# **BLOCKS OS**

# **An Engineering Project in Community Service**

## Phase – II Report

**Submitted by**

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

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

**Bachelor of Engineering and Technology**

****

**VIT Bhopal University**

**Bhopal**

**Madhya Pradesh**

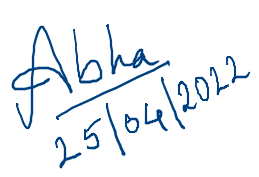
**21 April 2022**

****

**Bonafide Certificate**

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

**Index**

|  |  |
| --- | --- |
| **Serial Number** | **Topic** |
| 1 | Introduction |
| 1.1 | Motivation |
| 1.2 | Objective |
| 2 | Existing Work |
| 3 | Topic of the Work |
| 3.1 | System Design / Architecture |
| 3.2 | Working Principle |
| 3.3 | Results and Discussion |
| 3.3.1 | My Contribution |
| 4 | Conclusion |
| 5 | Reference |
| 6 | Plagiarism Report |

# **INTRODUCTION**

Blocks-OS is an open-source operating system made from scratch to understand how an Operating System works and focuses purely on educational purposes. End-users can use this project to learn more about how computers work by experiencing first-hand how concepts such as segmentation, interrupt-vectors, and memory management actually manifest themselves on their own computers.

Advantages of creating such a project is that

* It gives students a hands-on experience while learning an about Operating System
* By working on concepts like interrupt vectors, segmentation, students gain a deeper understanding of the computer itself.

## **1.1 Motivation**

There are varied reasons which motivate a person to develop his/her own Operating System. Some of the motivations for me to work on this project was:

* One of the main reasons for having an operating system with one is to manage the device's hardware and software resources, this made me explore more and come up with one of our own OS.
* Since we made it from scratch, therefore documenting it for the student community to use and learn was one of the driving forces.

**1.2 Objective**

The Current operating system available right now emphasizes more on the theoretical side of the subject. By documenting each step, our project will help the student developer community learn about the practical aspects of the subject.

Our project, i.e, BLOCK OS, is different from other pre-existing projects as existing projects are mostly incomplete, poorly documented, and with no community support. BLOCK OS is implemented in the following way:

* Our project is developed specifically for the student community to learn and follow the overall process of how an OS is developed from scratch, what goes behind the scenes of the working of operating systems, and a lot more.
* Documentation, as well as implementation, will be well defined and divided into modules.

# **2. LITERATURE REVIEW / EXISTING WORK**

There are numerous papers and projects on operating systems on the internet, but in order to work on any OS-related project, it is necessary to know what makes them unique, what they do, and what makes them follow the particular architecture.

One of the instructional operating systems is NachOS, an example of operating systems developed by developer community or by developers. It runs as a regular UNIX process. Implementing functionalities is easy for the same, like threads and remote procedure calls, recent hardware advances, such as RISC's, and the prevalence of memory hierarchies. Moreover, it implements protocol - a popular design technique.

Another example of an operating system is PintOS whose structure and form are inspired by the NachOS instructional operating system from the University of California, Berkeley. PintOS is written in C language much like other operating systems and is a simple operating system framework for the 80x86 architecture. It implements various functionalities like kernel threads support, loading and running user programs, and a file system at a very simple and basic level. PintOS can be easily run in a system simulator.

PintOS began as a substitute for NachOS, which had a strong similarity in design with it. PintOS is distinct from NachOS in two key aspects. To begin with, PintOS runs on real or simulated 80x86 hardware, whereas NachOS runs as a process on the host operating system. Second, unlike most real-world operating systems, PintOS is written in C++, whereas NachOS is written in C.

Wiki OS dev is a website that provides information about the creation of operating systems and serves as a community for those people interested in OS development with 693 wiki articles.

Although there are a lot more examples of such implementations of different types of operating systems, and after reviewing and studying some of them we have come up with an idea to build our operating system with the initial design derived from pre-existing operating system architecture with the implementation of some additional functionalities with our aim to learn and understand about how operating system works and is implemented from scratch.

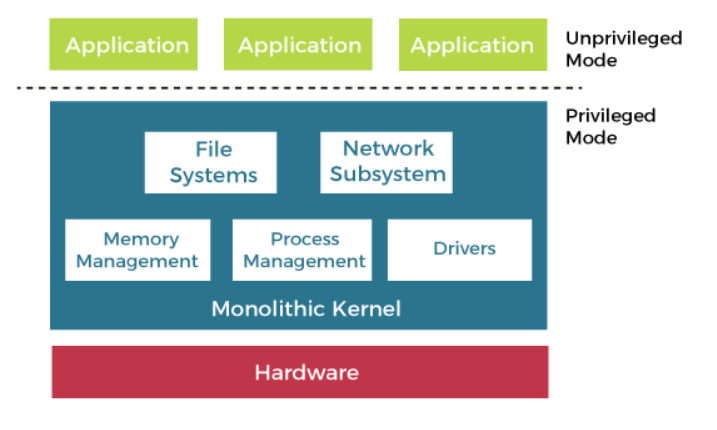
# **3. THE TOPIC OF THE WORK**

**3.1 System Design / Architecture**

The system architecture that has been followed while developing BLOCK-OS is Monolithic Architecture.

* The monolithic operating system is a basic software package within which file management, memory management, device management, and method management are directly controlled inside the kernel.​
* The kernel will access all the resources given within the system. In monolithic systems, every element of the software package is contained inside the kernel.​
* Monolithic kernel has two parts kernel space and user space, both parts communicate with each other through IPC.​
* One of the major advantages of having a monolithic kernel is that it provides memory management, CPU scheduling, file management, and other operating system functions through system calls.​

**Monolithic Kernel System diagram:**



**Features of Monolithic Architecture:**

* The code for the monolithic kernel is incredibly quick and strong.
* It works higher for performing arts smaller tasks because it will handle restricted resources.

**Advantage of Monolithic Architecture:**

* The execution time of the monolithic kernel is quick.
* In a monolithic kernel, a method runs fully in a very single address space.

**Disadvantages of Monolithic Architecture:**

* The entire OS fails if any service within the monolithic kernel fails.
* To add any new service the endure OS has to be changed.

**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 software of the operating system that makes it work and works perfectly in compliance with the available resources and protocols. The process involves different aspects that are overseen, executed and governed by the specific software.

Process management

Memory management

File management

Device management

I/O System management

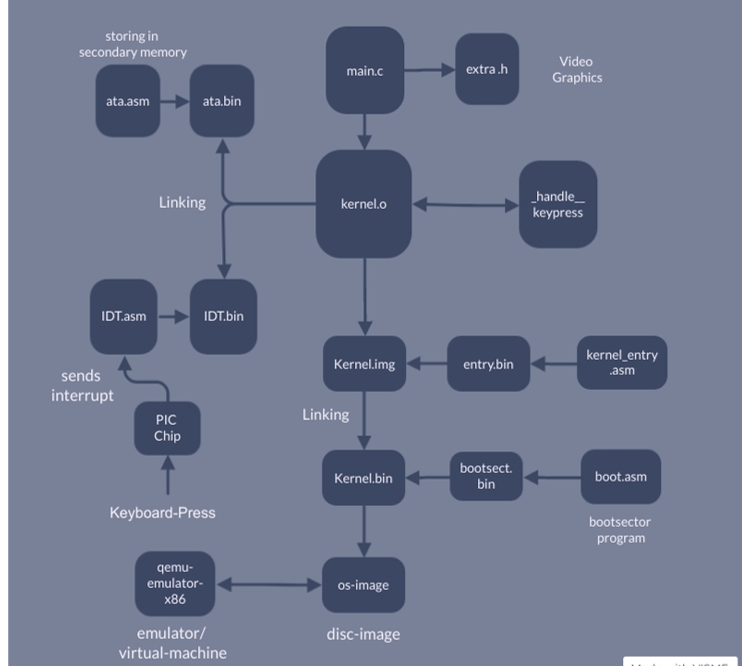
Secondary storage management

Command interpretation

Networking

Job accounting

****

****

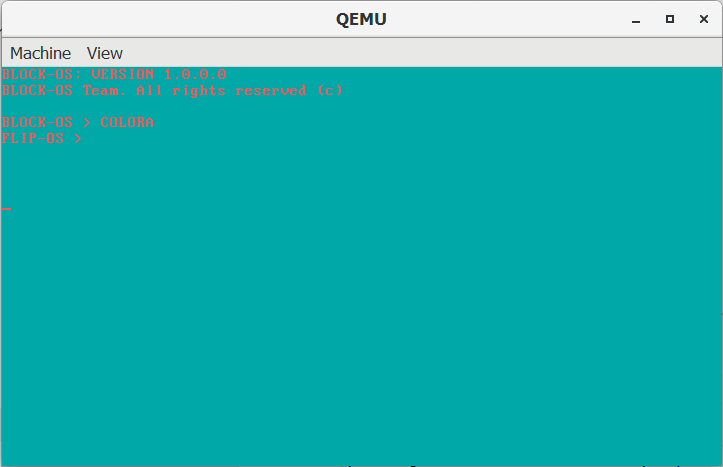
**3.3 Results and Discussion:**

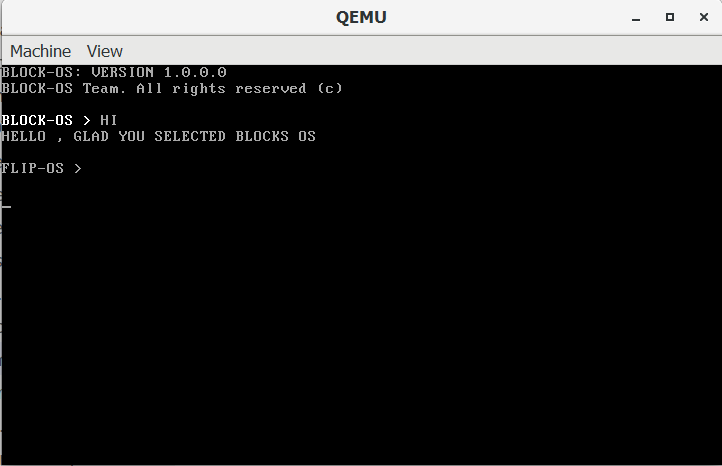
Block OS is an Operating System made from scratch keeping in mind it as a teaching tool for the student community to learn and explore OS and gain a better and deeper knowledge of it.

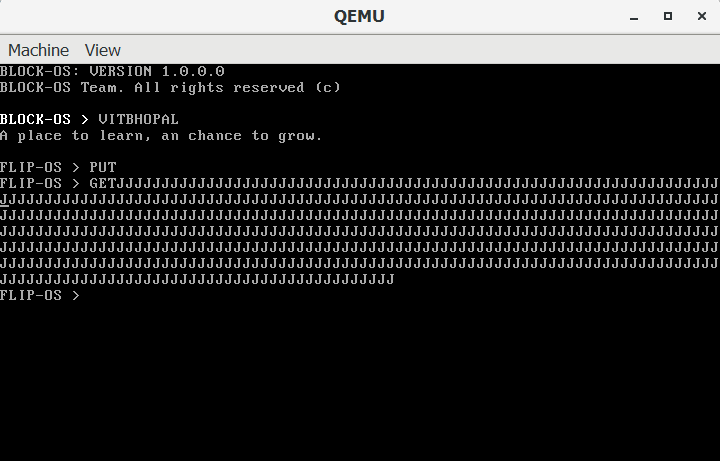
A descriptive documentation has been developed to be referred to while working around the Operating system for the student community. It provides step-wise implementation to follow.

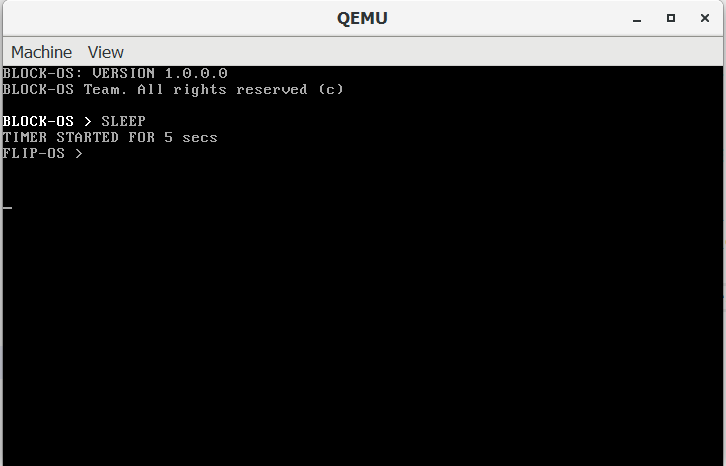
**FEATURES AND FUNCTIONALITY THAT HAS BEEN IMPLEMENTED IN BLOCK OS:**

* First, after exploring all the various OD modules, a build script was developed which compiles and assembles the source code to object files and links them to create binary executables.
* Second, the keyboard driver and respective assembly module, which loads the IDT into the memory and enables the system to take interrupts from the keyboard, was implemented.
* Next, an Interrupt service routine for the keyboard is implemented to handle the interrupts.
* The Interrupt descriptor table in the kernel using C, the data structure used by the x86 architecture is also implemented. The Interrupt descriptor table is used by the processor to determine the correct response to interrupts and exceptions.
* For hardware interrupts, IDT is triggered by three types of events: hardware interrupts, software interrupts, and processor exceptions.
* To convert the scan codes from the keyboard to ASCII values that are readable, we worked on the converter.
* To define each of our OS segment in contiguous memory locations, we implemented GDT using the Flat memory model.
* We have implemented a basic version of Paging on IA-32 Architecture
* We have implemented audio support. It can be triggered by typing a command in the console.
* We have provided support for Global constructors.
* We also implemented a Programmable Interval Timer
* We have implemented a basic video player.
* We have documented the flow of the processes followed while building this OS.
* We studied some previously built OS, to get a deeper understanding and compare as well as analyze their functionalities and hence added a Literature survey.









**3.3.1 My Contribution**:

My contribution in the course of the project was to create a well descriptive documentation which is written in an easy and understandable language to read and follow for one to start developing their own Operating System.

The points that were kept in mind while documentation were:

1. Easy to read and understand
2. Short and crisp descriptions
3. Step-wise implementation process
4. Sample code to be provided

For eg.,

Documentation of keyboard driver and respective assembly module, which loads the IDT into the memory and enables the system to take interrupts from the keyboard.

**Keyboard Drivers:**

1. What is a Keyboard Driver?

Keyboard is the primary input device for a computer. Working with a keyboard is a non avoidable factor in os development.

1. Why is it needed?

To implement the keyboard,we need to ask every time if there is any input or not. But this isn't an efficient way, as the keyboard is much slower than the processor. Asking it everytime will affect the processing speed badly. And it also will put a lot of traffic in the system bus.

1. How is it implemented?

The solution implemented by the x86 chip manufacturer uses interrupts.

Whenever a key is being pressed, The cpu will call a function that we pre define to it. The processor won’t do the keyboard logic, but it will let us execute some code whenever a key is being pressed.

In simple words, the code to handle the keyboard input passes its address to the interrupt descriptor table and tells the processor to load it. After loading all the parameters and the location of our keyboard handling code, It will call that code whenever a key is being pressed.

When we get this interrupt, We could try reading from the keyboard which gives us the key being pressed.

Note: What keyboard gives as the value for pressed key is not ascii. It is called scan codes. The OS converts it to ascii in the kernel.

The PIC Chip

**PIC** or **Programmable Interrupt Controller** is a chip in the computer

whose main job is to generate interrupts. When a key in keyboard is pressed,

The Chip inside keyboard tells to the pic chip inside our computer to

generate a #1 Interrupt. The pic chip will then decide the time to notify the

cpu about the interrupt. When the cpu gets the message which says a key is

being pressed, It executes a set of code to handle the interrupt.

1. Code Explanation.

In the main.c file, in the **start()** function, the kernel calls theinitIDT() function. The job of this function is to set the Interrupt Descriptor Table so that when a

key is pressed, It will call the isr1\_Handler() function.

1. The structIDT\_ENTRY could be used to define theidt. There are a total of

**256** possible idt entries. So we created 256 of them with struct

IDT\_ENTRY idt[256]

struct IDT\_ENTRY{

unsigned short base\_Lower;

unsigned short selector;

unsigned char zero;

unsigned char flags;

unsigned short base\_Higher;

};

struct IDT\_ENTRY idt[256];

unsigned int base;

1. The next command extern unsigned int isr1; says to the compiler to implement compilation so that it knows that there will be a section named isr1 in an outside source file. We could find this section in the IDT.asm file.

extern unsigned int isr1;

1. In the start() function in main.c, we copied the address of the isr1 section to the variable base with base = (unsigned int)&isr1; (We used the & Symbol to get the address).
2. In the initIDT() function, we mapped the #1 interrupt(Keyboard

interrupt(IRQ 1)) with:

idt[1].base\_Lower = (base & 0xFFFF);

idt[1].base\_Higher = (base >> 16) & 0xFFFF;

idt[1].selector = 0x08;

idt[1].zero = 0;

idt[1].flags = 0x8e;

Here we set the location of the isr1 section in the first two lines using the value in base variable.

The next three variables set the arguments necessary for the idt entry.

1. Then we prepared the pic chip to handle interrupts and finally, we called the

loadIdt() function. At the very top of the code, we said to the compiler that

the loadIdt() function will be outside the c source file with extern void

loadIdt();

1. When the loadIdt() code is executed, it jumps to that section we defined in

the IDT.asm file.

\_loadIdt:

lidt[idtDesc]

sti

Ret

lidt is a x86 command used to make the processor load the Interrupt

Descriptor Table. Here, it loads the idt entry Which the idtDesc section

points to.

The command sti is used to enable the interrupts. From now onwards , we

will get every interrupt.

1. From now onwards, the \_isr1 section in IDT.asm file will be called when

ever a keyboard key is being pressed.

1. The \_isr1 section calls the \_isr1\_Handler function we defined in the

main.c file

Whenever a key is pressed, the function isr1\_Handler() will be called. We

could implement whatever to do in that function.

1. According to our implementation we first obtained the scan code of key

being pressed with inportb(0x60) and passed it to the handleKeypress()

function with handleKeypress(inportb(0x60));

1. In the handleKeypress() function, we have an array of values named

Scancode[].

We could take the scan code as an index to point to the Scancode[] array

The arrangement of values in Scancode[] gives us the ascii representation

of the scan code.

1. Finally we called a function to print the character being pressed.

THE inportb AND outportb FUNCTIONS ARE FUNCTIONS THAT HELP US

COMMUNICATE WITH EXTERNAL DEVICES. in COMMAND ACCEPTS INPUT

FROM EXTERNAL DEVICES AND out COMMAND OUTPUTS COMMANDS AND

DATA TO EXTERNAL DEVICES.

Compile and run the code by executing the compile.bat file and try pressing

any character on keyboard, The program simply prints it to the screen.

**4. CONCLUSION:**

Through this project, we are developing an open source OS which is made purely for educational purposes for helping out the student community understand how an Operating System works.

Architecture that is followed here is Monolithic Architecture. The main method that is followed while creating this Operating system is the software development waterfall model method.​

The outcome of this project is to create a learning tool for the students to get a deeper understanding on how OS works.​

​

​**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)
6. **PLAGIARISM REPORT:**

<https://drive.google.com/drive/folders/1hcQo7eYmQTruP8ln4W_pvr2_76FkvkGd?usp=sharing>