# JEPPIAAR INSTITUTE OF TECHNOLOGY

**“Self-Belief | Self Discipline | Self Respect”**

**DEPARTMENT**

**OF**

**COMPUTER SCIENCE AND ENGINEERING**

**LECTURE NOTES**

**CS8493 – OPERATING SYSTEM**

**(Regulation 2017)**

**Year/Semester: II / 04 CSE**

**2020 – 2021**

**Prepared by**

**Mr.H.Shine**

**Assistant Professor / CSE**

# *UNIT - I*
# OPERATING SYSTEMS OVERVIEW

This Chapter describes about operating system basics, basic elements, Instruction Execution, Interrupts, Memory Hierarchy, Cache Memory, Direct Memory Access, Multiprocessor and Multicore Organization objectives and functions, Evolution of Operating System, Computer System Organization Operating System Structure and Operations, System Calls, System Programs, OS Generation and System Boot.

## Operating System

An Operating System is a **program that acts as an interface** between applications and the computer hardware.

An OS controls the execution of application programs.

## Operating System Goals

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use.
- Use the computer hardware in an efficient manner.

## Computer System Overview / Computer System Organization

Computer system can be **divided into four components**

- **Hardware:** provides basic computing resources - CPU, memory, I/O devices, System bus.
  **CPU:** Controls the operation of the computer and performs its data processing functions,
  **Memory:** Stores data and programs. Main memory – volatile (Content lost when power off) Secondary storage – Non volatile
  **I/O Devices:** Move data between the computer and its external environment
  **System Bus:** Provides for communication among processors, main memory, and I/O modules.
- **Operating system:** controls and coordinates use of hardware among various applications and users.
- **Application Programs:** define the ways in which the system resources are used to solve the computing problems of the users. Ex. Word processors, compilers, web browsers, database systems, video games
- **Users:** People, machines, other computers.

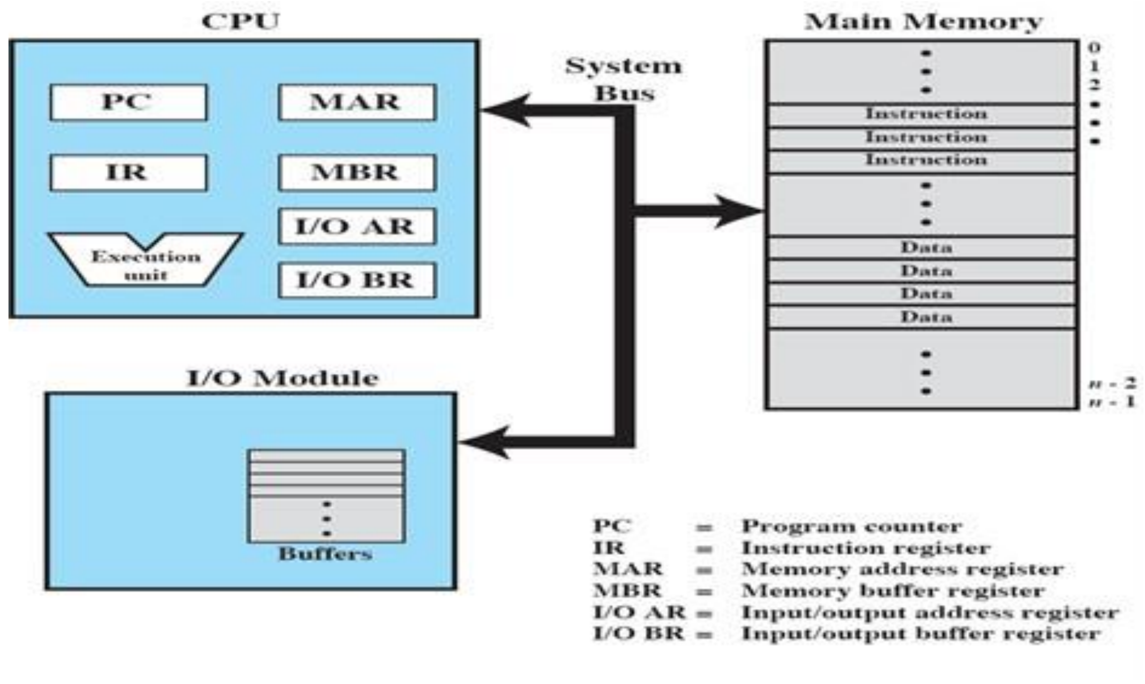


**Fig 1.1 Computer Components: Top Level -View**

Two internal registers – MAR & MBR (to the processor) are used is to exchange data with memory. Similarly, I/OAR and I/OBR are used is to exchange data with I/o devices.

**Memory address register (MAR),** which specifies the address in memory for the next read or write;

**Memory buffer register (MBR),** which contains the data to be written into memory or which receives the data read from memory.

**I/O address register (I/OAR)** specifies a particular I/O device.

**I/O buffer register (I/OBR)** is used for the exchange of data between an I/O module and the processor.

A memory module stores instruction and data. An I/O module transfers data from external devices to processor and memory, and vice versa. It contains internal buffers for temporarily holding data until they can be sent on.

**Computer Overview**

One or more CPUs, device controllers connect through common bus providing access to shared memory

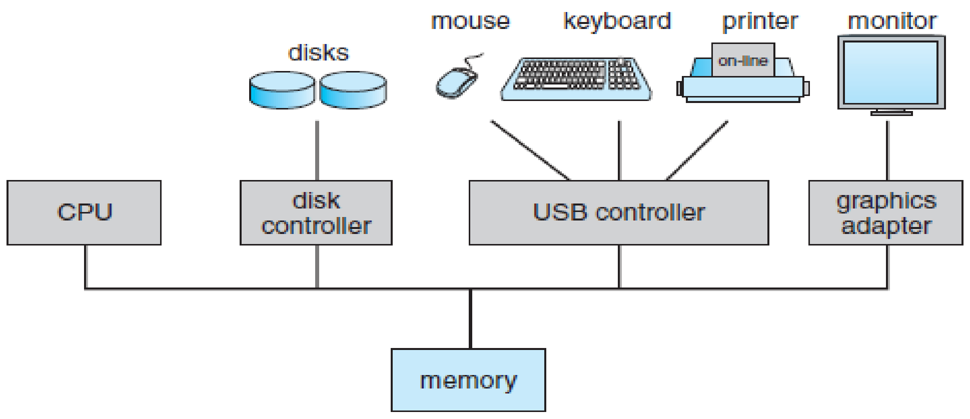


**Fig 1.2 Computer Components: Overview**

**Computer-System Operation**

- I/O devices and the CPU can execute concurrently.
- Each device controller is responsible for a particular device type.
- Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller.
- Device controller informs CPU that it has finished its operation by causing an interrupt.

**Instruction Execution**

A **program** is a set of instructions which are stored in memory. The CPU executes the instruction of the program one by one. The processing required for a single instruction execution is called an **instruction cycle**.

The **program counter** (PC) holds the address of the next instruction to be fetched. The fetched instruction is loaded into the instruction register (IR). The instruction contains bits that specify the action the processor is to take.

The processor interprets the instruction and performs the required action. Instruction Execution consists of 4 steps:

**Fetch:** The processor reads the instructions from memory one at a time

**Decode:** identify the operation to be performed

**Fetch Operand:** read the data

**Execute:** Executes each instruction

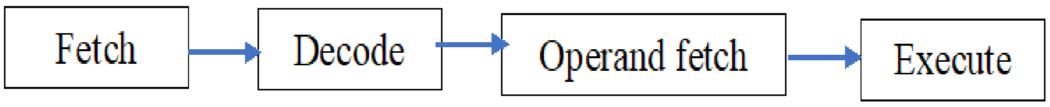


**Fig 1.3 Instruction Cycle - Stages of Instruction execution**

**Interrupts**

Interrupt is an external event to the currently executing process that affects the normal flow of execution. It Suspends a process, and it can be resumed later after processing the interrupt.

The **most common classes of interrupts** are

1. Software or program interrupts or trap or exception
2. Timer Interrupts
3. I/O Interrupts
4. Hardware Interrupts

| | |
|---|---|
| **Software *or* Program Interrupt** | Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction and reference outside a user's allowed memory space. |
| **Timer Interrupts** | Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis. |
| **I/O Interrupts** | Generated by an I / O controller, to signal normal completion of an operation to signal a variety of error conditions. |
| **Hardware Interrupts** | Generated by a failure, such as power failure or memory parity error. |

**Fig 1.4 Types of Interrupts**

**Interrupt handler**

Interrupt handler is a program that performs the necessary actions to execute the interrupts. It is generally part of the operating system.

In the interrupt stage, the processor checks to see if any interrupts have occurred. If an interrupt is pending, the processor suspends execution of the current program and executes an interrupt handler routine.

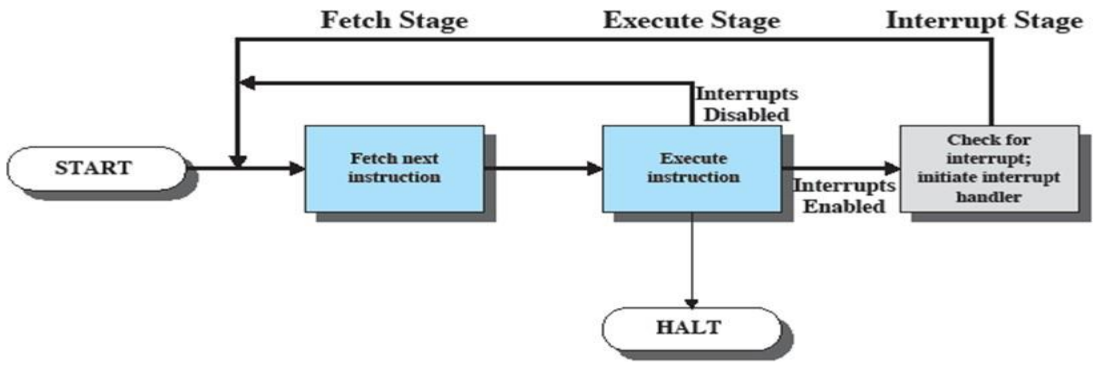


**Fig 1.5 Stages of Instruction execution with Interrupt Handler**

**Interrupt Processing**

An interrupt triggers a number of events, both in the processor hardware and in software.

The Figure below shows a typical sequence. When an I/O device completes an I/O operation, the following sequence of hardware events occurs:

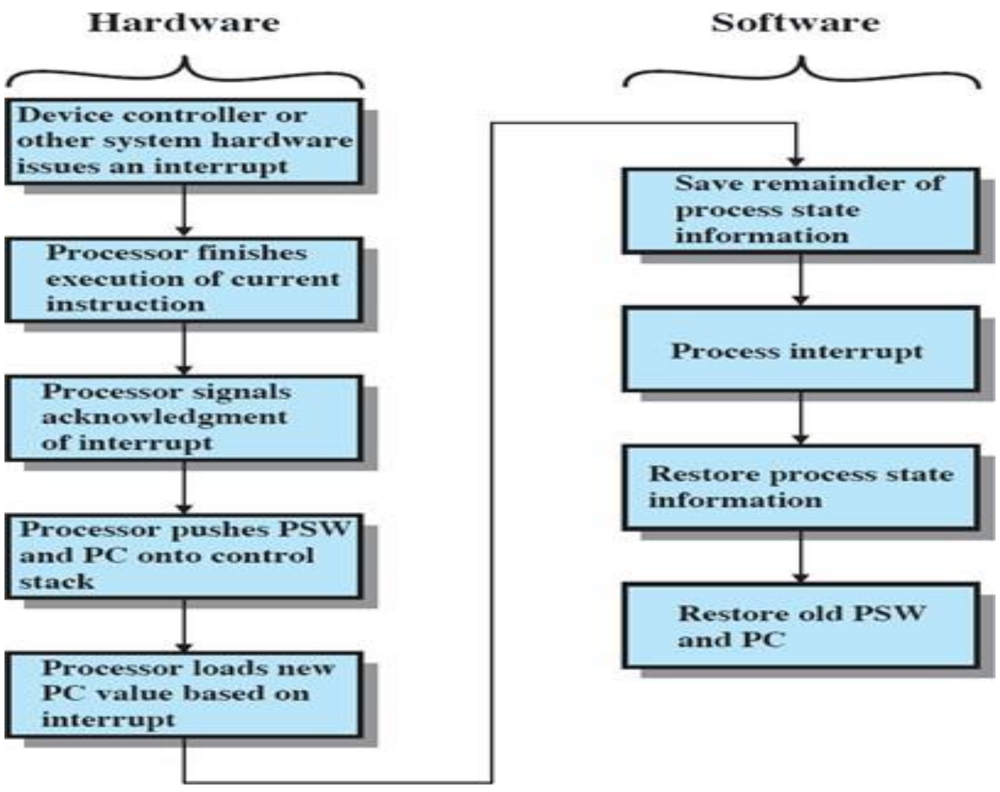

Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

**Fig 1.6 Interrupt Processing**

1. The device issues an interrupt signal to the processor.

2. The processor finishes execution of the current instruction before responding to the interrupt,

3. The processor tests for a pending interrupt request, if yes, sends an acknowledgment signal and remove its interrupt signal.

4. The processor prepares to transfer control to the interrupt routine. It saves information needed to resume the current program in program status word (PSW).

5. The processor then loads the program counter with the entry location of the interrupt-handling routine.

6. New interrupted routine state information are saved by saving the contents of all registers on the stack

7. The interrupt handler may now proceed to process the interrupt.

8. When interrupt processing is complete, the saved register values are retrieved from the stack

and restored to the registers.

9. The final act is to restore the PSW and program counter values from the stack.

**Multiple Interrupts**

Two approaches are used to deal with multiple interrupts.

- Disable interrupts
- Define priorities for interrupts

Disable interrupts: A disabled interrupt simply means that the processor ignores any new interrupt request signal while executing the current interrupt.

Define priorities for interrupts: define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted.

**The Memory Hierarchy**

The memory hierarchy is organized in order as registers, Cache, main memory, Secondary Memory.

The following advantages of memory hierarchy (Top to bottom level)

- Decrease in cost per bit
- Capacity Increase
- Increase access time
- Decreasing frequency of access to the memory by the processor

**Registers and cache memory:** is the smallest, more expensive, faster memories.

**Main memory or primary storage:** Computer programs must be in main memory for execution. It is also called random-access memory or RAM. Main memory is usually too small to store all needed programs and data permanently.

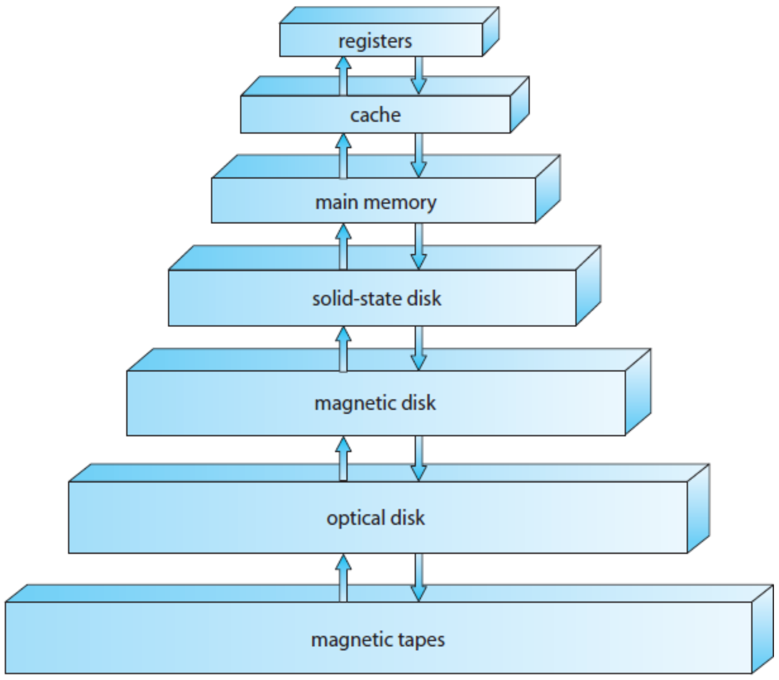


**Fig 1.7 Storage-device or memory hierarchy**

Main memory is a volatile storage device that loses its contents when power is turned off.

**Secondary Memory or auxiliary memory:** secondary storage is used as an extension of main memory. It is a Non Volatile storage that retains the contents even when the power is off.

The most common secondary-storage device is a magnetic disk, which provides storage for both programs and data.

**I/O Structure:** Computer system consists of CPUs and multiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device.

Depending on the controller, there may be more than one attached device. For instance, seven or more devices can be attached to the **small computer-systems interface (SCSI) controller.**

A device controller maintains some 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.

After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.

**Cache Memory**

Cache memory is a **small, high speed memory** between the processor and main memory. Cache is made up of semiconductors memory (high speed static RAM). It holds data for temporary purpose to reduce the time required to service I/O requests.

**Cache Principles:** The cache contains a copy of a portion of main memory. When the processor attempts to read a byte or word of memory, a check is made to determine if the byte or word is in the cache. If so, the byte or word is delivered to the processor.

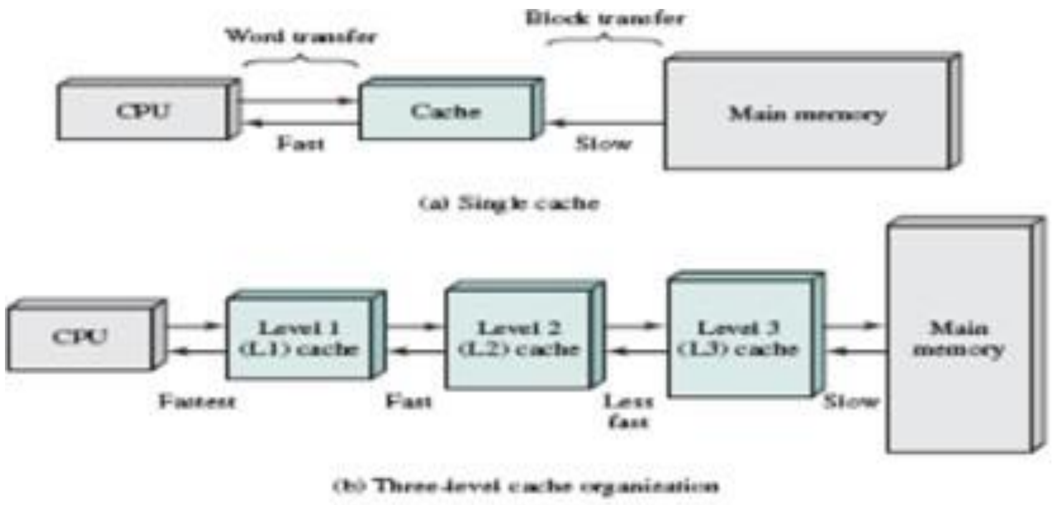


**Fig 1.8 Cache and Main Memory**

If not, a block of main memory, consisting of some fixed number of bytes, is read into the cache and then the byte or word is delivered to the processor.

**Some of the issues of cache memory**:

• Cache size

• Block size

• Mapping function

• Replacement algorithm

• Write policy

• Number of cache levels

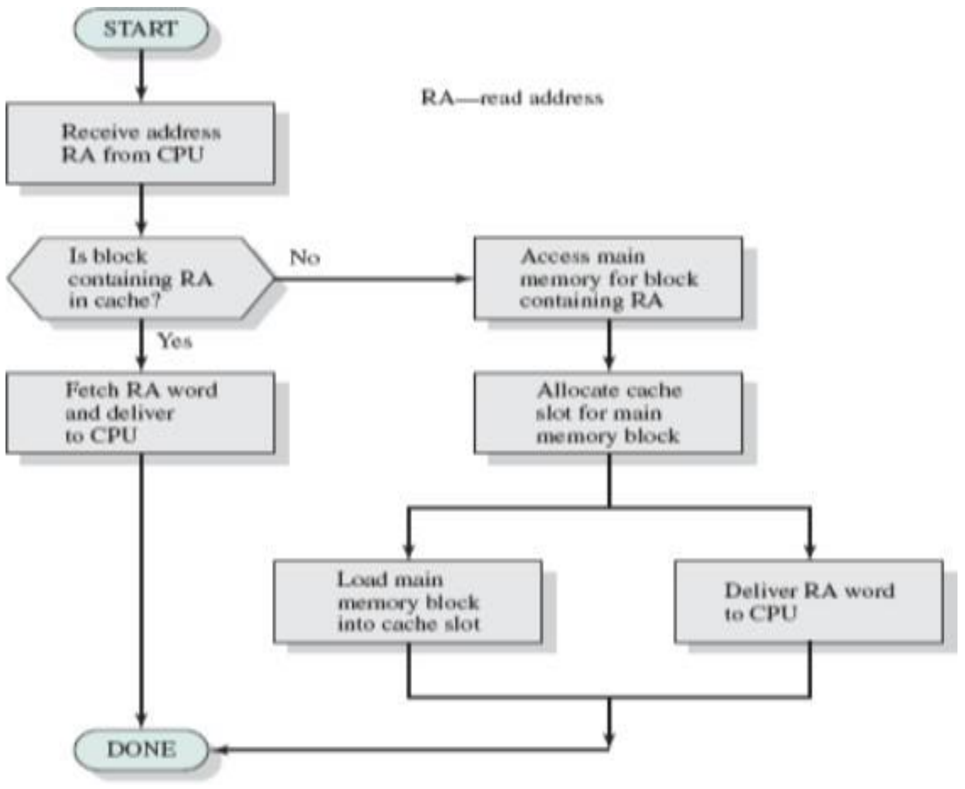

START
RA—read address
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache slot for main memory block
Load main memory block into cache slot
Deliver RA word to CPU
DONE

**Fig 1.9 Cache Read Operation**

**I/O operations**

Three techniques are possible for I/O operations: programmed I/O, interrupt-driven I/O , Direct memory access (DMA).

**Programmed I/O**

The I/O module performs the requested read / write and then sets the appropriate bits in the I/O status register but takes no further action to alert the processor.

The processor must determine whether the I/O instruction is completed or not. It does not interrupt the processor

**Interrupt- driven I/O**

The processor issues an I/O command to a module and then go on to do some other useful work.

I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

Interrupt-driven I/O, though more efficient than simple programmed I/O, still requires the active intervention of the processor to transfer data between memory and an I/O module

**Direct Memory Access**

Data transfer takes place using a separate control unit called DMA controller without continuous intervention of processor. The processor issues command to DMA module, by sending necessary information to DMA module.

- Whether a read or write is requested
- The address of the I/O device involved
- The starting location in memory to read data from or write data to
- The number of words to be read or written

Processor does other work. The DMA controller module will take care data transfer. It transfers the entire block of data, one word at a time, to or from memory without intervention of the processor.

DMA sends a signal to processor when the transfer is complete, system control is return to processor. Thus, the processor is involved only at the beginning and end of the transfer.

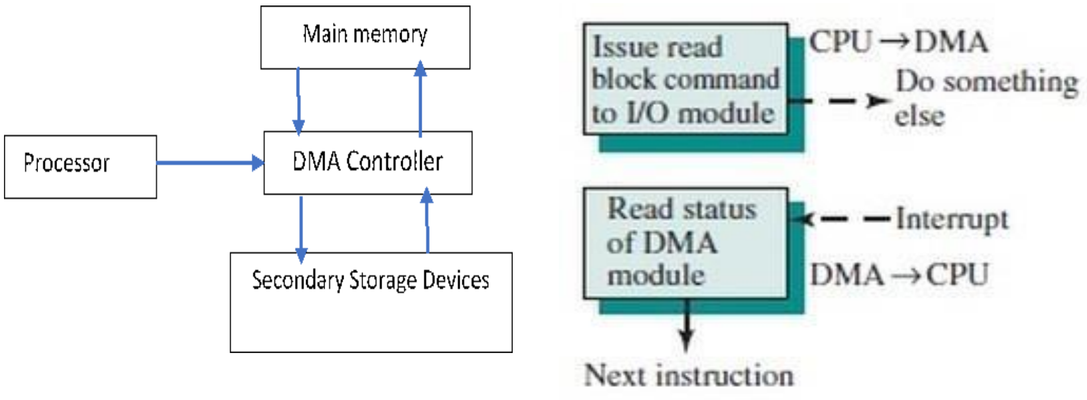


**Fig 1.10 DMA Data Transfer**

**Advantages:**

- DMA is more efficient than interrupt-driven or programmed I/O
- Since the processor can be working on something else while the peripheral can be populating memory
- Processor intervention is not required.

**Disadvantages**:

- When the processor needs the bus and must wait for the DMA module.
- Requires a DMA controller to carry out the operation, which increases the cost of the system

**Multiprocessor and Multicore Organization**

To provide parallelism, the two most popular approaches are used, symmetric multiprocessors (SMPs) and multicore computers and clusters.

Symmetric Multiprocessors systems described under the section Evolution of Operating System page.

**Multicore Computers:** A multicore computer, also known as a chip multiprocessor, combines two or more processors (called cores) on a single piece of silicon (called a die).

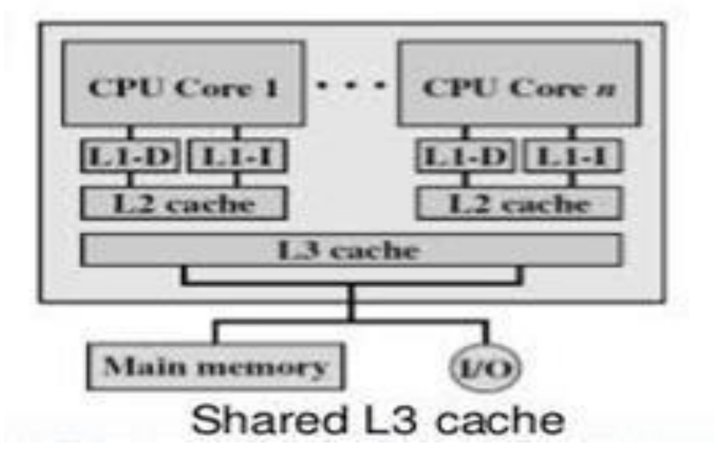


**Fig 1.11 Shared L3 Cache**

Each core consists of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction cache memory and L1 data caches.

In addition, it has L2 cache and, in some cases, L3 cache.

Example of a multicore system is the Intel Core i7, which includes four x86 processors, each with a dedicated L2 cache, and with a shared L3 cache

## Operating System Overview

### Operating System Objective

An Operating System is a program and acts as an interface between applications and the computer hardware.

### Major Design Objectives

- Efficient use of a computer system resources
- User Convenience: Good service, user friendly, ease of use
- Ability to evolve: New version can be evolved

### Operating System services / functions

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs

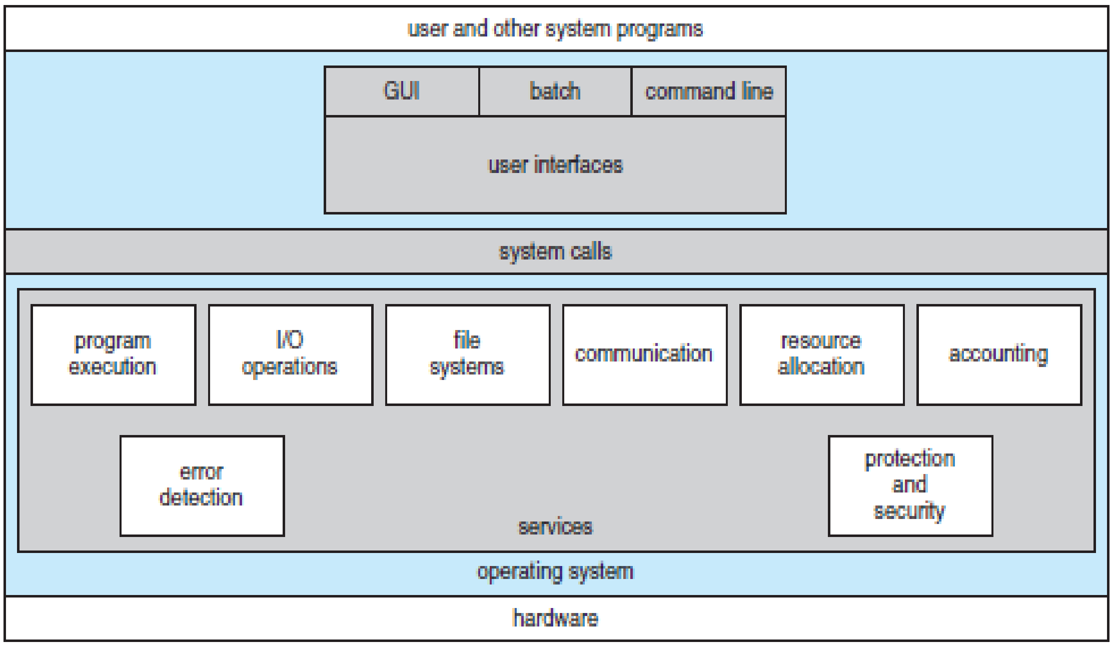


**Fig 1.12 A View of Operating System Service**

1. Program development: to assist the programmer in creating programs.
2. Program execution: OS handles all the scheduling duties to execute the program.
3. Access to I/O devices: Each I/O device requires its own peculiar set of instructions or control signals for operation.
4. Controlled access to files: read / write / modify access
5. Resource manager and Resource allocation and scheduling: the OS controls access to the system as a whole and to specific system resources.
6. Error detection and response: OS must provide a response that clears the error condition with the least impact on running applications.
7. Accounting: A good OS will collect usage statistics for various resources and monitor performance parameters such as response time.
8. Instruction set architecture (ISA) : The ISA defines machine language instructions that a computer can follow.
9. Application binary interface (ABI) : The ABI defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA.
10. Application programming interface (API): The API gives a program access to the hardware resources and services available in a system high-level language (HLL) library calls. Using an API enables application software to be ported easily, through recompilation, to other systems that support the same API.

**Two major functions of OS are**

- The Operating System as a User/Computer Interface
- The Operating System as Resource Manager

**Operating System as a User/Computer Interface:** The end user views a computer system in terms of a set of applications. The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion.

The end user generally is not concerned with the details of computer hardware. An application can be expressed in a programming language and is developed by an application programmer.

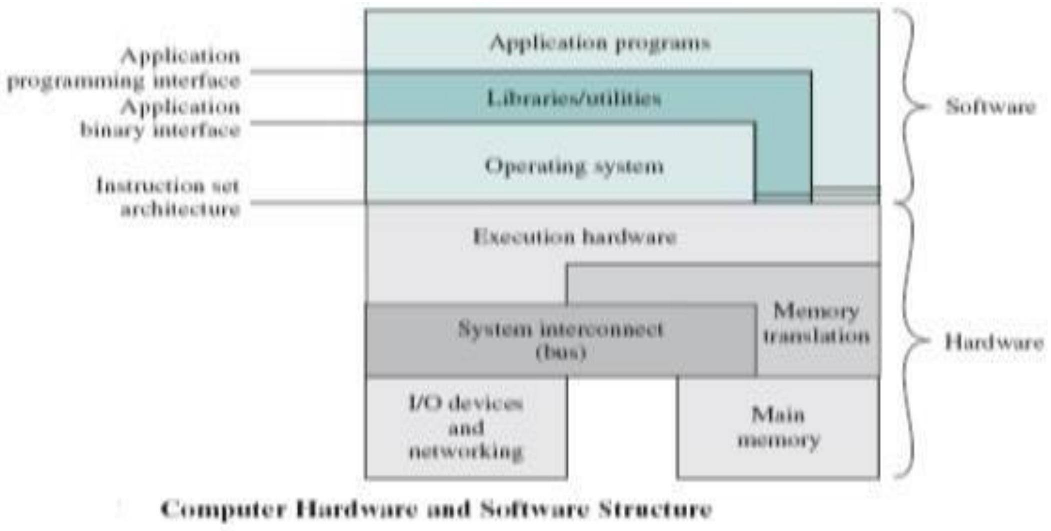

Application programming interface
Application binary interface
Instruction set architecture
Application programs
Libraries/utilities
Operating system
Execution hardware
System interconnect (bus)
Memory translation
I/O devices and networking
Main memory
Software
Hardware
Computer Hardware and Software Structure

**Fig 1.13 Computer Hardware and Software Structure**

Controlling the computer hardware by the end user is an overwhelmingly complex undertaking. To ease this, a set of system programs or utilities or library programs is provided. That assists in program creation, the management of files, and the control of I/O device. Ex : Loader, linker, compiler etc

**Operating System as Resource Manager:** A computer has a collection of resources. The OS is responsible for managing these resources. The OS functions in the same way as ordinary computer software; that is, it is a program or suite of programs executed by the processor.

The OS directs the processor to use the system resources and maintains the timing signals. A portion of the **OS called kernel**, **or nucleus** is in main memory.

Kernel stores the most frequently used functions. The remainder of main memory contains user programs and data. The memory management hardware in the processor and the OS jointly control the allocation of main memory.

**Evolution of Operating System**

Operating systems have evolved from slow and expensive systems to present-day technology where computing power has reached exponential speeds and relatively inexpensive costs. Different types of evolution are

- Serial processing

- Batch System
- Multiprogramming
- Time Sharing
- Real time Operating system
- Multiprocessor
- Distributed system

**Serial Processing:** 1940s to the mid-1950s, the programmer interacted directly with the computer hardware; there was no OS.

If an error halted the program, the error condition was indicated by the lights. If the program proceeded to a normal completion, the output appeared on the printer.

Two main problems of serial processing are

• **Scheduling:** hardcopy sign-up sheet to reserve computer time. Wasted computing time

• **Setup time:** Setup included loading the compiler, source program, loading and linking. If an error occurred - start over.

**Batch Systems:** The first batch OS was developed in the mid-1950s by General Motors for IBM. A **collection of jobs** called a batch.

Job is a predefined sequence of commands, programs and data that are combined into a single unit. Each job in the batch is independent of the other jobs in the batch.

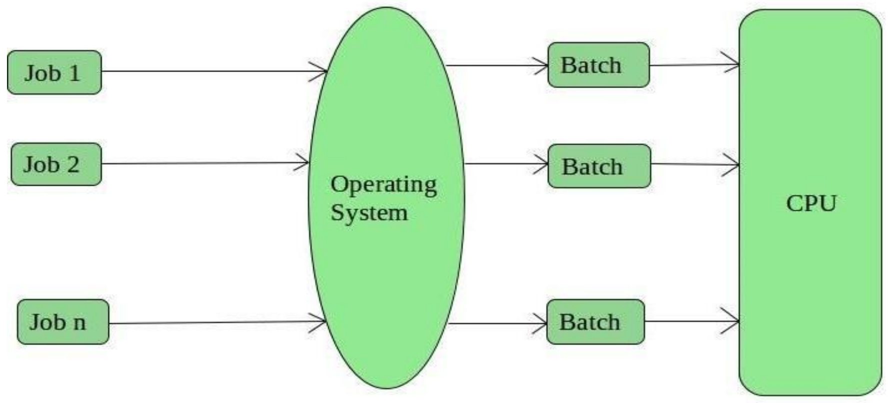


**Fig 1.14 Batch Operating System**

Batch processing uses a piece of software known as the monitor that controls the sequence of events. The monitor must always be in main memory and available for execution. It is also referred as resident monitor.

**Advantages:**

- Multiple users can share the batch systems
- It is easy to manage large work repeatedly in batch systems

**Disadvantages:**

- Batch systems are hard to debug and It is sometime costly
- The other jobs will have to wait for an unknown time if any job fails

**Multiprogrammed Systems:** In batch Operating system, CPU remains idle when job needs I/O Operation.

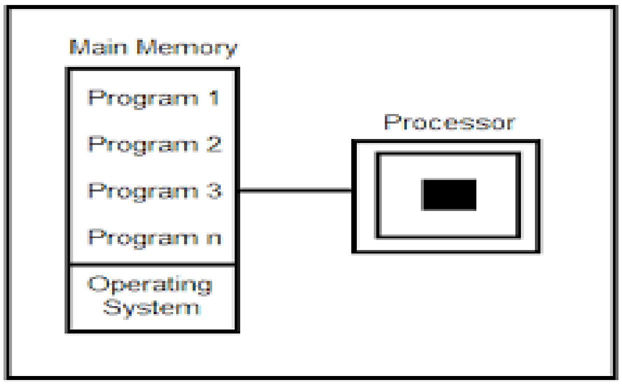


**Fig1.15 Multiprogramming OS**

To keep CPU busy, more than one programs are loaded for execution and CPU switch among all of them. It is knowns as multiprogramming.

It is the central theme of modern operating systems.

**Time-Sharing Systems or Multi Tasking systems:** Logical extension of multiprogramming – can execute many jobs.

Each user gets time of CPU as they use single system. The task can be from single user or from different users also. User interacts directly with the computer by supplying the information to the program.

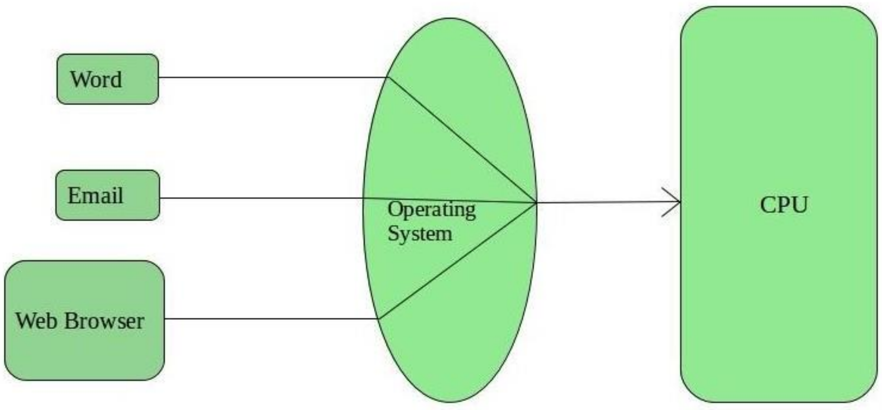


**Fig 1.16 Multiprogramming OS**

**Real-Time Operating System:** Real-time systems are used when there are time requirements are very strict like missile systems, air traffic control systems, robots etc.

The time interval required to process and respond to inputs is very small. This time interval is called response time. Two types of Real-Time Operating System 1.Hard Real-Time Systems 2. Soft Real time system

**Hard Real time System:** A Hard Real-Time System guarantees that critical tasks complete on time. Ex : saving life like automatic parachutes or air bags which are required to be readily available in case of any accident

**Soft Real Time System:** Where a critical real-time task gets priority over other tasks and retains that priority until it completes. These OSs are for applications where for time-constraint is less.

**Multiprocessor Systems:** Two or more central processing units (CPU) within a single computer system. These multiple CPUs are in a close communication sharing the computer bus, memory and other peripheral devices.

These systems are referred as tightly coupled systems. There are two types, Symmetric multiprocessor system and Asymmetric multiprocessor system

**Symmetric Multiprocessing**: All processors can perform the same functions and hence the term symmetric.

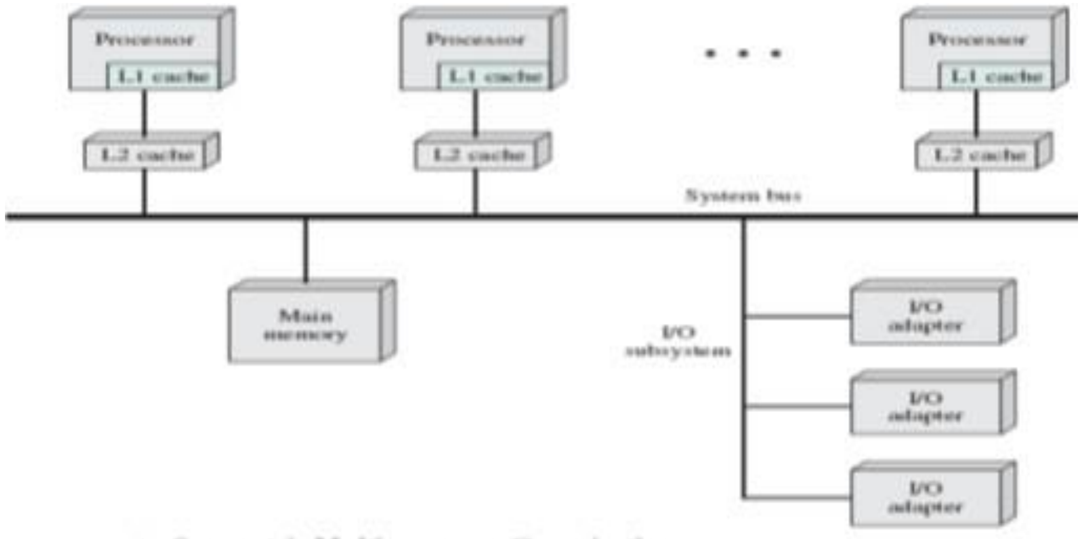


**Fig 1.17 Symmetric multiprocessor Organization**

Each processor runs an identical copy of operating system and communicate with each other. These processors share the same main memory and I/O facilities. Interconnected by a common bus, such that memory access time is approximately the same for each processor.

**Asymmetric Multiprocessing:** All processors are not same. hence the term asymmetric. Defines a master-slave relationship.

A master processor controls the whole operation. Master gives the instruction to the slave processor or slave processor will use some predefined set of tasks.

**Advantages:**

- Performance: a system with multiple processors will yield greater performance than one with a single processor.
- Availability: the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.
- Incremental growth: A user can enhance the performance of a system by adding an additional processor.
- Scaling: performance based on the number of processors configured in the system

**Distributed Operating System:** Multiple central processors are used by Distributed systems to serve multiple real-time applications and multiple users.

Processors communicate with each other through various communication lines. These are known as loosely coupled systems or distributed systems. Processors in this system may vary in size and function. They are referred as sites, nodes, computers, and so on.

**Advantages**

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- Failure of one site in a distributed system doesn't affect the others, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

**Operating System Structure**

There are three different types of Structures in operating system.

- Simple Structure
- Layered Approach
- Kernel Approach

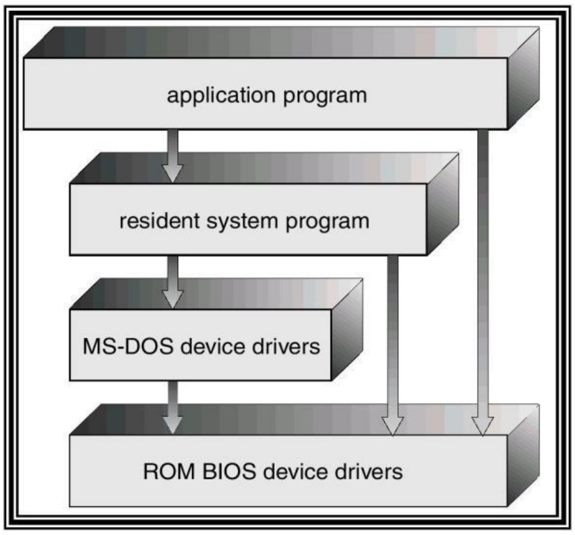
**Simple Structure:** It provides the most functionality in the least space. Ex : MS-DOS. In MS-DOS, the interfaces and levels of functionality are not well separated.

There is no CPU execution mode (User and kernel). Such freedom leaves MS-DOS vulnerable to malicious programs causing entire system crashes when user program fail.

**Layered Approach:** Second type of OS is design is layered approach.

Operating system is divided into number of layers. Each built on top of lower layers. The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

The main advantage of the layered approach is modularity. The modularity makes the debugging & verification easy.



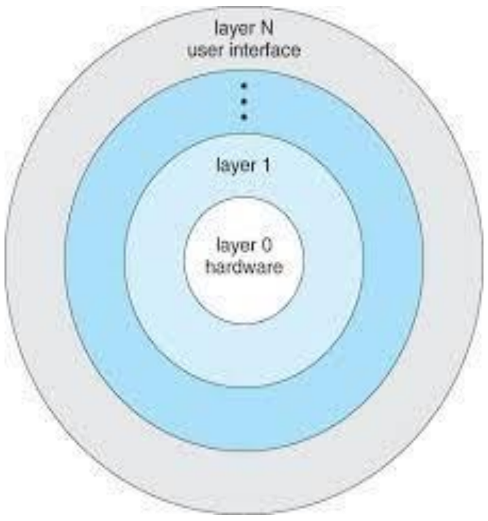


**Fig 1.18 Simple structure** **Fig 1.19 Layered structure**

**Kernel Approach:** The Kernel is a software code that resides in the central core of a operating system. It is responsible for running programs and providing secure access to the hardware.

Kernel has hardware abstraction to hide the underlying complexity from applications. The CPU can execute certain instruction only when it is in the kernel mode. These instruction are called privilege instruction.

The operating system puts the CPU in user mode when a user program is in execution so, that user program cannot interface with the operating system program. Four types of kernels:

1.Monolithic Kernels 2. Microkernels 3. Exokernels 4. Hybrid Kernels

**Monolithic kernels:** provide rich and powerful abstractions of the underlying hardware. The entire operating system runs as a single program in kernel mode.

**Microkernels:** provide a small set of simple hardware abstractions and use applications called servers to provide more functionality. The important functions are

Inter process-Communication, Memory Management, CPU-Scheduling

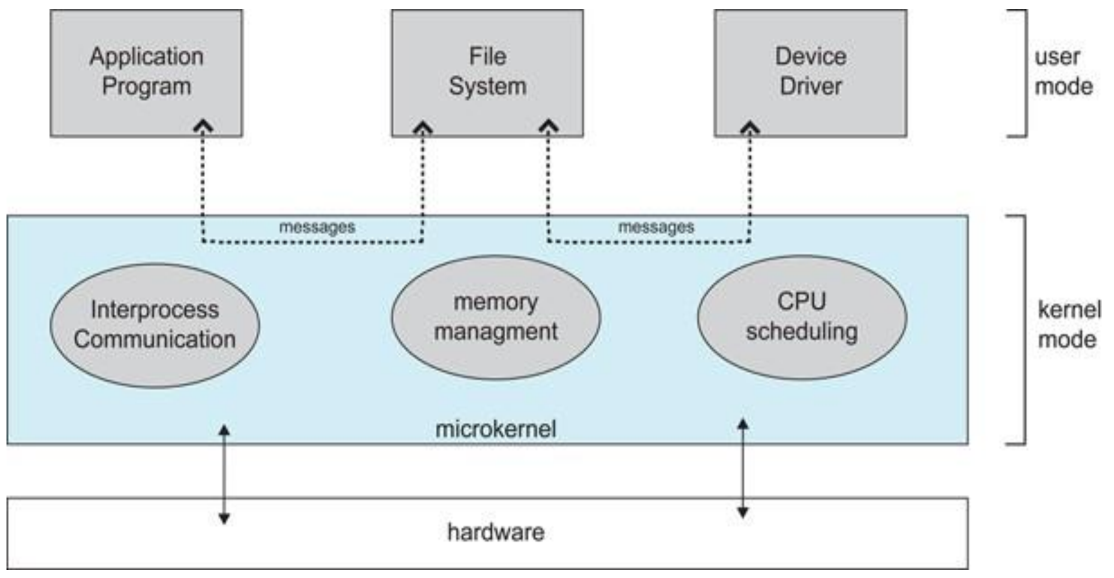


**Fig 1.20 Microkernel**

**Advantages of Microkernel:**

- More reliable (less code is running in kernel mode).
- Expansion of the system is easier. It is simply added in the system application without disturbing the kernel.

**Exokernels:** provide minimal abstractions, allowing low-level hardware access. In exokernel systems, library operating systems provide the abstractions typically present in monolithic kernels.

**Hybrid (modified microkernels):** are much like pure microkernels, except that they include some additional code in kernel space to increase performance.

**Operating System Operations**

There are two modes of operation, User mode and Kernel mode. A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode. 0 represents kernel mode, 1 represents user mode.

**User Mode:** The executing code has no ability to directly access hardware or reference memory.

When the computer system run user applications like creating a text document or using any application program, then the system is in the user mode.

When the user application requests for a service from the operating system or an interrupt occurs or system call, then there will be a transition from user to kernel mode to fulfill the requests.

**Kernel mode:** also called supervisor mode, system mode, or privileged mode. Most critical tasks of the operating system are executing in the kernel mode.

When the system boots, hardware starts in kernel mode to load operating system, Later user program runs in user mode

To provide protection to the hardware, we have privileged instructions which execute only in kernel mode. If user attempt to run privileged instruction in user mode then it will treat instruction as illegal and traps to OS.

**Some of the privileged instructions are:**

- Handling Interrupts
- To switch from user mode to kernel mode.
- Input-Output management.

**System Calls**

System calls provide the interface between a running program (process) and the operating system. System call is a technique by which a program executing in user mode can request the kernel service

These calls are generally available as assembly-language instructions, can be written in C or C++ as routines.

**Types of system calls:** System calls can be grouped roughly into five major categories:

- File management
- Process control
- I/O device management
- Information processing and maintenance
- Communications.

**File Management:** User can create a file using Create() system call. File name with the attributes are required for creating or deleting a file. Some operating systems provide many more calls, such as calls for file move and copy.

**The basic file management system calls are**

- Create file, delete file
- Open, close file
- Read, write
- Get file attributes, set file attributes.

**Process Control:** A running program needs to be able to load, execute, halt its execution either normally (end) or abnormally (abort).

**The process control system calls are**

- end, abort
- load, execute
- Create process and terminate process

- get process attributes and set process attributes.
- wait for time, wait event, signal event
- Allocate and free memory.

**I/O Device Management:** Users has to request the device, to ensure exclusive use of it. After finishing the task with the device, it should be released, similar to the open and close system calls for files.

**The I/O Device Management system calls are**

- Request device, release device.
- Read, write, reposition
- Get device attributes, set device attributes
- Logically attach or detach devices

**Information processing and maintenance:** Many system calls exist simply for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current time and date.

Other system calls may return information about the system, such as the number of current users, the version number of the operating system, the amount of free memory or disk space, and so on.

In addition, the operating system keeps information about all its processes, and system calls are used to access this information.

**The Information processing and maintenance system calls are**

- Get time or date, set time or date
- Get system data, set system data
- Get process, file, or device attributes
- Set process, file or device attributes

**Communications:** There are two common models of inter process communication: Message passing model and Shared-memory model.

**Message-passing model:** The communicating processes exchange messages with one another. Messages can be exchanged between the processes either directly or indirectly through a common mailbox.

Before communication can take place, a connection must be opened. The name of the other communicator must be known

**Shared-memory model:** Processes use shared memory create and shared memory attach system calls to create and gain access to regions of memory owned by other processes.

Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.

**system calls for communication are**

- Create, delete communication connection
- Send, receive messages
- Transfer status information
- Attach or detach remote devices

## SYSTEM PROGRAM

System programs provide a convenient environment for program development and execution. They can be divided into several categories:

**File management:** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.

**Status information:** The status such as date, time, amount of available memory or diskspace, number of users or similar status information.

**File modification:** Several text editors may be available to create and modify the content of files stored on disk or tape.

**Programming-language support:** Compilers, assemblers, and interpreters for common programming languages are often provided to the user with the operating system.

**Program loading and execution:** The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders.

**Communications:** These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. (email, FTP, Remote log in)

**Application programs:** Programs that are useful to solve common problems, or to perform common operations. Eg. Web browsers, database systems.

## Operating System Generation

Operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations.

The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN).

The operating system is normally distributed on disk or CD-ROM. The SYSGEN program reads from a given file, or asks the operator of the system for information.

**The following kinds of information must be determined.**

- What CPU is to be used?
- How much memory is available?
- What devices are available?
- What operating-system options are desired, or what parameter values are to be used?

The operating system then is completely compiled or generated. Data declarations, initializations, and constants, along with conditional compilation, produce an output object version of the operating system that is tailored to the system described.

All the code is always part of the system, and selection occurs at execution time, rather than at compile or link time.

## BOOT SYSTEM

After an operating system is generated, it must be made available for use by the hardware. The procedure of starting a computer by loading the kernel is known as booting the system.

A small piece of code known as the bootstrap program or bootstrap loader locates the kernel, loads it into main memory, and starts its execution.

When a CPU receives a reset event, when it is powered up or rebooted, the instruction register is loaded with a predefined memory location, and execution starts there.

Bootstrap program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup. All forms of ROM are also known as firmware. ROM is convenient because it needs no initialization and cannot be infected by a computer virus.

But executing code in firmware is slower than executing code in RAM. Some systems store the operating system in firmware and copy it to RAM for fast execution.

Some operating systems (windows, Mac OS X, and UNIX) store bootstrap program in firmware, and the operating system is on disk.

The bootstrap then runs a bit of code that can read a single block at a fixed location (say block zero) from disk into memory and execute the code from that boot block.

Now that the full bootstrap program has been loaded, it can traverse the file system to find the operating system kernel, load it into memory, and start its execution.

It is only at this point that the system is said to be running. The bootstrap program can also perform a variety of tasks. One task is to run diagnostics to determine the state of the machine.

If the diagnostics pass, the program can continue with the booting steps. It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory.