A

PROJECT REPORT

ON

WRITING A SIMPLE OPERATING SYSTEM KERNEL

In

OPERATING SYSTEM CSE 2005

VELLORE INSTITUTE OF TECHNOLOGY

VELLORE, TAMILNADU

SUBMITTED TO:

Prof. Saleem Durai M A

SCOPE

SUBMITTED BY:

Achyut Tripathi

Reg no: 17BCE0954

Slot: F1

Mobile no: 9651807367

Introduction:

We've all used an operating system (OS) before (e.g. Windows XP, Linux, etc.), and perhaps we have even written some programs to run on one; but what is an OS actually there for, how much of what I see when I use a computer is done by hardware and how much is done by software and how does the computer actually work?

It seems that, today, we take a lot for granted about how these wonderful machines actually work underneath all those layers of software that commonly come bundled with them and which are required for their day-to-day usefulness.

The first part of operating system is the Bootloader. Bootloader is a piece of program that runs before any operating system is running. It is used to boot other operating systems, usually each operating system has a set of bootloaders specific for it.

Bootloaders usually contain several ways to boot the OS kernel and also contain commands for debugging and/or modifying the kernel environment.

We will create 2 stage OS. First is to just display message on the screen with colors and second is to take input from user.

The bootloaders are generally written in 16-bit assembly (also called Real mode), then the bits can be extended to 32-bit (Protected mode).

So the bootloaders must be written in 16-bit assembly.

Code:

```
[bits 16] ; for 16 bit mode
[org 0x7c00] ; organize from BIOS addres

start:
        xor ax,ax           ; set ax register to 0
        mov ds,ax           ; set data segment(ds) to 0
        mov es,ax           ; set extra segment(es) to 0
        mov bx,0x8000

        mov ax,0x13         ;clears the screen
        int 0x10            ;call BIOS video interrupt

        ;set cursor to specific position on screen
        mov ah,0x02         ; set value for change to cursor position
        mov bh,0x00         ; page
        mov dh,0x06         ; row number
        mov dl,0x09         ; col number
        int 0x10

        mov si, start_os_intro
        call _print_DiffColor_String

        ;set cursor to specific position on screen
        mov ah,0x02
        mov bh,0x00
        mov dh,0x10
        mov dl,0x06
        int 0x10
```

```
mov si,press_key
call _print_GreenColor_String

mov ax,0x00         ; get keyboard input
int 0x16            ; interrupt for hold & read input

; load second sector into memory
mov ah, 0x02        ; load second stage to memory
mov al, 1           ; numbers of sectors to read into memory
mov dl, 0x80        ; sector read from fixed/usb disk
mov ch, 0           ; cylinder number
mov dh, 0           ; head number
mov cl, 2                       ; sector number
mov bx, _OS_Stage_2             ; load into es:bx segment :offset of buffer
int 0x13                        ; disk I/O interrupt
jmp _OS_Stage_2                 ; jump to second stage

start_os_intro db 'Welcome to My OS!',0
press_key db '>>>> Press any key <<<<',0
login_username db 'Username : ',0
login_password db 'Password : ',0
display_text db '! Welcome to my Operating System !', 0
os_info db 10, 'My Operating System, 16-Bit, version=1.0.0',13,0
login_label db ' Login please...', 0
author db 'Created By:- ACHYUT TRIPATHI', 0
reg_no db 'Reg. No:- 17BCE0954', 0

; print string without color
print_string:
```

```
mov ah, 0x0E            ; value to tell interrupt handler that take value from al & print it
.repeat_next_char:
lodsb                   ; get character from string
cmp al, 0               ; cmp al with end of string
je .done_print          ; if char is zero, end of string
int 0x10                ; otherwise, print it
jmp .repeat_next_char   ; jmp to .repeat_next_char if not 0

.done_print:
ret                     ;return

; print string with different colors
_print_DiffColor_String:
mov bl,1                ;color value
mov ah, 0x0E

.repeat_next_char:
lodsb
cmp al, 0
je .done_print
add bl,6                ;increase color value by 6
int 0x10
jmp .repeat_next_char

.done_print:
ret

; print string with green color
_print_GreenColor_String:
mov bl,10
```

```
mov ah, 0x0E

.repeat_next_char:
lodsb
cmp al, 0
je .done_print
int 0x10
jmp .repeat_next_char

.done_print:
ret

; print string with white color
_print_WhiteColor_String:
mov bl,15
mov ah, 0x0E

.repeat_next_char:
lodsb
cmp al, 0
je .done_print
int 0x10
jmp .repeat_next_char

.done_print:
ret

; boot loader magic number
times (510 - ($ - $$)) db 0x00          ;set 512 bytes for boot sector which are necessary
dw 0xAA55                               ; boot signature 0xAA & 0x55
```

```
;//////////////////////////////////////////////////////////////////////////////////

_OS_Stage_2:
        mov al,2                                ; set font to normal mode
        mov ah,0                                ; clear the screen
        int 0x10                                ; call video interrupt
        mov cx,0                                ; initialize counter(cx) to get input

        ; print login_label on screen

        ; set cursor to specific position on screen
        mov ah,0x02
        mov bh,0x00
        mov dh,0x00
        mov dl,0x20
        int 0x10

        mov si,login_label                      ; point si to login_username
        call print_string                       ; display it on screen

        ; read username

        ;set cursor to specific position on screen
        mov ah,0x02
        mov bh,0x00
        mov dh,0x05
        mov dl,0x00
```

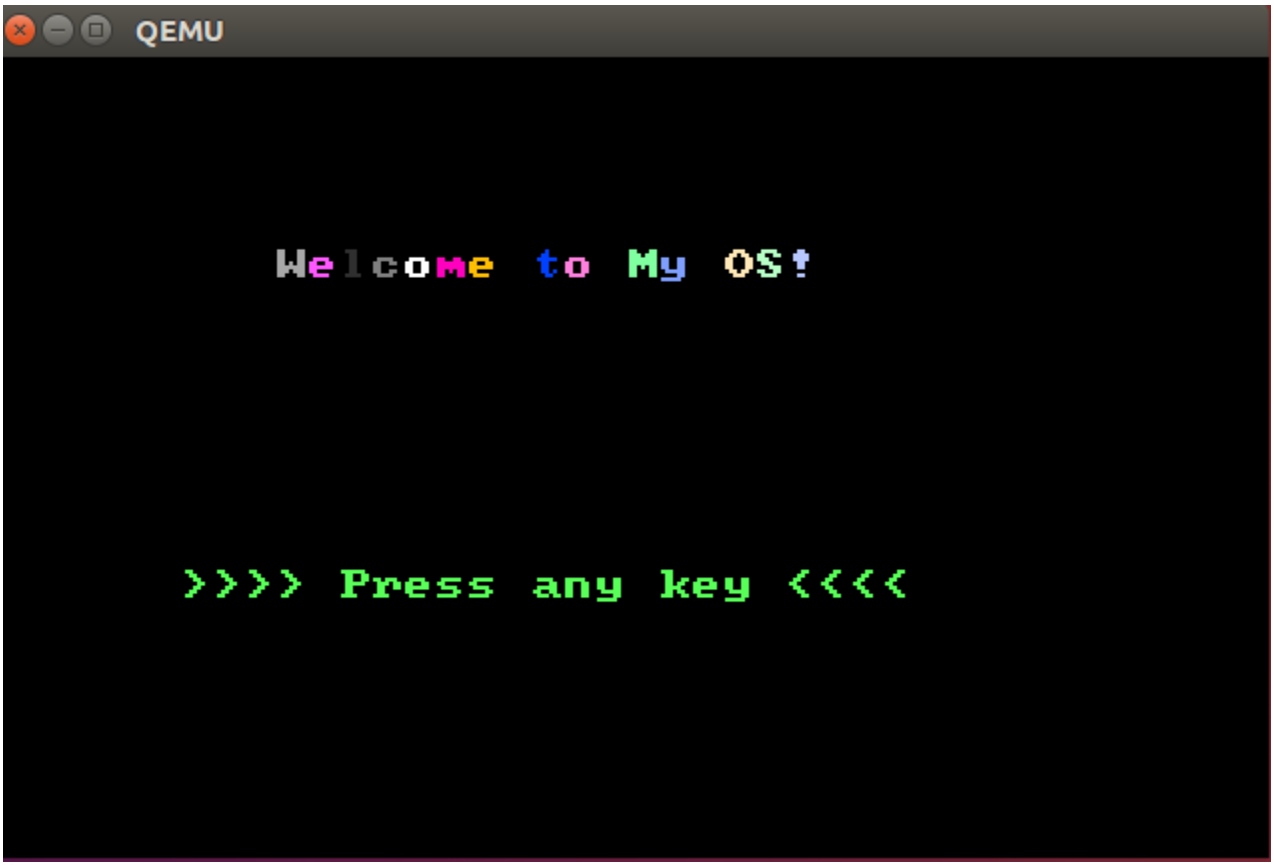
```
int 0x10
mov si,login_username                ; point si to login_username
call print_string                    ; display it on screen

_getUsernameinput:
mov ax,0x00                          ; get keyboard input
int 0x16                             ; hold for input
cmp ah, 0x1c                         ; Check whether enter is encountered or not
je .exitinput
mov ah,0x0E                          ;display input char
int 0x10
inc cx                               ; increase counter
cmp cx,5                             ; compare counter reached to 5
jbe _getUsernameinput                ; yes jump to _getUsernameinput
jmp .inputdone                       ; else jump to inputdone

.inputdone:
mov cx,0                             ; set counter to 0
jmp _getUsernameinput                ; jump to _getUsernameinput
ret                                  ; return

.exitinput:
hlt

;read password

;set x y position to text
mov ah,0x02
mov bh,0x00
mov dh,0x07
```

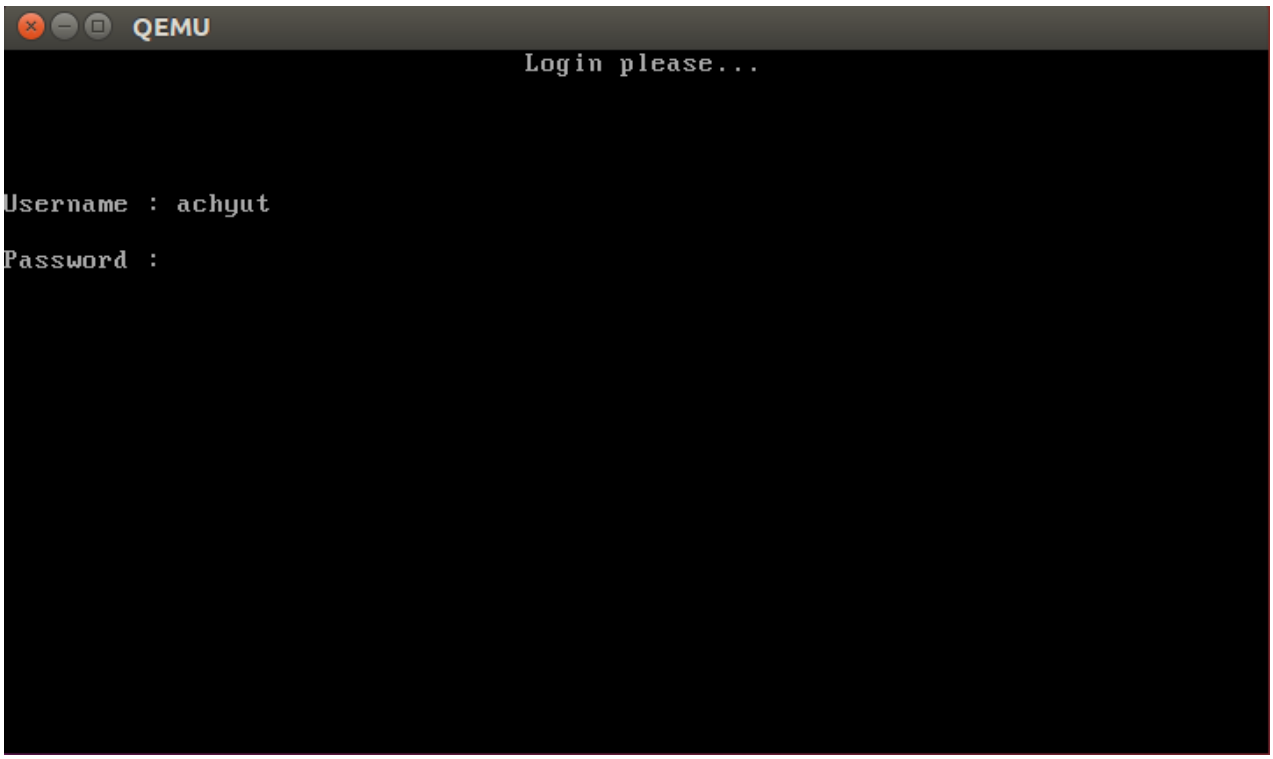
```
mov dl,0x00
int 0x10

mov si,login_password                    ; point si to login_username
call print_string                        ; display it on screen

_getPasswordinput:
mov ax,0x00
int 0x16
cmp ah, 0x1c
je .exitinput
inc cx
cmp cx,5
jbe _getPasswordinput
jmp .inputdone

.inputdone:
mov cx,0
jmp _getPasswordinput
ret

.exitinput:
hlt

mov al,2                                 ; set font to normal mode
mov ah,0                                 ; clear the screen
int 0x10                                 ; call video interrupt
mov cx,0                                 ; initialize counter(cx) to get input

;display display_text on screen
```

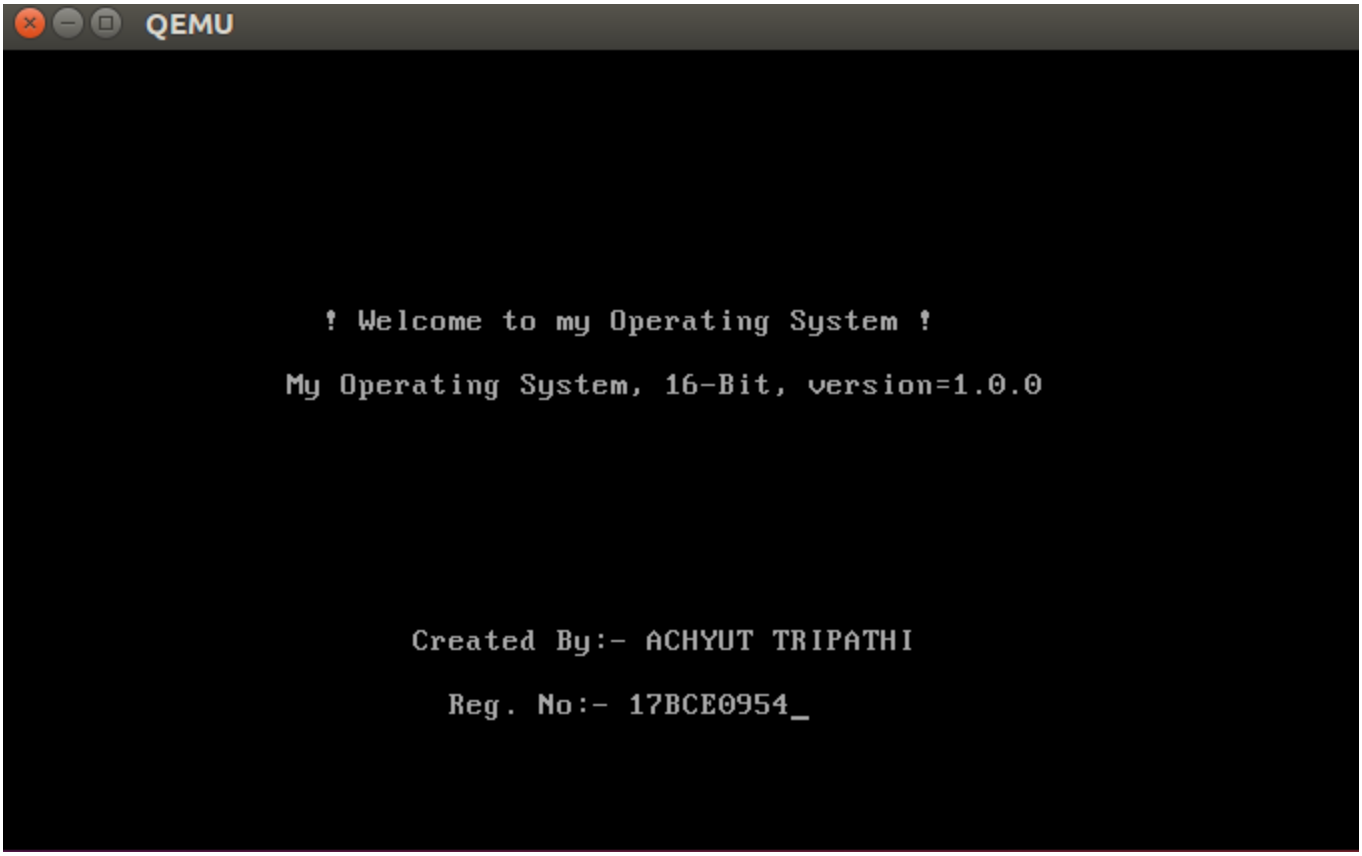
```
;set x y position to text
mov ah,0x02
mov bh,0x00
mov dh,0x08
mov dl,0x12
int 0x10

mov si, display_text ;display display_text on screen
call print_string

;set x y position to text
mov ah,0x02
mov bh,0x00
mov dh,0x9
mov dl,0x10
int 0x10

mov si, os_info ;display os_info on screen
call print_string

;set x y position to text
mov ah,0x02
mov bh,0x00
mov dh,0x12
mov dl,0x17
int 0x10

mov si, author
call print_string
```

```
;set x y position to text
mov ah,0x02
mov bh,0x00
mov dh,0x14
mov dl,0x19
int 0x10
mov si, reg_no
call print_string
```

Outputs:


QEMU
Welcome to My OS!
>>>> Press any key <<<<


QEMU
Login please...
Username : achyut
Password :



Conclusion:

Programming an OS is one of the most interesting and satisfying experience in terms of learning. It helps me to understand the working of BOOT sector of an operating system, coding in assembly language, working with qemu emulator, fundamental concepts of assembly registers and their function, importance of magic number (0xaa55) in the kernel design and lots of important concepts related to Operating System.

I'll further increase the functionalities of my operating system by adding calculator function and memory management.

References:

- https://en.wikipedia.org/wiki/BIOS_color_attributes
- https://en.wikipedia.org/wiki/BIOS_interrupt_call
- https://www.tutorialspoint.com/assembly_programming/assembly_registers.htm
- https://github.com/cfenollosa/os-tutorial
- https://www.quora.com/How-can-I-learn-to-make-my-own-OS

Special thanks to Prof Saleem Durai Sir for his unconditional support and guidance.