**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 fulfilment of the requirements for the degree of***

***Bachelor of Engineering and Technology***

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

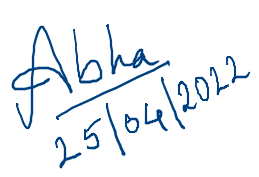
**21 April 2022**

****

**Bonafide Certificate**

Certified that this project report titled **“BLOCKS OS”** is the bonafide work of **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 | 5 |
| 1.2 | Objective | 5 |
| 2 | Existing Work | 6 |
| 3 | Topic of the Work | 7 |
| 3.1 | System Design / Architecture | 7 |
| 3.2 | Working Principle | 9 |
| 3.3 | Results and Discussion | 11 |
| 3.3.1 | My Contribution | 14 |
| 4 | Conclusion | 19 |
| 5 | Reference | 19 |
| 6 | Plagiarism Report | 19 |

# 

# 

# **1. INTRODUCTION**

Blocks-OS is an open-source operating system (built from the ground up) designed solely for educational purposes or to learn how an operating system works. End-users can use this project to learn more about how computers function by seeing how principles like segmentation, interrupt vectors, and memory management 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:

* The project can be worked on by a user to obtain a better grasp of the computer. ​
* Students get a better grasp of the computer by seeing concepts like segmentation, interrupt vectors, and memory management in action.
* When students are forced to programme 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, students can pick any branch of the entire project and begin implementing the rest of the features or a specific feature.

As the computer executes the application programmes of users, the OS oversees the CPU's operations, controls the input/output and storage functions of the computer system, and provides other support services. ​

## **1.1 Motivation**

Operating systems are created for a variety of purposes. Although each developer has their own motives, some (if not all) developers share the following:

* Having complete control over the machine.

When creating an application or other user space 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. While beginning an operating system as a homework assignment in a pre-tertiary or first-year setting is often considered a terrible idea (because to limited deadlines), a long-term project is perfectly acceptable. Typically, research studies are carried out to improve existing operating systems. However, a common novice error is underestimating the time required to create an operating system from the ground up.

* To replace the currently available operating systems.

Perhaps they don't have a feature that the developer need. Maybe they're just bad in general (Linux is bloated, Windows is unstable, etc.). This could be for profit, but any profits will most certainly take a long time to materialise.

* Because it's fun.

Because you have to do everything, low-level programming is a joyful and stimulating activity. This may appear more challenging (it is, don't worry), but it is also more enjoyable. You understand how everything works, how it all fits together, and how your programme operates on the inside.

## **Objective**

The current operating systems curriculum emphasizes the theoretical side of the subject. Our effort will aid the student developer community in understanding the practical element of the subject by properly documenting each step. ​

Apart from that, our BLOCK OS is distinct from previous initiatives in that previous projects are frequently incomplete, poorly documented, and lack community support. ​

Our project is designed exclusively for students to study and follow the general process of how an operating system is created from the ground up, as well as what goes on behind the scenes in the operation of operating systems. ​

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 defined and divided 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 must first understand what makes them special, what they perform, and why they follow the architecture they do.

NachOS is an example of an operating system developed by the developer community or by developers, and it is used to implement instructional operating systems. It's a standard UNIX process. Threads and remote procedure calls, as well as current hardware advancements like RISCs and the predominance of memory structures, make implementing functions simple. It also uses protocol, which is a prominent design strategy.

Furthermore, advanced software and design approaches such as protocol layering and object-oriented programming are available.

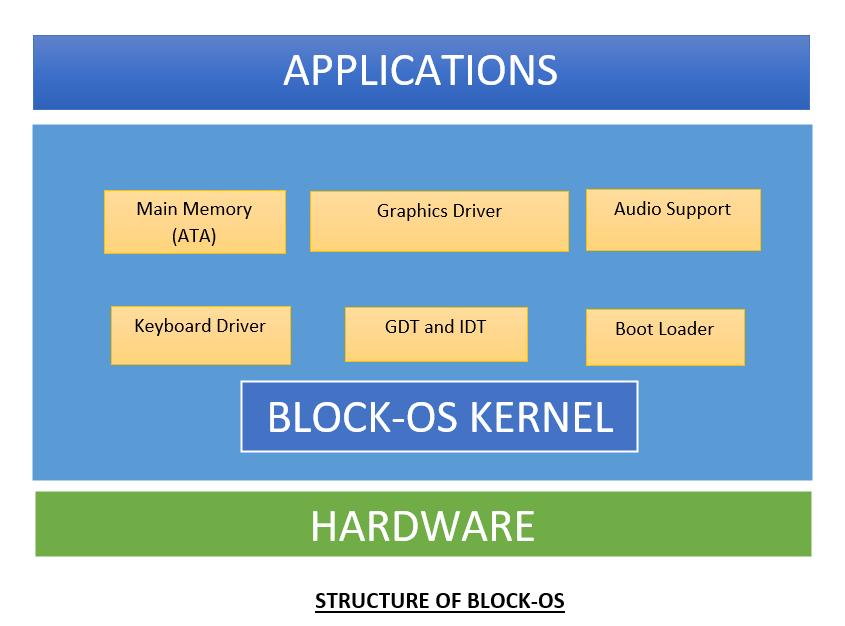
PintOS, whose structure and shape are inspired by the NachOS educational operating system from the University of California, Berkeley, is another example of an operating system. PintOS is a simple operating system framework for the 80x86 architecture, written in C like other operating systems. At a very basic level, it implements several capabilities such as kernel threads support, loading and running user programmes, and a file system. PintOS can be run in a system simulator with ease.

PintOS began as a replacement for NachOS, which had a striking visual resemblance to it. PintOS differs from NachOS in two important ways. PintOS, for example, runs on real or simulated 80x86 hardware, whereas NachOS runs as a process on the host operating system. Second, PintOS is developed in C++, whereas NachOS is written in C, unlike most real-world operating systems.

Wiki OS dev is a website with 693 wiki articles that gives information on the creation of operating systems and serves as a community for those interested in OS development. However, when it comes to real implementation, it's a little difficult to rely solely on it because it's more of a compressed guide with an overview of 1-2 functionalities for each.

Although there are many more examples of such implementations of various types of operating systems, after reviewing 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. The topic of the work**

****

# **3.1) System Design / Architecture**

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

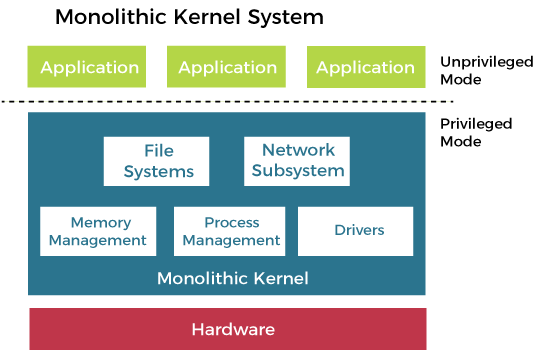
The kernel will have access to all of the system's resources. Every component of the software package is housed within the kernel in monolithic systems.

A monolithic kernel is a software package design in which the entire software package is contained within the kernel region.

**Monolithic System Architecture:**

The particular nature of the software package is not accommodated in a monolithic approach of software package architecture. Despite the fact that the planning adheres to the separation of concerns, no attempt is made to limit the privileges supplied to the various components of the software package. The software programme runs with the highest level of privileges. The monolithic software package's communication overhead is the same as the other package's, which is considered to be negligible.

**Monolithic Kernel System diagram:**



**Features of Monolithic System:**

The monolithic package has a simple structure, and all of the necessary components are included in the kernel. The monolithic kernel's code is lightning fast and tremendously powerful. Because it can handle less resources, it works better for performing arts modest jobs. All of the components will communicate directly with each other as well as with the kernel.

**Advantages of Monolithic Kernel:**

* The monolithic kernel has a fast execution time.
* Memory management, file management, method programming, and other services are enforced in a monolithic kernel under the same address space.
* A method in a monolithic kernel operates entirely in a single address space.
* The monolithic kernel is a single binary file that is static.

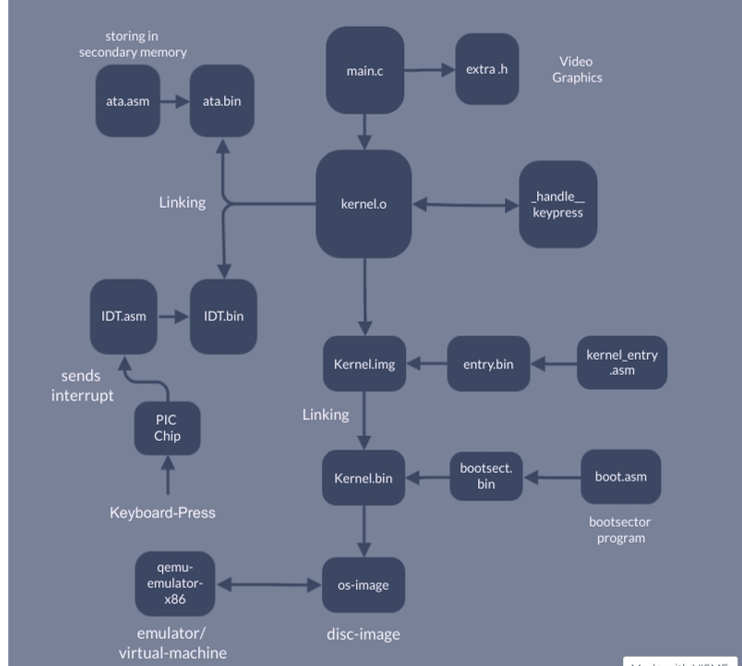
**Disadvantages of Monolithic Kernel:**

* If any service within the monolithic kernel fails, the entire OS will fail.
* To install a new service, the operating system must be upgraded.

**References of monolithic os content:**

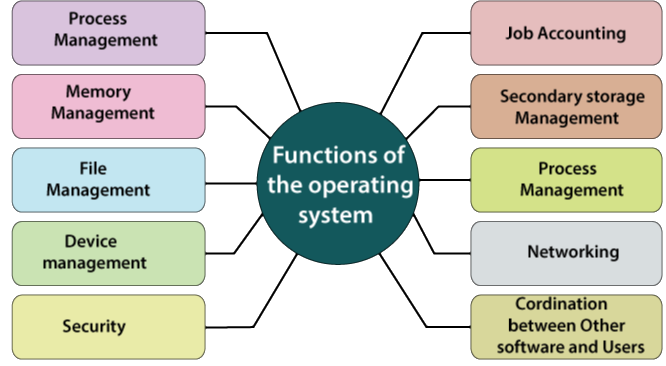
* https://www.javatpoint.com/monolithic-structure-of-operating-system
* https://www.tutorialspoint.com/monolithic-system-architecture

**3.2) Working Principle**

****

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

* Process management
* Memory management
* File management
* Device management
* I/O System management
* Security
* Secondary storage management
* Command interpretation
* Networking
* Job accounting

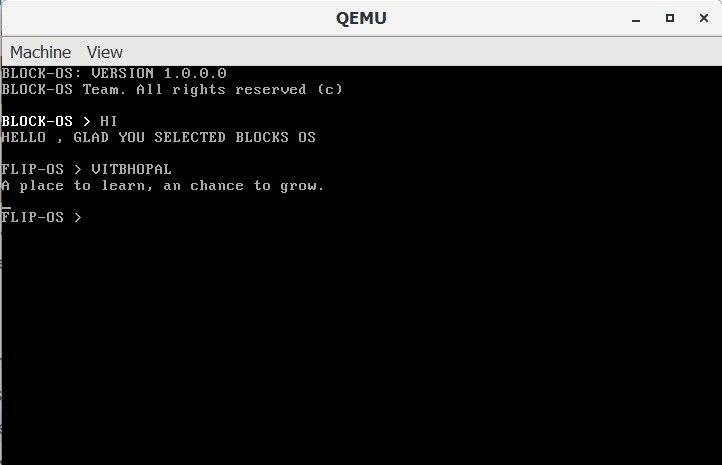


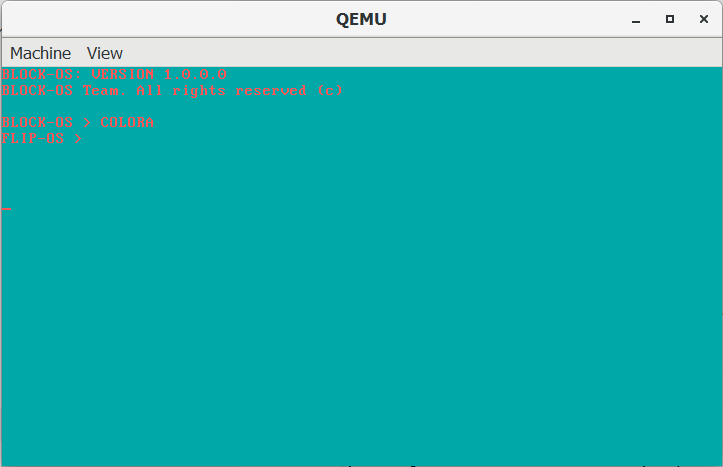
**3.3) Results and Discussion**

Blocks-OS is an open-source operating system built from the ground up. Our goal was to show how our operating system may be used as a teaching tool for students to gain a better knowledge of how it operates.

Students can learn from the development phase and gain a greater understanding of the fundamentals and essential principles of operating system development by using version control during development.

**Blocks OS Screenshots**





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

* We have explored various OS modules and developed the build script which compiles and assembles the source code to object files and links them to create binary executables.
* We have developed the keyboard driver and respective assembly module, which loads the IDT into the memory and enables the system to take interrupts from the keyboard.
* We have implemented an Interrupt service routine for the keyboard to handle the interrupts generated by the PIC chip and to read and store the scan codes from the console into the main memory.
* We implemented the Interrupt descriptor table in the kernel using C, the data structure used by the x86 architecture. The Interrupt descriptor table is used by the processor to determine the correct response to interrupts and exceptions.
* Use of the IDT is triggered by three types of events: hardware interrupts, software interrupts, and processor exceptions, which together are referred to as interrupts. We defined it for the hardware interrupts.
* We have also worked on the converter, which will be used to convert the scan codes from the keyboard to ASCII values that are readable.
* We designed our primary storage device or hard disk. When implementing a hard disk driver, we usually use the in and out assembly commands.
* We made use of ATA or Advanced Technology Attachment allows hard disks and CD-ROMs to be internally connected to the motherboard and perform input/output functions.
* We have implemented a basic version of Paging on IA-32 Architecture
* We have researched various operating system related topics like how the boot sector works, what is the magic number, GDT, etc to implement these all features in our operating system.
* We have implemented a boot sector using a magic number and a boot loader using C which resulted in successfully loading our operating system to the main memory.
* We have also worked on implementing switching of our OS from 16-bit mode to 32-bit mode i.e. protected mode. As a result, switching to protected mode allowed us to write our code in C rather than writing it in assembly language.
* We also implemented GDT using the Flat memory model that allowed us to define each segment of our OS in contiguous memory locations.
* We researched and worked on the graphic drivers for our OS. These drivers allow our Operating System to use the computer's graphics hardware.
* We have implemented audio support. It can be triggered by typing a command in the console.
* We have provided support for Global constructors.
* We have implemented a Programmable Interval Timer
* We have implemented a basic video player
* We have implemented various print, clear screen and colour change functions to display texts using the graphic drivers.
* We have documented the flow of the processes followed while building this OS.
* We have also worked on the Literature survey, which includes researching the previously built OS, so as to compare and analyze their functionalities.

**My Contribution**

**a) Video Player**

Video is a term that refers to moving images. When we encounter many photographs in a short period of time, our brain interprets them as moving or creates the illusion of "Video."

Poking the Video Memory is what the computer does to show us a video. This Feel is created by continuously changing the data stored in the video memory. However, the processor is far faster than our video card and display. As a result, it won't be able to display every frame the processor produces in a timely manner.

As a result, depending on the refresh rate of the monitor, the solution is to reduce the frame updating operation. However, the computer must be capable of playing 60 frames per second. The human eye can see 60 frames per second. As a result, producing video at a frame rate of more than 60 frames per second is pointless.

int start()

{

char \*TM\_START = (char \*)0xb8000;

int i;

char obj = 0;

while (1)

{

i = 0;

while (i < (2 \* 80 \* 25))

{

\*(TM\_START + i) = obj;

i++;

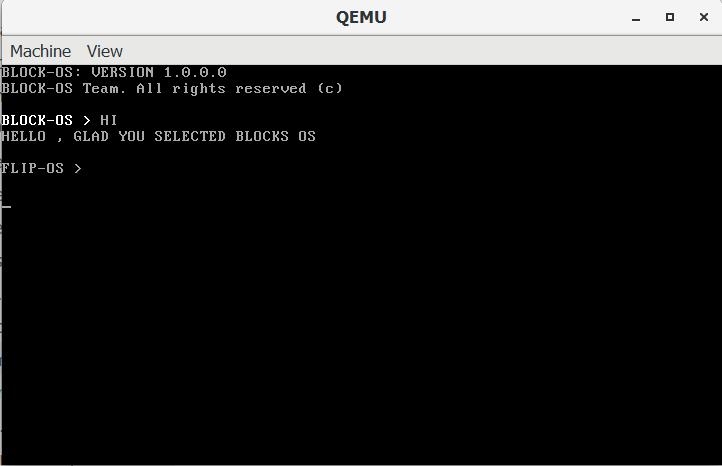
obj++;

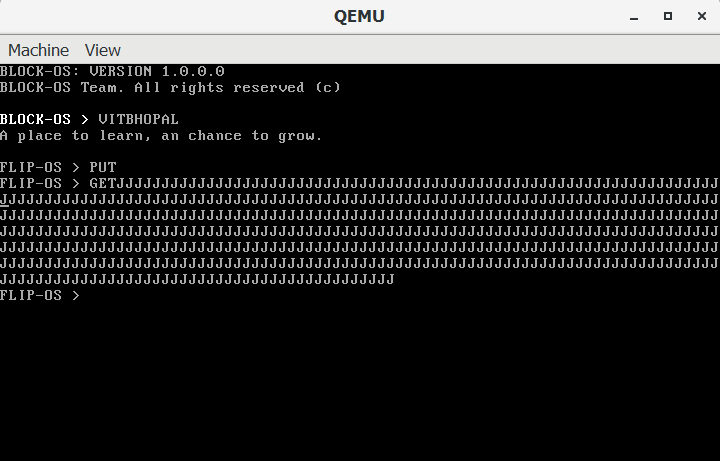
}

}

}

**b) Secondary Memory**



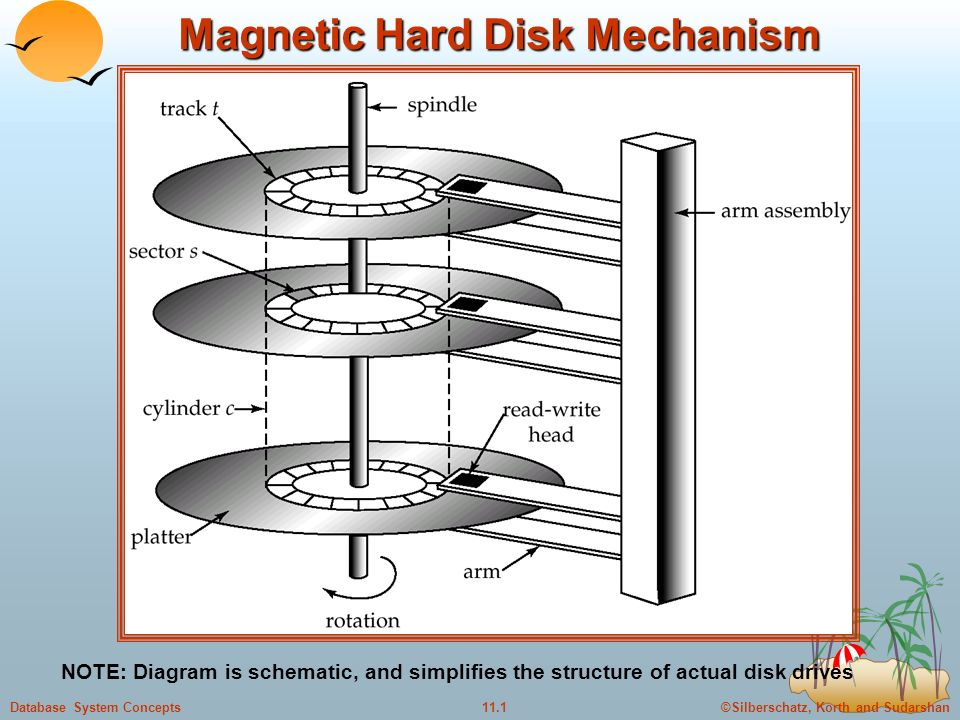


**a) System Design / Architecture**

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 phrase used to describe a hard disc.

****

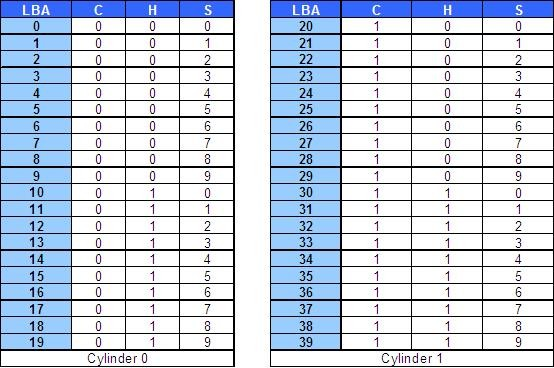
**b) Working Principle**

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.

****

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.

The following formula can be used to convert CHS addresses to LBA addresses:

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 reading 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:**

* <https://wiki.osdev.org/Main_Page>
* <https://github.com/cfenollosa/os-tutorial>
* <https://drive.google.com/file/d/1bUAbfE7OU6NjnyFwVGGkeHR11BPq1l32/view>
* <https://www.cs.bham.ac.uk/~exr/lectures/opsys/10_11/lectures/os-dev.pdf>
* [CSE 221 - Graduate Operating Systems](https://cseweb.ucsd.edu/classes/sp00/cse221/projects.html)

**6. Plagiarism Report**

* [Epics Plagiarism Report - 19BCE10071](https://drive.google.com/drive/folders/1dnMmdkMBGfKKg0J8S-gqKBGmyfn1pH4m?usp=sharing)