Processes and threads

Introduction to operating systems – review of computer organization – operating system structures – system calls – system programs – system structure – virtual machines. Processes: Process concept – Process scheduling – Operations on processes – Cooperating processes – Inter-process communication – Communication in client-server systems. Case study: IPC in Linux. Threads: Multi-threading models – Threading issues. Case Study: Pthreads library

Introduction :

What is an Operating System?

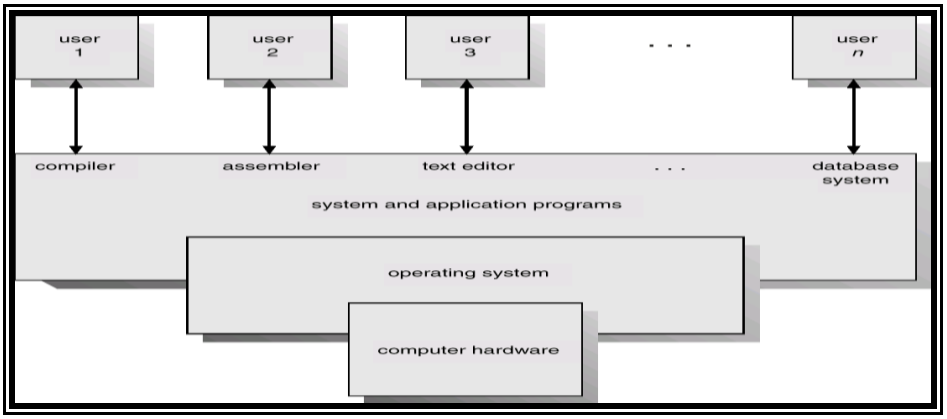
- An operating system is a program that manages the computer hardware.
- It also provides a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware.
- The purpose of an operating system is to provide an environment in which a user can execute programs.

Goals of an Operating System

- The primary goal of an operating system is thus to make the computer system convenient to use.
- The secondary goal is to use the computer hardware in an efficient manner.

Components of a Computer System

- An operating system is an important part of almost every computer system.
- A computer system can be divided roughly into four components.
    i. Hardware
    ii. Operating system
    iii. The application programs
    iv. Users



- The hardware - the central processing unit **(CPU),** the memory, and the Input/output **(I/O)** devices-provides the basic computing resources.
- 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.
- An operating system is similar to a *government. The* OS simply provides an environment within which other programs can do useful work.

Abstract view of the components of a computer system.

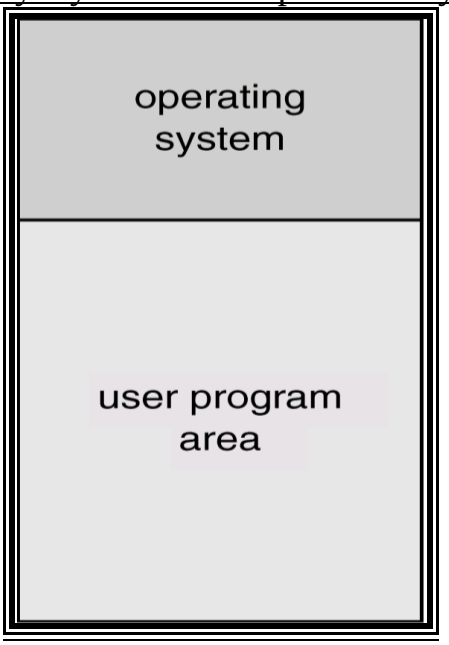
- Operating system can be viewed as a resource allocator.
- The OS acts as the manager of the resources (such as CPU time, memory space, file storage space, I/O devices) and allocates them to specific programs and users as necessary for tasks.
- An operating system is a control program. It controls the execution of user programs to prevent errors and improper use of computer.

Mainframe Systems

- Early computers were physically enormous machines run from a console.
- The common input devices were card readers and tape drives.
- The common output devices were line printers, tape drives, and card punches.
- The user did not interact directly with the computer systems.
- Rather, the user prepared a job - which consisted of the program, the data, and some control information about the nature of the job (control cards)-and submitted it to the computer operator.
- The job was usually in the form of punch cards.
- The operating system in these early computers was fairly simple.

- Its major task was to transfer control automatically from one job to the next.
- The operating system was always resident in memory

Memory layout for a simple batch system.



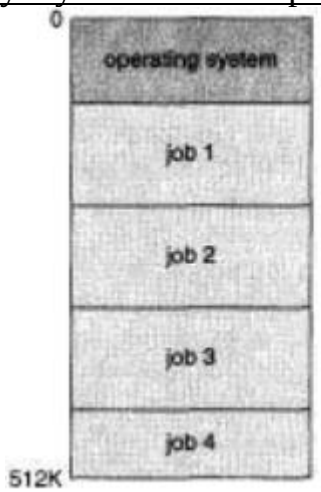
- A batch operating system, thus normally reads a stream of separate jobs.
- When the job is complete its output is usually printed on a line printer.
- The definitive feature of batch system is the lack of interaction between the user and the job while the job is executing.
- Spooling is also used for processing data at remote sites.

Multiprogrammed Systems

- A pool of jobs on disk allows the OS to select which job to run next, to increase CPU utilization.
- Multiprogramming increases CPU utilization by organizing jobs such that the CPU always has one to execute.
- The idea is as follows: The operating system keeps several jobs in memory simultaneously. This set of jobs is a subset of the jobs kept in the job pool.

The operating system picks and begins to execute one of the jobs in the memory.

Memory layout for a multiprogramming system.



Time-Sharing Systems

- Time sharing (or multitasking) is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.
- A time-shared operating system allows many users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to her use, even though it is being shared among many users.
-

Desktop Systems

- As hardware costs have decreased, it has once again become feasible to have a computer system dedicated to a single user. These types of computer systems are usually referred to as personal computers(PCS). They are

microcomputers that are smaller and less expensive than mainframe computers.

- Operating systems for these computers have benefited from the development of operating systems for mainframes in several ways.

Multiprocessor Systems

- **Multiprocessor systems** (also known as **parallel systems** or **tightly coupled systems)** have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

- Multiprocessor systems have three main advantages.
  - **Increased throughput.**
  - **Economy of scale.**
  - **Increased reliablility.**
- **If** functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors must pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether. This ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation.** Systems designed for graceful degradation are also called **fault tolerant.**

- Continued operation in the presence of failures requires a mechanism to allow the failure to be detected, diagnosed, and, if possible, corrected.
- The most common multiple-processor systems now use **symmetric multiprocessing** (SMP), in whch each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.
- Some systems use **asymmetric multiprocessing,** in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

Distributed Systems

- In contrast to the tightly coupled systems, the processors 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.

Advantages of distributed systems

- Resource Sharing
- Computation speedup
- Reliability
- Communication

Real-Time Systems

- Systems that control scientific experiments, medical imaging systems, industrial control systems, and certain display systems are real-time systems. Some automobile-engine fuel-injection systems, home-appliance controllers, and weapon systems are also real-time systems. A real-time system has well-defined, fixed time constraints.
- Real-time systems come in two flavors: hard and soft.
- A hard real-time system guarantees that critical tasks be completed on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Such time constraints dictate the facilities that are available in hard real-time systems.
- A less restrictive type of real-time system is a soft real-time system, where a critical real-time task gets priority over other tasks, and retains that priority until it completes.
- Soft real-time systems, however, have more limited utility than hard real-time systems. They are useful, in several areas, including multimedia, virtual reality, and advanced scientific projects.

Operating System Components

There are eight major operating system components.They are :

- Process management

- Main-memory management
- File management
- I/O-system management
- Secondary-storage management
- Networking
- Protection system
- Command-interpreter system

(i) Process Management

- A **process** can be thought of as a program in execution. A batch job is a process. A time shared user program is a process.
- A process needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task.
- A program by itself is not a process; a program is a *passive* entity, such as the contents of a file stored on disk, whereas a process is an *active* entity, with a **program counter** specifying the next instruction to execute.
- A process is the unit of work in a system.
- The operating system is responsible for the following activities in connection with process management:
    - Creating and deleting both user and system processes
    - Suspending and resuming processes
    - Providing mechanisms for process synchronization
    - Providing mechanisms for process communication
    - Providing mechanisms for deadlock handling

(ii) Main – Memory Management

- Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address.
- Main memory is a repository of quickly accessible data shared by the CPU and I/O devices.
- To improve both the utilization of the CPU and the speed of the computer's response to its users, we must keep several programs in memory.
- The operating system is responsible for the following activities in connection with memory management:
    - Keeping track of which parts of memory are currently being used and by whom.

- Deciding which processes are to be loaded into memory when memory space becomes available.
- Allocating and deallocating memory space as needed.

(iii) File Management

- File management is one of the most visible components of an operating system.
- The operating system is responsible for the following activities in connection with file management:
  - Creating and deleting files
  - Creating and deleting directories
  - Supporting primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - Backing up files on stable (nonvolatile) storage media

(iv) I/O System management

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. This is done using the I/O subsystem.
- The I/O subsystem consists of
  - A memory-management component that includes buffering, caching, and spooling
  - A general device-driver interface
  - Drivers for specific hardware devices

(v) Secondary storage management

- Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lost,the computer system must provide **secondary storage** to back up main memory.
- The operating system is responsible for the following activities in connection with disk management:
  - Free-space management
  - Storage allocation
  - Disk scheduling

(vi) Networking

- **A distributed system** is a collection of processors that do not share memory, peripheral devices, or a clock.
- Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks.
- The processors in the system are connected through a **communication network,** which can be configured in a number of different ways.

(vii) Protection System

- Various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.
- Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.

(viii) Command-Interpreter System

- One of the most important systems programs for an operating system is the command interpreter.
- It is the interface between the user and the operating system.
- Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).
- 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 on to a time-shared system, a program that reads and interprets control statements is executed automatically.
- This program is sometimes called the **control-card interpreter** or the **command-line interpreter**, and is often known as the **shell**.

Operating-System Services

The OS provides certain services to programs and to the users of those programs.

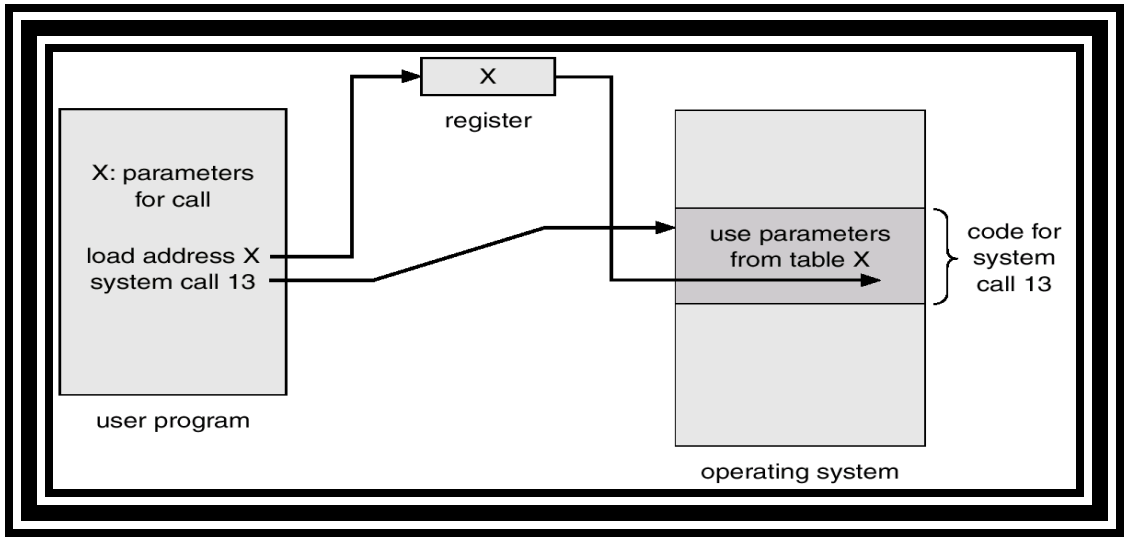
1. **Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
2. **I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device.
3. **File-system manipulation:** The program needs to read, write, create, delete files.
4. **Communications :** In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network.
5. **Error detection:** The operating system 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.
6. **Resource allocation:** Different types of resources are managed by the Os. When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
7. **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.
8. **Protection:** The owners of information stored in a multiuser computer system may want to control use of that information. Security of the system is also important.

Passing parameters to OS

Three general approaches are used to pass parameters to OS.

1. Pass parameters in registers.
2. In cases, where there may be more parameters than registers, the parameters are generally stored in a block or table in memory, and address of block is passed as parameter in register.
3. Parameters can also be placed, or *pushed,* onto the *stack* by the program, and *popped o f* the stack by the operating system.

Passing of parameters as a table.



System Calls

- System calls provide the interface between a process and the operating system.
- These calls are generally available as assembly-language instructions.
- System calls can be grouped roughly into five major categories:
    1. Process control
    2. file management
    3. device management
    4. information maintenance

5. communications.

1. **Process Control**
   - end,abort
   - load, execute
   - Create process and terminate process
   - get process attributes and set process attributes.
   - wait for time, wait event, signal event
   - Allocate and free memory.

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 attribtues, 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

Two types of communication models
(a) Message passing model
(b) Shared memory model

System Programs

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

1. **File management:** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
2. **Status information:** The status such as date, time, amount of available memory or diskspace, number of users or similar status information.
3. **File modification:** Several text editors may be available to create and modify the content of files stored on disk or tape.
4. **Programming-language support:** Compilers, assemblers, and interpreters for common programming languages are often provided to the user with the operating system.
5. **Program loading and execution:** The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders.
6. **Communications:** These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. (email, FTP, Remote log in)
7. **Application programs:** Programs that are useful to solve common problems, or to perform common operations.
   Eg. Web browsers, database systems.

Operating System Structures

1. **Simple Structure**
2. **Layered Approach**
3. **Microkernel**

Simple Structure

Many commercial systems do not have a well-defined structure. Frequently, such operating systems started as small, simple, and limited systems, and then grew beyond their original scope.

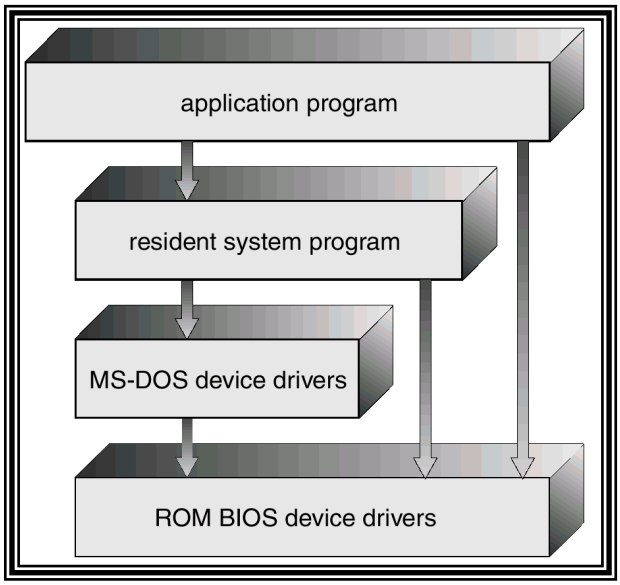
Example 1: MS-DOS.

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

For instance, application programs are able to access the basic I/O routines to write directly to the display and disk drives.

(-) Such freedom leaves MS-DOS vulnerable to malicious programs causing entire system crashes when user program fail.

(-) MS-DOS was also limited by the hardware



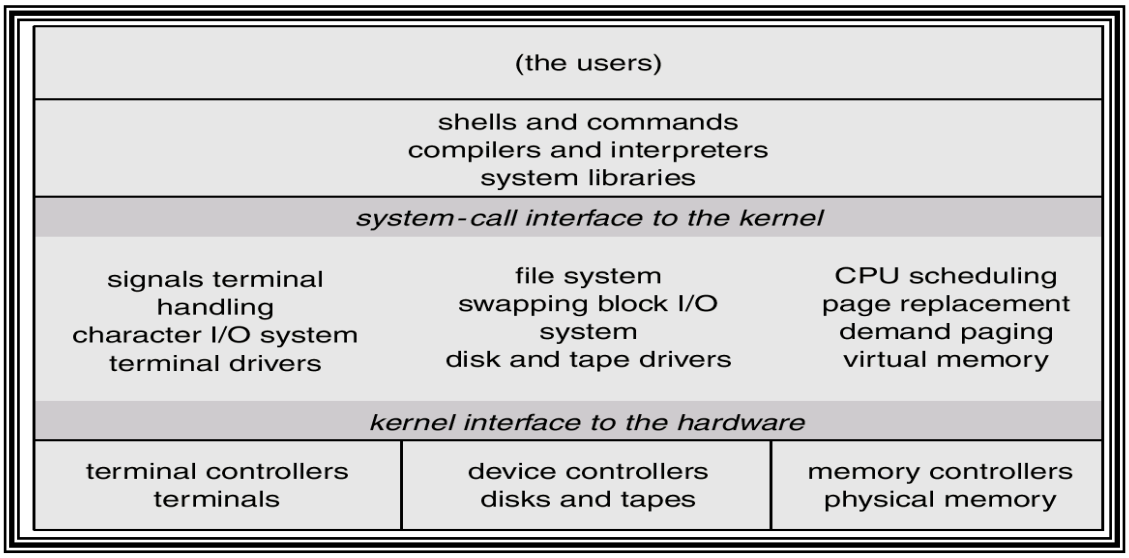
Example 2: UNIX

UNIX is another system that was initially limited by hardware functionality.

It consists of two separable parts:

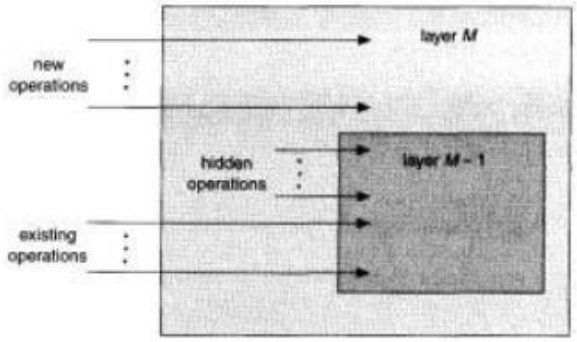
1. **Systems programs** – use kernel supported system calls to provide useful functions such as compilation and file manipulation.
2. **The kernel**
   - Consists of everything below the system-call interface and above the physical hardware

- Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level.



Layered Approach

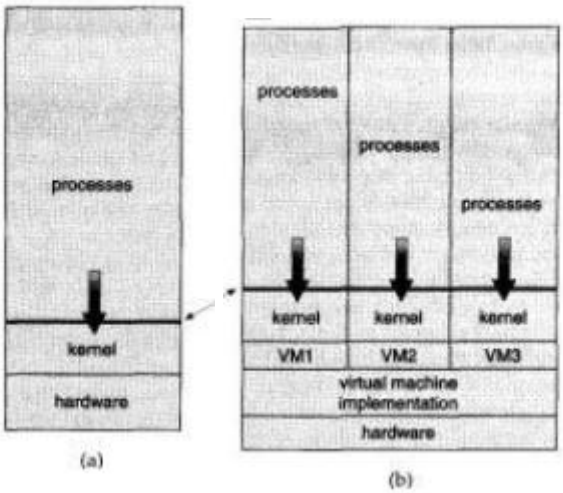
- Given proper hardware support, OS may be broken into smaller, more appropriate pieces.
- The modularization of a system can be done in many ways.
- One method is the layered approach, in which the operating system is broken up into a number of layers (or levels), each built on top of lower layers.
- The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.
- The main advantage of the layered approach is modularity.
- The modularity makes the debugging & verification easy.



Microkernel

- Remove the non-essential components from the kernel into the user space.
- 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

Virtual Machines



System models. (a) Nonvirtual machine. (b) Virtual machine.

- 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. Remember that the virtual-machine software itself will need substantial disk space to provide virtual memory. The solution is to provide I virtual disks, which are identical in all respects except size.
- Users thus are given their own virtual machines. They can then run any of the operating systems or software packages that are available on the underlying machine.

Implementation

- The virtual-machine concept is useful, it is difficult to implement. Much work is required to provide an exact duplicate of the underlying machine. machine has two modes: user mode and monitor mode.
- The virtual-machine software can run in monitor mode, since it is the operating system.
- The virtual machine itself can execute in only user mode. a virtual user mode and a virtual monitor mode, both of which run in a physical user mode.
- Those actions that cause a transfer from user mode to monitor 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 monitor mode on a virtual machine.
- When a system call, is made by a program running on a virtual machine, in virtual user mode, it will cause a transfer to the virtual-machine monitor in the real machine.
- When the virtual-machine monitor gains control, it can change the register contents and program counter for the virtual machine to simulate the effect of the system call.
- It can then restart the virtual machine, noting that it is now in virtual monitor mode.
- If the virtual machine then tries, to read from its virtual card reader, it will execute a privileged I/O instruction.
- When 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.

Virtual machine has several advantages:

- In Virtual machine environment there is complete protection of the various system resources.
- Each virtual machine is completely isolated from all other virtual machines, so we have no security problems as the various system resources are completely protected.
- No direct sharing of resources.
- Sharing in two ways
  i. Share a minidisk (files can be shared)
  ii. Define a network of virtual machines, each of which can send information over the virtual communication network.
- Virtual machine is a perfect vehicle for OS research and development.
- The OS runs on and controls the entire machine. Thus it is necessary to test all changes to the operating system carefully. This period is System Development time since it makes the system unavailable to users, which is eliminated by virtual machine system. System programs are given their own virtual machine and system development is done on the virtual machine, instead of a physical machine.
- Virtual machine solves system compatibility problems.

Java

- Example of continued utility of virtual machine involves the java language. Java is implemented by a compiler that generates byte code output.
- These byte codes are the instructions that run on the Java Virtual Machine (JVM)
- For java programs to run on a platform, that platform must have a JVM running on it.
- JVM implements stack based instructions set that includes the expected arithmetic, logical, data movement and flow control instructions.
- Java compilers can simply emit these byte code instructions and the JVM must implement the necessary functionality on each program .

Process Concept

- A process can be thought of as a program in execution.
- A process is the unit of the unit of work in a modern time-sharing system.

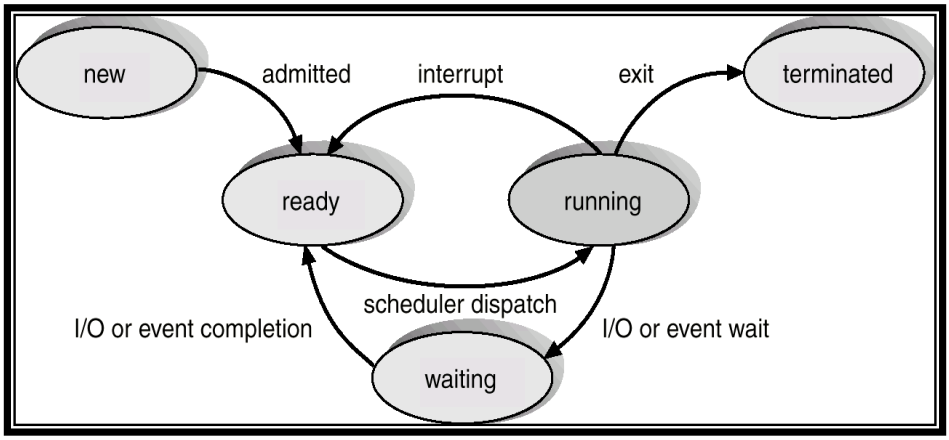
- A process generally includesthe process stack, which contains temporary data (such as method parameters,return addresses, and local variables), and a data section, which contains global variables.

Difference between program and process

- A program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

Process States:

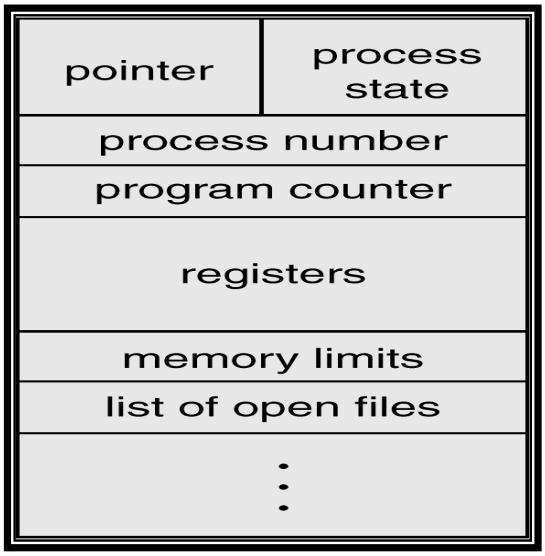
- As a process executes, it changes state.
- The state of a process is defined in part by the current activity of that process.
- Each process may be in one of the following states:
  - **New**: The process is being created.
  - **Running**: Instructions are being executed.
  - **Waiting**: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - **Ready**: The process is waiting to be assigned to a processor.
  - **Terminated**: The process has finished execution.



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 defines a process to the operating system.
- It contains the entire information about a process.
- Some of the information a PCB contans are:
    - **Process state**: The state may be new, ready, running, waiting, halted, and SO on.
    - **Program counter**: The counter indicates the address of the next instruction to be executed for this process.
    - **CPU registers**: The registers vary in number and type, depending on the computer architecture.
    - **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.
    - **Accounting information**: This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
    - **Status information**: The information includes the list of I/O devices allocated to this process, a list of open files, and so on.



Process Scheduling

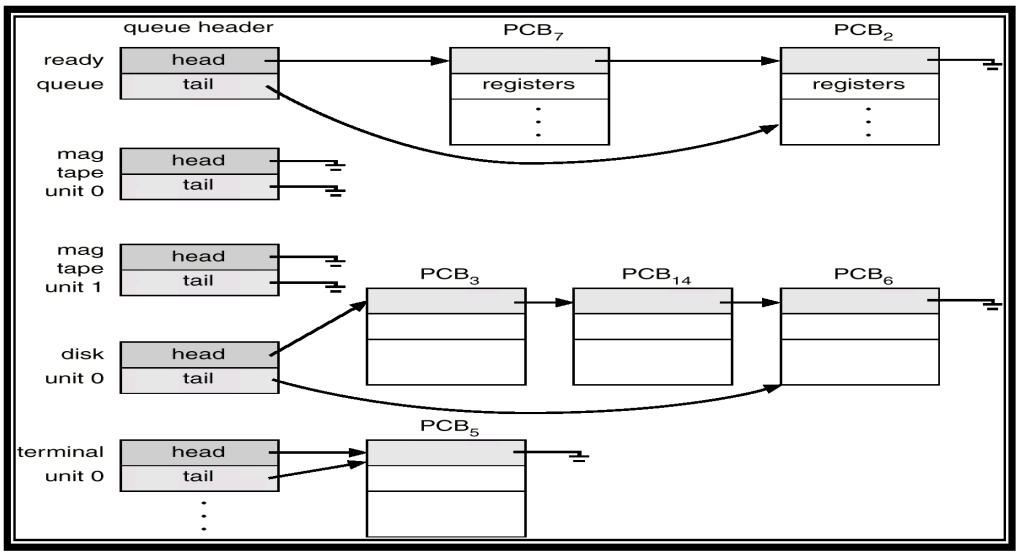
- The objective of multiprogramming is to have some process running at all times, so as to maximize CPU utilization.

Scheduling Queues

There are 3 types of scheduling queues .They are :

1. Job Queue
2. Ready Queue
3. Device Queue

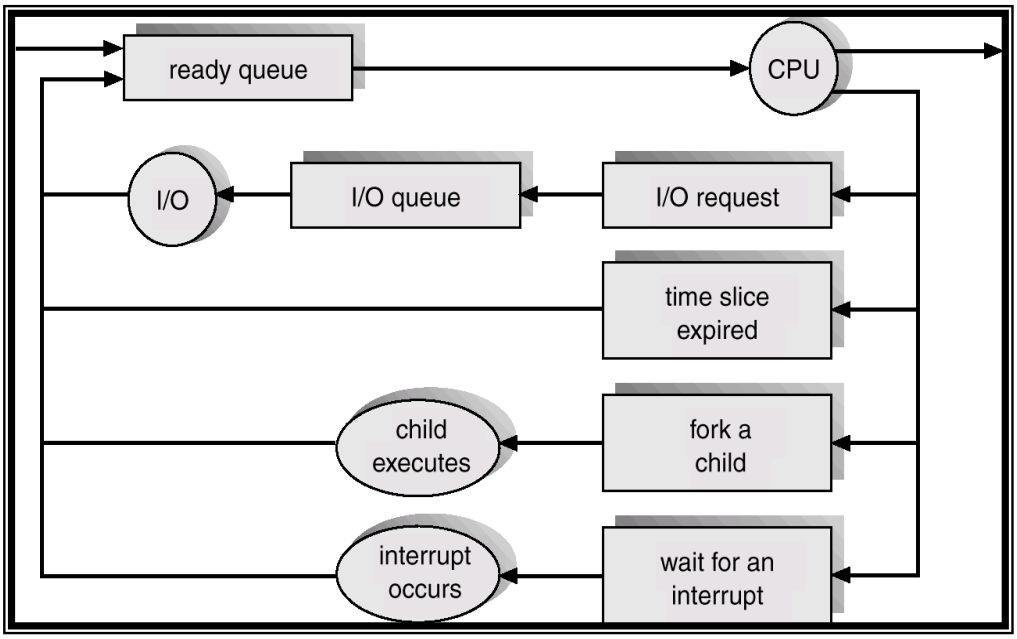
- As processes enter the system, they are put into a **job queue.**
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
- The list of processes waiting for an I/O device is kept in a **device queue** for that particular device.



- 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 tothe 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 subprocess and wait for its termination.

- The process could be removed forcibly from the CPU, as a result of aninterrupt, and be put back in the ready Queue.

- A common representation of process scheduling is a queueing diagram.



Schedulers

- A process migrates between the various scheduling queues throughout its lifetime.
- The operating system must select, for scheduling purposes, processes from these queues in some fashion.
- The selection process is carried out by the appropriate scheduler.

There are three different types of schedulers.They are:

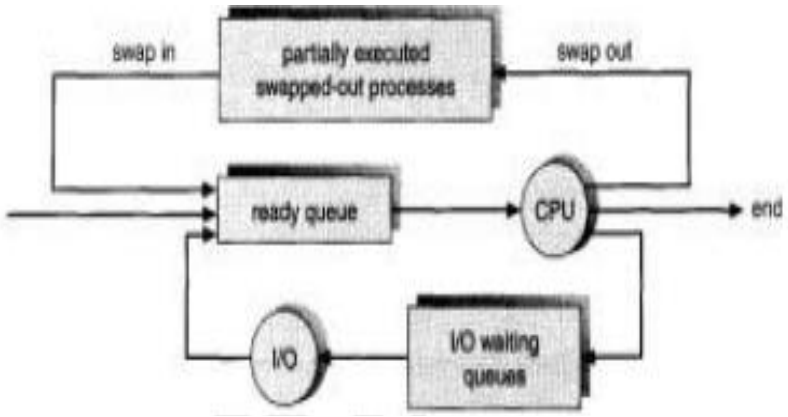
1. Long-term Scheduler or Job Scheduler
2. Short-term Scheduler or CPU Scheduler
3. Medium term Scheduler

- The **long-term scheduler**, or **job scheduler**, selects processes from this pool and loads them into memory for execution. It is invoked very infrequently.It controls the degree of multiprogramming.

- The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute, and allocates the CPU to one of them. It is invoked very frequently.
- Processes can be described as either **I/O bound** or **CPU bound**.
- An **I\O-bound process** spends more of its time doing I/O than it spends doing computations.
- A **CPU-bound process**, on the other hand, generates I/O requests infrequently,using more of its time doing computation than an I/O-bound process uses.
- The system with the best performance will have a combination of CPU-bound and I/O-bound processes.

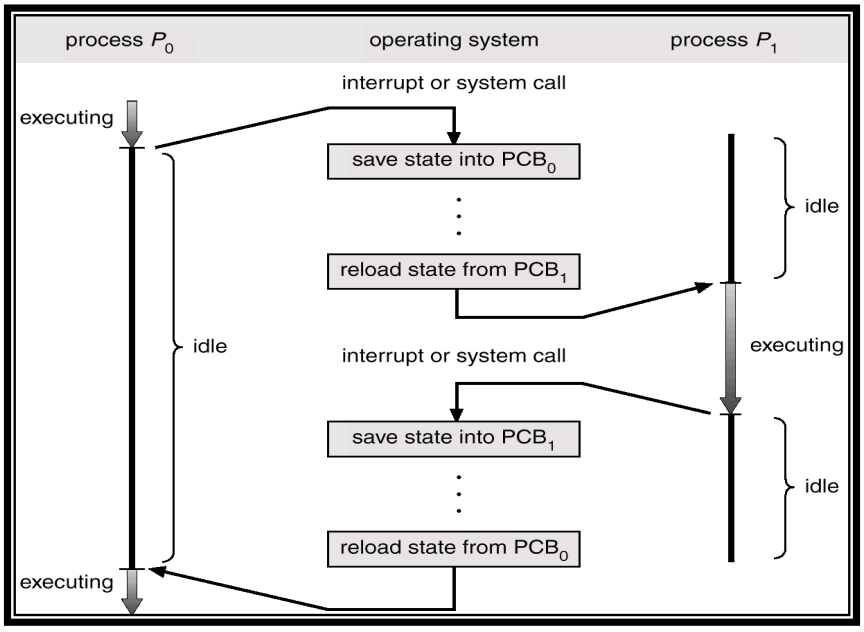
Medium term Scheduler

- Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.
- The key idea is medium-term scheduler, removes processes from memory and thus reduces the degree of multiprogramming.
- At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called swapping.



Context Switch

- 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.
- 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.



Operations on Processes

1. Process Creation

- A process may create several new processes, during the course of execution.
- The creating process is called a **parent** process, whereas the new processes are called the **children** of that process.

- When a process creates a new process, two possibilities exist **in terms of execution:**
    1. The parent continues to execute concurrently with its children.
    2. 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:
    1. The child process is a duplicate of the parent process.
    2. The child process has a program loaded into it.
- In UNIX, each process is identified by its process identifier, which is a unique integer. A new process is created by the **fork** system call.

2. 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.
- At that point, the process may return data (output) to its parent process (via the **wait** system call).
- A process can cause the termination of another process via an appropriate system call.
- A parent may terminate the execution of one of its children for a variety of reasons, such as these:
    1. The child has exceeded its usage of some of the resources that it has been allocated.
    2. The task assigned to the child is no longer required.
    3. The parent is exiting, and the operating system does not allow a child to continue if its parent terminates. On such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as **cascading termination**, is normally initiated by the operating system.

Cooperating Processes

- The concurrent processes executing in the operating system may be either **independent** processes or **cooperating** processes.
- A process is independent if it cannot affect or be affected by the other processes executing in the system.
- A process is cooperating if it can affect or be affected by the other processes executing in the system.
- Benefits of Cooperating Processes

1. Information sharing
2. Computation speedup
3. Modularity
4. Convenience

Example

Producer – Consumer Problem

- A producer process produces information that is consumed by a consumer process.
- For example, a print program produces characters that are consumed by the printer driver. A compiler may produce assembly code, which is consumed by an assembler.
- To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by the consumer.
    - **unbounded-buffer**: places no practical limit on the size of the buffer.
    - **bounded-buffer** : assumes that there is a fixed buffer size.

Shared data

```
#define BUFFER_SIZE 10
typedef struct {
        . . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

The shared buffer is implemented as a circular array with two logical pointers: **in** and **out.** The variable **in** points to the next free position in the buffer; **out** points to the first full position in the buffer. The buffer is empty when **in** == **out** ; the buffer is full when **((in** + 1) % **BUFFERSIZE)** == **out.**

Producer Process

```
while (1)
{
  while (((in + 1) % BUFFER_SIZE) == out);
      /* do nothing */
```

```
    buffer[in] = nextProduced;
    in = (in + 1) % BUFFER_SIZE;
}
```

Consumer process

```
while (1)
{
        while (in == out);
                /* do nothing */
        nextConsumed = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
}
```

Interprocess Communication

- Operating systems provide the means for cooperating processes to communicate with each other via an interprocess communication (PC) facility.
- IPC provides a mechanism to allow processes to communicate and to synchronize their actions.IPC is best provided by a message passing system.

Basic Structure:

- If processes P and Q want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them.
- Physical implementation of the link is done through a hardware bus , network etc,
- There are several methods for logically implementing a link and the operations:
    1. Direct or indirect communication
    2. Symmetric or asymmetric communication
    3. Automatic or explicit buffering
    4. Send by copy or send by reference

5. Fixed-sized or variable-sized messages

Naming

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

1. **Direct Communication**
   - Each process that wants to communicate must explicitly name the recipient or sender of the communication.
   - A communication link in this scheme has the following properties:
     i. A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
     ii. A link is associated with exactly two processes.
     iii. Exactly one link exists between each pair of processes.
   - There are two ways of addressing namely
     - Symmetry in addressing
     - Asymmetry in addressing
   - In symmetry in addressing, the send and receive primitives are defined as:
     send(P, message) →Send a message to process P
     receive(Q, message) →Receive a message from Q
   - In asymmetry in addressing , the send & receive primitives are defined as:
     send (p, message) → send a message to process p
     receive(id, message) → receive message from any process, id is set to the name of the process with which communication has taken place

2. Indirect Communication

- With indirect communication, the messages are sent to and received from mailboxes, or ports.
- The send and receive primitives are defined as follows:
  send (A, message) →Send a message to mailbox A.
  receive (A, message) →Receive a message from mailbox A.

- A communication link has the following properties:
    i. A link is established between a pair of processes only if both members of the pair have a shared mailbox.
    ii. A link may be associated with more than two processes.
    iii. A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox

3. Buffering

- A link has some capacity that determines the number of message that can reside in it temporarily. This property can be viewed as a queue of messages attached to the link.
- There are three ways that such a queue can be implemented.
- **Zero capacity** : Queue length of maximum is 0. No message is waiting in a queue. The sender must wait until the recipient receives the message. (message system with no buffering)
- **Bounded capacity**: The queue has finite length n. Thus at most n messages can reside in it.
- **Unbounded capacity**: The queue has potentially infinite length. Thus any number of messages can wait in it. The sender is never delayed

4. Synchronization

- Message passing may be either blocking or non-blocking.
    1. **Blocking Send** - The sender blocks itself till the message sent by it is received by the receiver.
    2. **Non-blocking Send** - The sender does not block itself after sending the message but continues with its normal operation.
    3. **Blocking Receive** - The receiver blocks itself until it receives the message.
    4. **Non-blocking Receive** – The receiver does not block itself.

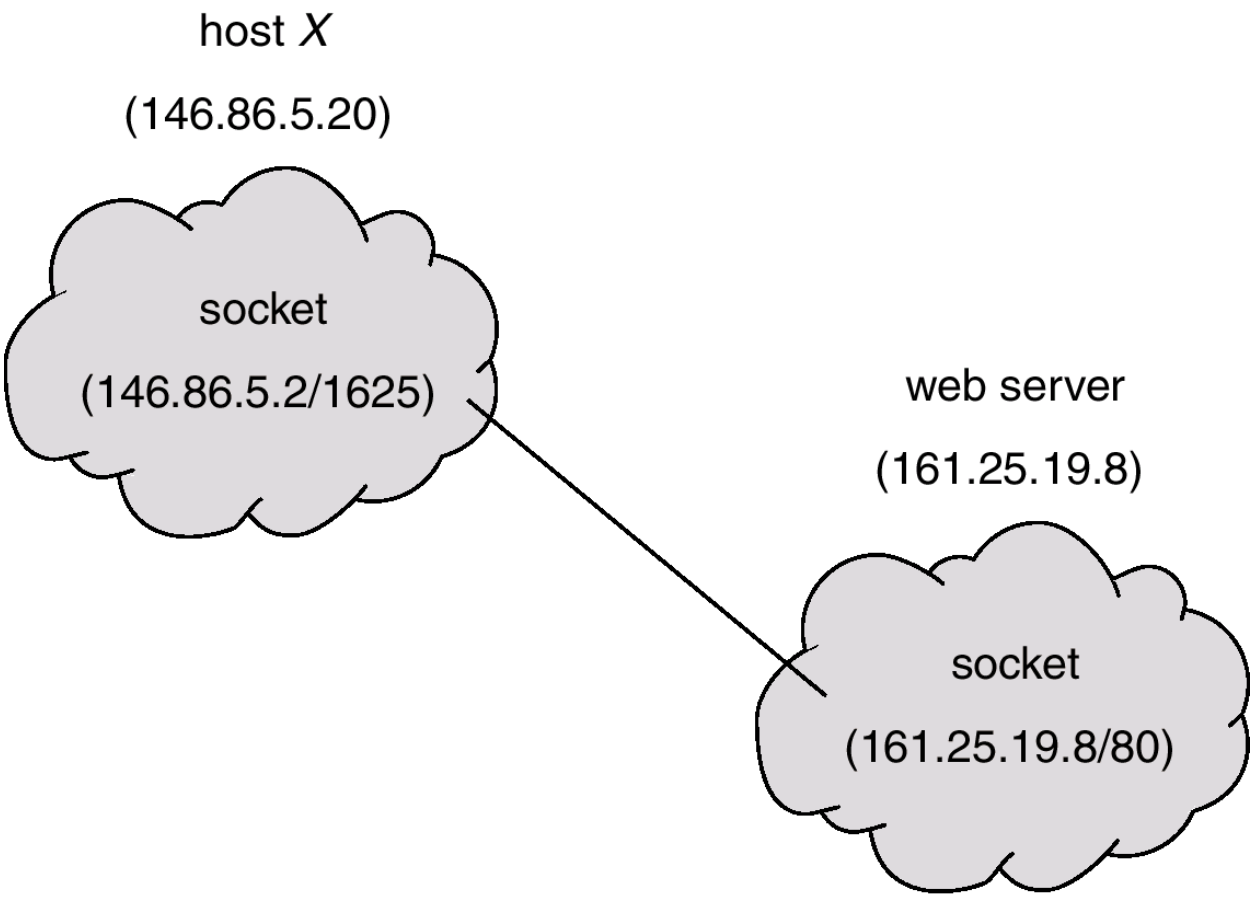
Communication in client – server systems:

There are two levels of communication

- Low – level form of communication – eg. Socket
- High – level form of communcation – eg.RPC , RMI

1. Sockets

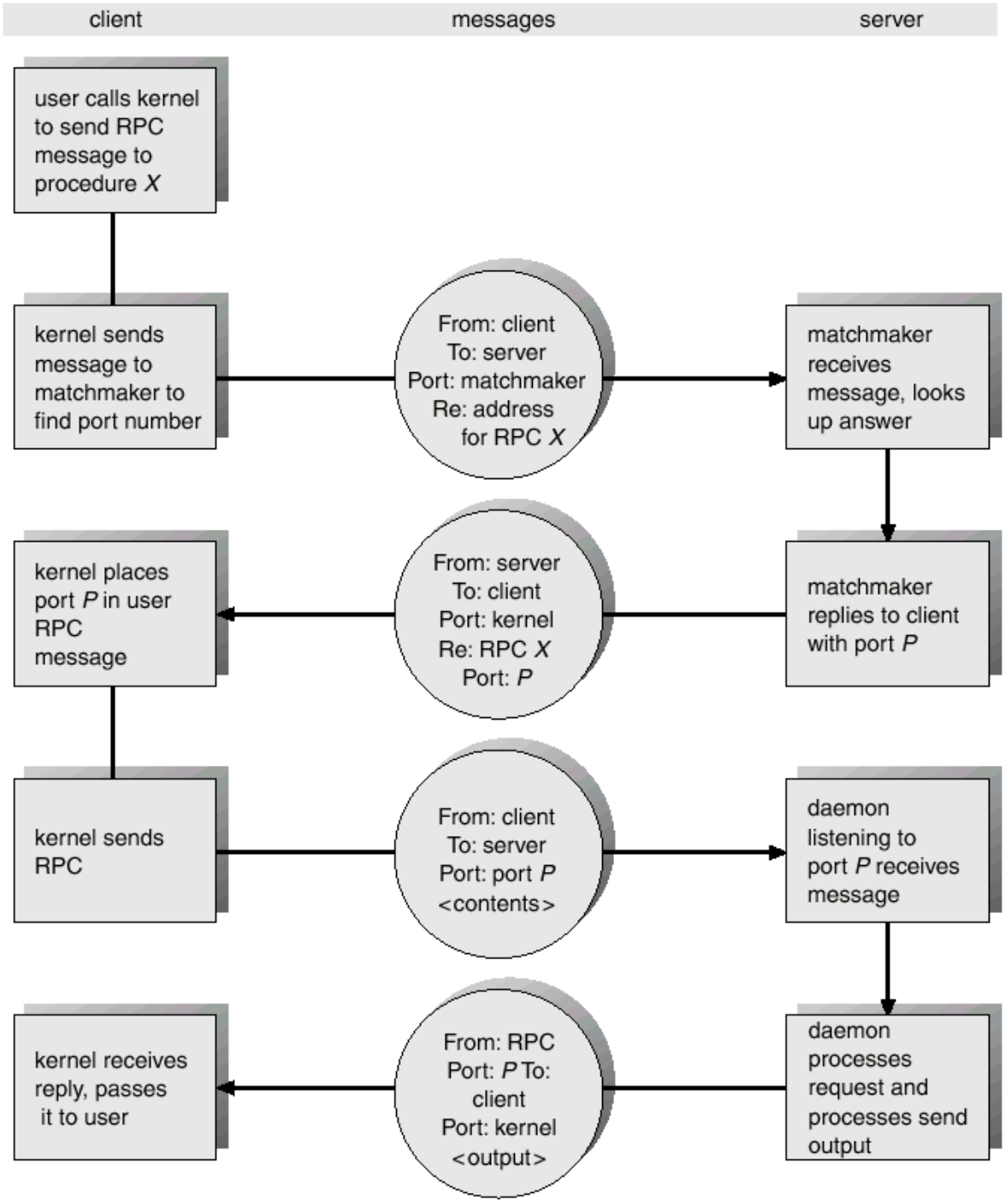
- A socket is defined as an endpoint for communication.
- A socket is made up of an IP address concatenated with a port number.
- It uses a client-server architecture.
- The server waits for incoming requests by listening to specified ports.
- All porst below 1024 are well known ports(eg. 23 is for telnet,21 for ftp ,80 for http etc)
- If a client makes a request for a connection,it is assigned a port number by the host computer (>1024)



- The above diagram shows that a client on host X with IP address 146.86.5.2/1625 wants to communicate with a web server at 161.25.19.8/80
- All connections established are unique.

2. Remote Procedure Call(RPC)

- The messages exchanged for RPC are well structured.
- They are addressed to an RPC daemon listening to a port on the remote system.
- Stubs – client-side proxy for the actual procedure on the server.
- The client-side stub locates the server and marshalls the parameters.
- Parameter Marshalling involves packaging the parameters into a form that can be transmitted over the network.
- The stub then transmits a message to the server using message passing.
- A similar stub on the server side receives this message and invokes the procedure on the server.
- The return values are passed back to the client using the same technique.


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
kernel places port P in user RPC message
From: server To: client Port: kernel Re: RPC X Port: P
matchmaker replies to client with port P
kernel sends RPC
From: client To: server Port: port P <contents>
daemon listening to port P receives message
kernel receives reply, passes it to user
From: RPC Port: P To: client Port: kernel <output>
daemon processes request and processes send output

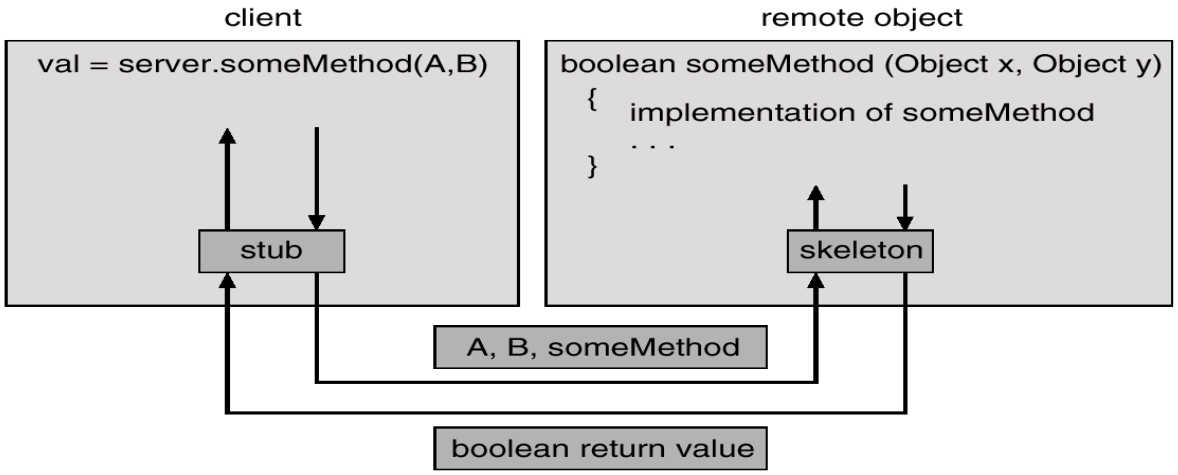
3. Remote Method Invocation(RMI)

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- It allows a thread to invoke a method on a remote object.

Differences between RPC and RMI

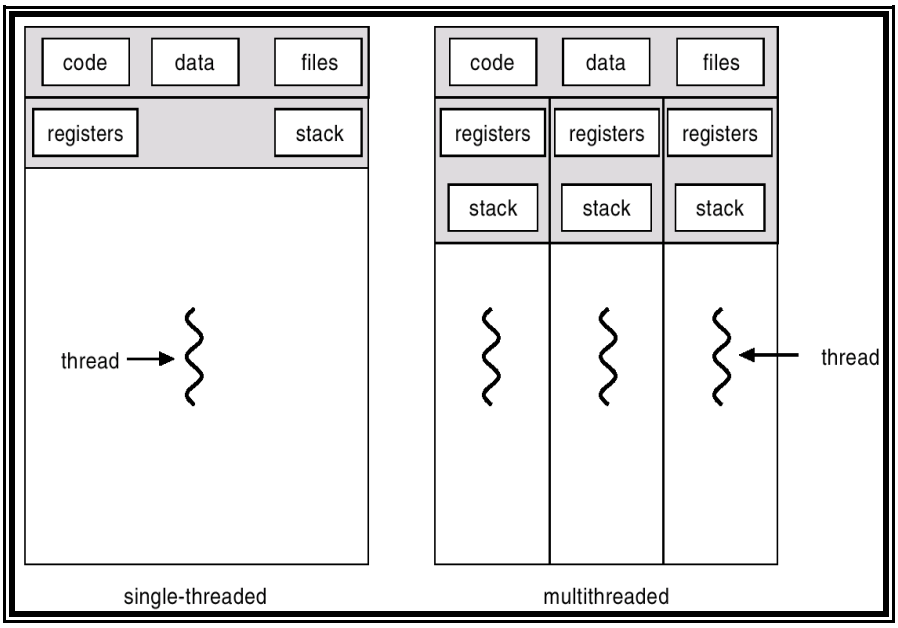
1. RPC supports procedural programming where only remote procedures or functions can be called. RMI is object based.
2. The parameters to remote procedures are ordinary data structures in RPC,with RMI we can pass objects as parameters to remote objects.

- RMI implements remote objects using stubs and skeletons.
- A stub is a proxy for the remote object.
- When a client invokes a remote method,the stub for the remote object is called. This client side stub creates a parcel containing the name of the method to be invoked on the server and the marshaled parameters for the method.
- The stub sends this parcel to the server.
- The skeleton for the remote system receives it.
- It is responsible for unmarshalling the parameters invoking the desired method on the server.
- The skeleton then marshalls the return value into a parcel and return it to the client.
- The stub unmarshalls the return value and passes it to the client.



Threads

- A thread is the **basic unit of CPU utilization**.
- It is sometimes called as a **lightweight process**.
- It consists of a thread ID ,a program counter, a register set and a stack.
- It shares with other threads belonging to the same process its code section , data section, and resources such as open files and signals.



- A traditional or heavy weight process has a single thread of control.
- If the process has multiple threads of control,it can do more than one task at a time.

Benefits of multithreaded programming

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

User thread and Kernel threads

User threads

- Supported above the kernel and implemented by a thread library at the user level.
- Thread creation , management and scheduling are done in user space.
- Fast to create and manage
- When a user thread performs a blocking system call ,it will cause the entire process to block even if other threads are available to run within the application.
- Example: POSIX Pthreads,Mach C-threads and Solaris 2 UI-threads.

Kernel threads

- Supported directly by the OS.
- Thread creation , management and scheduling are done in kernel space.
- Slow to create and manage
- When a kernel thread performs a blocking system call ,the kernel schedules another thread in the application for execution.
- Example: Windows NT, Windows 2000 , Solaris 2,BeOS and Tru64 UNIX support kernel threads.

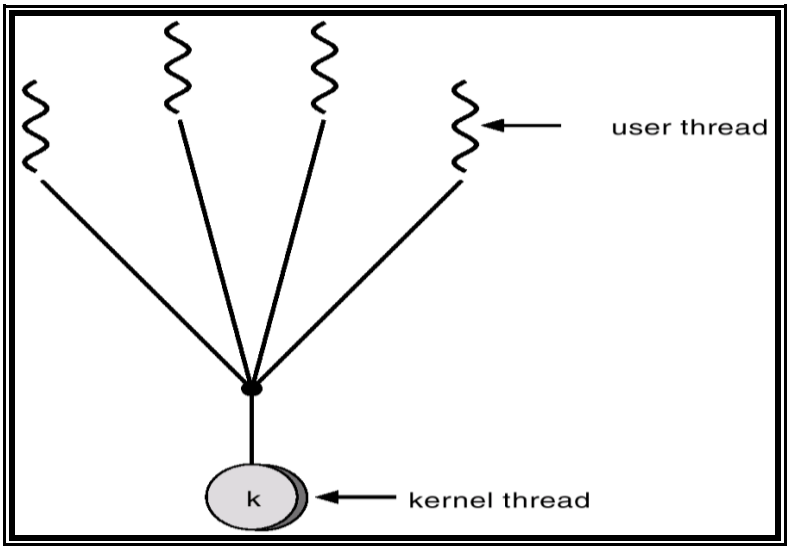
Multithreading models

1. Many-to-One
2. One-to-One
3. Many-to-Many

1. Many-to-One:

- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.

Many-to-One Model



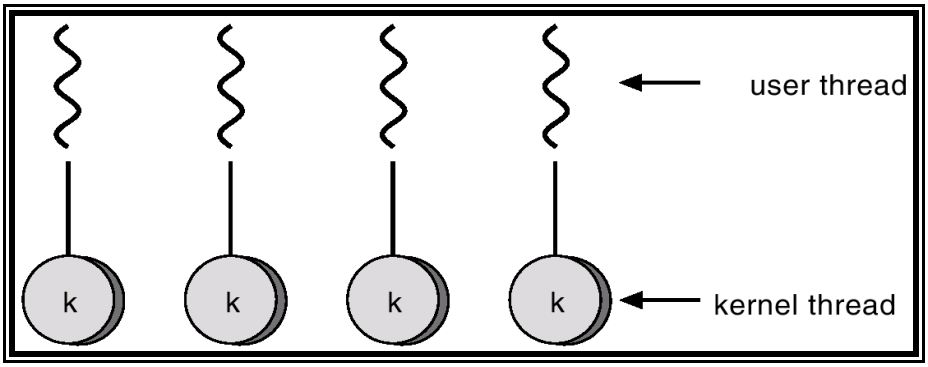
2.One-to-One:

- Each user-level thread maps to a kernel thread.
- Examples

  - Windows 95/98/NT/2000

  - OS/2

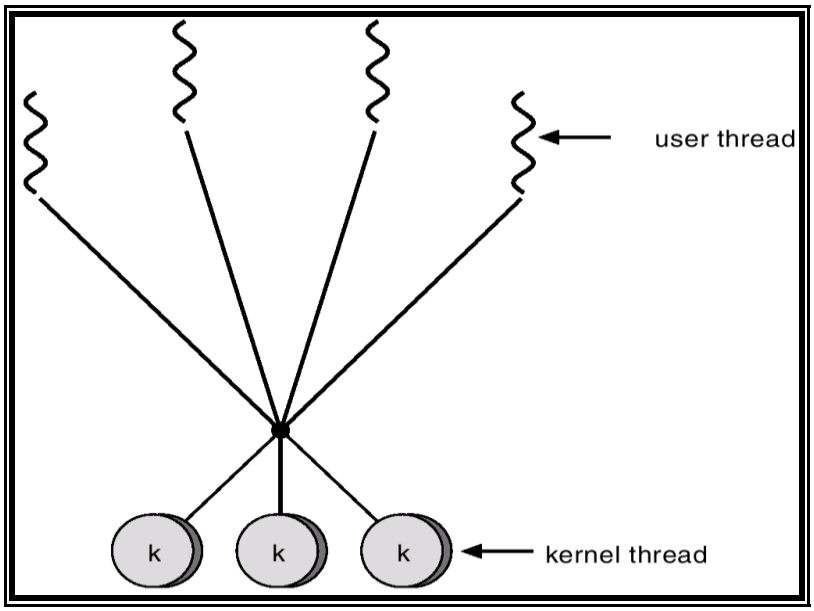
One-to-one Model



3.Many-to-Many Model:

- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- Solaris 2
- Windows NT/2000

Many-to-Many Model



Threading Issues:

1. **fork() and exec() system calls.**

   A fork() system call may duplicate allthreads or duplicate only the thread that invoked fork().

   If a thread invoke exec() system call ,the program specified in the parameter to exec will replace the entire process.

2. Thread cancellation.

It is the task of terminating a thread before it has completed .

A thread that is to be cancelled is called a target thread.

There are two types of cancellation namely

1. Asynchronous Cancellation – One thread immediately terminates the target thread.
2. Deferred Cancellation – The target thread can periodically check if it should terminate , and does so in an orderly fashion.

3. Signal handling

1. A signal is a used to notify a process that a particular event has occurred.
2. A generated signal is delivered to the process.
    a. Deliver the signal to the thread to which the signal applies.
    b. Deliver the signal to every thread in the process.
    c. Deliver the signal to certain threads in the process.
    d. Assign a specific thread to receive all signals for the process.
3. Once delivered the signal must be handled.
    a. Signal is handled by
        i. A default signal handler
        ii. A user defined signal handler

4. Thread pools

Creation of unlimited threads exhaust system resources such as CPU time or memory. Hence we use a thread pool.

In a thread pool , a number of threads are created at process startup and placed in the pool.

When there is a need for a thread the process will pick a thread from the pool and assign it a task.

After completion of the task,the thread is returned to the pool.

5. Thread specific data

Threads belonging to a process share the data of the process. However each thread might need its own copy of certain data known as thread-specific data.