1.4 Operating-System Operations

Now that we have discussed basic information about computer-system organization and architecture, we are ready to talk about operating systems. An operating system provides the environment within which programs are executed.

Internally, operating systems vary greatly, since they are organized along many different lines. There are, however, many commonalities, which we consider in this section.

Hadoop (Big Data in Clustered Systems)

- What is Hadoop?

“Hadoop is an open-source software framework”

Open-source → free, community-driven

Framework → provides tools + architecture, not just one program

- Purpose of Hadoop

“Used for distributed processing of large data sets (big data)”

Big data means:

Very large volume

Too big for a single machine

Needs parallel processing

Examples:

Logs

User activity data

Sensor data

1.4.1 Multiprogramming and Multitasking

One of the most important aspects of operating systems is the ability to run multiple programs, as a single program cannot, in general, keep either the CPU or the I/O devices busy at all times. Furthermore, users typically want to run more than one program at a time as well. Multiprogramming increases CPU utilization, as well as keeping users satisfied, by organizing programs so that the CPU always has one to execute. In a multiprogrammed system, a program in execution is termed a process.

Why do we need multiprogramming?

Problem:

A **single program**:

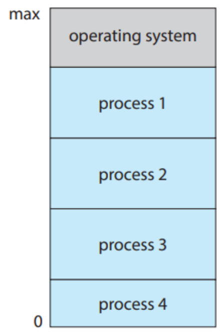
- Sometimes waits for I/O
- Sometimes waits for memory
- Sometimes waits for disk

Solution: Multiprogramming

Definition:

- **Multiprogramming** is the technique of keeping **multiple programs in memory** so that **the CPU always has something to execute**.

Multitasking is a logical extension of multiprogramming. In multitasking systems, the CPU executes multiple processes by switching among them, but the switches occur frequently, providing the user with a fast response time. Consider that when a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O. I/O may be interactive; that is, output goes to a display for the user, and input comes from a user keyboard, mouse, or touch screen



.Memory layout for a multiprogramming system

Multitasking Systems & Response Time

What is response time?

Response time =
⏱️ Time between **user request** and **system response**

Examples:

- Clicking a button
- Opening a file
- Switching between apps

In a multitasking system, **slow response = bad user experience**

So the OS must:

- Share CPU fairly
- Manage memory smartly
- Avoid blocking the whole system

Virtual Memory (Core Concept)

"A common method for ensuring reasonable response time is virtual memory."

What is Virtual Memory?

Virtual memory is a **memory-management technique** that allows a process to run **even if it is not fully loaded into RAM**.

Why Virtual Memory is needed

Problem without virtual memory:

- RAM is limited
- Programs are getting larger
- Only few programs could run at once

Solution:

- Load **only required parts** of a program into RAM

- Keep the rest on **disk (secondary storage)**

◆ Logical Memory vs Physical Memory

Logical Memory	**Physical Memory**
Seen by program	Actual RAM
Large & continuous	Limited & fragmented
Programmer-friendly	Hardware-controlled

1.4.2 Dual-Mode and Multimode Operation

Since the operating system and its users share the hardware and software resources of the computer system, a properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs —or the operating system itself— to execute incorrectly. In order to ensure the proper execution of the system, we must be able to distinguish between the execution of operating-system code and user-defined code. The approach taken by most computer systems is to provide hardware support that allows differentiation among various modes of execution. At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1). With the mode bit, we can distinguish between a task that is executed on behalf of the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode.

◆ **Why do we need protection at all?**

“The operating system and its users share hardware and software resources.”

This is dangerous if unmanaged.

1.4.3 Timer

We must ensure that the operating system maintains control over the CPU. We cannot allow a user program to get stuck in an infinite loop or to fail to call system services and never return control to the operating system. To accomplish this goal, we can use a timer. A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A variable timer is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs. For instance, a 10-bit counter with a 1-millisecond clock allows interrupts at intervals from 1 millisecond to 1,024 milliseconds, in steps of 1 millisecond.

LINUX TIMERS

On Linux systems, the kernel configuration parameter HZ specifies the frequency of timer interrupts. An HZ value of 250 means that the timer generates 250 interrupts per second, or one interrupt every 4 milliseconds. The value of HZ depends upon how the kernel is configured, as well the machine type and architecture on which it is running. A related kernel variable is jiffies, which represent the number of timer interrupts that have occurred since the system was booted. A programming project in Chapter 2 further explores timing in the Linux kernel.