Very Basic Operating System from Scratch

A TERM PROJECT REPORT

Submitted in partial fulfilment for the 2nd year, 4th semester of

Bachelor of Technology in

System Software CSD – 224

By

Avinal – 185067, 185067@nith.ac.in ,
Harsimranjeet – 185087, Pooja – 185101

Project Code on GitHub
https://github.com/Blitzar-to-Supernova/unnamedOS-SS

Submitted to
Dr. Jyoti Srivastava

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY
Hamirpur, Himachal Pradesh
India - 177005

Our Project

We tried building a very basic Operating System with limited number of functionalities. We had a template code base and we modified it to get in the form we have right now. There is no filesystem and any level of memory managements or process management. It is barebone, responds to some of the command. We can run it onto a virtual machine or maybe a real machine too. It has no GUI, just command line.

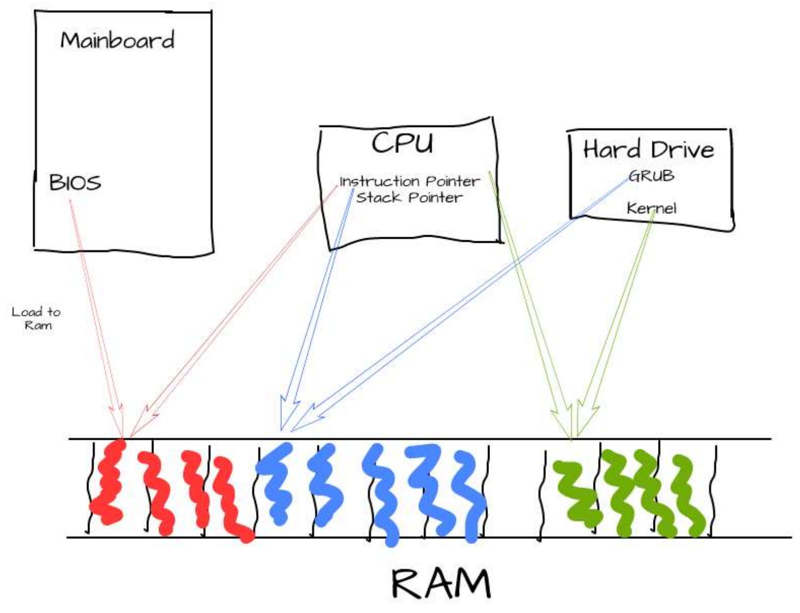
Some Theories

Operating System - An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Assembler - It is a program for converting instructions written in low-level assembly code into relocatable machine code and generating along information for the loader. It generates instructions by evaluating the mnemonics (symbols) in operation field and find the value of symbol and literals to produce machine code.

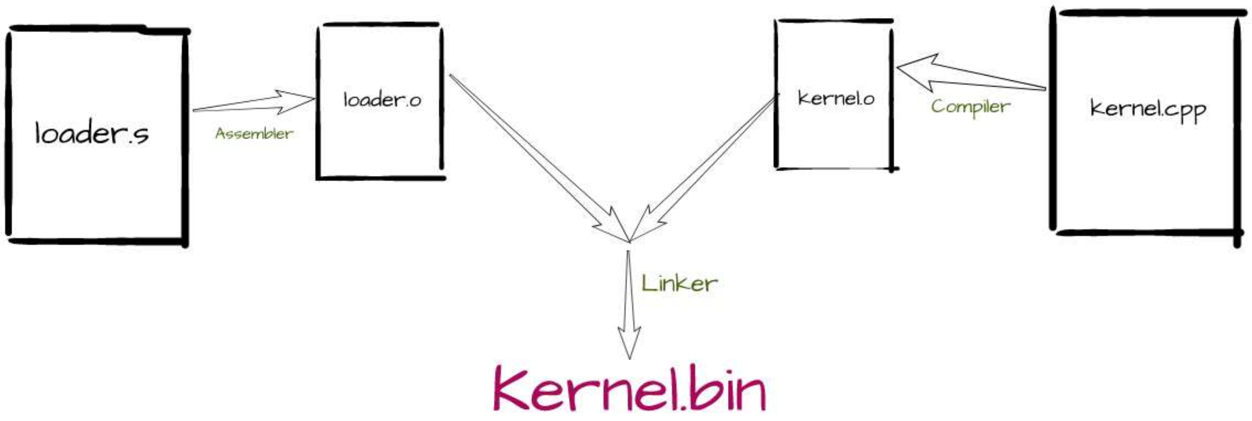
Loader - Loader is the program of the operating system which loads the executable from the disk into the primary memory (RAM) for execution. It allocates the memory space to the executable module in main memory and then transfers control to the beginning instruction of the program.

How does an Operating System works?



The Main board has special program called BIOS. At the start of the computer BIOS program is loaded into the ram. Then the Instruction Pointer of the CPU is set to this BIOS program. BIOS program reads the hard drive and loads the boot loader into the ram and sets instruction pointer to ram. Then boot loader executes and it reads the kernel from hard drive and loads it into the RAM.

The boot loader does not set the stack pointer. So, an operating system written in C++ there should be a method to set the stack pointer and then call the main function of the C++ program. Hence the Kernel of the OS should contain two programs. One is **loader written in Assembly** this can set the stack pointers and load the operating system into memory. Then the **operating system written in C++**.



File Description

- **boot.asm - main** is the starting point of our kernel. The code will set the stack pointer and will call the **main** function in our **kernel.c** file. Since we have written our operating system in **C** language and calling it in extern mode we do not need to mention **extern** in our **kernel.c** file. Also, we are proving interrupt codes to enable us input and output operations through keyboard. Mouse is still not supported.
- **kernel.c** – This is the starting point of our operating system. It contains a **main** function that will be called when operating system starts, and then it calls all other modules, such as built in functions and input output operations.
- **linker.ld** – This file combines all the files together and produces **kernel.bin.** This file later be used to produce disk image file for OS installation.
- **common.c** – Contains general purpose functions, for example, string copy, substring, split string, string to integer, integer to string.
- **error.c** – Code to show Blue Screen of Death error, it activates when there is some serious problem in system files or kernel.

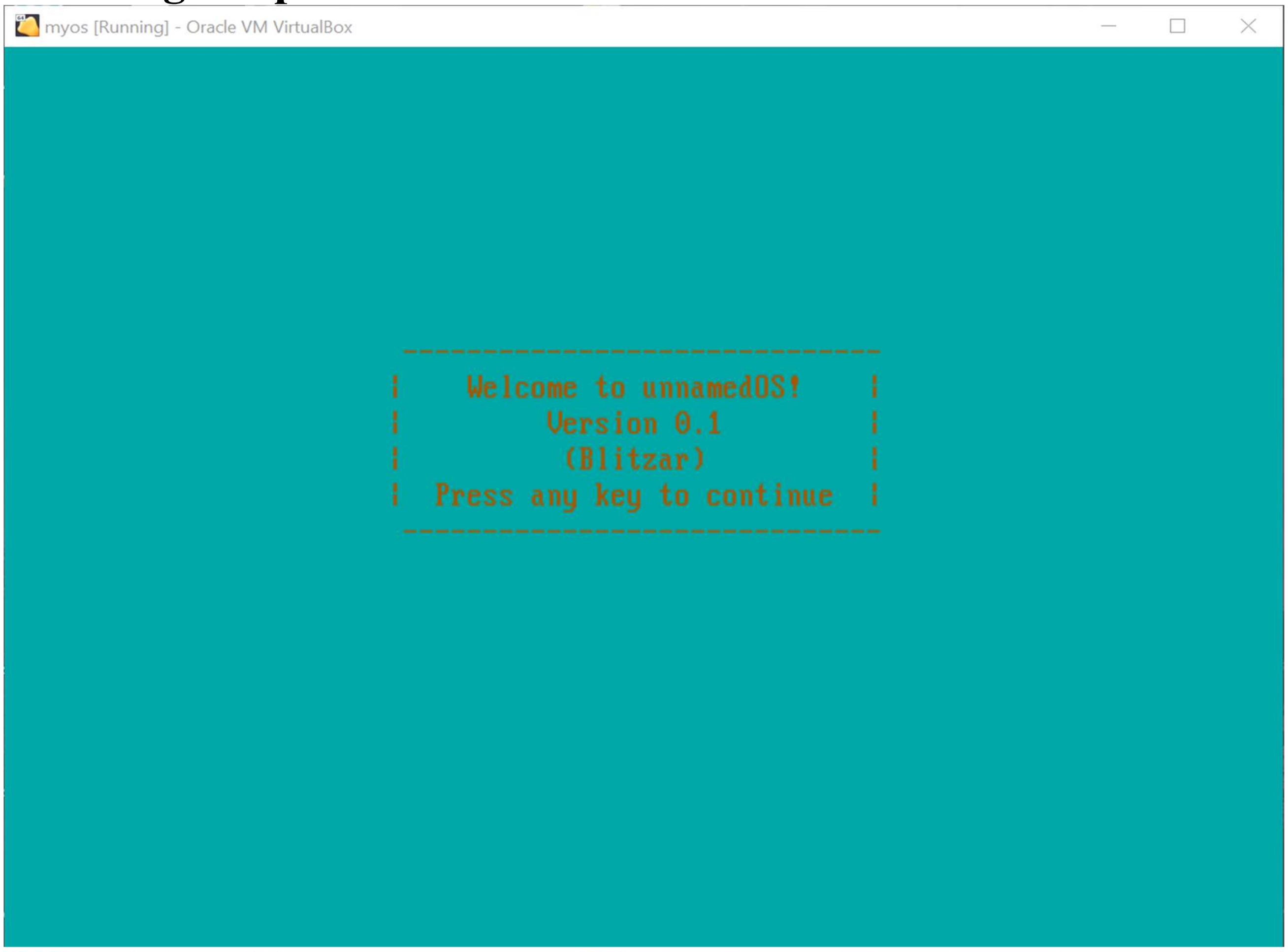
- **functions.c –** Contains various command line functions, currently we support these commands, sum, cls, fibo, setcolor, art, description is shown in help section.
- **gdt.c -** The Global Descriptor Table (**GDT**) is a data structure used by Intel x86-family processors starting with the 80286 in order to define the characteristics of the various memory areas used during program execution, including the base address, the size, and access privileges like executability and writability.
- **idt.c -** The Interrupt Descriptor Table (**IDT**) is a data structure used by the x86 architecture to implement an interrupt vector table. The **IDT** is used by the processor to determine the correct response to interrupts and exceptions.
- **isr.c -** An interrupt handler, also known as an interrupt service routine or ISR, is a special block of code associated with a specific interrupt condition. Interrupt handlers are initiated by hardware interrupts, software interrupt instructions, or software exceptions, and are used for implementing device drivers or transitions between protected modes of operation, such as system calls.
- **keyboard.c –** keyboard handler code. Contain description of each supported key.
- **screencontroller.c –** Output handler functions code.
- **shell.c -** Code to assist command line operations, everything on command line happens here.
- **timer.c –** Small snippet to provide timing related functionality.
- **vga.c –** Controls how characters are displayed on the screen including colour of the screen and text.

Available Commands

By typing 'help' we can see how many commands are available currently in the system. Given below is the description of commands.

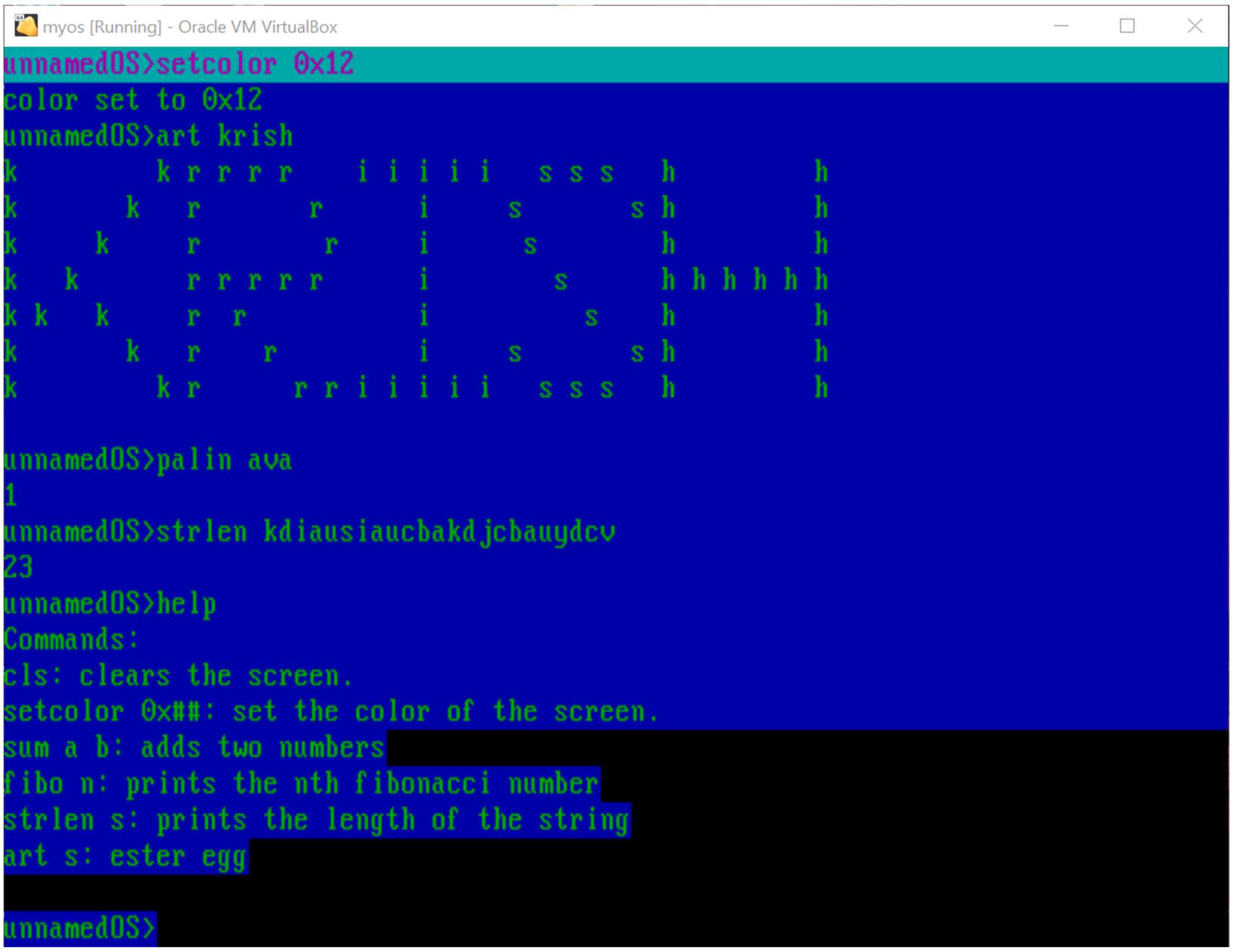
- **sum a b - Adds** two numbers and prints their sum.
- **fibo n –** Prints nth Fibonacci number
- **cls –** Clears the screen
- **setcolor 0x## -** Set colour of the background
- **art string –** Displays the string as a command line art.
- **strlen string –** Display the length of the string.
- **palin string –** Checks if the given string is a palindrome
- **binary num –** Convert the given number in binary

Working Snaps



myos [Running] - Oracle VM VirtualBox

```
 ------------------------------
 |    Welcome to unnamedOS!   |
 |        Version 0.1         |
 |         (Blitzar)          |
 |  Press any key to continue |
 ------------------------------
```



myos [Running] - Oracle VM VirtualBox

```
unnamedOS>setcolor 0x12
color set to 0x12
unnamedOS>art krish
k         k r r r r    i i i i i   s s s   h         h
k       k   r       r      i     s       s h         h
k     k     r        r     i      s        h         h
k   k       r r r r r      i        s      h h h h h h
k k   k     r  r           i          s    h         h
k       k   r    r         i     s       s h         h
k         k r      r r i i i i i   s s s   h         h

unnamedOS>palin ava
1
unnamedOS>strlen kdiausiaucbakdjcbauydcv
23
unnamedOS>help
Commands:
cls: clears the screen.
setcolor 0x##: set the color of the screen.
sum a b: adds two numbers
fibo n: prints the nth fibonacci number
strlen s: prints the length of the string
art s: ester egg

unnamedOS>
```