**BLOCKS OS**

**An Engineering Project in Community Service**

**Phase–II Report**

***Submitted by***

| **SERIAL NO** | **NAME** | **REGISTRATION NUMBER** |
| --- | --- | --- |
| 1 | Abhishek Srivastava | 19BCE10071 |
| 2 | V Surya Kumar | 19BCE10286 |
| 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 fulfillment of the requirements for the degree of***

***Bachelor of Engineering and Technology***

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

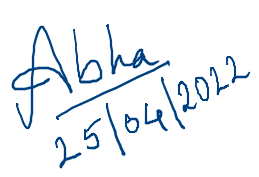
**21 April 2022**

****

**Bonafide Certificate**

Certified that this project report titled **“BLOCKS OS”** is the bonafide work of **19BCE10071 Abhishek Srivastava, 19BCE10286 V Surya Kumar, 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)**

|  |  |  |
| --- | --- | --- |
| **Serial Number** | **Topic** | **Page Number** |
| 1 | Introduction | 4 |
| 1.1 | Motivation | 4,5 |
| 1.2 | Objective | 5 |
| 2 | Existing Work | 5,6 |
| 3 | Topic of the Work | 6 |
| 3.1 | System Design / Architecture | 6,7 |
| 3.2 | Working Principle | 7,8 |
| 3.3 | Results and Discussion | 8,9,10 |
| 3.3.1 | My Contribution | 10,11,12,13 |
| 4 | Conclusion | 13 |

# 

# **1. INTRODUCTION**

Blocks-OS is an open-source operating system which is built for learning and educational purposes.

Users can use this project to learn what goes behind the scene when they use their operating system.

They will also be able to see the implementation of core operating system concepts like segmentation,

memory management.

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

* This project can be used to get a better understanding of the operating system.
* Users can see the capabilities and the limitations of the computer by seeing how it works under the hood.
* Users will be able to better appreciate the operating system as even the simplest command like moving a cursor takes quite some effort.
* Complex concepts like memory management, process synchronization will make more sense to the user.
* The extensive documentation of our project could be used at universities to make the operating systems course more dynamic and interesting.

An operating system plays a very important role by acting as a layer between user and the application,

making life easier for them.

## **1.1 Motivation**

Operating systems are designed to serve a range of functions. Despite the fact that each developer has their own motivations, the following are shared by many (if not all) of them:

* Controlling the machine completely.

When developing an application or other user-space programme, the programmer must take into account code written by others, such as the operating system, libraries, and other programmes. Knowing that your code is the only one running on a system is a fantastic feeling.

* While starting an operating system as a homework assignment in a pre-tertiary or first-year context is frequently regarded as a bad choice (because of tight deadlines), a long-term project is fine. The goal of most research studies is to improve existing operating systems. However, a common rookie mistake is underestimating the amount of time it takes to build an operating system from the bottom up.
* To find the better alternative if the operating systems that are currently available.
* They might not have a feature that the developer needs. Perhaps they're just bad people in general (Linux is bloated, Windows is unstable, etc.). It's possible that this is for profit, but any earnings will take a long time to appear.
* Low-level programming is a fun and stimulating activity because you have to do everything. This is more difficult (don't worry, it is), but it is also more enjoyable. You know how everything works, how everything fits together, and how your programme functions on the inside.

## **Objective**

The main objective of the project is to serve the student community and especially the students taking a course on operating systems at an undergraduate level.

Our project introduces a dynamic aspect to their course so the students are able to see the working of all the complex principles they are studying in their course.

By seeing how things work under the hood they would get more involved in the subject and hence this would make things interesting for them.

This project being an open source is also meant for people having interest in operating system development.

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

There are some operating systems projects built at the university level but all of them miss out on some of the important aspects of the working of operating systems. The once in full working condition lacks good documentation.

One such project is the Pint OS. It is a simple operating system framework for the 80x86 architecture. It supports kernel threads, loading and running user programs, and a file system, but it implements all of these in a very simple way. It was created at Stanford University by Ben Pfaff in 2004.It originated as a replacement for Not Another Completely Heuristic Operating System (Nachos), a similar system originally developed at UC Berkeley by Thomas E. Anderson, and was designed along similar lines. Like Nachos, Pintos is intended to introduce undergraduates to concepts in operating system design and implementation by requiring them to implement significant portions of a real operating system, including thread and memory management and file system access.Unlike Nachos, Pintos can run on actual x86 hardware, though it is often run atop an x86 emulator, such as Bochs or QEMU. Nachos, by contrast, runs as a user process on a host operating system, and targets the MIPS architecture.

Wiki OS Dev is a website with around 693 articles and it also serves as a strong community for those interested in operating systems development but solely depending on it serves no purpose as it has some of the major functionalities missing.

After studying and reviewing the existing project work we came to a conclusion that all of them miss out on a common thing and that is documentation so we decided to build BLOCKS-OS to serve the student community with detailed documentation of each and every line of code written in the implementation of our operating system.

# **3. The topic of the work**

# **3.1) System Design / Architecture**

The monolithic software package could be a simple one in which the kernel directly controls file, memory, device, and method management.

All of the system's resources will be accessible to the kernel. In monolithic systems, the kernel contains every component of the software package.

A monolithic kernel is a software package that encapsulates the whole software package within the kernel section.

**About Monolithic System Architecture:**

A monolithic kernel is a type of operating system in which the complete operating system runs in kernel space. Other operating system designs (such as the microkernel architecture) differ in that the monolithic model is the only one that defines a high-level virtual interface over computer hardware. All operating system services such as process management, concurrency, and memory management are implemented using a set of primitives or system calls. Modules for device drivers can be added to the kernel.

Most modern monolithic operating systems, such as OpenVMS, Linux, BSD, SunOS, AIX, and MULTICS, may dynamically load (and unload) executable modules at runtime, as can modular operating systems like OS-9.

The operating system's modularity is at the binary (image) level, not at the architecture level. Modular monolithic operating systems should not be confused with the architectural level of modularity found in server-client operating systems (and their variations, which are frequently marketed as hybrid kernels) that use microkernels and servers .

**Monolithic System Characteristics -:**

Users can take advantage of the following functionalities provided by the monolithic operating system:

* This sort of operating system has a straightforward structure. The kernel contains all of the components required for processing.
* It is superior for performing smaller jobs because it can handle minimal resources.
* Communication between components: All of the components, as well as the kernel, can communicate directly with one another.
* Operating system that is quick: The code for creating a monolithic kernel is extremely fast and reliable.

**References of monolithic os content:**

* https://www.javatpoint.com/monolithic-structure-of-operating-system
* https://www.tutorialspoint.com/monolithic-system-architecture

**3.2) Working Principle**

It is the operating system's software that allows it to function, and it does so flawlessly in accordance with the resources and protocols available. The specialized software oversees, executes, and manages various aspects of the process.

**Handling interrupts**

A modern CPU allows code to run in several processor modes, each of which grants varying access privileges to the system. When the CPU is in user mode, most programmes run. The CPU will refuse to execute any instructions for communicating directly with a device in user mode, for example, because such access could allow the programme to access data it shouldn't. The CPU also limits memory accesses in user mode to the limited portion of memory that is really dedicated to the present programme. The operating system, on the other hand, runs in supervisor mode, allowing direct connection with devices and access to all memory addresses.

**System Calls**

A system call is a request from a user program to the operating system asking the operating system to undertake some task on the user program's behalf. A request to open a file, run another programme, send a message to another computer, or display a line on the screen are all examples of system calls in a standard operating system.

**Library Functions**

When we write a C programme, the system calls appear to be standard function calls.

Printf(), for example, is a library function. This implies it's provided in a library for the compiler to use, but unlike a system call, it's not part of the operating system. When the compiler generates the programme, it looks for and includes any library functions that it utilizes in the executable file. As a result, printf() is a user application rather than an operating system function.

**3.3) Results and Discussion**

Blocks-OS is a completely free and open-source operating system. Our goal was to demonstrate how our operating system may be utilized as a teaching tool to help students better understand how it works.

By employing version control throughout development, students can learn from the development phase and obtain a better understanding of the fundamentals and essential principles of operating system development.

**Here are the features or functionality that we have implemented in Blocks OS:**

* We looked into different OS components and created a build script that compiles and assembles source code into object files and connects them together to create binary executables.
* We created the keyboard driver and associated assembly module, which loads the IDT into memory and allows the system to accept keyboard interrupts.
* We created a boot sector using a magic number and a boot loader using C, which resulted in our operating system being successfully loaded into main memory.
* We've also worked on transitioning our operating system from 16-bit to 32-bit mode, or protected mode. As a result, we were able to write our code in C rather than assembly language after switching to protected mode.
* We also used the Flat memory model to create GDT, which allowed us to describe each segment of our operating system in contiguous memory addresses.
* To handle the interruptions generated by the PIC chip and to receive and store the scan codes from the console into the main memory, we created an Interrupt service routine for the keyboard.
* We used C, the data structure used by the x86 architecture, to implement the Interrupt descriptor table in the kernel. The CPU consults the Interrupt descriptor table to determine the proper response to interrupts and exceptions.
* Hardware interrupts, software interrupts, and processor exceptions, all of which are referred to as interrupts, are the three sorts of events that cause the IDT to be used. It was created for hardware interrupts.
* We've also been working on the converter, which will be used to transform scan codes from the keyboard into readable ASCII values.
* Our major storage device, or hard drive, was created by us. In most cases, we employ the in and out assembly commands to create a hard disc driver.
* We used ATA (Advanced Technology Architecture). Hard disks and CD-ROMs can be attached to the motherboard internally and perform input/output operations.
* On the IA-32 Architecture, we have built a simple version of Paging.
* To integrate these features in our operating system, we examined numerous operating system-related issues such as how the boot sector works, what the magic number is, GDT, and so on.
* The flow of the processes used to create this OS has been documented.
* We've also worked on the Literature survey, which entails investigating previously created operating systems in order to compare and contrast their capabilities.
* We investigated several OS components and developed a build script that compiles and assembles source code into object files before connecting them to produce binary executables.
* The keyboard driver and related assembly module that loads the IDT into memory and allows the system to accept keyboard interrupts were developed by us.
* We investigated and developed graphic drivers for our operating system. These drivers enable our operating system to access the graphics hardware on the computer.
* Audio support has been implemented. It is activated by entering a command into the console.
* Global constructors have been supported by us.
* A Programmable Interval Timer has been implemented.
* A basic video player has been implemented.
* Using the graphic drivers, we created numerous print, clean screen, and color change operations to show texts.

**My Contribution**

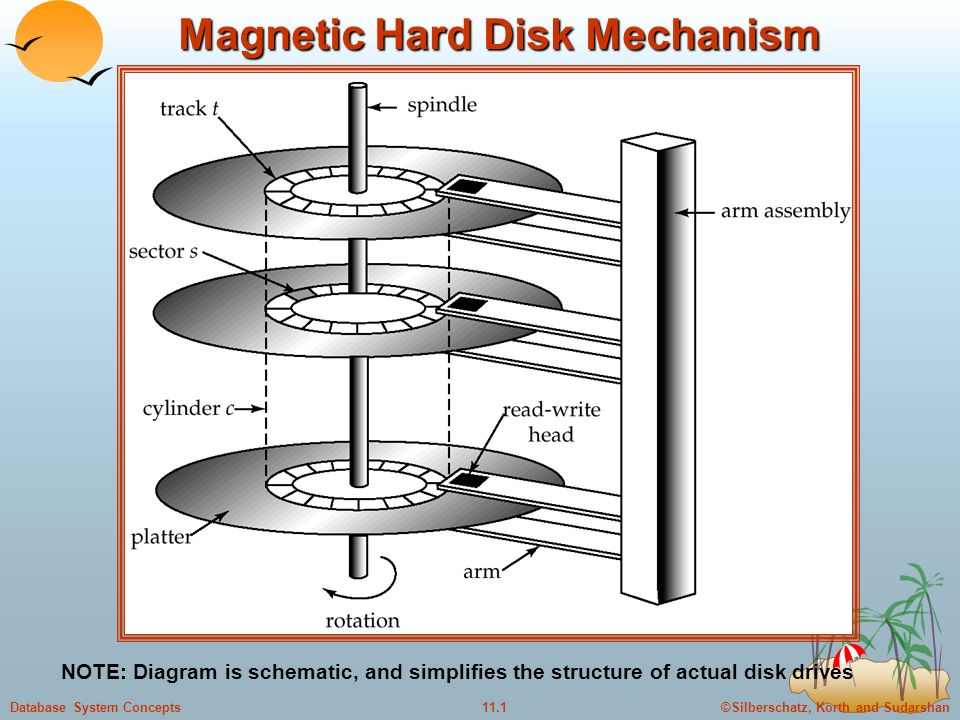
**Secondary Memory**

**a) System Design / Architecture**

Our primary "Secondary Storage Device" is a hard disc. In most cases, we employ the in and out assembly commands to create a hard disc driver. Hard discs are divided into two categories: HDD and SSD. We use a hard disc drive for our operating system.

**Structure of HDD**

In a hard disc drive (HDD), data is stored on platters in magnetic form. Hard drives typically have one to four platters stacked together. To read and write data, each of these platters will have its own Head. Tracks and sectors are further separated into these platters. On a hard drive, there are several tracks. There will be a set of sectors in each of these tracks. A sector is typically 512 bytes in size. The cylinder is a term used to describe a hard disc.

****

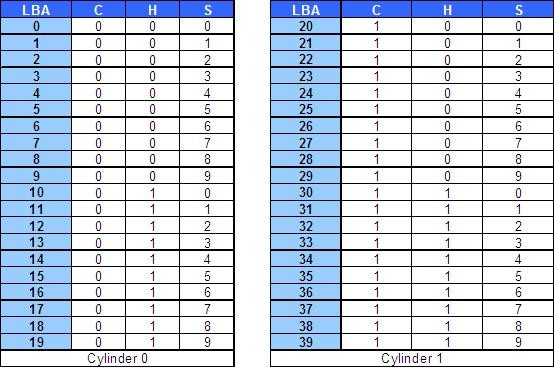
**b) Working Principle**

For the ATA technology, we created a hard disc driver. Hard drives and CD-ROMs can be internally attached to the motherboard and perform input/output operations thanks to ATA, or Advanced Technology Attachment.



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.

****

Chief among these was the cylinder-head-sector (CHS) scheme, where blocks were addressed by means of a tuple that defined the cylinder, head, and sector at which they appeared on the hard disk.

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

**c) Results and Discussion**

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. Then it calls the put() function, which in turn invokes the write function.

When we use the GET command, the same thing happens. It is in charge of the reading process. Then we used the get function, which runs the read function in the assembly file, and copied the value 0 to the blockAddr variable. That function returns to the C code after reading the 0th sector into the At[] array, and then outputs the contents of that array.

# **4. CONCLUSION**

Our project is an open-source operating system created solely to assist the student community. Monolithic System Architecture is the architecture that is used. The software development waterfall model approach is the major method used in the creation of this operating system. The goal of this project is to provide a learning tool that will allow students to have a better knowledge of how OS operates inside by giving them hands-on experience with ideas that they have only studied conceptually.

​**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:**

<https://drive.google.com/file/d/1G1_x78gY5Ukp6PHaSDc-t6czvzf-zdqg/view?usp=sharing>