Operating Systems Lab: Assignment 1

Members: Kousik Rajesh, Drishti Chouhan, Eklavya Jain, Mridul Garg

Q.1 . `asm("inc %0": "+r"(x);`

```
kousik@sloth:~/xv6-public$ gcc ex1.c
kousik@sloth:~/xv6-public$ ./a.out
Hello x = 1
Hello x = 2 after increment
OK
```

The **inc** stands for incrementing the value of the provided argument. **"+r"** stands for the fact that **x** is the input as well as the output. And **%0** denotes the first and the only argument, **x**.

Q.2

```
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si 2
[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    $0x7c4,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076]    0xfe076: jmp    0x5576cf26
0x0000e076 in ?? ()
(gdb) si
[f000:cf24]    0xfcf24: cli
0x0000cf24 in ?? ()
(gdb) si
[f000:cf25]    0xfcf25: cld
0x0000cf25 in ?? ()
(gdb) si
[f000:cf26]    0xfcf26: mov    %ax,%cx
0x0000cf26 in ?? ()
(gdb) si
[f000:cf29]    0xfcf29: mov    $0x8f,%ax
0x0000cf29 in ?? ()
(gdb) si
[f000:cf2f]    0xfcf2f: out    %al,$0x70
0x0000cf2f in ?? ()
(gdb) si
[f000:cf31]    0xfcf31: in     $0x71,%al
0x0000cf31 in ?? ()
(gdb) si
[f000:cf33]    0xfcf33: in     $0x92,%al
0x0000cf33 in ?? ()
```

```
[f000:cf35]    0xfcf35: or     $0x2,%al
0x0000cf35 in ?? ()
(gdb) si
[f000:cf37]    0xfcf37: out    %al,$0x92
0x0000cf37 in ?? ()
(gdb) si
[f000:cf39]    0xfcf39: mov    %cx,%ax
0x0000cf39 in ?? ()
(gdb) si
[f000:cf3c]    0xfcf3c: lidtl  %cs:(%esi)
0x0000cf3c in ?? ()
(gdb) si
[f000:cf42]    0xfcf42: lgdtl  %cs:(%esi)
0x0000cf42 in ?? ()
(gdb) si
[f000:cf48]    0xfcf48: mov    %cr0,%ecx
0x0000cf48 in ?? ()
(gdb) si
[f000:cf4b]    0xfcf4b: and    $0xffff,%cx
0x0000cf4b in ?? ()
(gdb) si
[f000:cf52]    0xfcf52: or     $0x1,%cx
0x0000cf52 in ?? ()
(gdb) si
[f000:cf56]    0xfcf56: mov    %ecx,%cr0
0x0000cf56 in ?? ()
(gdb) si
[f000:cf59]    0xfcf59: ljmpw  $0xf,$0xcf61
0x0000cf59 in ?? ()
(gdb) si
The target architecture is assumed to be i386
=> 0xfcf61:     mov    $0x10,%ecx
0x000fcf61 in ?? ()
(gdb) si
=> 0xfcf66:     mov    %ecx,%ds
0x000fcf66 in ?? ()
(gdb) si
=> 0xfcf68:     mov    %ecx,%es
0x000fcf68 in ?? ()
(gdb) si
=> 0xfcf6a:     mov    %ecx,%ss
0x000fcf6a in ?? ()
```

- The BIOS starts with the assumption that the architecture is i8086(1MB address space). We first switch the mode to i386 by making a jump to a previous address
  [f000:fff0]   0xffff0:   ljmp $0x3630:0xf000e05b
- The BIOS then zeros the **edx** and **ss** registers and sets **sp** register and configures the hardware devices using the **out** and **in** command
- The **cli instruction** disables interrupts so that the following instructions will be executed without interruption. It does so by clearing the interrupt flag
- The **cld instruction** clears the direction flag and sets it to zero indicating that memory grows from low to high
- Finally, it loads the boot sector at the address ***0x7c00***

Q3. a)

```
# Switch from real to protected mode
 lgdt    gdtdesc
 movl    %cr0, %eax
 orl     $CR0_PE, %eax
 movl    %eax, %cr0
# Following long jmp completes the transition to 32-bit protected mode
 ljmp    $(SEG_KCODE<<3), $start32#
```

b) Last instruction that Bootloader executes :

In **bootmain.c** it is where the **entry** function is called to enter the kernel

```
entry = (void(*)(void))(elf->entry);
entry();
```

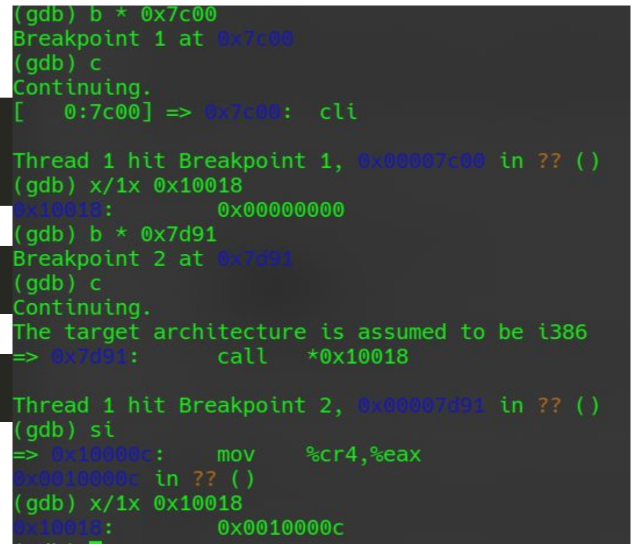
In **bootblock.asm** it is **call *0x10018** which calls the entry function

```
 7d87:  ff 15 18 00 01 00   call *0x10018
```

First instruction of the kernel :

```
movl    %cr4, %eax
```

(The first instruction is present at ***0x0010000c)***



c) The code segment which loads the kernel sectors from disk

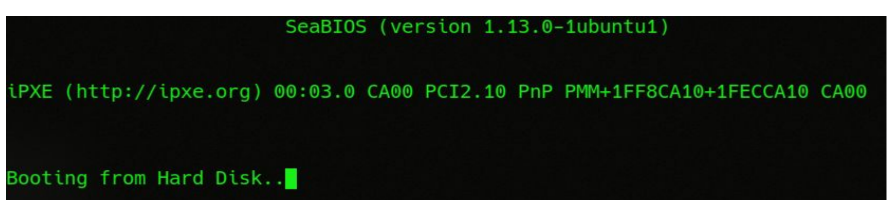
```
ph = (struct proghdr*)((uchar*)elf + elf->phoff);
eph = ph + elf->phnum;
for(; ph < eph; ph++){
  pa = (uchar*)ph->paddr;
  readseg(pa, ph->filesz, ph->off);
}
```

- ***ph*** Points to the start of the program header table found by adding an offset to **elf**
- ***elf->phnum*** contains the number of entries in the program header table.
- ***eph*** is a pointer to the entry just after the last entry in program header table
- We iterate through each sector and read them one by one by incrementing the pointer ***ph*** till we reach ***eph*** at which point we exit the for loop

Q.5 On changing the link address in Makefile from **0x7C00 to 0x7D00** the cpde expected to be at 0x7C00 is not present

The first instruction to go wrong is

```
ljmp    $0x3630,$0xf000e05b
```



The bootloader is stuck at booting from hard disk. When inspected using **GDB** I found that the **IP(instruction pointer)** begins to loop between address **6dc1** and **eeee** it overflows at **eeee** and then goes back to **6dc1** in an infinite loop.

Q.6

When BIOS enters Boot loader
All are zero
Doing an **objdump** for kernel we can see that it's **Load Memory Address(LMA) is at 0x00100000**
And since the kernel hasn't been loaded yet, these 8 words in memory are zero.

```
kousik@sloth:~/xv6-public$ objdump -h kernel

kernel:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000713a  80100000  00100000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
```

When Boot loader enters Kernel
The bootloader has now finished reading all sectors from disk using the **readseg() function** and the address ***0x00100000*** falls in the **.text** section of the kernel and hence when we now examine the memory near this address we find the kernel instructions which were newly loaded by the bootloader

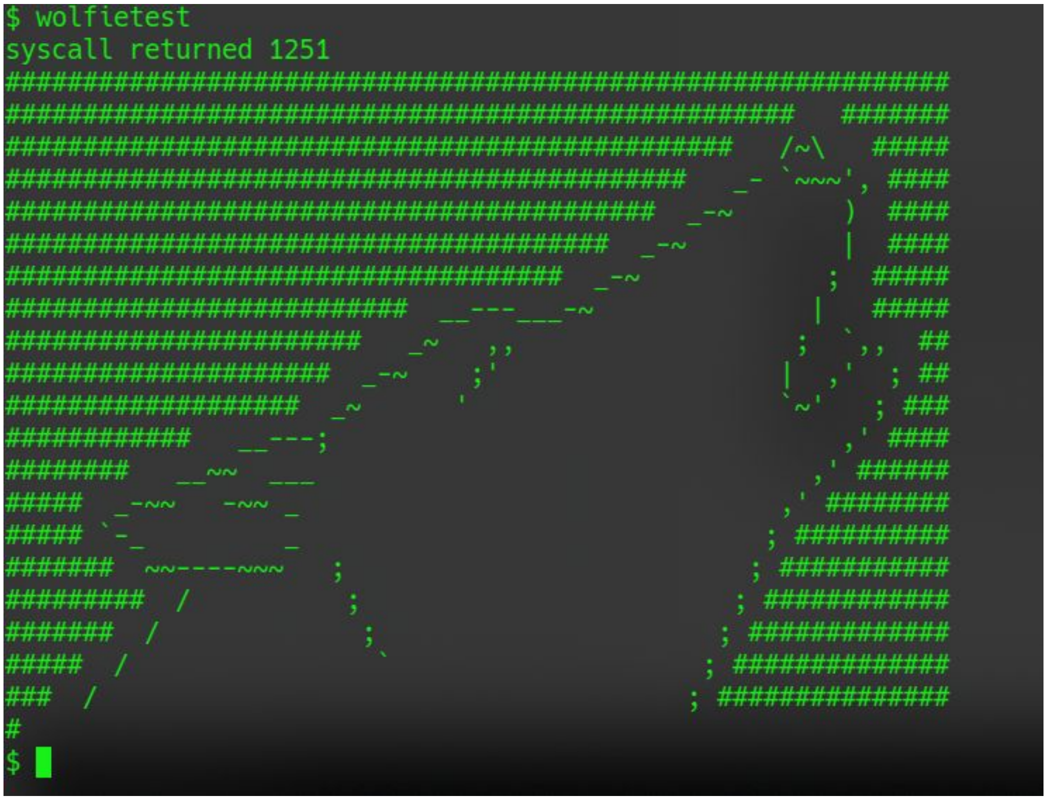
```
(gdb) b * 0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x00000000      0x00000000      0x00000000      0x00000000
0x100010:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb) b * 0x7d91
Breakpoint 2 at 0x7d91
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d91:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d91 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x1badb002      0x00000000      0xe4524ffe      0x83e0200f
0x100010:       0x220f10c8      0x9000b8e0      0x220f0010      0xc0200fd8
(gdb) x/8i 0x00100000
   0x100000:    add    0x1bad(%eax),%dh
   0x100006:    add    %al,(%eax)
   0x100008:    decb   0x52(%edi)
   0x10000b:    in     $0xf,%al
   0x10000d:    and    %ah,%al
   0x10000f:    or     $0x10,%eax
   0x100012:    mov    %eax,%cr4
   0x100015:    mov    $0x109000,%eax
(gdb)
```

Q.7 Files changed:

- ***sysfile.c*** : Add the system call which copies wolfie into the provided buffer and returns an integer
- ***user.h*** : Add a function declaration for the system call in the correct format
- ***usys.S*** : Add a SYSCALL() for wolfie
- ***syscall.h*** : Define a number for the wolfie syscall
- ***syscall.c*** : Add our custom defined syscall to the list of syscalls present in this file

Q.8 Files changed:

- ***wolfietest.c*** : This will be our user level application which uses the syscall and prints wolfie on the terminal
- ***Makefile***: Inform the Makefile compile our user level application wolfietest.c


$ wolfietest
syscall returned 1251