Making an Operating System from Scratch with Multitasking and Dynamic Memory Management

Ayushmaan Agarwal
19BCE2070
VIT University
Vellore, India
ayushmaan0901@gmail.com

Sharadindu Adhikari
19BCE2105
VIT University
Vellore, India
sharadindu@zohomail.in

Tanmay Kumar Agrawal
19BCE2106
VIT University
Vellore, India
tanmaytanu.10@gmail.com

Soumyadip Mondal
19BCE2107
VIT University
Vellore, India
soumopcm@gmail.com

***Abstract*—This project will be basically on how we can make an Operating System from scratch with some of the necessary things that it needs. We're going to use virtualbox to check at every point if our operating system is running properly. We are going to use a linux ubuntu package called xorriso that will help us in making the iso image file of my operating system which we can then put in the virtualbox for running. We are going to use the assembly language and c++ programming language to write the operating system initially and also use make files. We will use the default linux bootloader that is present in a linux system. We are also going to put there some multitasking features and file systems which will be totally written from scratch. The dynamic memory has many problems so we would solve that problem. After the completion of this project, we would properly understand how the operating system works along with its several components that are used in it.**

We have put in the drivers for keyboard and mouse so that we can use them. The global descriptor table was also set up in order to talk with the hardware ports and send messages to them according to the interrupts called by the keyboard driver and mouse driver. We have decided to put a very basic GUI to the operating system using a Video Graphics Card that is already present in the computer hardware. We have tried to make some windows on the screen and with the help of the mouse you can drag those windows around the screen. We have also set the background color and we can change the background colour according to our requirements. We have added multitasking in the operating system in which a reserve stack is made so that some of the tasks will go to that particular stack and will be run using its registers and different calls. There will be a task array pointer and one by one these tasks will be given to the CPU using the round robin scheduling algorithm.

The dynamic memory has also been added in which there will be memory chunks and according to the request of the size the chunks will be allocated to that task or process. We have overridden the malloc and delete functions already present there in CPP to allocate and deallocate memory. The overiddent functions will work according to what we have assigned to it. So, all the structure and the code of the OS will be in this report.

***Index Terms*—operating system, vga, virtual machine, gui, kernel, drivers, gdt, interrupts, ports, pci**

I. Introduction

The aim of this project is to make an operating system from scratch that has some basic requirements such as interrupt handler, drivers for making the drivers such as keyboard and mouse should work, peripheral component interconnect, some simple scheduling, multitasking and dynamic memory management. We have also tried to put some basic visual graphics for the change of color of the screen and making kind of windows on the screen which can be dragged around the screen using the mouse. We have applied the multitasking in which there is a reserve stack and then some of the tasks are done with the help of this task so that the kernel stack is not overburdened. Dynamic Memory Allocation is also added in the operating system so that a particular part of the RAM can be used for dynamically allocating the memory.

The main purpose of trying to make an Operating System from scratch was to understand how the operating system starts up and how it communicates with the hardware. The main motivation was the curiosity to know everything about the operating system. We've implemented different things such as the partial implementation of the graphics user interface, dynamic memory management and multitasking. By understanding the working of such magnificent things, we got confidence that we can make our own Operating System with different types of features in it.

The primary objective is to make a 32 bit Operating System that can do multiple things such as using drivers, multitasking, dynamic memory management. The operating system should be working and the things that have been implemented should be shown on the screen of the OS when it is booted up. These all things have been implemented successfully and detailed discussion about it has been done successfully.

II. Data Collection

A. Build an Operating System from Scratch: A Project for an Introductory Operating Systems Course

This research article was written by Michael D. Black, which was published in the journal SIGCSE 2019: Proceedings of the 40th ACM technical symposium on Computer Science education.

This paper describes how we can make a ground - up operating system which is capable of booting from a floppy disk on an actual machine. This paper was designed for students with no prior exposure to data structures, assembly language, or computer organization. This also helped in writing a system consisting of system calls, program execution, a file system, a command line shell, and support for multiprocessing. In this research paper several factors were there such as:

Keeping pre-written assembly code to an absolute minimum, and providing clear instructions on calling these functions. Instead of custom device drivers, we are using BIOS calls. Including no advanced data structure, binary math, stack handling, or assembly programming. This article helped quite a lot in clearing the things about how we can make an OS from scratch.

B. Operating System and Decision Making

This research article was written by Hussain A. Alhassan and Dr. Christian Bach, which was published in the journal ASEE 2014 Zone 1 Conference, April 3-5, 2017.

This paper helps us to understand the aspects that affect the decision with respect to the user's background and what core factors drive their choice of operating system. It tells us about the five factors such as convenience, capability, security, interface and recovery. This particular paper has compared many operating systems such as Unix, Windows, Macintosh and Linux, on the factors mentioned above. The result of the evaluation was that the majority of people choose the Windows operating system because they find it easy to work with, it has a friendly GUI. Because of these reasons, some people keep using Windows even if they are aware of weakness in stability and sometimes the security issue. When people go deeper and gain more skills, they want some advanced features and they plan to use assembly language, they would rather use Linux OS as it gives more flexibility to the user.

This paper helped us in knowing the different working of the operating systems that are available in the market so that we can choose the best functionalities for our OS making project.

C. Building an Adaptive Operating System for Predictability and Efficiency

This technical report was written by Gage Eads, Juan Clomenares, Steven Hofmeyr, Sarah Bird, Davide Bartolini, David Chou, Brian Glutzman, Krste Asanovic and John D. Kubiatowicz. This technical report was submitted to the Electrical Engineering and Computer Sciences University of California at Berkeley. 2016.

This approach suggested by this article is called Adaptive Resource Centric Computing (ARCC), in which the OS distributes resources to QoS domains called cells. These cells are actually explicitly parallel lightweight containers with guaranteed user level access to resources.

This technical paper has given explicit details about our implementation of ARCC on an experimental platform for resource management on multicore based computers. They also discussed the implementation of cells, user level scheduling and resource allocation policies. We can use this scheduling and resource allocation to get an idea about how we should do these things in our OS project.

D. Writing a Simple Operating System — from Scratch

This book was drafted by Nick Blundell for the School of Computer Science, University of Birmingham, UK.

This book has everything from the softwares available for CPU emulation, boot sector programming, interrupts, CPU registers and how to put it all together. This draft also shows how to write our own kernel and creating a boot sector to bootstrap our kernel and how we can find our way inside the boot sector of the kernel with the help of magic number.

It also tells us about the input - output buses and the programming that is needed for it and about the screen driver such as understanding the display devices and basic screen driver implementation. This draft has also talked about handling interrupts, keyboard driver, hard - disk driver and file system.

This draft was great in making an OS from scratch and we have used some methods that were explained in this particular draft.

E. A Survey Of Contemporary Instructional Operating Systems For Use In Under-graduate Courses

This article was written by Charles L. Anderson and Minh Nguyen for the Journal of Computing Sciences in Colleges, 2005.

This article has described the many instructional OS that is used in the different colleges around the globe for the purpose of teaching. They talked about the different platforms that can be used to demonstrate the OS and which languages are preferred with the assembly language in making the operating system. They described the working of different OS such as GeekOS, Nachos, Minix etc. The development environment needed for the development of the OS was also described.

From this research article we got to know how to set up the environment and what language to choose for the development of my operating system which we are writing from scratch.

F. Operating System from the Scratch: a Problem-Based Learning Approach for the Emerging Demands on OS Development.

This research article was written by Rene S. Pinto, Pedro Nobile, Edwin Mamani, Lourenco P. Junior, Helder J. F. Luz and Francisco J. Monaco, it was published in the International Conference on Computational Science, ICCS 2013.

This research paper includes abundant documentation, both external and internal, which explains everything and this also gives us the guided steps to implement an entire OS. They have used an OS called the TempOS, which has a 32 bit architecture. This paper is basically a systematic compilation of all the information of making an OS into a guide for laboratory practices which will definitely help in making a realistic and fully functional OS.

From this research paper we got to know how to take architectural decisions that will later result in a strong interdependence of different components. This will help a particular functionality to be performed by complex interactions among a lot of procedures and program functions that share the states.

III. Literature Review

After going through 20 research papers in this domain, we've jotted down their scope, limitations and relevance using as little words as possible. The summary has been presented in a Tabular manner:

Sl. No.	Title	Summary	Contribution	Research Gap
1	Build an Operating System from Scratch: A project for an Introductory Operating System course	This paper describes how we can make a ground - up operating system which is capable of booting from a floppy disk	Given significant information on Multi processing, system calls, and writing to files.	There was no account of the dynamic memory management as well as of the multitasking that has been done in the project
2	Operating System and Decision Making	This paper helps us to understand the aspects that affect the decision with respect to the user's background and what core factors drive their choice of operating system.	The proper working of the different types of operating system is reported and a working model is given.	They did not tell the working of a particular operating system from the root level
3	Building an Adaptive Operating System Predictability and Efficiency	This technical paper has given explicit details about our implementation of ARCC on an experimental platform for resource management on multicore based computers.	They reported about a new type of computing known as ARCC that improves the efficiency of the scheduling.	The CPU scheduling mechanism is very complex and if a small error is there it will not be easy to debug it.
4	Writing a Simple Operating System - from Scratch	This draft shows how to write our own kernel and create a boot sector to bootstrap our kernel and how we can find our way inside the boot sector of the kernel with the help of magic numbers	Here, everything is given for making an operating system from the software that one can use	This paper did not talk about how we can implement a graphic user interface which should have been talked about.
5	Operating System from the Scratch	This research paper includes abundant documentation, both external and internal, which explains everything and this also gives us the guided steps to implement an entire OS.	They modified an existing small Operating System TempOS and tried to implement different approaches in system calls and multi tasking	The TempOS is a bit of complicated operating system to work on so we had to write something from scratch which is not much complicated.
6	A Survey of Contemporary Instructional Operating Systems for Use in Undergraduate Courses	This article has described the many instructional OS that is used in the different colleges around the globe for the purpose of teaching.	They talked about how we can modify different Operating System such as GeekOS, Nachos, Minix etc.	The discussions were not of root level such as how we can modify the drivers, descriptor table etc.
7	Multitasking Operating Systems	This article focuses on multitasking operating system. MOS is one that can work on more than one task at a time by switching between the tasks very rapidly. The tasks may all pertain to a single user or to multiple users.	They talked about Context switch, cooperative multitasking, hardware interruption, multiprocessing, preemptive multitasking, time-sharing	Major drawback are heavy demands on a computer, which would require hi-fi configurations
8	Multiprogramming System vs. Multitasking System	This article focuses on multi programming systems; there are one or more programs loaded in main memory which are ready to execute. Only one program at a time is able to get the CPU for executing its instructions while all the others are waiting their turn.	They discussed about different elements of Multitasking and Multiprogramming, and how their differences affect the user base.	In multi-programming OS , OS switches the CPU from one process to another process. that switching is called the context switching.
9	Memory Management In Operating System	This paper depicts the memory the executives in a working framework and it will show the fundamental design of division in a working framework and essential of its assignment.	They comprehended about the irreplaceable idea of a working framework and its memory the executives and its division in memory	Several problems arises due to sub-allocators, fragmentation and paging
10	Computer’s Memory Management	This article is based on the fact that main memory is first scanned of all the holes that have been created and the hole that can fit the process’s memory requirements is assigned. One of the algorithms that is used to assign processes memory holes is the round robin algorithm.	They mainly talked about different points of view, relocation, and partitioning	The memory management talked about here cannot be complete if virtual memory is not considered.
11	Memory Management: Challenges and Techniques for traditional Memory Allocation Algorithms	This research paper gave a comparative analysis of some well-known memory management techniques to highlight issues for real time systems	Hoard, tertiary buddy system and two level segregated fit are suitable for real time applications with faster response time.	Sequential fit algorithm is slow, and its use in the entire OS will make the program lag
12	Understanding Enforcement of Flow of Processes in Operating Systems	This paper discussed core functionality of OSs, which is to keep the flow of processes or the execution of the OS running.	The paper helped understand how proper flow of processes have been enforced in OSs	Nothing major, since it primarily focused on the enforcement techniques
13	File Systems for Various Operating Systems: A Review	This article focused on every operation that needs the support of primary or secondary memory and studies including their computing environments, performance characteristics and other parameters.	Plethora of storage techniques or file systems have been introduced, which helped understand our requirements for the project	There can be performance limitation and single point failure problem
14	Role of File System in Operating System	This paper gives a view on file system in an operating system. The file system, simple file system, distributed file system, UNIX file system and LINUX file system are discussed in detail	Theoretically its contribution is: we may change the file into a directory by changing its bit	The processor will entrée one of the obtainable servers linked with that split which may evade performance.
15	Function-Level Multitasking Interface Design in an Embedded Operating System with Reconfigurable Hardware	This paper presents the multitasking interface design in an embedded operating system with reconfigurable hardware. The proposed approach has three main design features.	The reconfigurable hardware functions are managed and scheduled by the operating system. Applications can use any needed hardware function via invocation APIs, which is one of the key points we attributed in our project.	The design of multitasking support for reconfigurable hardware is not straightforward because hardware unit sharing is very different with software library sharing.
16	File System – A Component Of Operating System	This paper gives a light on the layer of file system in the computer system. All the upper layers and the lower layer are shown in it that gives a sound knowledge about the file system interaction between them.	Discussions on high-level details of file systems, as well as related topics such as the disk cache, the file system interface to the kernel.	The software is not built to comeback with the hardware technology. So there is a need to do research in this area to bridge the technology gap
17	Memory Management in Operating System	In this paper different memory allocation techniques have been discussed along with their comparative analysis. This paper also describes about the basic concept of virtual memory management and dynamic memory management	We understood about the indispensable concept of an operating system and its memory	If virtual memory doesn’t load (even partially), then it’s greatly disadvantageous.
18	A Survey of Distributed File Systems	This paper focuses on the continuing interest in distributed file systems bears testimony to the robustness of this model of data sharing.	Focus on a taxonomy of file system issues, a brief history of distributed file systems, and a summary of empirical research on file properties.	Permanent storage is a fundamental abstraction in computing.
19	Semantic File Systems	This paper described how a semantic file system can provide associative attribute-based access to the contents of an information storage system with the help of file type specific transducers.	Compatibility with existing file system protocols is provided by introducing the concept of a virtual directory.	Measurements show that transducers generate approximately 30 disk transfers per second, thereby saturating the disk.
20	Exploiting File System Awareness For Improvements To Storage Virtualization	This paper has shown that simple hints from the file system, can be exploited at the block layer to meet overall goals for the storage systems, which were otherwise harder to achieve.	Stable file systems can benefit from providing hints to the block layer	Block layer interface has a disadvantage of acting as an impediment to meeting higher-level goals for the storage stack entirely from within the file system

IV. Project Resource Requirements

A. Software Requirements

- VS Code / CLion Text Editor
- g++ Compiler (9.0.0 or above)
- xorriso (sudo apt-get install VirtualBox grub-legacy xorriso)
- binutils library (sudo apt-get install g++ binutils libc6-dev-i386)
- Oracle Virtual Box
- Linux based OS

B. Hardware Requirements

- Processor: Core 2 Duo or higher
- RAM: Minimum of 1024 MB
- OS architecture: Minimum 32 bits
- GPU: at least Intel HD Graphics

V. Discussions

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatt

A. Kernel

Bootloader is a bit sophisticated, it knows about partition table and file systems. It can deal with directory structure so it can go to the second partition and look into the directory "/boot/grub/grub.cfg" and load this file and read contents such as which Operating System is where on the hard drive. The loader that I have written tells the CPU to jump in the kernel. It also sets the stack pointer and jumps into other files such as kernel.cpp.

The linker.ld because we have programs with two different programming languages here C++ and Assembly language. We want to combine the two files and give us a file kernel.bin. This file will be put in the boot directory this is, "/boot/kernel.bin" and we have to write about this entry in grub.cfg.

When you start the Operating System CPU starts in a 32 bit compatibility mode. So, in the kernel we are in the 32 bit mode. The bootloader will not recognize this kernel that is written because the bootloader looks at kernel.bin, it will look for so -called ustric number in that file and if it is not there, it will not believe that it is actually a kernel. So we have to put the ustric number. The ustric number is a hexadecimal number 0x1BADB002. We have to put that in the kernel.bin file so that bootloader knows that kernel.bin is a kernel and then loads it. There is a multiboot structure containing some information such as the size of the RAM, it copies the magic number in the BX register.

Whatever that is made such as the interrupts, global descriptor table, peripheral component interconnect etc. These must be included in the kernel.cpp file and then activated or called from there only because eventually we are compiling the kernel which is the main part.

For the print function that has been implemented in order to show something on the screen, it uses a Video Memory that is already present in the hardware of the computer system. The Video Memory is a place where when you send some "const char*", it will be displayed on the screen and the good thing is that we can change the color according to our comfort. This particular thing has been used for the printf statement.

B. Drivers

The driver file was made to handle different kinds of drivers that are present in the computer system. Here we have implemented the Keyboard and the Mouse only. But we can link a maximum of 255 drivers at a time. The main job of this driver is to talk to activate, deactivate or reset drivers. There are functions in the driver class that can activate all the drivers associated with it in one go.

C. Global Descriptor Table

The Global Descriptor Table or gdt is a data structure that is used by the Intel processors. We have defined segments inside this table so that we can switch between the segments when the interrupts are called from the keyboard or the mouse or from any other place. They define the characteristics of the various memory areas used during program execution, including the base address, the size and access privileges like executability and writability. Here I have defined the Null Segment Selector, Unused Segment Selector, Code Segment Selector and the Data Segment Selector. These things are all necessary for handling the interrupts that cous from the hardware to the processors.

D. Interrupts

So basically an interrupt is a signal to the processor emitted by hardware or software indicating that an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) for Interrupt Handler. This ISR tells the processor for the controller what to do when the interrupt occurs. HTe interrupts can be either hardware interrupts or software interrupts. In the code also I have an Interrupt Manager Class and an Interrupt Handler class that handles the interrupts coming in from the mouse and the keyboard.

E. Ports

Ports are necessary unsigned integers or hexadecimal numbers that help in the communication of the interrupts. So basically is a communication endpoint. It identifies a specific process or a type of network service. It is mostly a 16 bit unsigned integer and is called a port number. I have implemented ports for 8 bit unsigned, 8 bit slow unsigned, 16 bit unsigned and 32 bit unsigned because we will be needing all of those types of ports.

F. Keyboard

Now we have the keyboard so in this particular keyboard class I have implemented a Keyboard Event Handler this is, this will handle the interrupts when the keys on the keyboard are pressed or released. In both of the situations different types of interrupts are called so we have to handle that accordingly.

Then we have the dataport and the commandport. The dataport is for reading the typed key values interrupt and the command port is telling what processor has to do accordingly. We also need to activate it first so it is done with the help of these data ports and command ports only.

G. Mouse

Similarly like the Keyboard we have the Mouse Event Handler and it handler the mouse interrupts such as the mouse button when clicked there is different type of interrupt but when it is released then different type of interrupt is there. We also have a usthod that changes the x and y axis values of the mouse so that the mouse can move accordingly. Here we also have the data ports and the command ports. The mouse is activated through the data ports and command ports. We have the buffer array which is used when the mouse cursor has to be moved.

H. Peripheral Component Interconnect (PCI)

It is a local computer bus for attaching hardware devices in a computer and is a part of the PCI Local Bus Standard. It supports the functions found on a processor bus but in a standardized format. The devices connected to the PCI bus appear to a bus master to be connected directly to its own bus and are assigned addresses in the processor's address space. With the help of this the user doesn't have to tell the interrupt about the different devices. In my operating system, the PCI is connected with 8 buses and then 1 bus can be connected to a maximum of 32 devices and 1 device can be connected to a maximum of 8 functions. Then there are window-id and device-id. Every major window has its own unique window-id. It is very useful when you want to decide which driver you want to instantiate to talk to a certain device. If VGA graphics are used then class-id and the subclass-id are enough to distinctly identify everything.

I. VGA

It is basically a video display controller and it contains the de facto graphics standard. It has got 20 x 200 pixels, 256 colors and something to build on a framework. This may not be enough today but it is good for my Operating System. We have to talk to the graphics directly and tell them to go into graphics mode. The VGA here has a fixed memory so no need to contact the PCI, drivers for controllers etc. So this video graphics mainly has a window and a widget. Everything is mainly applied with the help of the widget. A widget class has already been defined and in that many methods has been defined in those classes that will help multiple widgets to exist and they can even overlap each other. We have put a mouse handler there so that we can even drag those widgets around the screen. In the VGA many ports had to be called and assigned also so that we could implement the graphics. We had the miscellaneous port, cathode ray tube port, sequencing port, graphics controller index and attribute controller index. Every port has its own job to do in the Video Graphics Array.

J. Multitasking

It is basically the concurrent execution of multiple tasks over a certain period of time. New tasks can interrupt already started ones before they finish, instead of waiting for them to end. The multitasking automatically interrupts the running program, saving its state and loading the saved state of another program and transferring control to it. We will have our own stack in the RAM. We write the values of the processor on the stack. When we call it takes the functions and puts it in the reserve stack. The CPU copies the values of the reserve stack back to the processors. Basically the processor will not go to the regular interrupt subs that we created, it will go to the other one for executing and this is the difference. In the handler interrupt request of the new interrupt subs we need to just push something like 0 otherwise we would have just the interrupt request and the system would get confused. In the interrupt request it doesnot push anything so we have to push something so that the CPU structure works.

VI. Details of Algorithm

About Dynamic Memory Allocation:

We are having a class Memory Chunk which will have a pointer for the previous and the next memory chunk just like a doubly linked list. A boolean value will tell us if the memory is allocated or not. Memory Manager will be responsible for handling the space that is allocated. The manager will find the area in the RAM that can accommodate that particular process and then it will return a pointer to that place. If you request 1024 bytes then that size of memory chunk will be allocated. We have to find a place that is free with the help of the linked list.

If a large chunk is found and we want only 10 MB but the chunk is of 30MB then we have to split some of the memory. We have to chunk that particular memory area. We have allocated the process that has requested the memory. Now we get the region that it was holding after we free that place. So, we just have to subtract that size of memory chunk. The overridden mallock that we have created will throw an exception if it will not be able to find a memory chunk of that size.

Why have we chosen Dynamic over Static?

Static Allocation	Dynamic Allocation
Performed at static or compile time	Performed at dynamic or run time
Assigned to stack	Assigned to heap
Size must be known at compile time	Size may be unknown at compile time
First in, last out	No particular order of assignment
It is best if required size of memory known in advance	It is best if we don't have idea about how much memory require

Working of Dynamic Memory Allocation:

We have seen how dynamic allocation plays important role in memory management. Now we are going to present, how does dynamic memory allocation works?

We have used C programming language for this.

Problem: We would like to dynamically allocate memory for 10000 integers and after completing its work, de-allocate memory for recycled it.

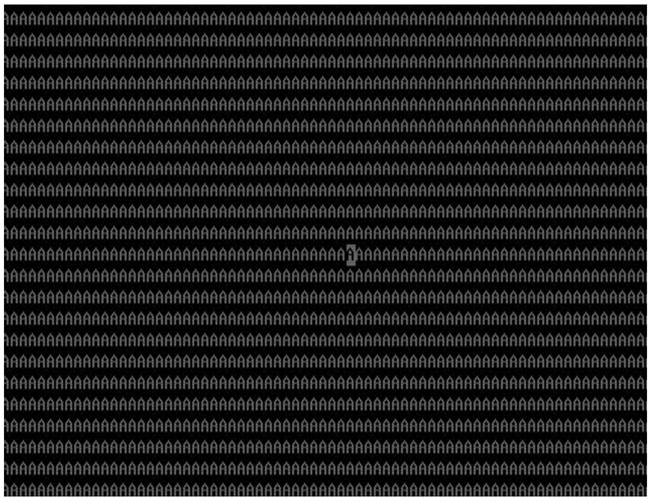
Solution: C Programing language has provided stdlib.h header file, which contains functions malloc(), calloc() and realloc() for allocation while free() for de-allocation of memory. The task is to allocate memory for 10000 integers; we can do this in two ways statically or dynamically. This is accomplished dynamically as follows: Suppose variable name: data (integer type). Dynamic allocation will require pointer variable instead of normal variable, symbol * (asterisk) shows pointer variable declaration. int * data;

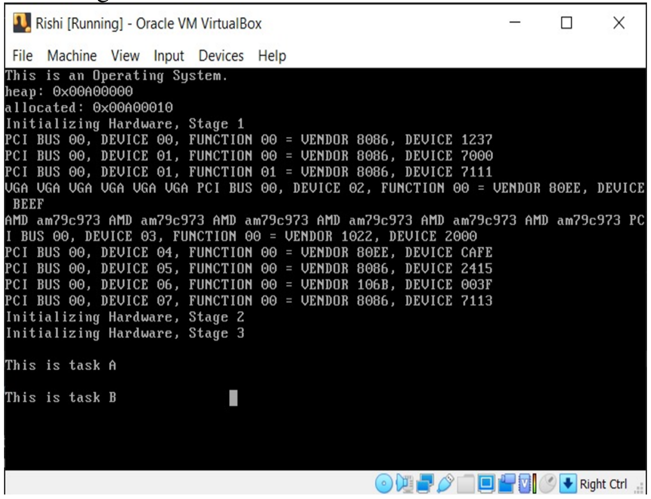
The above statement, declare 'data' as integer pointer. We are going to use malloc() function, and also use sizeof() function which will gives number of bytes require for integer data type. It will help because size of integer is changes as per operating system (Linux: 4 bytes and Windows: 2 bytes). data = (int *) malloc (10000 * sizeof(int)); malloc() function returns void pointer, so need to convert into our destination type means integer. Above statement will allocate memory for 10000 integers and 'data' pointer will point to first block of allocated memory. Now, next task is de-allocation of memory. We will use free() function to de-allocate memory. Following statement will accomplish this task: free(data); It will de-allocate memory to reuse, but still 'data' pointer will points to that memory location. We can called 'data' pointer is a dangling pointer because it pointing to memory even it is de-allocated. Dangling pointer is a pointer which is pointing to nothing. To remove dangling pointer we use following statement: data = NULL; Above statement will remove dangling pointer that means 'data' will point to null instead of any random memory. Above example shows how did efficiently handle memory using dynamic allocation.

We have to remember following points during dynamic allocation of memory: The memory allocator maintains track of what is free and what is allocated. - When we obtain memory from the global pool, we must assume that it contains garbage, and properly initialize it. - We must not assume that successive requests will allocate contiguous memory blocks. A dynamic memory allocation takes help of pointers during allocating memory to hold the address of allocated block. Array is decent entity; we could not change its size in middle of execution of program. - If Array = small then our program fails to handle large amount of data at run time. - If Array = big then lots of space has been wastage means inefficient use of memory.

The best solution is to create a data structure (language independent entity) which start from small piece of memory and add new piece of memory whenever require at time. Pointer is connection here which holds these pieces.

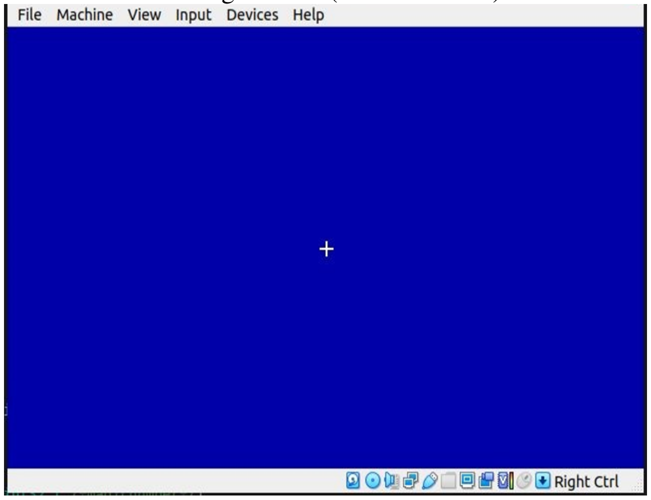
VII. Implementation Screenshots


This shows that task A and B are running concurrently in multitasking.

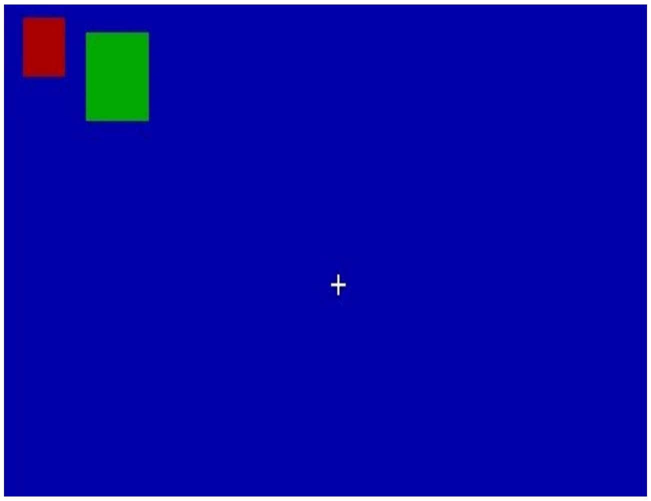

Rishi [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
This is an Operating System.
heap: 0x00A00000
allocated: 0x00A00010
Initializing Hardware, Stage 1
PCI BUS 00, DEVICE 00, FUNCTION 00 = VENDOR 8086, DEVICE 1237
PCI BUS 00, DEVICE 01, FUNCTION 00 = VENDOR 8086, DEVICE 7000
PCI BUS 00, DEVICE 01, FUNCTION 01 = VENDOR 8086, DEVICE 7111
VGA VGA VGA VGA VGA VGA PCI BUS 00, DEVICE 02, FUNCTION 00 = VENDOR 80EE, DEVICE
 BEEF
AMD am79c973 AMD am79c973 AMD am79c973 AMD am79c973 AMD am79c973 AMD am79c973 PC
I BUS 00, DEVICE 03, FUNCTION 00 = VENDOR 1022, DEVICE 2000
PCI BUS 00, DEVICE 04, FUNCTION 00 = VENDOR 80EE, DEVICE CAFE
PCI BUS 00, DEVICE 05, FUNCTION 00 = VENDOR 8086, DEVICE 2415
PCI BUS 00, DEVICE 06, FUNCTION 00 = VENDOR 106B, DEVICE 003F
PCI BUS 00, DEVICE 07, FUNCTION 00 = VENDOR 8086, DEVICE 7113
Initializing Hardware, Stage 2
Initializing Hardware, Stage 3

This is task A

This is task B
```

The above shows the Dynamic Memory and the Heap that has been allocated and also the information about the Peripheral Component Interconnect such as bus, vendor and device details are being shown. (VGA disabled)



This is the Video Graphics Mode that is implemented through VGA.



We can see the two widgets above that can be dragged with the help of the mouse given on the screen and can even be overlapped.

VIII. Novelty

The way that we have written the code may not be very advanced for fancy but it is quite simple and the biggest advantage of my code is that it is very simple. The idea and the code that we have written can be easily understood by the students that are new for this operating system project. The people who want to go in this field can start by reading the code that we have written. We have also used the ideas that were presented in the research papers that we have read.

We have made the dynamic memory management part using a doubly linked list which is very simple in itself and curb many problems. Several issues such as external fragmentation, which arises when there are many small gaps between the allocated memory blocks. Here as already told in the discussion that we have used the memory chunking methods to sort out this problem. The chunking also ensures that the memories don't overlap with each other so that no memory is ever lost. The dynamic allocation that we have done doesn't provide just a fixed block; you can change the size of the memory according to the requirements of the processes even when they are running so that is a bit unique part of the project.

IX. Result

The operating system has been successfully made and we have discussed all the things above that we have included and did in our operating system. We are quite glad that we were able to complete the operating system. Many more things can still be added by whatever we have done is solely written by us from scratch.

X. Scope of Future Work

Security: providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system.

Network: allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), private or other networks.

Adding a File System.

Acknowledgment

We would like to express our special thanks of gratitude to Professor Sudha S. who gave us the golden opportunity to do this wonderful project on Operating Systems, which also helped us in doing a lot of research work, thereby allowing us to dig deeper in some specific domains which otherwise wouldn't have been possible.

References

[1] Black, Michael D. "Build an operating system from scratch: a project for an introductory operating systems course." Proceedings of the 40th ACM technical symposium on Computer science education. 2019.

[2] Alhassan, Hussain A., and Christian Bach. "Operating system and decision making." ASEE 2014 Zone I Conference. 2017.

[3] Eads, Gage, et al. Building an adaptive operating system for predictability and efficiency. Tech. Rep. UCB/EECS-2014-137, EECS Department, University of California, Berkeley, 2016.

[4] Blundell, Nick. "Writing a Simple Operating System—from Scratch." (2009).

[5] Pinto, Renê S., et al. "Operating System from the Scratch: a Problem-Based Learning Approach for the Emerging Demands on OS Development." Procedia Computer Science 18 (2013): 2472-2481.

[6] Anderson, Charles L., and Minh Nguyen. "A survey of contemporary instructional operating systems for use in undergraduate courses." Journal of Computing Sciences in Colleges 21.1 (2005): 183-190.

[7] Scott Zimmer, JD. "Multitasking Operating System". Science J Rank. 2016.

[8] Assad H. Thary, Al Ghrairi. "Multiprogramming System vs. Multitasking System". JComp. Society (2017).

[9] Dinesh Kumar, Mandeep Singh, Harpreet Kaur. "Memory Management In Operating System". JETIR. Volume 6, Issue 4. (2019).

[10] Silberschatz, Galvin and Gagne. "Operating System Concepts with Java", Chapter 9, Memory Management. 2003.

[11] Muhammad Abdullah Awais. "Memory Management: Challenges and Techniques for traditional Memory Allocation Algorithms in Relation with Today's Real Time Needs". ACSIJ. Vol 5, Issue 2. March 2016.

[12] Zaffar Ahmed Shaikh, Intzar Ali Lashari. "Understanding Enforcement of Flow of Processes in Operating Systems". Case Studies Journal. Vol 6. Issue 5. May 2017.

[13] Isma Irum, Mudassar Raza, Muhammad Sharif. "File Systems for Various Operating Systems: A Review". Research journal of Applied Sciences, Engineering and Technology. y 4(17): 2934-2947. 2012.

[14] Faheem Hafeez. "Role of File System in Operating System". International Journal of Computer Science and Innovation Vol. 3, pp. 117-127. 2016.

[15] I-Hsuan Huang, Chih-Chun Wang, Shih-Min Chu, and Cheng-Zen Yang. "Function-Level Multitasking Interface Design in an Embedded Operating System with Reconfigurable Hardware". Yuan Ze University, Taiwan. 2014.

[16] Brijender Kahanwal, Tejinder Pal Singh, Ruchira Bhargava, Girish Pal Singh. "File System – A Component Of Operating System". IJARCS. Vol.2, No. 6, pp. 224-229. 2011.

[17] Durgesh Raghuvanshi. "Memory Management in Operating System". International Journal of Trend in Scientific Research and Development (IJTSRD). Vol.2. Issue 5. August 2018.

[18] M. Satyanarayanan. "A Survey of Distributed File Systems". CMU University Journal. February 1989.

[19] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, James W. O'Toole, Jr. "Semantic File Systems". PSRG. MIT Lab for Computer Science. 2006.

[20] Vivek Lakshmanan. "Exploiting File System Awareness For Improvements To Storage Virtualization". University of Toronto. 2009.