Gabriel Gaincarlo

CPSC 380 Operating Systems

Homework #1

Part I – Introduction

1. Although there are many practitioners of computer science, only a small percentage of them will be involved in the creation or modification of an operating system. Why, then, study operating systems and how they work? (4 pts)

We study operating systems because they manage hardware and software resources, provide the foundation for all applications, and teach key concepts like memory management, scheduling, and security.

2. In early computer systems without Direct Memory Access (DMA), the CPU was responsible for managing all data transfers to and from memory. How might this design choice have influenced the efficiency and feasibility of running multiple programs concurrently? (4 pts)

Without DMA, the CPU had to do all the data transfers itself, which slowed everything down. Since it was stuck managing I/O, it didn't have the time or speed to handle multiple programs running smoothly at once.

3. Modern operating systems operate in at least two distinct privilege levels. What are those two levels, and describe the purpose of separating these levels and explain how this separation contributes to system stability and security. (4 pts)

The two levels are **user mode** and **kernel mode**. User mode runs regular programs with limited access, while kernel mode controls hardware and system resources. Separating them stops user programs from crashing or damaging the whole system, which keeps things stable and secure.

4. What is the program that is initially loaded to when the computer is started up, and where is it that program typically stored? (4 pts)

The program is the **bootstrap loader (bootloader)**, and it's usually stored in **ROM or firmware** on the motherboard.

5. Operating systems provide a controlled interface between user programs and hardware resources. What mechanism allows user-level applications to request services from the operating system, and why is this mechanism essential for system integrity and functionality? (3 pts)

The mechanism is a **system call**. It's essential because it lets programs safely request OS services without directly accessing hardware, which protects system integrity and keeps everything running properly.

6. Multiplexing refers to the technique of sharing a single resource among multiple users or processes, either by dividing access over time (time multiplexing) or by allocating separate portions of the resource (space multiplexing). For each of the following system resources—CPU, memory, SSD/disk, network card, printer, keyboard, and display—identify whether time multiplexing, space multiplexing, or both are used to enable sharing. (7 pts)

he CPU is shared through **time multiplexing**, while **memory** is shared through **space multiplexing**. An **SSD/disk** and a **network card** use **both** time and space multiplexing. A **printer** and **keyboard** are shared using **time multiplexing**, and the **display** uses **both**.

7. In some operating systems, a function in a user-level library shares its name with the underlying mechanism that interacts directly with the kernel. Is it necessary for these names to match? If not, which layer's naming convention plays a more critical role in system behavior and why (4 pts)

No, the names don't have to match. The **kernel's naming** is more critical, since it defines how the OS actually handles requests and ensures the system works correctly, while library names are more about convenience for programmers.

8. Can the count = write(fd, buffer, nbytes); call return any value in count other than nbytes? If so, why? (3 pts)

Yes, it can. `write` might return fewer bytes than requested if there's not enough space, if the operation gets interrupted, or if only part of the data could be written at that time.

9. Explain what cache coherence is and why it might cause a problem for multiprocessor systems. (3 pts)

Cache coherence means keeping copies of the same data in different caches consistent. In multiprocessor systems, if one CPU updates its cache but others don't see the change, they could work with outdated data, causing errors

10. There are a number of file system and I/O related system calls. List 5 such calls and briefly explain what each call does. (5 pts)

open() – Opens a file and returns a file descriptor.

read() – Reads data from a file into a buffer.

write() – Writes data from a buffer to a file.

close() – Closes a file descriptor, freeing resources.

lseek() – Moves the file pointer to a specific location in a file.

11. From a programmer's perspective, many operating system interactions appear to be ordinary function calls. Why might it be important to distinguish between calls that interact directly with the operating system and those that do not? (4 pts)

It's important because OS calls can be slower, may fail, or require special permissions, while regular function calls just run in your program. Knowing the difference helps write safe, efficient, and correct code.

12. What is the purpose of interrupts in an operating system? What are the differences between a trap and an interrupt? (4 pts)

Interrupts let the CPU stop what it's doing to handle urgent tasks, like I/O. A **trap** is a software-generated interrupt (like an error or system call), while an **interrupt** usually comes from hardware (like a keyboard or timer).

13. What is the purpose of a CLI or shell? List a couple common Linux shells names (3 pts)

A CLI or shell lets users interact with the operating system by typing commands. Common Linux shells include **bash** and **zsh**.

14. To maintain a secure boundary between user applications and the operating system, special mechanisms are used when transferring information between the two. What are three common strategies operating systems use to pass data between user-level programs and privileged system routine? (3 pts)

Three common strategies are:

1. **System calls** – Programs request services safely through the OS.
2. **Message passing** – Data is sent between user programs and the OS in controlled messages.
3. **Shared memory** – The OS allocates memory that both user programs and the OS can access safely.

15. Some operating system functions available in UNIX do not have direct counterparts in the Windows API. When adapting a UNIX-based program to run on Windows, what challenges might arise from the absence of these functions, and how could they affect the program's behavior or design? (5 pts)

Without direct counterparts, certain UNIX functions may not work on Windows, so the program might fail, behave differently, or need major changes. Developers may have to rewrite parts, use emulation layers, or find alternative APIs, which can affect performance and design.