Operating System – IDATA2305

**Content**

* **PART ONE – OVERVIEW**
* *1 – Introduction*
* *2 – Operating-System Structures*
* **PART TWO – PROCESS MANAGEMENT**
* *3 – Processes*
* *4 – Thread & Concurrency*
* *5 – CPU Scheduling*
* **PART THREE – PROCESS SYNCHRONIZATION**
* *6 – Synchronization Tools*
* *7 – Synchronization Examples*
* *8 – Deadlocks*
* **PART FOUR – MEMORY MANAGEMENT**
* *9 – Main Memory*
* *10 – Virtual Memory*
* **PART FIVE – STORAGE MANAGEMENT**
* *11 – Mass-Storage Structure*
* *12 – I/O Systems*
* **PART SIX – FILE SYSTEM**
* *13 – File-System Interface*
* *14 – File-System Implementation*
* *15 – File-System Internals*
* **PART SEVEN – SECURITY AND PROTECTION**
* *16 – Security*
* *17 – Protection*
* **PART EIGHT – ADVANCED TOPICS**
* *18 – Virtual Machines*
* *19 – Networks and Distributed Systems*

**PART ONE – OVERVIEW**

An operating system acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent programs from interfering with the proper operation of the system.

1 – Introduction

An operating system is a software that manages a computer’s hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. Operating systems are everywhere, from cars and home appliances that include “Internet of Things” devices, to smartphones, personal computers, enterprise computers, and cloud computing environments. In order to understand the role of a operating system it is important to understand the CPU, memory, and I/O devices, as well as storage. A fundamental responsibility of an operating system is to allocate these resources to programs.

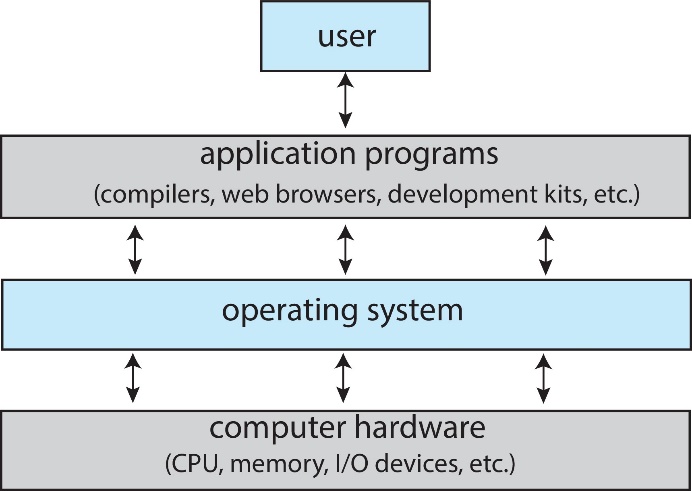
**Chapter Objectives**

* *Describe the general organization of a computer system and the role of interrupts.*
* *Describe the components in a modern multiprocessor computer system.*
* *Illustrate the transition from user mode to kernel mode.*
* *Discuss how operating systems are used in various computing environments.*
* *Provide examples of free and open-source operating systems.*

**What Operating Systems Do**

The computer system can be divided into four components: the ***hardware***, the ***operating*** ***system***, the ap***plication program***, and ***user***.

* The ***hardware*** – provides basic computing resources and consist of a CPU, also called central processing unit, memory, and I/O devices.
* The ***operating system*** controls the hardware and coordinates its use among the various applications programs for the various users. An operating system is like a government, because it has no useful function by itself, but simply provides an environment within which other programs can do useful work.
* The ***applications program*** defines the ways in which system resources are used to solve the computing problems of the users and are things as word processors, spreadsheets, compilers, and web browsers.
* The user is basically people, machines or other computers.

User View

The user’s view depends on the interface they are using. However, many computer users sit with a laptop or in front of a pc with a keyboard and a mouse. Now increasingly many users are interacting with mobile devices and they are replacing desktop and laptop computer systems for some users.

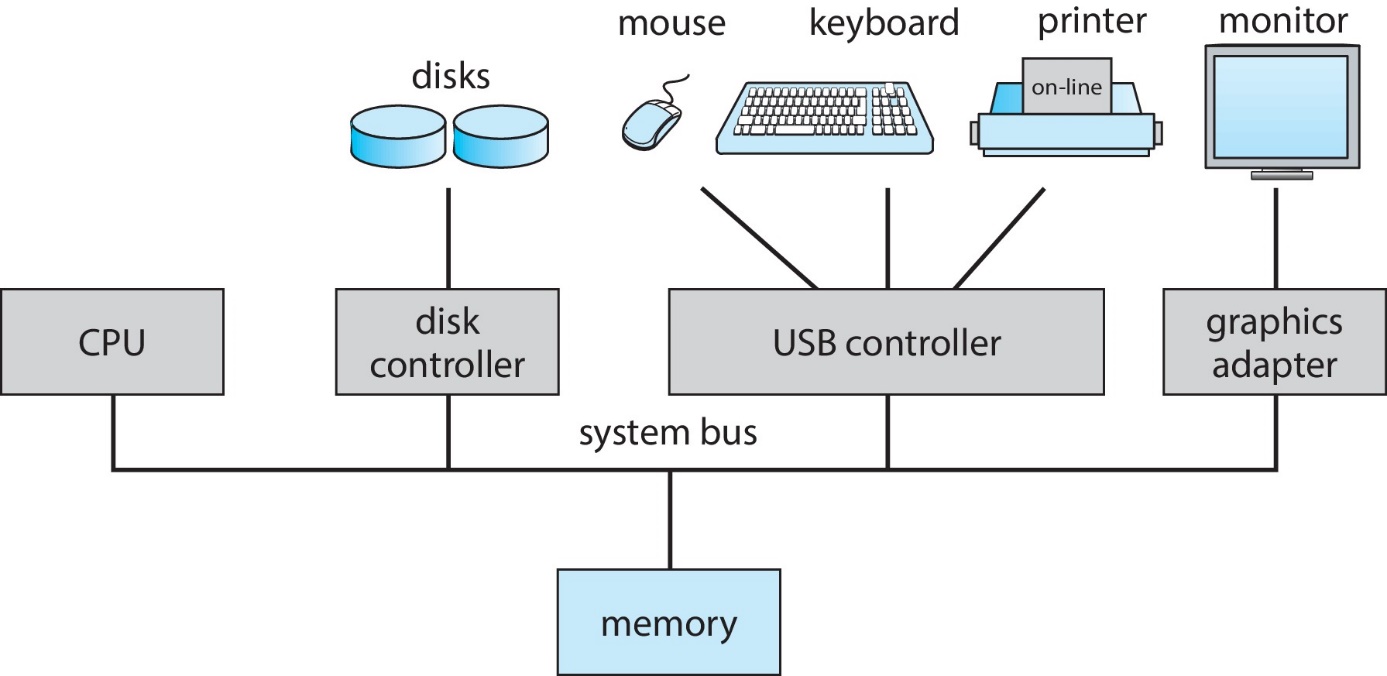
System View

We can view an operating system as a resource allocator because a computer system has many resources that may be required to solve a problem: CPU time, memory space, storage space, I/O devices, and so on. The operating system acts as a manager of these resources.

**Computer-System Organization**

A computer system consist of one or more CPUs and a number of device controllers through a common bus that provides access between components and shared memory. Each device controller is in charge of a specific type of device (for example, a disk drive, audio device, or graphics display). Depending on the controller, more than one device may be attached.

Typically, operating systems has a device driver for each device controller. This device driver understands the device and provides the rest of the operating system with a uniform interface to the device.



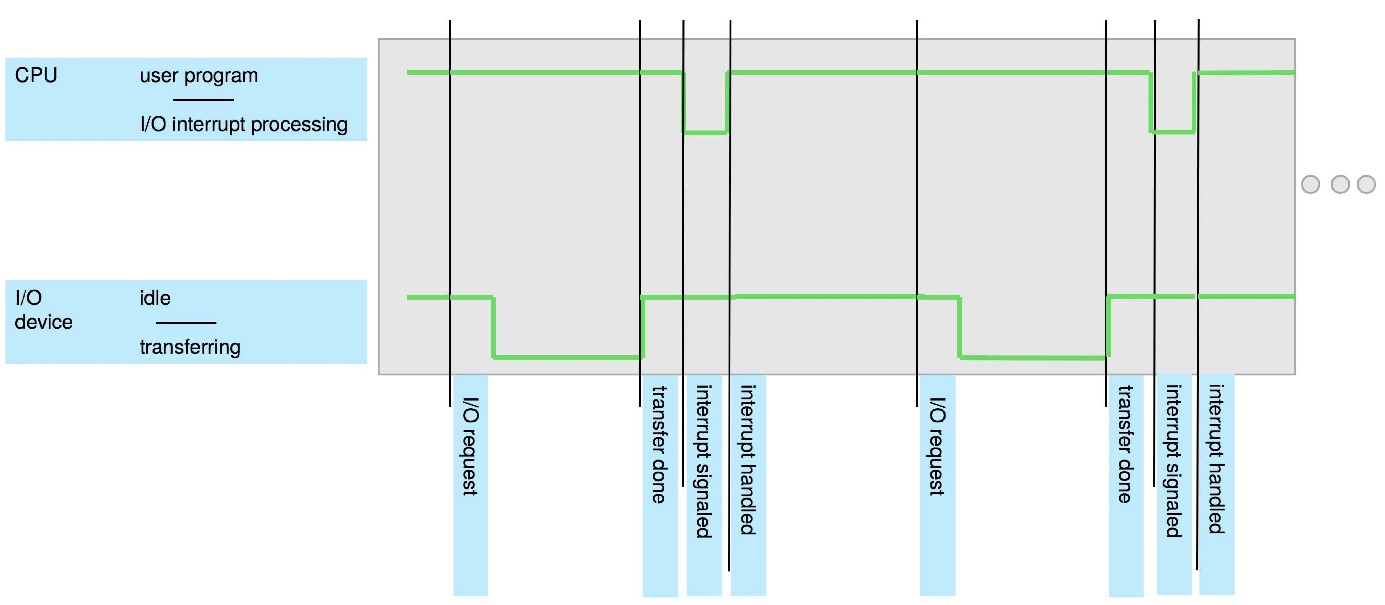
Interrupts

Consider a typical computer operation: a program performing I/O. To start an I/O operation, the device driver loads the appropriate registers in the device controller. The device controller, in turn, examines the contents of these registers to determine what action to take ( such as “read a character from the keyboard”). The controller starts the transfer of data from the device to its local buffer. Once the transfer of data is complete, the device controller informs the device driver that it has finished its operation. The device driver then gives control to other parts of the operating system, possibly returning the data or a pointer to the data if the operation was a read. For other operations, the device driver returns status information such as “write completed successfully” or “device busy”. But how does the controller inform the device driver that it has finished its operation? This is accomplished via an interrupt.

Interrupts - Overview

Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by the system bus. (There may be many buses within a computer system, but the system bus is the main communications path between the major components.) Interrupts are used for many other purposes as well and are key part of how operating systems and hardware interact.

Interrupts must be handled quickly, as they occur very frequently. A table of pointers to interrupt routines can be used instead of providing the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices. This array, or interrupt vector, of addresses is then indexed by a unique number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.



Interrupts – Implementation

The basic interrupt mechanism works as follows: The CPU hardware has a wire called the interrupt-request line that the CPU senses after executing every instruction. When the CPU detects that a controller has asserted a signal on the interrupt-request line, it reads the interrupt number and jumps to the interrupt-handler routine by using that interrupt number as an index into the interrupt vector. It then starts execution at the address associated with that index. The interrupt handler saves any state it will be changing during its operation, determines the cause of the interrupt, performs the necessary processing, performs a state restore, and executes a return\_from\_interrupt instruction to return the CPU to the execution state prior to the interrupt. We say that the device controller raises an interrupt by asserting a signal on the interrupt handler, and the handler clears the interrupt by servicing the device.

In a modern operating system, however, we need more sophisticated interrupt-handling features as:

1. We need the ability to defer interrupt handling during critical processing.
2. We need an efficient way to dispatch to the proper interrupt handler for a device.
3. We need multilevel interrupts, so that the operating system can distinguish between high- and low-priority interrupts and can respond with the appropriate degree of urgency.

In modern computer hardware, these three features are provided by the CPU and the interrupt-controller hardware.



Most CPUs have two interrupt request lines. One is non maskable interrupt, which is reserved for events such as unrecoverable memory errors. The second interrupt line is maskable: it can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The maskable interrupt is used by device controllers to request service. When we have a vector interrupt mechanism, its purpose is to reduce the need for a single interrupt handler to search all possibles sources for interrupts to determine which one needs service. A common way to solve this problem is using interrupt chaining, in which each element in the interrupt vector points to the head of a list of interrupt handlers.

Storage Structure

The CPU can load instructions only from memory, so any programs must first be loaded into memory to run. General-purpose computers run most of their programs from rewritable memory, called main memory (also called random-access memory, or RAM). Main memory commonly is implemented in a semiconductor technology called dynamic random-access memory (DRAM). Computers use other forms of memory as well. For example, the first program to run on a computer power-on is a bootstrap program. A bootstrap program is the first code that is executed when the computer system is started. The entire operating system depends on the bootstrap program to work correctly as it loads the operating system. The bootstrap program, operating system and RAM happens in the main memory when the bootstrap sends signals to the operating system and the operating system reads the OS and then starts the devices. Since RAM is volatile – loses its content when power is turned off or otherwise lost – we cannot trust it to hold the bootstrap program. Instead, for this and some other purposes, the computer uses electrically erasable programmable read-only memory (EEPROM) and other forms of firmware – storage that infrequently written to and is nonvolatile. So the RAM can’t retain the information when the power shuts down and need electricity to retain the information, and that is why it is called volatile.

***Why can’t we reside the programs in main memory permanently?***

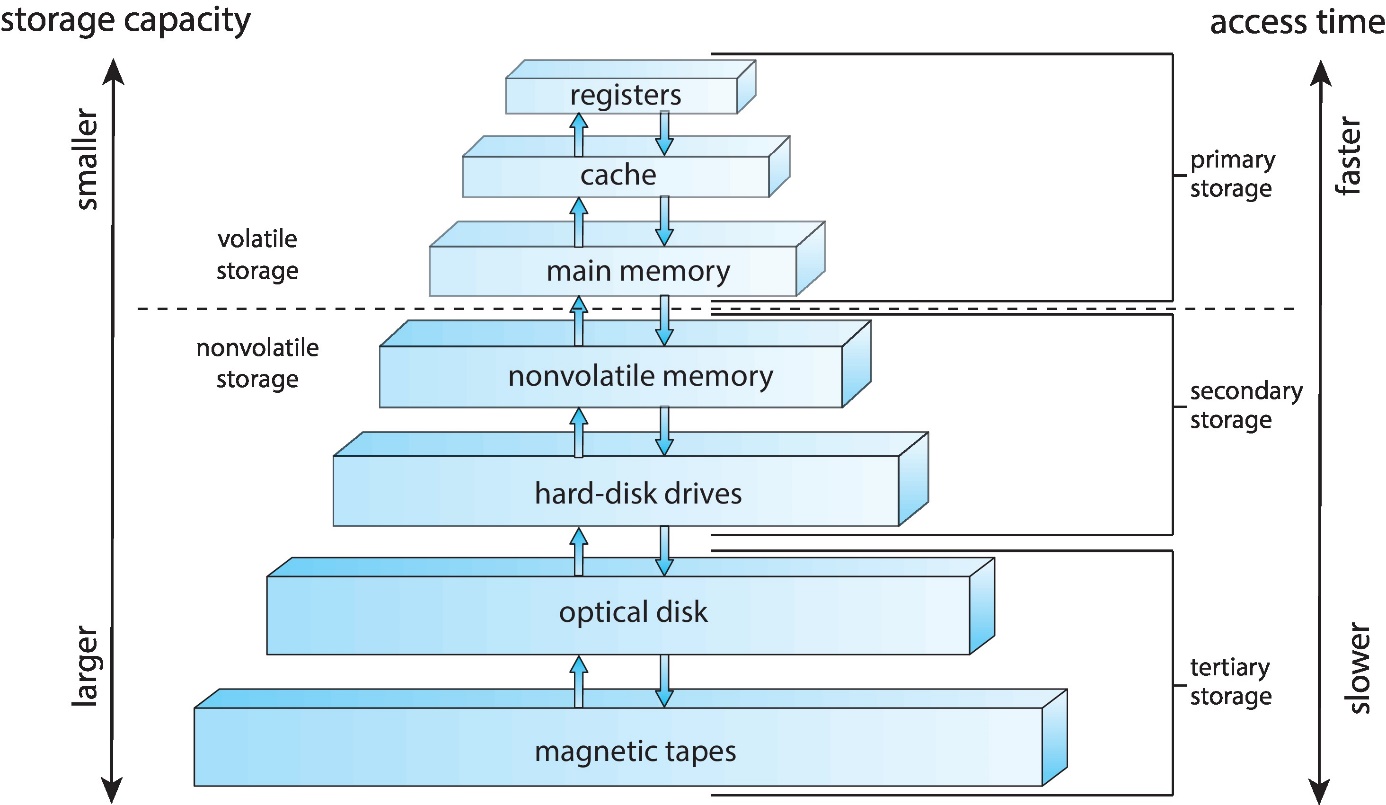
This is usually not possible on most systems for two reasons:

* Main memory consists of bootstrap program, operating system, and RAM. And as we know RAM is a volatile memory, so that it can’t retain information after shutdown. In conclusion here the main memory is volatile and it loses its content when power is turned off or otherwise lost.
* The second on is that the main memory is too small to store all the needed programs and data permanently.

***Secondary Storage***

Since the main memory can’t reside the programs permanently, most computer systems provide secondary storage as an extension of main memory. The main requirement for secondary storage is that it be able to hold large quantities of data permanently, so it is non-volatile storage. The main differences among the various storage systems lie in speed, size, and volatility.

**Examples of secondary storage:** The most common secondary storage devices are hard-disk drives (HDDs) programs and data. Most programs are stored in secondary storage until they are loaded into memory.



I/O Structure

***Direct Memory Access (DMA)***

Direct memory access is a feature of computer systems and allows certain hardware subsystems to access main system memory independently of the central processing unit (CPU). Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller (DMAC) when the operation is done.

**Computer-System Architecture**

A computer system can be organized in a number of different ways, which we can categorize roughly according to the number of general-purpose processors used.

***Single Processor Systems***

A single processor system contains only one processor. The computer used a single processor containing one CPU with a single processing core. The core is the component that executes instructions and registers for storing data locally. These systems have other special-purposes processors as well. They may come in the form of device-specific processors, such as disk, keyboards, and graphic controllers.

***Multiprocessor Systems***

On the more modern computers, from mobile devices to servers, multiprocessor systems now dominate the landscape of computing. This system has traditionally two or more processors, each with a single-core CPU. The processors share the computer bus and sometimes the clock, memory, and peripheral devices. The primarily advantage of multiprocessor system is increased throughput.

***Clustered Systems***

Another type of multiprocessor system is a clustered system, which gather together multiple CPUs. Clustered systems differ from the multiprocessor systems described in that they are composed of two or more individual systems or nodes joined together.

**Operating-System Operations**

An operation system provides the environment within which programs are executed. For a computer to start running, when it is powered up or rebooted it needs to have an initial program to run. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and how to start executing the system. To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory.