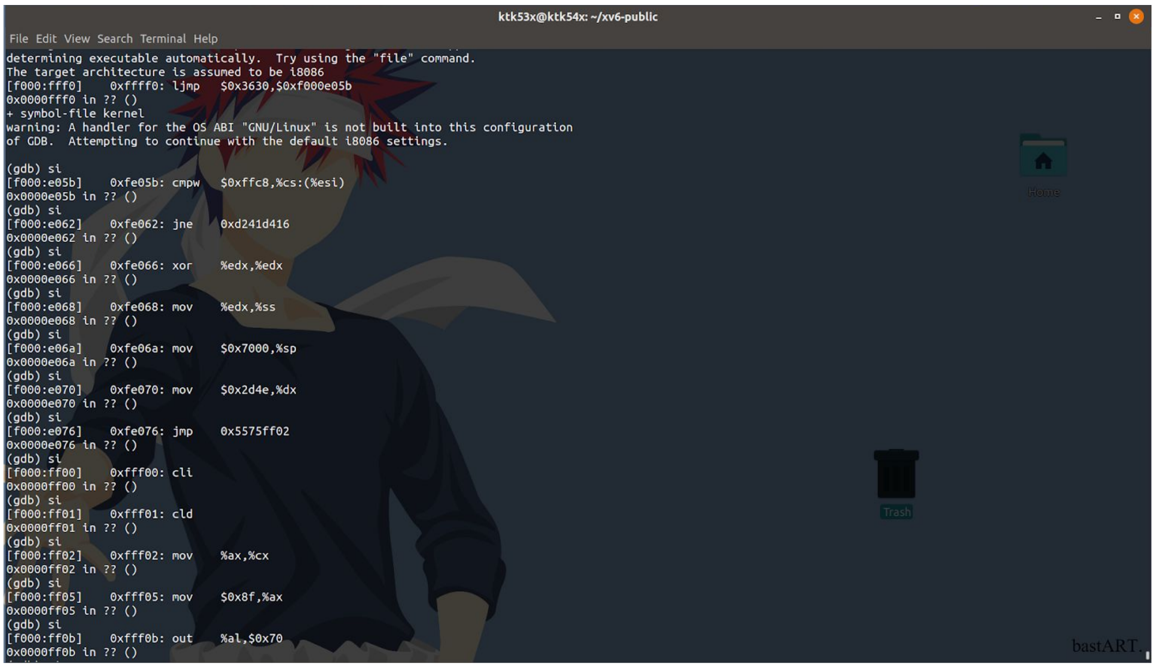
ASSIGNMENT-1

G-11

Kartikeya Saxena 180101034
Kushal Sangwan 180101096
Milind Prabhu 180101091
Aditya Patil 180101004

Ans. 1- The modified code is present in the file **ex1.c**

Ans. 2-


ktk53x@ktk54x: ~/xv6-public
File Edit View Search Terminal Help

```
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    0xd241d416
0x0000e062 in ?? ()
(gdb) si
[f000:e066]    0xfe066: xor    %edx,%edx
0x0000e066 in ?? ()
(gdb) si
[f000:e068]    0xfe068: mov    %edx,%ss
0x0000e068 in ?? ()
(gdb) si
[f000:e06a]    0xfe06a: mov    $0x7000,%sp
0x0000e06a in ?? ()
(gdb) si
[f000:e070]    0xfe070: mov    $0x2d4e,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076]    0xfe076: jmp    0x5575ff02
0x0000e076 in ?? ()
(gdb) si
[f000:ff00]    0xfff00: cli
0x0000ff00 in ?? ()
(gdb) si
[f000:ff01]    0xfff01: cld
0x0000ff01 in ?? ()
(gdb) si
[f000:ff02]    0xfff02: mov    %ax,%cx
0x0000ff02 in ?? ()
(gdb) si
[f000:ff05]    0xfff05: mov    $0x8f,%ax
0x0000ff05 in ?? ()
(gdb) si
[f000:ff0b]    0xfff0b: out    %al,$0x70
0x0000ff0b in ?? ()
```

The BIOS when loads jumps backwards from 0xf000:fff0 to 0xf000:0xe05b. It then set up an interrupt descriptor table and initializes various devices such as
the VGA display.

Ans. 3-
Readsect exact assembly instructions:

```
=> 0x7c90:      push   %ebp
0x00007c90 in ?? ()
(gdb)
=> 0x7c91:      mov    %esp,%ebp
0x00007c91 in ?? ()
(gdb)
=> 0x7c93:      push   %edi
0x00007c93 in ?? ()
(gdb)
=> 0x7c94:      push   %ebx
0x00007c94 in ?? ()
```

```
(gdb)
=> 0x7c95:	mov    0xc(%ebp),%ebx
0x00007c95 in ?? ()
(gdb)
=> 0x7c98:	call   0x7c7e
0x00007c98 in ?? ()
```

Begin and end of the for loop that reads the remaining sectors of the kernel from the disk:

Begin

```
=> 0x7d83:	cmp    %esi,%ebx
0x00007d83 in ?? ()
```

End

```
=> 0x7d94:	jbe    0x7d87
0x00007d94 in ?? ()
```

After these for loop it executes entry() and transfers control to the kernel

a)

```
ljmp    $0xb866, $0x87c31
ljmp    $(SEG_KCODE<<3), $start32
```

The code segment descriptor has a flag set that indicates that the code
should run in 32bit mode.
Once it has loaded the GDT register, the boot loader enables protected mode by
setting the 1 bit (CR0_PE) in register %cr0 . Enabling protected mode does
not immediately change how the processor translates logical to physical addresses; it is
only when one loads a new value into a segment register that the processor reads the
GDT and changes its internal segmentation settings. One cannot directly modify %cs,
so instead the code executes an ljmp (far jump) instruction , which allows a code
segment selector to be specified.

The boot loader then switches the processor from real mode to
32-bit protected mode, because it is only in this mode that software can access all the
memory above 1MB in the processor's physical address space

b)

Last instruction of the boot loader executed:

```
7d87:   ff 15 18 00 01 00        call   *0x10018
entry = (void(*)(void))(elf->entry);
entry();
```

First instruction of the kernel it just loaded:

```
0x10000c:      mov    %cr4,%eax
```

c)
The information about the **number of program segments** is stored in the phnum attribute of the elf binary header . Further the **number of sectors in a particular segment** is calculated by ph->filesz(count) / SECTSIZE(=512) with appropriate offset settings. The elf header has fixed length, it is the variable-length program segment which needs to be calculated.

Ans. 4-
Line 1:
a, b and c are pointers to integer variables. a is allocated 16 bytes of memory on the stack. b is allocated 16 bytes of information on the heap. The pointer c is declared but is uninitialized. So it stores some junk pointer.

Line 2:
The for loop on line 15 changes the value of integers in array a to 100, 101, 102, 103. The line "c=a;" makes the pointer c point to the same integer as a. Therefore when c[0] is assigned 200 it changes the first element in array a because c is just another name for array a.

Line 3:
c[1]=300; - Changes a[1] to 300 as c is an alias for a.
*(c+2)=301; - Another way of saying c[2]=301. a[2] is set to 301.
3[c]=302; -Another way of saying c[3]=302. a[3] is set to 302

Line 4:
c=c+1; - This makes c point to the location of a[1]
*c=400; - This changes a[1] to 400

Line 5:
c = (int *) ((char *) c + 1); - The hexadecimal value of the address stored in pointer c increases by only 1 since we typecast it to a character pointer before incrementing it. This is because the size of a character type data in C is 1 byte. The pointer c is then typecast back into an integer type. At the end of this c points to a segment of 4 bytes beginning from the second byte of a[1] and ending at the first byte of a[2].

The contents of a[2] and a[3] at this point look as follows.

```
a[2]=400                              a[3]=301
10010000 10000000 00000000 00000000   00101101 10000000 00000000 00000000
```

After *c=500 it changes to:

```
a[2]=128144                           a[3]=256
10010000 11110100 00000001 00000000   00000000 10000000 00000000 00000000
```

Line 6:
b=(int*)a+1; - Increases hexadecimal address value by 4.
c = (int *) ((char *) a + 1); - Increases hexadecimal address value by only 1

Ans. 5-

1- Changed link address from 0x7c00 to 0x7c10.

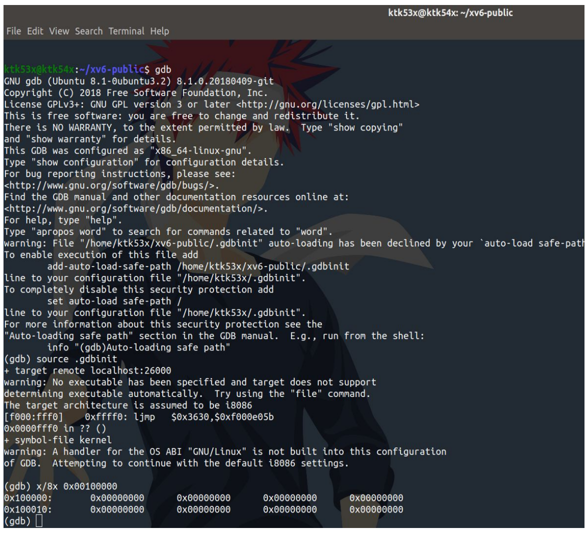
2- The make clean was then executed and the makefile was executed again.

3- The code stops working after the line at 0x7c2c and jumps to 0xfe05b. It gets into an infinite loop after execution of a few instructions.

```
(gdb)
[   0:7c2c] => 0x7c2c:          ljmp   $0xb866,$0x87c41
0x00007c2c in ?? ()
(gdb)
[f000:e05b]    0xfe05b:         cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
```

Ans. 6-

They are different because during the running of the bootloader, kernel instructions get loaded at 0x100000. So before its execution , zeros are stored in those words , while after it, initial instructions of the kernel are there.

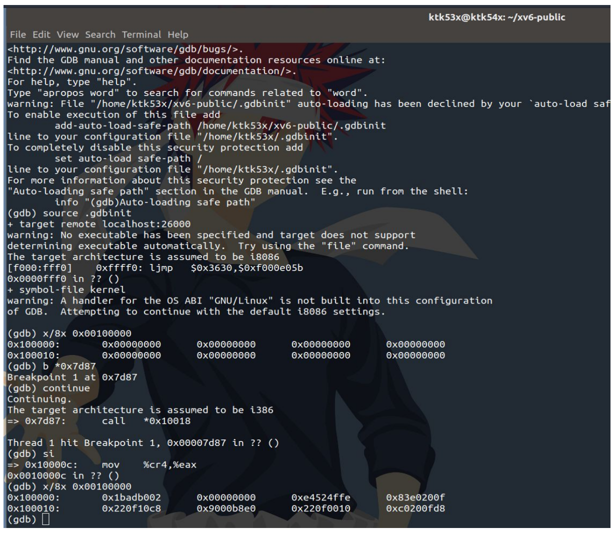


```
ktk53x@ktk54x: ~/xv6-public
File  Edit  View  Search  Terminal  Help

ktk53x@ktk54x:~/xv6-public$ gdb
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: File "/home/ktk53x/xv6-public/.gdbinit" auto-loading has been declined by your `auto-load safe-path
To enable execution of this file add
        add-auto-load-safe-path /home/ktk53x/xv6-public/.gdbinit
line to your configuration file "/home/ktk53x/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ktk53x/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) x/8x 0x00100000
0x100000:       0x00000000      0x00000000      0x00000000      0x00000000
0x100010:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb) 
```

The point BIOS enters bootloader



```
ktk53x@ktk54x: ~/xv6-public
File  Edit  View  Search  Terminal  Help
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: File "/home/ktk53x/xv6-public/.gdbinit" auto-loading has been declined by your `auto-load saf
To enable execution of this file add
        add-auto-load-safe-path /home/ktk53x/xv6-public/.gdbinit
line to your configuration file "/home/ktk53x/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/ktk53x/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) x/8x 0x00100000
0x100000:       0x00000000      0x00000000      0x00000000      0x00000000
0x100010:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb) b *0x7d87
Breakpoint 1 at 0x7d87
(gdb) continue
Continuing.
The target architecture is assumed to be i386
=> 0x7d87:      call   *0x10018

Thread 1 hit Breakpoint 1, 0x00007d87 in ?? ()
(gdb) si
=> 0x10000c:    mov    %cr4,%eax
0x0010000c in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x1badb002      0x00000000      0xe4524ffe      0x83e0200f
0x100010:       0x220f10c8      0x9000b8e0      0x220f0010      0xc0200fd8
(gdb) 
```

The point bootloader enters kernel

Ans. 7-

The 5 files which are edited to add the system call are :

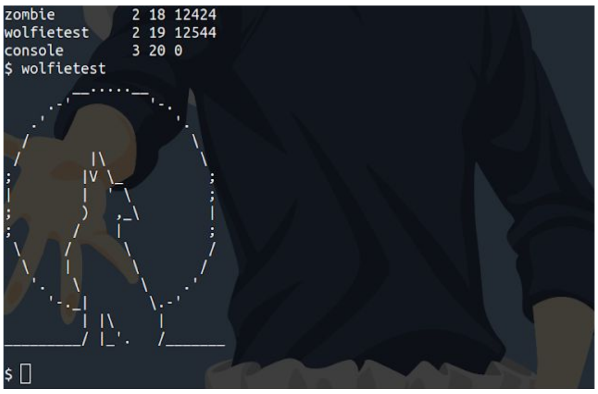
Syscall.h: This file assigns a number to every system call in xv-6 system. Before adding wolfie there were 21 system calls , hence wolfie was assigned number 22.
Line:#define SYS_wolfie 22

Syscall.c: It contains an array of function pointers(syscalls[]) which uses index defined in systemcall.h to point to the respective system call function stored at a different memory location. We also put a function prototype here(but not implementation).
Line1:Extern int sys_wolfie(void);
Line2:[SYS_wolfie] sys_wolfie,

Sysproc.c: This is where the implementation of our system call is written.

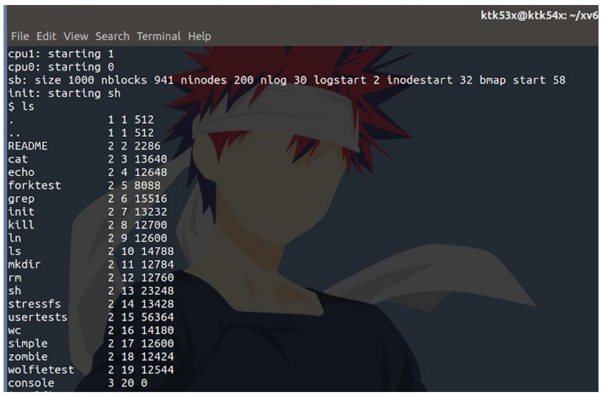
user.h and usys.S: They act as an interface for our system to access the system call. The function prototype is added in user.h(included as header file in our program) while instruction to treat it as a system call is included in usys.S
usys.S: SYSCALL(wolfie)
user.h: int wolfie(void*,int);

Ans. 8-



```
zombie         2 18 12424
wolfietest     2 19 12544
console        3 20 0
$ wolfietest
        __.....__
     .-'         '-.
   .'                '.
  /                    \
 /         |\           \
;         |V \_          ;
|         |  ' \         ;
;         )   ,_\        |
;        /    |          ;
 \      /      \        /
  \     |       \      /
   '.    \       \   .'
     '-._|        \.-'
         | |\      |
_________/ |_'.    /_______

$ 
```



```
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 13640
echo           2 4 12648
forktest       2 5 8088
grep           2 6 15516
init           2 7 13232
kill           2 8 12700
ln             2 9 12600
ls             2 10 14788
mkdir          2 11 12784
rm             2 12 12760
sh             2 13 23248
stressfs       2 14 13428
usertests      2 15 56364
wc             2 16 14180
simple         2 17 12600
zombie         2 18 12424
wolfietest     2 19 12544
console        3 20 0
```

The wolf image output   ls command output

Makefile:
The program(wolfietest) is included in the UPROG list which are written to fs.img as well as in EXTRA list which is included when making dist.

wolfietest.c:
The user level application wolfietest.c contains the instructions to allocate space to the buffer and then call the system call to write the ascii wolfie string in the buffer. In the end, the image is printed to the console