# TRIBHUVAN UNIVERSITY

## INSTITUTE OF ENGINEERING

# Advanced College of Engineering and Management

Department of Electronics and Computer Engineering


**A Minor Project Report On**

**Basic Operating System**

[CT654]

**Submitted By:**

Bishal Shrestha – ACE074BCT016

Kritika Simkhada– ACE074BCT024

Gaurab Silwal–ACE074BCT018

# TRIBHUVAN UNIVERSITY

## INSTITUTE OF ENGINEERING

## Advanced College of Engineering and Management

Department of Electronics and Computer Engineering


**A Minor Project Report**

**On**

**Basic Operating System**

**Submitted To:**

Department of Electronics and Computer Engineering

ACEM

**Submitted By:**

Bishal Shrestha – ACE074BCT016

Kritika Simkhada– ACE074BCT024

Gaurab Silwal–ACE074BCT018

**Supervised By:**

Er. Kushal Dahal

Er. Bikash Acharya

# CERTIFICATE OF APPROVAL

The undersigned certifies that the minor project entitled “Basic Operating System” submitted by Bishal Shrestha, Gaurab Silwal, Kritika Simkhada to the Department of Electronics & Communication and Computer Engineering in partial fulfillment of requirement for the degree of Bachelor of Engineering in Computer Engineering. The project was carried out under special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled, bonafide and ready to undertake any commercial and industrial work related to their field of study and hence we recommend the award of Bachelor of Computer Engineering degree.

_____ _____

**Er. Kushal Dahal** **Er. Bikash Acharya**

(Project Supervisor) (Project Supervisor)

_____

**Er. Pradip Khanal**

(Project Coordinator)

_____

**Er. Ajaya Shrestha**

Head of the Department

Department of Electronics and Computer Engineering

# COPYRIGHT

# ACKNOWLEDGEMENT

We take this opportunity to express our deepest and sincere gratitude to our supervisor **Er. Kushal Dahal**, Lecturer, Electronics and Computer Department and **Er. Bikash Acharya**, Lecturer, Electronics and Computer Department for their insightful advice, motivating suggestions, invaluable guidance, help and support in successful completion of this project and also for their constant encouragement and advice throughout our Bachelor's programme.

We express our deep gratitude to **Er. Pradip Khanal,** Lecturer, Electronics and Computer Department for his regular support, co-operation, and coordination.

The in-time facilities provided by the department throughout the Bachelors program are also equally acknowledgeable.

We would like to convey our thanks to the teaching and non-teaching staff of the Department of Electronics & Communication and Computer Engineering, acem for their invaluable help and support throughout the period of Bachelor's Degree. We are also grateful to all our classmates for their help, encouragement and invaluable suggestions.

Finally, yet more importantly, we would like to express our deep appreciation to our parents, sister and brother for their perpetual support and encouragement throughout the Bachelor's degree period.

**Bishal Shrestha (074/BCT/016)**

**Gaurab Silwal (074/BCT/018)**

**Kritika Simkhada (074/BCT/024)**

# ABSTRACT

As a computer engineering student, we are taught from 8085 and 8086 architecture, assembly programming to high level object oriented programming language. Different subjects such as Microprocessors, Computer Organization and Architecture, Data Structure and Algorithm and Operating System that are part of our Engineering curriculum gives sufficient knowledge to build our own simplest operating system. This will give us core knowledge and implementation of our curriculum as an Engineer. In order to implement these knowledge, developing a basic operating system from scratch would be the best way to do so. We found that even during present time, there is not much active development of new operating systems due to the immense knowledge required to design one. Therefore, in this project we try to develop an OS that performs basic operations such as displaying through VGA, taking input from keyboard and mouse and memory management.

## Keywords

1. Kernel
2. Bootloader
3. VGA
4. GDT
5. IDT

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ABBREVIATIONS

1. BIOS: Basic Input Output System
2. CLI: Command Line Interface
3. CPU: Central Processing Unit
4. DS: Data Segment
5. GDT: Global Descriptor Table
6. GRUB: Grand Unified Bootloader
7. GUI: Graphical User Interface
8. IDT: Interrupt Descriptor Table
9. IVT: Interrupt Vector Table
10. MMIO: Memory Mapper I/O
11. OS: Operating System
12. RAM: Random Access Memory
13. ROM: Read Only Memory
14. SD: Segment Descriptor
15. SI: Source Index
16. VGA: Video Graphics Array
17. VRAM: Video Random Access Memory

# 1. Introduction

## 1.1 Background

Operating System is the primary System Software required for any bare computational system to run. The Operating System acts as a bridge between Hardware and the User. Operating System is the first program to start when the computer is turned on and all the other subsequent programs are launched by the OS. Depending upon the use, there are different types of OS such as distributed OS and real time OS. The use of OS is abundant in Embedded Systems, where miniature specific-purpose design is implemented rather than general-purpose design. Currently, most of the computerized system contains OS, making its availability everywhere. OS has two types of interface, Command Line Interface (CLI) and Graphical User Interface (GUI) making the use of OS simplistic and reliable. For the sake of basicity, this project is based on CLI. The lowest part of an OS is the kernel. Kernel is the core of an OS which handles and maintains all the processes, device drivers, I/O management and file management.

## 1.2 Motivation

Operating System is the core of every digital system currently operating in this world. It is the primary level that helps software interact with the binary hardware. There are numerous Operating System Software till date that operate on a global basis. Some of the leading OSs are Linux, Windows, MacOS and so on.

The curriculum of Computer Engineering is designed such that a student is taught every skill required in the field of engineering. The fact that there are very few Operating System Softwares being built at the current time and the curiosity to understand how a computer works from core encouraged us to take this path.

## 1.3 Problem

There are many problems encountered in this project. The first and foremost being lack of resources online through which we could navigate easily. Building an OS from scratch requires an immense amount of knowledge from hardware level to assembly level and higher level language.

When the OS is compiled above another OS, there arises an issue of our OS not being compatible in other systems which is solved by the use of Cross-Compiler. Designing an operating system, we must use complex data structures such as Global Descriptor Table and Interrupt Descriptor Table which is hard to implement in coding consuming more time. Programming in low level, we must make sure that the OS is byte-perfect which means we must calculate every byte invested in the computation. During the boot, stack must already be initialized, which is not done by the system, this was solved by defining stack during the loading in assembly.

## 1.4 Scope and Application

Most of the electronic devices contain OS whether being specific-purpose or general-purpose. Among which x86 architecture CPUs are widely used in many general-purpose computers. Hence, this Operating System Software can be implemented in any x86 compatible computer system.

The main application of Designing Operating System from scratch is in the field of research and development (R&D). This project can be used in Universities to learn about the basic operating system through code. Moreover, it is favorable when a person wants full control over the CPU and is comfortable with Character User Interface (CUI). Furthermore, this project can be expanded into more advanced and user-friendly versions such as current popular Operating Systems such as Linux. Since, this project focuses on the development of kernel, this OS can be modified on the functionality as per the requirements.

## 2. Objective

The objective of our project is:

- To implement Basic Kernel from scratch to understand the core of an Operating System.

# 3. Literature Review

An operating system is simply a program with special privileges on the hardware which let them manage all the other programs. They are the first program to start when the computer is turned on and all the other subsequent programs are launched by the OS.

Back in the 1950s, the computers were simply mainframes without any operating system that rely on punch cards input as well as magnetic and paper tapes. Back then, interfacing with early peripherals was very low level, requiring programmers to know all the hardware details about each device. Also, programmers used to have rare access to every model of a peripheral to test their code on. Programmers themselves had to write the best possible program they could by reading manuals. So, to make it easier for programmers, the concept of OS was developed. Operating systems were implemented as a bridge between software programs and hardware peripherals. In the late 50s, University of Manchester, UK, wrote a computer program for their own computer (known as Atlas computer) and the program was called 'Atlas Supervisor'. At that time, Atlas Supervisor was considered to be the first recognizable modern operating system that not only loaded programs automatically, like earlier batch systems but also ran several programs at the same time on its single CPU. Moreover, in the late 1960s, the first version of the UNIX operating system was developed. Since, UNIX easily adapted to the new systems so it quickly achieved wide acceptance. Even modern operating systems like Apple OS, different versions of Linux rely on the UNIX OS. Today, the vast majority of the modern operating systems are dominated by Microsoft Windows, Apple OS and different versions of Linux.

With the advancement of technology, people are getting more and more dependent on machines and gadgets. As the operation of any computer system requires an operating system in it, we tried to develop our own operating system that works on basic memory management. In the first place when our operating system is booted to the computer, a simple color text mode (VGA) mode will be displayed on the computer screen. Our operating system will be able to manage the inputs from the keyboards and the mouse and will be able to display the corresponding output on the screen. Also we will implement a data structure called Global Descriptor Table (GDT) to manage the segments of the memory. Similarly, another data structure called Interrupt Descriptor

Table (IDT) will handle the interrupts (signals issued in response to any software or hardware events).

# 4. Feasibility Study

This project is applied in any x86 architecture compatible environment. The following analysis will ensure that no problem will arise during actual implementation of the project.

## 4.1 Technical Feasibility

This study defines if the project can be feasible in terms of technical requirements or not. All the necessary technologies are easily available everywhere. All the modern computer systems support backward compatibility which means this project can be run in any computer system having x86 architecture. To expand the services and functionality, the system can be programmed. So, if there will be any update needed, upgrading is possible. Hence, we can conclude that it is technically feasible.

## 4.2 Operational Feasibility

A project needs to be applicable in the real world. If it exists only in the theoretical approach it cannot be considered beneficial. This project is feasible operationally. This project is being used in every person's day to day life, no user training is required. Any failure in the system can be maintained.

## 4.3 Economical Feasibility

The development of this basic OS required only three human resources using their own development environment and the finalized product can be simply deployed to the market through the internet. Moreover, this basic OS can simply run in any computer system with x86 compatibility. This makes zero production cost and hence it can be said that this is economically feasible.

## 4.4 Schedule Feasibility

The majority of time consumed by the project is in research and coding and also in analysis. Once operational software is developed, the project can be implemented to all the IA-32 standard computer systems. A time schedule was defined before the initiation of the project. There was a deadline for the completion of the project and it was completed by the defined period. The majority of the time was taken in research and documentation. The project was completed within the scheduled time frame.

# 5. Requirement Analysis

For implementation on a real computer system there are some hardware requirements as well but for a prototype there are no hardware requirements but there are some software requirements.

## 5.1 Functional requirements

- x86 architecture compatibility
- I/O from I/O devices (Keyboard, Monitor)
- Memory Management
- Interrupt Management

## 5.2 Non-Functional requirements

- Compatibility
- Extensibility
- Maintainability
- Manageability
- Portability

# 6. System Design and Architecture

## 6.1 System Design

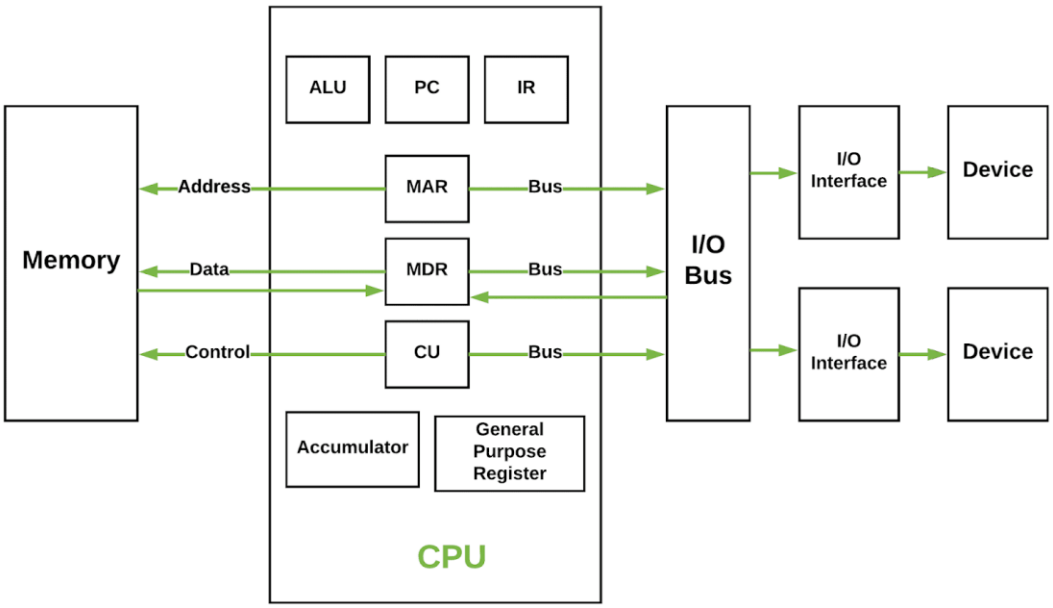


*Figure 6.1.1: Von Neumann Architecture*

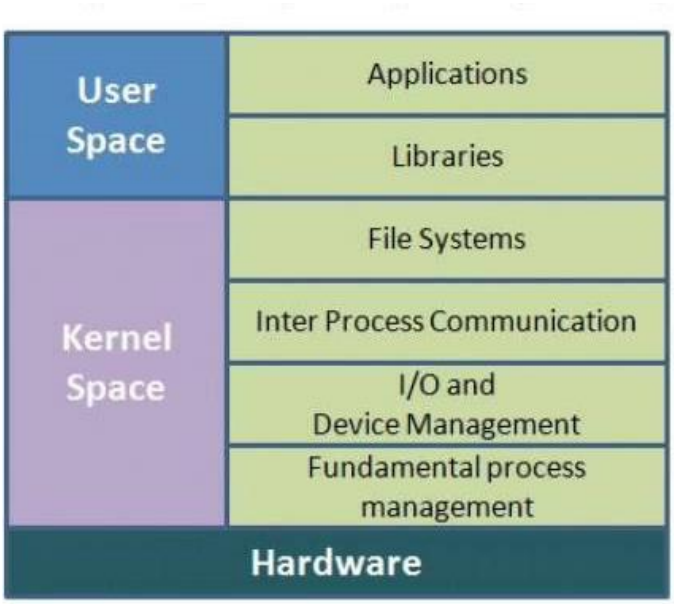


*Figure 6.1.2: Monolithic architecture of Operating System*

## 6.2 Data Flow Diagram

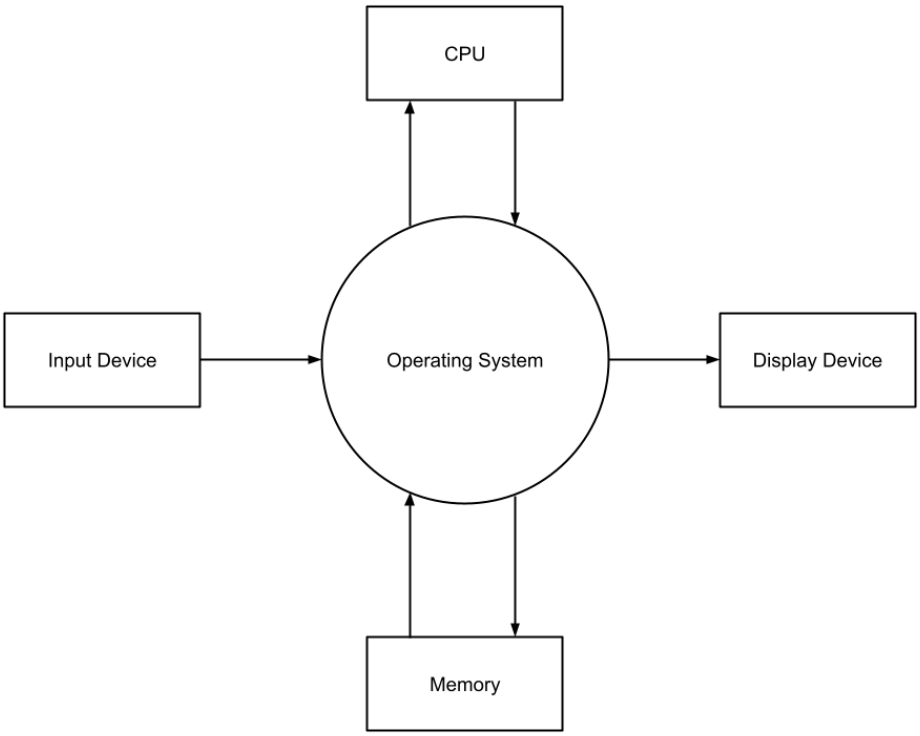


*Figure 6.2.1: Context Level Diagram (Level 0 DFD)*

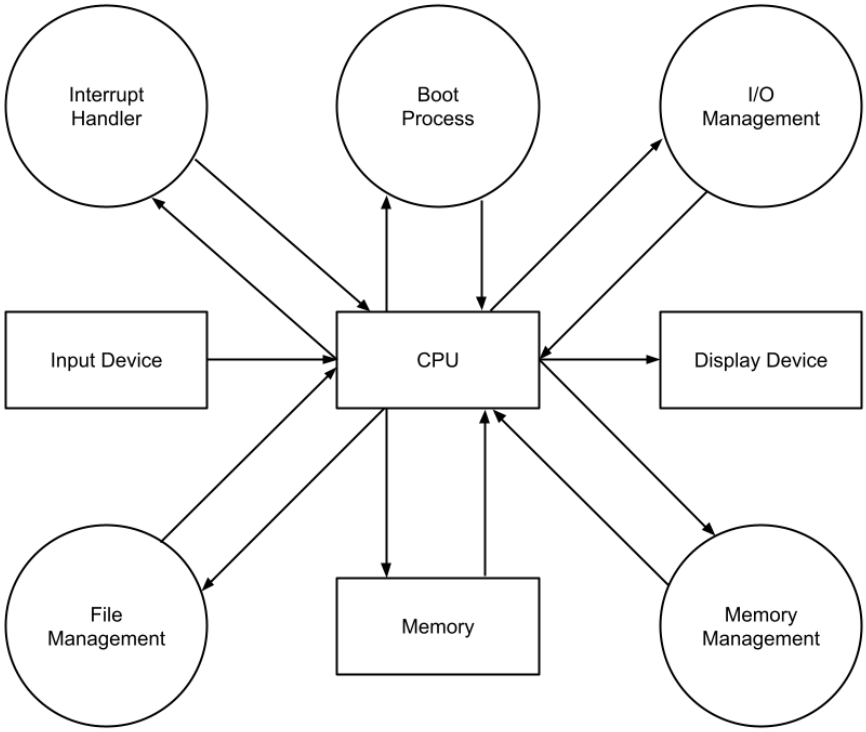


*Figure 6.2.2: Level 1 DFD*

## 6.3 Use Case Diagram

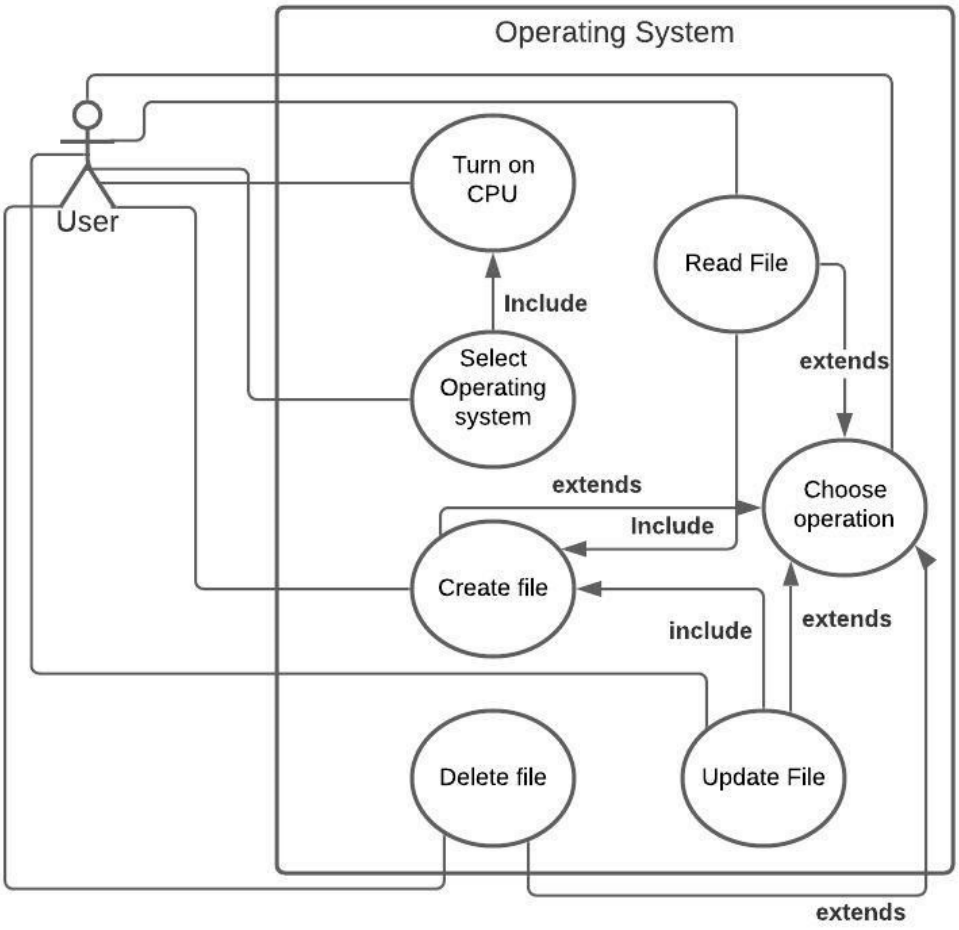


*Figure 6.3: Use case diagram of operating system to be designed*

## 6.4 Sequence Diagram

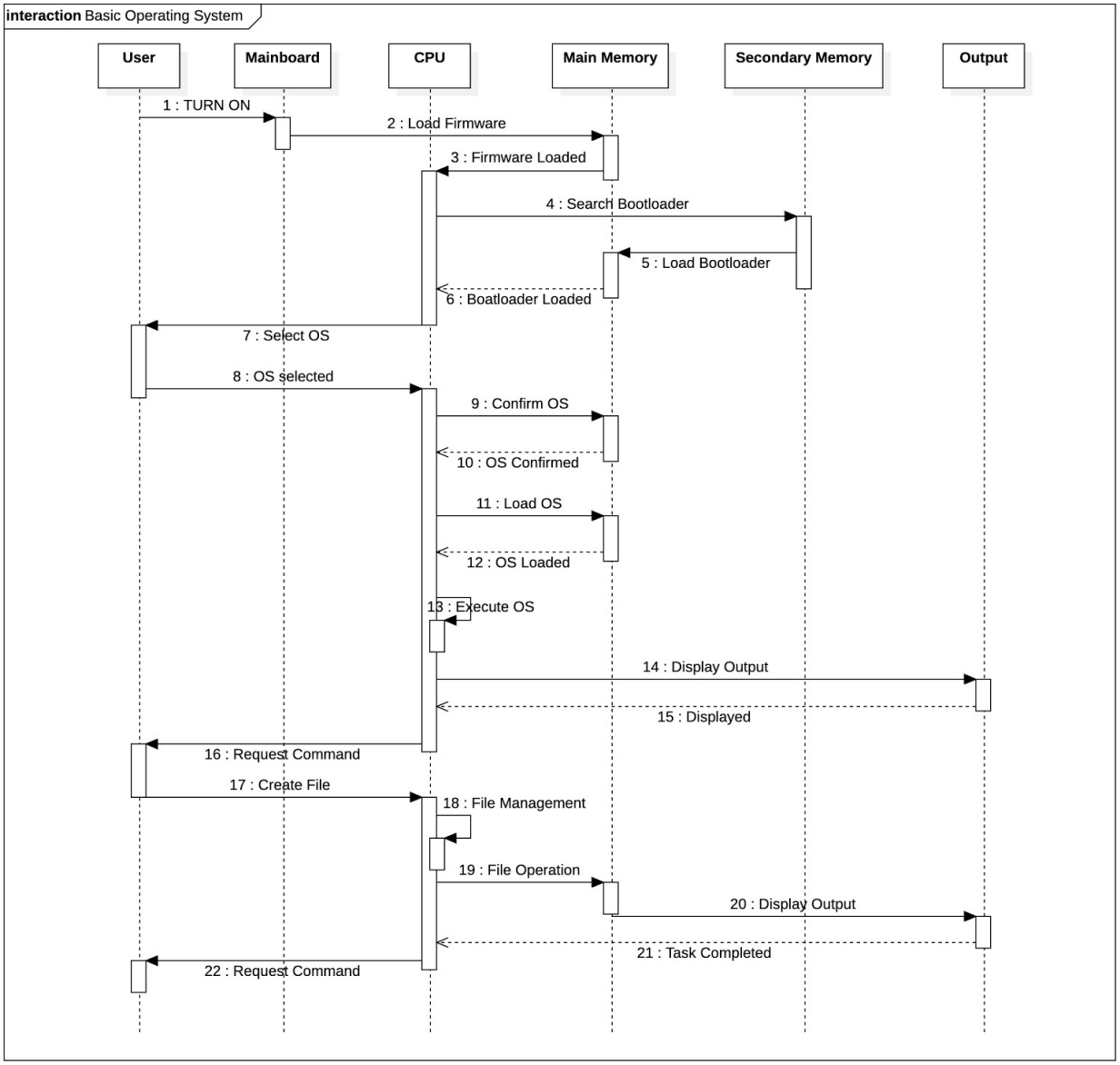


*Figure 6.4.1: Sequence Diagram of Basic Operating System*

# 7. Methodology

The development of Operating Systems from scratch begins with in-depth knowledge on the topic. The designing of OS from boot to final stage of the project is listed and explained below.

## 7.1 Boot Process

When we boot a computer, it needs to start up without the knowledge of an Operating System. It needs to search and load our Operating System to the memory. As a pre-OS environment, Motherboard contains Basic Input/Output Software (BIOS), a collection of software routines that are initially loaded from a chip into memory and initialized when the computer is switched on. BIOS provides auto-detection and basic control of computer's essential devices such as screen, keyboard and hard disks.

After BIOS completes its low-level tests of the hardware, it needs to boot OS stored in one of the devices. However, BIOS has no notion of file-system which makes it unable to load files that represent OS. Hence, BIOS finds OS in the first sector of one of the disks (i.e. Cylinder 0, Head 0, Sector 0), known as the *boot sector*.

BIOS then loads Bootloader from the boot sector to the main memory. Bootloader is a sophisticated special operating system software which contains partition tables and file systems so it can deal with directory structure. Bootloader reads the directory in the second partition (i.e. /boot/grub/grub.cfg) which contains the information on the Operating System intended to run. Bootloader needs to be certain that the boot sector contains boot code for execution of OS or simply data. It looks for the last 32-bit word of an intended boot sector (512B in size) where the *magic number (boot signature)* must be set. CPU begins executing the first boot sector it finds that ends with the magic number.

For the kernel to be booted in a uniform way by Multiboot-compliant bootloaders, we use 0x1BADB002 as magic number. In x86 architecture, it handles multi-byte values in *little-endian* format, where less significant bytes proceed more significant bytes.

```
e9  fd  ff  00  00  00  00  00  00  00  00  00  00  00  00  00
00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
*
00  00  00  00  00  00  00  00  00  00  00  00  02  b0  ad  1b
```

*Figure 7.1.1: A machine code boot sector with each byte displayed in hexadecimal.*

Along with the magic number, the multiboot header also requires a flag and checksum field defined. If bit 0 in the flags word is set, then all boot modules loaded along with the operating system must be aligned on page (4KB) boundaries. Checksum is used for special purposes by bootloaders and its value must be the sum of magic number and flags.

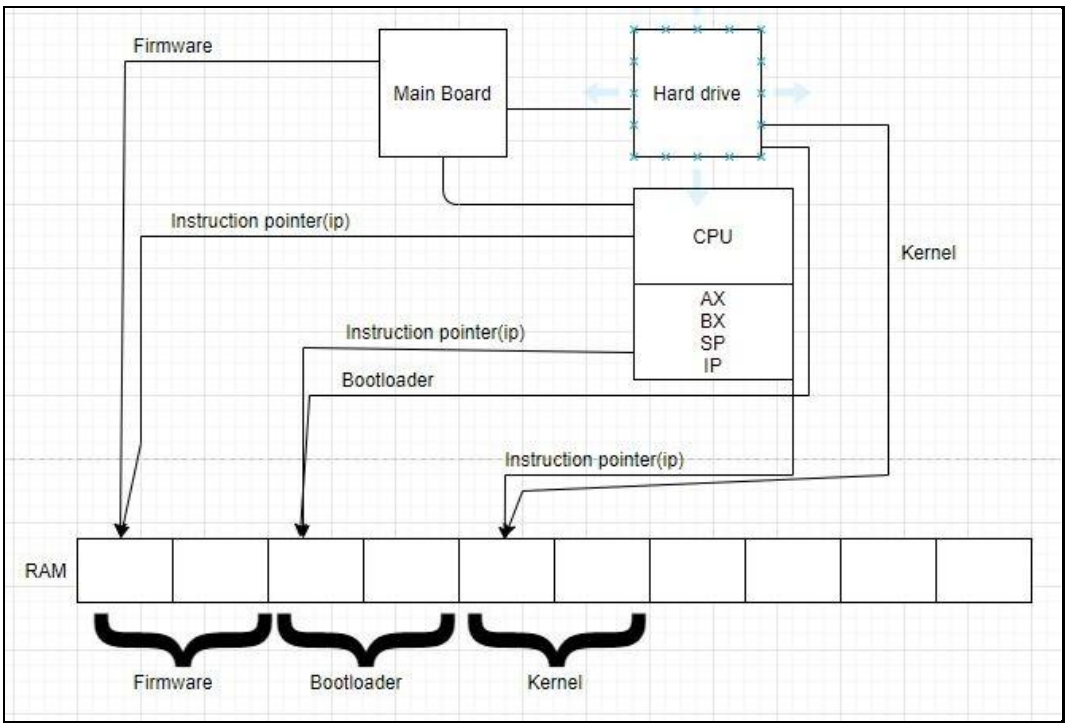


*Figure 7.1.2: System flow during boot process.*

To sum it up, when a computer is turned on, the mainboard loads BIOS firmware into the main memory where the CPU points its Instruction Pointer to it. Firmware then tells

the CPU to look into the first sector of the hard drive for the boot sector. Then the CPU loads the bootloader into the main memory which then searches for the Operating System Software verified using magic number. After the bootloader finds its intended Operating System, it loads Kernel into the main memory through which the OS starts to operate as shown in figure 7.1.2.

## 7.2 Creation of Binary Executable File

For the development of OS, both low-level programming language and high-level programming language are used. Assembly language is used for machine level programming such as setting the value in the registers, calling the main function and setting the flags. Object-Oriented language, C++ is used as a high-level language for higher level functionality such as VGA, GDT & IDT.

GNU/Linux assembler, ‘as’ is used to assemble the assembly files whereas GNU/Linux compiler, ‘g++’ is used to compile cpp files. These produce two different types of object files, machine code output of an assembler or compiler. These two different object files need to be linked together which is done by GNU/Linux linker, ‘ld’ which gives us .bin file, binary executable file which executes the OS after being loaded into the main memory.

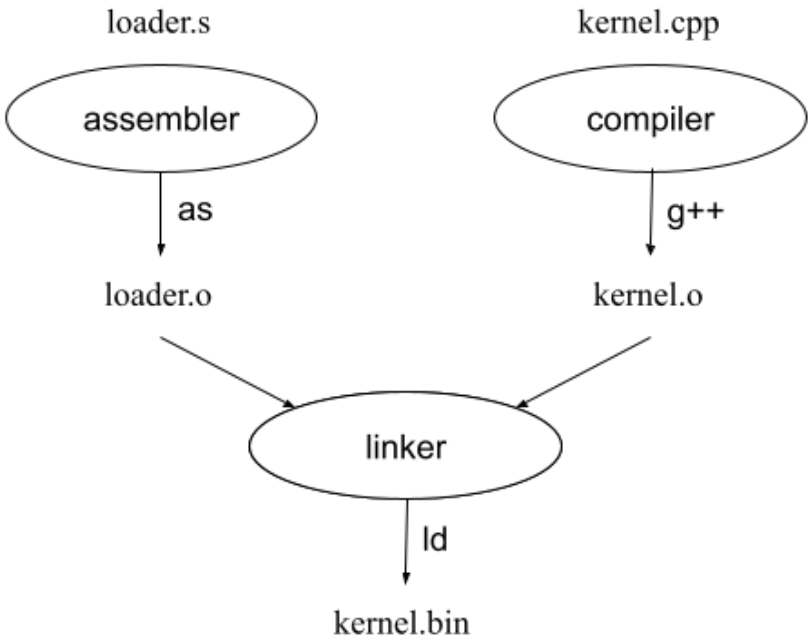


*Figure 7.2.1: Process to creation of .bin file*

## 7.3 X86 architecture

X86 is a family of instruction set architecture initially developed by Intel based on the Intel 8086 microprocessor and its 8088 variant. The x86 family of computers follow the Von Neumann Architecture. The term “x86” came into being because the names of several successors to Intel’s 8086 processor end in “86”. Intel’s 32-bit 80386 (later known as i386) gradually replaced the earlier 16-bit chips; this extended programming model was originally referred to as the i386 architecture but Intel later dubbed it IA-32.

In x86 architecture, byte addressing is enabled and words are stored in memory with *little-endian* byte order. Memory access to unaligned addresses is allowed for all valid word sizes. In 32-bit, the registers are prefixed with ‘e’ such as eax and ebx. In 32-bit, there are additional general registers, ‘fs’ and ‘gs’ and has 32-bit memory offset.

### 7.3.1 Real Mode

Real mode, also called read address mode, is an operating mode of all x86-compatible CPUs. In real mode, segmentation is achieved by shifting the segment address left by 4 bits and adding an offset in order to receive a final 20-bit address (DS x 10h + SI). Thus the total address space in real mode is $2^{20}$ bytes, or 1 MB. Every type of access (code, data or stack) has a default segment register associated with it (for data the register is usually DS, for code it is CS, and for stack it is SS).

Real mode provides no support for memory protection, multi-tasking, or code privilege levels (Ring levels).

### 7.3.2 Protected Mode

Protected mode, also called protected virtual address mode, is also an operational mode of x86-compatible CPUs. Protected mode supports memory protection, safe multi-tasking and code privilege levels, virtual memory and paging.

In protected mode, a segment register no longer contains the physical address of the beginning of a segment, but contains a “selector” that points to a system-level structure called a segment descriptor. A segment descriptor contains the physical address of the

beginning of the segment, the length of the segment, and access permissions to that segment.

We can enter protected mode by defining complex data structure, GDT. It describes permission levels when accessing memory.

The different protection ring for privilege levels are:

- Ring 0: Accessible to the kernel, in kernel mode. This ring has direct access to the CPU and the system memory, so any instructions requiring the use of either will be executed here.
- Ring 1: It interacts with and controls hardware connected to the computer. Playing a song through speaker or displaying video on monitor are examples of instructions that would need to run in this ring.
- Ring 2: It is used for instructions that need to interact with system storage, loading or saving files (I/O).
- Ring 3: It is the least privileged ring, accessible to user processes that are running in user mode. This ring has no direct access to the CPU or memory and therefore has to pass any instructions involving these to ring 0.

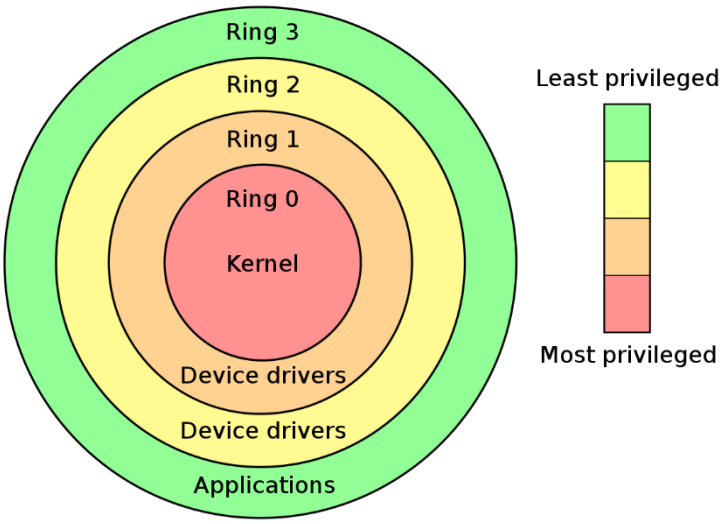


*Figure 7.3.1: Protection ring*

## 7.4 Video Graphics Array (VGA)

Video Graphics Array (VGA) is a video display controller and accompanying de-facto graphics standard. The video memory of VGA is mapped to the PC's memory (MMIO) via a window in the range between segments 0xA0000 and 0xBFFFF. A video buffer is a segment of *memory mapped* as Video Memory where we can change what region of memory is mapped to video memory. The typical segments are:

- 0xA0000 for VGA graphics mode (64KB)
- 0xB0000 for monochrome text mode (32 KB)
- 0xB8000 for color text mode (32 KB)

### 7.4.1 APA Graphics

All Points Addressable (APA) is a display mode that consists of a pixel array where every cell can be referenced individually. In this mode, every cell represents a *pixel*, the smallest unit that can be represented on a display. These pixels can be manipulated directly by using a pixel buffer.

### 7.4.2 Text Modes

A text mode is a display mode where the content on the screen is internally represented in terms of character rather than pixels, as with APA. In this mode, it uses two buffers: A character map representing the pixels for each individual character to be displayed (attributes), and a buffer that represents what characters are in each cell (characters). Some text modes also allow attributes, which may provide a character color, or even blinking, underlined, inversed, brightened, etc.

We use color text mode giving us 80x25 character cells accessible from memory 0xB8000. Each character cell occupies 2 bytes, first byte is for ASCII value of the character, and second byte to configure attributes of the character.

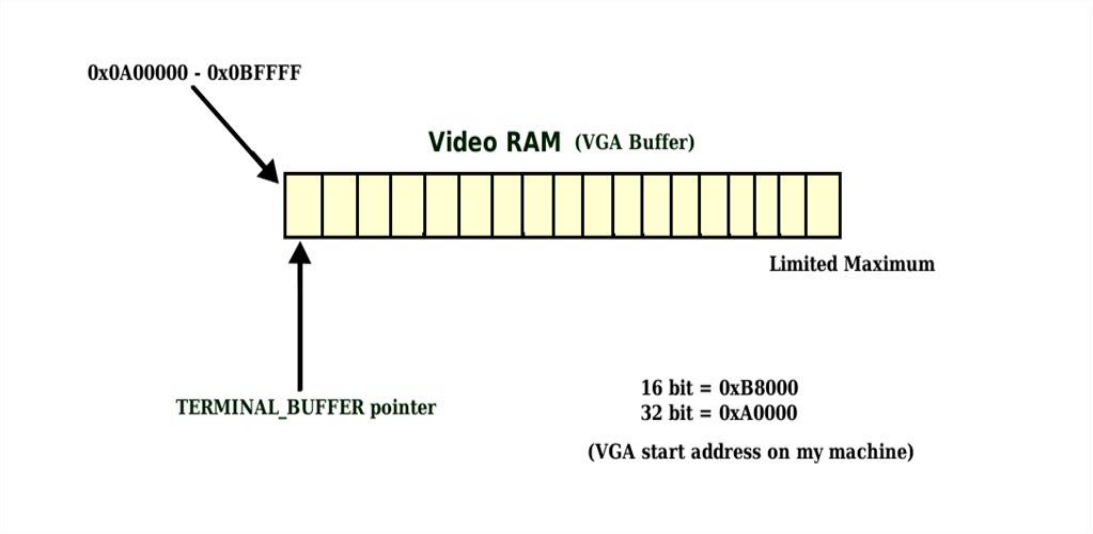


*Figure 7.4.1: Structure of Memory Mapped VGA*

In a 16-bit real mode system, VGA can be accessed using VGA Video BIOS Interface, a set of video interrupts (software interrupt 0x10). Because these are BIOS interrupts, they can only be used in real or v86 modes. For 32-bit protected mode, these BIOS interrupts cannot be used. Therefore, we have to manually create a service routine to access VGA through memory location.

In addition, VGA provides 15 colors, black(0), blue(1), green(2), cyan(3), red(4), magenta(5), brown(6), light grey(7), dark grey(8), light blue(9), light green(10), light cyan(11), light red(12), light magenta(13), yellow(14), white(15).

## 7.5 Global Descriptor Table (GDT)

Global descriptor table (GDT) is the memory structure that describes the CPU attributes of the segment. Each descriptor in the table is 8-byte wide among which the first descriptor is always a null descriptor and cannot be used to access memory. Usually, this null descriptor is used to store the pointer to the GDT itself. The remaining descriptors in the GDT describe the attributes of the segment like base address of the segment, limit size of the segment, privilege level and the like. The basic structure of a descriptor in the GDT is shown in figure 7.5.1.

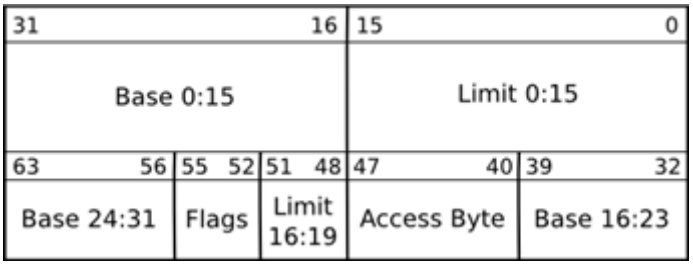


*Figure 7.5.1: Structure of a single descriptor in the GDT*

- Base Address (32 bits) defines the beginning of the segment in the physical memory.
- Segment Limit (20 bits) defines the size of the segment.
- Remaining bits are used for describing other aspects of the segment like privilege level of the memory access, presence of the segment, descriptor type (system or non-system) and so on.

GDT Descriptor is used to load GDT to the memory, where GDT Descriptor is 6-bytes in size containing 16-bit GDT size and 32-bit GDT address.

## 7.6 Interrupt Descriptor Table (IDT)

When a specific interrupt arises, the CPU looks up for an entry for that specific interrupt from a table called Interrupt Descriptor Table (IDT). Protected mode uses Interrupt Descriptor Table (IDT) which is the counterpart of Real mode Interrupt Vector Table (IVT). For locating the address of Interrupt Service Routine (ISR). It is similar to the GDT in structure. The location of the IDT is stored in the IDTR register of the CPU, which can be loaded or stored using LIDT/SIDT instructions. Since, there are 256 different interrupts, so IDT should have 256 entries (gate descriptors), each corresponding to the specific interrupt. Each entry in IDT is 8-byte long and contains either of the three gate descriptors: task gate, interrupt gate or trap gate. The structure of the gate descriptors in the IDT is shown in figure 7.6.1.

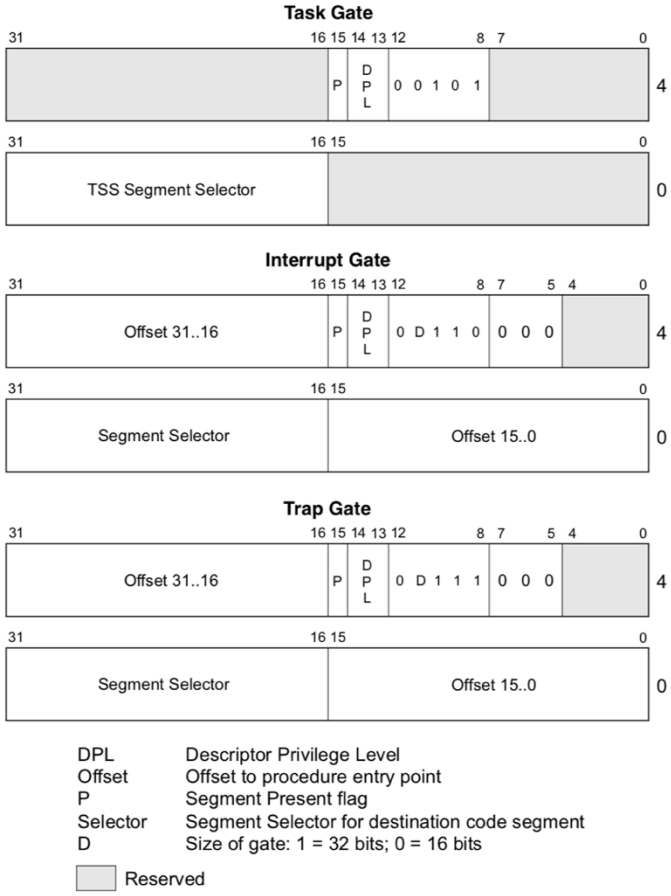


*Figure 7.6.1: Structure of different gate descriptors in the IDT*

- Offset (32 bits) represents the address of the entry point of an ISR.
- Segment Selector (16 bits) points the valid descriptor in our GDT.
- Remaining bits are used to describe gate-type, privilege level, storage segment and so on.

**- Task Gate:** A task gate is a special type of gate which is stored in IDT and GDT and is used for task switching. Its main objective is to trigger the hardware task switches in reply to interrupt arrival.

**- Interrupt Gate:** This gate is used to transfer control to the interrupt handler (also known as Interrupt Service Routine) with interrupt disabled upon entry and re-enabled upon return.

**- Trap Gate:** This gate also transfers control to the interrupt handler but interrupts remain unchanged.

## 7.7 Memory Management

Memory management is one of the most important parts of any operating system. Allocating and freeing memory within the required time is the major responsibility of the operating system kernel. For example, a user writes a program to solve a particular problem, then the basic task of an operating system is to make the hardware execute this program for the user and provide him/her the desired solution. For the execution of any program, the code and the data of the program have to be kept somewhere in the system. Thus the concept of memory arises here. The basic hierarchy of the memory in the computer system is shown in figure 7.7.1.

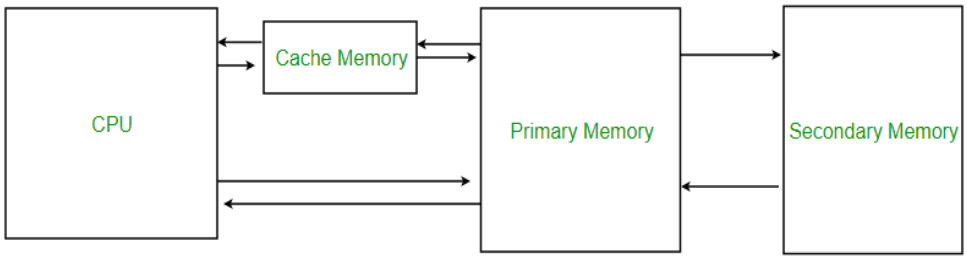


*Figure 7.7.1: Basic Memory Hierarchy in the Computer Architecture*

In memory management, we basically deal with three main components i.e. Processor, Main Memory and Secondary Memory. Overall data of a program is stored in the secondary memory and the data required during run time is stored in a main memory. The size of the main memory is small such that its access time is also less. Thus, computer systems are designed in such a way that our processor will interact with main memory for processing instructions. But the problem here is that our CPU generates a logical address (which actually works on secondary memory). A physical address is required to access a main memory. So, the role of an OS are:

- To fetch data from the secondary memory and allocate it in the suitable location of the main memory.
- To translate logical addresses into physical addresses.

# 8. Result


*Fig 8.1: Bootloader*


*Fig 8.2: Main screen*


*Fig 8.3: Using Commands*


*Fig 8.4: Help Command*


*Fig 8.5: Arithmetic Operations (addition)*


*Fig 8.6: Arithmetic Operations (subtraction)*


*Fig 8.7: Arithmetic Operations (division)*


*Fig 8.8: Arithmetic Operations (multiplication)*

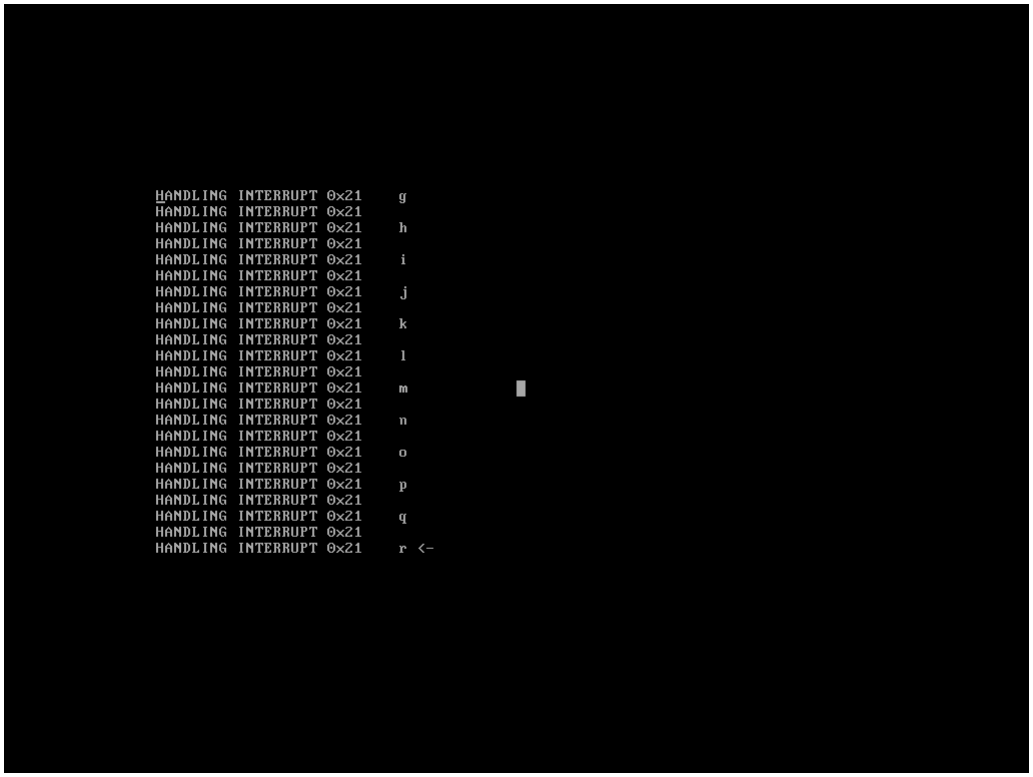


*Fig 8.9: Keyboard Interrupt*


*Fig 8.10: Mouse Interrupt*

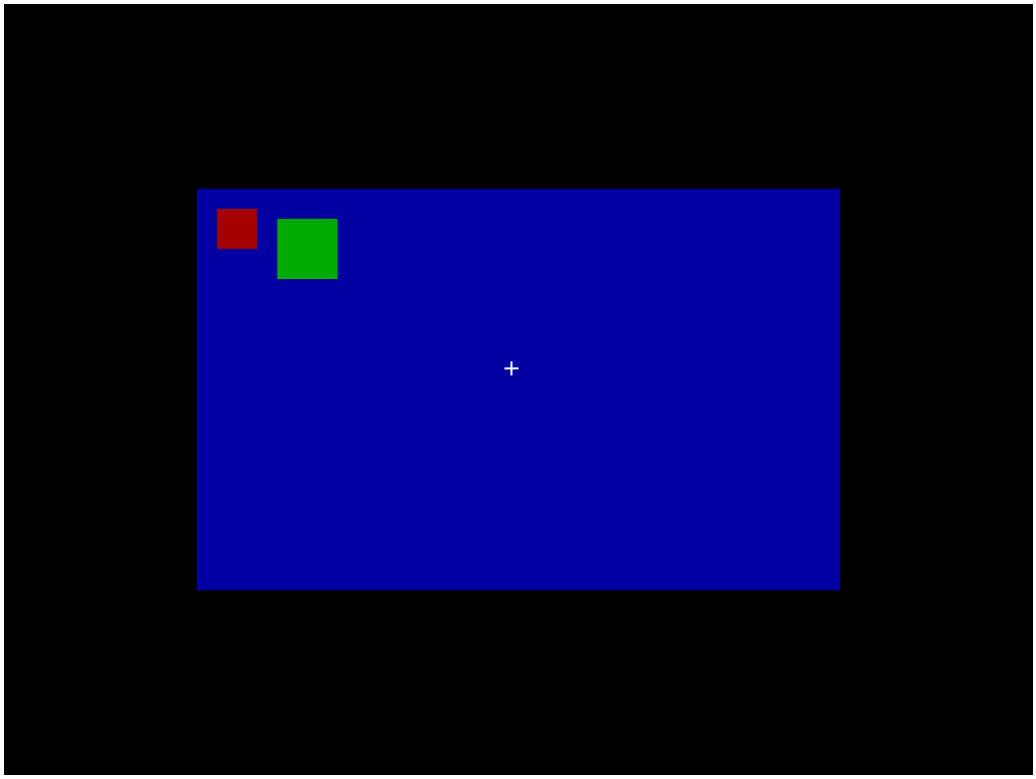

*Fig 8.11: Video graphic array representation*

# 9. Time Schedule

The project was divided into six phases, the phases along with their timeline is listed and illustrated below:

- Research
- Requirement Analysis
- Coding
- Testing
- Implementation
- Documentation

| S.N. | Tasks | Week 1 | Week 2 | Week 3 | Week 4-6 | Week 7 | Week 8 |
|---|---|---|---|---|---|---|---|
| 1 | Research | | | | | | |
| 2 | Requirement Analysis | | | | | | |
| 3 | Coding | | | | | | |
| 4 | Testing | | | | | | |
| 5 | Implementation | | | | | | |
| 6 | Documentation | | | | | | |

*Table 9.1: Time scheduling and management of the project*

# 10. Conclusion

The main purpose of selection of this project was to gain knowledge on how an operating system works on a bare hardware providing the user-friendly interface on the surface and management of operations on inside. On the timeline of project, we understood from loading of OS to the main memory to installing OS features such as system commands and arithmetic operations. In this minor project, we learnt and implemented the boot loading, GDT for segmentation of data and code segment, IDT for defining the Interrupts and VGA to display. During the process, we encountered numerous errors and problems due to complex data structure and algorithm. Such errors were tackled by the help of online OS Development community. At the final stage, we were able to write a basic operating system having basic features such as CUI, taking input from keyboard and mouse and performing operations by taking the pre-programmed commands.

# 11. Limitations and future works

## 11.1 Limitations

- The current OS do not handle multiple active interrupt.
- The secondary storage device is not yet accessed.
- There is no any file system through which we can manipulate file system.
- It has no extensive graphical user interface.
- The functionality of mouse buttons are yet to be defined.
- This basic OS do not have networking feature.

## 11.2 Future works

- To have fully functional GUI.
- To have all the functions of keyboard and mouse operational.
- To access secondary storage device.
- To implement file system.
- To implement network system.
- Ability to handle multiple interrupts.
- Ability to handle multiple processes.

# 12. References/Bibliography


1. OSDev, OSDev, http://wiki.osdev.org/Main_Page [Accessed on Sep 10, 2020].
2. Nick Blundell, Writing Operating System from Scratch, 2010
3. Andrew S. Tanenbaum, 2008, Modern Operating Systems, Prentice Hall, Inc.
4. Wikipedia, X86, https://en.wikipedia.org/wiki/X86 [Accessed on Sep 20, 2020].
5. Wikipedia, GNU GRUB, https://en.wikipedia.org/wiki/GNU_GRUB [Accessed on Sep 21, 2020].
6. Wikipedia, VGA, https://en.wikipedia.org/wiki/Video_Graphics_Array [Accessed on Oct 11, 2020].
7. Brokenthorn, VGA, http://www.brokenthorn.com/Resources/OSDevVga.html [Accessed on Oct 11, 2020].
8. Wikipedia, Page (Computer Memory), https://en.wikipedia.org/wiki/Page_(computer_memory) [Accessed on Oct 5, 2020].
9. Erik Helin, Adam Renberg, The little book about OS development, https://littleosbook.github.io/ [Accessed on Oct 4, 2020].
10. IncludeHelp, https://www.includehelp.com/operating-systems/ [Accessed on 6, 2020].