LONDON METROPOLITAN UNIVERSITY

Module Code & Module Title

Level 5 – CT5052NP

Assessment Type

Logbook 1.

Semester

2023/24 Spring/Autumn

Student Name: Subanta Poudel

London Met ID: 20048736

College ID: NP04CP4S210115

Assignment Due Date: September 30, 2024

Assignment Submission Date: September 30, 2024

Submitted To: Prasant Adhikari

Word Count (Where Required): 1200

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Contents

Introduction

According to Tanenbaum, “The Kernel is the part of the operating system that is always resident in memory and facilitates interaction between the software and hardware.” The kernel provides basic mechanisms and services on which other parts of an operating system rely. The kernel performs essential tasks such as device management, memory management, process scheduling and providing system calls for application software to interact with hardware resources. (Tanenbaum & Bos, 2020)

Stalling defines the kernel as the “the portion of the operating system that includes the most fundamental, low-level services, such as managing processes, memory, and I/O devices.” It acts as an intermediary between applications and the physical hardware of the system. The kernel is responsible for managing critical functions like process control and communication, and the extended kernel, which may include device drivers and system services. (Stallings, 2018)

According to authors of Operating System Concepts “The central component of an operating system that manages operations of the computer and hardware, most notably memory and CPU time, while maintaining security and ensuring resource allocation.” They are loaded into memory and functions as the core controller of system activities. The kernel controls the execution of all processes in the system, manages hardware resources, and offers system calls to allow software to interact with the hardware. (Silberschatz, Galvin , & Gange, 2020)

Objective

The objective of this log is to explore the structure, types, history, and processes related to kernels, providing and overview of their functionality in modern operating systems.

Types of Kernels

Kernels can be categorized based on structure and their managing system resources. Structural design and functionality.

1. Types of Kernels Based on Structural Design
   a. Monolithic Kernel

      A monolithic kernel is a type of kernel where the entire operating system operates in a single layer within the kernel space. However, this also increases the risk of system crashes, as errors in any module can affect the entire system. Monolithic kernels have better performance than microkernels because there is less overhead in terms of system calls, although they are more prone to system crashes due to their lack of isolation between system services. (Tanenbaum & Bos, 2020). Examples: Linux, UNIX.

   b. Microkernel

      Microkernel aims to keep only the essential functions in kernel space, such as inter-process communication (IPC), basic scheduling, and minimal memory management. Other services like device drivers, file systems, and networking are handled in user space as separate processes. Microkernels can provide better security and reliability by running services in user space, which reduces the risk of a single failure affecting the whole system, but may introduce performance costs due to frequent user-to-kernel space transitions. (Stallings, 2018). Examples: MINIX, QNX.

2. Types of Kernels Based on Functionality
   a. Hybrid Kernel

      Hybrid kernels are a combination of both monolithic and microkernel designs. However, they also suggest that this structure can sometimes inherit the complexities of both architectures. (Silberschatz, Galvin , & Gange, 2020). Examples: Windows NT, macOS (XNU).

   b. Exokernel

      Exokernel provides minimal abstractions and gives applications and gives application developers more direct control over hardware resources themselves. This can lead to highly efficient resource use, but it increases the complexity of application development. (Tanenbaum & Bos, 2020). Examples: MIT’s Exokernel and Nemesis OS.

Popular Kernels and their History

Operating systems like IOS, Windows, and Ubuntu are built on distinct kernels that have evolved over time to provide enhanced performance, security, and functionality. Below is an exploration of the kernels used by IOS, Windows, and Ubuntu, as well as their evolution.

1. IOS (XNU Kernel)
   The kernel used by IOS is called XNU, which stands for "X is not Unix." XNU is a hybrid kernel that combines elements of the Mach microkernel with components form BSD Unix. Which was initially developed by NeXT, a company founded by Steve Jobs after he departed from Apple. When Apple acquired NeXT in 1996, the XNU kernel became the foundation for macOS and, later, IOS.

- Mach Microkernel: XNU incorporates the Mach microkernel, which was developed at Carnegie Mellon University in the 1980s. Mach provides core kernel services such as low-level memory management and inter-process communication (IPC).
- BSD Unix Components: XNU also integrates components from the BSD Unix operating system, which provides features such as process control, file systems, and networking capabilities. This integration gives XNU the advantages of Unix's stability and maturity.

XNU's hybrid nature allows Apple to leverage the modularity and efficiency of Mach while retaining the robust features of BSD Unix, making it suitable for a wide range of devices. (Stallings, 2018)

2. Windows (NT Kernel)
   The Windows NT kernel, first introduced in the early 1990s, is a hybrid kernel that incorporates elements of both monolithic and microkernel architectures.

   - Hybrid Design
   - Evolution

The hybrid kernel approach allows Windows NT to achieve high performance while maintaining modularity and stability, which is crucial for enterprise-level reliability. (Tanenbaum & Bos, 2020).

3. Ubuntu (Linux Kernel)

Ubuntu, a popular Linux distribution, is based on the Linux kernel, which was first developed by Linus Torvalds in 1991. The Linux kernel is a monolithic kernel, meaning that all core operating system services run in kernel space.

- Open-Source Development:
- Monolithic Kernel:
- Evolution:

The role of the Linux kernel in modern operating system ecosystem, nothing its adaptability and efficiency in a variety of computing environments. (Silberschatz, Galvin , & Gange, 2020).

Boost Process

The boot process is a sequence of steps occurred when computers are powered on and operating system is loaded into memory, enabling the management of hardware resources and execution of applications. (Tanenbaum & Bos, 2020).

Step 1: Power-On and BIOS/UEFI Initialization

- BIOS/UEFI initialization: System's firmware initializes key hardware components, including CPU, RAM, Storage and Peripheral devices. Also conducts Power-On Self-Test
- POST: they check components like RAM, keyboard, and storage drives. If any components fail, it halts the boot process, after displaying and error message or beep code.
- Boot Device Selection: After successful POST, it identifies the boot device and hand overs the control to the next phase.

BIOS or UEFI acts the critical intermediary between the hardware initialization and the operating system, enabling the system to detect and prepare necessary components before the kernel takes control. (Tanenbaum & Bos, 2020)

Step 2: Bootloader

After BIOS/UEFI has initialized and determined the boot device,

- GRUB
- Loading the Kernel
- Initial RAM Disk

The bootloader acts as a bridge between the low-level firmware (BIOS/UEFI) and the operating system, ensuring that the kernel and necessary drivers are available in memory for the kernel initialization phase. (Stallings, 2018).

Step 3: Kernel Initialization

Once the bootloader has successfully loaded the kernel into memory, the kernel begins its initialization phase

- Memory Management Setup
- Process Management
- Device Drivers Initialization
- Interrupt Handling

Kernel initialization is the point at which the operating system takes full control of the system, configuring memory, processes, and devices in preparation for the user environment. (Tanenbaum & Bos, 2020)

Step 4: Init/Systemd

After the kernel has completed its initialization, it launches the first user-space process.

- Init
- Systemd

That init and systemd serve as the final stages of the boot process, transitioning the system from the kernel's hardware management to user-space operations, enabling the system to be fully functional for the user. (Silberschatz, Galvin , & Gange, 2020).

Conclusion

The boot process is structured and critical sequence of operations that transits a computer system from a powered-off state to a fully operational state. From the initialization of hardware components by BIOS/UEFI to loading execution of the kernel and user-space processes like init or system, each step is crucial for ensuring that the system boots efficiently and securely.

References

Silberschatz, A., Galvin , P. B., & Gange, G. (2020). *OperatingnSystem Concepts.* Wiley.

Stallings, W. (2018). *Operating Systems: Internals and Design Principles.* pearson.

Tanenbaum, A. S., & Bos, H. (2020). *Modern Operating Systems* (Vol. 4). Amsterdam: pearson.