iJRASET
International Journal For Research in Applied Science and Engineering Technology



# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 10 **Issue:** VIII **Month of publication:** August 2022

**DOI:** https://doi.org/10.22214/ijraset.2022.46432

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 10 Issue VIII Aug 2022- Available at www.ijraset.com*

# Design A Mini Operating System Using Open Source System

Dr. Ravindra Nath Katiyar[1], Sadhna Yadav[2]
[1]*Associate Professor, Department of Computer Science and Engineering, University Institute of Engineering and Technology, Chattrapati Shahuji Maharaj University,Kanpur, Uttar Pradesh, India 208024*
[2]*Assistant Professor, Department of Computer Science and Engineering, Allen House Institute of Technology, Kanpur, Uttar Pradesh, India 208008*

***Abstract:*** ***COSMOS (C# Open Source Managed Operating System) Paper provides a framework and tool-chain to develop an OS purely in managed code (C# and VB .NET). Cosmos supports integration with Visual Studio, and therefore is much easier to use. However, the framework is not rich, and therefore, there is still a lot of work to be done on OS developer side (E.g., it does not support mouse, file system etc.) In this paper, a microkernel based OS is developed using COSMOS framework for x86 based computer systems. This paper demonstrates device driver (PS/2) pointing device and GUI System. Since Cosmos does not provide support file system, mouse, keyboard, we have designed appropriate drivers for screen, mouse and keyboard. We have developed Event handlers and drivers for each device.In this part, we have developed drivers for Hard Disk and RAM to detect them in order to achieve partial fulfilment. From the development of general purpose computers, software industry has also evolved. However, there is not much improvement in OS development, as compared with other fields of software world: while other fields are enjoying power of garbage collection, OS developers are still doing their job in classic languages like C, C++ and assembly. Although there has been some research work regarding this, none of them was successful. This project is an attempt to develop a mini OS in pure managed code, particularly C# and .NET Framework 4.0.***

## I. INTRODUCTION

In whole history of computers, operating systems are most challenging and complex piece of software ever developed. Despite the complexity, very little efforts are made to make whole OS designing process easier, and as a result, operating systems are still developed in C, C++ and assembly language. Although modern languages like C# have made software development process much easier, they can't be directly used for OS development due to the fact they are compiled to intermediate language (Byte-code or IL), instead of machine language for underlying architecture. Since development of GC based languages, various efforts have been made to harness the power of GC in OS development lifecycle.

### *A. Motivation*

The Project is followed by using Cosmos Framework which is a wide known platform for building kernel. Cosmos Framework is explained as follow:-

1) *COSMOS Framework:* Open Source Managed Operating System written in C# Paper is a framework and tool chain for creating an operating system that only uses managed code (C# and VB.NET). Cosmos is significantly simpler to use because it allows integration with Visual Studio. There is still a lot of work to be done on the OS developer side because the framework is not robust (E.g., it does not support mouse, file system etc.). In this study, a microkernel-based OS for x86-based computer systems is built utilising the COSMOS framework. Device driver (PS/2) pointing device and GUI System are demonstrated in this work. We have created suitable drivers for Screen, Mouse, and Keyboard since Cosmos does not support file systems, mice, or keyboards. For each device, we have created Event handlers and drivers. In order to partially fulfil this need, we have created drivers for RAM and Hard Disk that can recognise them.

### *B. Solution*

The Solution of the event mentioned above is to develop a kernel using Cosmos Framework provided by Microsoft Corporation which is very helpful in achieving the requirements. This Development kit requires four tools which are as follow:-

1) Vmware studio
2) Vmware VIX API
3) Microsoft Visual Studio
4) Cosmos User Kit v92560

©IJRASET: All Rights are Reserved 

### C. Related Work

Operating systems are the most difficult and sophisticated piece of software ever created in the history of computing. Operating systems are still created in C, C++, and assembly language despite the complexity of the process; virtually nothing is done to make it simpler. However, because they are compiled to intermediate language (Byte-code or IL), rather than machine language for the underlying architecture, newer languages like C# cannot be utilised directly for OS development. Numerous initiatives have been made to utilise the capabilities of GC in the OS development lifecycle since the development of GC-based languages. The following are a few noteworthy initiatives in this direction:

1) *Java OS:* Java OS was the first attempt to develop an OS purely in managed code (Java in particular). It was announced by Sun Microsystems in 1996, and was claimed to run in anything from net computers to pagers. In 1998, IBM was hired to accelerate the project. The end of project was declared in 1999 by both companies, and was declared a legacy system in early 2003.
2) *Singularity:* Singularity is an experimental OS by Microsoft Research started in 2003. It was designed as a highly-dependable OS in which the kernel, device drivers, and applications are all written in managed code (C# in particular). However, C/C++ code has been used for executing managed code. This project was closed in 2010 and released in public domain under restricted licensing terms.
3) *MOSA:* Managed Operating System Alliance (MOSA) is an open source OS development tool-chain by developers mainly from USA, Germany and Netherlands. This paper provides command line tools to compiled managed libraries (.NET Framework) to native code, boot loader configuration, ISO image generation etc.

### D. Challenges

The challenges faced in building this kernel is based on the development kit which is provided to us. This kit builds an interface between the visual studio and virtual machine. Virtual machine needs drivers to operate the kernel. But the problem begins from here as cosmos does not provide any drivers for the hardware units such as mouse, display, keyboard and hard disk etc. These challenges made this paper very hard to implement and it is one of few successful builds of any kernel based on C# all over the globe.

### E. Contribution

This kernel is an important step in the direction of building such type of kernels, which works independently on the hardware provides to it. C# based kernel provides an efficiency that the operating system will work still if one of its hardware is not functioning properly. This project will help in further development of shell and other functions to the users who are interested in this type of project.

### F. Organization of Report

Chapter 2 will talk about functioning of basic tasks of kernel such as MBR. Chapter 3 will show information about memory management schemes and RAM functionality. Chapter 4 will tell about kernel level authentication.
Chapter 5 contains information about display drivers which are very important to display working of the kernel.
Chapter 6 and Chapter 7 will provide information about mouse and keyboard drivers. Chapter 8 tells about font handling.

## II. MASTER BOOT RECORD

### A. Boot Loader Initialization

In this part, the kernel loads in the main memory and loads MBR file into it so that necessary codes will execute and can build a directory structure for the processes which are going to be executed.

### B. RAM Detection

This part tells about the detection of RAM and to find its capacity and display it to the user when the operating system is loading using MBR. In this, we have built a function to know the capacity of RAM which is as follow:

### C. Font Initialization

Font Initialization is very basic need that an operating system requires. In order to achieve that basic functionality, we have defined some fonts and initialized the as follow:

## III. MEMORY MANAGEMENT INTERFACE

### A. Hard Disk Detection

This part of report contains information about hard disk and its detection. It is done by following commands which are responsible for executing the file system.

### B. File System Initialization

After detection of hard disk, we have to initialize the file system. The file system is initiated from FAT and VFS which includes important drivers and handlers.

1) *File Allocation Table (FAT):* A traditional file system that is straightforward and reliable is FAT. It can't provide the same performance, durability, and scalability as some contemporary file systems, but it still performs well even in lightweight implementations. However, it is a well-suited format for data exchange between computers and devices of almost any type and age from the early 1980s (when it was introduced) up to the present because it is supported for compatibility reasons by nearly all currently developed operating systems for personal computers, many mobile devices, and embedded systems. FAT was initially developed in the late 1970s for use on floppy discs, but it was quickly modified and throughout the next two decades of the DOS and Windows 9x operating systems, it was almost exclusively used on hard discs. FAT is no longer the default file system for use on Microsoft Windows PCs due to the emergence of more powerful machines and operating systems, as well as the creation of more complex file systems for them. Today, floppy discs, USB sticks, flash and other solid-state memory cards and modules, as well as numerous portable and embedded devices, still frequently use FAT file systems. As the default file system for digital cameras, DCF uses FAT. In the boot process of EFI-compliant systems, FAT is also used. The FAT index table, which was statically created at the time of formatting and is used frequently by the file system, is where the name of the file system comes from. Each cluster, or continuous region of disc storage, has an entry in the table. Each entry has a marker indicating the end of the file, free disc space, or specific reserved disc areas, or it contains the number of the subsequent cluster in the file. The operating system can then navigate the FAT table, seeking up the cluster number of each subsequent part of the disc file as a cluster chain until the end of the file is reached. The root directory of the disc holds the number of the first cluster of each file in that directory. Sub-directories are implemented similarly by special files that include the directory entries of the corresponding files. The maximum number of clusters has substantially increased as disc drives have progressed, and as a result, so has the amount of bits used to identify each cluster. The table element bit count determines the names of the following main FAT format releases: 12, 16, and 32 (FAT12, 16). (FAT32). These variations are all still in use today. While generally maintaining backwards compatibility with current software, the FAT standard has also been expanded in other ways. Uses Even in extremely lightweight implementations, the FAT file system provides a respectable level of resilience and efficiency. As a result, practically every operating system for personal computers currently in use, as well as several home computers and a variety of embedded systems, support it. The majority of portable devices, including PDAs, digital cameras, camcorders, media players, or mobile phones, support FAT file systems because they are the default file system for removable media (with the exception of CDs and DVDs). As a result, FAT file systems are frequently found on floppy discs, super-floppies, memory and flash memory cards, or USB flash drives. FAT16 and FAT32 are commonly found on the bigger media, but FAT12 is ubiquitous on floppy drives. While FAT was also often used on hard drives during the DOS and Windows 9x eras, its use has decreased since Windows XP was released and the more recent NTFS has taken over. Hard drives that are anticipated to be used by many operating systems, such as those seen in shared Windows, Linux, and DOS environments, nevertheless use FAT. Many operating systems offer FAT support through official or third-party file system handlers as a result of the ubiquitous use of media with the FAT file system. For instance, even if they also support more advanced file systems like ext4 or btrfs, Linux, FreeBSD, BeOS, and J Node have built-in support for FAT. FAT file systems are supported on discs other than the boot drive by Mac OS 9 and Mac OS X. Through the Cross DOS package, Amiga OS supports FAT. The main drawbacks of the NTFS file system are the size overhead for small volumes and the extremely limited support by anything other than the NT-based versions of Windows, as the exact specification is a trade secret of Microsoft. For many purposes, the NTFS file system is superior to FAT in terms of features and reliability. This worry has been somewhat reduced because to the advent of NTFS-3G since mid-2006, which has greatly enhanced NTFS support in UNIX-like operating systems. Without third-party drivers, NTFS is still incompatible with DOS-like operating systems, making it challenging to use a DOS floppy for recovery. Microsoft offered a recovery console to get around this problem, but by default it had very limited functionality due to security concerns. This disadvantage is finally being eliminated by the migration of recovery tools to boot CDs based on Bart PE, Linux (with NTFS-3G), or WinPE. FAT has been around for more than three decades and is often utilised in embedded solutions on desktop

and portable computers. It remains the most widely used file system in the world as a result. In order to optimise platform portability, the DCF file system, used by nearly all digital cameras since 1998, defines a logical file system with 8.3 filenames and mandates the use of FAT12, FAT16, FAT32, or ex FAT for its physical layer. Internally, the EFI system partition (partition type 0xEF) during the boot process of EFI-compliant systems also uses FAT. FAT has been standardised as ECMA-107 and ISO/IEC 9293:1994 for floppy discs (superseding ISO 9293:1987). These standards only support short 8.3 filenames and cover FAT12 and FAT16; lengthy filenames with VFAT are partially patented.

2) *Nomenclature:* Technically, "FAT file system" refers to all three of the file system's primary variations—FAT12, FAT16, and FAT32—but most parties make a clear distinction between them when it's essential. Since the introduction of FAT32, Microsoft has stopped making a distinction between the three formats and now refers to both FAT12 and FAT16 as "FAT," with "FAT32" receiving special treatment in dialogue boxes and documentation. If the precise type of file system being used is not stated or cannot be clearly selected, such as when the question "Do you want to format as FAT or FAT32?" is asked rather than "Do you want to format as FAT12, FAT16, or FAT32" The fact that the term "FAT16" can refer to either the entire family of FAT file systems with 16-bit wide cluster entries or, more specifically, only the original implementation of it with 16-bit sector entries, when it is necessary to distinguish between the original and the later implementation, adds to the confusion within the family of FAT16 file systems. Although the newer variant with 32-bit sector entries is technically known as "FAT16B," it is also referred to as "FAT16" in everyday speech, particularly since the original variant is now comparatively uncommon and is typically only implemented on small media when backwards compatibility with DOS versions prior to 3.31 is necessary. Additionally, the term "VFAT" has given rise to a number of misunderstandings because it is occasionally incorrectly used to refer to a different variant of the FAT file system to be distinguished from the FAT12, FAT16, and FAT32 file systems, when in fact it only specifies an optional extension that can be used on top of any FAT file system, including FAT12, FAT.16, or FAT32. Operating systems that do not support the VFAT extension can still read volumes using VFAT long-filenames as long as the underlying file system is supported. This article employs accepted prefixes for the byte unit in order to be precise and technically correct: 1 kB (kilobyte) is made up of 1000 bytes (103 bytes), while 1024 bytes (210 bytes) make up 1 KB

3) *Virtual File System (VFS):* A more abstract file system called a virtual file system (VFS) or virtual file system switch sits on top of a more tangible file system. A VFS's main function is to give client applications uniform access to many concrete file system types. For instance, a VFS can be used to transparently access both local and network storage devices without the client application realising. Applications can access files on local file systems of those sorts without needing to be aware of the type of file system they are accessing thanks to this method of bridging the gaps between Windows, Mac OS, and Unix file systems. An interface (or "contract") between the kernel and a certain file system is specified by a VFS. As a result, by merely upholding the contract, it is simple to add support for new file system types to the kernel. The operating system supplier might only make backward-compatible changes to the contract, ensuring that concrete file system support built for a given release of the operating system would work with future versions of the operating system, or the contract's terms could change inconsistently over time, necessitating the need for concrete file system support to be recompiled and possibly modified before recompilation. Sometimes the term "Virtual File System" refers to a file or a collection of files (not necessarily contained within a physical file system) that serve as a controllable container and should, via the use of software, provide the functionality of a physical file system. SolFS, a single-file virtual file system in an emulator like PCTask or so-called WinUAE, Oracle's VirtualBox, Microsoft's Virtual PC, and VMware are examples of such containers. This sort of file system's main advantage is that it is centralised and simple to remove. Even though a single-file virtual file system may have all the essential characteristics of any file system (virtual or otherwise), access to these file systems' internal structures is sometimes restricted to programmes created particularly to use single-file virtual file systems (instead of implementation through a driver allowing universal access). Performance is comparatively poor compared to other virtual file systems, which is another significant issue. The expense of moving virtual files around when data is written to or deleted from the virtual file system is the main cause of low performance.

4) *Implementation of Single-file Virtual File Systems:* Direct examples of single-file virtual file systems include emulators, such as PC Task and Win UAE, which encapsulate not only the file system data but also emulated disk layout. This makes it simple to transfer an OS installation using removable media or over a network, just like you would any other piece of software.

5) *PC Task:* A 4.77MHz Intel PC 8088-based system was simulated by the Amiga emulator PC Task (and later an 80486SX clocked at 25 MHz).Users of PC Task could create a file of large size on the Amiga file system, and this file would be virtually accessed from the emulator as if it were a real PC Hard Disk. The file could be formatted with the FAT16 file system to store normal MS-DOS or Windows files.

6) *Win UAE:* Large single files on Windows can be processed as Amiga file systems thanks to the UAE for Windows, Win UAE. In WinUAE this file is called a hardfile. UAE could also treat a directory on the host file system -- (Windows, Linux, Mac OS, Amiga OS) as an Amiga file system.

### *C. Disk Partitioning*

After Disk detection and its initialization, disk partitioning occurs which is another important process to make the file system of the given hard disk executable. The disk space assigned to the kernel is limited due to some testing reasons. The space allotted to it is 10 GB, which is detected in boot screen and we can create partition using keyboard. We can specify the space allotted. Partition can be divided into two types which are logical and physical partitions. Disk partitioning is the process of separating a hard disc drive (HDD) into various logical storage units, often known as partitions, in order to treat a single physical disc drive as though it were several discs and to enable the usage of various file systems on each partition. These partitions on the HDD can be created, resized, deleted, and otherwise modified using a software tool called a partition editor. A partition consists of several HDD cylinders, i.e. There are start and end cylinders for each partition (the size of cylinders varying from disc to disk). The disc drive that is connected to it can be partitioned using the following code.

### *D. Disk Formatting*

The process of setting up a data storage device, such as a hard drive, solid-state drive, floppy disc, or USB flash drive, for the first time is known as formatting. The formatting operation could occasionally also result in the creation of one or more new file systems. Low-level formatting is a term used to describe the first step of the formatting process, which prepares the fundamental medium. The second step of the process, which makes the data storage device visible to an operating system, is known as partitioning. The third step, which is frequently referred to as "high-level formatting," involves creating a new file system. The term "format" is believed to denote an operation in which a new disc media is fully prepared to store files. In some operating systems, all or portions of these three operations can be merged or repeated at different levels. Typically, formatting a disc leaves most of the data on the disc media, if not all of it; some or most of this data may be recovered with specialised tools. By overwriting all files and available space in one operation, special tools can delete all user data.

## IV. KERNEL LEVEL AUTHENTICATION

A technique for identifying a specific user using credentials provided by the operating system of the user's computer is known as operating system (OS) authentication. The OS password or digital certificates stored on the user's PC can serve as these credentials.
Possible benefits of using OS authentication

1) You do not have to keep track of multiple user names and passwords; if the login to your computer is successful, you do not have to enter another user name and password to connect to the database.
2) The database administrator (DBA) does not have to keep track of password changes, since that is changed on each user's computer.
3) Possible drawbacks of using OS authentication
4) Using operating system authentication with certain database products (those that don't use digital certificates in addition to user name and password) could be an increased security risk; When an OS account's password is discovered, access is given without the additional security provided by a distinct database account.
5) Additional configuration in the database may be needed to support OS authentication.

## V. DISPLAY DRIVER INTERFACE

An interface function between a microprocessor, microcontroller, ASIC, or general-purpose peripheral interface and a specific type of display device, such as an LCD, LED, OLED, e-paper, CRT, Vacuum fluorescent, or Nixie, is provided by a display driver in electronics and computer hardware. A display driver may also comprise a state machine made of discrete logic and other components. In order to make the display show the desired text or image, the display driver will typically accept commands and data using an industry-standard general-purpose serial or parallel interface, such as TTL, CMOS, RS232, SPI, I2C, etc. and generate signals with suitable voltage, current, timing, and demultiplexing.
The display driver may be a microcontroller designed specifically for the application and may include RAM, Flash memory, EEPROM, and/or ROM. Firmware and display fonts may be present in fixed ROM. The Hitachi HD44780 LCD controller is a well-known illustration of a display driver IC.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

### A. *Screen Drivers*

Screen Drivers are used to provide interface that provides us the screen display with resolutions, different algorithms and line, rectangle, and pixel mapping technique.

### B. *Drivers*

1) *CosmosVGA:* CosmosVGA is the display driver which is default display drivers provided by kit. This driver helps us to boot kernel on X-86 machine.
2) *VMWareSVGA:* The *VMware SVGA II* device is the virtual graphics card implemented by all VMware virtualization products. This is the graphics card that your virtual machine sees if it is running on VMware Workstation, Fusion, Player, or ESX. There are no physical graphics cards that resemble this one in exactly the same way. It resembles a physical device in many respects, but its programming paradigm has also been idealised or altered in other ways to make it simpler to replicate. The physical graphics hardware in your computer can be used by VMware's desktop virtualization products (Fusion, Workstation, and Player) to implement acceleration for the VMware SVGA device. Currently it supports a small amount of 2D acceleration, cursor acceleration, video overlay support, and 3D graphics with Shader Model 2.0. All of this acceleration is provided via a virtualized graphics interface, so the same drivers in the VM work regardless of what physical graphics card, if any, is physically available. This paper is a package of developer-oriented documentation for the details of this virtualized graphics interface. It consists of some basic documentation, as well as a package of example programs which demonstrate how to draw 2D and 3D graphics inside a virtual machine. Without an OS or any other graphics drivers loaded, these examples operate on the (virtual) bare metal. For instructional reasons and for those creating 3D drivers, this package is made available. Hobbyist OS writers and developers who want to test out low-level 3D graphics without having to deal with the challenging programming interface of a physical GPU may find it particularly intriguing. If you're developing typical user-level apps that you want to execute within a virtual machine, this code won't be of any assistance to you. It is intended for driver writers and presupposes some basic familiarity with graphics hardware

### C. *Handlers*

1) *CosmosVGA:* Cosmos VGA provides handlers for Cosmos VGA drivers which are used to switch modes between text and graphics etc.
2) *VMWareSVGA:* VMWareSVGA also provides support for its legacy drivers i.e. VMWareSVGA Drivers which provides different line drawing algorithms and modes of view.

## VI. FILE MANAGEMENT INTERFACE

### A. *File Allocation Table*

A number of commercially accepted file systems using the File Allocation Table (FAT) design are known by this name. A traditional file system that is straightforward and reliable is FAT. [4] It can't provide the same performance, durability, and scalability as some contemporary file systems, but it still performs well even in lightweight implementations. The format is suitable for data exchange between computers and devices of almost any type and age from the early 1980s (when it was introduced) up to the present because it is supported for compatibility reasons by almost all currently developed operating systems for personal computers, many mobile devices, and embedded systems. FAT was initially developed in the late 1970s for use on floppy discs, but it was quickly modified and throughout the next two decades of the DOS and Windows 9x operating systems, it was almost exclusively used on hard discs. FAT is no longer the default file system for use on Microsoft Windows PCs due to the emergence of more powerful machines and operating systems, as well as the creation of more complex file systems for them. [5] Today, floppy discs, USB sticks, flash and other solid-state memory cards and modules, as well as numerous portable and embedded devices, still frequently use FAT file systems. As the default file system for digital cameras, DCF uses FAT. In the boot process of EFI-compliant systems, FAT is also used. The FAT index table, which was statically created at the time of formatting and is used frequently by the file system, is where the name of the file system comes from. Each cluster, or continuous region of disc storage, has an entry in the table. Each entry has a marker signifying the end of the file, free disc space, or specific reserved disc locations, or it contains the number of the subsequent cluster in the file. The operating system can then navigate the FAT table, seeking up the cluster number of each subsequent part of the disc file as a cluster chain until the end of the file is reached. The root directory of the disc holds the number of the first cluster of each file in that directory.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

1) *FAT Handlers:* Different file system-related functions are offered by FAT handlers.
2) *FAT Support:* The FAT index table, which was statically created at the time of formatting and is used frequently by the file system, is where the name of the file system comes from. Each cluster, or continuous region of disc storage, has an entry in the table. Each entry has a marker signifying the end of the file, free disc space, or specific reserved disc locations, or it contains the number of the subsequent cluster in the file.

### *B. Virtual File System*

A more abstract file system called a virtual file system (VFS) or virtual file system switch sits on top of a more tangible file system. A VFS's main function is to give client applications uniform access to many concrete file system types. For instance, a VFS can be used to transparently access both local and network storage devices without the client application realising. Applications can access files on local file systems of those sorts without needing to be aware of the type of file system they are accessing thanks to this method of bridging the gaps between Windows, Mac OS, and Unix file systems. An interface (or "contract") between the kernel and a certain file system is specified by a VFS. As a result, by merely upholding the contract, it is simple to add support for new file system types to the kernel. The operating system supplier might only make backward-compatible changes to the contract, ensuring that concrete file system support built for a given release of the operating system would work with future versions of the operating system, or the contract's terms could change inconsistently over time, necessitating the need for concrete file system support to be recompiled and possibly modified before recompilation. Sun Microsystems released SunOS 2.0 in 1985, which featured one of the earliest virtual file system techniques on Unix-like systems. It enabled transparent access to both nearby UFS file systems and distant NFS file systems for Unix system calls. This is why Sun's VFS design was frequently imitated by Unix companies who licenced Sun's NFS code. A SunOS VFS implementation of the MS-DOS FAT file system was created, but it wasn't released as a product until SunOS 4.1. Other file systems may also be plugged into it. The VFS mechanism in System V Release 4 was built on the SunOS implementation. Under SunOS 4.0, John Heidemann created a stacking VFS for the unfinished Ficus file system. With different but similar semantics, this approach allowed for code reuse across file system kinds (e.g., an encrypting file system could reuse all of the naming and storage-management code of a non-encrypting file system). As part of his thesis research, Heidemann modified this work for use in 4.4BSD; successors of this code underpin the file system implementations in contemporary BSD derivatives, including Mac OS X. The File System Switch in System V Release 3, the Generic File System in Ultrix, and the VFS in Linux are additional virtual file systems for Unix. In OS/2 and Microsoft Windows, the virtual file system mechanism is called the Installable File System. The File system in User space (FUSE) mechanism allows user land code to plug into the virtual file system mechanism in Linux, Net BSD, FreeBSD, Open Solaris, and Mac OS X. In Microsoft Windows, virtual file systems can also be implemented through user land Shell namespace extensions; however, they do not support the lowest-level file system access application programming interfaces in Windows, so File systems that are implemented as namespace extensions won't be accessible by all programmes. Although they can be modified to use FUSE techniques and therefore seamlessly integrate into the system, KIO and GVFS/GIO provide comparable mechanisms in the KDE and GNOME desktop environments, respectively, with comparable limitations.

## VII. MOUSE DRIVER INTERFACE

Pointing devices (mouse or touch-pad) play a key role in user interaction with GUI. They provide point-and-click mechanism to eliminate use of keyboard for a large number of tasks. However, to make them work in a computer, we need special software to communicate with the device. This special software is known as device driver, as it “drives” the hardware properly. Operating system interacts with the driver, and all the low level communication is handled by the driver.

### *A. System Architecture*

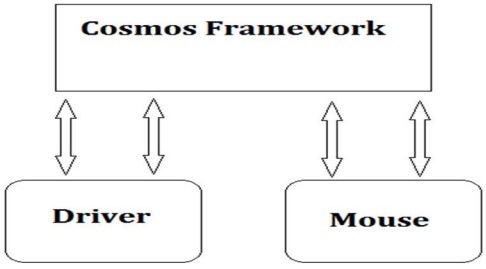



Figure 7-1

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 10 Issue VIII Aug 2022- Available at www.ijraset.com*

## VIII. KEYBOARD DRIVER INTERFACE

The PS/2 Keyboard uses serial communication to interact with a PS/2 controller. The PS/2 Keyboard driver would use this interface without being concerned with lower level details (such what type of PS/2 controller the device is plugged into), and each different type of PS/2 controller driver should ideally provide some sort of standard/simple "send byte/receive byte" interface. The PS/2 Keyboard can receive orders, respond to those actions with messages, and provide scan codes that show when a key has been depressed or released. The next step is to keep track of which keys are currently being pressed after you have key codes. Consider a computer game where the "WASD" keys are used to control player movement; pressing the "A" key causes the player to rotate in a clockwise direction. How does the game know if the "A" key is being used right now? You'd need a flag array for this, with each flag denoting a different key code. The keyboard driver itself can utilise the same "array of flags" to identify whether a shift key, control key, alt key, etc. is down, which can be quite handy when attempting to translate the key code into an actual ASCII character or Unicode code point. This is a hidden benefit. For instance, depending on the status of caps lock and whether or not a shift key is being held down at the moment, pressing the "a" key can result in the character "a," "A," or even nothing at all (e.g. "control-a" or "alt-a"). Additionally, keep in mind that (for instance) a "WASD" game doesn't mind if the keys are "W," "A," "S," and "D." The "T-shaped" arrangement of keys on the left side of the keyboard is what the game is interested in. The keys in the same area could be completely different if the keyboard is something else. (e.g. they would be '<', 'A', 'O' and 'E' keys on a Dvorak keyboard). This helps to explain my preference of having an 8-bit key code where the highest 3 bits determine which row on the keyboard and the lowest 5 bits determine which column (it's easy for a game to ask about the state of the third key on the left of the third row). Once you're able to keep track of which keys are currently being pressed, the next step is to (attempt to) convert the key into an ASCII character or Unicode code point. At this point you need to know what type of keyboard the user has - is it "US QWERTY", or "French AZERTY", some form of Dvorak, or perhaps it's Arabic. To handle many different keyboard layouts, the keyboard driver needs to use tables to convert key codes into ASCII characters or Unicode code points; so that you only need to load a different "Key Mapping" table to support different keyboard layouts. However, it's not quite that simple. Different keyboard layouts can have different meta keys, different status LEDs, etc. The Key Mapping table has to sort these differences out too. This is why you don't want to detect if the keyboard LEDs have changed earlier, but want to send the "set LEDs" command (if necessary) after you've found the entry for the key code in your key map table. The final step of processing is to combine all relevant information into some sort of "keypress packet" structure, and send it to whomever (e.g. GUI). The entire "keypress packet" might include the following:

*1)* Unicode code point (if applicable)
*2)* Key code
*3)* Pressed/released flag
*4)* Various other key states (shift, alt, control, etc)
*5)* Various "toggle" states (CapsLock, ScrollLock, NumberLock, etc.)

## IX. FONT HANDLING

A computing industry standard called Unicode ensures that text expressed in the majority of the world's writing systems is consistently encoded, represented, and handled. The most recent version of Unicode comprises a repertoire of more than 110,000 characters representing 100 scripts and numerous symbols. It was created in conjunction with the Universal Character Set standard and published in book form as The Unicode Standard. Text expressed in the majority of the world's writing systems is reliably encoded, represented, and handled thanks to a computing industry standard known as Unicode. More than 110,000 characters representing 100 scripts and countless symbols can be found in the most recent version of Unicode. The Unicode Standard, a book containing both its creation and the Universal Character Set standard, was released..[1] As of September 2013, the most recent version is Unicode 6.3. The standard is maintained by the Unicode Consortium. Unicode's success at unifying character sets has led to its widespread and predominant use in the internationalization and localization of computer software. The standard has been implemented in many recent technologies, including modern operating systems, XML, the Java programming language, and the Microsoft .NET Framework. Different character encodings can be used to implement Unicode. The most widely used encodings are UTF-8, UTF-16, and UCS-2, which is no longer in use. Since all ASCII characters have the same code values in both the UTF-8 and ASCII encodings, UTF-8 takes one byte for any ASCII characters and up to four bytes for all other characters. Each character in the current Unicode standard cannot be encoded using UCS-2's 16-bit code unit (two 8-bit bytes). By utilising two 16-bit units (4 8 bit) to handle each additional character, UTF-16 expands UCS-2 by employing one 16-bit unit for the characters that UCS-2 could represent.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

## X. CONCLUSION

In this paper, we have achieved many requirements which are executed as per the kernel basis. We have received following snapshot as the output of our kernel bootable ISO which are as follow:

### *A. Snapshots*

#### *1) RAM Snapshot*




Figure 10-1

#### *2) File System Snapshot*

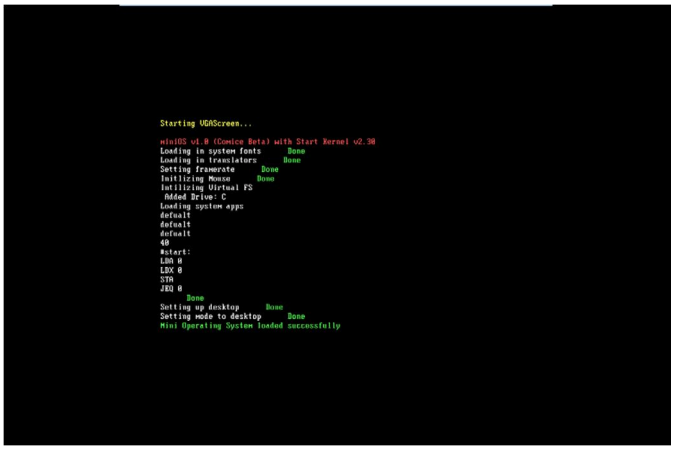



Figure 10-2

#### *3) Font Loading*




Figure 10-3

4) *Authentication Process*




Figure 10-4

5) *Hard Disk Detection and Partitioning*

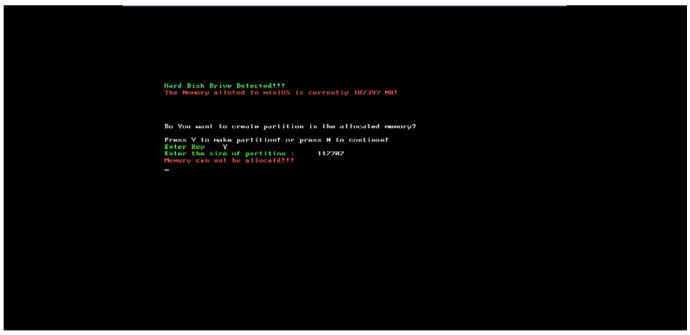



Figure 10-5

6) *Home Screen*

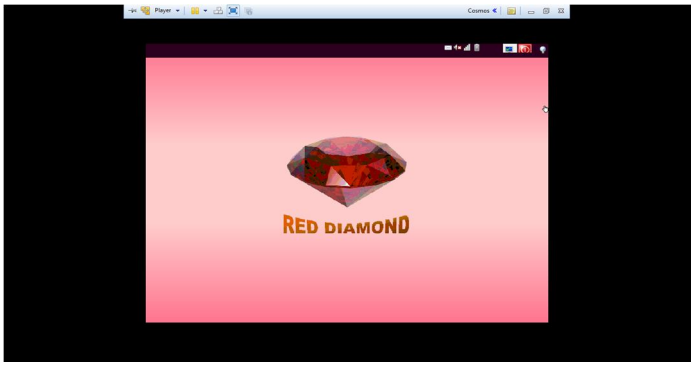



Figure 10-6

7) *Status Bar and Task Bar*




Figure 10-7

*8) Resolutions List and Events*

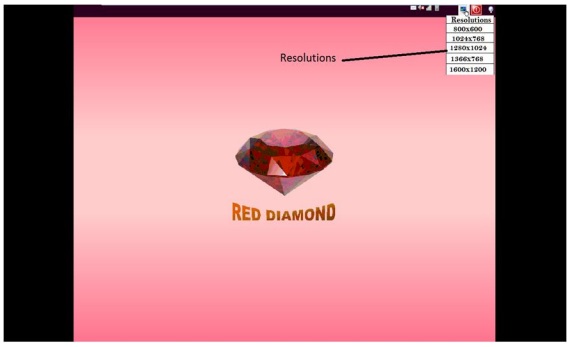



Figure 10-8

*9) Window Application Rendering on Click event*

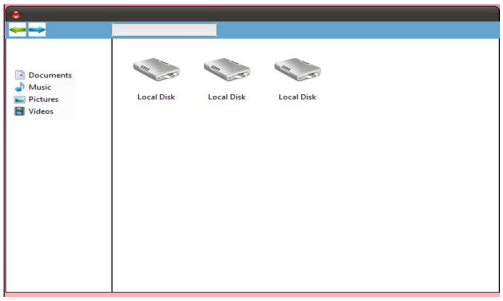



Figure 10-9

*10) Icon Selection on Hovering the pointer*



The icons background is blue when it is selected.

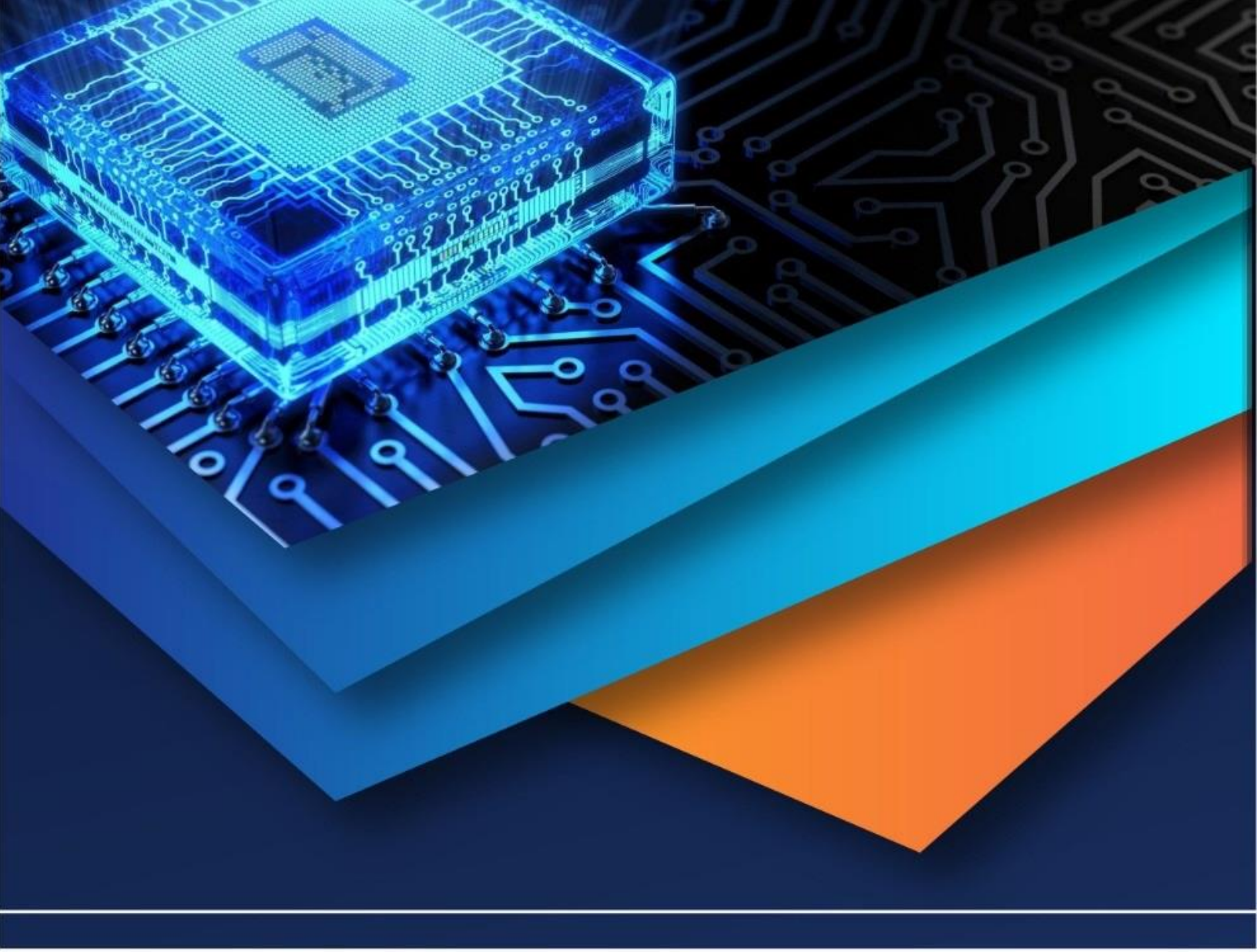


The Icon remains intact when mouse is not hovered on it.

Figure 10-10

## REFERENCES

[1] Support for visual studio and COSMOS dev. kit from the developers of Microsoft Organization and http://www.microsoft.com.
[2] Singularity OS : basic prototype and structure .Source code is available at http://singularity.codeplex.com/SourceControl/changeset/view/45126
[3] Managed code OS from http://www.mosa-project.org/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)