Operating System

UNIT -1

❖ Introduction to Operating Systems:

- An operating System (OS) is an intermediary between users and computer hardware. It provides users an environment in which a user can execute programs conveniently and efficiently.
- . An operating System controls the allocation of resources and services such as memory, processors, devices and information.
- A computer system has many resources (hardware and software), which may be require to complete a task. The commonly required resources are input/output devices, memory, file storage space, CPU etc.
- The operating system acts as a manager of the above resources and allocates them to specific programs and users, whenever necessary to perform a particular task.
- Therefore operating system is the resource manager i.e. it can manage the resource of a computer system internally.
- The resources are processor, memory, files, and I/O devices. In simple terms**, an operating system is the interface between the user and the machine.**

What is Operating System?

An operating system is an important part of almost every computer system**.** A computer system can be divided roughly into **four components:** the **hardware,** the **operating system,** the **applications programs,** and the **users (figure).**

Four Components of a Computer System

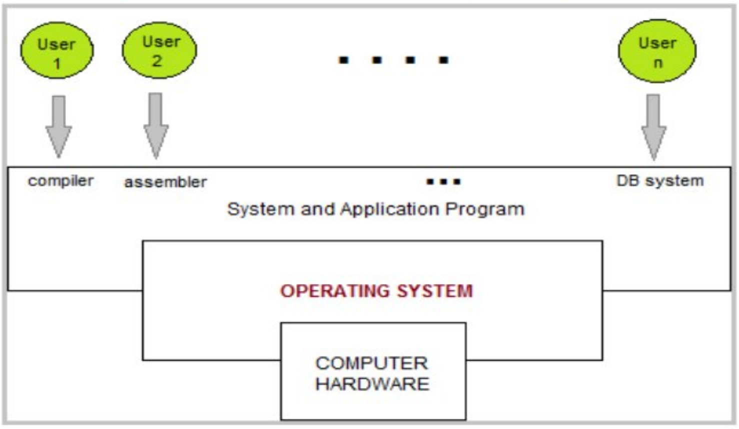


Figure: Abstract view of the components of a computer system

- The **hardware—**the **central processing unit (CPU),** the **memory,** and the **input/output (I/O) devices**—provides the basic computing resources for the system.
- The **application programs**—such as word processors, spreadsheets, compilers, and web browsers—define the ways in which these resources are used to solve the computing problems of the users.
- The operating system controls and coordinates the use of the hardware among the various application programs for the various users.
- The components of a computer system are its hardware, software, and data.
- The operating system provides the means for proper use of these resources in the operation of the computer system. An operating system is similar to a *government.*
- Operating systems can be explored from two viewpoints: the user and the system.

Two Views of Operating System

- User's View
- System View

User View:

- ➢ The user's view of the computer varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources.
- ➢ The goal is to maximize the work (or play) that the user is performing.
- ➢ A user sits at a terminal connected to a **mainframe** or **minicomputer.**
- ➢ In still other cases, users sit at **workstations** connected to networks of other workstations and **servers.**
- ➢ For example, embedded computers in home devices and automobiles may have numeric keypads andmay turn indicator lights on or off to show status, but they and their operating systems are designed primarily to run without user intervention.

System View:

- ➢ From the computer's point of view, the operating system is the program most intimately involved with the hardware. we can view an operating system as a **resource allocator.**
- ➢ A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. An operating system is a control program.
- ➢ A **control program** manages the execution of user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices

❖ Functions of Operating System:

- An operating system is a program that acts as a user-computer GUI (Graphical user interface). It controls the execution of all types of applications
- .The operating system performs the following functions in a device.

1. Instruction
2. Input/output Management
3. Memory Management
4. File Management
5. Processor Management
6. Job Priority
7. Special Control Program
8. Scheduling of resources and jobs
9. Security
10. Monitoring activities
11. Job accounting

- **Instruction:** The operating system establishes a mutual understanding between the various instructions given by the user.
- **Input/output Management:** What output will come from the input given by the user, the operating system runs this program.
- This management involves coordinating various input and output devices. It assigns the functions of those devices where one or more applications are executed.
- **Memory Management:** The operating system handles the responsibility of storing any data, system programs, and user programs in memory.
- This function of the operating system is called memory management.
- **File Management:** The operating system is helpful in making changes in the stored files and in replacing them. It also plays an important role in transferring various files to a device.
- **Processor Management:** The processor is the execution of a program that accomplishes the specified work in that program. It can be defined as an execution unit where a program runs.
- **Job Priority:** The work of job priority is creation and promotion. It determines what action should be done first in a computer system.
- **Special Control Program:** The operating systems make automatic changes to the task through specific control programs. These programs are called Special Control Program.
- **Scheduling of resources and jobs:** The operating system prepares the list of tasks to be performed for the device of the computer system.

- **Security:** Computer security is a very important aspect of any operating system. The reliability of an operating system is determined by how much better security it provides us. Modern operating systems use a firewall for security. A firewall is a security system that monitors every activity happening in the computer and blocks that activity in case of any threat.
- **Monitoring activities:** The operating system takes care of the activities of the computer system during various processes. This aborts the program if there are errors. The operating system sends instant messages to the user for any unexpected error in the input/output device. It also provides security to the system when the operating system is used in systems operated by multiple users. So that illegal users cannot get data from the system.
- **Job accounting:** It keeps track of time & resources used by various jobs and users.

Types of Operating Systems:

Following are some of the most widely used types of Operating system.

- Simple Batch System
- Multiprogramming Batch System
- Multiprocessor System
- Desktop System
- Distributed Operating System
- Clustered System
- Real time Operating System
- Handheld System

1. Simple Batch Systems:

The user has to submit a job (written on cards or tape) to a computer operator.

- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched In this type of system, there is **no direct interaction between user and the computer** together by type of languages and requirement.

- Then a special program, the monitor, manages the execution of each program in the batch. The monitor is always in the main memory and available for execution.

Advantages of Simple Batch Systems

1. No interaction between user and computer.
2. No mechanism to priorities the processes.

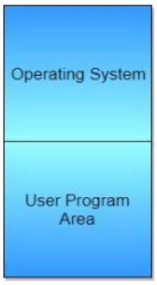


Figure: Memory layout for a simple batch system

2. Multiprogramming Batch Systems:

In this the operating system picks up and begins to execute one of the jobs from memory.

Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).

- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing Systems are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.

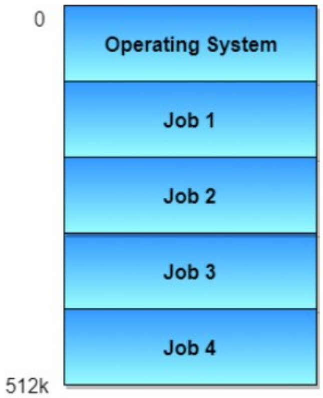


Figure: Memory layout for a multiprogramming system.

3. Multiprocessor system

- A Multiprocessor system consists of several processors that share a common physical memory.
- Multiprocessor system provides higher computing power and speed.
- In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.

Advantages of Multiprocessor Systems

1. Enhanced performance.
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

4. Desktop Systems:

- Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs.
- PC operating systems therefore were neither **multiuser** nor **multitasking**.
- However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness.
- These systems are called **Desktop Systems** and include PCs running Microsoft Windows and the Apple Macintosh.
- Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.
- **Microcomputers** were immediately able to adopt some of the technology developed for larger operating systems.
- On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern.

Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.

5. Distributed Operating System

- The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.
- These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks.
- The main benefit of distributed systems is its low price/performance ratio.

Advantages Distributed Operating System:

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

Types of Distributed Operating Systems

Following are the two types of distributed operating systems used:

1. Client-Server Systems
2. Peer-to-Peer Systems

Client-Server Systems

- **Centralized systems** today act as **server systems** to satisfy requests generated by **client systems**. The general structure of a client-server system is depicted in the figure below:

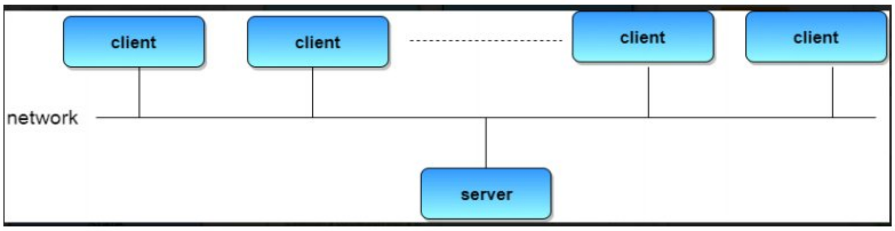


Figure: General structure of a client-server system.

- Server Systems can be broadly categorized as: **Compute Servers** and **File Servers**.

- **Compute Server systems**, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File Server systems**, provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems

- The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems.
- When PCs were introduced in the 1970s, they were designed for **personal** use and were generally considered standalone computers.
- With the beginning of widespread public use of the Internet in the 1990s for electronic mail and FTP, many PCs became connected to computer networks.
- In contrast to the **Tightly Coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock.

- Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines.
- These systems are usually referred to as loosely coupled systems (or distributed systems). The general structure of a client-server system is depicted in the figure below:

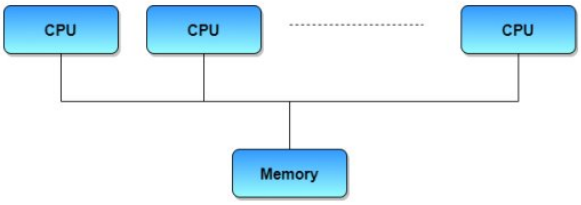


Figure: Symmetric multiprocessing architecture.

6. Clustered Systems

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine.
- The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
- **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.

- **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
- **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system's usage and it's features should expand greatly as **Storage Area Networks (SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

7. Real Time Operating System

- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- The Real-Time Operating systems which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**
- While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

8. Handheld Systems

- Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet.
- They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.
- Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently.
- This includes returning all allocated memory back to the memory manager once the memory is no longer being used.

- Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as **Bluetooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as cameras and MP3 players expand their utility.

Unit-1

Chapter-1

Computer System Operation

1. Computer System Operation:

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer , a temporary data storage holding place.
- Devices include such thinks as a printer, a disk, a tape drive, etc.
- CPU moves data from/to main memory to/from local buffers along the system bus, which is just a set of wire connections along which data can be sent.
- 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.

The modern computer system consists of the CPU and a number of device controllers are connected through a common bus that provides access to shared memory (figure).

Each device controller is in charge of a specific type of device (for example: disk drives, audio devices, and video display).

The CPU and the device controllers can execute concurrently, competing for memory cycles.

To ensure orderly access to the shared memory, a memory controller is provided whose function is to synchronize access to the memory.

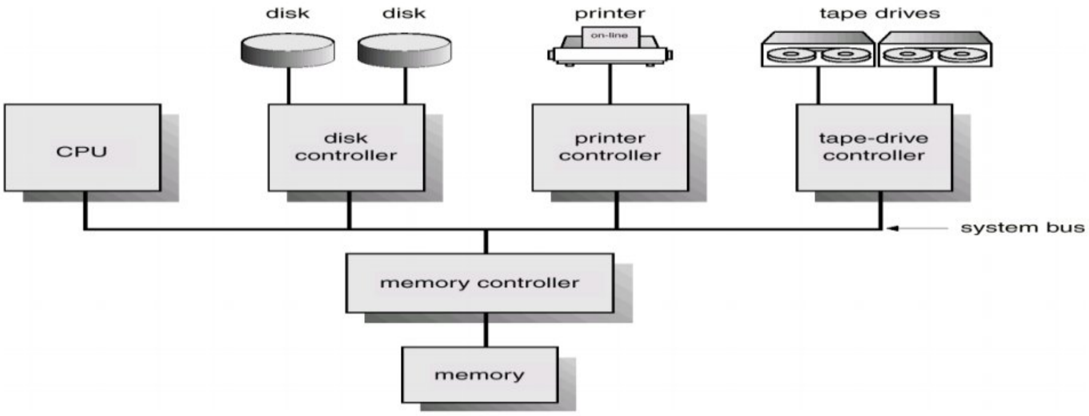


Figure: A modern computer system

- For a computer to start running – for instance, when it is powered up or rebooted—it needs to have an initial program to run.
- This initial program, or **bootstrap program,** tends to be simple.
- Typically, it is stored in **read-only memory (ROM)** such as firmware or EEPROM within the computer hardware.
- 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 to start executing that system.
- To accomplish this goal, the bootstrap program must locate and load into memory the operating-system kernel.
- The OS then starts executing the first process, such as “init,” and waits for some event to occur.
- The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software.
- H/W may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.
- S/W may trigger an interrupt by executing a special called a **system call** (also called a **monitor call**).

Modern OS are **interrupt driven.** If there are no processes to execute, no I/O devices to service, and no users to whom to responds, an OS will sit quietly, waiting for something to happen. Events are almost always signaled by the occurrence of an interrupt or a trap. A **trap** (or an **exception)** is a software-generated interrupt caused either by an error **(for example**, division by zero or invalid memory access) or by a specific request from a user program that an OS service be performed.

Common Functions of Interrupts:

- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt.
- A trap is a software-generated interrupt caused either by an error or a user request.
- An operating system is interrupt driven.

Interrupt Handling:

The operating system preserves the state of the CPU by storing registers and the program counter.

■ Determines which type of interrupt has occurred:

✦ Polling

✦ vectored interrupt system

■ Separate segments of code determine what action should be taken for each type of interrupt

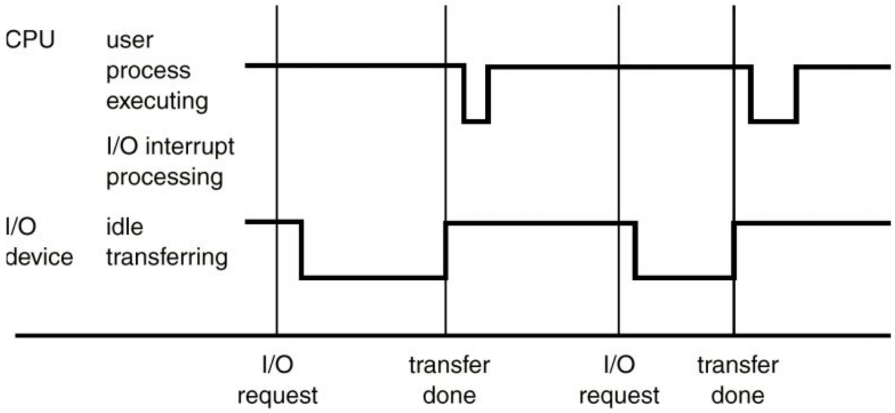


Fig: Interrupt Time Line for a Single Process Doing Output

2. I/O Structure:

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.

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

After I/O starts, control returns to user program only upon I/O completion.

✦ Wait instruction idles the CPU until the next interrupt
✦ Wait loop (contention for memory access).
✦ At most one I/O request is outstanding at a time, no simultaneous I/O processing.
■ After I/O starts, control returns to user program without waiting for I/O completion.
✦ System call – request to the operating system to allow user to wait for I/O completion.
✦ Device-status table contains entry for each I/O device indicating its type, address, and state.
✦ Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

- **I/O Interrupts**
- **DMA Structure**

I/O Interrupts:

- To start an I/O operation, the device driver 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 find a read request, 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 via an interrupt that it has finished its operation.
- The device driver then returns control to 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.

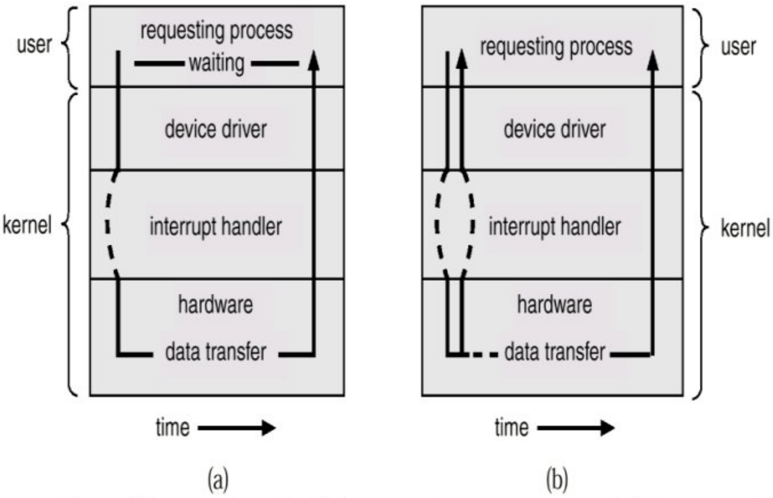


Figure: Two i/o methods (a) synchronous and (b) asynchronous

- Once the I/O is stared, two courses of action are possible. In the simple case, the I/O is stared; then, at I/O completion, control is returned to the user process. This case is known as **synchronous** I/O.
- The other possibility, called **asynchronous** I/O, returns control to the user program without waiting for the I/O to complete. The I/O then can continue while other system operations occur shown in figure.
- Waiting for I/O completion may be accomplished in one of two ways. Some computers have a special **wait** instruction that idles the CPU until the next interrupt.
- Machines that do not have such an instruction may have a wait loop:
  Loop: jmp Loop
- The CPU always waits for I/O completion, at most one I/O request is outstanding at a time.
- A system call is then needed to allow the user program to wait for I/O completion, if desired.
- If no user program are ready to run, and the operating system has no other work to do, we still require the **wait** instruction or idle loop, as before.
- We also need to be able to keep track of many I/O requests at the same time.
- The operating system uses a table containing an entry indicates the device: the **device-status table(figure).**
- Each table entry indicates the device's type, address, and state.
- If the device is busy with a request, the type of request and other parameters will be stored in the table entry for that device.
- Since it is possible for other processes to issues **requests** to the same device, the operating system also maintain a wait queue-a list of waiting requests-for each I/O device.
- If there additional requests waiting in the queue for this device, the operating system starts processing the next request.
- Finally,control is returned from the I/O interrupt.

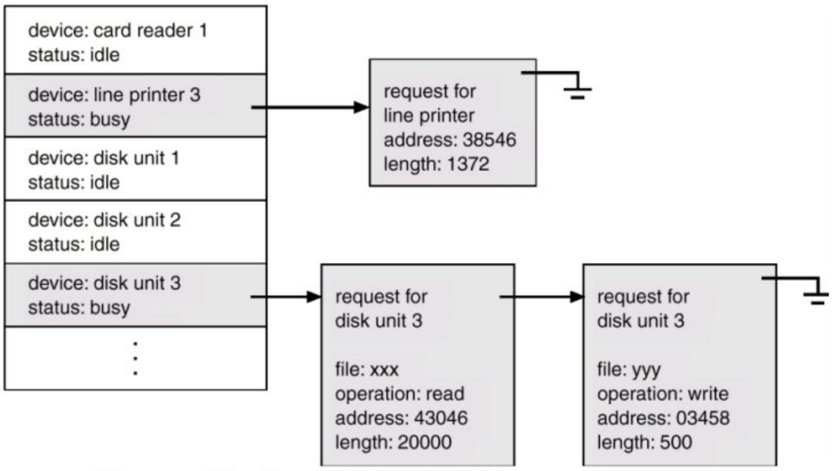


Figure: Device-Status Table

Direct Memory Access Structure:

- Direct memory access(DMA) is a method that allows an input /output (i/o) device to send or receive data directly to or from the main memory,by passing the CPU to speed up memory operations.
- The process is managed by a chip known as a DMA controller(DMAC).

➢ Used for high-speed I/O devices able to transmit information at close to memory speeds.

■ Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.

■ Only on interrupt is generated per block, rather than the one interrupt per byte.

Direct memory access (DMA) is a **mode of data transfer** between the memory and I/O devices. This happens **without the involvement** of the processor. We have two other methods of data transfer, **programmed I/O** and **Interrupt driven I/O**. Let's revise each and get acknowledge with their drawbacks.

In **programmed I/O**, the processor keeps on scanning whether any device is ready for data transfer. If an I/O device is ready, the processor **fully dedicates** itself in transferring the data between I/O and memory.

It transfers data at a **high rate, but it can't get involved in any other activity** during data transfer. This is the major **drawback** of programmed I/O.

In **Interrupt driven I/O**, whenever the device is ready for data transfer, then it raises an **interrupt to processor**. Processor completes executing its ongoing instruction and saves its current state. It then switches to data transfer which causes a **delay**.

Here, the processor doesn't keep scanning for peripherals ready for data transfer. But, it is **fully involved** in the data transfer process. So, it is also not an effective way of data transfer.

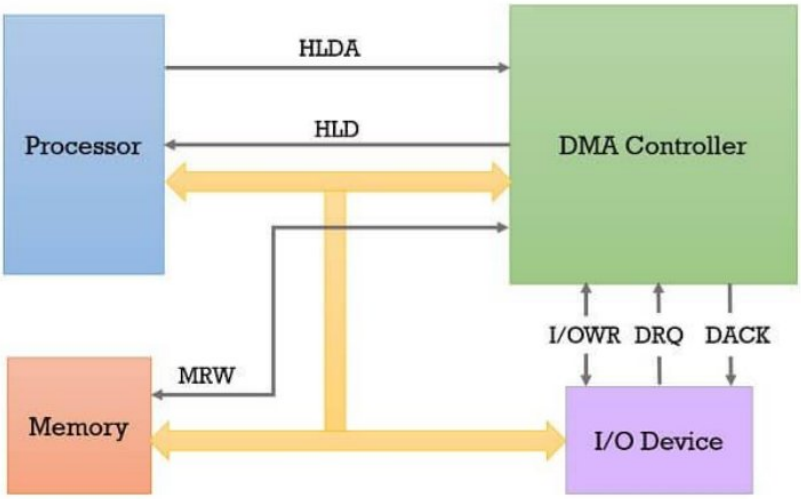
The above two modes of data transfer are not useful for transferring a large block of data. But, the DMA controller completes this task at a faster rate and is also effective for transfer of large data block.

The DMA controller transfers the data in **three modes:**

Burst Mode: Here, once the DMA controller gains the charge of the system bus, then it releases the system bus only after **completion** of data transfer. Till then the CPU has to wait for the system buses.

Cycle Stealing Mode: In this mode, the DMA controller **forces** the CPU to stop its operation and **relinquish the control over the bus** for a **short term** to DMA controller. After the **transfer of every byte**, the DMA controller **releases** the **bus** and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte.

Transparent Mode: Here, the DMA controller takes the charge of system bus only if the **processor does not require the system bus**.



DMA Controller Data Transfer

Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (**DRQ**) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (**HLD**).

CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (**HLDA**) to DMA controller.

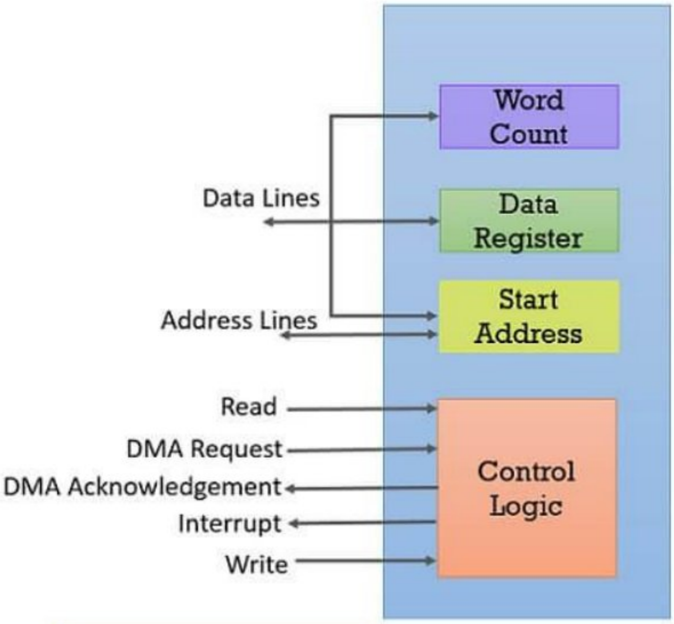
After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device **(DACK)** that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.

When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.

Now the DMA controller can be a separate unit that is shared by various I/O devices, or it can also be a part of the I/O device interface.

Direct Memory Access Diagram:

After exploring the working of DMA controller, let us discuss the block diagram of the DMA controller. Below we have a block diagram of DMA controller.



DMA Block Diagram

Whenever a processor is requested to read or write a block of data, i.e. transfer a block of data, it instructs the DMA controller by sending the following information.

The first information is whether the data has to be read from memory or the data has to be written to the memory. It passes this information via **read or write control lines** that is between the processor and DMA controllers **control logic unit**.

The processor also provides the **starting address** of/ for the data block in the memory, from where the data block in memory has to be read or where the data block has to be written in memory. DMA controller stores this in its **address register**. It is also called the **starting address register**.

The processor also sends the **word count**, i.e. how many words are to be read or written. It stores this information in the **data count** or the **word count** register.

The most important is the **address of I/O device** that wants to read or write data. This information is stored in the **data register.**

Advantages:

- Transferring the data without the involvement of the processor will **speed up** the read-write task.
- DMA **reduces the clock cycle** requires to read or write a block of data.
- Implementing DMA **also reduces the overhead of the processor.**

Disadvantages:

- As it is a hardware unit, it would **cost** to implement a DMA controller in the system.
- Cache **coherence** problem can occur while using DMA controller.

DMA controller transfers the data block at the **faster rate** as data is directly accessed by I/O devices and is not required to pass through the processor which save the clock cycles.

DMA controller transfers the block of data to and from memory in three modes **burst mode**, **cycle steal mode** and **transparent mode**.

DMA can be configured in various ways it can be a part of individual I/O devices, or all the peripherals attached to the system may share the same DMA controller.

Thus the DMA controller is a convenient mode of data transfer. It is preferred over the programmed I/O and Interrupt-driven I/O mode of data transfer.

3. Storage Structure:

- Computer programs must be in main memory(also called random-access memory or RAM) to be executed.

- Main memory – only large storage area(millions to billions of bytes) that the CPU can access directly.
- It is implemented in a semiconductor technology called **dynamic random-access memory (DRAM),** which forms an array of memory words.
- Each word has its own address. Interaction is achieved through a sequence of load or store instruction to specific memory addresses.
- The load instruction moves a word ,from main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to main memory.
- Aside from explicit load and store, the CPU automatically loads instructions from main memory for execution.

A typically instruction-execution cycle, as executed on a system with a **von Neumann architecture,** will first fetch instruction from memory and will store that instruction in the instruction register.

- ➢ The programs and data to reside in main memory permanently.
- ➢ This arrangement is not possible for the following two reasons:
  1. **Main memory is usually too small to store all needed programs and data permanently.**
  2. **Main memory is a volatile storage device that loses its contents when power is turned off or otherwise lost.**

- ➢ The most computer system provide **secondary storage** as an extension of main memory.
- ➢ The main requirements for secondary storage is that it be able to hold large quantities of data permanently.

✦ Disk surface is logically divided into tracks, which are subdivided into sectors.

✦ The disk controller determines the logical interaction between the device and the computer.

- ➢ The most common secondary –storage device is a **magnetic disk**, which provides storage of both programs and data.
- ➢ Most programs(wed browsers, compliers, word processors, spread sheets, etc) are store on a disk until they are loaded into memory.

1.Main Memory:

- Main memory refers to a physical memory that is the internal memory to the computer.
- Main memory is also known as RAM and primary storage.
- The computer is able to change only data that is in **main memory.**
- Main memory and the registers built into the processor itself are the only storage that the CPU can access directly.
- There are machine instructions that take memory addresses as arguments, but none that take disk addresses.
- Usually, special I/O instructions allows data transfers between these **registers** and **system memory.**
- To allow more convenient access to I/O devices, many computer architectures provide **memory-mapped I/O.**

- Read and writes to these memory addresses cause the data to be transferred to and from the device registers.
- Memory-mapped I/O is also convenient for other devices, such as the serial and parallel ports used to connect modem and printers to a computer.
- The CPU transfer data through these kinds of devices by reading and writing a few device registers, called an I/O **Port.**
- **A** memory buffer used to accommodate a speed differential, called a **cache.**

2. Magnetic Disks:

- Magnetic disks provide the bulk of secondary storage for modern computer system.
- Conceptually, disks are relatively simple shown in figure.

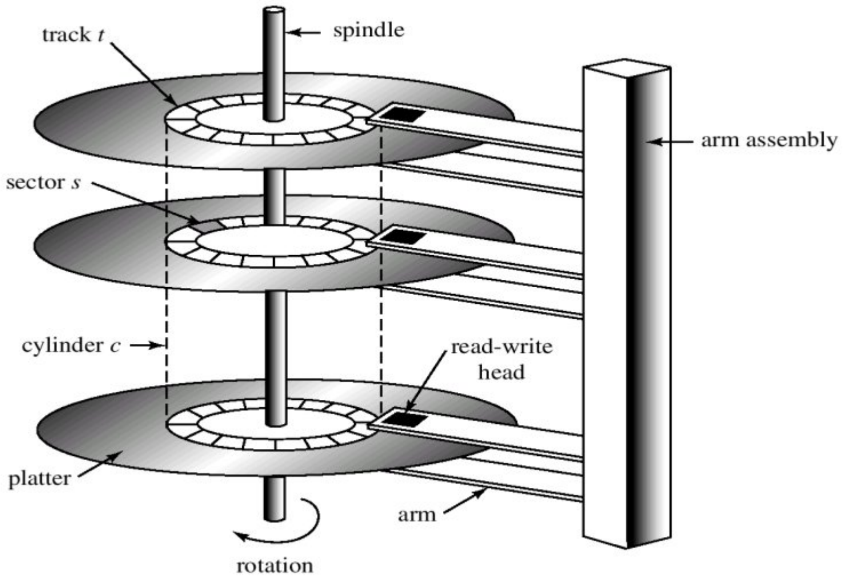


Figure: Moving-Head Disk Mechanism.

- Each disk **platter** has a flat circular shape, like a CD.
- Common platter diameters range from 1.8 to 5.25 inches.
- The two surface of a platter are covered with a magnetic material.
- We store information by recording it magnetically on the platters.

A read – write head "flies" just above each surface of every platter. The heads are attached to a **disk arm**, which moves all the heads as a unit. The surface of a platter is logically divided into circular **tracks**, which are subdivided into sectors. The set of tracks that are at one arm position forms a **cylinder**.

- When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 200 times per second. Disk speed has two parts. The **transfer rate** is the rate at which dataflow between the drive and the computer.
- The **positioning time**, sometimes called the **random-access time,** consists of the time to move the disk arm to the desired cylinder, called

the **seek time,** and the time for the desired sector to rotate to the disk head, called **rotational latency.**

- The disk platters are coated with a thin protective layer, sometimes the head will damage the magnetic surface. This accident is called a head crash.
  - ➢ A disk can be removable, allowing different disks to be mounted as needed.
  - ➢ Removable magnetic disks generally consists of one platter, held in a plastic case to prevent damage while not in the disk drive.
  - ➢ Floppy disks are inexpensive removable magnetic disks that a soft plastic case containing a flexible platter.
- **A** disk drive is attached to a computer by a set of wires called an I/O bus. Several kinds of buses are available, including **enhanced integrated drive electronics (EIDE), advanced technology attachment (ATA), and SCSI** buses.
- The data transfers on a bus are carried out by special electronic processors called **controllers**. The host controller is the controller at the computer end of the bus. A **disk controller** is build into each disk drive.

3.Magnetic Tapes:

- **Magnetic tape** is a medium for magnetic recording, made of a thin, coating on a long, narrow strip of plastic film.
- A device that stores computer data on magnetic tape is known as a tape drive.
- **Magnetic tape** was used as an early secondary-storage medium.
- Although it is relatively permanent and can hold large quantities of data, its access time is slow in comparision to that of main memory.
- A tape is kept in a spool, and is wound or rewound past a read – write head.
- Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives.
- Tape capacities vary greatly, depending on the particular kind of tape drive. Tape hold 2 to 3 times more data than some does a large disk drive.
- Tapea and their drivers are usually categorized by width, including 4,8, and 19 millimeters, 1/2 and1/2 inch.

4.Storage Hierarchy:

➢ Storage systems organized in hierarchy.

- Speed
- Cost
- Volatility

➢ Caching – copying information into faster storage system; main memory can be viewed as a last cache for secondary storage.

- In fact, many early storage devices, including paper tape and core memories, are relegated to museums now that magnetic tape and **semiconductor memory** have become faster and cheaper.
- The top three levels of memory shown in figure may be constructed using semiconductor memory.
- In addition to having differing speed and cost, the various storage systems are either **volatile or nonvolatile.**

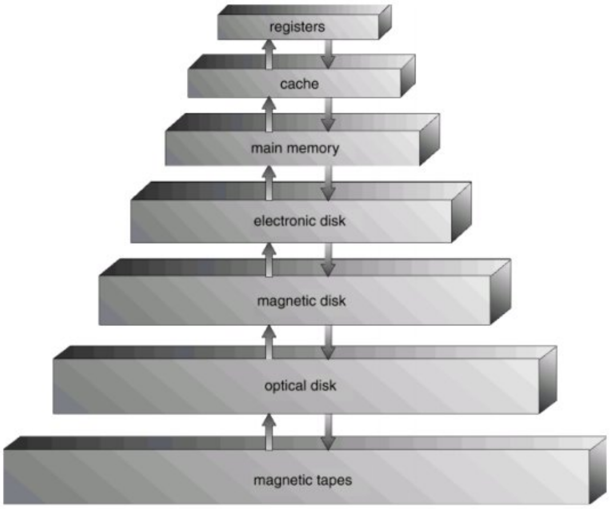
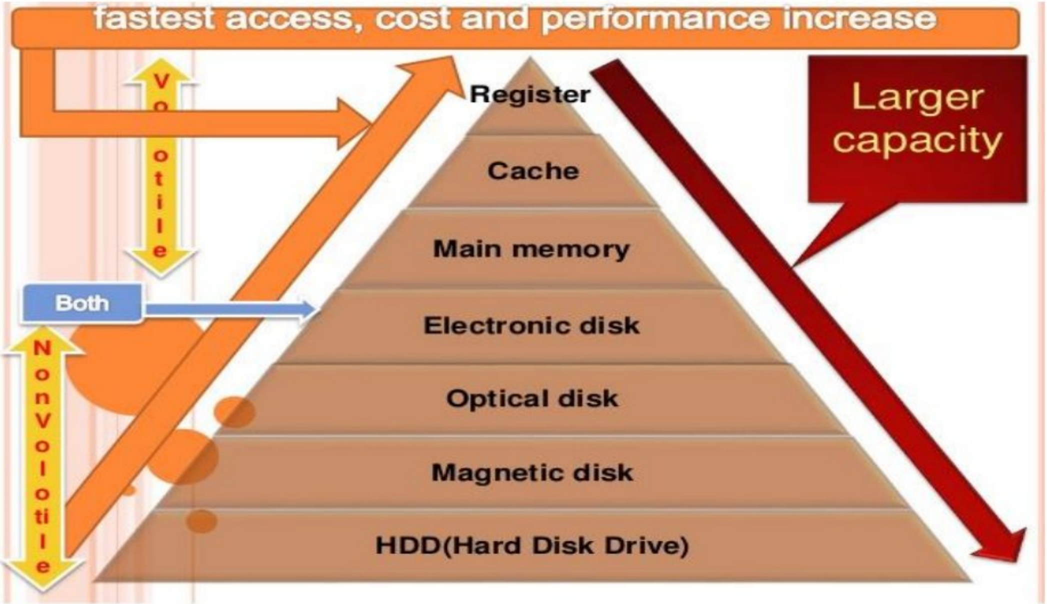


Figure: Storage-device hierarchy.

- **Volatile Storage** is a type of computer memory that needs power to preserve stored data. If the computer is switched off, anything stored in the volatile memory is removed or deleted.
- All random access memory (RAM) other than the CMOS RAM used in the BIOS is volatile. RAM is typically used as a primary storage or main memory in computer systems.
- Volatile storage is also known as volatile memory or temporary memory.
- **Non-volatile** memory is typically used for the task of **secondary storage**, or long-term persistent storage.
-



- The most widely used form of **primary storage** today is a volatile form of **random access memory (RAM),** meaning that when the computer is shut down, anything contained in RAM lost.
- In the hierarchy shown in figure, the storage systems above the **electronic disk are volatile,** whereas those below are nonvolatile.
- An electronic disk can be designed to be either volatile or nonvolatile.
- Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between **two components.**
- **1. Caching**
- **2. Coherency and Consistency**

Caching:

- **Caching** is an important principle of computer systems. Information is normally kept in some storage system (such as main memory).
- As it is used, it is copied into a faster storage system-the cache-on a temporary basis.
- If it is, we use the information directly from the cache; if it is not, we use the information from the main storage system, putting a copy in the cache under the assumption that we will need it again.
- Because caches have limited size, cache management is an important design problem.
- Careful selection of the cache size and of a replacement policy can result in **80 to 99 percent** of all accesses being in the cache, greatly increasing performance.
- Main memory can be viewed as a fast cache for secondary storage, since data in secondary storage must be copied into main memory for use, and data must be in main memory before being moved to secondary storage for safekeeping.
- The file-system data, which resides permanently on secondary storage, may appear on several levels in the storage hierarchy.

- At the highest level, the operating system may maintain a cache of file-system data in main memory.
- Also, electronic RAM disks (also known as solid-state disks) may be used for high-speed storage that is accessed through the file-system interface.
- The bulk of secondary storage is on magnetic disks.
- Use of high-speed memory to hold recently-accessed data.
  - ■ Requires a cache management policy.
  - ■ Caching introduces another level in storage hierarchy.
- This requires data that is simultaneously stored in more than one level to be consistent.

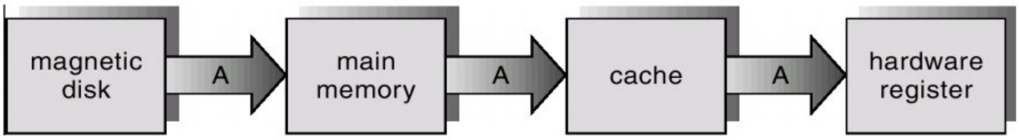


Figure: Migration of A from Disk to Register.

Coherency and Consistency:

- In a hierarchical storage structure, the same data may appear in different levels of the storage system.
- **For example**, suppose that an integer **A** is located in file **B** that is to be incremented by **1**, and file **B** resides on magnetic disk.
- The increment operation proceeds by first issuing an I/O operation to copy the disk block on which A resides to main memory.
- This operation is followed by copying A to the cache and to an internal register.
- Thus, the copy of A appears in several places:
  - On the magnetic disk,
  - In main memory,
  - In the cache and
  - In an internal register
- shown in figure. Once the increment takes place in the internal register, the value of A differs in the various storage systems.
- The value of A becomes the same only after the new value of A is written from the internal register back to the magnetic disk.
- ❖ The situation becomes more complicated in a multiprocessor environment where, in addition to maintaining internal registers, each of the CPUs also contains a local cache.
- ❖ In such an environment, a copy of A may exits simultaneously in several caches.
- ❖ Since the various CPUs can all execute concurrently, we must make sure that an update to the value of **A** in one cache is immediately reflected in all other caches where **A** resides.
- ❖ This situation is called **cache coherency**, and it is usually a hardware problem.

5.Hardware Protection

- Early computer systems were single-user programmer-operated systems.

- When the programmers operated the computer from the console, they had complete control over the system.
- As operating systems developed, however, this control was given to the operating system.
- Early operating systems were called **resident monitors,** and starting with the resident monitor, the operating system began to perform many of the functions, especially I/O, for which the programmer had previously been responsible.
- Without protection against these sorts of errors, either the computer must execute only one process at a time, or all output must be suspect.
- A properly designed operating system must ensure that an incorrect 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.

There are four components of hardware protection.

- **Dual-Mode Operation**
- **I/O Protection**
- **Memory Protection**
- **CPU Protection**

1. **Dual-Mode Operation:**

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.
- Provide hardware support to differentiate between at least two modes of operations.

1. User mode – execution done on behalf of a user.

2. Monitor mode (also called kernel mode or system mode, supervisor mode, or privileged mode) execution done on behalf of operating system.

- Mode bit added to computer hardware to indicate the current mode: monitor (0) or user (1)
- When an interrupt or fault occurs hardware switches to monitor mode.
- Privileged instructions can be issued only in monitor mode.
- 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 designating some of the machine instructions that may cause harm as **privileged instructions.**
- The hardware allows privileged instructions to be executed only in kernel mode.

2. **I/O Protection:**

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (i.e., a user program that, as part of its execution, stores a new address in the interrupt vector).

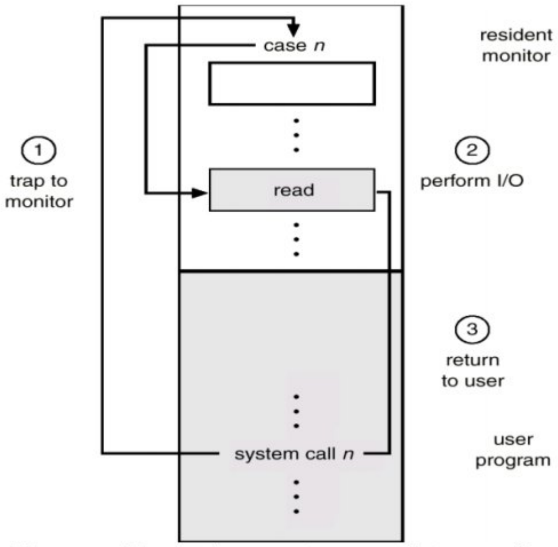


Figure: Use of a system call to perform I/O.

- A user program may disrupt the normal operation of the system by issuing illegal I/O instructions, by accessing memory locations within the operating system itself, or by refusing to give up the CPU.
- To prevent users from performing illegal I/O, we define all I/O instruction to be privileged instructions.
- Thus, users cannot issue I/O instructions directly; they must do it through the operating system.
- For I/O protection to be complete, we must be sure that a user program can never gain control of the computer in monitor mode.
- If it could, I/O protection could be compromised.
- Consider a computer executing in user mode.
- It will switch to monitor mode whenever an interrupt or trap occurs, jumping to the address determined from the interrupt vector.
- If a user program, as part of its execution, stores a new address in the interrupt vector, this new address could overwrite the previous address with an address in the user program.
- The user program could gain control of the computer in monitor mode.
- Thus, to do I/O, a user program executes a system call to request that the operating system perform I/O on its behalf shown in figure.
- The operating system, executing in monitor mode, checks that the request is valid, and does the I/O requested.
- The operating system then returns to the user.

3.Memory Protection:

- Even if the user did not gain unauthorized control of the computer, modifying the interrupt service routines would probably disrupt the proper operation of the computer system and of its spooling and buffering.
- Must provide memory protection at least for the interrupt vector and the interrupt service routines.
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
- We can provide this protection by using two register, usually **a base and a limit,** as illustrated in figure.
- **Base register –** holds the smallest legal physical memory address.
    - **Limit register** – contains the size of the range.
    - Memory outside the defined range is protected.

➢ For example, if the base register holds 300040 and limit register is 120900, then the program can legally access all addresses from 300040 through 420940 inclusive.

➢ 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 the monitor, which treats the attempt as a fatal error in figure.

➢ This scheme prevents the user program from modifying the code or data structures of either the operating system or other users.

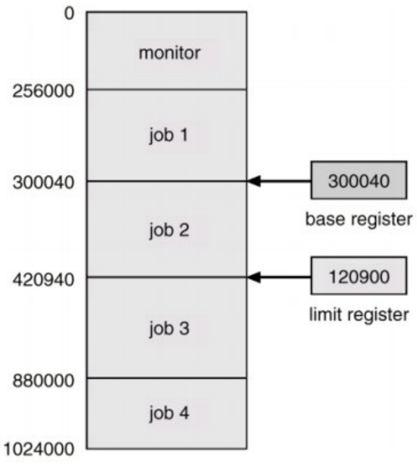


Figure: A base and a limit register define a logical address space.

➢ The base and limit registers can be loaded by only the operating system, which uses a special privileged instruction.

➢ Since privileged instructions can be executed in only monitor mode, and since only the operating system executes in monitor mode, only the operating system can load the base and limit registers.

- This scheme allows the monitor to change the value, but prevents user programs from changing the registers contents. The operating system, executing in monitor mode, is given unrestricted access to both monitor and users memory.

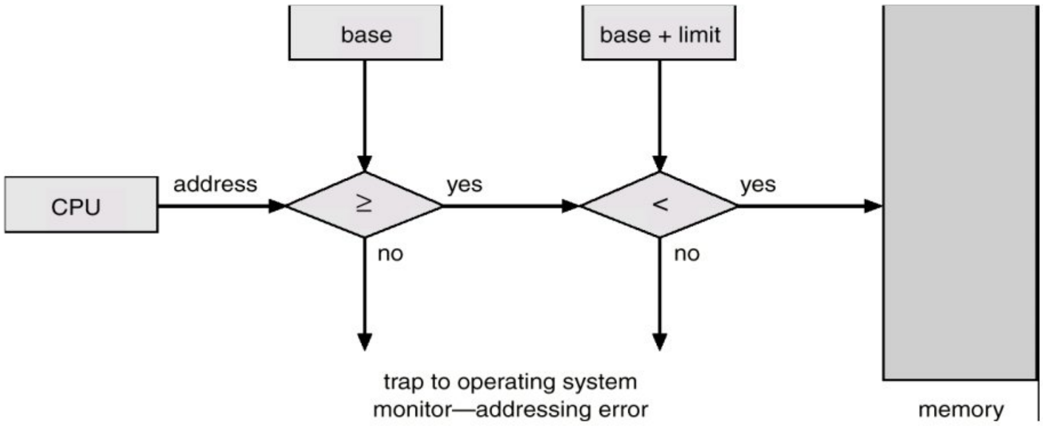


Figure: Hardware address protection with base and limit registers.

4.CPU PROTECTION:

- Timer – interrupts computer after specified period to ensure operating system maintains control.
- 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.
- Timer commonly used to implement time sharing.
- Time also used to compute the current time.
- Load-timer is a privileged instruction.
- Thus, we can use the timer to prevent a user program from running too long.
- A simple technique is to initialize a counter with the amount of time that a program is allowed to run.
- A program with a 7-minute time limit, for example, would have its counter initialized to 420.
- Every second, the timer interrupts and the counter is decremented by 1.
- As long as the counter is positive, control is returned to the user program.
- When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

- A more common use of a timer is to implement time sharing.

- ➢ In the most straightforward case, 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 at the end of each time slice to perform various housekeeping tasks, such as adding the value N to the record that specifies the amount of time the user program has executed.
- ➢ The operating system also saves registers, internal variables, and buffers, and changes several other parameters to prepare for the next program to run. This procedure is known as a **context switch.**

- ➢ Another use of the timer is to compute the current time. A timer interrupt signals the passage of second period, allowing the operating system to compute the current time in reference to some initial time.

6. NETWORK STRUCTURE:

Network: A network consists of two or more computers that are linked in order to share resources (such as printers and CDs), exchange files, or allow electronic communications.

- ➢ The Computers on a network may be linked through cables, telephone lines, radio waves, satellite, or infrared light beams.

There are basically two types of Networks.

1. Local-Area Network
2. Wide-Area Network

Local-Area Network:

- A Local Area Network (LAN) Is a private network that connects computers and devices within a limited area like a residence, an office, a building or a campus.
- Data Transfer rate is generally high. They range from 100Mbps to1000Mbps.
- In general a LAN uses only type of transmission medium, commonly category 5 coaxial cables.
- A LAN may be set up using wired or wireless connections.
- A LAN that is completely wireless is called Wireless LAN or WLAN.
- LAN uses either Ethernet or Token-ring technology.
- A Typical LAN may consists of a number of different computers, various shared peripheral devices(such as laser printers or magnetic –tape drives).

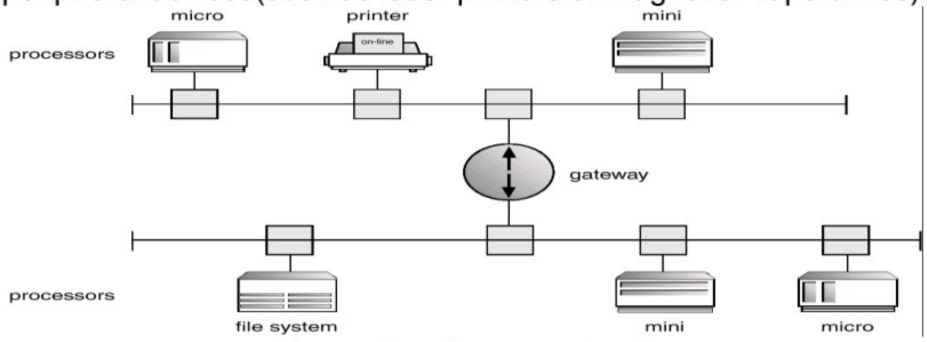


Figure: local-area network.

Wide-Area Network:

- A Wide area network (WAN) is a network that exists over a large-scale geographical area.
- WAN includes the technologies to transmit data, image, audio, and video information over long distances and among different LANs and MANs.
- The world's most popular WAN is the Internet.
- In, Wide Area Network, computers are connected through public networks such as the telephone system, fiber-optic cables, and satellite links or leased line.
- Typically they have low data transfer rate and high propagation delay, i.e. they have low communication speed.
- They generally have a higher bit error rate

Example of WAN

1. The Internet
2. 4G mobile broadband system
3. A network of bank cash dispensers

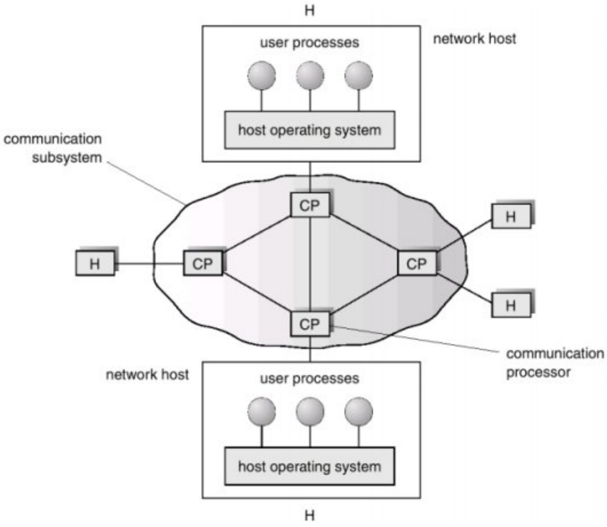


Figure: Wide-Area Network

Chapter 2

OPERATING SYSTEM COMPONENTS

1. OPERATING SYSTEM COMPONENTS:

- Operating system provides the environment within which programs are executed.
- To construct such an environment, the system is partitioned into small modules with a well-defined interface.
- The design of a new operating system is a major task. It is very important that the goals of the system be will defined before the design begins.
- The type of system desired is the foundation for choices between various algorithms and strategies that will be necessary.
- A system as large and complex as an operating system can only be created by partitioning it into smaller pieces.
- Each of these pieces should be a well defined portion of the system with carefully defined inputs, outputs, and function. Obviously, not all systems have the same structure.
- However, many modern operating systems share the system components outlined below:

1. **Process Management**
2. **Main-Memory Management**
3. **File Management**
4. **I/O-System Management**
5. **Secondary-Storage Management**
6. **Networking**
7. **Protection System**
8. **Command-Interpreter System**

Process Management:-

- The CPU executes a large number of programs. While its main concern is the execution of user programs, the CPU is also needed for other system activities. These activities are called processes.
- A process is a program in execution. Typically, a batch job is a process. A time-shared user program is a process. A system task, such as spooling, is also a process.
- In general, a process will need certain resources such as CPU time, memory, files, I/O devices, etc., to accomplish its task. These resources are given to the process when it is created.
- For example, a process whose function is to display on the screen of a terminal the status of a file, say F1, will get as an input the name of the file F1 and execute the appropriate program to obtain the desired information.

- A process is the unit of work in a system.
- Such a system consists of a collection of processes, some of which are operating system processes, those that execute system code, and the rest being user processes, those that execute user code.
- All of those processes can potentially execute concurrently.
- The operating system is responsible for the following activities in connection with processes managed.
  - **Creation and deletion of both user and system processes.**
  - **Suspending are resuming of processes.**
  - **Provision of mechanisms for process synchronization.**
  - **Providing mechanisms for process communication.**
  - **The provision of mechanisms for deadlock handling**.

Main Memory Management:

- Memory is central to the operation of a modern computer system. Memory is a large array of words or bytes, each with its own address.
- Interaction is achieved through a sequence of reads or writes of specific memory address. The CPU fetches from and stores in memory.
- In order for a program to be executed it must be mapped to absolute addresses and loaded in to memory.
- As the program executes, it accesses program instructions and data from memory by generating these absolute is declared available, and the next program may be loaded and executed.
- There are many different algorithms depends on the particular situation.
- Selection of a memory management scheme for a specific system depends upon many factor, but especially upon the hardware design of the system. Each algorithm requires its own hardware support.
- The operating system is responsible for the following activities in connection with memory management.
  - **Keep track of which parts of memory are currently being used and by whom.**
  - **Decide which processes are to be loaded into memory when memory space becomes available.**
  - **Allocating and deallocating memory space as needed.**

File Management:

- File management is one of the most visible services of an operating system.
- Computers can store information in several different physical forms; magnetic tape, disk, and drum are the most common forms.
- Each of these devices has it own characteristics and physical organization.

- For convenient use of the computer system, the operating system provides a uniform logical view of information storage.
- The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. Files are mapped, by the operating system, onto physical devices.
- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic or alphanumeric.
- Files is a sequence of bits, bytes, lines or records whose meaning is defined by its creator and user. It is a very general concept.
- The operating system implements the abstract concept of the file by managing mass storage device, such as types and disks. Also files are normally organized into directories to ease their use.
- Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed.
- ❖ The operating system is responsible for the following activities in connection with file management:

- ❖ **Creating and deleting files**
- ❖ **Creating and deleting directories**
- ❖ **Supporting of primitives for manipulating files and directories**
- ❖ **Mapping files onto secondary storage.**
- ❖ **Backup of files on stable (non volatile) storage media**.

I/O-System Management:

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user.
- **For example,** in Unix, the peculiarities of I/O devices are hidden from the bulk of the operating system itself by the I/O system.
- The I/O system consists of:
  - ❖ **A buffer caching system**
  - ❖ **A general device driver code**
  - ❖ **Drivers for specific hardware devices.**
  - ❖ **Only the device driver knows the peculiarities of a specific device**.

Secondary-Storage Management

- The main purpose of a computer system is to execute programs. These programs, together with the data they access, must be in main memory , or **primary memory** during execution.
- Since the main memory is too small to permanently accommodate all data and program, the computer system must provide secondary storage to backup main memory.

- Most modem computer systems use disks as the primary on-line storage of information, of both programs and data.
- Most programs, like compilers, assemblers, sort routines, editors, formatters, and so on, are stored on the disk until loaded into memory, and then use the disk as both the source and destination of their processing.
- Hence the proper management of disk storage is of central importance to a computer system.
- There are few alternatives. Magnetic tape systems are generally too slow. In addition, they are limited to sequential access. Thus tapes are more suited for storing infrequently used files, where speed is not a primary concern.

The operating system is responsible for the following activities in connection with disk management

- **Free space management**
- **Storage allocation**
- **Disk scheduling**.

Networking:

- A **distributed system** is a collection of processors that do not share memory or a clock.
- Instead, each processor has its own local memory, and the processors communicate with each other through various communication lines, such as high speed buses or telephone lines.
- Distributed systems vary in size and function.
- They may include **microprocessors, workstations, minicomputers, and large general purpose computer systems.**
- The processors in the system are connected through a communication network, which can be configured in the number of different ways.
- The network may be fully or partially connected.
- The communication network design must consider routing and connection strategies, and the problems of connection and security.
- A distributed system provides the user with access to the various resources the system maintains.
- Access to a shared resource allows computation speed-up, data availability, and reliability.
- The protocols that create a distributed system can have a great effects on that system's utility and popularity.
- The innovation of the **world wide web** was to create a new access method for information sharing.
- It improved on the existing **file-transfer protocol (FTP)** and **network file-system (NFS)** protocol by removing the need for a user to log in before she is allowed to use a remote resource.
- It defined a new protocol, hypertext transfer protocol (HTTP), for use in communication between a web server and a web browser.

- ❖ A web browser then just needs to send a request for information to a remote machine's web server, and the information (text, graphics, links to other information) is returned.

Protection System:

- The various processes in an operating system must be protected from each other's activities.
- For that purpose, various mechanisms which can be used to ensure that the files, memory segment, CPU and other resources can be operated on only by those processes that have gained proper authorization from the operating system.
- **For example**, memory addressing hardware ensure that a process can only execute within its own address space.
- The timer ensures that no process can gain control of the CPU without relinquishing it.
- Finally, no process is allowed to do it's own I/O, to protect the integrity of the various peripheral devices.
- **Protection** is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer controls to be imposed, together with some means of enforcement.
- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.
- Early detection of interface errors can often prevent contamination of a healthy subsystem by a subsystem that is malfunctioning.
- An unprotected resource cannot defend against use (or misuse) by an unauthorized or incompetent user.

Command-Interpreter System:

- ❖ One of the most important system for an operating system is its **command interpreter**. The command interpreter is the primary interface between the user and the rest of the system.
- ❖ Some operating systems include the command interpreter in the **kernel.** Other operating system , such as a MS-dos and UNIX.
- ❖ Many commands are given to the operating system by **control statements**. When a new job is started in a batch system or when a user logs-in to a time-shared system, a program which reads and interprets control statements is automatically executed.
- ❖ This program is variously called the **control card interpreter**, the **command line interpreter**, and is often known as the **shell**.
- ❖ Its function is quite simple: get the next command statement, and execute it.

- ➢ The command statement themselves deal with process management, I/O handling, secondary storage management, main memory management, file system access, protection, and networking.

2. OPERATING SYSTEM SERVICES:

- An operating system provides an environment for the execution of programs.It provides certain services to programs and to the users of those programs.
- These operating system services are provided for the convenience of the programmer, to make the programming task easier.

Operating –system services function:

User interface:

- Almost all operating systems have a **user interface** (UI).
- This interface can take several forms. One is a **command-line interface(CLI),** which uses text commands and a method.
- Another is a **batch interface,** in which commands and directives to control those commands are entered into files, and those files are executed.
- Most commonly/ a **graphical user interface (GUI)** is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text.

Program execution:

- The system must be able to load a program into memory and to run that program.
- The program must be able to end its execution, either normally or abnormally (indicating error).

I/O Operations:

- A running program may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a CRT screen).
- For efficiency and protection, users usually cannot control I/O devices directly.
- Therefore, the operating system must provide a means to do I/O.

File-system manipulation:

- The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information.
- Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership.

Communications:

- There are many circumstances in which one process needs to exchange information with another process.
- Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network.
- Communications may be implemented via **shared memory** or through **message passing,** in which packets of information are moved between processes by the operating system.

Error detection:

- The operating system needs to be constantly needs to be aware of possible errors.
- Errors may occur in the CPU and memory hardware (**such as a memory error or a power failure**),
- In I/O devices (**such as a parity error on tape, a connection failure on a network, or lack of paper in the printer**),
- And in the user program (**such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time**).
- For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

Resource allocation:

- When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
- Many different types of resources are managed by the operating system.
- Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code.
- There may also be routines to allocate printers, modems, USB storage drives, and other peripheral devices.

Accounting:

- We want to keep track of which users use how much and what kinds of computer resources.
- This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.
- Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

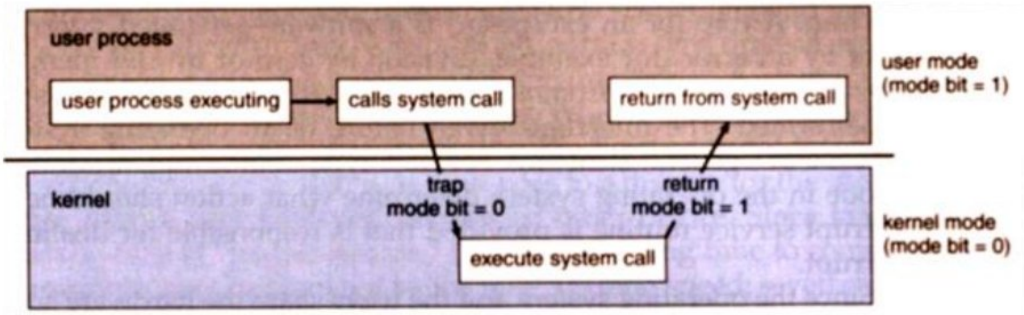
Protection and security:

- **Protection** involves ensuring that all access to system resources is controlled.
- **Security** of the system from outsiders is also important.
- If a system is to be protected and secure, precautions must be instituted throughout it.

3. SYSTEM CALLS:

- **System calls** provide the interface between a process and the operating system. These calls are generally available as assembly-language instruction, and they are usually listed in the various manuals used by the assembly-language programmers.
- Several languages-such as C, C++.and Perl- have been defined to replace assembly language for systems programming.

- These languages allows system calls to be made directly.
- **For example**: UNIX system calls may be invoked directly from a C or C++ program.
- As an example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file.
- The first input that the program will need is the names of the two files: the input file and the output file.



- Application developers often do not have direct access to the system calls, but can access them through an **Application Programming Interface (API).**
- The functions that are included in the API invoke the actual system calls. By using the API, certain benefits can be gained:

- Portability: as long a system supports an API, any program using that API can compile and run.
- Ease of Use: using the API can be significantly easier then using the actual system call.

System Call Parameters

Three general methods exist for passing parameters to the OS:

1. Parameters can be passed in registers.
2. When there are more parameters than registers, parameters can be stored in a block and the block address can be passed as a parameter to a register.
3. Parameters can also be pushed on or popped off the stack by the operating system.

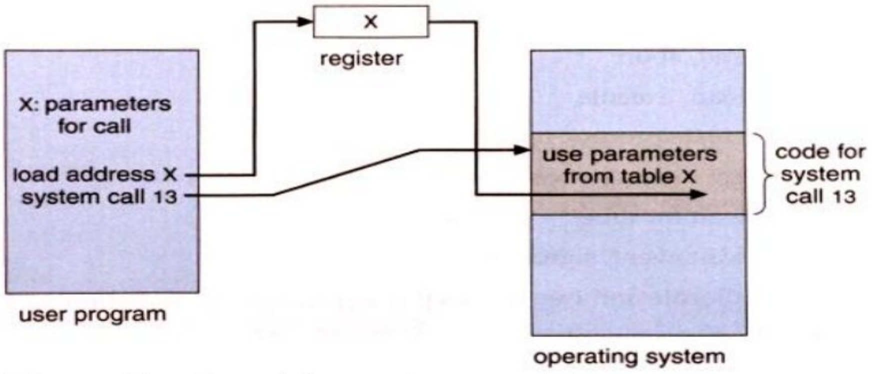
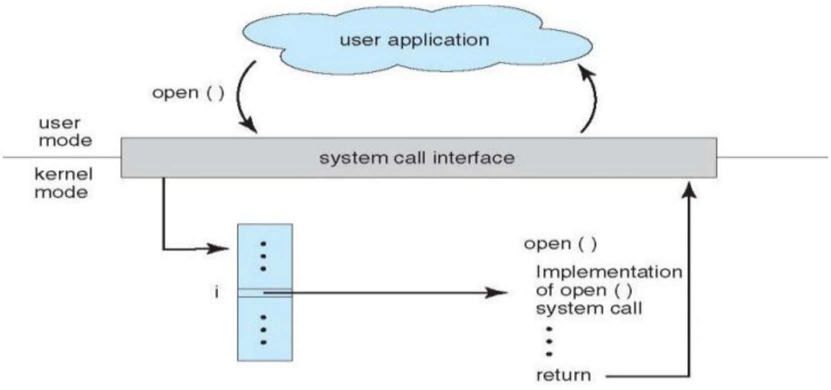
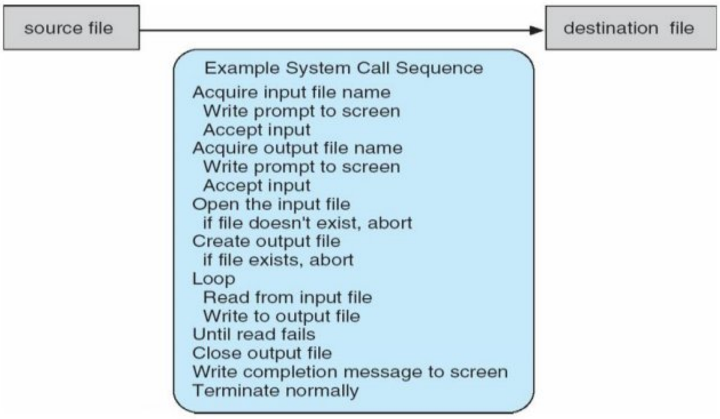


Figure: Passing of Parameters as a table



Example of system calls

- System call sequence to copy the contents of one file to another file



```
Example System Call Sequence
Acquire input file name
  Write prompt to screen
  Accept input
Acquire output file name
  Write prompt to screen
  Accept input
Open the input file
  if file doesn't exist, abort
Create output file
  if file exists, abort
Loop
  Read from input file
  Write to output file
Until read fails
Close output file
Write completion message to screen
Terminate normally
```

Types of System Calls

- System calls can be grouped roughly into five major categories:
  1. **Process control,**
  2. **File Management,**
  3. **Device Management,**
  4. **Information Maintenance,** and
  5. **Communication.**

1. Process Control

- A running program needs to be able to stop execution either normally or abnormally.
- When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.
- If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.
- The dump is written to disk and may be examined by a **debugger** to determine the cause of the problem.
- Under either normal or abnormal circumstances, the operating system must transfer control to the command interpreter.

- **Process control**
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
- **File management**
  - create file, delete file
  - open, close
  - read, write, reposition
  - get file attributes, set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- **Information maintenance**
  - 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**
  - create, delete communication connection
  - send, receive messages
  - transfer status information
  - attach or detach remote devices

Figure : Types of system calls.

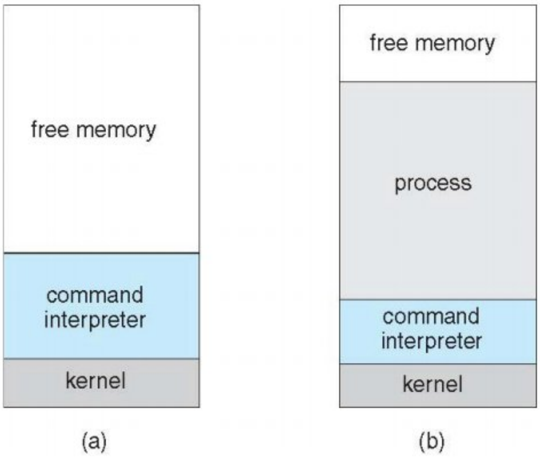


Figure: MS-DOS execution (a) At system startup (b) Running a program

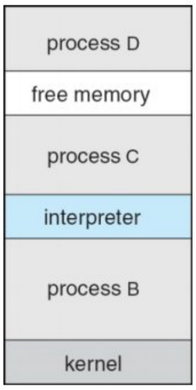


Figure: UNIX running multiple program

2.File Management

- We first need to be able to **create and delete files**. Either system call requires the name of the file and perhaps some of the file's attributes.
- Once the file is created, we need to open it and to use it. We may also **read, write, or reposition.**

- Some common system calls are *create*, *delete*, *read*, *write*, *reposition*, or *close*.
- File attributes include the file name, a file type, protection codes, accounting information.
- At least two system calls, get file attribute and set file attribute, are required for this function.
- Some operating system provide many more system calls.

3. Device Management

- A program, as it is running, may need additional resources to proceed. Additional resources may be more memory, tape drives, access to files.
- Process usually require several resources to execute, if these resources are available, they will be granted and control returned to the user program; otherwise, the program will have to wait until sufficient resources are available.
- These resources are also thought of as devices. Some are physical, such as a video card, and others are abstract, such as a file.
- User programs request the device, and when finished they release the device. Similar to files, we can read, write, and reposition the device.

4. Information Management

- Some system calls exist purely for transferring information between the user program and the operating system.
- For example, most system have a system call to return the current **time and date**.
- 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

- The OS also keeps information about all its processes and provides system calls to report this information.

5. Communication

- There are two models of inter process communication, the **message-passing model** and the **shared memory model.**
- In the message-passing model, the communicating processes exchange messages with one another to transfer information.
- 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, be it another process on the same system or a process on another computer connected by a communications network.

- Each computer in a network has a *host name* by which it is commonly known. A host also has a network identifier, such as an IP address.
- The get host id and get processed system calls do this translation.
- The identifiers are then passed to the general purpose **open and close** calls provided by the file system or to specific **open connection and close connection** system calls, depending on the system's model of communication.

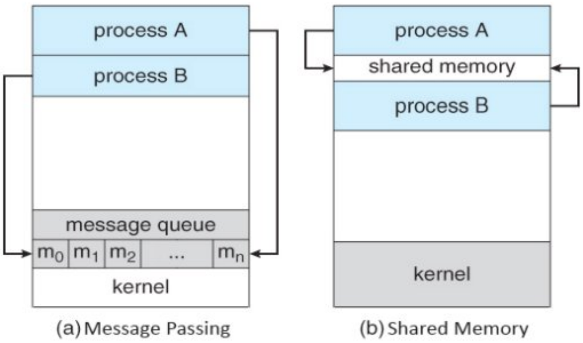


Figure: Communications models. (a) Msg passing. (b) Shared memory.

- Message-passing uses a common mailbox to pass messages between processes.
- Both of the models just discussed are common in operating systems, and most systems implement both.
- Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided.
- It is also easier to implement than is shared memory for inter computer communication.

4. System Programs:

- Another aspect of a modern system is the collection of system programs.

At the lowest level is hardware. Next is the operating system, then the system programs, and finally the application programs.

- System programs provide a convenient environment for program development and execution.
- Some of them are simply userinterfaces to system calls; others are considerably more complex.
- System programs provide a convenient environment for program development and execution. The can be divided into:
  - **File manipulation**
  - **Status information**
  - **File modification**
  - **Programming language support**
  - **Program loading and execution**

- **Communications**

- **File manipulation :** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
- **Status information.** Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information.
- Others are more complex, providing detailed performance, logging, and debugging information.
- Typically, these programs format and print the output to the terminal or other output devices or files.
- Some systems also support a registry, which is used to store and retrieve configuration information.

- **File modification.** Several text editors may be available to create and modify the content of files stored on disk or other storage devices.
- **Programming-language support.** Compilers, assemblers, debuggers and interpreters for common programming languages (such as C, C++, Java, Visual Basic, and PERL) are often provided to the user with the operating system.
- **Program loading and execution.** Once a program is assembled or compiled, it must be loaded into memory to be executed.
- The system may provide absolute loaders, relocate able loaders, linkage editors, and overlay loaders.
- Debugging systems for either higher-level languages or machine language are needed as well.
- **Communications.** These programs provide the mechanism for creating virtual connections among processes, users, and computer system.
- They allow users to send messages to one another's screens, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.
- In addition to systems programs, most operating systems are supplied with programs that are useful in solving common problems or performing common operations.
- Such programs include web browsers, word processors and text formatters, spreadsheets, database systems, compilers, plotting and statistical-analysis packages, and games. These programs are known as **system utilities** or **application programs.**

5. SYSTEM STRUCTURE:

- A system as large and complex as a modern operating system must be engineered carefully if it is to function properly and be modified easily.
- A common approach is to partition the task into small components rather than have one monolithic system.

- Each of these modules should be a well-defined portion of the system, with carefully defined inputs, outputs, and functions.
- MS-DOS –written to provide the most functionality in the least space
- Not divided into modules
- Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.

1.Simple structure:

- Many commercial systems do not have well-defined structures.
- Frequently, such operating systems started **as small, simple, and limited systems** and then grew beyond their original scope.
- **MS-DOS is an example of such a system.**
- It was written to provide the most functionality in the least space, so it was not divided into modules carefully. Figure shows in its **structure.**
- Another example of limited structuring is the original UNIX operating system. **UNIX** is another system that initially was limited by hardware functionality.
- It consists of two separable parts**: the kernel** and **the system programs**.
- The kernel is further separated into a series of interfaces and device drivers, which have been added and expanded over the years as UNIX has evolved.
- We can view the traditional UNIX operating system as being layered, as shown in Figure.
- Everything below the system call interface and above the physical hardware is the kernel.
- The kernel provides the **file system, CPU scheduling, memory management**, and other operating-system functions through system calls. Taken in sum, that is an enormous amount of functionality to be combined into one level.
- This monolithic structure was difficult to implement and maintain.
- With proper hardware support, operating systems can be broken into pieces that are smaller and more appropriate than those allowed by the original MS-DOS or UNIX systems.
- The operating system can then retain much greater control over the computer and over the applications that make use of that computer.
- Implementers have more freedom in changing the inner workings of the system and in creating modular operating systems. Under the top-down approach, the overall functionality and features are determined and are separated into components.

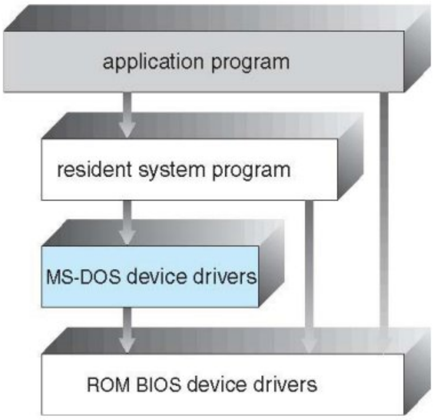


FIGURE: MS-DOS LAYERED STRUCTURE

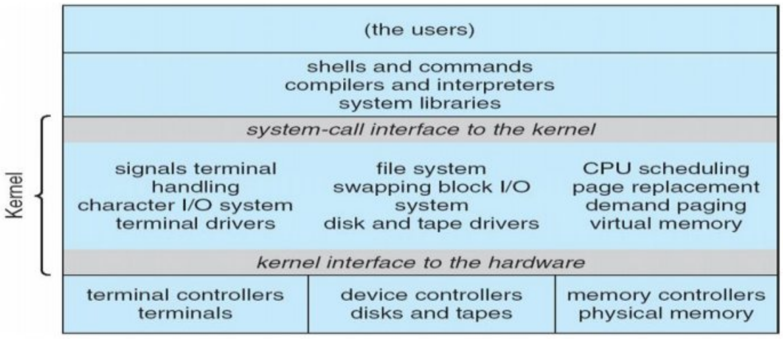


Figure: UNIX system structure.

2.Layered Approach:

- A system can be made modular in many ways.
- One method is the **layered approach,** in which the operating system is broken up into a number of layers (levels).
- The **bottom layer (layer 0**) is the hardware; the **highest (layer N)** is the user interface. This layering structure is depicted in Figure.
- An operating-system layer is an implementation of an abstract object made up of data and the operations that can manipulate those data.
- A typical operating-system layer—say, layer M—consists of data structures and a set of routines that can be invoked by higher-level layers. Layer *M,* in turn, can invoke operations on lower-level layers.
- The **main advantage of the layered approach is simplicity of construction and debugging.**
- The layers are selected so that each uses functions (operations) and services of only lower-level layers.
- This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, definition, it uses only the basic hardware (which is assumed correct) to implement its functions.

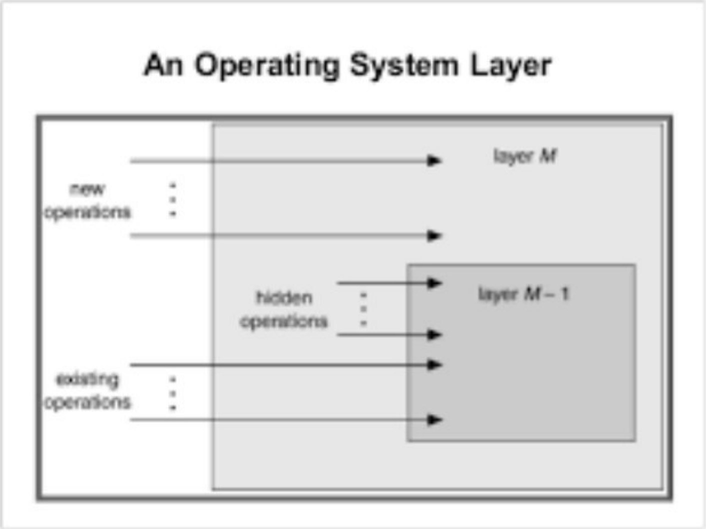


Figure : An Operating-System layer

- Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.
- Thus, the design and implementation of the system is simplified. Each layer is implemented with only those operations provided by lower-level layers.
- A layer does not need to know how these operations are implemented; it needs to know only what these operations do.
- Hence, each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.
- The major difficulty with the layered approach involves appropriately defining the various layers. Because a layer can use only lower-level layers, careful planning is necessary.

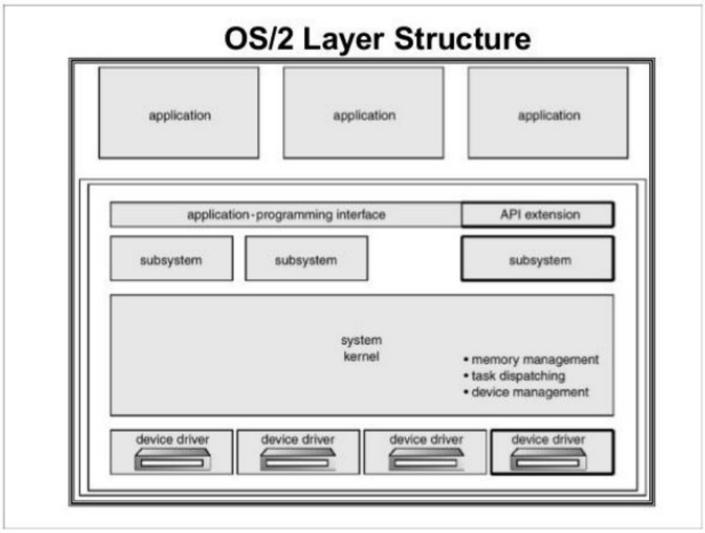


Figure : OS/2 layer structure.

- For example, the device driver for the backing store (disk space used by virtual-memory algorithms) must be at a lower level than the memory-management.
- The backing-store driver would normally be above the CPU scheduler, because the driver may need to wait for I/O and the CPU can be rescheduled during this time.
- A final problem with layered implementations is that they tend to be less efficient than other types.
- For instance, when a user program executes an I/O operation, it executes a system call that is trapped to the I/O layer, which calls the memory-management layer, which in turn calls the CPU-scheduling layer, which is then passed to the hardware.
- At each layer, the parameters may be modified, data may need to be passed, and so on.
- Each layer adds overhead to the system call; the net result is a system call that takes longer than does one on a non layered system.
- These limitations have caused a small backlash against layering in recent years.
- Fewer layers with more functionality are being designed, providing most of the advantages of modularized code while avoiding the difficult problems of layer definition and interaction.

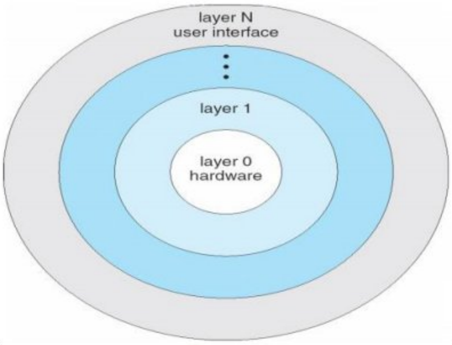


Figure: A layer operating system

3.Microkernel:

- As the UNIX operating system expanded, the kernel became large and difficult to manage.
- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the kernel using the **microkernel** approach.
- This method structures the operating system by removing all nonessential components from the kernel and implementing them as system and user-level programs.
- The result is a smaller kernel. There is little consensus regarding which services should remain in the kernel and which should be implemented in user space. Typically, however, micro kernels provide minimal process and memory management, in addition to a communication facility.
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.
- Communication is provided by *message passing.*
- For example, if the client program wishes to access a file, it must interact with the file server.
- The client program and the service never interact directly.
- Rather, they communicate indirectly by exchanging messages with the microkernel.
- The benefit of the microkernel approach is ease of extending the operating system.
- The resulting operating system is easier to port from one hardware design to another. The microkernel also provides more security and reliability, since most services are running as user—rather than kernel—processes.
- If a service fails, the rest of the operating system remains untouched.
- QNX is a real-time operating system that is also based on the microkernel design.
- The QNX microkernel provides services for message passing and process scheduling.
- It also handles low-level network communication and hardware interrupts.
- All other services in QNX are provided by standard processes that run outside the kernel in user mode.

- ➢ Unfortunately, micro kernels can suffer from performance decreases due to increased system function overhead.
- ➢ Consider the history of Windows NT.
- ➢ The first release had a layered microkernel organization.
- ➢ However, this version delivered low performance compared with that of Windows 95.
- ➢ Windows NT 4.0 partially redressed the performance problem by moving layers from user space to kernel space and integrating them more closely.
- ➢ By the time Windows XP was designed, its architecture was more monolithic than microkernel.

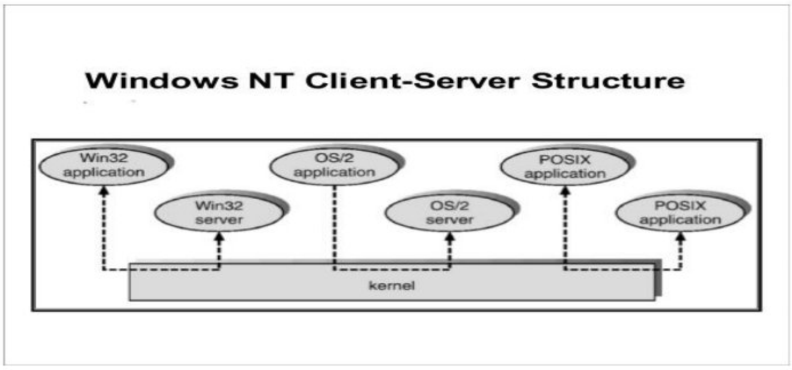


Figure: Windows NT client-server structure.

Microkernel System Structure

- ➢ Moves as much from the kernel into "*user*"space
- ➢ Communication takes place between user modules using message passing

Benefits:

- Easier to extend a microkernel
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

6. VIRTUAL MACHINES:

- ➢ A **virtual machine** takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- ➢ A virtual machine provides an interface identical to the underlying bare hardware.
- ➢ The operating system **host** creates the illusion that a process has its own processor and (virtual memory).
- ➢ Each **guest** provided with a (virtual) copy of underlying computer.
- ➢ The layered approach described is taken to its logical conclusion in the concept of a **virtual machine.**

- The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.
- A major difficulty with the virtual-machine approach involves disk systems.
- Suppose that the physical machine has three disk drives but wants to support seven virtual machines.
- Clearly, it cannot allocate a disk drive to each virtual machine, because the virtual-machine software itself will need substantial disk space to provide virtual memory and spooling.
- The solution is to provide virtual disks—termed *minidisks* in IBM's VM operating system—that are identical in all respects except size.
- The system implements each minidisk by allocating as many tracks on the physical disks as the minidisk needs.
- Obviously, the sum of the sizes of all minidisks must be smaller than the size of the physical disk space available.

Virtual Machines History and Benefits

- First appeared commercially in IBM mainframes in 1972
- Fundamentally, multiple execution environments (different operating systems) can share
  the same hardware,
- Protect from each other,
- Some sharing of file can be permitted, controlled, Commutate with each other, other physical systems via networking Useful for development,
- Testing **Consolidation** of many low-resource use systems onto fewer busier systems.
- "Open Virtual Machine Format", standard format of virtual machines, allows a VM to run within many different virtual machine (host) platforms.

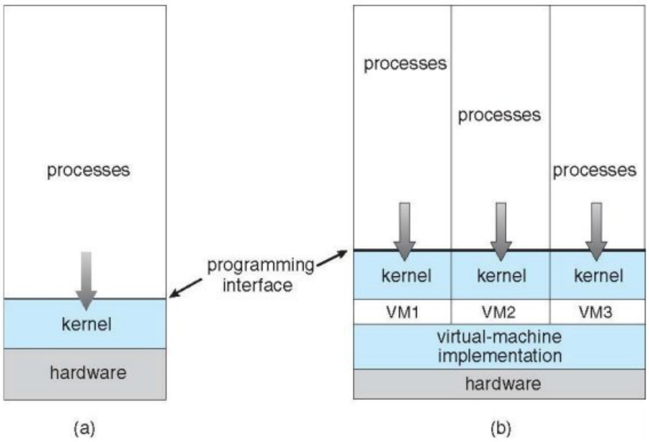


FIGURE: System models (a) Non virtual machine (b) virtual machine.

1.Implementation:

- Although the virtual-machine concept is useful, it is difficult to implement.
- Much work is required to provide an *exact* duplicate of the underlying machine.
- Remember that the underlying machine has two modes: **user mode and kernel mode**.
- The virtual-machine software can run in kernel mode, since it is the operating system. The virtual machine itself can execute in only user mode.
- Just as the physical machine has two modes, however, so must the virtual machine.
- Consequently, we must have a virtual user mode and a virtual kernel mode, both of which run in a physical user mode.
- Those actions that cause a transfer from user mode to kernel mode on a real machine (such as a system call or an attempt to execute a privileged instruction) must also cause a transfer from virtual user mode to virtual kernel mode on a virtual machine.
- It can then restart the virtual machine, noting that it is now in virtual kernel mode.
- The major difference, of course, is time.
- Whereas the real I/O might have taken 100 milliseconds, the virtual I/O might take less time (because it is spooled) or more time (because it is interpreted).
- In addition, the CPU is being multi programmed among many virtual machines, further slowing down the virtual machines in unpredictable ways.
- In the extreme case, it may be necessary to simulate all instructions to provide a true virtual machine.
- VM works for IBM machines because normal instructions for the virtual machines can execute directly on the hardware.
- Only the privileged instructions (needed mainly for I/O) must be simulated and hence execute more slowly.

2.Benefits:

- There are two primary advantages to using virtual machines.
- First, by completely protecting system resources, the virtual machine provides a robust level of security.
- Second, the virtual machine allows system development to be done without troublesome normal system operation.
- Each virtual machine is completely isolated from all other virtual machines, so we have no security problem as the various system resources are completely protected.

- At the same time, however, there is no direct sharing of resources. Two approaches to provide sharing have been implemented
- First, it is possible to share a minidisk and thus to share files. This scheme is modeled after a physical shared disk but is implemented by software.

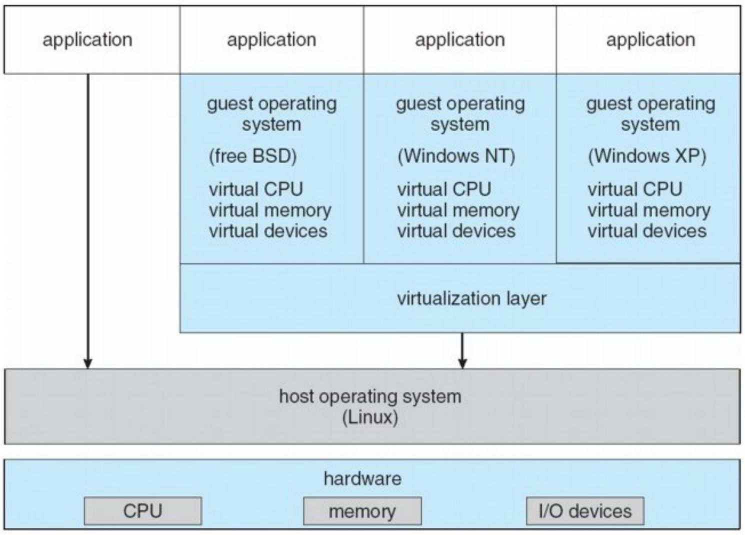
- Second, it is possible to define a network of virtual machines, each of which can send information over the virtual communications network.
- Again, the network is modeled after physical communication networks but is implemented in software.
- Operating systems are large and complex programs, and it is difficult to be sure that a change in one part will not cause obscure bugs in some other part.
- The operating system, however, runs on and controls the entire machine.
- Therefore, the current system must be stopped and taken out of use while changes are made and tested.

- This period is commonly called ***system development time.*** Since it makes the system unavailable to users, system development **n** time is often scheduled late at night or on weekends, when system load is low.
- A virtual-machine system can eliminate much of this problem. System programmers are given their own virtual machine, and system development is done on the virtual machine instead of on a physical machine.
- Normal system operation seldom needs to be disrupted for system development.

Examples

- Despite the advantages of virtual machines, they received little attention

for a number of years after they were first developed.



3.The Java Virtual Machine:

- Java is a very popular object-oriented programming language introduced by Sun Microsystems in 1995.
- In addition to a language specification and a large API library, Java also provides a specification for a **Java virtual machine—**or **JVM.**
- Java objects are specified with the **class** construct; a Java program consists of one or more classes.

- For each Java class, the compiler produces an architecture-neutral **byte code** output (.class) file that will run on any implementation of the JVM.
- The JVM is a specification for an abstract computer. It consists of a **class loader, a class verifier,** and a Java interpreter that executes the architecture-neutral byte codes, as diagrammed shown in Figure.

- The class loader loads the compiled **.class** files from both the Java program and the Java API for execution by the Java interpreter.
- After a class is loaded, the verifier checks that the **.class** file is valid Java byte code and does not overflow or underflow the stack.
- It also ensures that the byte code does not perform pointer arithmetic, which could provide illegal memory access.
- If the class passes verification, it is run by the Java interpreter.
- The JVM also automatically manages memory by performing **garbage collection**—the practice of reclaiming memory from objects no longer in use and returning it to the system.
- Much research focuses on garbage collection algorithms for increasing the performance of Java programs in the virtual machine.
- The JVM may be implemented in software on top of a host operating system, such as Windows, Linux, or Mac OS X, or as part of a web browser.
- Alternatively, the JVM may be implemented in hardware on a chip specifically designed to run Java programs. If the JVM is implemented in software, the Java interpreter interprets the byte code operations one at a time.
- A faster software technique is to use a **just-in-time (JIT)** compiler.

- Here, the first time a Java method is invoked, the byte codes for the method are turned into native machine language for the host system.
- These operations are then cached so that subsequent invocations of a method are performed using the native machine instructions and the byte code operations need not be interpreted all over again.
- A technique that is potentially even faster is to run the JVM in hardware on a special Java chip that executes the Java byte code operations as native code, thus bypassing the need for either a software interpreter or a just-in-time compiler.

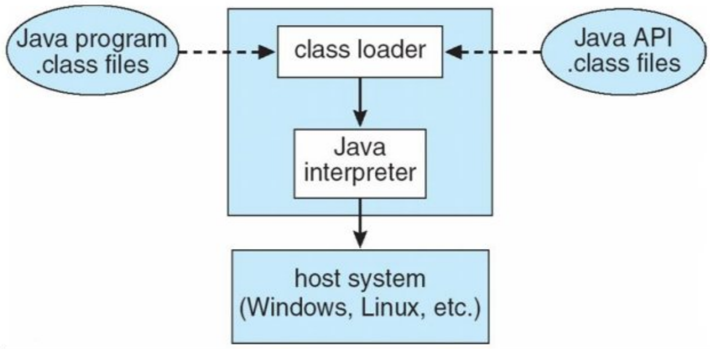


Figure: The Java Virtual Machine

- Java takes advantages of the complete environment that a virtual machine implements.
- Its virtual-machine design provides a secure, efficient, object-orient, portable, and architecture-neutral platform on which to run java programs.

7. SYSTEM DESIGN AND IMPLEMENTATION:

1. Design Goals:

- The first problem in designing a system is to define goals and specifications.At the highest level, the design of the system will be affected by the choice of hardware and the type of system: It is quite complicated to define all the goals and specifications of the operating system while designing it.

- The design changes depending on the type of the operating system ie, if it is **batch system, time shared system, single user system, multi user system, distributed system etc.**

- At the highest level, system design is dominated by the choice of hardware and system type.
- Beyond this level, the requirements can be divided into two groups: **user goals, and system goals.**
- **User goals:**
- **User goals** include convenience, reliability, security, and speed.
- The operating system should be convenient, easy to use, reliable, safe and fast according to the users.
- However, these specifications are not very useful as there is no set method to achieve these goals.
- **System goals:**
- **System goals** include ease of design, implementation, maintenance, flexibility, and efficiency.
- The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system.
- But there is not specific method to achieve these goals as well.

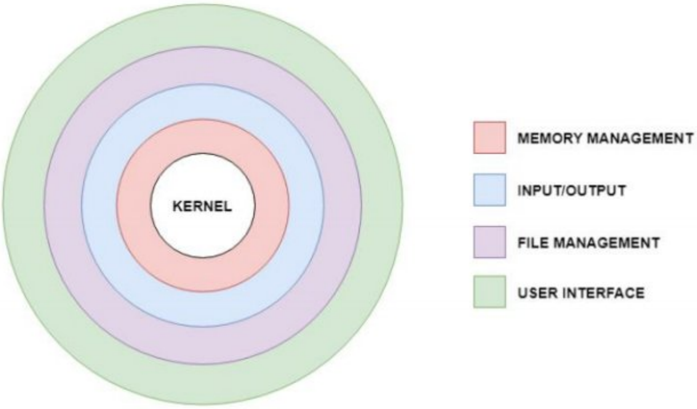
2.Mechanisms and Policies

- There is no specific way to design an operating system as it is a highly creative task.
- However, there are general software principles that are applicable to all operating systems.
- A subtle difference between mechanism and policy is that mechanism shows how to do something and policy shows what to do.
- Policies may change over time and this would lead to changes in mechanism.
- So, it is better to have a general mechanism that would require few changes even when a policy change occurs.
- For example - If the mechanism and policy are independent, then few changes are required in mechanism if policy changes.
- If a policy favors I/O intensive processes over CPU intensive processes, then a policy change to preference of CPU intensive processes will not change the mechanism.

3. Implementation

- At first, operating systems were written in assembly, but now C/C++ is the language commonly used
- Small blocks of assembly code are still needed, especially related to some low level I/O functions in device drivers, turning interrupts on and off and the **Test and Set Instruction for Synchronization Facilities**.
- Using higher level languages allows the code to be written faster. It also makes the OS much easier to port to different hardware platforms.
- An operating system is a construct that allows the user application programs to interact with the system hardware.
- Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.
- There are many problems that can occur while designing and implementing an operating system.
- These are covered in operating system design and implementation.

Result This program has been successfully executed.

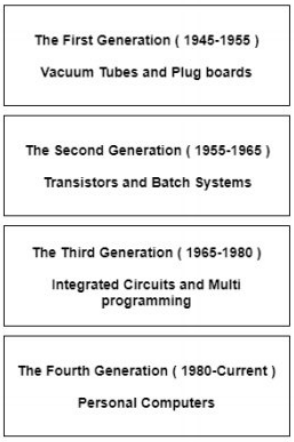


Layered Operating System Design

- The operating system needs to be implemented after it is designed.
- Earlier they were written in assembly language but now higher level languages are used.
- The first system not written in assembly language was the Master Control Program (MCP) for Burroughs Computers.
- **Advantages of Higher Level Language**
- There are multiple advantages to implementing an operating system using a higher level language such as: the code is written more fast, it is compact and also easier to debug and understand.
- Also, the operating system can be easily moved from one hardware to another if it is written in a high level language.
- **Disadvantages of Higher Level Language**
- Using high level language for implementing an operating system leads to a loss in speed and increase in storage requirements.
- However in modern systems only a small amount of code is needed for high performance, such as the CPU scheduler and memory manager.
- Also, the bottleneck routines in the system can be replaced by assembly language equivalents if required.

8. System Generation:

Operating Systems have evolved over the years. So, their evolution through the years can be mapped using generations of operating systems. There are four generations of operating systems. These can be described as follows −



The First Generation (1945-1955)
Vacuum Tubes and Plug boards
The Second Generation (1955-1965)
Transistors and Batch Systems
The Third Generation (1965-1980)
Integrated Circuits and Multi programming
The Fourth Generation (1980-Current)
Personal Computers

OPERATING SYSTEM GENERATIONS

The First Generation (1945 - 1955): Vacuum Tubes and Plugboards

Digital computers were not constructed until the second world war. Calculating engines with mechanical relays were built at that time. However, the mechanical relays were very slow and were later replaced with vacuum tubes. These machines were enormous but were still very slow.

These early computers were designed, built and maintained by a single group of people. Programming languages were unknown and there were no operating systems so all the programming was done in machine language. All the problems were simple numerical calculations.

By the 1950's punch cards were introduced and this improved the computer system. Instead of using plugboards, programs were written on cards and read into the system.

The Second Generation (1955 - 1965): Transistors and Batch Systems

Transistors led to the development of the computer systems that could be manufactured and sold to paying customers. These machines were known as mainframes and were locked in air-conditioned computer rooms with staff to operate them.

The Batch System was introduced to reduce the wasted time in the computer. A tray full of jobs was collected in the input room and read into the magnetic tape. After that, the tape was rewound and mounted on a tape drive. Then the batch operating system was loaded in which read the first job from the tape and ran it. The output was written on the second tape. After the whole batch was done, the input and output tapes were removed and the output tape was printed.

The Third Generation (1965 - 1980): Integrated Circuits and Multiprogramming

Until the 1960's, there were two types of computer systems i.e the scientific and the commercial computers. These were combined by IBM in the System/360. This used integrated circuits and provided a major price and performance advantage over the second generation systems.

The third generation operating systems also introduced multiprogramming. This meant that the processor was not idle while a job was completing its I/O operation. Another job was scheduled on the processor so that its time would not be wasted.

The Fourth Generation (1980 - Present): Personal Computers

Personal Computers were easy to create with the development of large-scale integrated circuits. These were chips containing thousands of transistors on a square centimeter of silicon. Because of these, microcomputers were much cheaper than minicomputers and that made it possible for a single individual to own one of them.

The advent of personal computers also led to the growth of networks. This created network operating systems and distributed operating systems. The users were aware of a network while using a network operating system and could log in to remote machines and copy files from one machine to another.

Chapter-3

PROCESSES

Topics:

- **Process Concepts**
- **Process Scheduling**
- **Operation on Processes**
- **Inter process Communication**
- **Communication in client server System**

❖ Introduction to Processes:

- Early computer systems allowed only one program to be executed at a time.
- Current-day computer systems allow multiple programs to be loaded into memory and executed concurrently.
- A process is the unit of work in a modern time-sharing system.
- A system therefore consists of a collection of processes: **operating system processes executing system code** and **user processes executing user code.**

Differences between Process and Program:

Process	Program
The process is basically an instance of the computer program that is being executed.	A Program is basically a collection of instructions that mainly performs a specific task when executed by the computer.
A process has a **shorter lifetime**.	A Program has a **longer lifetime**.
A Process requires resources such as memory,	A Program is stored by hard-disk and

Process	Program
CPU, Input-Output devices.	does not require any resources.
A process has a dynamic instance of code and data	A Program has static code and static data.
Basically, a process is the **running instance** of the code.	On the other hand, the program is the **executable code**.

❖ Process Concepts:

- A batch system executes jobs, whereas a time-shared system has user programs, or tasks.
- Even on a single-user system such as Microsoft Windows, a user may be able to **run several programs** at one time: **a word processor, a web browser, and an e-mail package.**
- Even if the user can execute only one program at a time, the operating system may need to support its own internal programmed activities, such as memory management.
- In many respects, all these activities are similar, so we call all of them **processes.**

1. **The process**
2. **Process state**
3. **Process Control Block**
4. **Thread**

1.The Process:

- A process is a program in execution. A process is more than the program code, which is sometimes known as the **text section.**

- It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.
- A process generally includes the process **stack**, which contains temporary data (such as method parameters return addresses, and local variables), and a **data section**, which contains global variables.

2.Process State:

- As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

➢ Each process may be in one of the following states:

❖ **New**: The process is being created.
❖ **Running**: Instructions are being executed.
❖ **Waiting**: The process is waiting for some event to occur.
❖ **Ready**: The process is waiting to be assigned to a processor.
❖ **Terminated**: The process has finished execution.

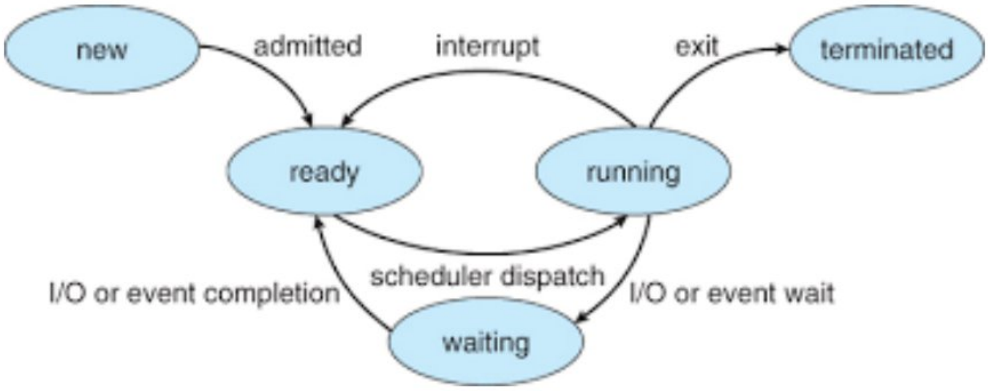


Figure: Diagram of process state.

- These state names are arbitrary, and they vary across operating system.
- Only one process can be running on any processor at any instant, although many processes may be **ready and waiting**. The state diagram corresponding to these is presented in above figure.

3.Process Control Block

- Each process is represented in the operating system by a **Process Control Block(PCB) –** also called a task control block. A PCB is shown in figure.
- It contains many pieces of information associated with a specific process, including these:
- **Process state:** The state may be new, ready, running, waiting, terminated and so on.

- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
- **CPU register:** The register vary in number and type, depending on the computer architecture**.** They include **accumulators, index registers, stack pointers,** and **general-purpose registers,** plus any condition-code information**.**
- **CPU- Scheduling Information: This** information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory–Management Information:** This information may include such information as the value of the base and limit registers, the page tables or the segment tables, depending on the memory system used by the operating system.

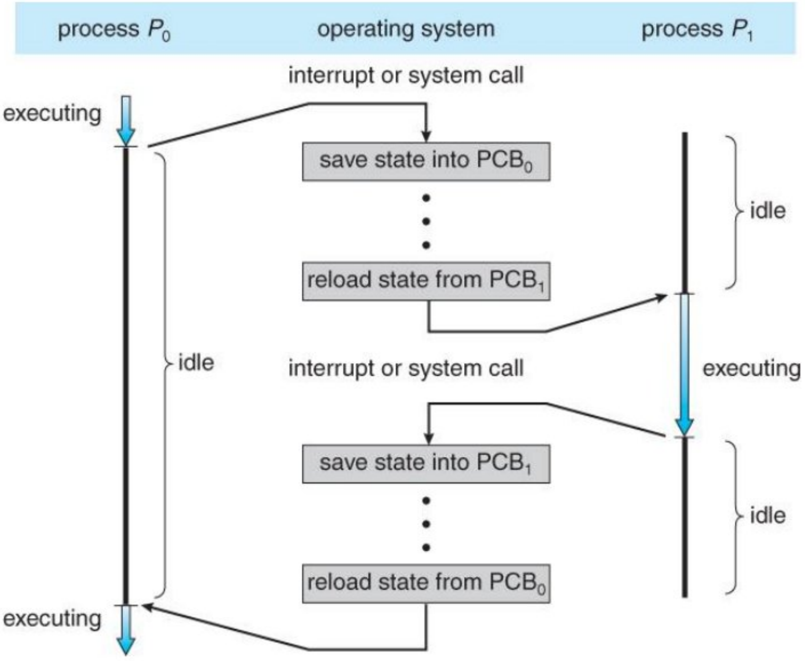


Diagram showing CPU switch from process to process

- **Accounting information:** This information includes the amount of **CPU and real time used, time limits, account numbers, job or process numbers,** and so on.
- **I/O status information:** The information include the list of I/O devices allocated to this process, a list of open files, and so on.

Threads:

- A **thread** is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history. A **thread** is also called a lightweight process.

❖ Process Scheduling:

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues:

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue consists of all the processes in the system.
- **Ready queue** – The processes that are residing in main memory, ready and waiting to execute are kept on a list called **ready queue**. A new process is always put in this queue.
- **Device queues** –The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O **device queue.**

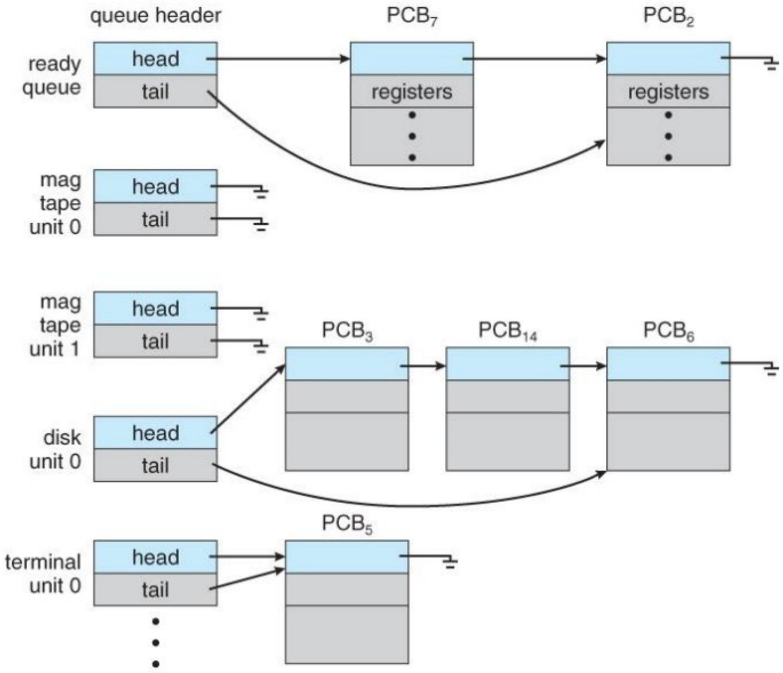
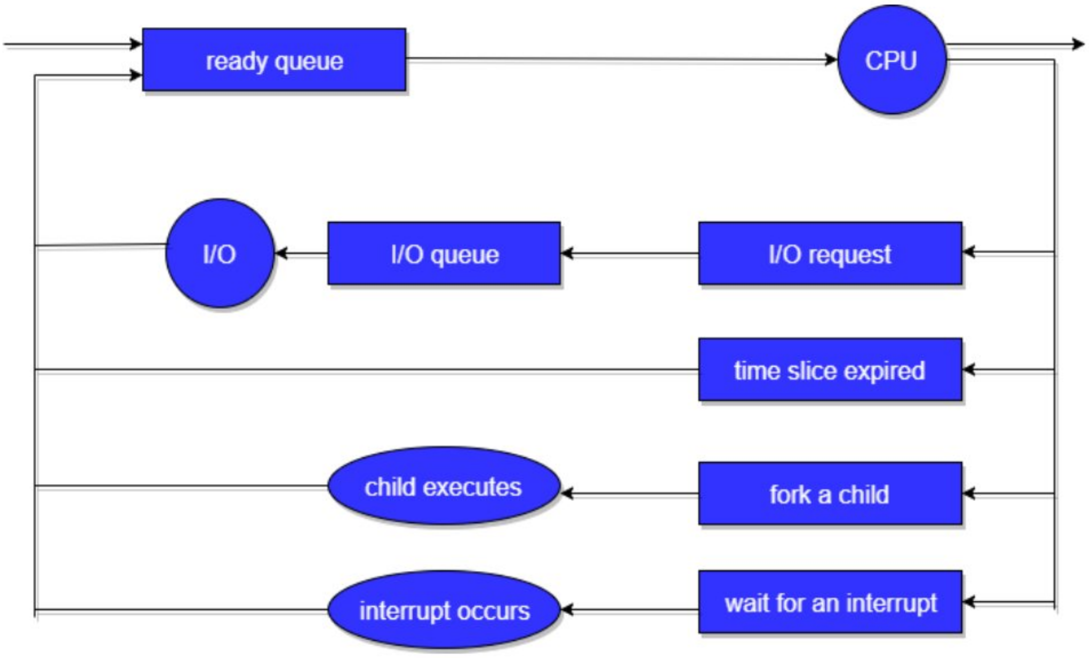


Figure: The ready queue and various I/O device queue

A common representation of process scheduling is a queueing diagram, such as that in figure. Each rectangular box represents a queue.



Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicates the flow of processes in the system

A new process is initially put in the ready queue. It waits in the ready queue until it is selected for **execution or dispatched.** Once the process is assigned to the CPU and is executing, one of several events could occur.

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new process and wait for it termination.
- The process could be removed from the CPU, as a result of an interrupt, and be back in the ready queue.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

- It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.
- The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
- On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

- It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

- Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

- Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.
- A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion.
- In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage.
- This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

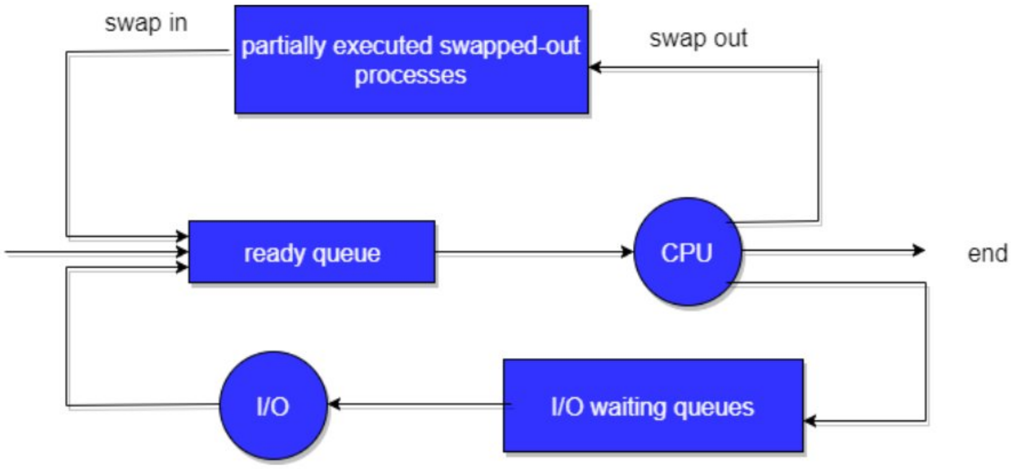


Figure: Addition of medium term scheduling to the queueing diagram

Comparison among Scheduler

S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping

			scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch

1. Switching the CPU to another process requires **saving** the state of the old process and **loading** the saved state for the new process. This task is known as a **Context Switch**.
2. The **context** of a process is represented in the **Process Control Block(PCB)** of a process; it includes the value of the CPU registers, the process state and memory-management information. When a context switch occurs, the Kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
3. Context switch time is **pure overhead**, because the **system does no useful work while switching**. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions(such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.
4. Context Switching has become such a performance **bottleneck** that programmers are using new structures(threads) to avoid it whenever and wherever possible.

❖ Operations on Processes:

The processes in the system can execute concurrently, and they must be created and deleted dynamically. Thus, the operating system must provide a mechanism for **process creation and termination**.

Process Creation:

- A process may create several new processes, via a create-process system call, during the course of execution.
- The creating process is called a parent process, whereas the new processes are called the children of that process.
- Each of these new processes may in turn create other processes, forming a tree of processes in **figure.**

When the process creates a new process, two possibilities exist in terms of execution.

- The parent continues to execute concurrently with its children.
- The parent waits until some or all of its children have terminated.

There are also two possibilities in terms of the address space of the new process.

- The child process is a duplicate of the parent process.
- The child process has a program loaded into it.

Process Termination:

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

- The child has exceeded its usage of some of the resources that it has been allocated. This requires the parent to have a mechanism to inspect the state of its children.
- The task assigned to the child is no longer required.
- The parent is exiting ,and the operating system does not allow a child to continue if its parent terminates.

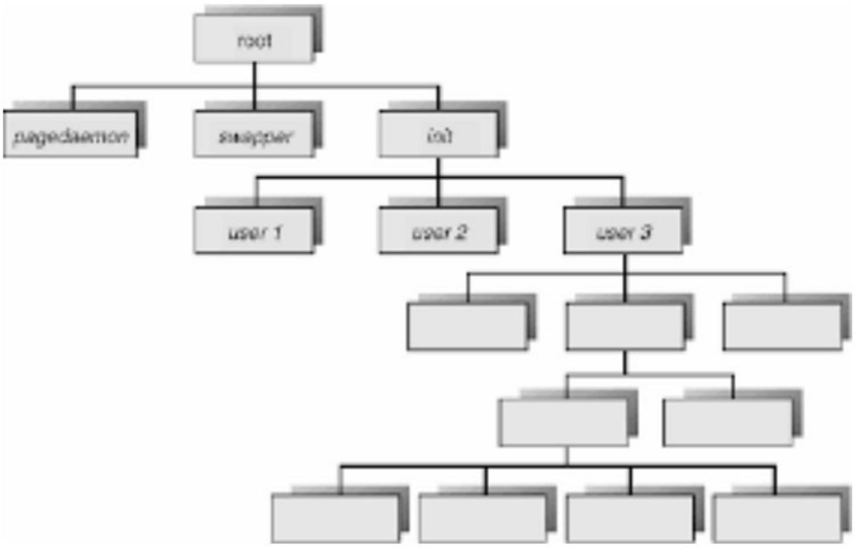
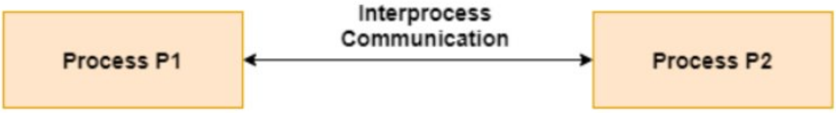


Figure: A tree of processes on a typical UNIX system

❖ INTERPROCESS COMMUNICATION:

Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other. This communication could involve a process letting another process know that some event has occurred or the **transferring of data from one process to another.**



A diagram that illustrates interprocess communication is as following Interprocess communication is the mechanism provided by the operating system that allows processes to communicate with each other.

Message Passing System:

In this article, you will learn another popular method of interprocess communication called message passing. You will learn different techniques and characteristics of message passing.

The cooperating processes can communicate with the help of a message passing facility. There are different ways communication happens:

- Between same threads of a process.

- Between processes on same node or computer.
- Between two processes on different nodes or computers.
  Example: chat system over Internet

A message-passing system has atleast two primitive operations:

1. send(message)
2. receive(message)

The messages could be of a fixed size or a variable size. If system uses fixed size messages, then system level implementation is easy. If chosen variable size messages, the system level implementation is difficult, but the programming level task is easy.

If P and Q process wants to communicate, then a communication link must exist between them. There are several ways to implement physical link(shared memory, hardware, or network), but we are interested in logical implementation of a link. Here are different ways to communicate using message passing:

1. Direct or Indirect communication
2. Synchronous or asynchronous communication
3. Automatic or explicit buffering.

Naming: Processes that want to communicate must have a way to refer to each other. They can use either **direct or indirect communication.**

Direct Communication:

Under direct communication, processes must explicitly name the sender or receiver in the communication.

The send() and receive() are defined as:

send(P, message) – send a message to process P
receive(Q, message) – receive a message from process Q

The communication link in direct communication has the following properties:

- The link is established automatically. Processes need each other's identity to send message.
- A link is associated with exactly two processes.
- Between two processes, there is only one link.

There is a symmetric in addressing. In asymmetrical addressing, the sending process needs to address the receiver, but the recipient does not need to address the sender.

Direct Messaging:

send(p, message)- send message to process P

received(id, message) – receive message from any process. Id is replaced with name of the sender process.

- The disadvantages in symmetric and asymmetric schemes is in changing the identifier of a process.
- We need to change all the other process definition and references to the old identifier and replace with the new one.

Indirect Communication:

- In indirect communication, message are sent and received from **mailboxes or ports.** The processes can place messages into a mailbox or remove messages from them. The mailbox has a unique identification.

Two process can communicate only if they have a shared mailbox.

send(A, message) – send a message to mailbox A

receive (A, message) – receive a message from mailbox A

In this scheme, a communication link has the following properties:

1. A link is established between a pair of processes only if both have a same shared mailbox.
2. A link may be associated with more than two processes.
3. There may be different links, with each link corresponding to one mailbox, between pair of communicating processes.

Example: P1, P2 and P3 all share mailbox A. P1 sends message to A, while P2 and P3 execute a receive() from A. Which process will receive the message? The answer depends on one of the chosen method given below.

- Allow link to associated with at – most 2 processes.
- Allow at most 1 process to execute receive() operation.
- Choose randomly at- most one process to receive message or choose an algorithm to receive message such as round – robin(each process take turn to receive message).

The mailbox may be owned either by a process or by the OS.

Indirect Messaging:

If process owns (mailbox part of process address space) then:

Distinguish between owner(can only receive through own mailbox).
The user (can only send message to the mailbox).
When process with mailbox terminates, mailbox disappears, and all other processes should be notified.

If OS owns the mailbox, then

- It is independence process, not attached to a process.
- Then the OS must allows the process to:
  1. Create a mailbox
  2. Send and receive messages through mailbox.

3. Delete the mailbox.

- Process becomes the owner of new mailbox by default.
- Owner process can only receive messages through this mailbox.
- Ownership and receiving privileges can be passed using system calls to other processes. This will result in multiple receivers for each mailbox.

Synchronous and Asynchronous Communication:

Communication happens using send() and receive(). There are many options for these two primitives. Message passing may be blocking or non blocking also known as synchronous and asynchronous.

- **Blocking send-** sending is blocked, until a message is received by receiving process or mailbox.
- **Non-blocking send** – sending process sends the message and resumes operation.
- **Blocking receive** – receiver blocks until a message is available.
- **Non –blocking receive** – the receiver retrieves either a valid message or a null.

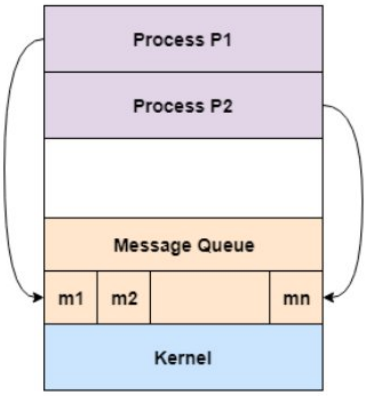
Automatic and Explicit Buffering

The messages exchanged between communicating processes resides in a temporary queue. The queue can be implemented in three ways:

1. Zero capacity
2. Bounded capacity
3. Unbounded capacity

- **Zero capacity –** with zero capacity the link cannot have messages waiting. Blocking send is the correct option.
- **Bounded capacity –** the queue is of finite length n. If queue not full, sender can continue to send messages. If full, blocking send.
- **Unbounded capacity –** Infinite queue length; any number of messages wait and sender never block

A diagram that demonstrates message passing model of process communication is given as follows −



Message Passing Model

In the above diagram, both the processes P1 and P2 can access the message queue and store and retrieve data.

Advantages of Message Passing Model

Some of the advantages of message passing model are given as follows −

- The message passing model is much easier to implement than the shared memory model.
- It is easier to build parallel hardware using message passing model as it is quite tolerant of higher communication latencies.

Disadvantage of Message Passing Model

The message passing model has slower communication than the shared memory model because the connection setup takes time.

❖ Communication in client server system:

Sockets:

A **socket** is one endpoint of a two way communication link between two programs running on the network. The **socket** mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.

- A socket is made up of an IP address concatenated with a port number. In general, socket use a client-server architecture.
- The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection.

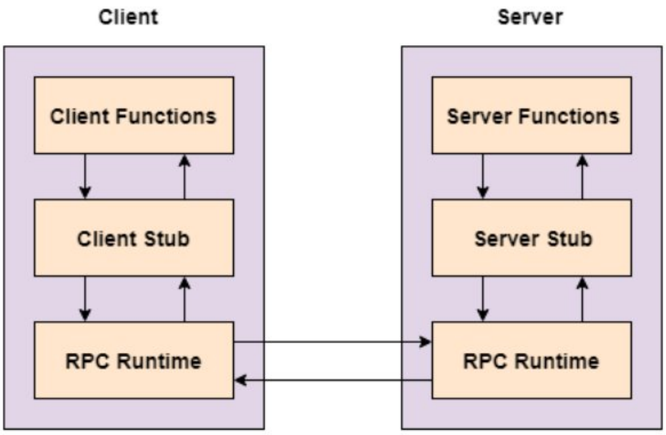
REMOTE PROCEDURE CALL (RPC):

- A remote procedure call is an interprocess communication technique that are used for client-server based applications. It is also known as a subroutine call or a function call.
- A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server.
- When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows −

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

A diagram that demonstrates this is as follows −



Advantages of Remote Procedure Call

Some of the advantages of RPC are as follows −

- Remote procedure calls support process oriented and thread oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimum in remote procedure calls.
- Remote procedure calls can be used in distributed environment as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of Remote Procedure Call

Some of the disadvantages of RPC are as follows

- The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
- There is no flexibility in RPC for hardware architecture. It is only interaction based.
- There is an increase in costs because of remote procedure call.
- ➢ A diagram that illustrates remote procedure calls is given as follows:

Remote Method Invocation(RMI)

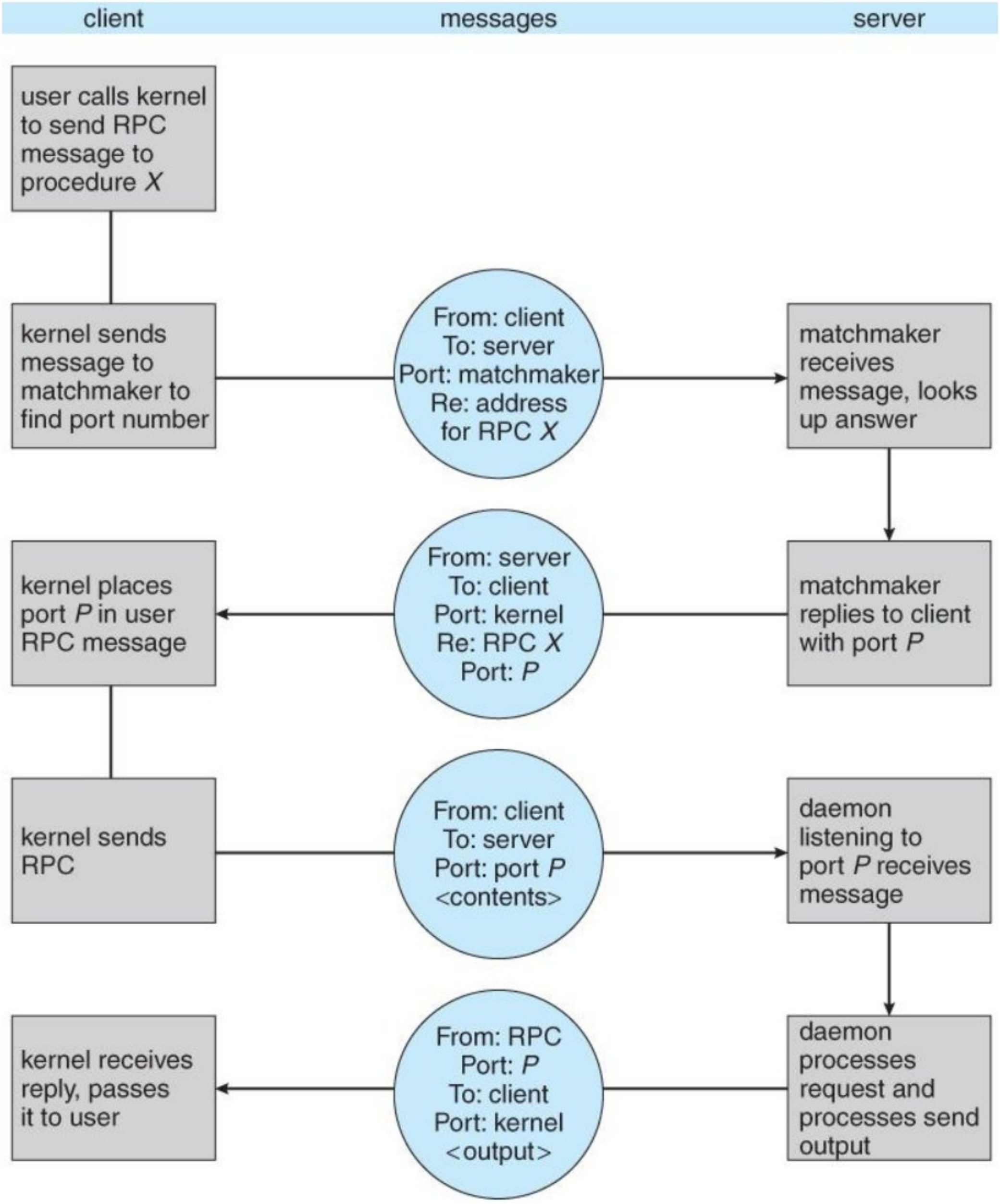
RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.


client
messages
server
user calls kernel to send RPC message to procedure *X*
kernel sends message to matchmaker to find port number
From: client To: server Port: matchmaker Re: address for RPC *X*
matchmaker receives message, looks up answer
matchmaker replies to client with port *P*
From: server To: client Port: kernel Re: RPC *X* Port: *P*
kernel places port *P* in user RPC message
kernel sends RPC
From: client To: server Port: port *P* <contents>
daemon listening to port *P* receives message
daemon processes request and processes send output
From: RPC Port: *P* To: client Port: kernel <output>
kernel receives reply, passes it to user

Let us now discuss the components of this architecture.

- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.

- **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub

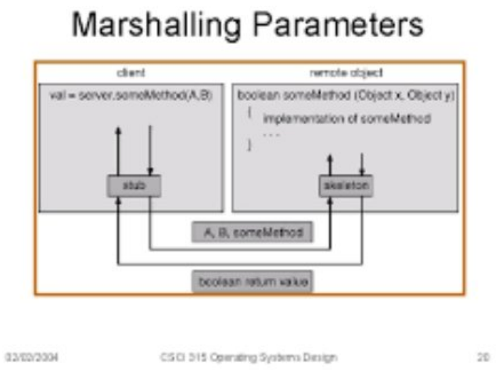
The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

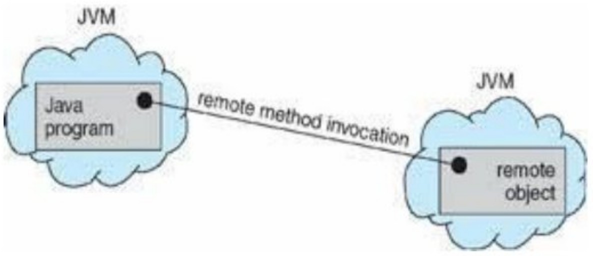
1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.



Marshalling Parameters

client
remote object
val = server.someMethod(A,B)
boolean someMethod (Object x, Object y)
{ implementation of someMethod
. . .
}
stub
skeleton
A, B, someMethod
boolean return value

02/02/2004 CSCI 315 Operating Systems Design 20



Marshalling and Unmarshalling

Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

Goals of RMI

Following are the goals of RMI −

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects

Chapter-4
THREADS

TOPICS:

- **Thread**
- **Multithreading Models**
- **Threading Issues**
- **Pthread**

THREADS:

- The implementation of **threads** and processes differs between **operating systems**, but in most cases a **thread** is a component of a process.
- Multiple **threads** can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources.
- A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.
- A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.
- A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism.
- Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.
- Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control.
- Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

The following figure shows the working of a single-threaded and a multithreaded process.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.

- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

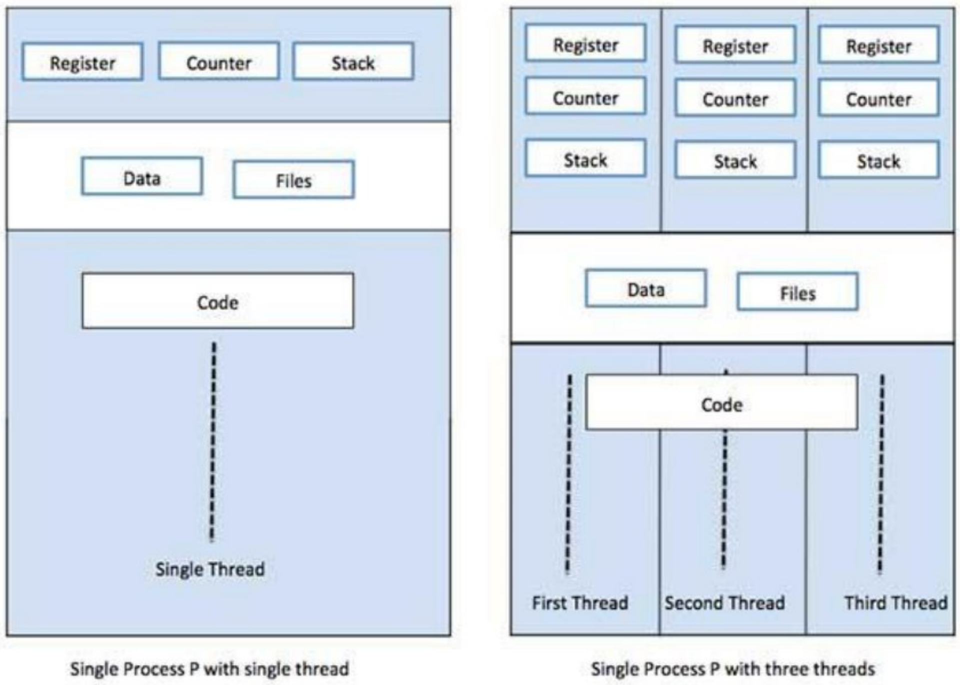


Figure: Single and Multithreaded processes

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.

4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Benefits:

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower overhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectures

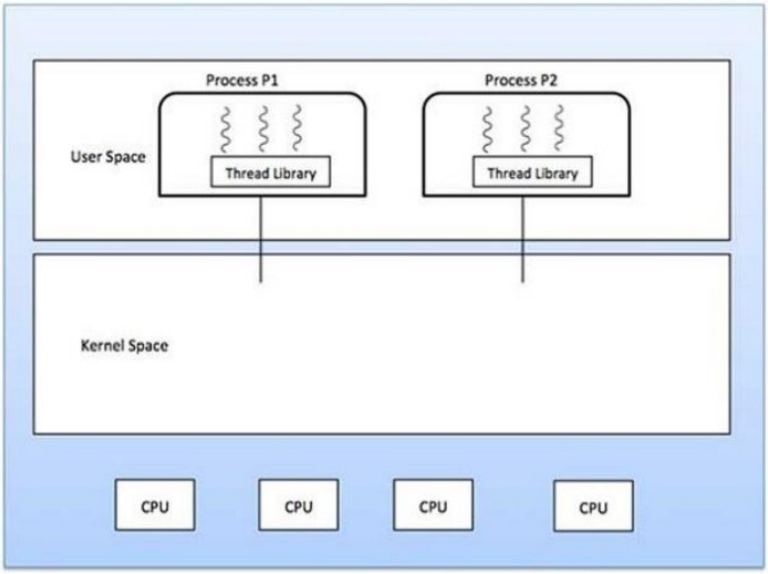
Types of Thread

Threads are implemented in following two ways −

- **User Level Threads** − User managed threads.
- **Kernel Level Threads** − Operating System managed threads acting on kernel, an operating system core.

User Level Threads

- In this case, the thread management kernel is not aware of the existence of threads.
- The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts.
- The application starts with a single thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads

- In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system.
- Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis.
- The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

❖ Multithreading Models

- ❖ Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach.
- ❖ In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process.
- ❖ Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

Many to Many Model

- ❖ The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.
- ❖ The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads.
- ❖ In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine.
- ❖ This model provides the best accuracy on concurrency and when a thread performs a blocking system call, the kernel can schedule another thread for execution.

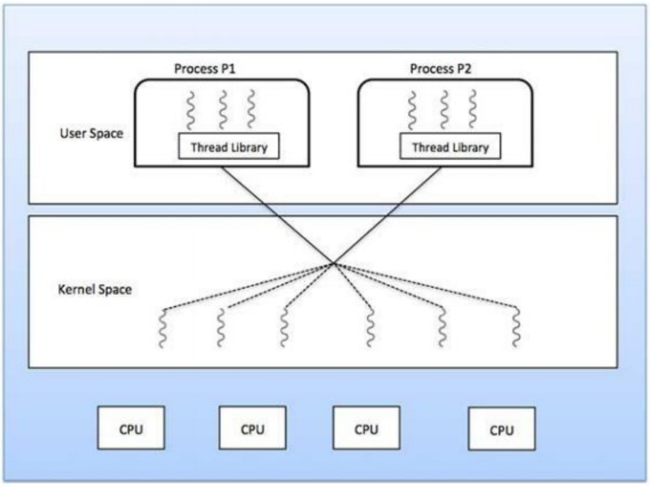
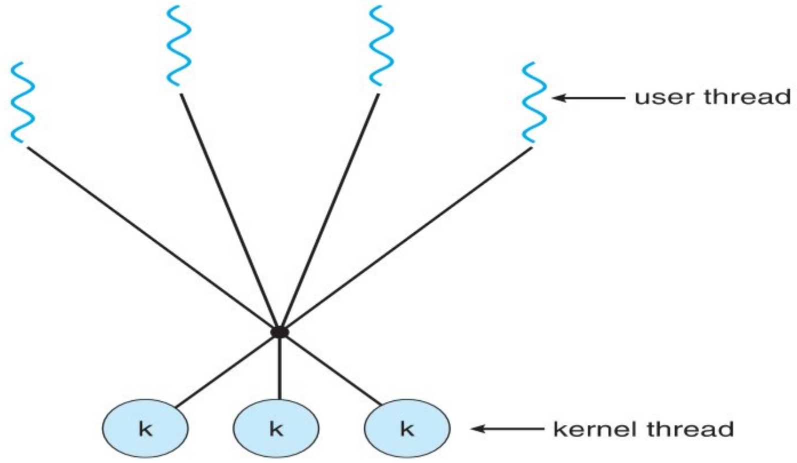


Figure: Many to Many model



Many to One Model

- Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library.
- When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.
- If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.

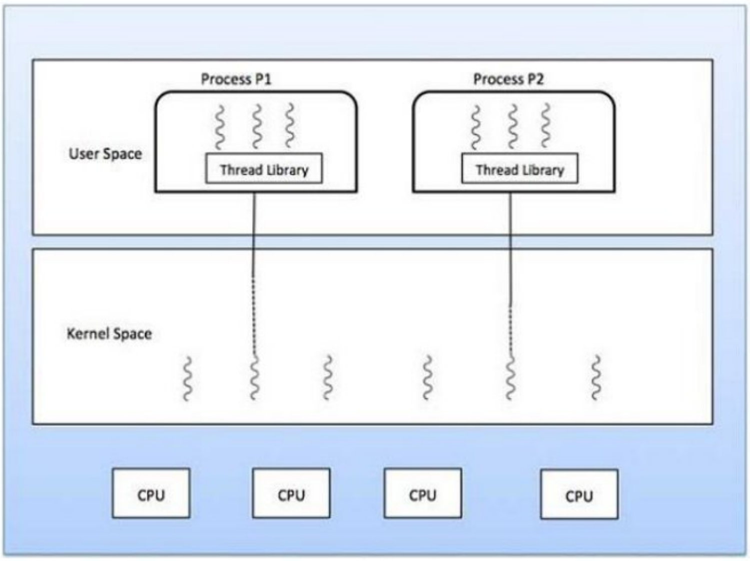
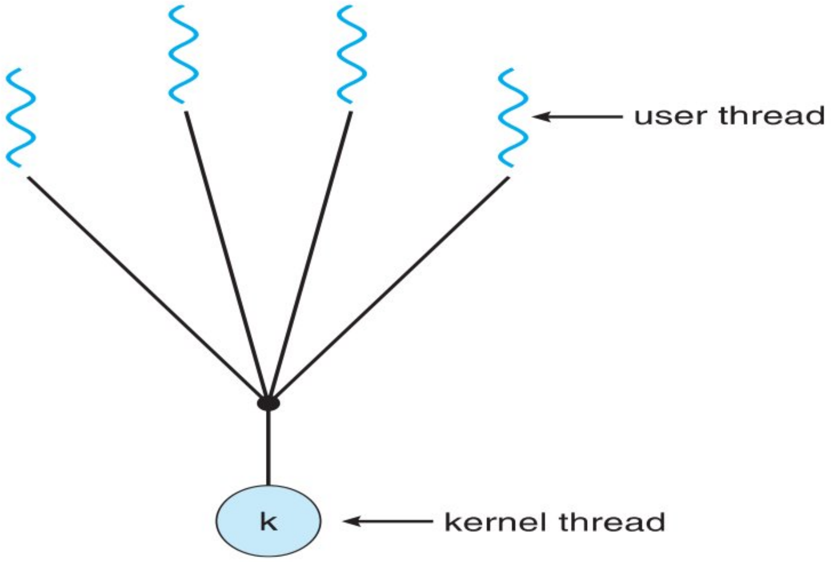


Figure: Many to One model



One to One Model

- ❖ There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model.
- ❖ It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.
- ❖ Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

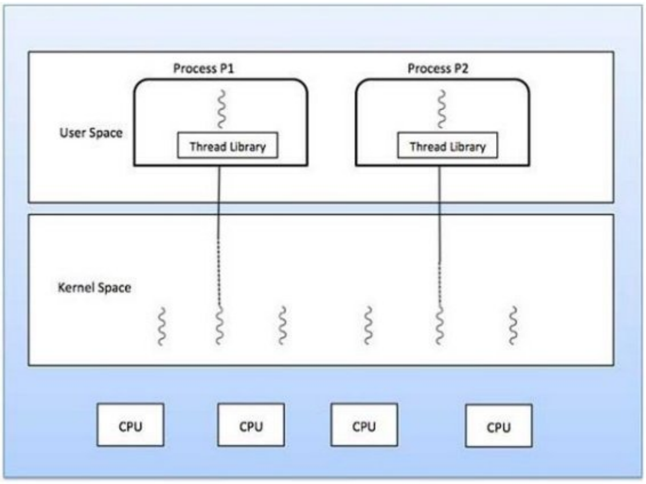
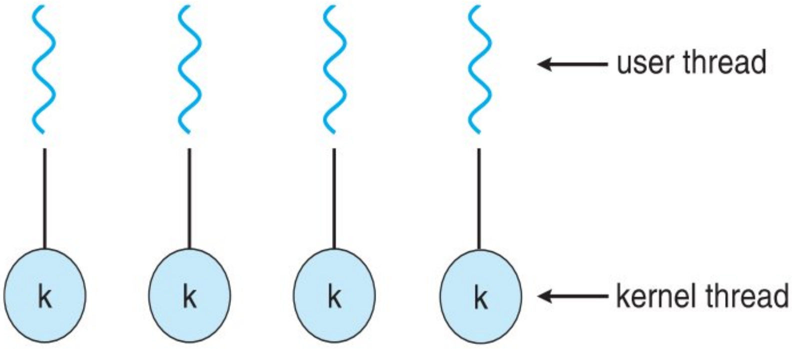


Figure: One to One model



❖ Threading Issues:

- ❖ **The fork and exec system call**
- ❖ **Cancellation**
- ❖ **Signal Handling**
- ❖ **Thread pools**
- ❖ **Thread-specific data**

The fork and exec system call:

- The fork system call is used to create a separate, duplicate process.
- In a multithreaded program, the semantics of the fork and exec system calls changes.
- Some UNIX system have chosen to have two versions of fork, **one that duplicates all threads** and **another that duplicates only the thread that invoked the fork system call**.
- The exec system call typically works in the same way.
- The two versions of **fork** depends upon the application. If **exec** is called immediately after forking, then duplicating all threads is unnecessary, as the program specified in the parameters to exec will replace the process.
- The separate process does not call exec after forking, the separate process should duplicate all threads.

Cancellations:

- **Thread cancellation** is the task of terminating a thread before it has completed.
- **For example**: if multiple threads are concurrently searching through a database and one thread return thr result, the remaining threads might be cancelled.
- **A thread** that is to be cancelled is often reffered to as the **target thread.**
- **` Two general approaches:**

- **Asynchronous cancellation -** terminates the target thread immediately
- **Deferred cancellation-** allows the target thread to periodically check if it should be cancelled.

Signal Handling

- Signals are used in UNIX systems to notify a process that a particular event has occurred.
- A signal handler is used to process signals
  - Signal is generated by particular event
  - Signal is delivered to a process
  - Signal is handled by one of **two signal handlers**:
    - **Default signal handler**
    - **user-defined signal handler**

Every signal has default handler that kernel runs when handling signal

1. User-defined signal handler can override default
2. For single-threaded, signal delivered to process

Where should a signal be delivered for multi-threaded?

- Deliver the signal to the thread to which the signal applies
- Deliver the signal to every thread in the process
- Deliver the signal to certain threads in the process
- Assign a specific thread to receive all signals for the process.

Thread pools:

- Create a number of threads in a pool where they await work

Advantages:

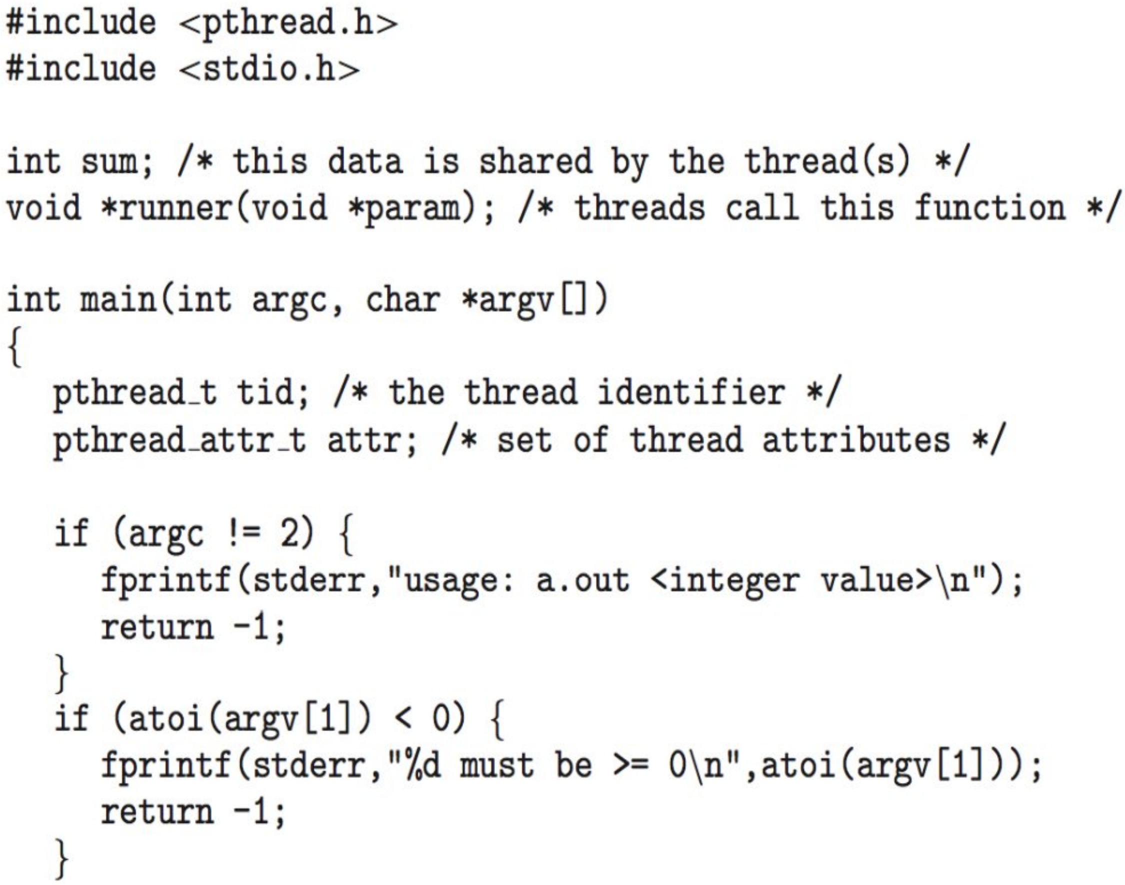
- Usually slightly faster to service a request with an existing thread than create a new thread
- Allows the number of threads in the application(s) to be bound to the size of the pool
- Separating task to be performed from mechanics of creating task allows different strategies for running task
- ➢ i.e.Tasks could be scheduled to run periodically

Thread-Specific Data:

- Threads belonging to a process share the data of the process.
- Each thread might need its own copy of certain data in some circumstances. We will call such data thread-specific data.
  For Example: in a transaction-processing system, we might service each transaction in a separate thread.

❖ Pthreads:

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- *Specification*, not *implementation*
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

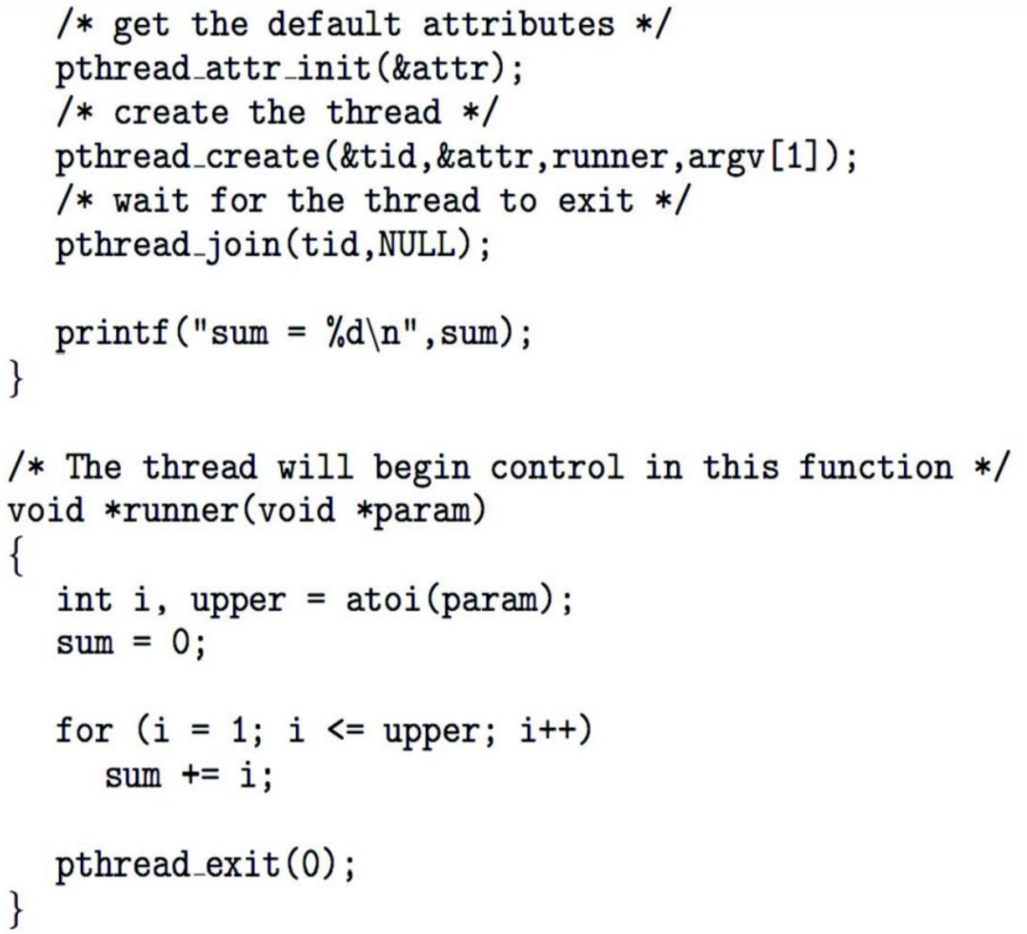


```
#include <pthread.h>
#include <stdio.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid; /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  if (argc != 2) {
    fprintf(stderr,"usage: a.out <integer value>\n");
    return -1;
  }
  if (atoi(argv[1]) < 0) {
    fprintf(stderr,"%d must be >= 0\n",atoi(argv[1]));
    return -1;
  }
```



```
  /* get the default attributes */
  pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid,&attr,runner,argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}

/* The thread will begin control in this function */
void *runner(void *param)
{
  int i, upper = atoi(param);
  sum = 0;

  for (i = 1; i <= upper; i++)
    sum += i;

  pthread_exit(0);
}
```