# **BLOCKS OS**

# **An Engineering Project in Community Service**

## Phase – II Report

**Submitted by**

|  |  |  |
| --- | --- | --- |
| **SERIAL NO** | **NAME** | **REGISTRATION NUMBER** |
| 1 | V Surya Kumar | 19BCE10286 |
| 2 | Abhishek Srivastava | 19BCE10071 |
| 3 | Anjali Singh | 19BCG10003 |
| 4 | Pratul Maurya | 19BCY10036 |
| 5 | Pravir Kadian | 19BCE10006 |
| 6 | Saransh Pratap Singh | 19BCY10035 |
| 7 | Viplav Khubchandani | 19BAI10106 |
| 8 | C S Soujanya Mudliar | 19BCG10094 |

***in partial fulfilment of the requirements for the degree of***

***Bachelor of Engineering and Technology***

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

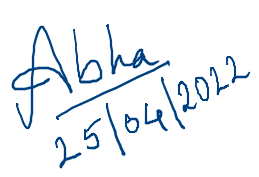
**21 April 2022**

****

**Bonafide Certificate**

Certified that this project report titled **“BLOCKS OS”** is the bonafide work of **19BCE10286 V Surya Kumar, 19BCE10071 Abhishek Srivastava, 19BCE10006 Pravir Kadian, 19BCG10003 Anjali Singh, 19BCY10036 Pratul Maurya, 19BCY10035 Saransh Pratap Singh, 19BAI10106** **Viplav Khubchandani, 19BCG10094 C.S. Soujanya Mudliar** who carried out the project work under my supervision.

This project report (Phase II) is submitted for the Project Viva-Voce examination held on 21st April 2022.



Dr. Abha Trivedi

**Supervisor**

**Comments & Signature (Reviewer 1)**

**Comments & Signature (Reviewer 2****)**

# 1. **INTRODUCTION**

Blocks-OS is an open-source operating system built from the ground up for educational purposes and to learn how an operating system works. End-users can use this project to learn more about how computers function by seeing how concepts like interrupt vectors, memory management, and segmentation are implemented on their own systems.

A project like this has several major advantages over a higher-level project that isolates students from the machine:

* Students gain a better understanding of the computer by seeing experiencing concepts like segmentation, interrupt vectors, and memory management in action.
* When students are forced to develop without the usual library services, they discover how much help they can obtain from operating systems.
* The ability to test their operating systems on their own machine helps to make the often abstract notions more concrete.
* Using the project as boiler-plate code and thorough documentation for reference, helps students pick any branch of the entire project and begin creating rest features or any individual feature.

## **1.1 Motivation**

Operating systems are created for a variety of purposes. Although each developer may have their own motives, some developers share the following:

* **Controlling the machine completely.**

When creating an application or other userspace programme, the developer must consider code written by others, such as the operating system, libraries, and other applications. It's a tremendous feeling to know that your code is the only one running on a machine.

* **Research.**

Many operating system developments begin as homework or research assignments. Typically, research works are carried out to improve existing operating systems.

* **Low-Level Programming.**

Because one must do everything, low-level programming is a fascinating and stimulating activity. This may appear to be more difficult, but it is also more enjoyable.

As a software developer, we should understand how systems work, how everything fits together, and how your programme works on the inside.

## **1.2** **Objective**

Theoretical aspects of operating systems are addressed in current operating system curriculum. Our initiative will aid the student developer community in learning about the technical aspects of the subject by documenting each step.

BLOCK OS is unlike any other existing project since most similar projects are inadequate, poorly documented, and lack community support. The following is how BLOCK OS is implemented:

* Our project was created particularly for students to understand and follow the general process of how an operating system is constructed from the ground up, what goes on behind the scenes of how operating systems work, and much more.
* So, in order to design an OS that any learner can follow, we'll need a lot of documentation that's well-organized and sequenced so that anyone who reads it can comprehend it quickly.
* Both the documentation and the implementation will be clearly outlined and organized into modules.

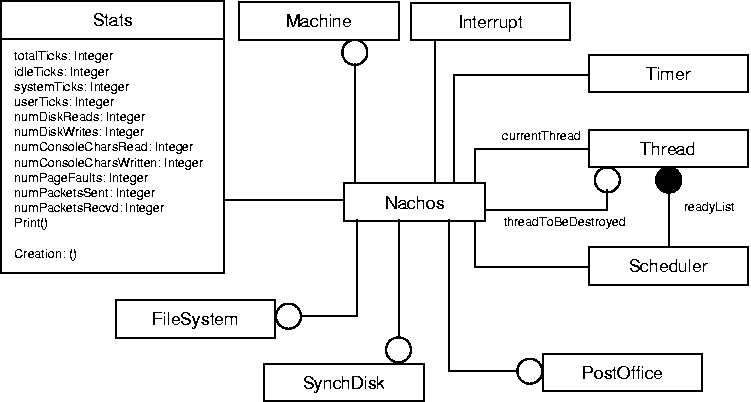
# **2.** **Existing Work / Literature Review**

On the internet, there are various articles and projects on operating systems, but in order to work on any OS-related project, you first should understand what makes them special, what they perform, and why they follow the architecture they do.

Nachos is a software that can be used to teach undergraduate and possibly graduate-level operating systems courses. It was designed by Thomas Anderson at the University of California, Berkeley, and is used by many colleges throughout the world.

Nachos is a user-process on a host operating system that was first created in C++ for MIPS. The code for any user programmes running on top of the Nachos operating system is executed by a MIPS simulator. Nachos code has been ported to a variety of architectures.

The Nachos system includes a variety of assignments in addition to the Nachos code. Nachos' purpose is to teach students to concepts in operating system design and implementation by incorporating major functionality within the system.



Pintos is a computer programme that serves as a foundation for a simple instructional operating system for the x86 instruction set architecture. It has kernel threads, user programme loading and execution, and a file system, but it does so in a very simplistic fashion. Ben Pfaff invented it in 2004 at Stanford University. It began as a substitute for Nachos, a comparable system created by Thomas E. Anderson at UC Berkeley, and was constructed in a similar manner. Pintos, like Nachos, is designed to introduce undergraduates to operating system design and implementation concepts by asking them to implement important components of a genuine operating system, such as thread and memory management, and file system access. Pintos also teaches pupils how to debug problems.

Although there are many more examples of such implementations of various types of operating systems, after analysing and studying a few of them, we came up with the idea of building our operating system with an initial design derived from pre-existing operating system architecture and the addition of some additional functionalities with the goal of learning and understanding how operating systems work and are implemented from the ground up.

# 

‘

# **3. Topic of the work**

## System Design / Architecture

The monolithic operating system is a very simple operating system in which the kernel controls file management, memory management, device management, and process management. The kernel will have access to all of the system's resources. Every component of the software package is housed within the kernel in monolithic systems. ​

A monolithic kernel is made up of two parts: kernel space and user space, which communicate with one another via IPC.

The following are some of the benefits of a monolithic kernel:

* Because functions like memory management, file management, and process scheduling are all implemented in the same address area, the monolithic kernel executes quickly.
* In the monolithic kernel, a process executes completely in a single address space.
* The monolithic kernel is a single binary file that is static.

## Working Principle

It is the operating system's software that makes it operate, and it works flawlessly in accordance with the resources available and protocols. Different components of the process are overseen, executed, and managed by the specific software.

**Boot Sector**

* The Boot Sector is the part of the hard drive where we put the programmes we want to run. It usually contains code to load the operating system that is installed on that machine. The code in the boot sector, however, is not the first to be performed.
* When we turn on our computer, it begins to copy some code called Bios from the ROM (Read Only Memory) to the main memory (RAM). The CPU then performs it.
* Boot sector by initialising first 510 byte of hard disk as 0 and then initialising last two bytes with value aa and 55 as shown in code below.

**Boot Loader**

* The boot loader is a component of the boot sector that loads the remaining operating system components. ​
* In a c file, we created a boot loader that loads 30 sectors into main memory.
* We introduced the START() method in our C programme and developed a linker file that automatically calculates where the entry function is present to execute this C file.

**Audio**

A speaker can have either "in" and "out" positions. A value of 1 causes the speaker to move to the "out" position, whereas a value of 0 causes it to move to the "in" position. If the speed of repetition is within the range of what the speaker can emit and what the human ear can hear, audible tones are produced. ​

In the extra.h file, we have implemented three functions:

· void scream(int fr); // passes the frequency of the sound that will be played

· void play(unsigned int fr ); // plays the sound​

· void stop( ); //stops the sound

The audio is played when a user enters the PLAY() command.

### **Video Graphics: Graphic Driver​**

Using the graphic driver, we built a variety of printf commands and colour changing functions, including:

### **Simple Video Player**

* Video is a term used to describe moving images that we perceive as videos. ​
* To provide us with a video, the computer continuously changes the data stored in the video memory, giving us the impression that it is moving or giving us a sense of "Video."
* Because our processor is so fast, it needs to keep the frame update rate low in order to keep up with the refresh rate of our monitor.

### **Secondary Memory**​

A hard disc, often known as a primary "secondary storage device," is a crucial component of a computer. Its capabilities are also utilised to maximise the usage of the primary storage device or RAM, in addition to storage. This is used in concepts like paging. We commonly use the in and out assembly commands when implementing a hard disc driver.

* There are several methods for reading and writing to a hard disc. The LBA (Linear Block Address) mode is what we've employed. This is the simplest approach to read/write to a hard disc; all we need is the sector's Block address. ​
* Passing 0 allows us to enter the first sector (The boot sector). Please note that writing to the 0th sector may cause your computer to become unbootable; however, you may always copy a boot loader to that sector. ​
* Earlier techniques that exposed the physical details of the storage device to the operating system software were replaced by the LBA scheme.

The cylinder-head-sector (CHS) scheme was the most common, in which blocks were addressed using a tuple that identified the cylinder, head, and sector where they existed on the hard disc.

CHS addresses can be converted to LBA addresses using the following formula:​

LBA = (( C x HPC ) + H ) x SPT + S - 1​

where,​

C, H and S are the cylinder number, the head number, and the sector number​

LBA is the logical block address​

HPC is the number of heads per cylinder​

SPT is the number of sectors per track

We devised two instructions, GET and PUT, to make the hard disc work in ATA in LBA mode. The write method is called by the PUT commands, and it writes a character array to the Hard Disk. The read technique was used to retrieve the array of characters from the Hard Disk by the GET command. ​

When we input and enter the PUT command, it first copies the value 0 to blockAddr, then sets the value 'J' in every cell in the At[] character array, and lastly adds a null character.

### **Keyboard Driver​**

The CPU will call a function that we have defined whenever a key is pushed. The CPU will not handle keyboard logic, but it will allow us to run code whenever a key is pressed. ​

* The CPU will call a function that we have defined whenever a key is pushed. The CPU will not handle keyboard logic, but it will allow us to run code whenever a key is pressed. ​
* To handle keyboard input, we'll need to write some code, send its address to the interrupt descriptor table, and tell the processor to load it
* It will call our keyboard handling code whenever a key is pressed after loading all of the parameters and the location of that code.
* When this interrupt occurs, we can try reading from the keyboard to see which key has been pushed. ​
* What keyboard gives as the value for pressed key is not ascii. It is named as scan codes.
* Modern keyboard drivers connect with the computer via USB interface, and while we are utilising PS/2 keyboards, USB keyboards will also operate because they simulate the older PS/2 keyboard.
* The PIC, or Programmable Interrupt Controller, is a computer chip that generates interrupts. When a key on the keyboard is pressed, the keyboard's chip instructs the computer's pic chip to issue an interrupt. ​
* The photo chip will then determine when the cpu should be notified of the interrupt. When the cpu receives the message that a key is being pressed, it performs the code that we previously instructed the cpu to run when a key is pushed.

## 

### **Programmable Interval Timer**​

* This is used to generate delays in the OS code, which can then be used by other components in procedural statements for delay control. ​
* An oscillator, a prescaler, and three separate frequency divisions make up the Programmable Interval Timer (PIT) chip (Intel 8253/8254). Each frequency divider includes an output that is used to control external circuitry by the timer. ​
* The PIT chip's oscillator operates at 1.193182 MHz (approximately).
* The following I/O ports are used by the PIT chip
* 0x40, 0x41, 0x42, 0x43
* The Timer IRQ works as follows: By using the PIT to generate a hardware interrupt every n milliseconds, we may make a basic timer. Decrement this variable until it reaches 0 every time the timer interrupt is called. Implementing the sleep function, which waits for the interval.

### **Global Constructors**

* Global constructors are similar to those found on global C++ objects, and they are designed to run before the main function. ​
* The constructors are in charge of reading command line arguments, initialising the standard library (memory allocation, signals, etc. ), performing global constructors, and eventually exiting (main(argc, argv)).
* This is something that should be done before calling the main method.
* ​Invoking the global constructors/destructors is as easy as traversing the array and running each element.

## Results and Discussion

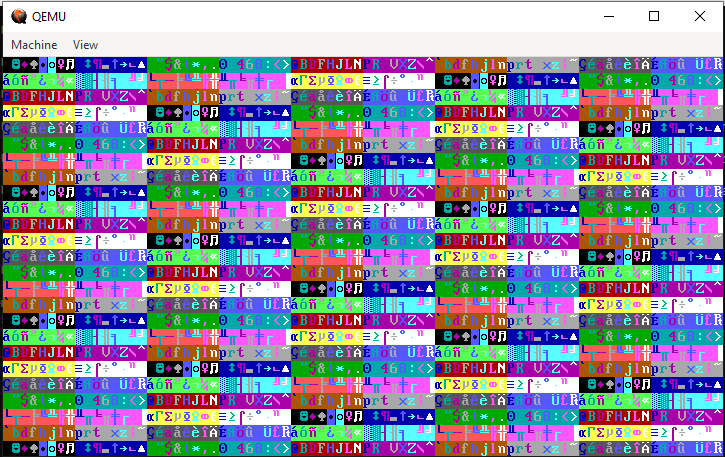
* We have successfully implemented the followingon our operating system from scratch:
  + Set the BIOS and kernel to their default settings.
  + For the booting procedure, there is a Boot Sector and a Boot Loader.
  + GDT implementation (Global descriptor table).
  + Change from 16-bit to 32-bit protected mode.
  + Support for C language has been added.
  + A Graphic Driver was implemented to display graphics on the screen. Also a simple video player was implemented to better facilitate our OS.
  + COLORA (for altering the colour of the screen monitor), VITBHOPAL (for displaying message on the screen), and other features have been added.
  + ISR has been implemented (Interrupt Descriptor Table).
  + Using ISR and PIC, create a keyboard driver.
  + ATA is used to write and read data from hard discs.
  + The PLAY() command can be used to provide audio capability.
  + Basic arithmetic operations
  + Paging on IA-32 architecture
  + Programmable Interval Timer
* Students can learn from the development phase and gain a greater understanding of the fundamentals and fundamental principles of operating system development by using version control during development and the documentation we developed recording the development phases.

**d)** **Individual Contribution**

* I researched and worked on the graphic drivers for our OS. The graphics driver is a program that controls how your graphic components work with the rest of our system. These drivers allow our Operating System to use the computer's graphics hardware.
* Even during the development phase, we need to print the values in variables to debug any issue arising. By inspecting every feature of Text Mode Video Graphics, it's difficult for us to print strings to the screen.
* As a result, I've used the graphic driver to build numerous print and colour changing features, such as the:
  + *cls* function that is used to clear the screen.
  + *set monitor color* is a command that changes the colour of the monitor's screen.
  + The *print string* function is used with an address of a null terminated character array to print string values on the screen.
  + *print char*, on the other hand, prints character values on the screen.
  + The scroll feature is implemented via the *scroll* function. We don't have to use the scroll() function ourselves. The printing functions will take care of it.
  + The *print color* string function accepts two arguments and prints a string on the screen with a specific background and foreground colour.
  + The *print color char* function prints char values of a specific background and foreground colour on the screen in a similar way.



* I implemented a simple Video Player that is able to show output of a video frame. By changing the data contained in the video memory continuously it gives us the feel of watching a “Video”. Since our processor is very fast when compared to our video card and monitor, therefore it will not be able to display every frame the processor generates.
* We solve this by minimizing the frame updating procedure depending on the refresh rate of our display hardware.



One Frame Of Output During Video Playing:

# **4. CONCLUSION**

* The primary goal behind the Block OS project was to produce an operating system for the student community and to demonstrate how an operating system is developed step by step.
* The software development waterfall model technique, which involves a sequential development process that flows like a waterfall across all aspects of the project, is the major method used to create this Operating system (like, analysis, design, development, and testing).
  + Designing - A roadmap is prepared to follow at each stage of the development process.
  + Development- We created a basic startup software in development.
  + Testing - Qemu, virtual computers, and cloud OS were used to test the basic programmes produced.
* This helps the students to follow along with the project development while implementing their own modules.gain a greater understanding of the fundamentals and fundamental principles of operating system development.

# **5. Reference**

1. <https://wiki.osdev.org/Main_Page>

2. <https://github.com/cfenollosa/os-tutorial>

3. <https://drive.google.com/file/d/1bUAbfE7OU6NjnyFwVGGkeHR11BPq1l32/view>

4. <https://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf>

5.  [CSE 221 - Graduate Operating Systems](https://cseweb.ucsd.edu/classes/sp00/cse221/projects.html)

# 

# **PLAGIARISM REPORT:**

**(Report is in 2 parts due to word limit 2000 in plagiarism checker).**

[**Plagiarism Report**](https://drive.google.com/drive/folders/1hts0anZCXElKe_F0icHKONDRSKNndHNJ?usp=sharing)