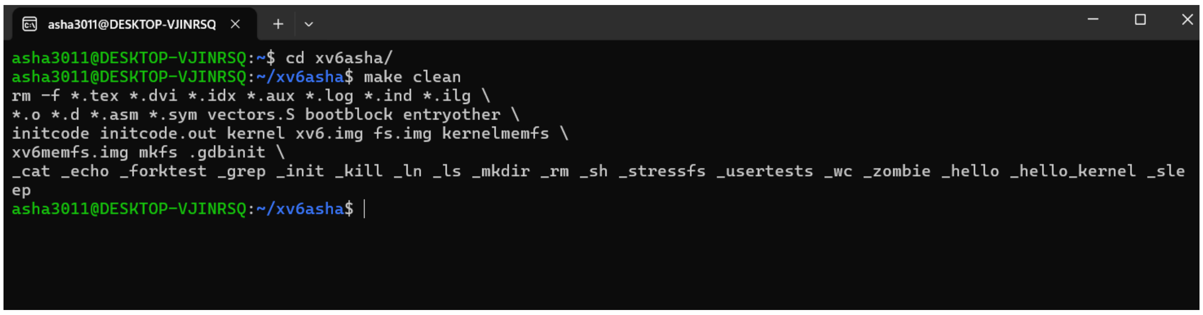
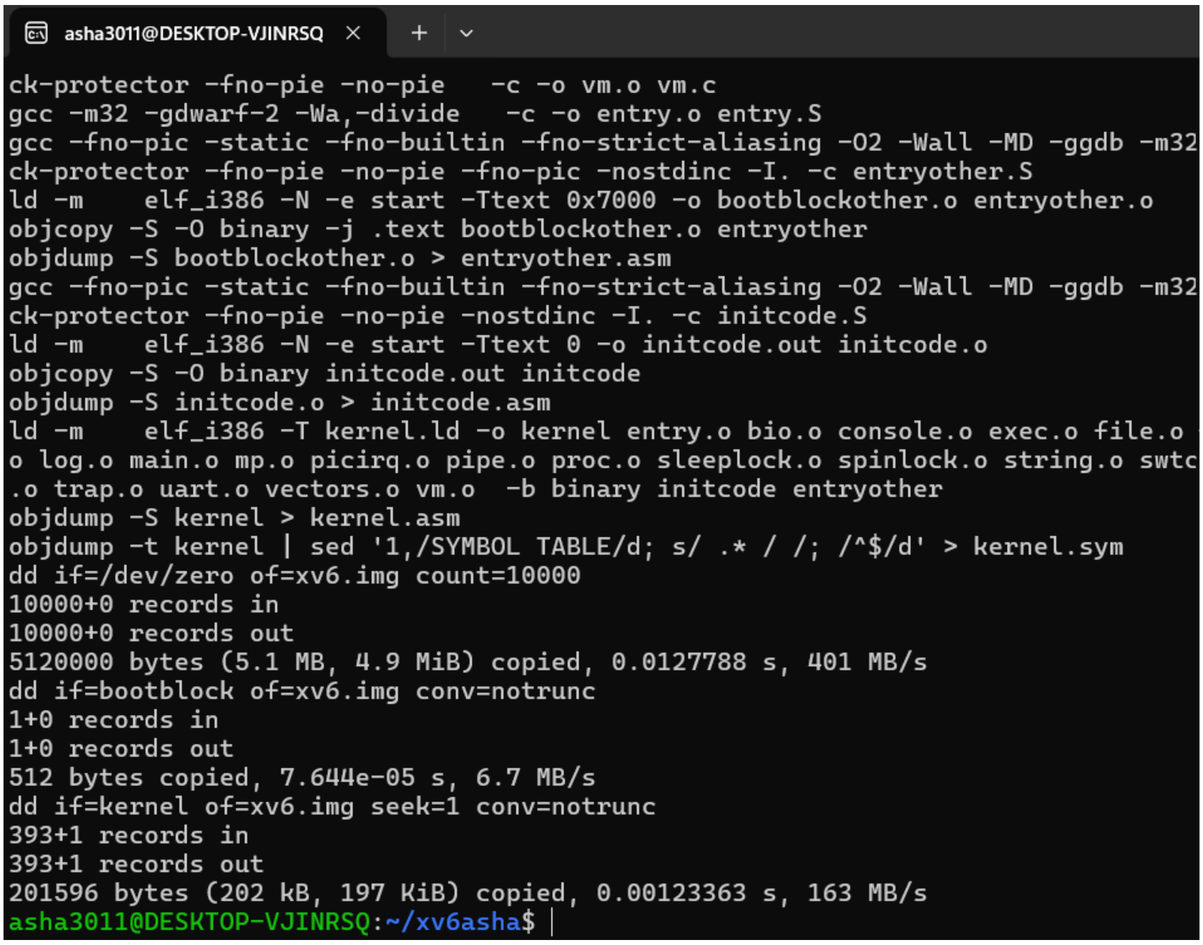
Project-1 Report

Successfully ran make clean



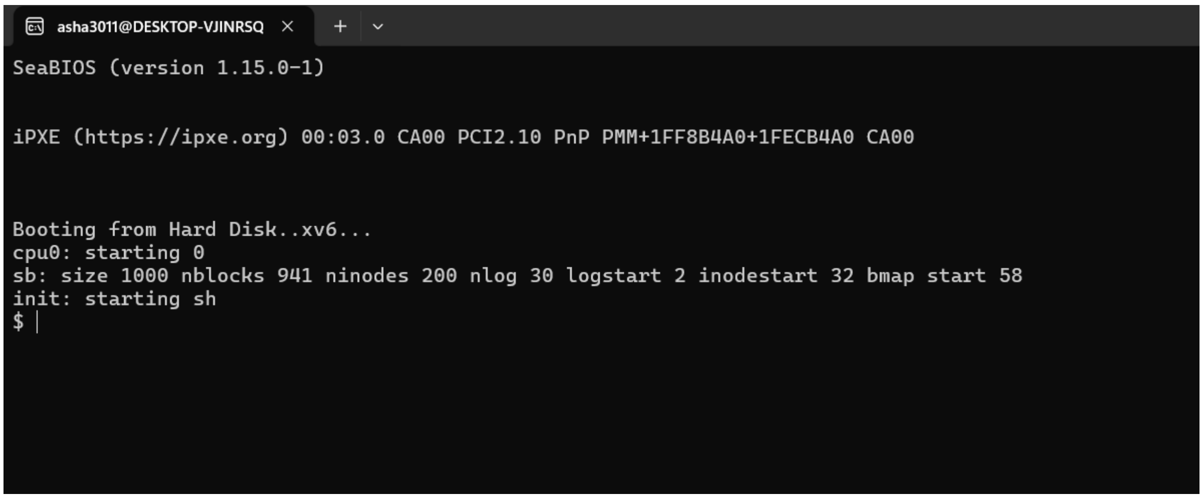
```
asha3011@DESKTOP-VJINRSQ:~$ cd xv6asha/
asha3011@DESKTOP-VJINRSQ:~/xv6asha$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest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

Successfully ran make



```
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 swtc
.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/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0127788 s, 401 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 7.644e-05 s, 6.7 MB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
393+1 records in
393+1 records out
201596 bytes (202 kB, 197 KiB) copied, 0.00123363 s, 163 MB/s
asha3011@DESKTOP-VJINRSQ:~/xv6asha$ |
```

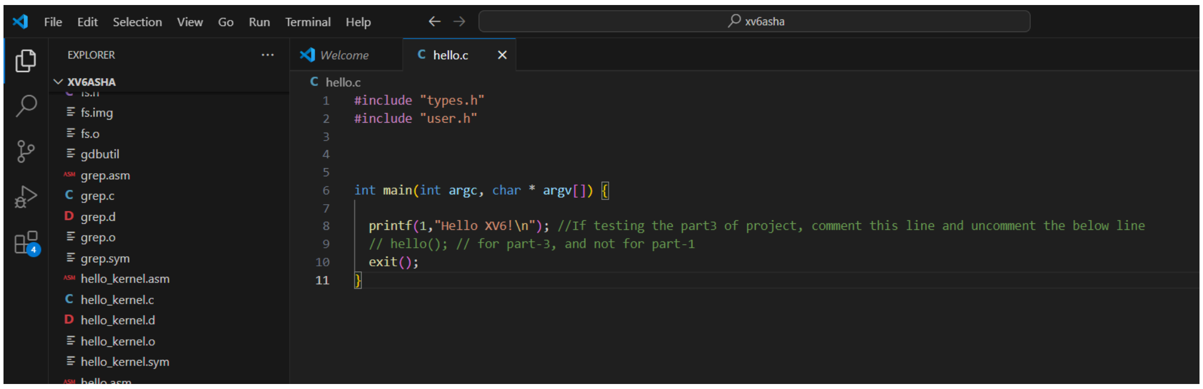
Successfully ran make qemu-nox



asha3011@DESKTOP-VJINRSQ

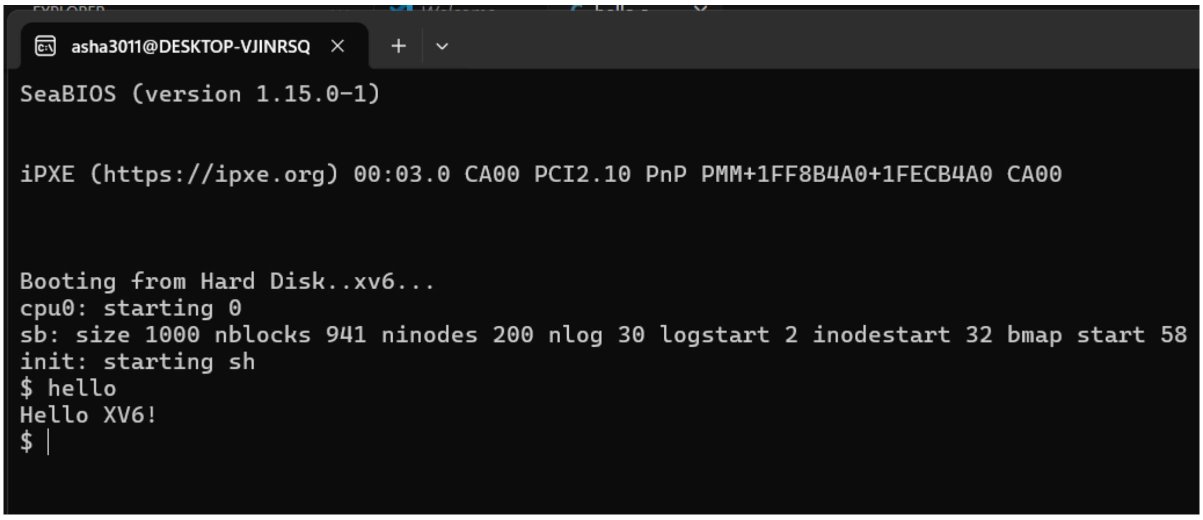
```
SeaBIOS (version 1.15.0-1)


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



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



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

  printf(1,"Hello XV6!\n"); //If testing the part3 of project, comment this line and uncomment the below line
  // hello(); // for part-3, and not for part-1
  exit();
}
```

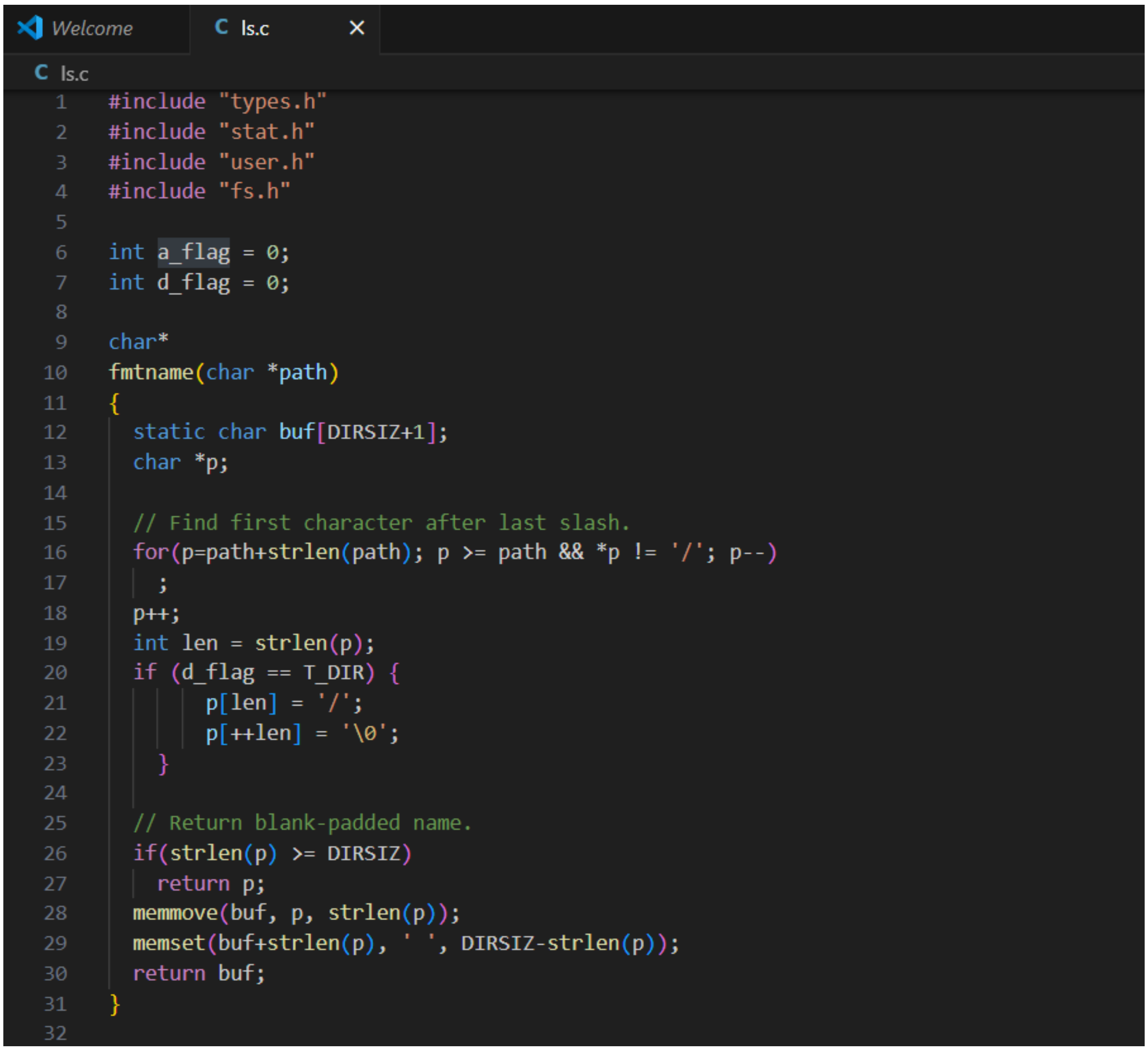
Executed hello command



asha3011@DESKTOP-VJINRSQ

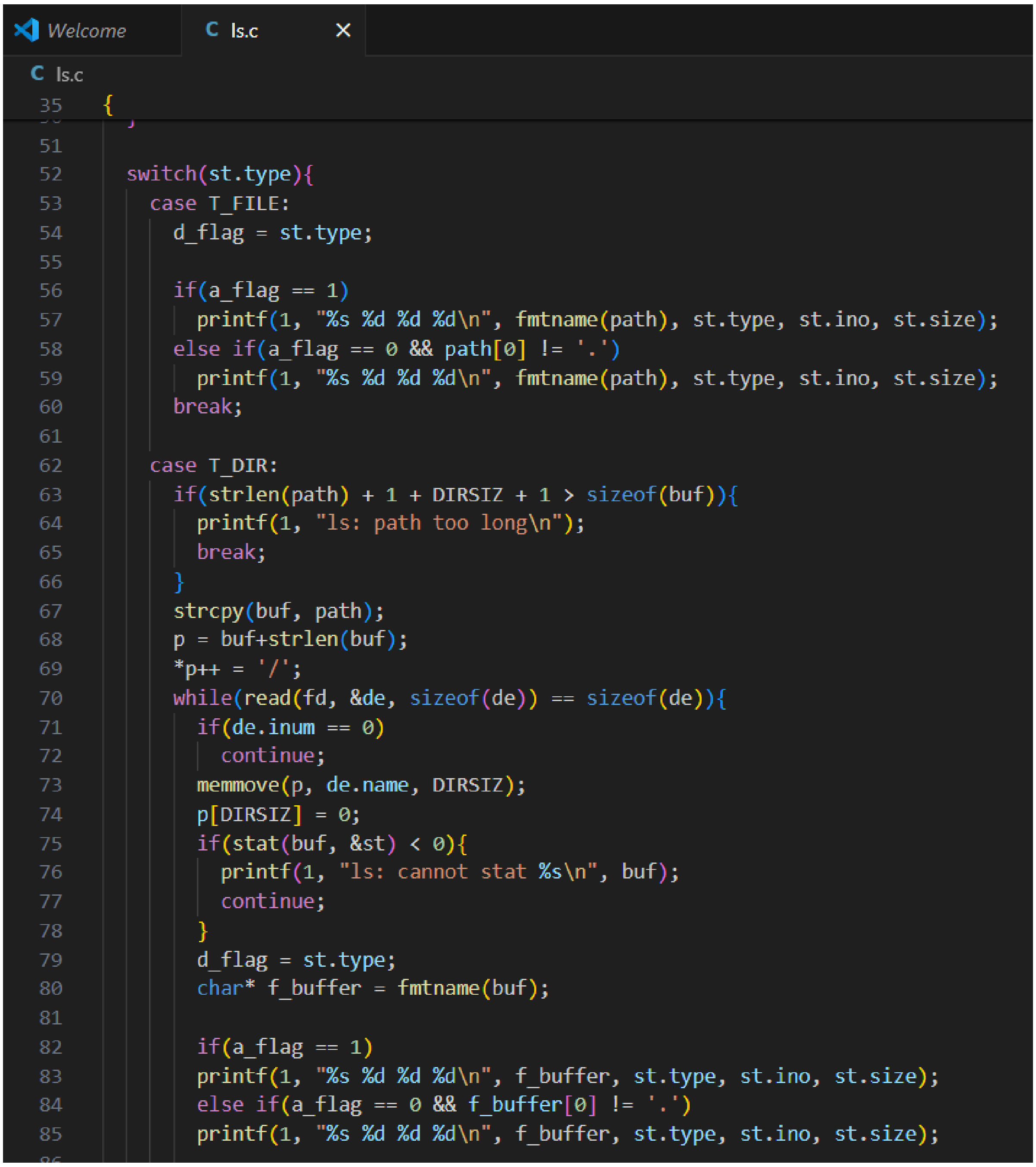
```
SeaBIOS (version 1.15.0-1)


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

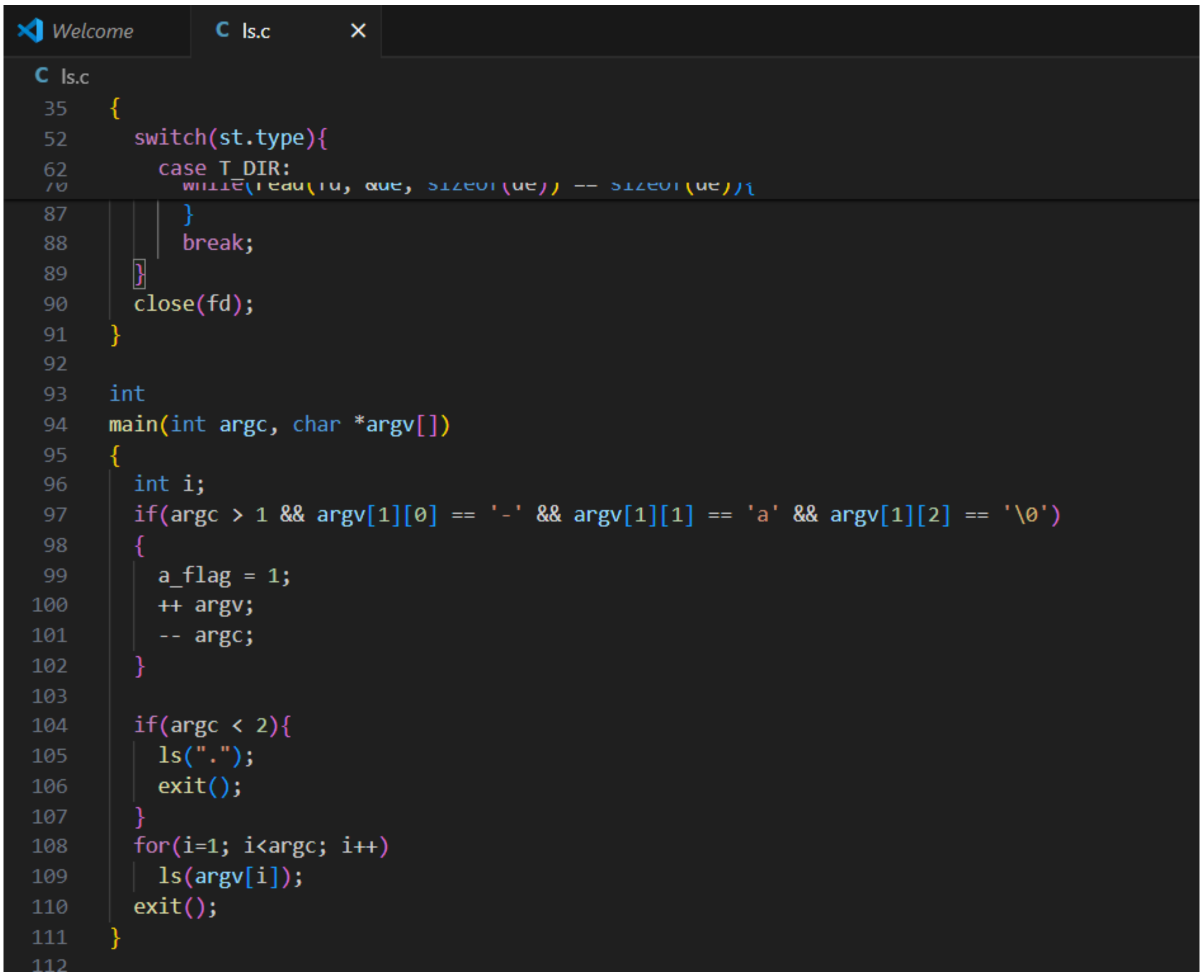


```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fs.h"

int a_flag = 0;
int d_flag = 0;

char*
fmtname(char *path)
{
  static char buf[DIRSIZ+1];
  char *p;

  // Find first character after last slash.
  for(p=path+strlen(path); p >= path && *p != '/'; p--)
    ;
  p++;
  int len = strlen(p);
  if (d_flag == T_DIR) {
      p[len] = '/';
      p[++len] = '\0';
    }

  // Return blank-padded name.
  if(strlen(p) >= DIRSIZ)
    return p;
  memmove(buf, p, strlen(p));
  memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));
  return buf;
}

```

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

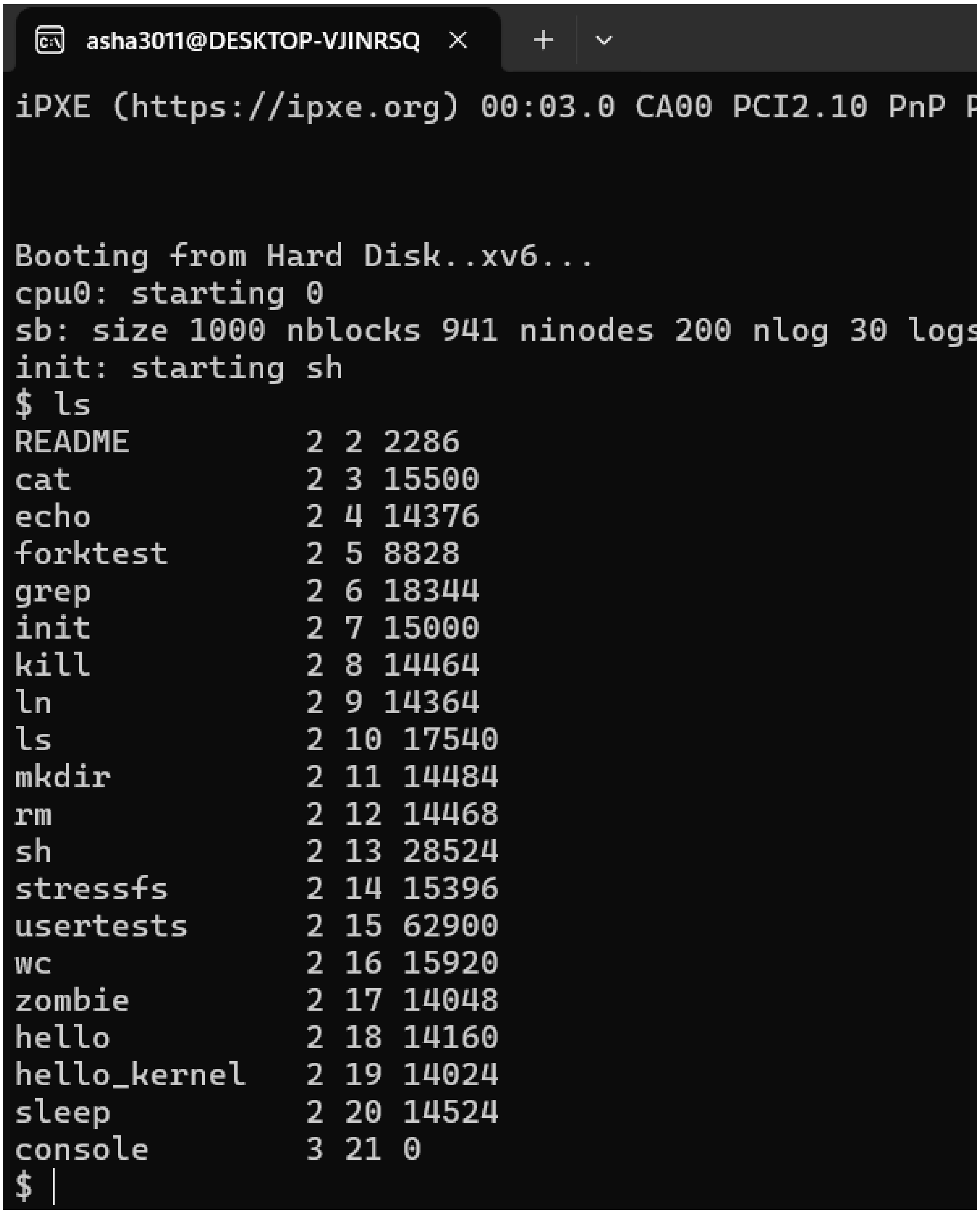


```c
{

  switch(st.type){
    case T_FILE:
      d_flag = st.type;

      if(a_flag == 1)
        printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
      else if(a_flag == 0 && path[0] != '.')
        printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
      break;

    case T_DIR:
      if(strlen(path) + 1 + DIRSIZ + 1 > sizeof(buf)){
        printf(1, "ls: path too long\n");
        break;
      }
      strcpy(buf, path);
      p = buf+strlen(buf);
      *p++ = '/';
      while(read(fd, &de, sizeof(de)) == sizeof(de)){
        if(de.inum == 0)
          continue;
        memmove(p, de.name, DIRSIZ);
        p[DIRSIZ] = 0;
        if(stat(buf, &st) < 0){
          printf(1, "ls: cannot stat %s\n", buf);
          continue;
        }
        d_flag = st.type;
        char* f_buffer = fmtname(buf);

        if(a_flag == 1)
        printf(1, "%s %d %d %d\n", f_buffer, st.type, st.ino, st.size);
        else if(a_flag == 0 && f_buffer[0] != '.')
        printf(1, "%s %d %d %d\n", f_buffer, st.type, st.ino, st.size);
```



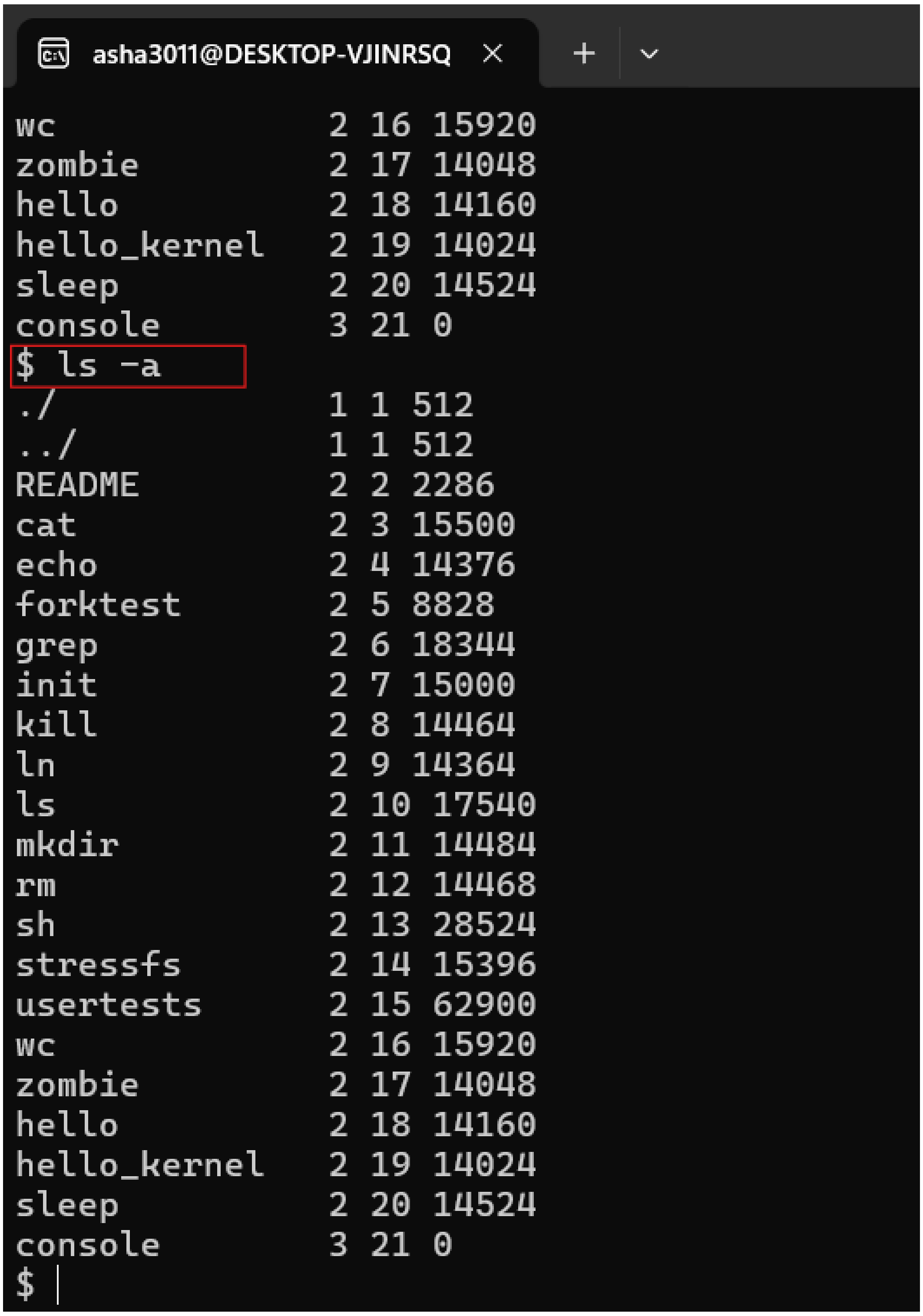
```c
{
  switch(st.type){
    case T_DIR:
      while(read(fd, &de, sizeof(de)) == sizeof(de)){
      }
      break;
  }
  close(fd);
}

int
main(int argc, char *argv[])
{
  int i;
  if(argc > 1 && argv[1][0] == '-' && argv[1][1] == 'a' && argv[1][2] == '\0')
  {
    a_flag = 1;
    ++ argv;
    -- argc;
  }

  if(argc < 2){
    ls(".");
    exit();
  }
  for(i=1; i<argc; i++)
    ls(argv[i]);
  exit();
}
```

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



```
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
$ 
```

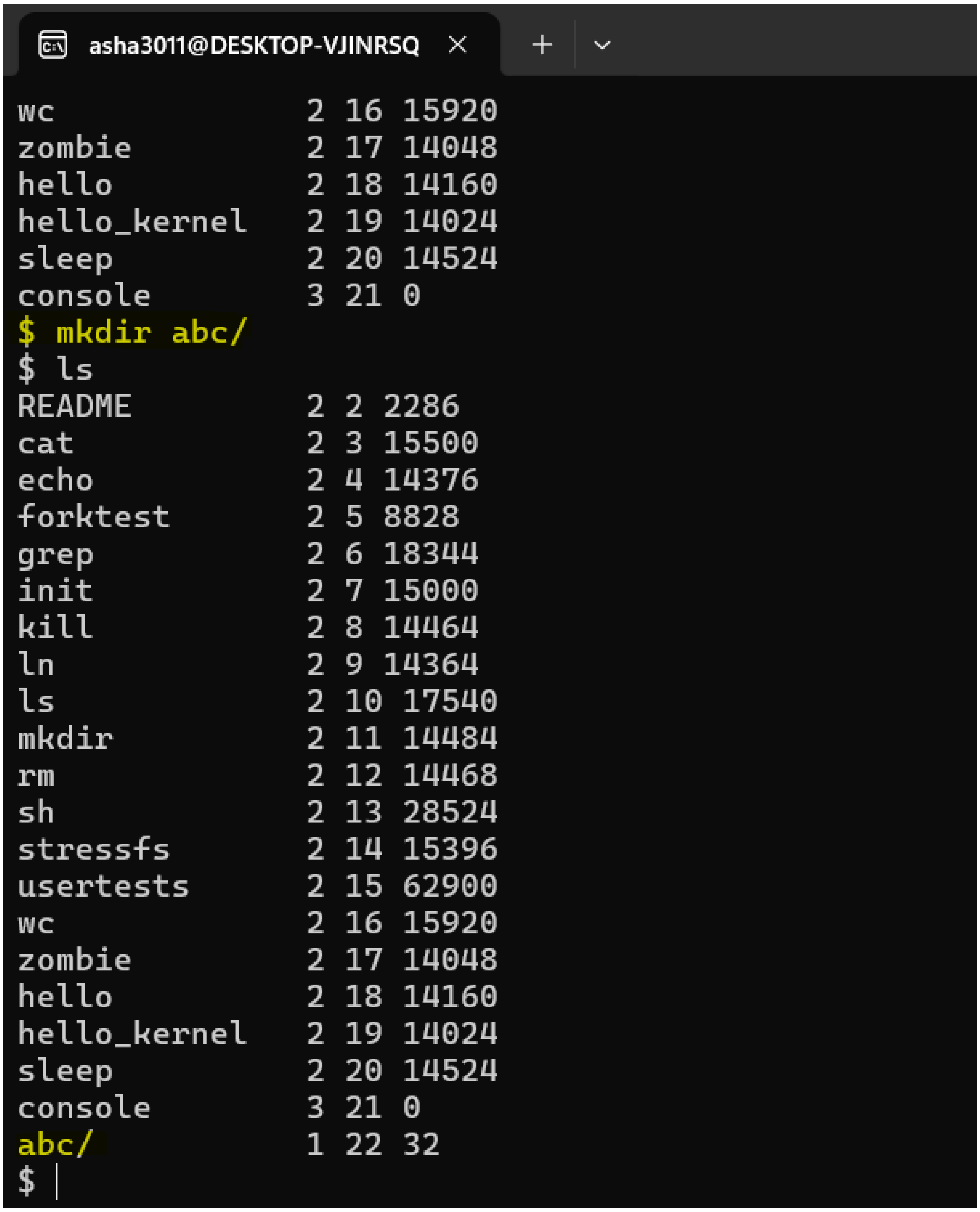
and shows "." files when -a flag is set



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.

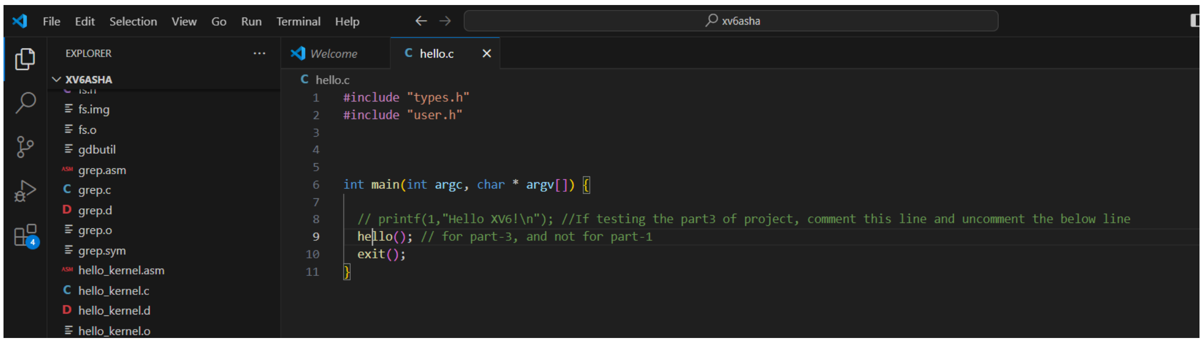


```
wc             2 16 15920
zombie         2 17 14048
hello          2 18 14160
hello_kernel   2 19 14024
sleep          2 20 14524
console        3 21 0
$ mkdir abc/
$ ls
README         2 2 2286
cat            2 3 15500
echo           2 4 14376
forktest       2 5 8828
grep           2 6 18344
init           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

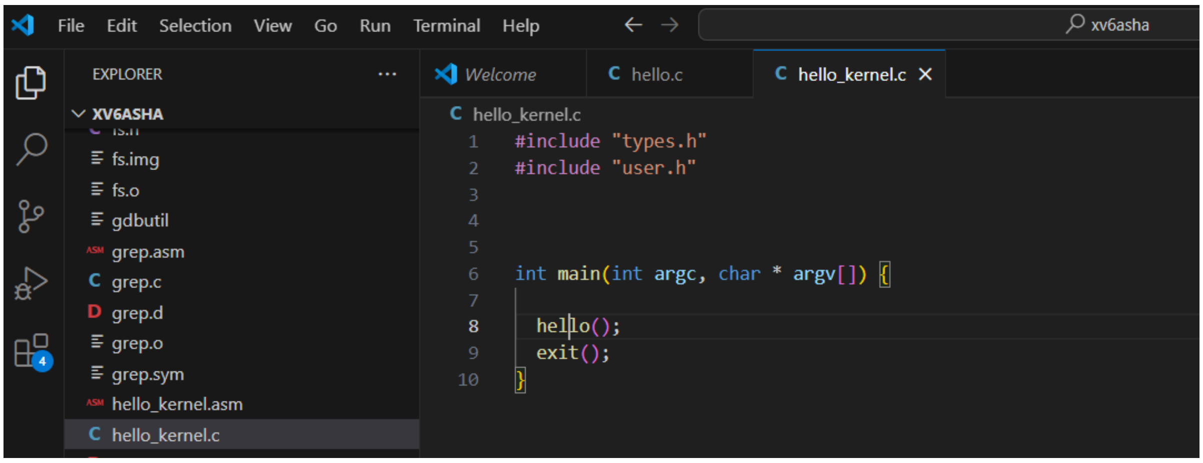
Created a system call for hello.

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



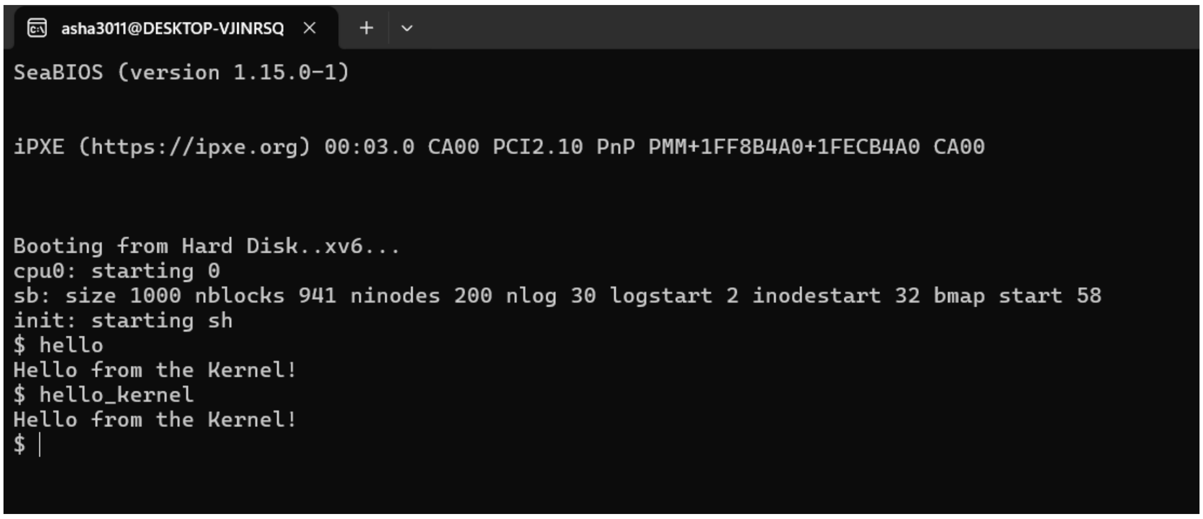
```c
#include "types.h"
#include "user.h"



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

  // printf(1,"Hello XV6!\n"); //If testing the part3 of project, comment this line and uncomment the below line
  hello(); // for part-3, and not for part-1
  exit();
}
```

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



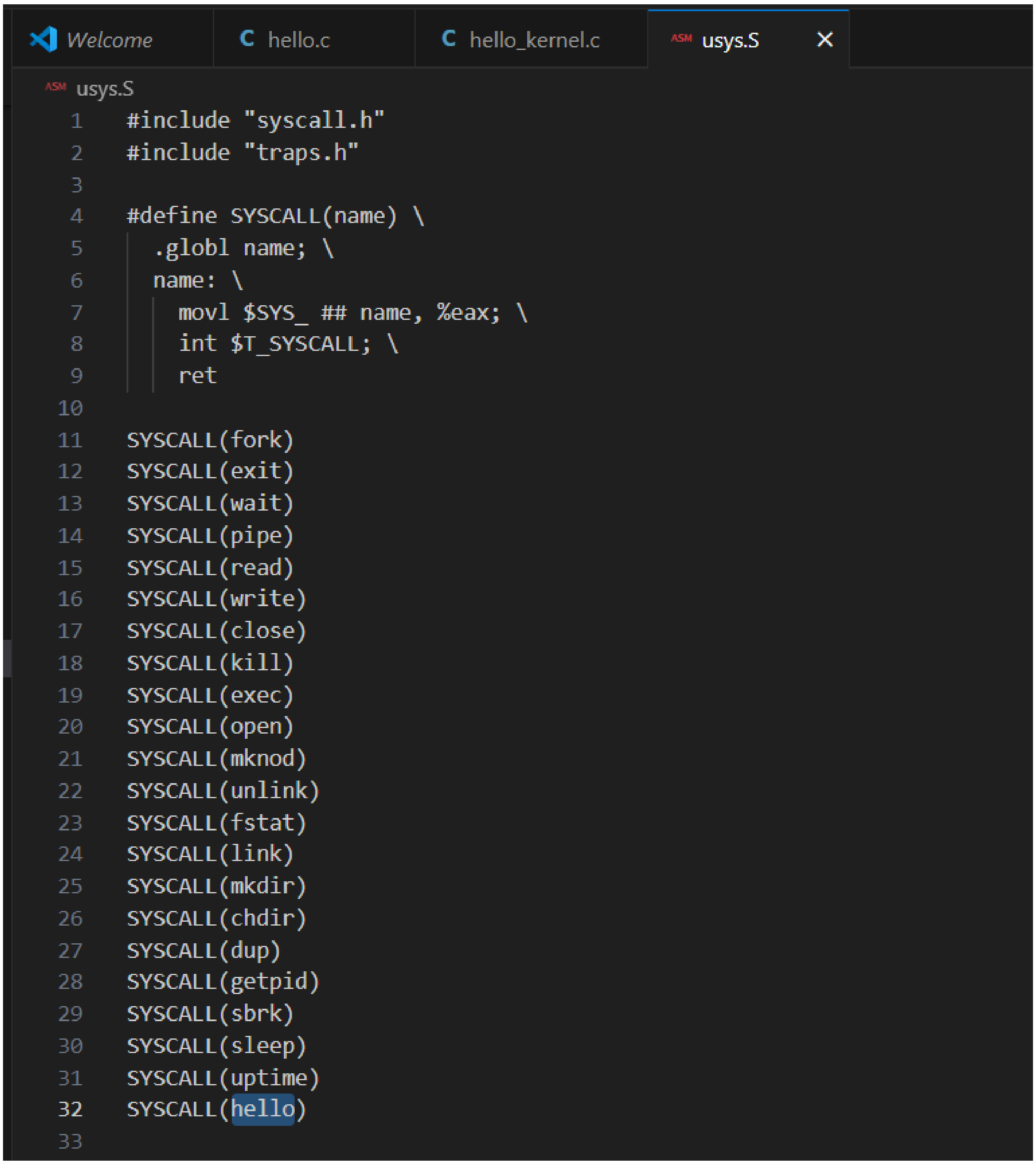
```c
#include "types.h"
#include "user.h"



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

    hello();
    exit();
}
```

executed hello commands for part-3

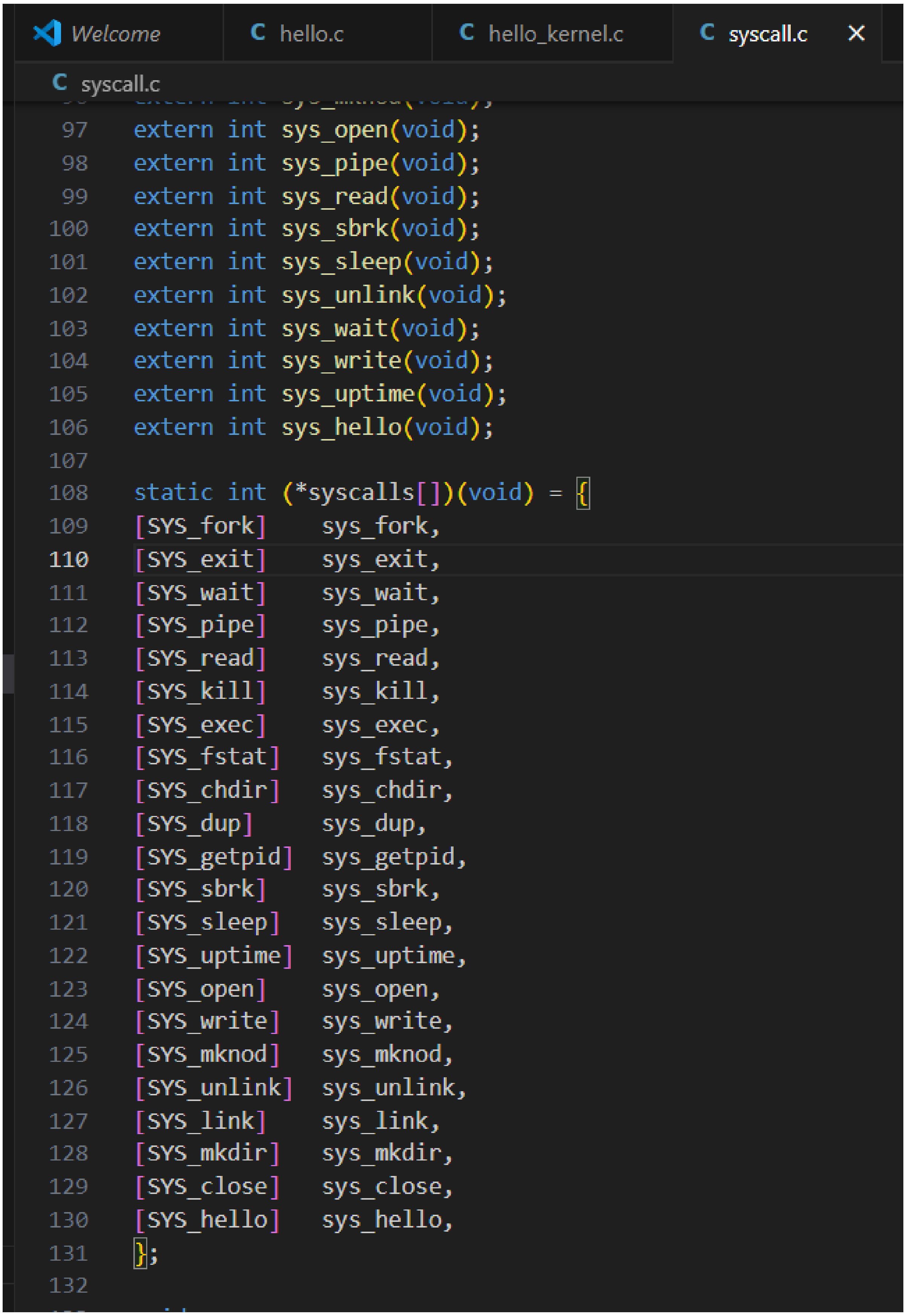


```
SeaBIOS (version 1.15.0-1)


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!
$ |
```

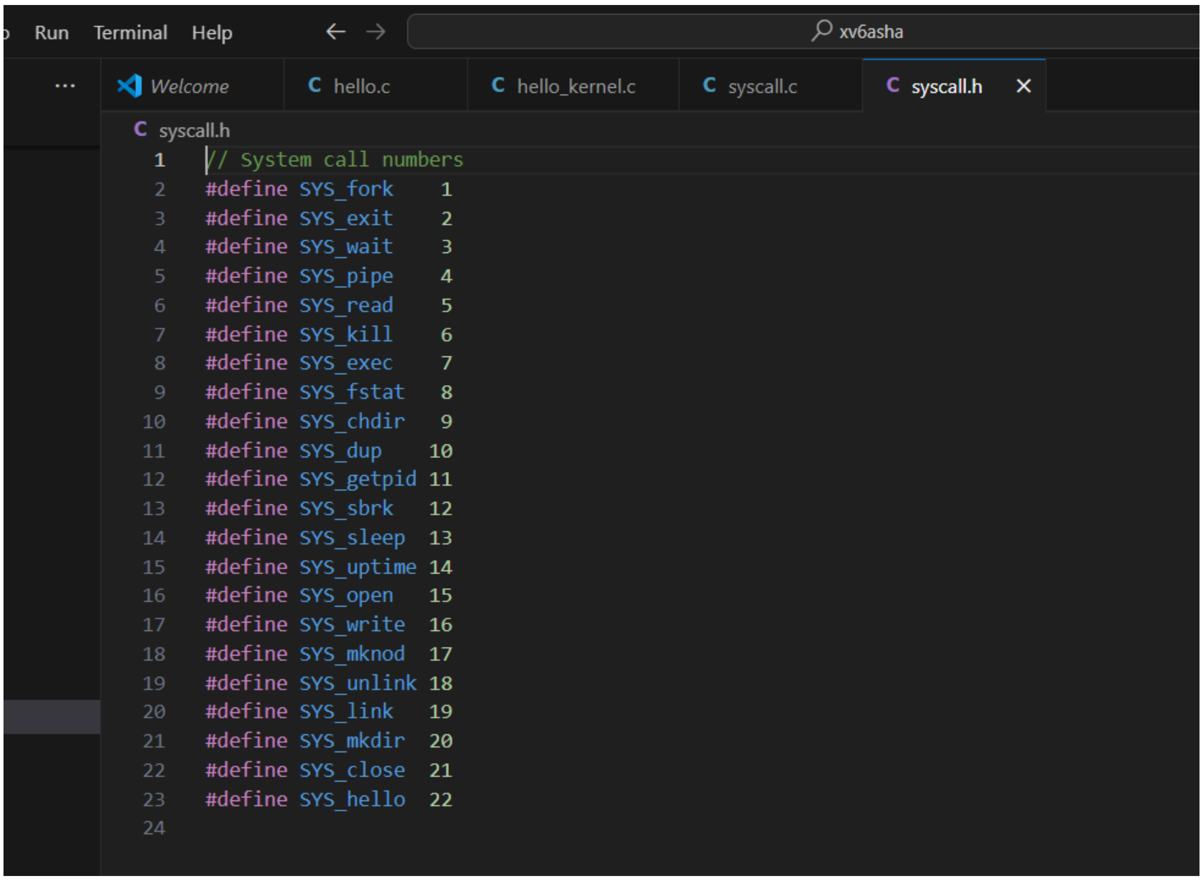
Below are the screenshots of code additions done in certain files for creating the hello system call.



```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(hello)

```

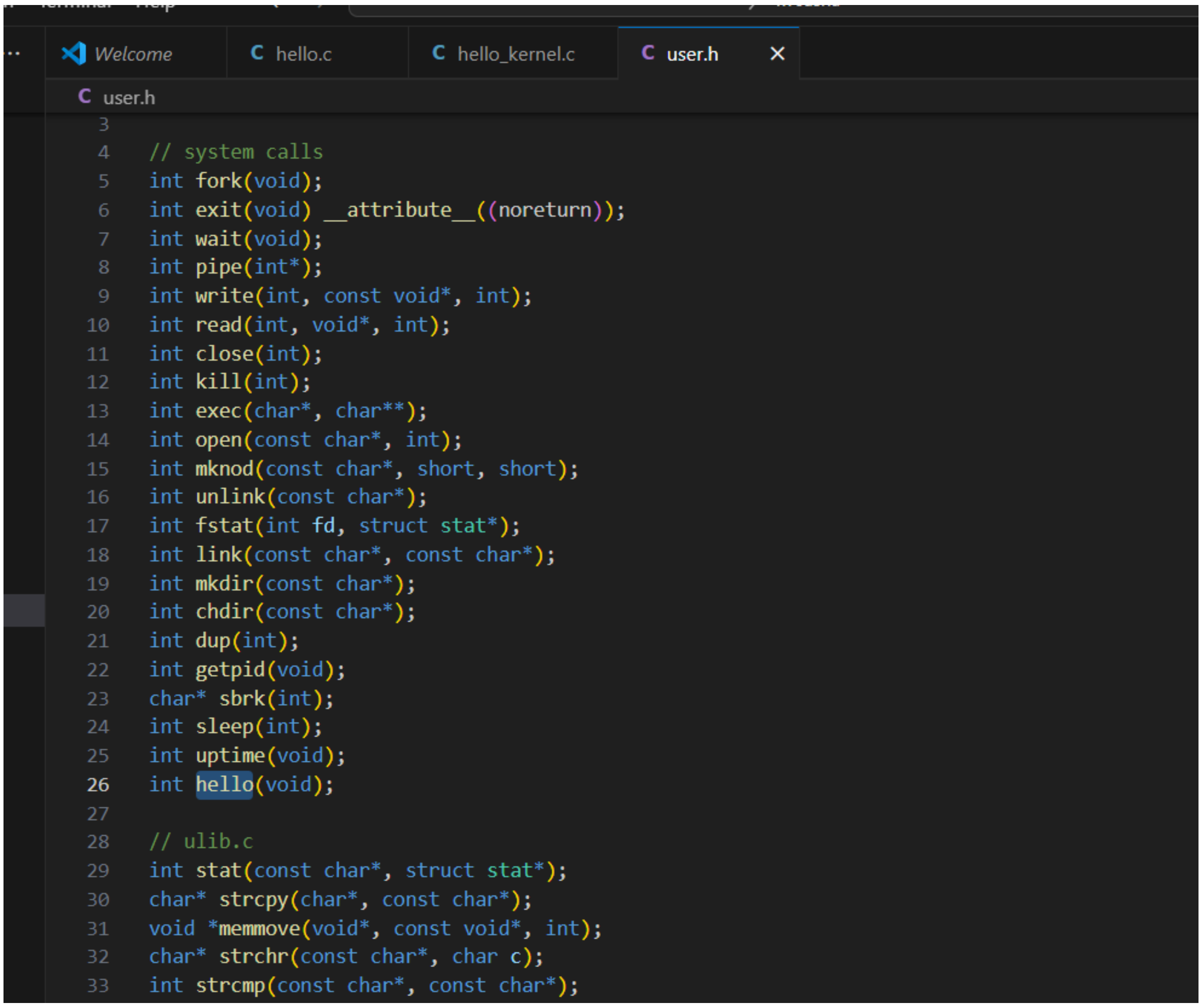


```c
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_hello(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
[SYS_hello]   sys_hello,
};
```



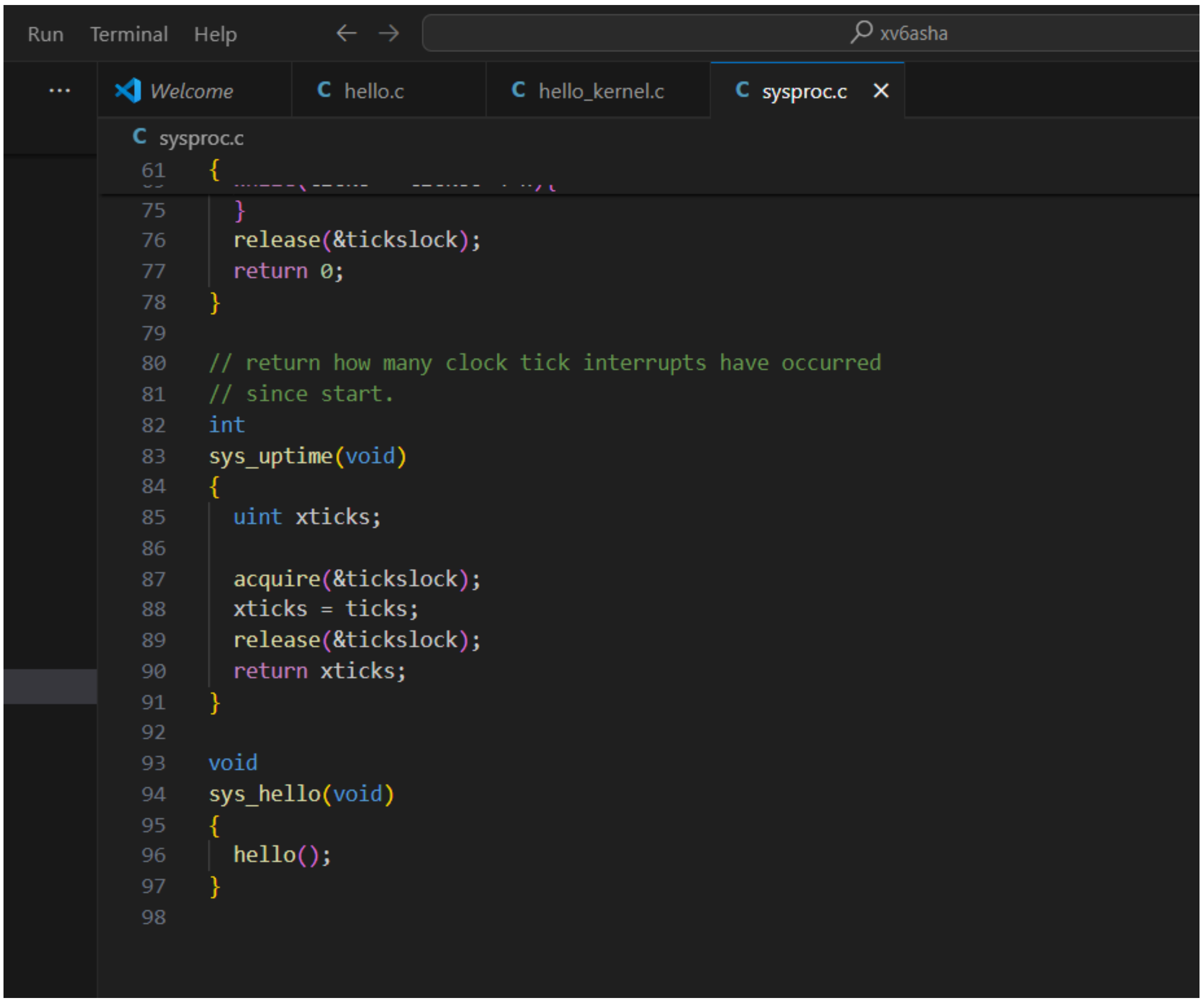
```c
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_hello  22
```

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



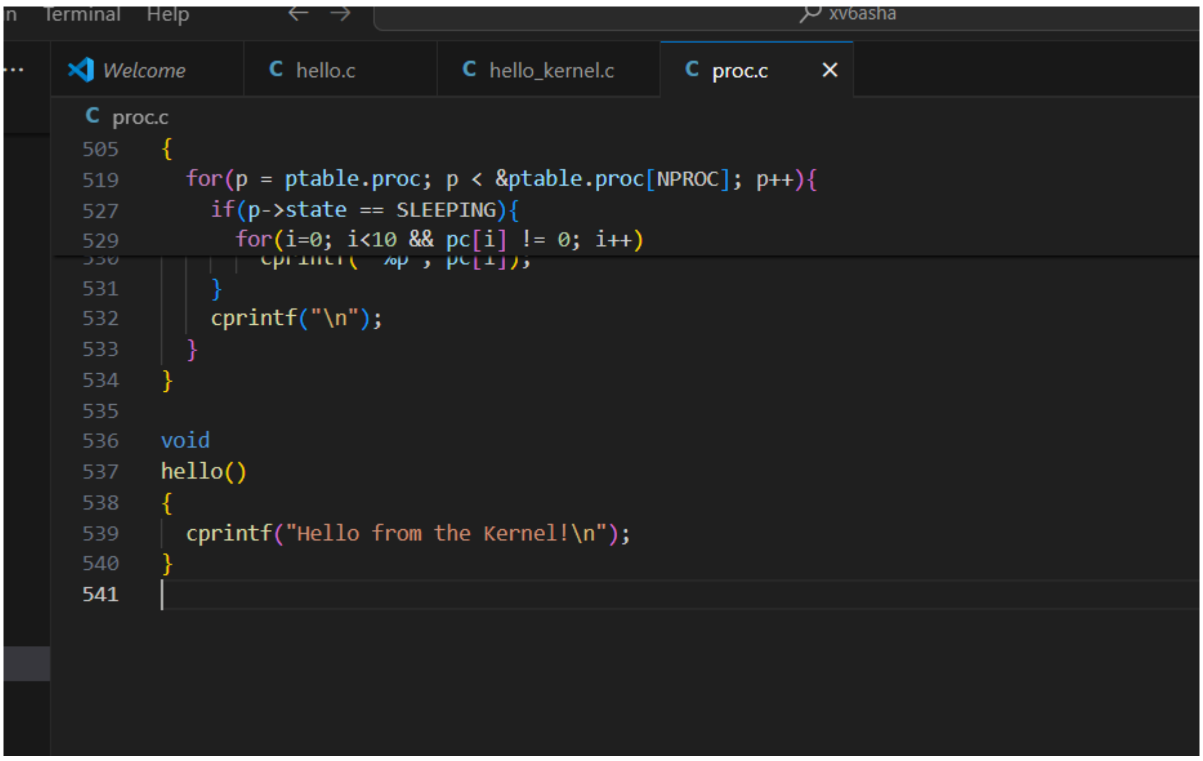
```c

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int hello(void);

// ulib.c
int stat(const char*, struct stat*);
char* strcpy(char*, const char*);
void *memmove(void*, const void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
```

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



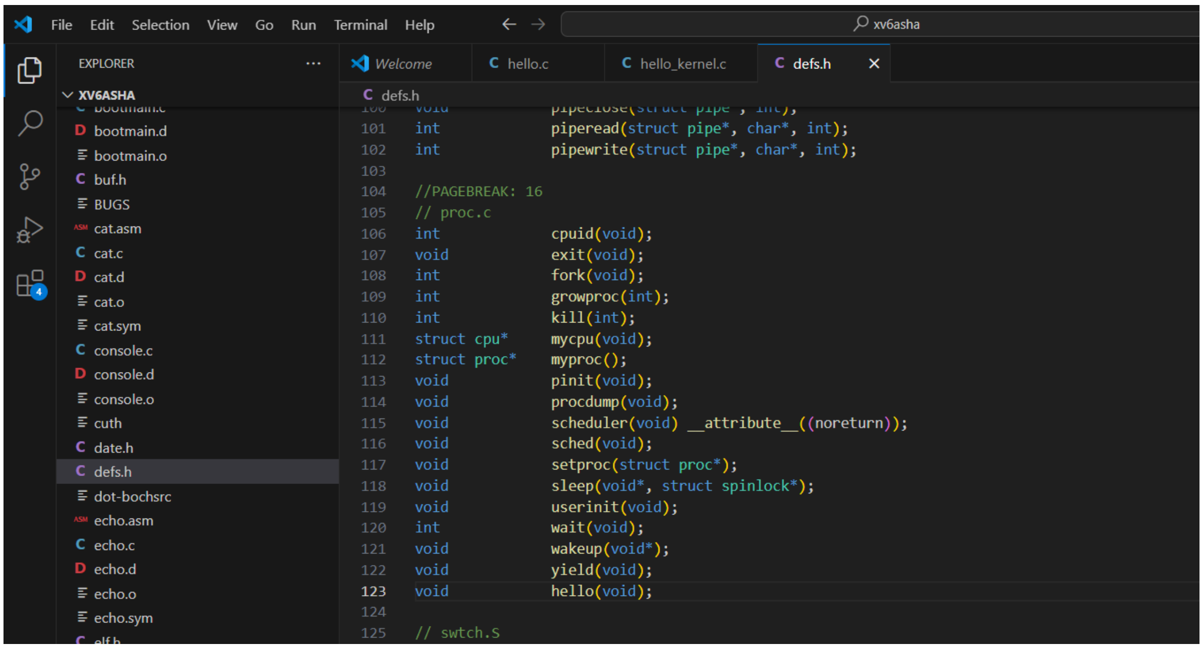
```
{
  }
  release(&tickslock);
  return 0;
}

// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
  uint xticks;

  acquire(&tickslock);
  xticks = ticks;
  release(&tickslock);
  return xticks;
}

void
sys_hello(void)
{
  hello();
}
```

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



```
{
  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    if(p->state == SLEEPING){
      for(i=0; i<10 && pc[i] != 0; i++)
    }
    cprintf("\n");
  }
}

void
hello()
{
  cprintf("Hello from the Kernel!\n");
}
```

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.



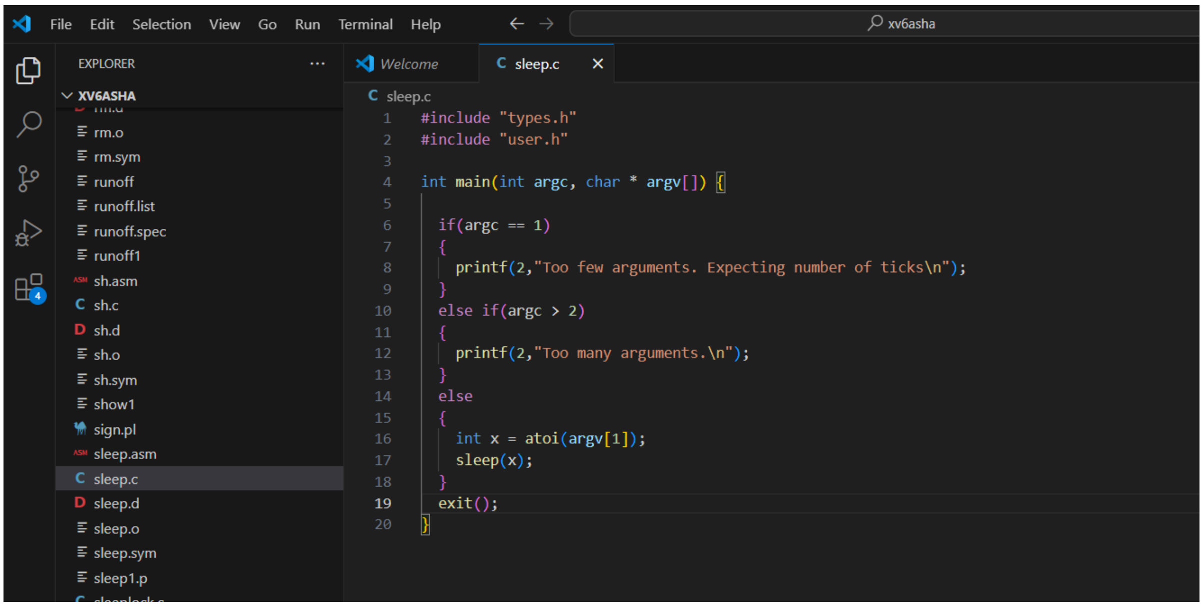
```
int             piperead(struct pipe*, char*, int);
int             pipewrite(struct pipe*, char*, int);

//PAGEBREAK: 16
// proc.c
int             cpuid(void);
void            exit(void);
int             fork(void);
int             growproc(int);
int             kill(int);
struct cpu*     mycpu(void);
struct proc*    myproc();
void            pinit(void);
void            procdump(void);
void            scheduler(void) __attribute__((noreturn));
void            sched(void);
void            setproc(struct proc*);
void            sleep(void*, struct spinlock*);
void            userinit(void);
int             wait(void);
void            wakeup(void*);
void            yield(void);
void            hello(void);

// swtch.S
```

Part-4 => sleep.c

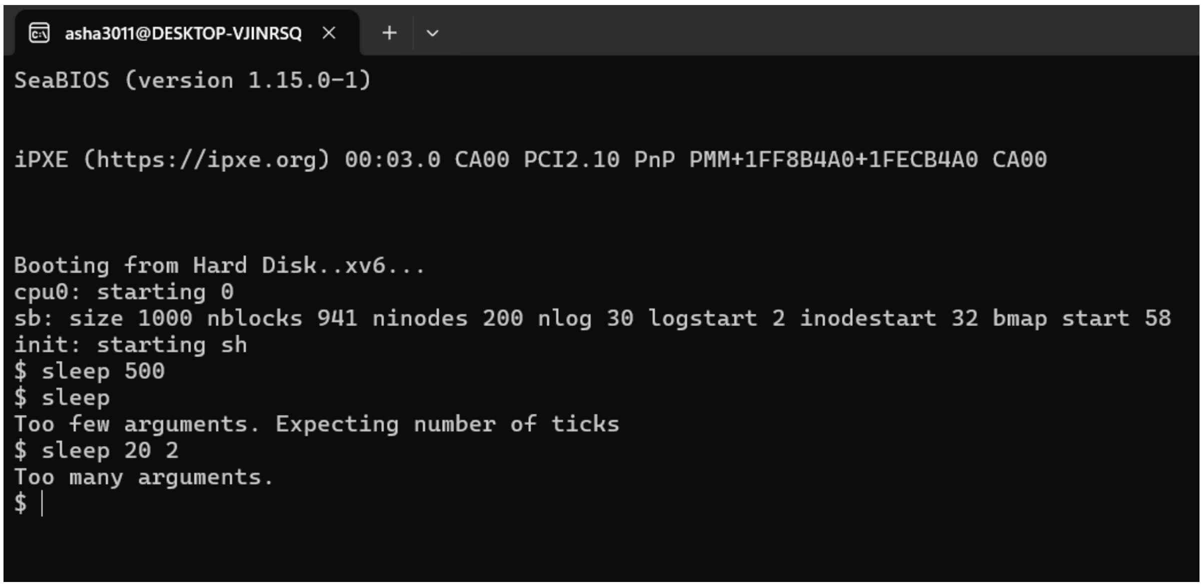
Read ticks from command line arguments.
if ticks are not entered by the user or extra unnecessary arguments are given, an appropriate error message is shown.

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

note: 100 ticks = 1 second



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

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

  if(argc == 1)
  {
    printf(2,"Too few arguments. Expecting number of ticks\n");
  }
  else if(argc > 2)
  {
    printf(2,"Too many arguments.\n");
  }
  else
  {
    int x = atoi(argv[1]);
    sleep(x);
  }
  exit();
}
```



```
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.
$ 
```