Module Code & Module Title

Level 5

Assessment Type

Logbook

Semester

2023/24 Spring/Autumn

Student Name: Arij Lamichhane

London Met ID: 22085885

College ID: NP04CP4S230014

Assignment Due Date: 30/09/2024

Assignment Submission Date: 28/09/2024

Submitted To: Mr. Prashant Adhikari

<u>Table of Figures</u>

Kernel is a small piece of code that is loaded into the memory when the computer boots. Kernel is the most common term for the core of operating system. This computer code contains instructions that allows the kernel to manage hardware devices, memory allocation, system processes, and other programs.

As OS became more prominent and more complicated, interest in rational segmentation of the program grew. Comprehensive OS functions and user support will be built on top of this skeletal software base. The kernel provides all else on the machine with critical facilities and defines many of the features of higher applications. Thus, as a synonym for "kernel," we also use the word "operating system (OS). In a modern general-purpose machine, the operating system kernel has the highest degree of privilege. The kernel governs how scarce resources such as CPU running time and physical memory pages are used by processes. on the device and arbitrates access to protected hardware, as shown in Fig. 1. The kernel is the component that allows a process on the system to access files, the network, or display configuration data. The Operating System has two primary functions: it essentially needs to be used as an extension machine. As a computer system manager, it has to handle and administer all sorts of tools reasonably. Furthermore, specific systems are responsible for protecting the computing system and offering application specific services like networking, graphical interface, etc. Amongst the most challenging aspects of research are security monitoring and ensuring that no new bugs have been implemented.

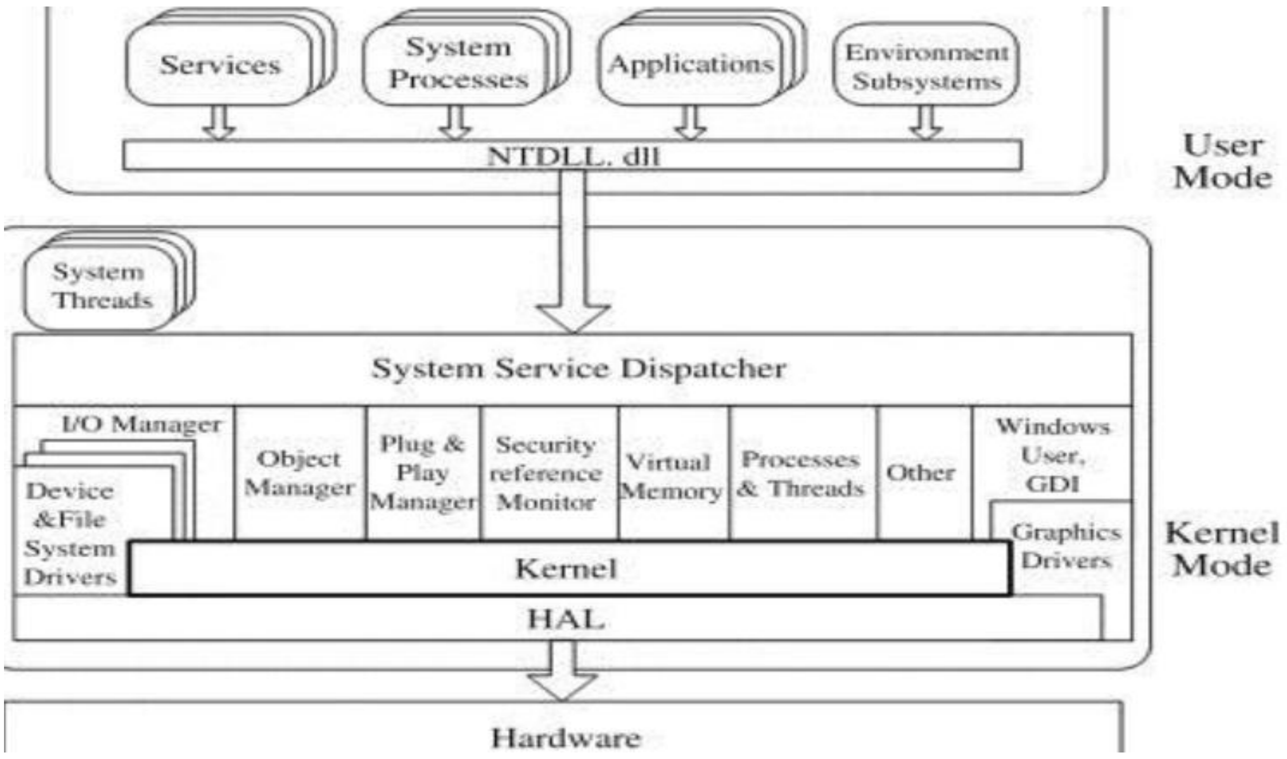

Fig: The abstract view of a kernel.

T**he major components of a kernel include:**

Process Management: This deals with process creation, process scheduling and the termination of the process. It enables the control of the CPU and enables scheduling of tasks.

Memory Management: It is responsible for controlling the memory of the system through the allocation of memory space to processes and cover-protection issues also deals with functions concerning the paging of memory spaces.

Device Management: The kernel communicates with the devices through an intermediate called drivers; this means it is possible to send commands to a printer, keyboard, or disk drives among others.

File System Management: This deals with receipt of data and transmission of data to storage media it offers files and directories for data organization.

The functions of kernel are as follows:

- Managing hardware resources, Controls hardware devices through device drivers.
- offers an interface through which programs can ask the kernel for services.
- Ensuring system security.
- Optimizing system performance.
- Handles the creation, scheduling, and termination of processes.
- Handling interrupts and exceptions.

Objectives of kernel:

Objectives of kernel are as follows:

Abstraction: Abstract hardware complexities, allowing application developers to focus on higher-level programming without needing to manage hardware directly.

Security and Protection: Implement security measures to protect system resources from unauthorized access and ensure data integrity through user permissions and access controls.

Stability and Reliability: Ensure that the system operates reliably and can recover from errors or crashes, maintaining user trust and system integrity.

Resource Management: Efficiently manage hardware resources such as CPU, memory, and Internet of devices to ensure optimal performance and utilization.

Types of Kernels:

A kernel is the core component of an operating system that manages system resources and communication between hardware and software. Different types of kernels can be categorized based on various bases for division. Let's explore two common bases: architecture and functionality.

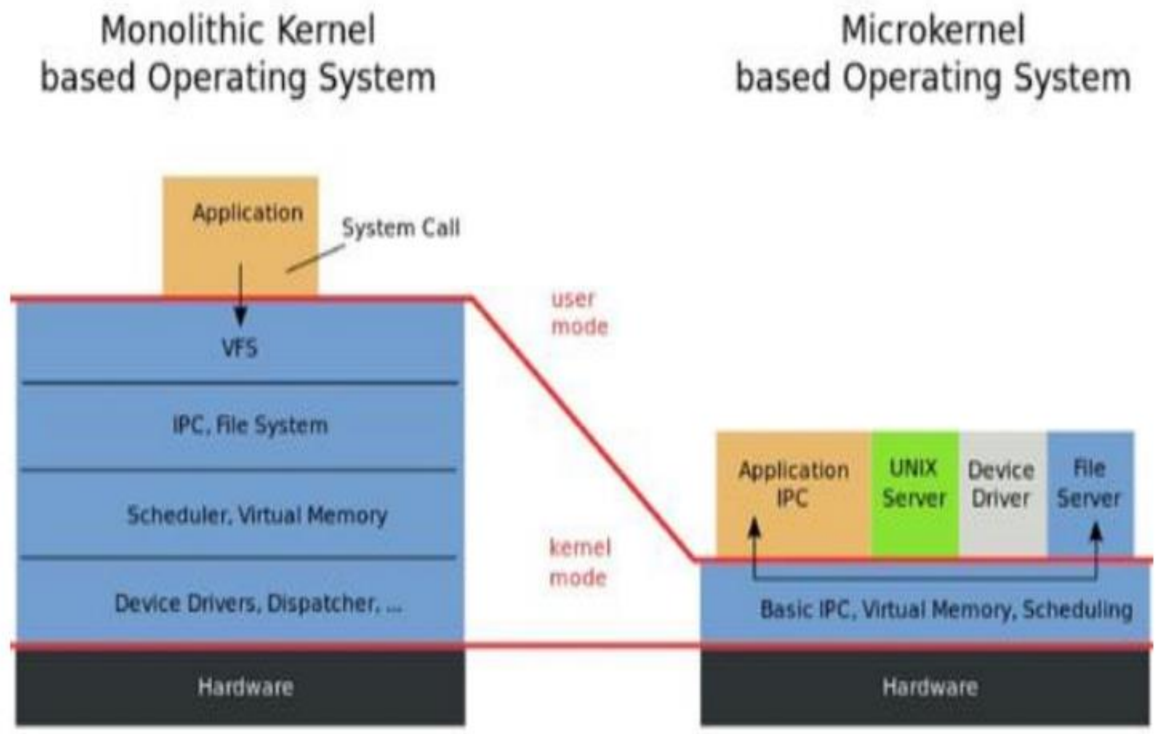
Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
user mode
kernel mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

Fig: The architecture of a Transform of Monolithic Kernel to Microkernel.

1.Based on Architecture

This classification considers the structure and design of the kernel, focusing on how the system's functionalities are managed.

a. Microkernel: Microkernels have minimal 'built-in' functions, scheduling, memory management, and IPC. True microkernels dictate only those fundamental services, for example, the sharing of data between the lower layers of hardware and higher layers of processes. Other programs that run on Linux also call the user space such as device drivers, file systems, and network protocols.

All other OS features are allocated to hardware userland drivers but controlled by the kernel. Most of these feature's help in making the system more steady and secure because kernel core is not affected by crashes to user spaces services as it contains application drivers and window managers and it can be coded and quickly substituted in languages not used by the kernel without changing the kernel. . Still, it might be a little slower in terms of performance since there can be high activity in the communication between the user space and the kernel.

Examples: QNX, MINIX, and Mach.

b. Monolithic Kernel: it has all the microkernel functionality plus hardware drivers, virtual file system (VFS) and protocol stacks. This reduces inter-process communication requirements and enhances system security but requires careful system design and reduces flexibility available to the designer. In a monolithic kernel, all the operating system services like memory management, process scheduling and input/ output operations are collectively located in a large kernel space.

All components execute in same address space, and therefore it is possible that the program executes faster because of fewer contextual switches. But it is buggy, because if one service fails, the entire system goes down. Examples: Linux, UNIX

2. Types of Kernels Based on Structure (Machine Learning Kernels)

Kernels can be classified according to their structure into so-called machine learning kernels. Kernel in the case of the machine learning especially in support vector machines

(SVMs) and kernel methods, can be divided in according to the function used to map the data into a higher space.

a.Windows OS – Hybrid Kernel: The Windows operating system uses a hybrid kernel, which combines aspects of both monolithic and microkernel designs. While it contains monolithic components, it also runs some services (such as device drivers and file systems) in user space, aiming for modularity without decreasing performance. The Advantages of Hybrid kernel is it aims to balance performance with modularity, allowing for efficiency and easy development.

The Disadvantage of Hybrid kernel is critics often claim that hybrid kernels fail to achieve best of either monolithic or microkernel design, as they are slightly less efficient than pure monolithic kernels and microkernels.

b. macOS (Apple os) - Monolithic with Microkernel Elements (XNU Kernel) :

Apple's macOS and iOS operating systems are based on the **XNU kernel**, which is a hybrid that mix a **monolithic** design with **microkernel** components. The XNU kernel was developed for NeXTSTEP and later adapted for macOS. It includes elements of the Mach microkernel for handling IPC (Inter-process Communication) and virtual memory, combined with components from the FreeBSD monolithic kernel for performance. Advantages: Also, similar to Windows, macOS kernel has the performance and modularity optimization as its goal. The improvement done by placing some services in user space is that the microkernel component enhances system stability and security. The disadvantage of mac's os kernel is Although XNU does incorporate both design paradigms, it remains vulnerable to the performance overhead as microkernels' relies on IPC for services.

Operating systems rely on kernels, the core component that manages the hardware-software interface, resource allocation, and system operations. Three popular operating systems, iOS, Windows, and Ubuntu, each use distinct kernels that have evolved from different technological backgrounds.

2. Popular Kernels and Their History: iOS, Windows, and Ubuntu

Operating systems rely on kernels, the core component that manages the hardware-software interface, resource allocation, and system operations. Three popular operating systems, **iOS**, **Windows**, and **Ubuntu**, each use distinct kernels that have evolved from different technological backgrounds.

1. iOS - XNU Kernel (XNU)

Kernel Used:

- **XNU (X is Not Unix)** is the kernel used in **iOS**. Originally developed by NeXT for the NeXTSTEP operating system, XNU became part of Apple's operating systems after Apple acquired NeXT in 1996. It powers both iOS (for mobile devices) and macOS (for desktop and laptop computers).

History:

- **NeXTSTEP (1980s)**: XNU was originally developed for NeXTSTEP, which was built upon the Mach microkernel developed at Carnegie Mellon University.
- **Apple Acquisition**: In 1996, Apple acquired NeXT and integrated NeXTSTEP into the development of macOS. XNU was modified to integrate the **Mach microkernel** and elements from **BSD Unix** (Berkeley Software Distribution).
- **Evolution in iOS**: With the release of iOS in 2007, Apple adapted XNU for mobile devices, optimizing it for lower power consumption and enhanced security.

Key Features:

- **Hybrid Kernel**: XNU combines the efficiency of a microkernel (Mach) and the functionality of a monolithic kernel (BSD Unix), leveraging the benefits of both types.
- **Security**: XNU provides robust security measures, including sandboxing and code signing, which are critical for mobile devices.
- **Performance**: Optimized for the mobile environment, XNU efficiently manages system resources on devices with limited hardware capabilities.

2. Windows - NT Kernel

Kernel Used:

- **NT Kernel** is the core component of **Windows** operating systems. First introduced with **Windows NT** in 1993, it is now used in all major Windows operating systems, including **Windows 10** and **Windows 11**.

History:

- **Development of NT (1990s)**: The NT kernel was designed by a team led by Dave Cutler, who previously worked on VMS at Digital Equipment Corporation (DEC). The goal was to create a more portable, scalable, and secure operating system.
- **Windows NT 3.1 (1993)**: This was the first release of the NT kernel, targeting enterprise environments with support for multiprocessor systems and enhanced security.
- **Evolution in Consumer OS**: Over time, Microsoft merged its consumer-oriented Windows 95/98 line with the more robust NT kernel, starting with **Windows XP** in 2001. Since then, the NT kernel has powered all Windows operating systems.

Key Features:

- **Modular Architecture**: The NT kernel uses a layered architecture, allowing it to easily support different subsystems and applications, including POSIX, OS/2, and Windows APIs.
- **Security and Stability**: NT was designed with a focus on security, supporting user-mode and kernel-mode operations, virtual memory management, which improves stability and reduces the risk of system crashes.
- **Portability**: Initially designed to be portable across different processor architectures (x86, MIPS, Alpha), the NT kernel now supports modern platforms, including x86-64 and ARM.

3. Ubuntu - Linux Kernel

Kernel Used:

- **Linux Kernel** is at the heart of **Ubuntu**, a popular distribution of the Linux operating system. The Linux kernel is an open-source, monolithic kernel developed by Linus Torvalds in 1991.

History:

- **Linus Torvalds' Development (1991)**: The Linux kernel was created by Linus Torvalds as a free alternative to the proprietary MINIX operating system, which was used in academia.
- **Open-Source Community**: As Torvalds released the source code under the GPL (General Public License), the Linux kernel quickly grew through contributions from a global community of developers.
- **Ubuntu (2004)**: Ubuntu was created by Canonical in 2004, using the Linux kernel as its core. It is based on Debian, another Linux distribution, and aims to be user-friendly and accessible to a broader audience.

Key Features:

- **Monolithic Kernel**: Unlike XNU's hybrid model, the Linux kernel is monolithic, meaning it includes device drivers, memory management, file systems, and network stacks all within the kernel space.
- **Customizability**: The open-source nature of Linux allows users and organizations to customize and modify the kernel to suit specific needs.
- **Security and Flexibility**: The Linux kernel has a reputation for its robust security model, with features like mandatory access controls, namespaces, and capabilities to restrict privileged actions.

Boot Process:

The boot process is a crucial phase in the operation of a computer system, as it is responsible for loading the operating system (OS) from a non-volatile storage device into the system's memory. This process sets the stage for the system to execute higher-level operations. In modern computing, the boot process has evolved into a complex series of steps, encompassing initialization, hardware checks, and OS loading.

Steps in the Boot Process:

1.Power On and BIOS/UEFI Execution:

When a system is powered on, the **BIOS (Basic Input/Output System)** or **UEFI (Unified Extensible Firmware Interface)** is the first code to run It is planted in ROM (Read-Only Memory) or flash memory.

The BIOS/UEFI activate hardware components like the CPU, memory, and input/output devices. It also performs the **POST (Power-On Self-Test)**, which ensures that the system's hardware is functioning correctly.

BIOS/UEFI Role:

BIOS was the old firmware interface while UEFI being the latest interface, secure, faster boot, and efficient for large storage devices.

The BIOS or UEFI firmware identifies the bootable devices and selects the appropriate one (such as a hard disk, SSD, or USB drive) from which the OS will be loaded.

2. Boot Loader Execution:

When the BIOS identified the bootable device, it loads an independent program called the boot loader into the memory. The boot loader is responsible for loading the OS kernel.

Famous boot loaders include **GRUB (GNU GRand Unified Bootloader)** in Linux systems and **Windows Boot Manager** for Windows OS.

Key Functions:

- The boot loader provides a user interface, allowing the user to select from multiple installed operating systems (if applicable).
- It prepares the system for loading the OS kernel by setting up essential parameters like memory mapping.

3.OS Kernel Loading:

- The boot loader locates the **kernel** (the core of the operating system) and loads it into memory.
- The kernel is responsible for managing system resources, hardware communication, and providing a platform for running applications.

Kernel Initialization:

- Once the kernel is loaded, it begins hardware drivers, memory management, and other key system functions.
- The kernel sets up interrupt handlers and launches system processes that handle basic operations like input/output, networking, and file systems.

4. User-Space Initialization:

- After the kernel initialization, the boot process moves to **user-space initialization**. The kernel starts an initial process, such as `init` (in Unix-based systems) or `system` (in modern Linux distributions).
- This process is responsible for launching user-space services, daemons, and the graphical user interface (GUI) or command-line interface (CLI) through which the user interacts with the system.

5. **Login and User Interaction**:

a. Once all system services are up and running, the OS presents a **login screen** (if applicable) or a command prompt, allowing the user to access the system.

b. At this stage, the computer is fully operational, and the user can begin interacting with it.

6. Conclusions:

Modern operating systems are built on kernels, which are responsible for controlling process control, system calls, and hardware resources. Based on certain design ideas, every major OS kernel, including Linux (monolithic), iOS (XNU), and Windows (NT), has developed. While Linux provides openness and flexibility, Windows NT emphasizes portability and modularity. XNU, on the other hand, combines BSD and Mach to create a unique hybrid system that runs on both macOS and iOS.

These kernels' shared objectives of resource management and system stability are evident despite their disparities. Although there are alternatives, like microkernels, they frequently have trade-offs in terms of complexity and performance. As the foundation of operating systems, kernels are still crucial for ensuring that hardware and software interact properly and are continuously improved to suit the demands of modern computing.

7.References:

- Singh, A. (2006). *Mac OS X Internals: A Systems Approach*. Addison-Wesley Professional.
- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows Internals, Part 1*. Microsoft Press.
- Bovet, D. P., & Cesati, M. (2005). *Understanding the Linux Kernel*. O'Reilly Media.
- Custer, H. R. (1993). *Inside Windows NT*. Microsoft Press.
- McKusick, M. K., Neville-Neil, G. V., & Watson, R. N. M. (2014). *The Design and Implementation of the FreeBSD Operating System*. Addison-Wesley.