# MODULE 1

## 1. Introduction to Operating Systems, System Structures

- What Operating Systems Do?
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Operating System Services
- System Calls
- Types of System Calls
- System Programs

## 2. Process Management

- Process Concept
- Operations on Processes
- Inter-Process Communication

## 3. Multi-Threaded Programming

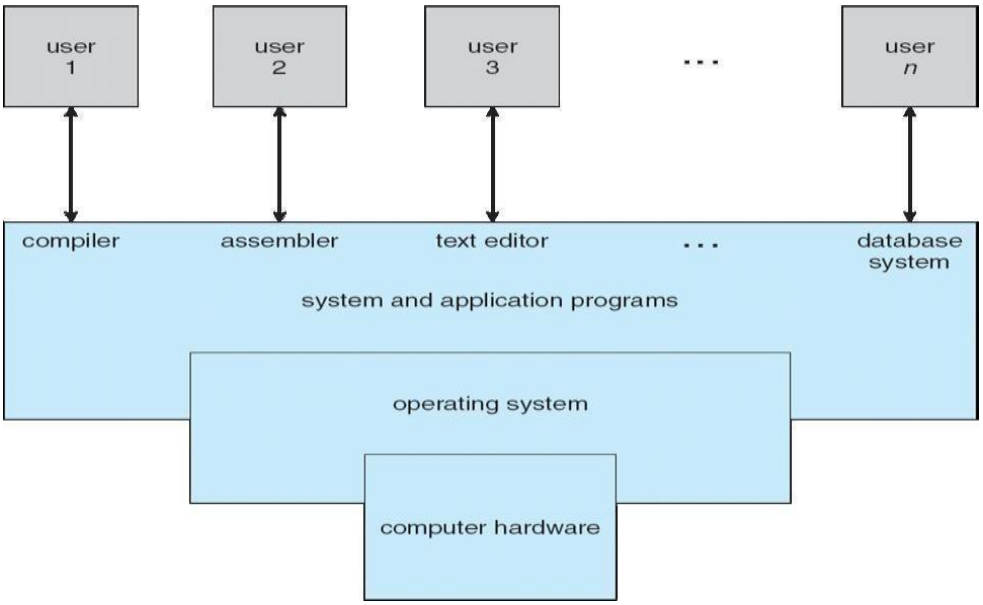
- Overview
- Multithreading Models

# INTRODUCTION TO OPERATING SYSTEMS, SYSTEM STRUCTURES

## What is an Operating System?

An operating system is system software that acts as an intermediary between a user of a computer and the computer hardware. It is software that manages the computer hardware and allows the user to execute programs in a convenient and efficient manner.

## What Operating Systems do?




Abstract view of the components of a computer system.

Computer system mainly consists of four components-

- **Hardware** – provides basic computing resources CPU, memory, I/O devices
- **Operating system** - Controls and coordinates use of hardware among various applications and users
- **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users, Word processors, compilers, web browsers, database systems, video games
- **Users** - People, machines, other computers

**Operating System can be viewed from two viewpoints– User views & System views**

***User Views: -*** The user's view of the operating system depends on the type of user.

- If the user is using **standalone system**, then OS is designed for ease of use and high performances. Here resource utilization is not given importance.
- If the users are at different **terminals** connected to a mainframe or minicomputers, by sharing information and resources, then the OS is designed to maximize resource utilization. OS is designed such that the CPU time, memory and I/O are used efficiently and no single user takesmore than the resource allotted to them.
- If the users are in **workstations**, connected to networks and servers, then the user have a system unit of their own and shares resources and files with other systems. Here the OS is designed for both ease of use and resource availability (files).
- Other systems like embedded systems used in home device (like washing m/c) & automobilesdo not have any user interaction. There are some LEDs to show the status of its work.
- Users of **hand-held systems**, expects the OS to be designed for ease of use and performance per amount of battery life.

***System Views:-*** Operating system can be viewed as a **resource allocator and control program.**

- **Resource allocator** – The OS acts as a manager of hardware and software resources. CPU time, memory space, file-storage space, I/O devices, shared files etc. are the different resources required during execution of a program. There can be conflicting request for these resources by different programs running in same system. The OS assigns the resources to the requesting program depending on the priority.
- **Control Program** – The OS is a control program and manage the execution of user program to prevent errors and improper use of the computer.

## Computer System Architecture

Categorized roughly according to the number of general-purpose processors used.

## Single-Processor Systems –

- The variety of single-processor systems range from PDAs through mainframes. On a single processor system, there is one main CPU capable of executing instructions from user processes. It contains special-purpose processors, in the form of device-specific processors, for devices such as disk, keyboard, and graphics controllers.
- All special-purpose processors run limited instructions and do not run user processes. These are managed by the operating system; the operating system sends them information about their next task and monitors their status.
- For example, a disk-controller processor, implements its own disk queue and scheduling algorithm, thus reducing the task of main CPU. Special processors in the keyboard, converts the keystrokes into codes to be sent to the CPU.
- The use of special-purpose microprocessors is common and does not turn a single-processor system into a multiprocessor. If there is only one general-purpose CPU, then the system is a single-processor system.

## Multi -Processor Systems (parallel systems or tightly coupled systems)

Systems that have two or more processors in close communication, sharing the computer bus, theclock, memory, and peripheral devices are the multiprocessor systems.

**Multiprocessor systems have three main advantages:**

1. **Increased throughput -** In multiprocessor system, as there are multiple processors execution of different programs take place simultaneously. Even if the number of processors is increased the performance cannot be simultaneously increased. This is due to the overhead incurred in keeping all the parts working correctly and also due to the competition for the shared resources. The speed-up ratio with N processors is not N, rather, it is less than N. Thus the speed of the system is not has expected.
2. **Economy of scale -** Multiprocessor systems can cost less than equivalent number of many single-processor systems. As the multiprocessor systems share peripherals, mass storage, and power supplies, the cost of implementing this system is economical. If several

processes are working on the same data, the data can also be shared among them.

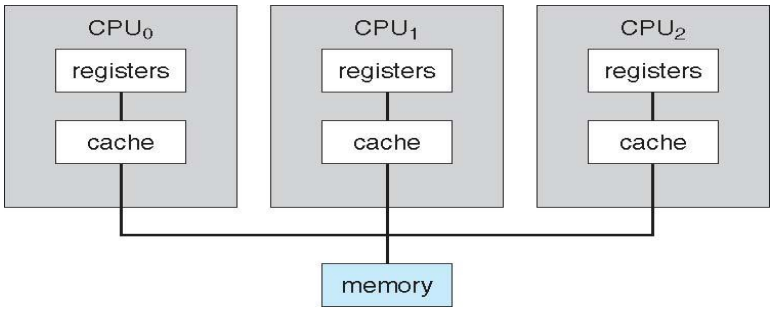
3. **Increased reliability-** In multiprocessor systems functions are shared among several processors. If one processor fails, the system is not halted, it only slows down. The job of the failed processor is taken up, by other processors.

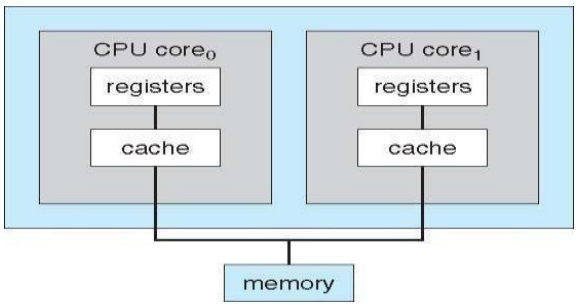
Two techniques to maintain 'Increased Reliability' - **graceful degradation & fault tolerant**

- **Graceful degradation –** As there are multiple processors when one processor fails other process will take up its work and the system goes down slowly.
- **Fault tolerant –** When one processor fails, its operations are stopped, the system failure is then detected, diagnosed, and corrected.

## Different types of multiprocessor systems

1. **Asymmetric multiprocessing – (Master/Slave architecture)** Here each processor is assigned a specific task, by the master processor. A master processor controls the other processors in the system. It schedules and allocates work to the slave processors.
2. **Symmetric multiprocessing (SMP)** – All the processors are considered peers. There is no master-slave relationship. All the processors have their own registers and CPU, only memory is shared.



The benefit of this model is that many processes can run simultaneously. N processes can run if there are N CPUs—without causing a significant deterioration of performance. Operating systems like Windows, Windows XP, Mac OS X, and Linux—now provide support for SMP. A recent trend in CPU design is to include multiple compute cores on a single chip. The communication between processors within a chip is faster than communication between two single processors.

## Clustered Systems

Clustered systems are two or more individual systems connected together via a network and sharing software resources. Clustering provides high availability of resources and services. The service will continue even if one or more systems in the cluster fail. High availability is generally obtained by storing a copy of files (s/w resources) in the system.

**There are two types of Clustered systems – asymmetric and symmetric**

1. **Asymmetric clustering** – one system is in hot-standby mode while the others are 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.
2. **Symmetric clustering** – two or more systems are running applications, and are monitoring each other. This mode is more efficient, as it uses all of the available hardware. If any system fails, its job is taken up by the monitoring system.

Other forms of clusters include parallel clusters and clustering over a wide-area network (WAN). Parallel clusters allow multiple hosts to access the same data on the shared storage. Cluster technology is changing rapidly with the help of SAN (storage-area networks). Using SAN resources can be shared with dozens of systems in a cluster that are separated by miles.

# Operating System Structure

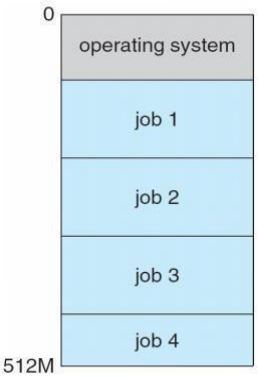
## Multiprogramming

One of the most important aspects of operating systems is the ability to multiprogramming. A single user cannot keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs, so that the CPU always has one to execute.

- The operating system keeps several jobs in memory simultaneously as shown in figure.

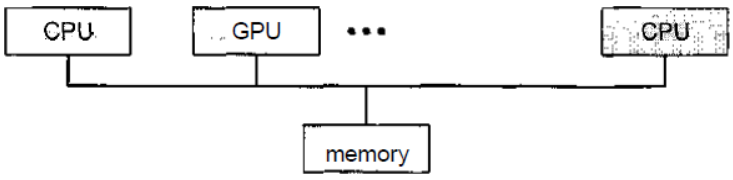
This set of jobs is a subset of the jobs kept in the job pool. Since the number of jobs that can be kept simultaneously in memory is usually smaller than the number of jobs that can be kept in the job pool (in secondary memory). The operating system picks and begins to execute one of the jobs in memory. Eventually, the job may have to wait for some tasks, such as an I/O operation, to complete. In a non-multiprogramming system, the CPU would sit idle.

- In a multiprogrammed system, the operating system simply switches to, and executes, anotherjob. When that job needs to wait, the CPU is switched to another job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back. Thus, the CPU is never idle.
- Multiprogrammed systems provide an environment in which the various system resources (forexample, CPU, memory, and peripheral devices) are utilized effectively, but they do not provide for user interaction with the computer system.




Memory layout for a multiprogramming system

## Multitasking Systems (Time sharing Systems)




- In Time sharing (or multitasking) systems, a single 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. The user feels that all the programs are being

executed at the same time.

- Time sharing requires an interactive (or hands-on) computer system, which provides direct communication between the user and the system. The user gives instructions to the operating system or to a program directly, using a input device such as a keyboard or a mouse, and waits for immediate results on an output device. Accordingly, the response time should be short— typically less than one second.
- A time-shared operating system allows many users to share the computer simultaneously. 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 his use only, even though it is being shared among many users.
- A multiprocessor system is a computer system having two or more CPUs within a single computer system, each sharing main memory and peripherals. Multiple programs are executed by multiple processors parallel.

## Operating-System Operations

Modern operating systems are interrupt driven. If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are signaled by the occurrence of an interrupt or a trap. A trap (or an exception) is a software-generated interrupt. For each type of interrupt, separate segments of code in the operating system determine what action should be taken. An interrupt service routine is provided that is responsible for dealing with the interrupt.

## Dual-Mode Operation

Since the operating system and the user programs share the hardware and software resources of the computer system, it has to be made sure that an error in a user program cannot cause problems to other programs and the Operating System running in the system. The approach taken is to use a hardware support that allows us to differentiate among various modes of execution.

The system can be assumed to work in two separate modes of operation:

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

- A hardware bit of the computer, called the mode bit, is used to indicate the current mode: kernel (0) or user (1). With the mode bit, we are able to distinguish between a task that is executed by the operating system and one that is executed by the user.
- When the computer system is executing a user application, the system is in user mode. Whena user application requests a service from the operating system (via a system call), the transition from user to kernel mode takes place.

At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode. Whenever a trap or interrupt occurs, the hardware switches from user mode to kernel mode (that is, changes the mode bit from 1 to 0). Thus, whenever the operating system gains control of the computer, it is in kernel 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.

- The hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction but rather treats it as illegal and traps it to the operating system. The instruction to switch to user mode is an example of a privileged instruction.
- Initial control is within the operating system, where instructions are executed in kernel mode. When control is given to a user application, the mode is set to user mode. Eventually, control is switched back to the operating system via an interrupt, a trap, or a system call.

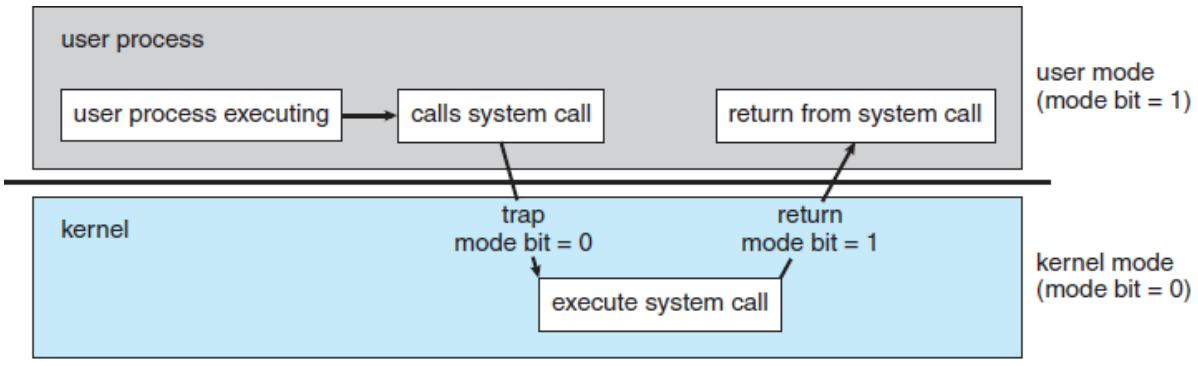
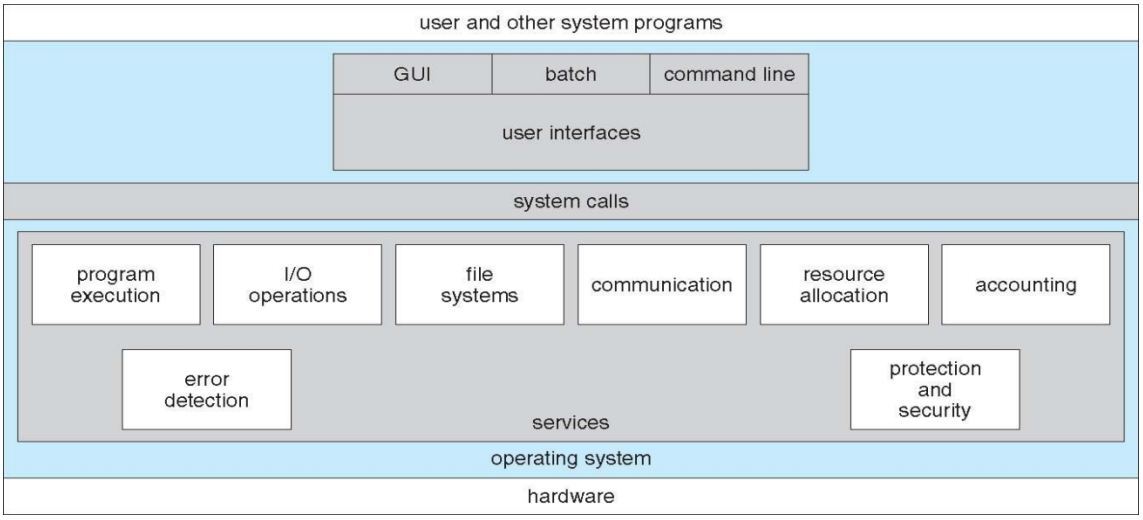



**Figure** Transition from user to kernel mode.

## 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.




**OS provide services for the users of the system, including:**

- **User Interfaces** - Means by which users can issue commands to the system. Depending onthe operating system these may be a **command-line interface** ( e.g. sh, csh, ksh, tcsh, etc.), a **Graphical User Interface** (e.g. Windows, X-Windows, KDE, Gnome, etc.), or a **batch command systems**.
  - In Command Line Interface (CLI) - commands are given to the system.
  - In Batch interface - commands and directives to control these commands are put in a file and then the file is executed.
  - In GUI systems - windows with pointing device to get inputs and keyboard to enter the text.
- **Program Execution** - The OS must be able to load a program into RAM, run the program, and terminate the program, either normally or abnormally.
- **I/O Operations** - The OS is responsible for transferring data to and from I/O devices, including keyboards, terminals, printers, and files. For specific devices, special functions are provided (device drivers) by OS.
- **File-System Manipulation** – Programs need to read and write files or directories. The

services required to create or delete files, search for a file, list the contents of a file and change the file permissions are provided by OS.

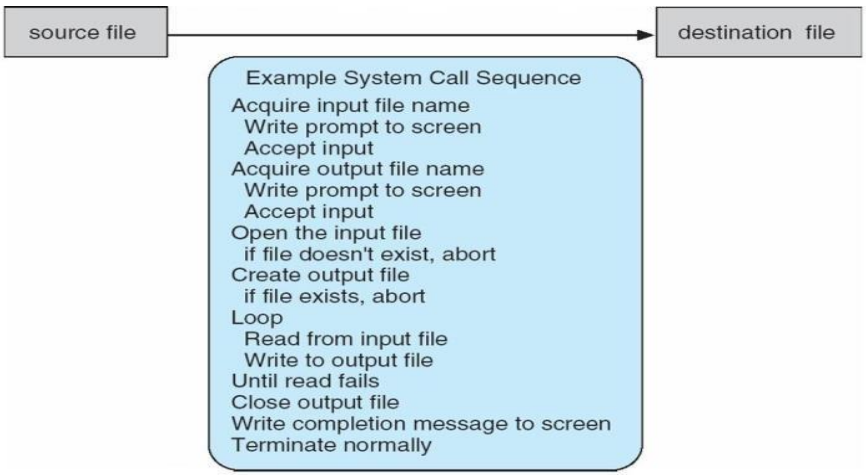
- **Communications** - Inter-process communications, IPC, either between processes running on the same processor, or between processes running on separate processors or separate machines. May be implemented by using the service of OS- like shared memory or message passing.
- **Error Detection** - Both hardware and software errors must be detected and handled appropriately by the OS. Errors may occur in the CPU and memory hardware (such as power failure and memory error), 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).

**OS provide services for the efficient operation of the system, including:**

- **Resource Allocation** – Resources like CPU cycles, main memory, storage space, and I/O devices must be allocated to multiple users and multiple jobs at the same time.
- **Accounting** – There are services in OS to keep track of system activity and resource usage, either for billing purposes or for statistical record keeping that can be used to optimize future performance.
- **Protection and Security** – The owners of information (file) in multiuser or networked computer system may want to control the use of that information. When several separate processes execute concurrently, one process should not interfere with other or with OS. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders must also be done, by means of a password.

## System Calls

- System calls provides an interface to the services of the operating system. These are generally written in C or C++, although some are written in assembly for optimal performance.
- The below figure illustrates the sequence of system calls required to copy a file content fromone file (input file) to another file (output file).

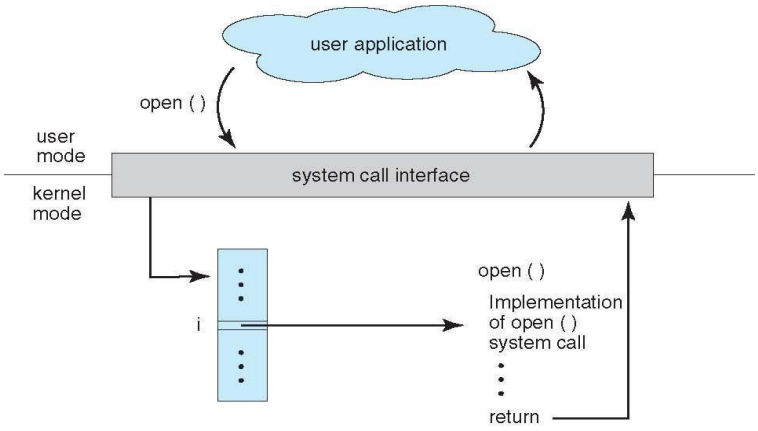



An example to illustrate how system calls are used: writing a simple program to read data from one file and copy them to another file.

- There are number of system calls used to finish this task. The first system call is to write a message on the screen (monitor). Then to accept the input filename. Then another system call to write message on the screen, then to accept the output filename.
- When the program tries to open the input file, it may find that there is no file of that name or that the file is protected against access. In these cases, the program should print a message on the console (another system call) and then terminate abnormally (another system call) and create a new one (another system call).
- Now that both the files are opened, we enter a loop that reads from the input file (another system call) and writes to output file (another system call).
- Finally, after the entire file is copied, the program may close both files (another system call), write a message to the console or window (system call), and finally terminate normally (final system call).
- Most programmers do not use the low-level system calls directly, but instead use an "Application Programming Interface", API.
- Instead of direct system calls provides for greater program portability between different systems. The API then makes the appropriate system calls through the system call interface, using a system call table to access specific numbered system calls.
- Each system call has a specific numbered system call. The system call table (consisting of system call number and address of the particular service) invokes a particular service

routine for a specific system call.

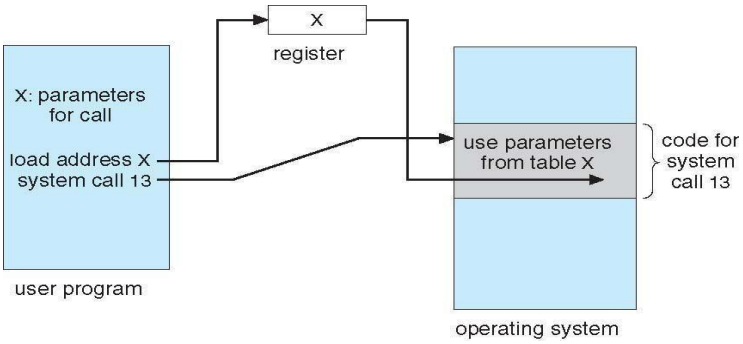
- The caller need know nothing about how the system call is implemented or what it does during execution.




The handling of a user application invoking the open() system call

Three general methods used to pass parameters to OS are –

- To pass parameters in registers.
- If parameters are large blocks, address of block (where parameters are stored in memory) issent to OS in the register. (Linux & Solaris).
- Parameters can be pushed onto the stack by program and popped off the stack by OS.




Passing of parameters as a table.

## Types of System Calls

The system calls can be categorized into six major categories:

1. Process Control
2. File management
3. Device management

4. Information management
5. Communications
6. Protection

## 1) Process Control

- Process control system calls include end, abort, load, execute, create process, terminate process, get/set process attributes, wait for time or event, signal event, and allocate and free memory.
- Processes must be created, launched, monitored, paused, resumed, and eventually stopped.
- When one process pauses or stops, then another must be launched or resumed
- Process attributes like process priority, max. allowable execution time etc. are set and retrieved by OS.
- After creating the new process, the parent process may have to wait (wait time), or wait for an event to occur (wait event). The process sends back a signal when the event has occurred (signal event)

## 2) File Management

The file management functions of OS are –

- File management system calls include create file, delete file, open, close, read, write, reposition, get file attributes, and set file attributes.
- After creating a file, the file is opened. Data is read or written to a file.
- The file pointer may need to be repositioned to a point.
- The file attributes like filename, file type, permissions, etc. are set and retrieved using systemcalls.
- These operations may also be supported for directories as well as ordinary files.

## 3) Device Management

- Device management system calls include request device, release device, read, write, reposition, get/set device attributes, and logically attach or detach devices.

- When a process needs a resource, a request for resource is done. Then the control is granted to the process. If requested resource is already attached to some other process, the requesting process has to wait.
- In multiprogramming systems, after a process uses the device, it has to be returned to OS, so that another process can use the device.
- Devices may be physical (e.g. disk drives), or virtual / abstract (e.g. files, partitions, and RAM disks).

## 4) Information Maintenance

- Information maintenance system calls include calls to get/set the time, date, system data, and process, file, or device attributes.
- These system calls care used to transfer the information between user and the OS. Information like current time & date, no. of current users, version no. of OS, amount of free memory, disk space etc. are passed from OS to the user.

## 5) Communication

- Communication system calls create/delete communication connection, send/receive messages,transfer status information, and attach/detach remote devices.
- The message passing model must support calls to:
    - Identify a remote process and/or host with which to communicate.
    - Establish a connection between the two processes.
    - Open and close the connection as needed.
    - Transmit messages along the connection.
    - Wait for incoming messages, in either a blocking or non-blocking state.
    - Delete the connection when no longer needed.
- The shared memory model must support calls to:
    - Create and access memory that is shared amongst processes (and threads. )
    - Free up shared memory and/or dynamically allocate it as needed.
- Message passing is simpler and easier, (particularly for inter-computer communications), and is generally appropriate for small amounts of data. It is easy to implement, but there are system calls for each read and write process.

- Shared memory is faster, and is generally the better approach where large amounts of data areto be shared. This model is difficult to implement, and it consists of only few system calls.

## 6) Protection

- Protection provides mechanisms for controlling which users / processes have access to which system resources.
- System calls allow the access mechanisms to be adjusted as needed, and for non-privileged users to be granted elevated access permissions under carefully controlled temporary circumstances.

- 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

## System Programs

A collection of programs that provide a convenient environment for program development and execution (other than OS) are called **system programs or system utilities.**

**System programs may be divided into five categories:**

1. **File management** - programs to create, delete, copy, rename, print, list, and generally manipulate files and directories.
2. **Status information** - Utilities to check on the date, time, number of users, processes running, data logging, etc. System registries are used to store and recall configuration information for particular applications.
3. **File modification** - e.g. text editors and other tools which can change file contents.
4. **Programming-language support** - E.g. Compilers, linkers, debuggers, profilers, assemblers, library archive management, interpreters for common languages, and support for make.
5. **Program loading and execution** - loaders, dynamic loaders, overlay loaders, etc., as well asinteractive debuggers.
6. **Communications** - Programs for providing connectivity between processes and users, including mail, web browsers, remote logins, file transfers, and remote command execution.

# PROCESS MANAGEMENT

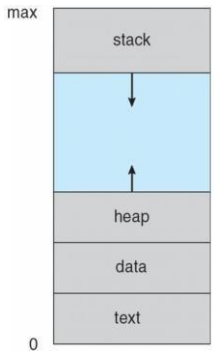
## Process Concept

- A process is a program under execution.
- Its current activity is indicated by PC (Program Counter) and the contents of the processor's registers.

## The Process

Process memory is divided into four sections as shown in the figure below:

- The stack is used to store temporary data such as local variables, function parameters,function return values, return address etc.
- The heap which is memory that is dynamically allocated during process run time
- The data section stores global variables.
- The text section comprises the compiled program code.
- Note that, there is a free space between the stack and the heap. When the stack is full, it grows downwards and when the heap is full, it grows upwards.



Process in memory

## Process State

A Process has 5 states. Each process may be in one of the following states –

1. **New -** The process is in the stage of being created.
2. **Ready -** The process has all the resources it needs to run. It is waiting to be assigned to the processor.

3. **Running –** Instructions are being executed.
4. **Waiting -** The process is waiting for some event to occur. For example, the process may be waiting for keyboard input, disk access request, inter-process messages, a timer to go off, or a child process to finish.
5. **Terminated -** The process has completed its execution.

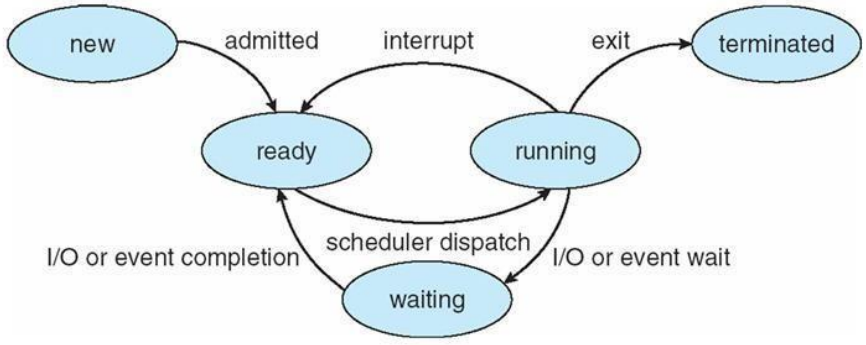



Diagram of process state

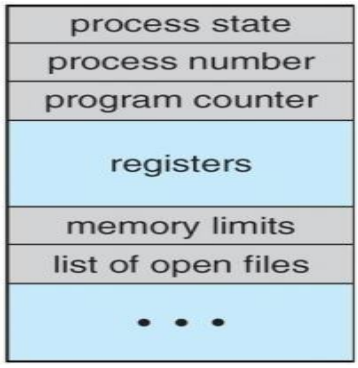
## Process Control Block

For each process there is a Process Control Block (PCB), which stores the process-specific information as shown below –

- **Process State –** The state of the process may be new, ready, running, waiting, 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. They include accumulators, index registers, stack pointers, and general-purpose registers. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.
- **CPU scheduling information-** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **Memory-management information –** This includes information such as the value of the baseand limit registers, the page tables, or the segment tables.
- **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 –** This information includes the list of I/O devices allocated

to the process, a list of open files, and so on.

The PCB simply serves as the repository for any information that may vary from process to process.




Process Control Block (PCB)

**CPU Switch from Process to Process**

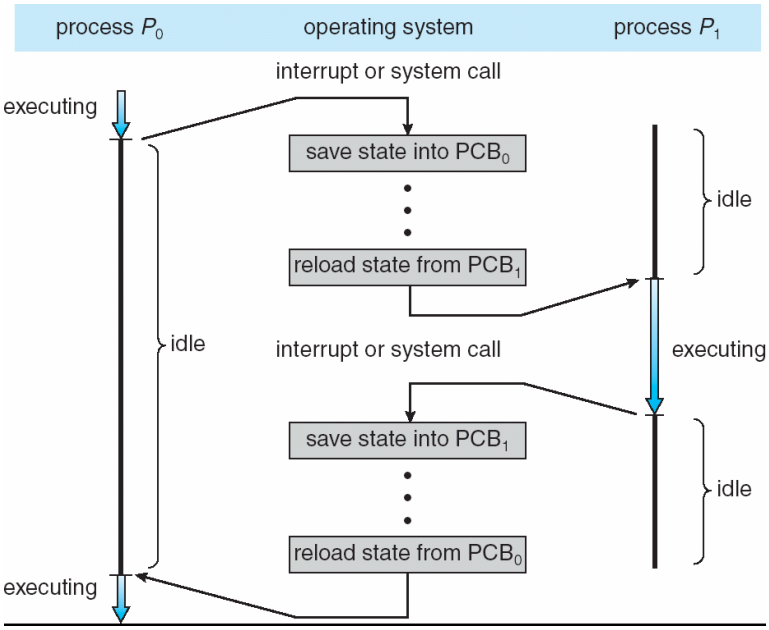



Diagram showing CPU switch from process to process.

## Operations on Processes

### Process Creation

- A process may create several new processes. The creating process is called a parent process, and the new processes are called the children of that process. Each of these

new processes may in turn create other processes. Every process has a unique process ID.

- On typical Solaris systems, the process at the top of the tree is the 'sched' process with PID of 0. The 'sched' process creates several children processes – init, pageout and fsflush. Pageout and fsflush are responsible for managing memory and file systems. The init process with a PID of 1, serves as a parent process for all user processes.

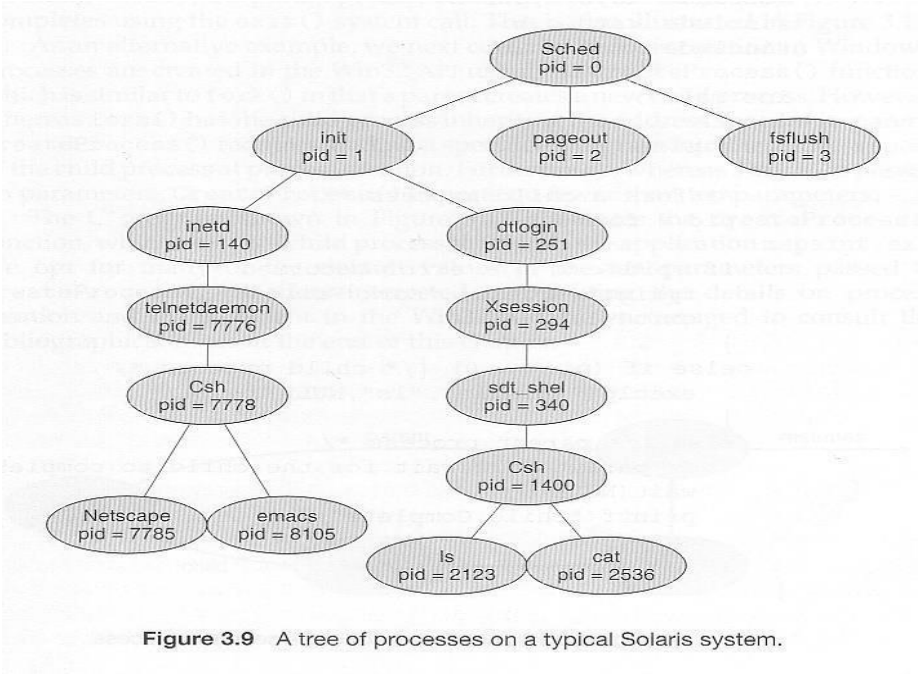



**Figure 3.9** A tree of processes on a typical Solaris system.

A process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task. When a process creates a subprocess, the subprocess may be able to obtain its resources in two ways:

- Directly from the operating system
- Subprocess may take the resources of the parent process.

The resource can be taken from parent in two ways –

- The parent may have to partition its resources among its children
- Share the resources among several children.

There are two options for the parent process after creating the child:

- Wait for the child process to terminate and then continue execution. The parent

makes await() system call.

- Run concurrently with the child, continuing to execute without waiting.

Two possibilities for the address space of the child relative to the parent:

- The child may be an exact duplicate of the parent, sharing the same program and data segments in memory. Each will have their own PCB, including program counter, registers, and PID. This is the behaviour of the fork system call in UNIX.

The child process may have a new program loaded into its address space, with all new code and data segments. This is the behaviour of the spawn system calls in Windows

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

    /* fork a child process */
    pid = fork();

    if (pid < 0) {/* error occurred */
       fprintf(stderr, "Fork Failed");
       exit(-1);
    }
    else if (pid == 0) {/* child process */
       execlp("/bin/ls","ls",NULL);
    }
    else {/* parent process */
       /* parent will wait for the child to complete */
       wait(NULL);
       printf("Child Complete");
       exit(0);
    }
}
```

**Figure 3.10** C program forking a separate process.

In UNIX OS, a child process can be created by **fork()** system call. The fork system call, if successful, returns the PID of the child process to its parents and returns a zero to the child process. If failure, it returns -1 to the parent. Process IDs of current process or its direct parent can be accessed using the getpid( ) and getppid( ) system calls respectively.

The parent waits for the child process to complete with the wait() system call. When the child process completes, the parent process resumes and completes its execution.

In windows the child process is created using the function **createprocess( ).** The createprocess( ) returns 1, if the child is created and returns 0, if the child is not created.

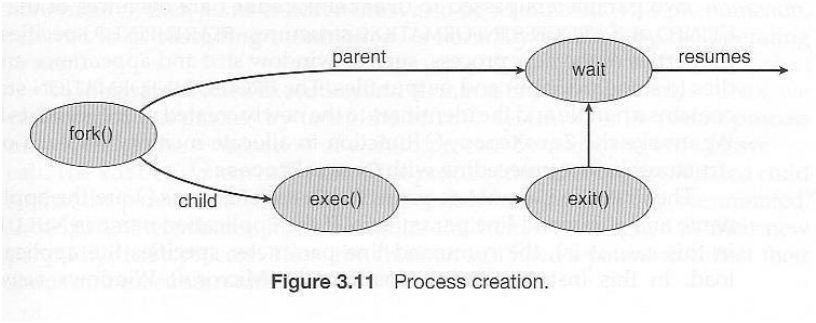



**Figure 3.11** Process creation.

## Process Termination

- A process terminates when it finishes executing its last statement and asks the operating system to delete it, by using the exit () system call. All of the resources assigned to the process like memory, open files, and I/O buffers, are deallocated by the operating system.
- A process can cause the termination of another process by using appropriate system call. The parent process can terminate its child processes by knowing of the PID of the child.
- A parent may terminate the execution of children for a variety of reasons, such as:
  - The child has exceeded its usage of the resources, it has been allocated.
  - The task assigned to the child is no longer required.
  - The parent is exiting, and the operating system terminates all the children. This iscalled **cascading termination.**

## Interprocess Communication

**Interprocess Communication**- Processes executing may be either co-operative or independent processes.

- Independent Processes – processes that cannot affect other processes or be affected by otherprocesses executing in the system.
- Cooperating Processes – processes that can affect other processes or be affected by otherprocesses executing in the system.

Co-operation among processes are allowed for following reasons –

- **Information Sharing** - There may be several processes which need to access the same

file. So the information must be accessible at the same time to all users**.**

- **Computation speedup** - Often a solution to a problem can be solved faster if the problem canbe broken down into sub-tasks, which are solved simultaneously (particularly when multiple processors are involved.)
- **Modularity** - A system can be divided into cooperating modules and executed by sending information among one another.
- **Convenience** - Even a single user can work on multiple tasks by information sharing.

Cooperating processes require some type of inter-process communication. This is allowed by two models:

1. Shared Memory systems
2. Message passing systems.

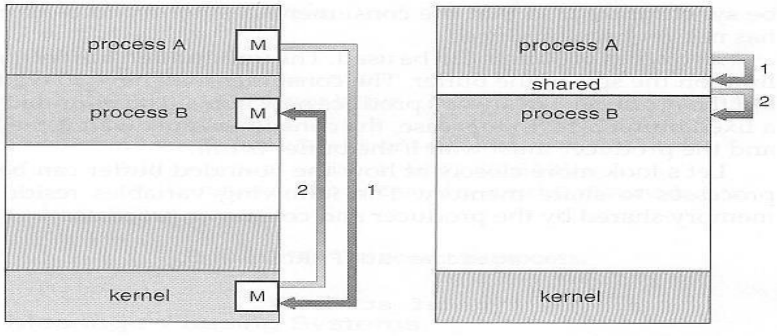



**Figure 3.13** Communications models. (a) Message passing. (b) Shared memory.

| Sl No | **Shared Memory** | **Message passing** |
|---|---|---|
| 1. | A region of memory is shared by communicating processes, into whichthe information is written and read | Message exchange is done amongthe processes by using objects. |
| 2. | Useful for sending large block of data | Useful for sending small data. |
| 3. | System call is used only to create shared memory | System call is used during every read and write operation. |
| 4. | Message is sent faster, as there are no system calls | Message is communicated slowly. |

## 1) Shared-Memory Systems

- A region of shared-memory is created within the address space of a process, which needs to communicate. Other process that needs to communicate uses this shared memory.
- The form of data and position of creating shared memory area is decided by the process. Generally, a few messages must be passed back and forth between the cooperating processes first in order to set up and coordinate the shared memory access.
- The process should take care that the two processes will not write the data to the shared memory at the same time.

### Producer-Consumer Example Using Shared Memory

- This is a classic example, in which one process is producing data and another process is consuming the data.
- The data is passed via an intermediary buffer (shared memory). The producer puts the data to the buffer and the consumer takes out the data from the buffer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced. In this situation, the consumer must wait until an item is produced.
- There are two types of buffers into which information can be put –
  - Unbounded buffer
  - Bounded buffer
- **With Unbounded buffer**- there is no limit on the size of the buffer, and so on the data produced by producer. But the consumer may have to wait for new items.
- **With bounded-buffer** – As the buffer size is fixed. The producer has to wait if the buffer is full and the consumer has to wait if the buffer is empty.

This example uses shared memory as a circular queue. The in and out are two pointers to the array. Note in the code below that only the producer changes "in", and only the consumer changes "out".

**First the following data is set up in the shared memory area**

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
}item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

**The producer process – Note that the buffer is full when [ (in+1) % BUFFER_SIZE == out]**

```
item nextProduced;

while (true) {
     /* produce an item in nextProduced */
     while (((in + 1) % BUFFER-SIZE) == out)
        ; /* do nothing */
     buffer[in] = nextProduced;
     in = (in + 1) % BUFFER_SIZE;
}
```

**Figure** The producer process.

**The consumer process – Note that the buffer is empty when [ in == out]**

```
item nextConsumed;

while (true) {
     while (in == out)
        ; //do nothing

     nextConsumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
     /* consume the item in nextConsumed */
}
```

**Figure** The consumer process.

## 2) Message-Passing Systems

A mechanism to allow process communication without sharing address space. It is used in distributedsystems.

- Message passing systems uses system calls for "send message" and "receive message".
- A communication link must be established between the cooperating processes before messages can be sent.
- There are **three methods** of creating the link between the sender and the receiver.
  - Direct or indirect communication (naming)
  - Synchronous or asynchronous communication (Synchronization)
  - Automatic or explicit buffering.

### 1) Naming

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

**a) Direct communication**- the sender and receiver must explicitly know each other's name. Thesyntax for send() and receive() functions are as follows-

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

Properties of communication link:

- A link is established automatically between every pair of processes that wants to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly one pair of communicating processes
- Between each pair, there exists exactly one link.

Types of addressing in direct communication –

- Symmetric addressing – the above-described communication is symmetric communication.Here both the sender and the receiver processes have to name each other to communicate.
- Asymmetric addressing – Here only the sender's name is mentioned, but the

receiving datacan be from any system.

**send (P**, message) --- Send a message to process **P**

**receive (id**, message). Receive a message from any process

Disadvantages of direct communication – any changes in the identifier of a process, may have to change the identifier in the whole system (sender and receiver), where the messages are sent and received.

**b) Indirect communication -** uses shared mailboxes, or ports.

A mailbox or port is used to send and receive messages. Mailbox is an object into which messages canbe sent and received. It has a unique ID. Using this identifier messages are sent and received.

Two processes can communicate only if they have a shared mailbox. The send and receive functions are –

- **send** (A, message) – send a message to mailbox A
- **receive** (A, message) – receive a message from mailbox A

Properties of communication link:

- A link is established between a pair of processes only if they have a shared mailbox
- A link may be associated with more than two processes
- Between each pair of communicating processes, there may be any number of links, each linkis associated with one mailbox.
- A mail box can be owned by the operating system. It must take steps to –
  - create a new mailbox
  - send and receive messages from mailbox
  - delete mailboxes.

**2) Synchronization**

The send and receive messages can be implemented as either **blocking or non-blocking**.

- **Blocking (synchronous) send** - sending process is blocked (waits) until the message isreceived by receiving process or the mailbox.

- **Non-blocking (asynchronous) send** - sends the message and continues (does not wait)
- **Blocking (synchronous) receive** - The receiving process is blocked until a message is available.
- **Non-blocking (asynchronous) receive** - receives the message without block. The receivedmessage may be a valid message or null.

**3) <u>Buffering</u>**

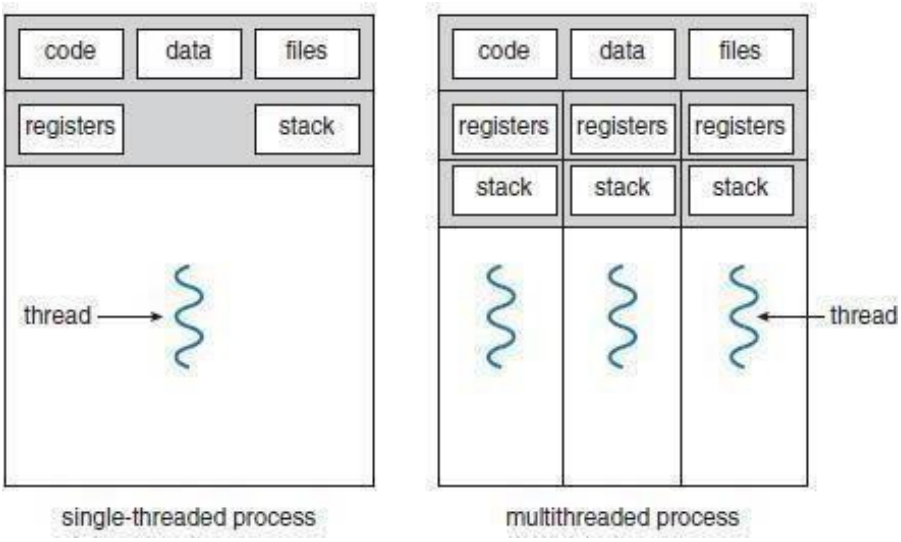
When messages are passed, a temporary queue is created. Such queue can be of three capacities:

- **Zero capacity** – The buffer size is zero (buffer does not exist). Messages are not stored in thequeue. The senders must block until receivers accept the messages.
- **Bounded capacity**- The queue is of fixed size(n). Senders must block if the queue is full.After sending 'n' bytes the sender is blocked.
- **Unbounded capacity** - The queue is of infinite capacity. The sender never blocks.

# MULTITHREADED PROGRAMMING

## Overview

- A thread is a basic unit of CPU utilization
- It consists of
  - thread ID
  - PC
  - register-set and
  - stack
- It shares with other threads belonging to the same process its code-section &data-section.
- A traditional (or heavy weight) process has a single thread of control.
- If a process has multiple threads of control, it can perform more than one task at a time. Such a process is called **multithreaded process.**




## Motivation for Multithreaded Programming

1. The software-packages that run on modern PCs are multithreaded. An application is implemented as a separate process with several threads of control. For ex: A word processor may have

- First thread for displaying graphics
- Second thread for responding to keystrokes
- Third thread for performing grammar checking

2. In some situations, a single application may be required to perform several similar tasks. For ex: A web-server may create a separate thread for each client requests. This allows the server to service several concurrent requests.
3. RPC servers are multithreaded
    - When a server receives a message, it services the message using separate concurrent threads.
4. Most OS kernels are multithreaded.
    - Several threads operate in kernel, and each thread performs a specific task, such as managing devices or interrupt handling.

## Benefits of Multithreaded programming

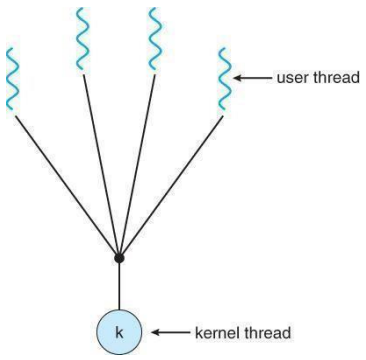
- **Responsiveness**- A program may be allowed to continue running even if part of it is blocked. Thus, increasing responsiveness to the user.
- **Resource Sharing**- By default, threads share the memory (and resources) of the process to which they belong. Thus, an application is allowed to have several different threads of activity within the same address-space.
- **Economy**- Allocating memory and resources for process-creation is costly. Thus, it is more economical to create and context-switch threads.
- **Utilization of Multiprocessor Architectures**- In a multiprocessor architecture, threads may be running in parallel on different processors. Thus, parallelism will be increased.

## Multithreading Models

- Support for threads may be provided at either
    - ✓ The user level, for **user threads** or
    - ✓ By the kernel, for **kernel threads**.
- User-threads are supported above the kernel and are managed without kernel support. Kernel-threads are supported and managed directly by the OS.

- Three ways of establishing relationship between user-threads and kernel-threads:
    1. Many-to-one model
    2. One-to-one model
    3. Many-to-many model

**1) Many-to-one model**




Many user-level threads are mapped to one kernel thread.

**Advantages:**

- Thread management is done by the thread library in user space, so it is efficient.

**Disadvantages:**

- The entire process will block if a thread makes a blocking system-call.
- Multiple threads are unable to run in parallel on multiprocessors.

**For example:**

- Solaris green threads
- GNU portable threads.

**2) One-to-one model**

Each user thread is mapped to a kernel thread.

**Advantages:**

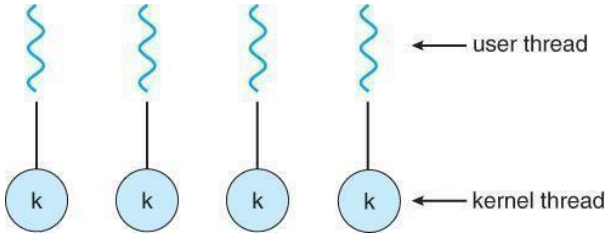
- It provides more concurrency by allowing another thread to run when a thread makes a blocking system-call.
- Multiple threads can run in parallel on multiprocessors.

**Disadvantage:**

- Creating a user thread requires creating the corresponding kernel thread.

**For example:**

- Windows NT/XP/2000, Linux




**3) Many-to-many model**

Many user-level threads are multiplexed to a smaller number of kernel threads.

**Advantages:**

- Developers can create as many user threads as necessary
- The kernel threads can run in parallel on a multiprocessor.
- When a thread performs a blocking system-call, kernel can schedule another thread for execution.

**Two Level Model**

- A variation on the many-to-many model is the two level-model
- Similar to M:N, except that it allows a user thread to be bound to kernel thread.

**For example:**

- HP-UX
- Tru64 UNIX