Chapter Two O.S Dr. Amani yousif

**Lecture 3**

**Computer System Structures**

**3.1 Computer System Operation:**

A modern, general-purpose computer system consists of a CPU and several device controllers that are **connected through a common bus** that provides access to the shared memory system, CPU other devices can execute concurrently competing for memory cycles.as appear in figure (1)

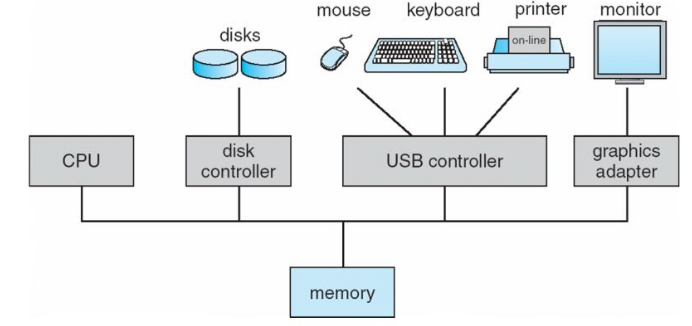


Figure (1) device controller

**Booting:**

It is the operation of bringing the **operating system kernel** from the secondary storage and put it in main storage to execute it in CPU. There is a program bootstrap which is performing this operation when the computer is powered up or rebooted.

***Bootstrap software***: it is an initial program and simple it is stored in read-only memory (**ROM**) such as firmware or EEPROM within the computer hardware.

**Jobs of Bootstrap program:**

1. Initialize all the aspects of the system, from CPU registers to devise controllers to memory contents.
2. Locate and load the **operating system kernel** into memory then the operating system starts executing the first process, such as “**init**” and waits for some event to occur.

Types of events are either **software events** (**system call**) or **hardware events** (signals from the hardware devices to the CPU through the system bus and known as an **interrupt**).

***Note***: all modern operating system is “interrupt-driven”.

***Trap (exception):*** it is a software-generated interrupt caused either by an **error** (ex: division by zero or invalid memory access) or by a **specific request** from a user program that an operating system service is performed.

***Interrupt vector (IV):*** it is a fixed location (an array) in the low memory area (first 100 locations of RAM) of the operating system when the interrupt occurs the CPU stops what it's doing and transfer execution to a fixed location (IV) contain starting address of the interrupt service routine(ISR), on completion the CPU resumes the interrupted computation.

***Interrupt Service Routine***: is it a routine provided to be responsible for dealing with the interrupt.

**3.2 I/O structure**

All I/O devices have general and special structure, a device controller maintains **small local buffer storage** and a **set of special purpose registers**. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. The size of a local buffer within a device controller varies from one controller to another, depending on the particular device being controlled.

**3.3 I/O interrupt**

To start an I/O operation the CPU loads the appropriate registers within the device controller. The device controller, in turn examines the contents of these registers to determine what action to take .for example if it finds a read request the controller will start the transfer of data from the device to its local buffer.

Once the transfer of data is complete; the device controller informs the CPU that it has finished its operation. It accomplishes this communication by triggering an interrupt. This situation will occur in general as the result of a user process requesting I/O.

Once the I/O is started two actions are possible. In the simplest case, the I/O is started; then I/O completion, control is returned to user process. This case is known as synchronous. The other possibility called asynchronous I/O, return control to the user program without waiting for the I/O to complete , the I/O then continue while other system operations occurs. **The main advantage of asynchronous I/O is the increased system efficiency.**

**Synochronous I/O**: the I/O is started then at I/O completion, the control is returned to the user process, wait instruction idle the CPU until the next interrupt.

**Asynchronous I/O**: After I/O starts, control returns to user program without waiting for I/O completion.it include the following operations:

1. System call – request to the operating system to user program that I/O not complete.
2. Device-status table contains entry for each I/O device indicating its type, address, and state.
3. Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt

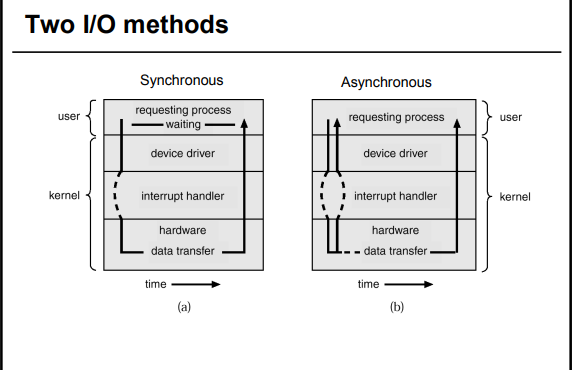


Figure (2 ) synchronous and asynchronous I/o methods

**Spooling**

Spooling is a process in which data is temporarily held to be used and executed by a device, program or the system. Data is sent to and stored in memory or other volatile storage until the program or computer requests it for execution.

"Spool" is technically an acronym for simultaneous peripheral operations online.

**3.4 Hardware protection:**

when we have a single user any error occurs to the system then we could determine that this error must be caused by the user program, but when we begin to deal with **spooling**, **multiprogramming**, and sharing disk to hold many users data this sharing both **improved utilization and increase problems**.

In a multiprogramming system, where one **erroneous** program might **modify the program** or **data of another program**, or even **the resident monitor itself**. **MS-DOS and the Mac OS** both allow this kind of error.

A **properly designed operating system must ensure** that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

Many programming errors are detected by the hardware these errors are normally handled by the operating system.

**Dual-Mode Operation:**

To ensure proper operation, we must protect the operating system and all other programs and their data from any malfunctioning program.

The approach taken by many operating systems provides hardware support that allows us to differentiate among various modes of execution. Protection is needed for any shared resource. We need two separate modes of operation:

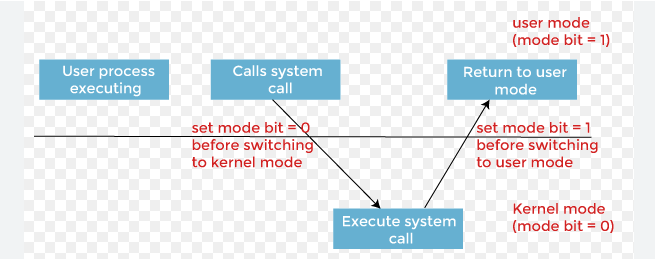
1. User mode
2. Monitor mode(supervisor mode, system mode or privileged mode).

***A bit***, called the ***mode bit*** is added to the hardware of the computer to indicates the current mode: monitor (0) or user (1) with mode bit we could distinguish between a task that is executed on behalf of the operating system, and one that is executed on behalf of the user.

At system boot time the hardware started in monitor mode. The O.S is then loaded, and starts user operation in user mode whenever a trap or an interrupt occurs, the hardware switch from the user mode to monitor mode (that is, chanced the state of the mode bit to 0). Whenever the O.S system gains control of computer, it is in monitor mode as appeared in figure (3).

The dual mode of operation provides us with the means for protecting the operating system from errant users, and errant users from one another.

We accomplish this protection by designing some of machine instructions that may cause harm as **privilege instructions**. The hardware allows privileged instructions to be executed in only monitor mode. If an attempted is made to execute a privilege instruction in user mode, the hardware does not execute the instruction, but rather treats the instruction as illegal and traps to the operating system.



Figure( 3)system call

**3.5 I/O Operation Protection:**

A use program may disrupt the normal operation of the system by issuing illegal I/O instruction we can use various mechanisms to ensure that such disruption cannot take place in the system.

One of them is by defining all I/O instructions to be **privileged instructions**. Thus users cannot issue I/O instructions directly they must do it through the operating system, by executing a system call to request that the operating system performing I/O on its behalf. The operating system, executing in monitor mode, check that the request is valid, and (if the request is valid) does the I/O requested. The operating system then returns to the user.

**3.6 Memory Protection:**

To ensure correct operation, we must protect **the interrupt vector** and **interrupt service routine** from modification by a user program. This protection must be provided by the hardware, we need the ability to determine the range of legal addresses that the program may access, and protect the memory outside that space. We could protect by using two registers a **base register and a limit register**

The base register holds the smallest legal physical memory address

The limit register contains the size of range.

The base and limit registers controlled by privileged instructions.

This protection is accomplished by the CPU hardware comparing every address generated in user mode with the registers. Any attempt by a program executing in user mode to access monitor memory or other user’s memory results in a trap to monitor.

0

Monitor

Job1

Job2

Job3

Job4

256000

300040 base register

420940 limit register = 120900

880000

1024000

Figure (4 ) base and limit reg.

The **base register** holds the smallest legal physical memory address. **Limit register**: contains the size of the range.

This protection is accomplished by the CPU hardware comparing every address generated in user mode with the registers. Any attempt by a program executing in user mode to access monitor memory or other users’ memory results in a trap to the monitor, which treats the attempts as a fatal error.

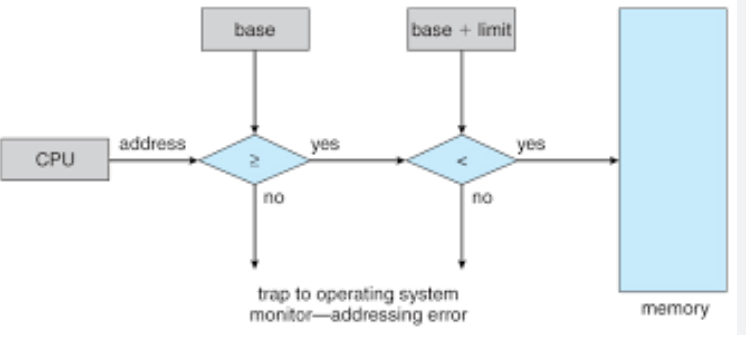


Figure (5) memory protection

**3.7 CPU Protection:**

In addition to protecting I/O and memory, we must ensure that the operating system maintains control. We must prevent the user from getting stuck in an infinite loop or not calling system services, and never returning control to the operating system. To accomplish this goal, we can use ***a timer***.

The 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.

We can use the timer to prevent a user program from running too long Simple technique is to initialize a counter with the amount of time that a program is allowed to run.

Amore common use of a timer is to implement **time-sharing**. In most cases, the timer could be set to interrupt every N millisecond, where N is the time slice that each user is allowed to execute before the next user gets control of the CPU. The operating system is invoked to perform housekeeping tasks.

This procedure is known as **context switching**, following a context switch, the next program continues with its execution **from the point**