DIGITAL NOTES
ON
OPERATING SYSTEMS

B.TECH

II YEAR - I SEM

(2019-20)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

SYLLABUS

(R18A0504) OPERATING SYSTEM

	L	T/P/D	C
II Year B.Tech IT -I Sem	**3**	**-/-/-**	**3**

Objectives:

Students will be able:

- To learn the mechanisms of OS to handle processes and threads and their communication
- To learn the mechanisms involved in memory management in contemporary OS
- To gain knowledge on distributed operating system concepts that includes architecture, Mutual exclusion algorithms, deadlock detection algorithms and agreement protocols
- To know the components and management aspects of concurrency management

UNIT - I:

Introduction: Concept of Operating Systems, Generations of Operating systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System.

UNIT - II:

Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching

Thread: Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time. Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJF, RR. Multi processor scheduling: Real Time scheduling: RM and EDF.

UNIT - III:

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer\Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

UNIT - IV:

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and External fragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, Protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU).

UNIT - V:

I/O Hardware: I/O devices, Device controllers, Direct memory access Principles of I/O Software: Goals of Interrupt handlers, Device drivers, Device independent I/O software, Secondary-Storage Structure: Disk structure, Disk scheduling algorithms

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), Free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance.

Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks

TEXT BOOKS:

1. Operating System Concepts Essentials, 9th Edition by AviSilberschatz, Peter Galvin, Greg Gagne, Wiley Asia Student Edition.
2. Operating Systems: Internals and Design Principles, 5th Edition, William Stallings, Prentice Hall of India.

REFERENCE BOOKS:

1. Operating System: A Design-oriented Approach, 1st Edition by Charles Crowley, Irwin Publishing
2. Operating Systems: A Modern Perspective, 2nd Edition by Gary J. Nutt, Addison-Wesley
3. Design of the Unix Operating Systems, 8th Edition by Maurice Bach, Prentice-Hall of India
4. Understanding the Linux Kernel, 3rd Edition, Daniel P. Bovet, Marco Cesati, O'Reilly and Associates

Course Outcomes :

At the end of the course students will be able to:

1. Create processes and threads.
2. Develop algorithms for process scheduling for a given specification of CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time.
3. For a given specification of memory organization develop the techniques for optimally allocating memory to processes by increasing memory utilization and for improving the access time.
4. Design and implement file management system.
5. Develop the I/O management functions in OS for the given I/O devices and OS.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

INDEX

UNIT - I

Introduction: Concept of Operating Systems, Generations of Operating systems, Types of Operating Systems, OS Services, System Calls, Structure of an OS - Layered, Monolithic, Microkernel Operating Systems, Concept of Virtual Machine. Case study on UNIX and WINDOWS Operating System.

Basic Operating System Concepts

Each computer system includes a basic set of programs called the *operating system*. The most important program in the set is called the *kernel*. It is loaded into RAM when the system boots and contains many critical procedures that are needed for the system to operate. The other programs are less crucial utilities; they can provide a wide variety of interactive experiences for the user—as well as doing all the jobs the user bought the computer for—but the essential shape and capabilities of the system are determined by the kernel. The kernel provides key facilities to everything else on the system and determines many of the characteristics of higher software. Hence, we often use the term "operating system" as a synonym for "kernel."

The operating system must fulfill two main objectives:

- Interact with the hardware components, servicing all low-level programmable elements included in the hardware platform.
- Provide an execution environment to the applications that run on the computer system (the so-called user programs).Some operating systems allow all user programs to directly play with the hardware components (a typical example is MS-DOS). In contrast, a Unix-like operating system hides all low-level details concerning the physical organization of the computer from applications run by the user. When a program wants to use a hardware resource, it must issue a request to the operating system. The kernel evaluates the request and, if it chooses to grant the resource, interacts with the proper hardware components on behalf of the user program.To enforce this mechanism, modern operating systems rely on the availability of specific hardware features that forbid user programs to directly interact with low-level hardware components or to access arbitrary memory locations. In particular, the hardware introduces at least two different *execution modes* for the CPU: a nonprivileged mode for user programs and a privileged mode for the kernel. Unix calls these *User Mode* and *Kernel Mode* , respectively.In the rest of this chapter, we introduce the basic concepts that have motivated

the design of Unix over the past two decades, as well as Linux and other operating systems. While the concepts are probably familiar to you as a Linux user, these sections try to delve into them a bit more deeply than usual to explain the requirements they place on an operating system kernel. These broad considerations refer to virtually all Unix-like systems. The other chapters of this book will hopefully help you understand the Linux kernel internals.

Generations of operating systems

Operating systems, like computer hardware, have undergone a series of revolutionary changes called generations. In computer hardware, generations have been marked by major advances in componentry from vacuum tubes (first generation), to transistors (second generation), to integrated circuitry (third generation), to large-scale and very large-scale integrated circuitry (forth generation). The successive hardware generations have each been accompanied by dramatic reductions in costs, size, heat emission, and energy consumption, and by dramatic increases in speed and storage capacity.

(1) the zeroth generation (1940s)

Early computing systems had no operating system. Users had complete access to the machine language. They hand-coded all instructions.

(2) the first generation (1950s)

The operating systems of the 1950s were designed to smooth the transition between jobs. Before the systems were developed, a great deal of time was lost between the completion of one job and the initiation of the next. This was the beginning of batch processing systems in which jobs were gathered in groups or batches. Once a job was running, it had total control of the machine. As each job terminated (either normally or abnormally), control was returned to the operatin system that "cleaned up after the job" and read in and initiated the next job.

(3) The second generation (early 1960s)

The second generation of operating systems was characterized by the development of shared systems with multiprogramming and beginnings of multiprocessing. In multiprogramming systems several user programs are in main storage at once and the processor is switched rapidly between the jobs. In multiprocessing systems, several processors are used on a single computer system to increase the processing power of the machine.

(4) The third generation (Mid-1960s to Mid-1970s)

The third generation of operating systems effectively began with the introduction of the IBM system/360 family of computers in 1964. third generation computers were designed to be general-purpose systems. They were large, often ponderous, systems purporting to be all things to all people. The concept sold a lot of computers, but it took its toll. Users running particular applications that did not require this kind of power payed heavily in increased run-time over head, learning time, debugging time, maintenance, etc.

(5) The fourth generation (Mid-1970s to present)

Fourth generation systems are the current state of the art. Many designers and users are still smarting from their experiences with third generation operating systems and are careful before getting involved with complex operating systems.

With the widespread use of computer networking and on-line processing, user gain access to networks of geographically dispersed computers through various type of terminals. The microprocessor has made possible the development of the personal computer, one of the most important developments of social consequence in the last several decades. Now many users have dedicated computer systems available for their own use at any time of the day or night. Computer power that cost hundreds of thousands of dollars in the early 1960s is now available for less than a thousand dollars.

OS Services

An Operating System supplies different kinds of services to both the users and to the programs as well. It also provides application programs (that run within an Operating system) an environment to execute it freely. It provides users the services run various programs in a convenient manner.

Here is a list of common services offered by an almost all operating systems:

- User Interface
- Program Execution
- File system manipulation
- Input / Output Operations
- Communication
- Resource Allocation
- Error Detection
- Accounting

- Security and protection

This chapter will give a brief description of what services an operating system usually provide to users and those programs that are and will be running within it.

SYSTEM CALLS

System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++, although certain low-level tasks (for example, tasks where hardware must be accessed directly) may have to be written using assembly-language instructions.

TYPES OF SYSTEM CALLS

System calls can be grouped roughly into six major categories: process control, file manipulation, device manipulation, information maintenance, communications, and protection

A running program needs to be able to halt its execution either normally (end()) or abnormally (abort()). If a system call is made to terminate the currently running program abnormally, or if the program runs into a problem
and causes an error trap, a dump of memory is sometimes taken and an error message generated. The dump is written to disk and may be examined by a debugger—a system program designed to aid the programmer in finding and correcting errors, or bugs—to determine the cause of the problem

File Management :System calls that need to be able to create() and delete() files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open() it and to use it. We may also read(), write(), or reposition() (rewind or skip to the end of the file, for example). Finally, we need to close() the file, indicating that we are no longer using it.

Device Management: A system with multiple users may require us to first request() a device, to ensure exclusive use of it. After we are finished with the device, we release() it. These functions are similar to the open() and close() system calls for files. Once the device has been requested (and allocated to us), we can read(),
write(), and (possibly) reposition() the device, just as we can with files.

Information Maintenance: Many system calls exist simply for the purpose of transferring information between the user program and the operating system. time() and date(). dump() - This provision is useful for debugging. A program trace lists each system call as it is executed.

Communication: There are two common models of interprocess communication: the messagepassing model and the shared-memory model. message-passing model, the communicating processes exchange messages with one another to transfer information.

process name, and this name is translated into an identifier by which the operating system can refer to the process. The get hostid() and get processid() system callsshared-memory model, processes use shared memory create() and shared memory attach()

Protection: Protection provides a mechanism for controlling access to the resources provided by a computer system.set permission() and get permission(), which manipulate the permission settings of resources such as files and disks. The allow user() and deny user() system calls specify whether particular users can—or cannot—be allowed access to certain resources.

OPERATING SYSTEM STRUCTURE

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

Simple Structure: OS started as small, simple, and limited systems and then grew beyond their original scope. MS- DOS is an example of such a system.

Layered Approach: layered approach, in which the operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer N) is the user interface.

Microkernels: In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach. This method structures the operating system by removing all nonessential components fromthe kernel and implementing them as system and user-level programs. The result is a smaller kernel.

Modules: operating-system design involves using loadable kernel modules. Here, the kernel has a set of core components and links in additional services via modules, either at boot time or during run time. This type of design is common in modern implementations of UNIX, such as Solaris, Linux, and Mac OS X, as well as Windows

Hybrid Systems: In practice, very few operating systems adopt a single, strictly defined structure. Instead, they combine different structures, resulting in hybrid systems that address performance, security, and usability issues. For example, both Linux and Solaris are monolithic, because having the operating system in a single address space provides very efficient performance. However, they are also modular, so that new functionality can be dynamically added to the kernel.

Mac OS X : The Apple Mac OS X operating system uses a hybrid structure. It hass a layered system.

iOS: iOS is a mobile operating system designed by Apple to run its smartphone, the

iPhone, as well as its tablet computer, the iPad.

Android: The Android operating system was designed by the Open Handset Alliance (led primarily by Google) and was developed for Android smartphones and tablet computers.

UNIT - II:

Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching

Thread: Definition, Various states, Benefits of threads, Types of threads, Concept of multithreads Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers,

Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time. Scheduling algorithms: Pre-emptive and Non pre-emptive, FCFS, SJF, RR. Multi processor scheduling: Real Time scheduling: RM and EDF.

Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections – stack, heap, text and data. The following image shows a simplified layout of a process inside main memory

Stack

The process Stack contains the temporary data such as method/function parameters, return address and local variables.

Heap

This is dynamically allocated memory to a process during its run time.

Text

This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.

Data

This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language –

```c
#include <stdio.h>


int main() {
  printf("Hello, World! \n");
  return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task

is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start**state or while running it by but interrupted by the scheduler to assign CPU to some other process.

3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.
4	**Waiting** **Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.**
5	**Terminated or Exit** **Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.**

Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating

	system.
4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB

Operating System - Process Scheduling

Definition

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

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

Process Scheduling Queues

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

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.
- The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.
- Two-State Process Model
- Two-state process model refers to running and non-running states which are described below

S.N.	State & Description
1	**Running**

	When a new process is created, it enters into the system as in the running state.
2	**Not Running** Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Schedulers

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

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

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

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

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

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Comparison among Scheduler

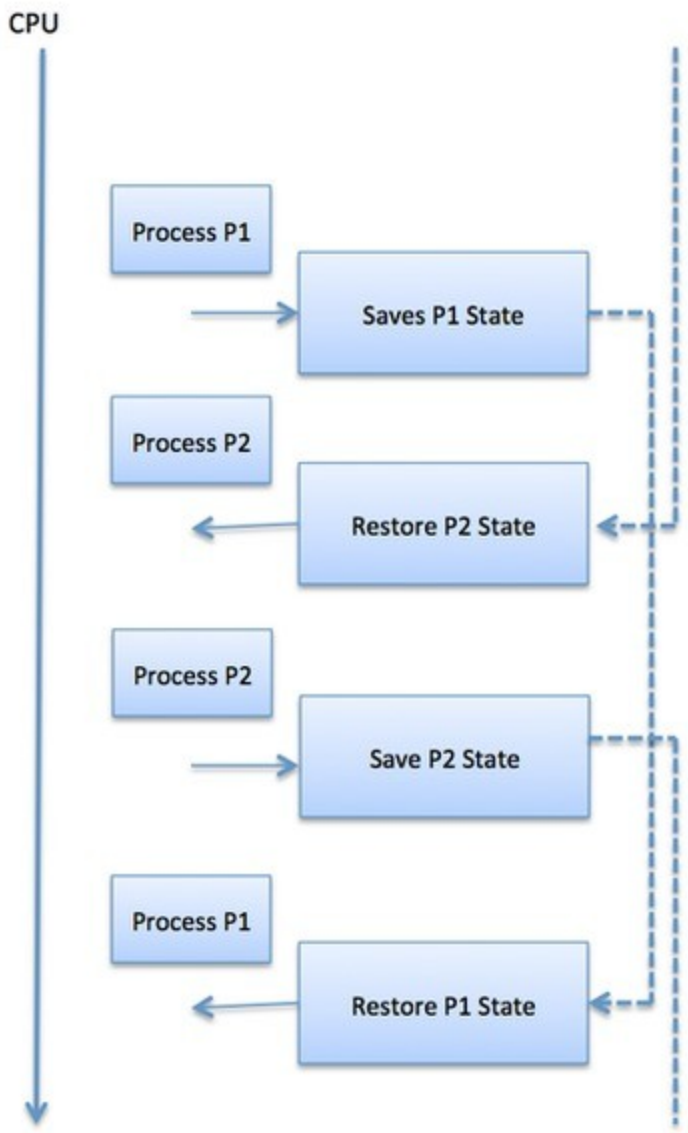
S.N.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.

4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Context switches are computationally intensive since register and memory state must be saved and restored. To avoid the amount of context switching time, some hardware systems employ two or more sets of processor registers. When the process is switched, the following information is stored for later use.

- Program Counter
- Scheduling information
- Base and limit register value
- Currently used register
- Changed State

- I/O State information
- Accounting information

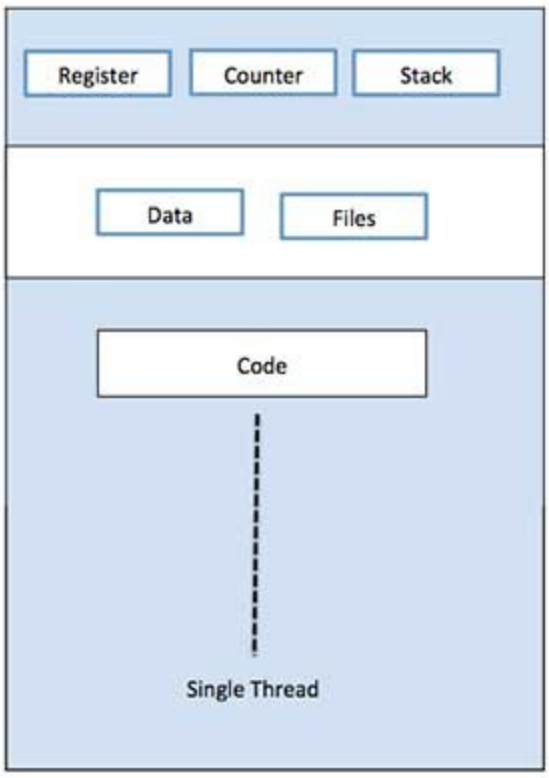
What is Thread?

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

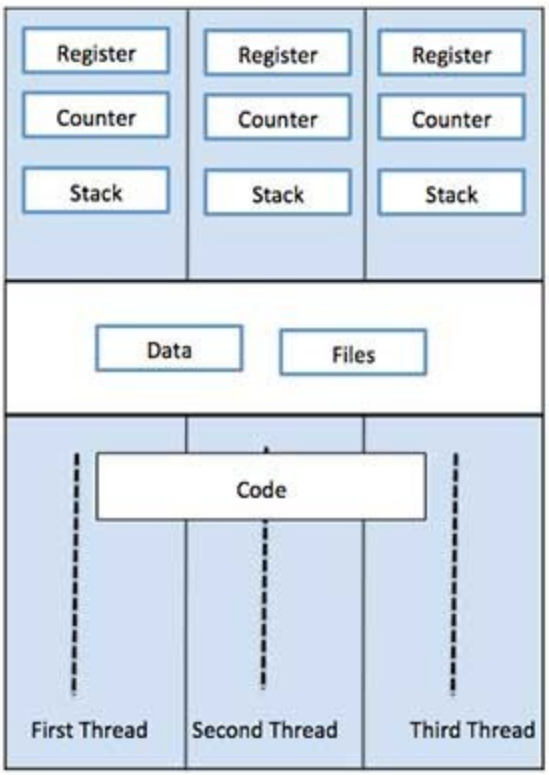
A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. The following figure shows the working of a single-threaded and a multithreaded process.



Single Process P with single thread



Single Process P with three threads

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to

		interact with operating system.
3	In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change

		another thread's data.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

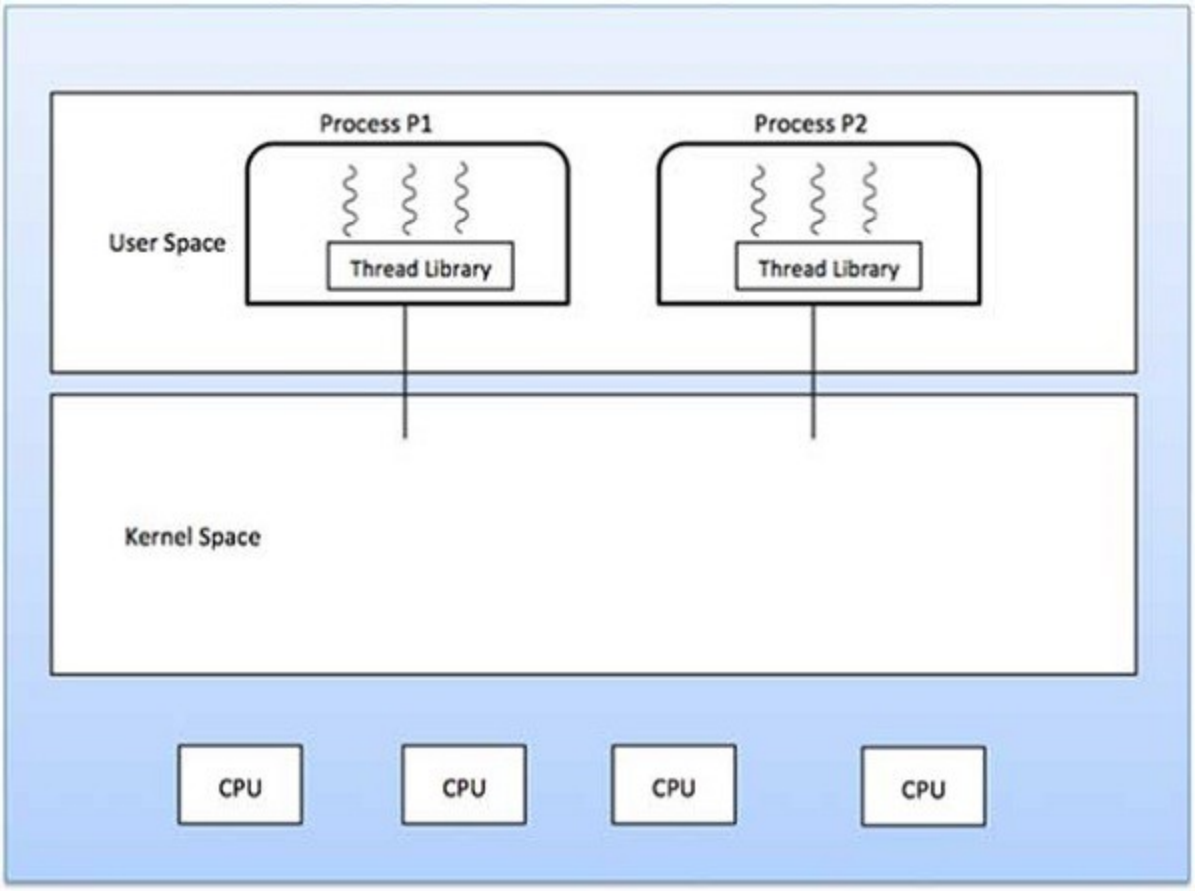
Types of Thread

Threads are implemented in following two ways −

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

User Level Threads

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



Advantages

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

Disadvantages

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

Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

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

Disadvantages

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

Multithreading Models

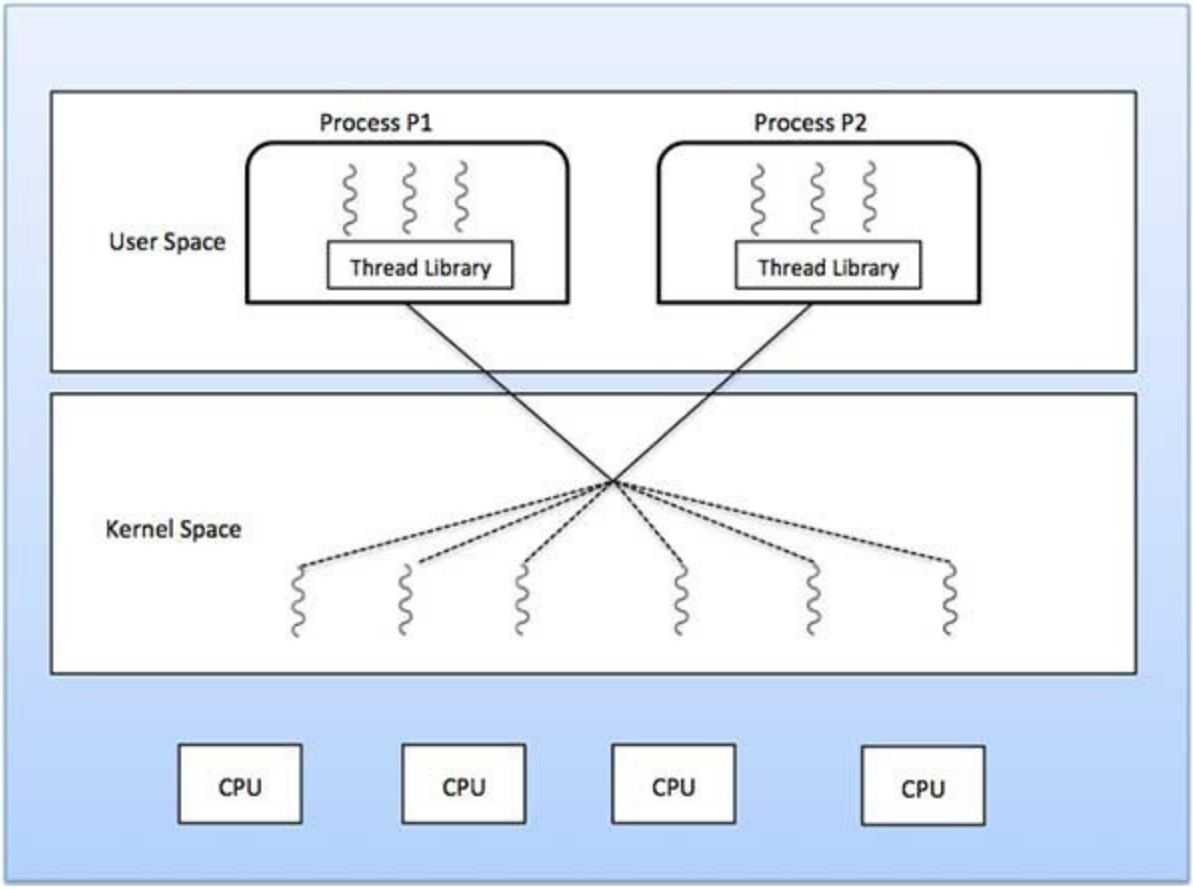
Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the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

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

Many to Many Model

The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads.

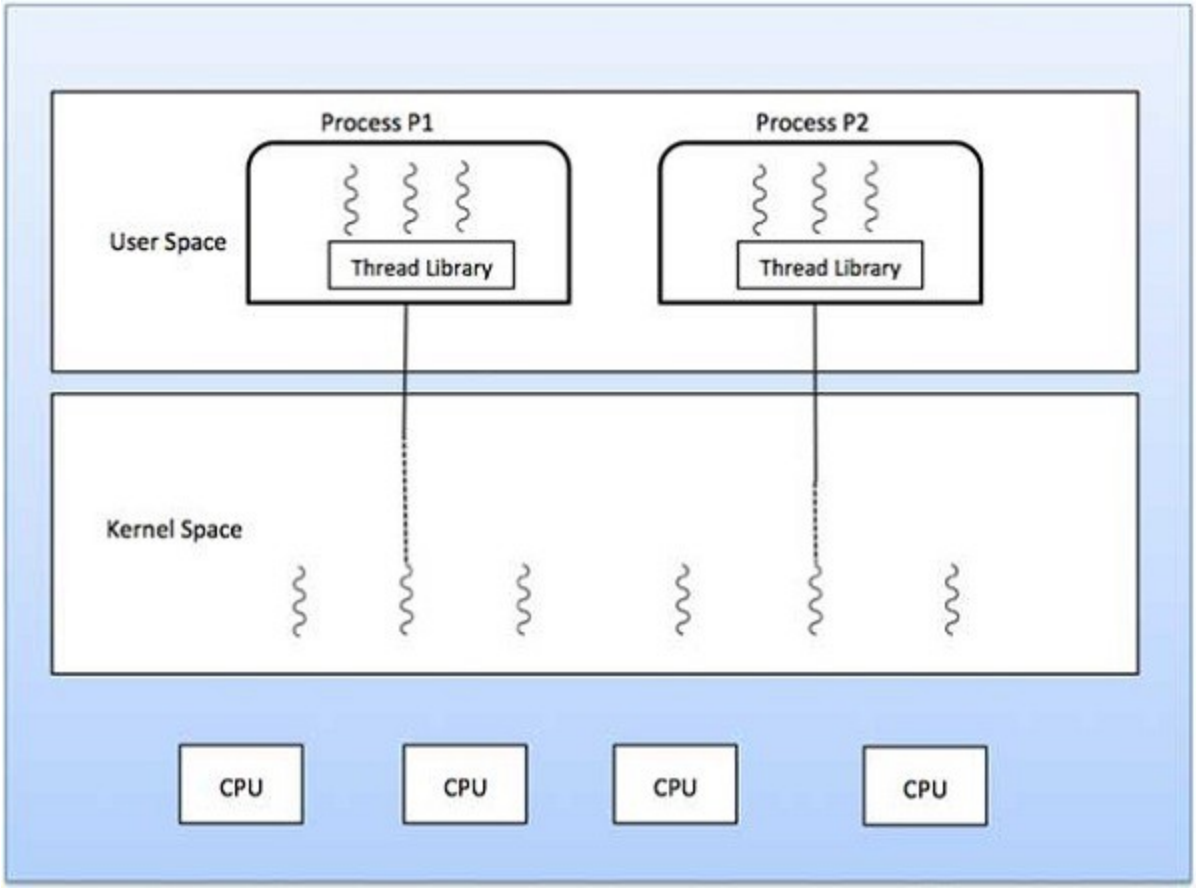
The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallel on a multiprocessor machine. This model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



Many to One Model

Many-to-one model maps many user level threads to one Kernel-level thread. Thread management is done in user space by the thread library. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

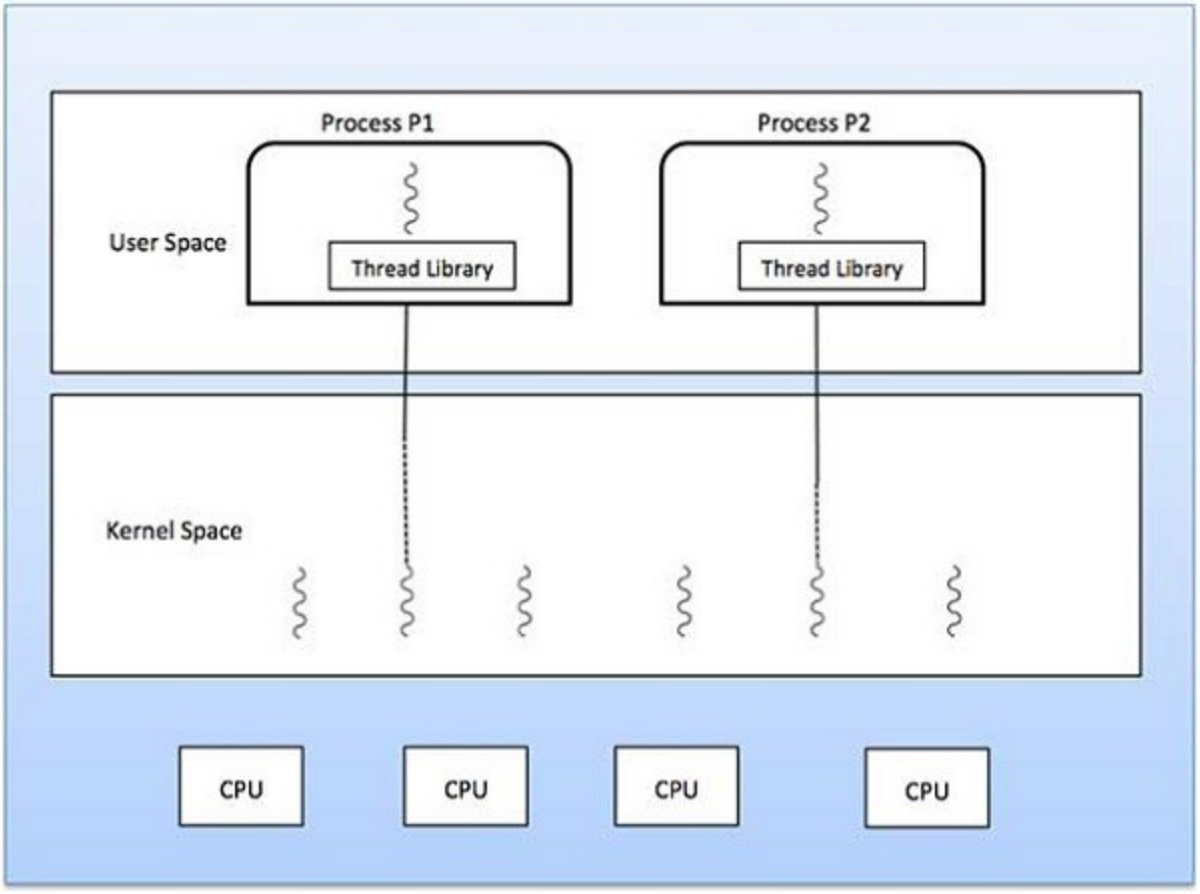
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

There is one-to-one relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Difference between User-Level & Kernel-Level Thread

S.N.	User-Level Threads	Kernel-Level Thread
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.

4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

UNIT - III

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion, Hardware Solution, Strict Alternation, Peterson's Solution, The Producer\Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

Inter-process Communication:

BASICS OF INTERPROCESS COMMUNICATION

- Information sharing: Since some users may be interested in the same piece of information (for example, a shared file), you must provide a situation for allowing concurrent access to that information.
- Computation speedup: If you want a particular work to run fast, you must break it into sub-tasks where each of them will get executed in parallel with the other tasks. Note that such a speed-up can be attained only when the computer has compound or various processing elements like CPUs or I/O channels.
- Modularity: You may want to build the system in a modular way by dividing the system functions into split processes or threads.
- Convenience: Even a single user may work on many tasks at a time. For example, a user may be editing, formatting, printing, and compiling in parallel.

Working together with multiple processes, require an interprocess communication (IPC) method which will allow them to exchange data along with various information. There are two primary models of interprocess communication:

1. shared memory and
2. message passing.

In the shared-memory model, a region of memory which is shared by cooperating processes gets established. Processes can be then able to exchange information by reading and writing all

the data to the shared region. In the message-passing form, communication takes place by way of messages exchanged among the cooperating processes.

SHARED MEMORY CONCEPTS:

Interprocess communication (IPC) usually utilizes shared memory that requires communicating processes for establishing a region of shared memory. Typically, a shared-memory region resides within the address space of any process creating the shared memory segment. Other processes that wish for communicating using this shared-memory segment must connect it to their address space.

A process can be of two type:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via shared memory method and via message passing.communication. First, we will discuss the shared memory method of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing

simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process. Let's discuss an example of communication between processes using shared memory method.

(i) Shared Memory Method

Ex: Producer-Consumer problem

There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes shares a common space or memory location known as buffer where the item produced by Producer is stored and from where the Consumer consumes the item if needed. There are two version of this problem: first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on size of buffer, the second one is known as bounded buffer problem in which producer can produce up to a certain amount of item and after that it starts waiting for consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then producer will start producing items. If the total produced item is equal to the size of buffer, producer will wait to get it consumed by the Consumer. Sim-
ilarly, the consumer first check for the availability of the item and if no item is available, Consumer will wait for producer to produce it. If there are items available, consumer will consume it. The pseudo code are given below:

Shared Data between the two Processes:

```
struct item{

    // diffrent member of the produced data
    // or consumed data
    ---------
}

// An array is needed for holding the items.
// This is the shared place which will be
// access by both process
// item shared_buff [ buff_max ];

// Two variables which will keep track of
// the indexes of the items produced by producer
// and consumer The free index points to
// the next free index. The full index points to
// the first full index.
```

```
int free_index = 0;
int full_index = 0;
```

Producer Process Code:

```
item nextProduced;

while(1){

    // check if there is no space
    // for production.
    // if so keep waiting.
    while((free_index+1) mod buff_max == full_index);

    shared_buff[free_index] = nextProduced;
    free_index = (free_index + 1) mod buff_max;
}
```

Consumer Process Code:

```
item nextConsumed;

while(1){

    // check if there is an available
    // item  for consumption.
    // if not keep on waiting for
    // get them produced.
    while((free_index == full_index);

    nextConsumed = shared_buff[full_index];
    full_index = (full_index + 1) mod buff_max;
}
```

In the above code, The producer will start producing again when the (free_index+1) mod buff max will be free because if it it not free, this implies that there are still items that can be consumed by the Consumer so there is no need to produce more. Similarly, if free index and full index points to the same index, this implies that there are no item to consume.

ii) Messaging Passing Method

Now, We will start our discussion for the communication between processes via message passing. In this method, processes communicate with each other without using any kind of of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follow:

- Establish a communication link (if a link already exists, no need to establish it again.)

- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destinaion) or **send**(message)
  – **receive**(message, host) or **receive**(message)

The message size can be of fixed size or of variable size. if it is of fixed size, it is easy for OS designer but complicated for programmer and if it is of variable size then it is easy for programmer but complicated for the OS designer. A standard message can have two parts: **header and body.**
The **header part** is used for storing Message type, destination id, source id, message length and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

Direct and Indirect Communication link
Now, We will start our discussion about the methods of implementing communication link. While implementing the link, there are some questions which need to be kept in mind like :

1. How are links established?
2. Can a link be associated with more than two processes?
3. How many links can there be between every pair of communicating processes?
4. What is the capacity of a link? Is the size of a message that the link can accommodate fixed or variable?
5. Is a link unidirectional or bi-directional?

A link has some capacity that determines the number of

messages that can reside in it temporarily for which Every link has a queue associated with it which can be either of zero capacity or of bounded capacity or of unbounded capacity. In zero capacity, sender wait until receiver inform sender that it has received the message. In non-zero capacity cases, a process does not know whether a message has been received or not after the send operation. For this, the sender must communicate to receiver explicitly. Implementation of the link depends on the situation, it can be either a Direct communication link or an In-directed communication link.

Direct Communication links are implemented when the processes use specific process identifier for the communication but it is hard to identify the sender ahead of time.
For example: the print server.

In-directed Communication is done via a shred mailbox (port), which consists of queue of messages. Sender keeps the message in mailbox and receiver picks them up.

Producer Code:

```
void Producer(void){

    int item;
```

```
    Message m;

    while(1){

       receive(Consumer, &m);
       item = produce();
       build_message(&m , item ) ;
       send(Consumer, &m);
    }
  }
```

Consumer Code:

```
void Consumer(void){

    int item;
    Message m;

    while(1){

       receive(Producer, &m);
       item = extracted_item();
       send(Producer, &m);
       consume_item(item);
    }
  }
```

Examples of IPC systems

1. Posix : uses shared memory method.
2. Mach : uses message passing
3. Windows XP : uses message passing using local procedural calls

Communication in client/server Architecture:

There are various mechanism:

- Pipe
- Socket
- Remote Procedural calls (RPCs)

The above three methods will be discussed later article as all of them are quite conceptual and deserve their own separate articles.

METHODS FOR HANDLING DEADLOCKS

We can deal with the deadlock problem in one of three ways:

- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
- We can allow the system to enter a deadlocked state, detect it, and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

DEADLOCK PREVENTION

Deadlock prevention provides a set of methods to ensure that at least one of the necessary conditions cannot hold.

Mutual Exclusion

The mutual exclusion condition must hold. That is, at least one resource must be nonsharable. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good example of a sharable resource.

Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources. One protocol that we can use requires each process to request and be allocated all its resources before it begins execution.

No Preemption

The third necessary condition for deadlocks is that there be no preemption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol. If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are preempted.

Circular Wait

The fourth and final condition for deadlocks is the circular-wait condition. One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration

DEADLOCK AVOIDANCE

A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist. The resourceallocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

Safe State

A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence. A sequence of processes
<P1, P2, ..., Pn> is a safe sequence for the current allocation state if, for each Pi , the resource requests that Pi can still make can be satisfied by the currently available resources plus the resources held by all Pj, with j < i. In this situation, if the resources that Pi needs are not immediately available, then Pi can wait until all Pj have finished. When they have finished, Pi can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When Pi terminates, Pi+1 can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe.

converted to a request edge. Similarly, when a resource Rj is released by Pi , the assignment edge Rj → Pi is reconverted to a claim edge Pi → Rj.

BANKER'S ALGORITHM

The resource-allocation-graph algorithm is not applicable to a resourceallocation system with multiple instances of each resource type. The deadlockavoidance algorithm that we describe next is applicable to such a system but is less efficient than the resource-allocation graph scheme. This algorithm is commonly known as the banker's algorithm.

DEADLOCK DETECTION

If a system does not employ either a deadlock-prevention or a deadlockavoidance algorithm, then a deadlock situation may occur. In this environment, the system may provide:

- An algorithm that examines the state of the system to determine whether a deadlock has occurred
- An algorithm to recover from the deadlock.

RECOVERY FROM DEADLOCK

When a detection algorithm determines that a deadlock exists, several alternatives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically.

Process Termination

To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.

- Abort all deadlocked processes. This method clearly will break the deadlock cycle, but at great expense. The deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.
- Abort one process at a time until the deadlock cycle is eliminated. This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

Resource-Allocation-Graph Algorithm

If we have a resource-allocation system with only one instance of each resource type, we can use a variant of the resource-allocation graph defined for deadlock avoidance. In addition to the request and assignment edges already described, we introduce a new type of edge, called a claim edge. A claim edge Pi → Rj indicates that process Pi may request resource Rj at some time in the future. This edge resembles a request edge in direction but is represented in the graph by a dashed line. When process Pi requests resource Rj , the claim edge Pi → Rj is

Resource Preemption

To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

PROTECTION

Protection refers to a mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.

GOALS OF PROTECTION

Protection can improve reliability by detecting latent errors at the interfaces between component subsystems. mechanisms are distinct from policies. Mechanisms determine how something will be done; policies decide what will be done. The separation of policy and mechanism is important for flexibility. Policies are likely to change from place to place or time to time.

PRINCIPLES OF PROTECTION

a guiding principle can be used throughout a project, such as the design of an operating system. Following this principle simplifies design decisions and keeps the system consistent and easy to understand. A key, time-tested guiding principle for protection is the principle of least privilege. It dictates that programs, users, and even systems be given just enough privileges to perform their tasks.

DOMAIN OF PROTECTION

A computer system is a collection of processes and objects. By objects, we mean both hardware objects (such as the CPU, memory segments, printers, disks, and tape drives) and software objects (such as files, programs, and semaphores). Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined and meaningful operations. Objects are essentially abstract data types.

Domain Structure

To facilitate the scheme just described, a process operates within a protection domain, which specifies the resources that the process may access. Each domain defines a set of objects and the types of operations that may be invoked on each object. The ability to execute an operation on an object is an access right.

ACCESS MATRIX

Our general model of protection can be viewed abstractly as a matrix, called an access matrix. The rows of the access matrix represent domains, and the columns represent objects. Each entry in the matrix consists of a set of access rights. Because the column defines objects explicitly, we can omit the object name from the access right. The entry access(i,j) defines the set of operations that a process executing in domain Di can invoke on object Oj

IMPLEMENTATION OF THE ACCESS MATRIX

Global Table

The simplest implementation of the access matrix is a global table consisting of a set of ordered triples <domain, object, rights-set>. Whenever an operation M is executed on an object Oj within domain Di , the global table
is searched for a triple <Di , Oj , Rk>, with M ∈ Rk . If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.

Access Lists for Objects

Each column in the access matrix can be implemented as an access list for one object, as described in Obviously, the empty entries can be discarded. The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a nonempty set of access rights for that object.

Capability Lists for Domains

Rather than associating the columns of the access matrix with the objects as access lists, we can associate each row with its domain. A capability list for a domain is a list of objects together with the operations allowed on those objects. An object is often represented by its physical name or address, called a capability. To execute operation M on object Oj , the process executes the operation M, specifying the capability (or pointer) for object Oj as a parameter. Simple possession of the capability means that access is allowed.

ACCESS CONTROL

Each file and directory is assigned an owner, a group, or possibly a list of users, and for each of those entities, access-control information is assigned. A similar function can be

added to other aspects of a computer system. A good example of this is found in Solaris 10. Solaris 10 advances the protection available in the operating system by explicitly adding the principle of least privilege via role-based access control (RBAC).

REVOCATION OF ACCESS RIGHTS

In a dynamic protection system, we may sometimes need to revoke access rights to objects shared by different users. Various questions about revocation may arise:

• Immediate versus delayed. Does revocation occur immediately, or is it delayed? If revocation is delayed, can we find out when it will take place?

• Selective versus general. When an access right to an object is revoked, does it affect all the users who have an access right to that object, or can we specify a select group of users whose access rights should be revoked?

• Partial versus total. Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?

• Temporary versus permanent. Can access be revoked permanently (that is, the revoked access right will never again be available), or can access be revoked and later be obtained again?

Capability-Based Systems

In this section, we survey two capability-based protection systems. These systems differ in their complexity and in the types of policies that can be implemented on them.

An Example: Hydra

Hydra is a capability-based protection system that provides considerable flexibility. The system implements a fixed set of possible access rights, including such basic forms of access as the right to read, write, or execute a memory segment. In addition, a user (of the protection system) can declare other rights. The interpretation of user-defined rights is performed solely by the user's program, but the system provides access protection for the use of these rights, as well as for the use of system-defined rights. These facilities constitute a significant development in protection technology

An Example: Cambridge CAP System

A different approach to capability-based protection has been taken in the design of the Cambridge CAP system. CAP's capability system is simpler and superficially less powerful than that of Hydra. However, closer examination shows that it, too, can be used to provide secure protection of user-defined objects. CAP has two kinds of capabilities. The ordinary kind is called a data capability. It can be used to provide access to objects, but the only rights provided are the standard read, write, and execute of the individual storage segments associated with the object. Data capabilities are interpreted by microcode in the CAP machine.

The second kind of capability is the so-called software capability, which is protected, but not interpreted, by the CAP microcode. It is interpreted by a protected (that is, privileged) procedure, which may be written by an application programmer as part of a subsystem.

LANGUAGE-BASED PROTECTION

The designers of protection systems have drawn heavily on ideas that originated in programming languages and especially on the concepts of abstract data types and objects

Compiler-Based Enforcement

A variety of techniques can be provided by a programming-language implementation to enforce protection, but any of these must depend on some degree of support from an underlying machine and its operating system. For example, suppose a language is used to generate code to run on the Cambridge CAP system. On this system, every storage reference made on the underlying hardware occurs indirectly through a capability. This restriction prevents any process from accessing a resource outside of its protection environment at any time.

Protection in Java

Because Java was designed to run in a distributed environment, the Java virtual

machine—or JVM—has many built- in protection mechanisms. Java programs are composed of classes, each of which is a collection of data fields and functions (called methods) that operate on those fields. The JVM loads a class in response to a request to create instances (or objects) of that class. One of the most novel and useful features of Java is its support for dynamically loading untrusted classes over a network and for executing mutually distrusting classes within the same JVM.

UNIT - IV

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and External fragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, Protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU).

MEMORY MANAGEMENT:

In general, to rum a program, it must be brought into memory.
Input queue – collection of processes on the disk that are waiting to be brought into memory to run the program. User programs go through several steps before being run
Address binding: Mapping of instructions and data from one address to another address in memory. Three different stages of binding:
Compile time: Must generate absolute code if memory location is known in prior.
Load time: Must generate relocatable code if memory location is not known at compile time Execution time: Need hardware support for address maps (e.g., base and limit registers).
Logical vs. Physical Address Space

Logical address – generated by the CPU; also referred to as

“virtual address“ Physical address – address seen by the

memory unit.

Logical and physical addresses are the same in compile-time and load-time

address-binding schemes" Logical (virtual) and physical addresses differ in

execution-time address- binding

scheme"

Memory-Management Unit (MMU)

It is a hardware device that maps virtual / Logical address to physical address

In this scheme, the relocation register‘s value is added to Logical address

generated by a user process. The user program deals with logical addresses; it

never sees the real physical addresses

Logical address range: 0 to max

Physical address range: R+0 to R+max, where R—value in relocation register.

Dynamic Loading

Through this, the routine is not loaded until it is called.
Better memory-space utilization; unused routine is never loaded

Useful when large amounts of code are needed to handle infrequently occurring cases

No special support from the operating system is required implemented
through program design Dynamic Linking

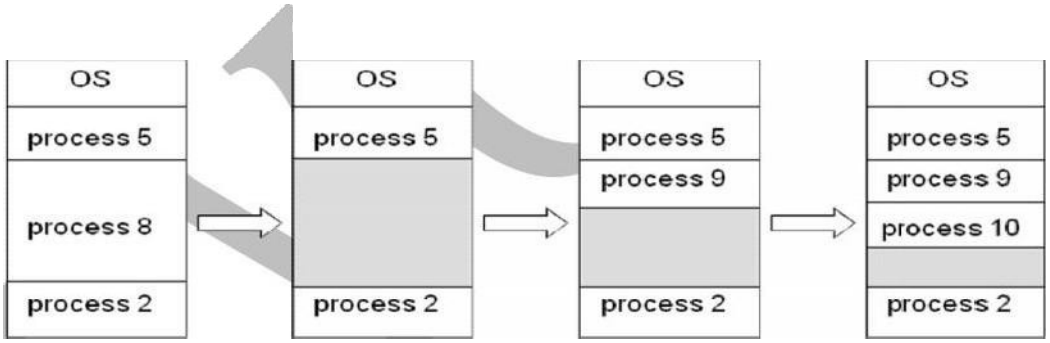
Linking postponed until execution time & is particularly useful for libraries
Small piece of code called stub, used to locate the appropriate memory- resident library routine or function. Stub replaces itself with the address of the routine, and executes the routine
Operating system needed to check if routine is in processes’ memory address
Shared libraries: Programs linked before the new library was installed will continue

using the older library. Swapping

A process can be swapped temporarily out of memory to a backing store (SWAP OUT)and then brought backinto memory for continued execution (SWAP IN).
Backing store – fast disk large enough to accommodate copies of all memory images for all users & it must provide direct access to these memory images
Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
Transfer time: Major part of swap time is transfer time. Total transfer time is directly proportional to the amount of memory swapped.
Example: Let us assume the user process is of size 1MB & the backing store is a standard hard disk with a transfer rate of 5MBPS.
Transfer time = 1000KB/5000KB per second
= 1/5 sec = 200ms

Contiguous Allocation
Each process is contained in a single contiguous
section of memory. There are two methods namely:

Fixed – Partition
Method Variable
–
Partition
Method Fixed –
Partition
Method :
Divide memory into fixed size partitions, where each partition has exactly one process.
The drawback is memory space unused within a partition is
wasted.(eg.when process size < partition size)
Variable-partition method:
Divide memory into variable size partitions, depending upon the size of
the incoming process. When a process terminates, the partition becomes
available for another process.



As processes complete and leave they create holes in the main memory.

Hole – block of available memory; holes of various size are scattered throughout memory.

Dynamic Storage- Allocation Problem:

How to satisfy a request of size n‘ from a list

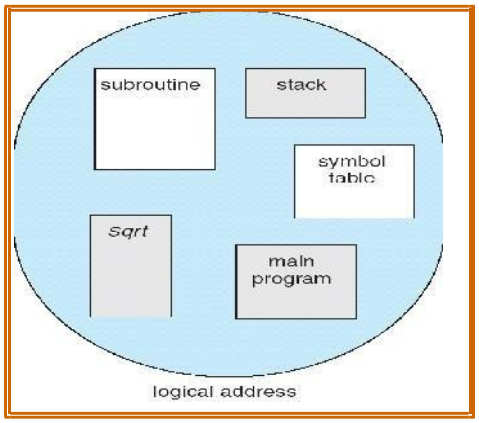
of free holes? Solution:

First-fit: Allocate the first hole that is big enough.
Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
Worst-fit: Allocate the largest hole; must also search entire list. Produces
the largest leftover hole.

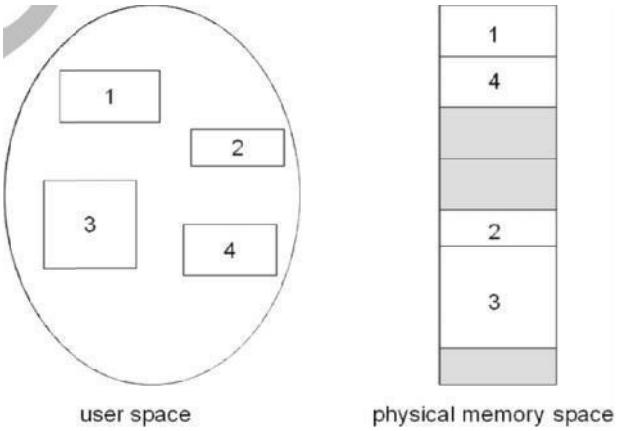
NOTE: First-fit and best-fit are better than worst-fit in terms of speed
and storage utilization Fragmentation:
o External Fragmentation – This takes place when enough total memory space exists to satisfy a request, but it is not contiguous i.e, storage is fragmented into a large number of small holes scattered throughout the main memory.

o Internal Fragmentation – Allocated memory may be slightly larger than requested memory. Example: hole = 184 bytes Process size = 182 bytes.
We are left with a hole of 2 bytes. o Solutions

Coalescing: Merge the adjacent holes together.
Compaction: Move all processes towards one end of memory, hole towards other end of memory, producing one large hole of available memory. This scheme is expensive as it can be done if relocation is dynamic and done at execution time.
Permit the logical address space of a process to be non-contiguous. This is achieved through two memory management schemes namely paging and segmentation.
Segmentation
Memory-management scheme that supports user view of memory
A program is a collection of segments. A segment is a logical unit such as:Main program, Procedure, Function, Method, Object, Local variables, global variables, Common block, Stack,



Symbol table, arrays Logical View of Segmentation



Segmentation Hardware

- Logical address consists of a two tuple : <Segment-number, offset>
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **Base** – contains the starting physical address where the segments reside in memory
  - **Limit** – specifies the length of the segment
- ***Segment-table base register (STBR)*** points to the segment table's location in memory
- ***Segment-table length register (STLR)*** indicates number of segments used by a program; Segment number=s' is legal, if *s* < STLR
- **Relocation**.
  - dynamic
  - by segment table
- **Sharing**.
  - shared segments
  - same segment number
- **Allocation**.
  - first fit/best fit
  - external fragmentation
- **Protection:** With each entry in segment table associate:
  - validation bit = 0 □ illegalsegment
  - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage- allocation problem
- A segmentation example is shown in the following diagram

EXAMPLE:

- Another advantage of segmentation involves the sharing of code or data.

- Each process has a segment table associated with it, which the dispatcher uses to define the hardware segment table when this process is given the CPU.

Segments are shared when entries in the segment tables of two different processes point to the same physical location.

Segmentation with paging

The IBM OS/ 2.32 bit version is an operating system running on top of the Intel 386 architecture. The 386 uses segmentation with paging for memory management. The

maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes.

- The local-address space of a process is divided into two partitions.
  - The first partition consists of up to 8 KB segments that are private to that process.
  - The second partition consists of up to 8KB segments that are shared among all the processes.
  - Information about the first partition is kept in the **local descriptor table (LDT)**, information about the second partition is kept in the **global descriptor table (GDT)**.

Each entry in the LDT and GDT consist of 8 bytes, with detailed information about a particular segment including the base location and length of the segment.

The logical address is a pair (selector, offset) where the selector is a16-bit number:

Where s designates the segment number, g indicates whether the segment is in the GDT or LDT, and p deals with protection. The offset is a 32-bit number specifying the location of the byte within the segment in question**.**

- The base and limit information about the segment in question are used to generate a linear- address.
- First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system. If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.
- The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. Since we page the page table, the page number is further divided into a 10-bit page directory pointer and a 10-bit
- To improve the efficiency of physical memory use. Intel 386 page tables can be swapped to disk. Inthis case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or on disk.

If the table is on disk, the operating system can use the other 31 bits to specify the disk location of thetable; the table then can be brought into memory on demand.

Paging

- It is a memory management scheme that permits the physical address space of a process to be noncontiguous.
- It avoids the considerable problem of fitting the varying size memory chunks on to the backing store.

(i) Basic Method:

Divide logical memory into blocks of same size called **"pages"**. o Divide physical memory into fixed-sized blocks called **"frames"** o Page size is a power of 2, between 512 bytes and 16MB.

Address Translation Scheme

o Address generated by CPU(logical address) is divided into**:**

- □ **Page number *(p)*** – used as an index into a page table which contains base address of each page in physical memory
- □ **Page offset *(d)*** –combined with base address to define the physical address i.e.,

Physical address = base address + offset

Demand Paging

- o It is similar to a paging system with swapping.
- o Demand Paging - Bring a page into memory only when it is needed
- o To execute a process, swap that entire process into memory. Rather than swapping theentireprocessintomemoryhowever, we use Lazy Swapper"
- o **Lazy Swapper** - Never swaps a page into memory unless that page will be needed.
- o **Advantages**
  - □ Less I/O needed

- Less memory needed
- Faster response
- More users

Page Replacement

- If no frames are free, we could find one that is not currently being used & free it.
- We can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory.
- Then we can use that freed frame to hold the page for which the process faulted.

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame

- If there is a free frame , then use it.
- If there is no free frame, use a page replacement algorithm to select a **victim** frame

Write the victim page to the disk, change the page & frame tables accordingly

1. Read the desired page into the (new) free frame. Update the page and frame tables.

UNIT-5

I/O Hardware: I/O devices, Device controllers, Direct memory access Principles of I/O Software: Goals of Interrupt handlers, Device drivers, Device independent I/O software, Secondary-Storage Structure: Disk structure, Disk scheduling algorithms

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), Free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance.

Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks.

File Concept:

A file is a named collection of related information that is recorded on secondary storage.

From a user's perspective, a file is the smallest allotment of logical secondary

storage; that **is, data cannot be written to secondary storage unless they are within a file.**

Examples of files:

- A text file is a sequence of characters organized into lines (and possibly pages). A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements. An object file is a sequence of bytes organized into blocks understandable by the system's linker.

- **File Attributes**
- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number identifies the file within the file system. It is the non-human readable name for the file.
- **Type:** This information is needed for those systems that support different types.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words or blocks)and possibly the maximum allowed sizeare included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time, date and user identification:** This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

File Operations

- Creating a file
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file

Access Methods

1. Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

- The bulk of the operations on a file is reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file). Such a file can be reset to the beginning and, on some systems, a program may be able to skip forward or back ward n records, for some integer n-perhaps only for n=1. Sequential access is based on a tape model of a file, and works as well on sequential-access devices as it does on random – access ones.

2. Direct Access

- Another method is direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. The direct- access methods is based on a disk model of a file, since disks allow random access to any file block.
- For direct access, the file is viewed as a numbered sequence of blocks or records. A direct-access file allows arbitrary blocks to be read or written. Thus, we may read block 14, then read block 53, and then write block7. There are no restrictions on the order of reading or writing for a direct-access file.
-

Direct – access files are of great use for immediate access to large amounts of information. Database is often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer, and then read that block directly to provide the desired information.

Directory and Disk Structure

<u>There are five directory structures. They are</u>

1. Single-level directory
2. Two-level directory
3. Tree-Structured directory

4. Acyclic Graph directory
5. General Graph directory

1. Single – Level Directory

- The simplest directory structure is the single- level directory.
- All files are contained in the same directory.

- **Disadvantage:**
  - ➢ When the number of files increases or when the system has more than one user, since all filesare in the same directory, they must have unique names.

2. Two – Level Directory

- In the two level directory structures, each user has her own user file directory (UFD).
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFDis

indexed by user name or account number, and each entry points to the UFD for that user.

- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name.
- Although the two – level directory structure solves the name-collision problem

Disadvantage:

➢ Users cannot create their own sub-directories.

3. Tree – Structured Directory

- A tree is the most common directory structure.
- The tree has a root directory. Every file in the system has a unique path name.
- A **path name** is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- A directory is simply another file. But it is treated in a special way.
- All directories have the same internal format.
- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- Path names can be of two types: absolute path names or relative path names.
- An absolute path name begins at the root and follows a path down to the specified file, giving the directory

Names on the path:

1. Acyclic Graph Directory.

- An acyclic graph is a graph with no cycles.
- To implement shared files and subdirectories this directory structure is used.
- An acyclic – graph directory structure is more flexible than is a simple tree structure, but it is also more complex. In a system where sharing is implemented by symbolic link, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed.
- Another approach to deletion is to preserve the file until all references to it are deleted. To implements **approach, we must have some mechanism for determining that the last reference to the file has been deleted.**

1. Multiple Users:

- When an operating system accommodates multiple users, the issues of file sharing, file naming and file

protection become preeminent.

- The system either can allow user to access the file of other users by default, or it may require that a user specifically grant access to the files.
  - These are the issues of access control and protection.
  - To implementing sharing and protection, the system must maintain more file and directory attributes than a on a single-user system.
  - The owner is the user who may change attributes, grand access, and has the most control over the file or

directory.

- The group attribute of a file is used to define a subset of users who may share access to the file.
- Most systems implement owner attributes by managing a list of user names and associated user identifiers

(user Ids).

- When a user logs in to the system, the authentication stage determines the appropriate user ID for the user. That user ID is associated with all of user's processes and threads. When they need to be user readable, they are translated, back to the user name via the user name list. Likewise, group functionality can be implemented as a system wide list of group names and group identifiers.
- Every user can be in one or more groups, depending upon operating system design decisions. The user's**group Ids is also included in every associated process and thread.**

2. Remote File System:

- Networks allowed communications between remote computers.
- Networking allows the sharing or resource spread within a campus or even around the world.
- User manually transfer files between machines via programs like **ftp**.
- A **distributed file system** (DFS) in which remote directories is visible from the local machine.
- The **World Wide Web**: A browser is needed to gain access to the remote file and separate operations (essentially a wrapper for ftp) are used to transfer files.

a) The client-server Model:

- Remote file systems allow a computer to a mount one or more file systems from one ormore remote machines.
- A server can serve multiple clients, and a client can use multiple servers, depending onthe

implementation details of a given client –server facility.

- Client identification is more difficult. Clients can be specified by their network name or other identifier, such as IP address, but these can be spoofed (or imitate). An unauthorized client can spoof the server into deciding that it is authorized, and the unauthorized client could be allowed access.

b) Distributed Information systems:

- Distributed information systems, also known as distributed naming service, have been devised toprovide a unified access to the information needed for remote computing.
- Domain name system (DNS) provides host-name-to-network address translations for theirentire Internet (including the World Wide Web).
- Before DNS was invented and became widespread, files containing the same information were sent

via e-mail of ftp between all networked hosts.

c) Failure Modes:

- **Redundant arrays of inexpensive disks** (**RAID**) can prevent the loss of a disk from resulting in the loss of data.
- Remote file system has more failure modes. By nature of the complexity of networking system and the required interactions between remote machines, many more problems can interfere with the proper operation of remote file systems.

d)

e) Consistency Semantics:

- It is characterization of the system that specifies the semantics of multiple users accessing a shared file

simultaneously.

- These semantics should specify when modifications of data by one user are observable by other users.
- The semantics are typically implemented as code with the file system.
- A series of file accesses (that is reads and writes) attempted by a user to the same file is always enclosed between the open and close operations.
- The series of access between the open and close operations is a **file session**.

(i) UNIX Semantics:

The UNIX file system uses the following consistency semantics:

1. Writes to an open file by a user are visible immediately to other users that have this file open at thesame time.
2. One mode of sharing allows users to share the pointer of current location into the file. Thus, the advancing of the pointer by one user affects all sharing users.

(ii) Session Semantics:

The Andrew file system (AFS) uses the following consistency semantics:

1. Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously.
2. Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect this change.

(iii) Immutable –shared File Semantics:

- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has two key properties:

 Its name may not be reused and its contents may not be altered.

File Protection

(i) Need for file protection.

- When information is kept in a computer system, we want to keep it safe from **physical damage**

(reliability) and **improper access** (protection).

- Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or though computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or

month) to maintain a copy should a file system be accidentally destroyed.

• File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs in the file-system software can also cause file contents to be lost.

• Protection can be provided in many ways. For a small single-user system, we might provide protectionby physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multi-user system, however, other mechanisms are needed.

(ii) Types of Access

• Complete protection is provided by prohibiting access.

• Free access is provided with no protection.

• Both approaches are too extreme for general use.

• What is needed is **controlled access**.

• Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

1. **Read:** Read from the file.
2. **Write:** Write or rewrite the file.
3. **Execute:** Load the file into memory and execute it.
4. **Append:** Write new information at the end of the file.
5. **Delete:** Delete the file and free its space for possible reuse.
6. **List:** List the name and attributes of the file.

(iii) Access Control

• Associate with each file and directory an access-control list (ACL) specifying the user name and thetypes of access allowed for each user.

• When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs and the user job is denied access to the file.

• This technique has two undesirable consequences:

 Constructingsuchalistmaybeatediousandunrewardingtask,especiallyifwedonotknowin advance.the list of users in the system.

 Thedirectoryentry,previouslyoffixedsize,nowneedstobeofvariablesize,resultinginmore complicated space management.

- To condense the length of the access control list, many systems recognize three classifications of users in connection with each file:

  - ➢ **Owner:** The user who created the file is the owner.
  - ➢ **Group:** A set of users who are sharing the file and need similar access is a group, or work group.
  - ➢ Universe: All other users in the system constitute the universe.

File System Implementation- File System Structure

- Disk provide the bulk of secondary storage on which a file system is maintained.
- Characteristics of a disk:

1. They can be rewritten in place, it is possible to read a block from the disk, to modify the block and towrite it back into the same place.
2. They can access directly any given block of information to the disk.

- To produce an efficient and convenient access to the disk, the operating system imposes one or more file system to allow the data to be stored, located and retrieved easily.
- The file system itself is generally composed of many different levels. Each level in the design uses the features of lower level to create new features for use by higher levels.

Layered File System

- The I/O control consists of device drivers and interrupt handlers to transfer information between themain memory and the disk system .
- The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (forexample, drive –1, cylinder 73, track 2, sector 10)

Directory Implementation

1. Linear List

- The simplest method of implementing a directory is to use a linear list of file names with pointer to the data

blocks.

- A linear list of directory entries requires a linear search to find a particular entry.
- This method is simple to program but time- consuming to execute. To create a new file, we must first search the but time – consuming to execute.

- The real disadvantage of a linear list of directory entries is the linear search to find a file.

2. Hash Table

- In this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in thelinear

list.

- Therefore, it can greatly decrease the directory search time.
- Insertion and deleting are also fairly straight forward, although some provision must be made for collisions.

situation where two file names hash to the same location.

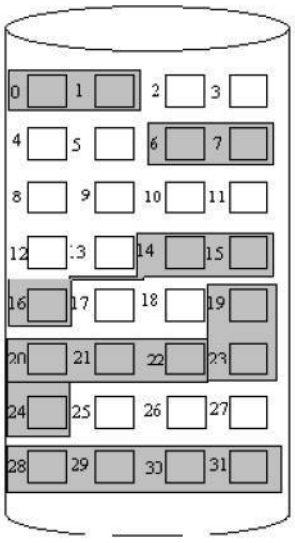
- The major difficulties with a hash table are its generally fixed size and the dependence ofthe hash function on that size.

Allocation Methods

- The main problem is how to allocate space to these files so that disk space is utilized effectively and files.can be accessed quickly.
  - There are three major methods of allocating disk space:
    1. Contiguous Allocation
    2. Linked Allocation
    3. Indexed Allocation

1. Contiguous Allocation

- The contiguous – allocation method requires each file to occupy a set of contiguous blocks on the disk.



Directory

file	start	length
Count	9	2
tr	14	3
mail	19	5
list	28	4
f	6	2

- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b,. b+1, b+2,….,b+n-1.
- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

Disadvantages:

1. Finding space for a new file.

- The contiguous disk space-allocation problem suffer from the problem of external fragmentation. As file are allocated and deleted, the free disk space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, no one of which is large enough to store the data.

2. Determining how much space is needed for a file.

When the file is created, the total amount of space it will need must be found an allocated how does.

creator know the size of the file to be created?

- If we allocate too little space to a file, we may find that file cannot be extended. The other possibility is to find a larger hole, copy the contents of the file to the new space, and release the previous space. This series of actions may be repeated as long as space exists, although it can be time – consuming. However, in this case, the user never needs to be informed explicitly about what is happening ; the system continues despite the problem, although more and more slowly.
- Even if the total amount of space needed for a file is known in advance pre-allocation may be inefficient.
- A file that grows slowly over a long period (months or years) must be allocated enough space for its final size, even though much of that space may be unused for a long time the file, therefore has a large amount of internal fragmentation.

To overcome these disadvantages:

- Use a modified contiguous allocation scheme, in which a contiguous chunk of space called as an **extent** is allocated initially and then, when that amount is not large enough another chunk of contiguous space an extent is added to the initial allocation.
- Internal fragmentation can still be a problem if the extents are too large, and external fragmentation can be a problem as extents of varying sizes are allocated and deallocated.

2. Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered any where on the disk.
- The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks.might start at block 9, continue at block 16, then block 1, block 10, and finally bock 25.

- Each block contains a pointer to the next block. These pointers are not made available to the user.
- There is no external fragmentation with linked allocation, and any free block on the free space list can beused to satisfy a request.
- The size of a file does not need to the declared when that file is created. A file can continue to grow as long as free blocks are available consequently, it is never necessary to compacts disk