

```
1 #include "types.h"
2 #include "user.h"
3
4
5
6 int main(int argc, char * argv[]) {
7
8     hello();
9     exit();
10 }
```

executed hello commands for part-3



The screenshot shows a terminal window with the following output:

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00

Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ hello
Hello from the Kernel!
$ hello_kernel
Hello from the Kernel!
$ |
```

Below are the screenshots of code additions done in certain files for creating the hello system call.

ASM usys.S

```
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)
19  SYSCALL(exec)
20  SYSCALL(open)
21  SYSCALL(mknod)
22  SYSCALL(unlink)
23  SYSCALL(fstat)
24  SYSCALL(link)
25  SYSCALL(mkdir)
26  SYSCALL(chdir)
27  SYSCALL(dup)
28  SYSCALL(getpid)
29  SYSCALL(sbrk)
30  SYSCALL(sleep)
31  SYSCALL(uptime)
32  SYSCALL(hello)
33
```



Welcome

C hello.c

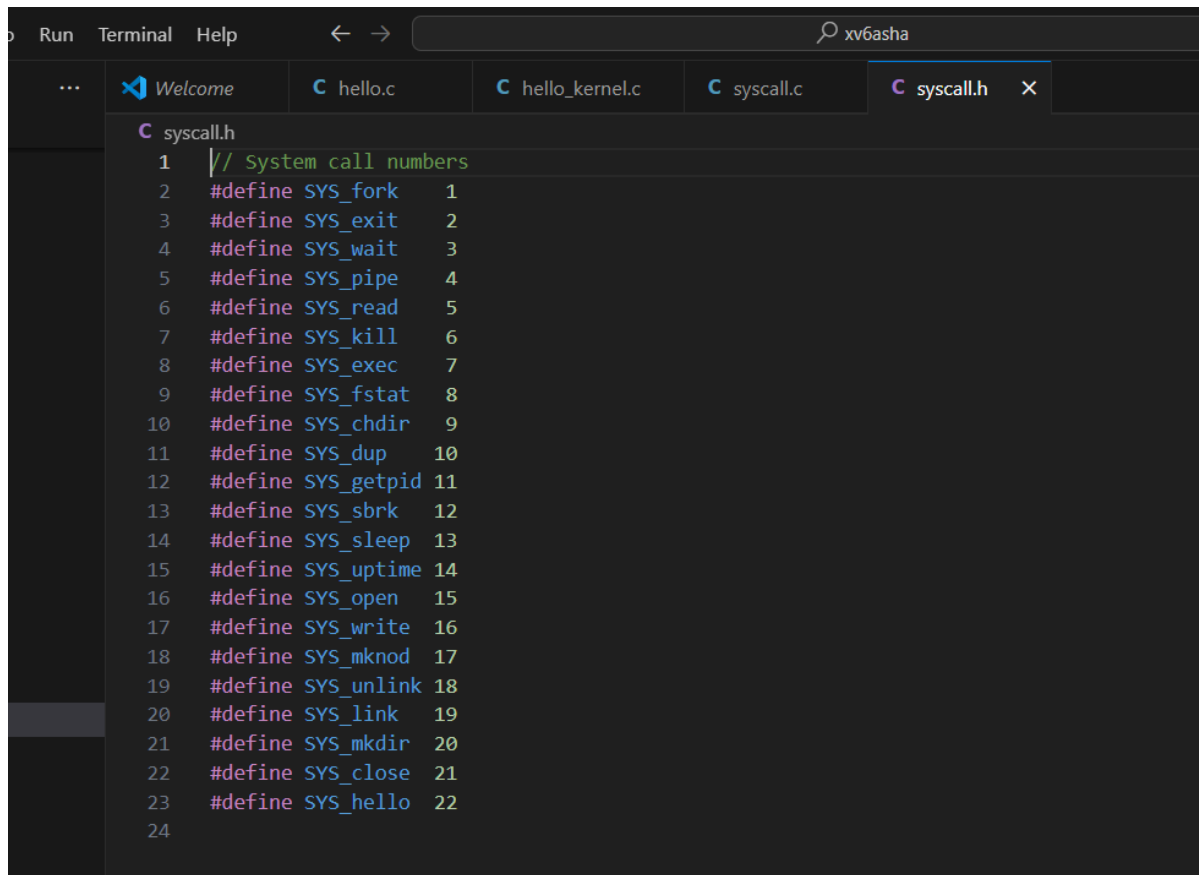
C hello_kernel.c

C syscall.c



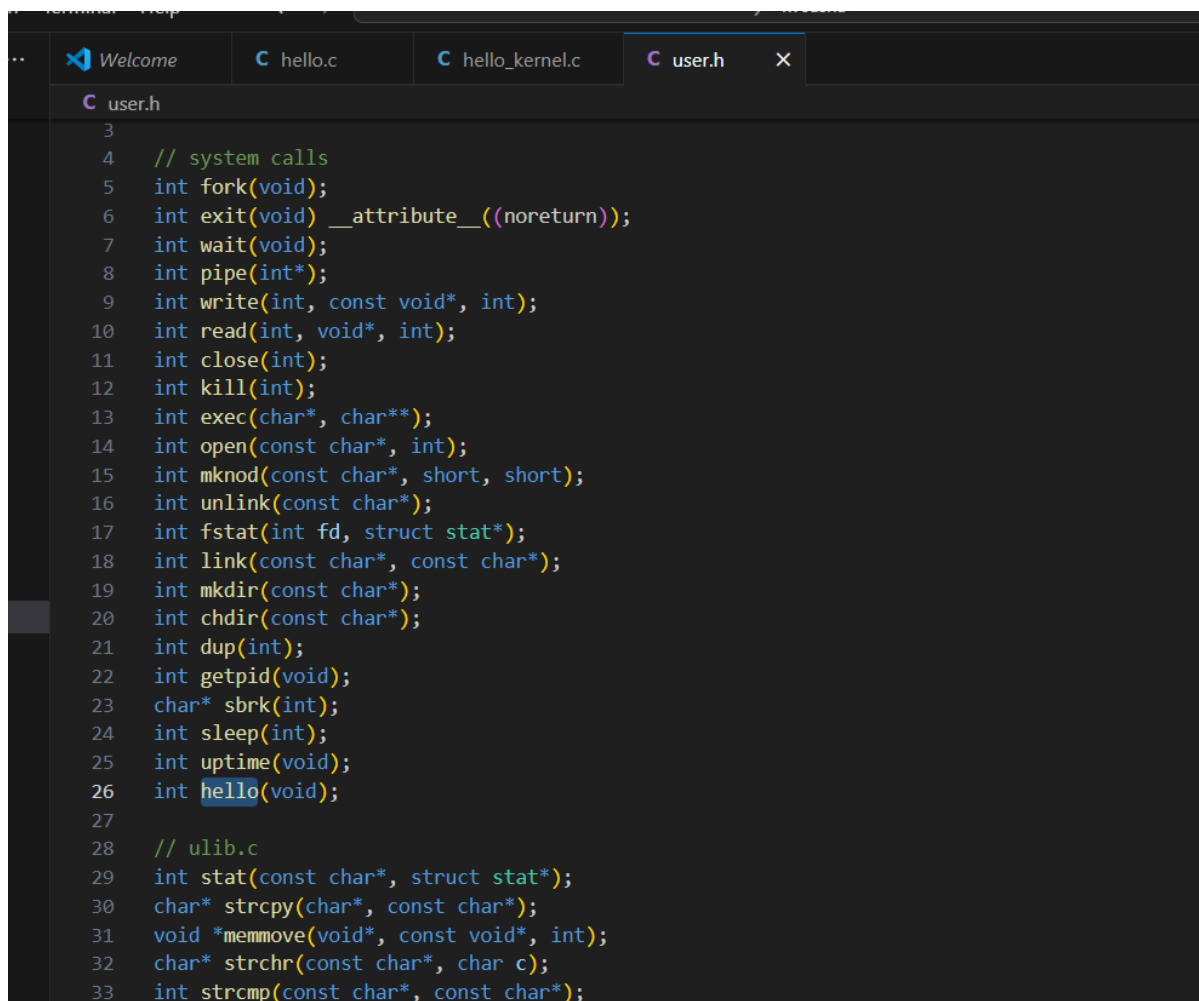
C syscall.c

```
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_hello(void);
107
108 static int (*syscalls[])(void) = {
109     [SYS_fork]    sys_fork,
110     [SYS_exit]    sys_exit,
111     [SYS_wait]    sys_wait,
112     [SYS_pipe]    sys_pipe,
113     [SYS_read]    sys_read,
114     [SYS_kill]    sys_kill,
115     [SYS_exec]    sys_exec,
116     [SYS_fstat]   sys_fstat,
117     [SYS_chdir]   sys_chdir,
118     [SYS_dup]     sys_dup,
119     [SYS_getpid]  sys_getpid,
120     [SYS_sbrk]    sys_sbrk,
121     [SYS_sleep]   sys_sleep,
122     [SYS_uptime]  sys_uptime,
123     [SYS_open]    sys_open,
124     [SYS_write]   sys_write,
125     [SYS_mknod]   sys_mknod,
126     [SYS_unlink]  sys_unlink,
127     [SYS_link]    sys_link,
128     [SYS_mkdir]   sys_mkdir,
129     [SYS_close]   sys_close,
130     [SYS_hello]   sys_hello,
131     };
132
```



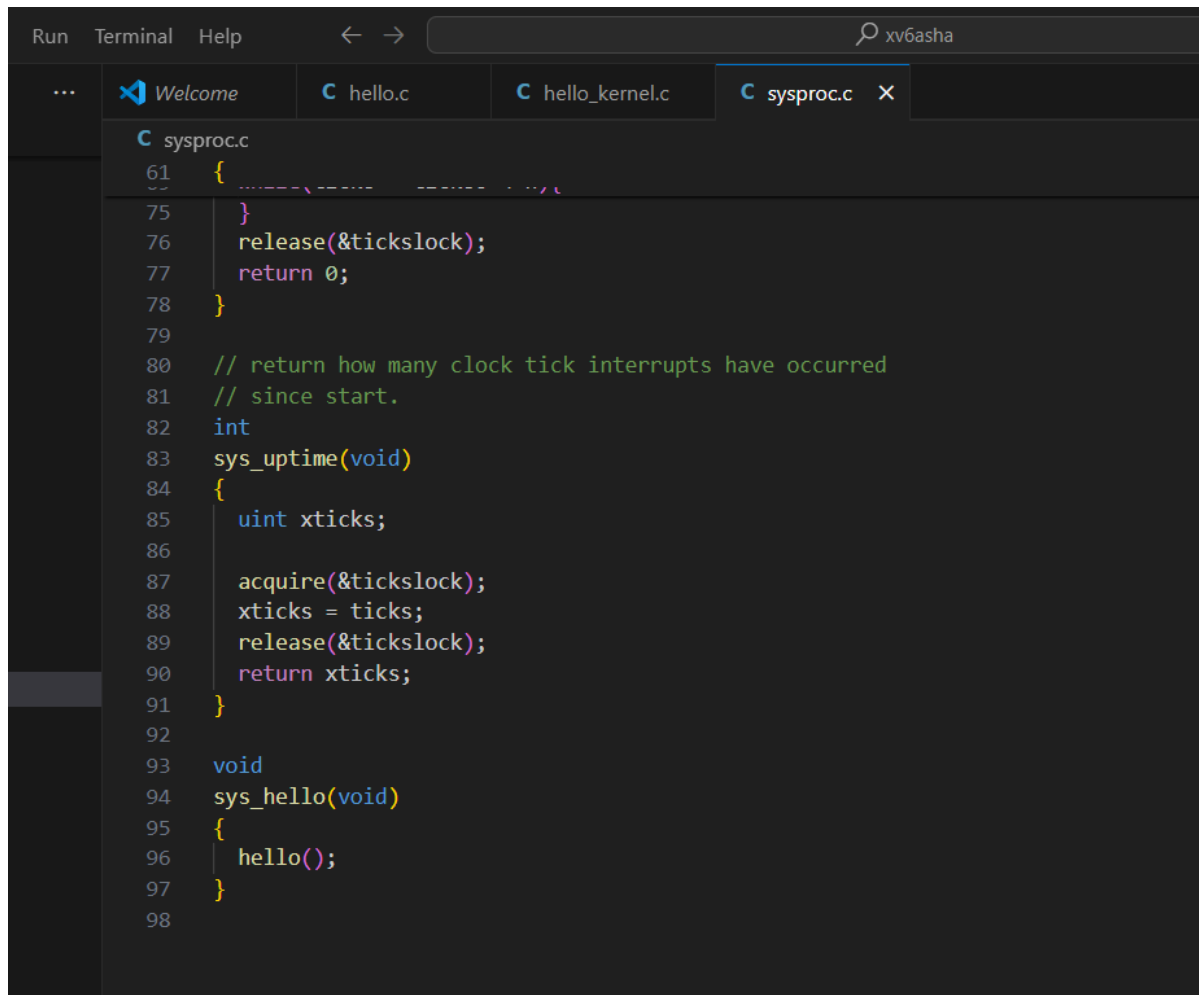
```
1 // System call numbers
2 #define SYS_fork 1
3 #define SYS_exit 2
4 #define SYS_wait 3
5 #define SYS_pipe 4
6 #define SYS_read 5
7 #define SYS_kill 6
8 #define SYS_exec 7
9 #define SYS_fstat 8
10 #define SYS_chdir 9
11 #define SYS_dup 10
12 #define SYS_getpid 11
13 #define SYS_sbrk 12
14 #define 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
24
```

user.h includes all the system calls that a user program can make use of.



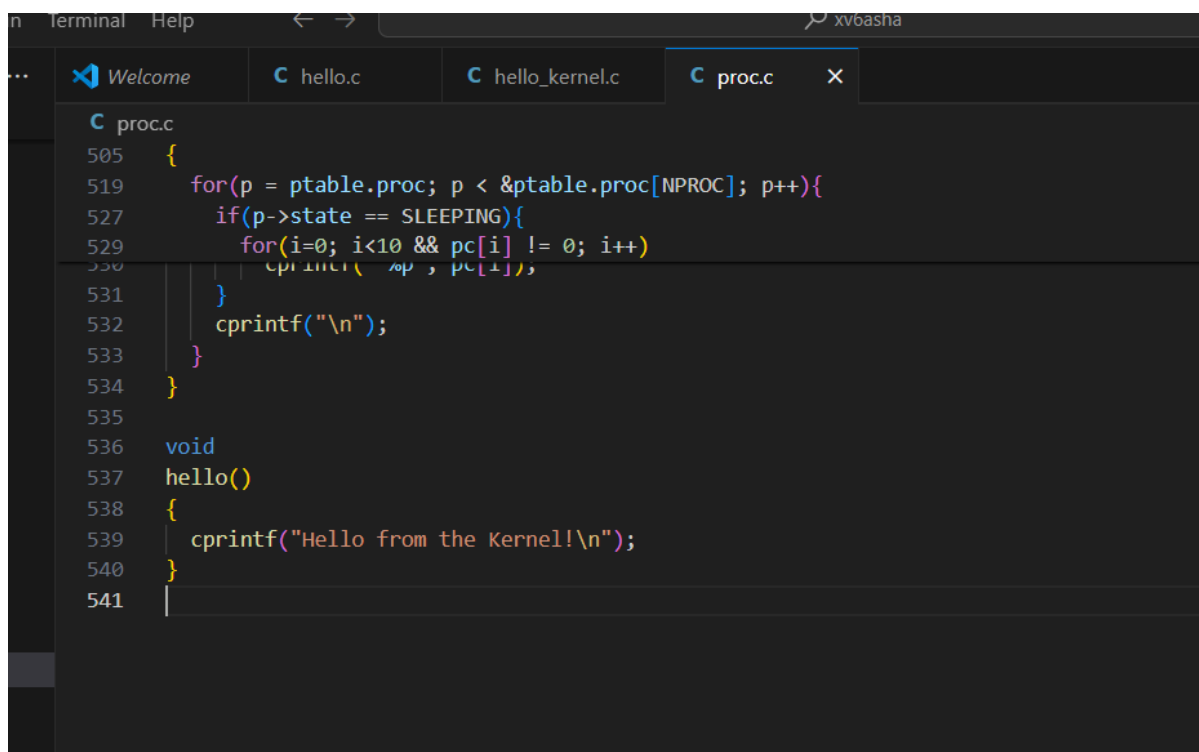
```
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const 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);
27
28 // ulib.c
29 int stat(const char*, struct stat*);
30 char* strcpy(char*, const char*);
31 void *memmove(void*, const void*, int);
32 char* strchr(const char*, char c);
33 int strcmp(const char*, const char*);
```

sys_hello functions in return makes a call to hello function written in proc.c



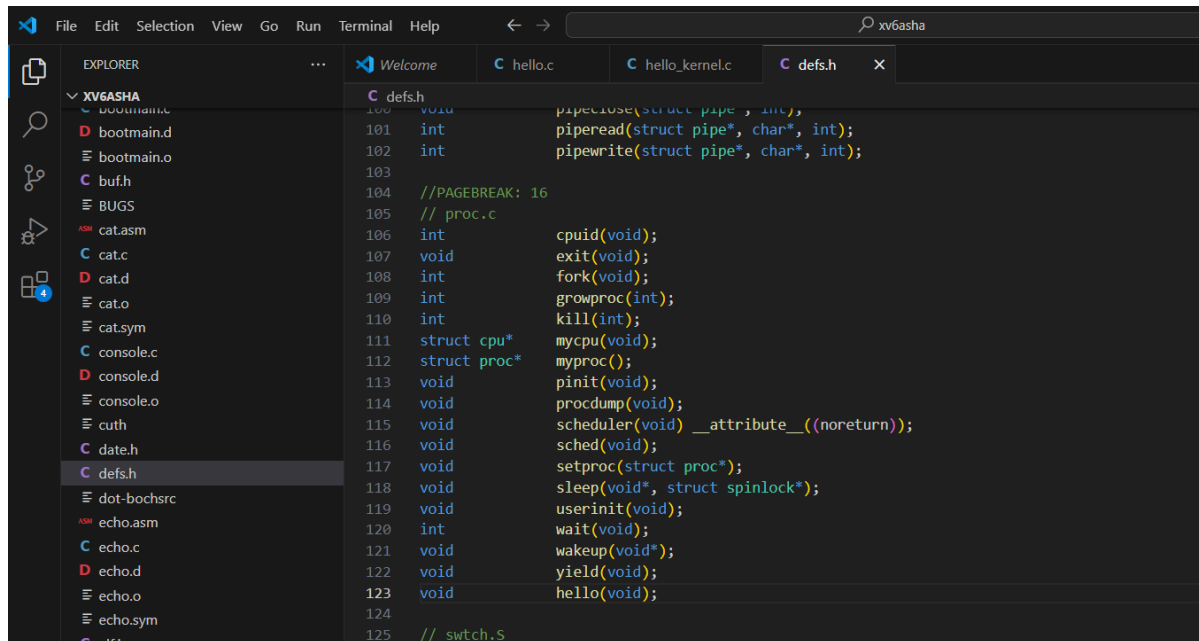
```
61 {
62     // ...
63 }
64
65 // return how many clock tick interrupts have occurred
66 // since start.
67 int
68 sys_uptime(void)
69 {
70     uint xticks;
71
72     acquire(&tickslock);
73     xticks = ticks;
74     release(&tickslock);
75     return xticks;
76 }
77
78 void
79 sys_hello(void)
80 {
81     hello();
82 }
```

core logic of hello system call, just like most of the system calls, is written in proc.c



```
505 {
506     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
507         if(p->state == SLEEPING){
508             for(i=0; i<10 && pc[i] != 0; i++)
509                 cprintf("sp, pc[i],");
510         }
511         cprintf("\n");
512     }
513 }
514
515 void
516 hello()
517 {
518     cprintf("Hello from the Kernel!\n");
519 }
520
521
```

The definition of hello function of proc.c is included in defs.h header file. To make a call to this function, defs.h file has to be included.



The screenshot shows the Visual Studio Code editor with the 'defs.h' file open. The Explorer sidebar on the left shows a project named 'XV6ASHA' with various files including 'defs.h'. The main editor window displays the contents of 'defs.h', which includes declarations for pipe-related functions, process management functions, and a 'hello' function. The code is as follows:

```
100 void    pipeclose(struct pipe*, int);
101 int      piperead(struct pipe*, char*, int);
102 int      pipewrite(struct pipe*, char*, int);
103
104 //PAGEBREAK: 16
105 // proc.c
106 int      cpuid(void);
107 void     exit(void);
108 int      fork(void);
109 int      growproc(int);
110 int      kill(int);
111 struct cpu* mycpu(void);
112 struct proc* myproc();
113 void     pinit(void);
114 void     procdump(void);
115 void     scheduler(void) __attribute__((noreturn));
116 void     sched(void);
117 void     setproc(struct proc*);
118 void     sleep(void*, struct spinlock*);
119 void     userinit(void);
120 int      wait(void);
121 void     wakeup(void*);
122 void     yield(void);
123 void     hello(void);
124
125 // switch.s
```

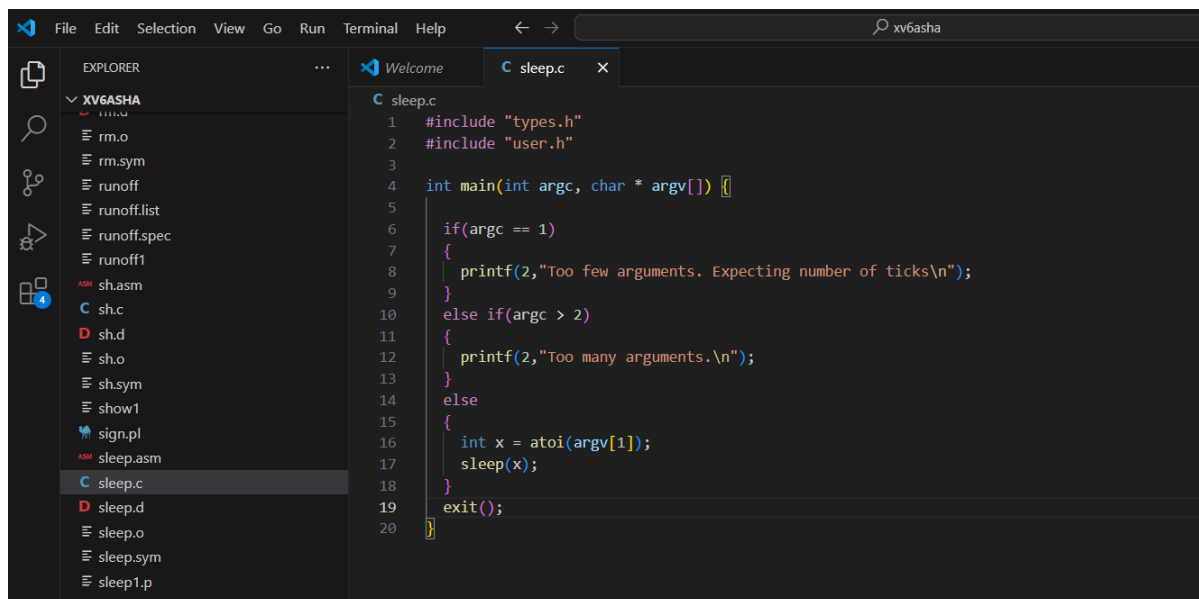
Part-4 => sleep.c

Read ticks from command line arguments.

if ticks are not entered by the user or extra unnecessary arguments are given, an appropriate error message is shown.

else the program runs successfully, waits for ticks/100 seconds and exits without any errors.

note: 100 ticks = 1 second



The screenshot shows the Visual Studio Code editor with the 'sleep.c' file open. The Explorer sidebar on the left shows the 'XV6ASHA' project with 'sleep.c' selected. The main editor window displays the code for 'sleep.c', which includes headers for 'types.h' and 'user.h', and implements a 'main' function that checks command-line arguments for the number of ticks to sleep. The code is as follows:

```
1 #include "types.h"
2 #include "user.h"
3
4 int main(int argc, char * argv[]) {
5
6     if(argc == 1)
7     {
8         printf(2, "Too few arguments. Expecting number of ticks\n");
9     }
10    else if(argc > 2)
11    {
12        printf(2, "Too many arguments.\n");
13    }
14    else
15    {
16        int x = atoi(argv[1]);
17        sleep(x);
18    }
19    exit();
20 }
```

SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00

Booting from Hard Disk..xv6...

cpu0: starting 0

sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58

init: starting sh

\$ sleep 500

\$ sleep

Too few arguments. Expecting number of ticks

\$ sleep 20 2

Too many arguments.

\$ |