T.Y.B.C.A. SCIENCE : SEMESTER-V

NEW SYLLABUS
CBCS PATTERN

PRINCIPLES OF OPERATING SYSTEMS

Dr. Ms. MANISHA BHARAMBE
Mrs. VEENA GANDHI

NIRALI PRAKASHAN
MANAGEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

Principles of Operating Systems

For B.C.A. (Science) : Semester - V

[Course Code BCA-353 : Credits-04]

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Dr. Ms. Manisha Bharambe
M.Sc (Comp. Sci.), M.Phil, Ph.D (Comp. Sci.)
Vice Principal and BCA Coordinator
MES Abasaheb Garware College
Pune 4

Mrs. Veena Gandhi
M.C.S., M.Phil (Comp. Sci.), UGC-NET
Head, Department of BCA(Science)
Abeda Inamdar Senior College
Pune 1

Price ₹ 265.00

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N5873

Principles of Operating System **ISBN 978-93-5451-097-7**

First Edition : July 2021

© : Authors

The text of this publication, or any part thereof, should not be reproduced or transmitted in any form or stored in any computer storage system or device for distribution including photocopy, recording, taping or information retrieval system or reproduced on any disc, tape, perforated media or other information storage device etc., without the written permission of Authors with whom the rights are reserved. Breach of this condition is liable for legal action.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may have crept in. Any mistake, error or discrepancy so noted and shall be brought to our notice shall be taken care of in the next edition. It is notified that neither the publisher nor the authors or seller shall be responsible for any damage or loss of action to any one, of any kind, in any manner, therefrom.

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39, Fax - (020) 25511379
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

➢ DISTRIBUTION CENTRES

PUNE

Nirali Prakashan (For orders within Pune) : 119, Budhwar Peth, Jogeshwari Mandir Lane, Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

Nirali Prakashan (For orders outside Pune) : S. No. 28/27, Dhayari, Near Asian College Pune 411041
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

MUMBAI

Nirali Prakashan : 385, S.V.P. Road, Rasdhara Co-op. Hsg. Society Ltd.,
Girgaum, Mumbai 400004, Maharashtra; Mobile : 9320129587
Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

➢ DISTRIBUTION BRANCHES

JALGAON

Nirali Prakashan : 34, V. V. Golani Market, Navi Peth, Jalgaon 425001, Maharashtra,
Tel : (0257) 222 0395, Mob : 94234 91860; Email : niralijalgaon@pragationline.com

KOLHAPUR

Nirali Prakashan : New Mahadvar Road, Kedar Plaza, 1st Floor Opp. IDBI Bank, Kolhapur 416 012
Maharashtra. Mob : 9850046155; Email : niralikolhapur@pragationline.com

NAGPUR

Nirali Prakashan : Above Maratha Mandir, Shop No. 3, First Floor,
Rani Jhanshi Square, Sitabuldi, Nagpur 440012, Maharashtra
Tel : (0712) 254 7129; Email : niralinagpur@pragationline.com

DELHI

Nirali Prakashan : 4593/15, Basement, Agarwal Lane, Ansari Road, Daryaganj
Near Times of India Building, New Delhi 110002 Mob : 08505972553
Email : niralidelhi@pragationline.com

BENGALURU

Nirali Prakashan : Maitri Ground Floor, Jaya Apartments, No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003, Karnataka; Mob : 9449043034
Email: niralibangalore@pragationline.com

Other Branches : Hyderabad, Chennai

Note : Every possible effort has been made to avoid errors or omissions in this book. In spite this, errors may have crept in. Any type of error or mistake so noted, and shall be brought to our notice, shall be taken care of in the next edition. It is notified that neither the publisher, nor the author or book seller shall be responsible for any damage or loss of action to any one of any kind, in any manner, therefrom. The reader must cross check all the facts and contents with original Government notification or publications.

niralipune@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks

Preface ...

We take an opportunity to present this book entitled as **"Principles of Operating Systems"** to the students of **B.C.A.(Science), Semester - V** as per the New Syllabus, (CBCS Pattern 2019).

The book covers theory of Process Scheduling and Multithreaded Programming, Process Synchronization, Deadlocks, Memory Management, File System, Disk Scheduling.

A special words of thank to Shri. Dineshbhai Furia, Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takale, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

Authors

■■■

Syllabus ...

1. **Process Scheduling and Multithreaded Programming** **[14 Hrs]**
   - 1.1 Process Scheduling – Scheduling Queues, Schedulers, Context Switch
   - 1.2 Operations on Process – Process Creation with Program Using fork(), Process Termination
   - 1.3 Interprocess Communication – Shared Memory System, Message Passing Systems
   - 1.4 Multithreaded Programming – Overview, Multithreading Models
   - 1.5 Basic Concept – CPU-I/O Burst Cycle, CPU Scheduler, Pre-emptive Scheduling, Dispatcher
   - 1.6 Scheduling Criteria
   - 1.7 Scheduling Algorithms – FCFS, SJF, Priority Scheduling, Round Robin Scheduling, Multiple Queue Scheduling, Multilevel Feedback Queue Scheduling
   - 1.8 Thread Scheduling
2. **Process Synchronization** **[08 Hrs]**
   - 2.1 Background
   - 2.2 Critical Section Problem
   - 2.3 Semaphores: Usage, Implementation
   - 2.4 Classic Problems of Synchronization – The Bounded Buffer Problem, The Reader Writer Problem, The Dining Philosopher Problem
3. **Deadlocks** **[10 Hrs]**
   - 3.1 System Model
   - 3.2 Deadlock Characterization – Necessary Conditions, Resource Allocation Graph
   - 3.3 Deadlock Prevention
   - 3.4 Deadlock Avoidance – Safe state, Resource-Allocation-Graph Algorithm, Banker's Algorithm
   - 3.5 Deadlock Detection
   - 3.6 Recovery from Deadlock – Process Termination, Resource Pre-emption
4. **Memory Management** **[12 Hrs]**
   - 4.1 Background – Basic Hardware, Address Binding, Logical Versus Physical Address Space, Dynamic Loading, Dynamic Linking and Shared Libraries, Overlays
   - 4.2 Swapping
   - 4.3 Contiguous Memory Allocation – Memory Mapping and Protection, Memory Allocation, Fragmentation
   - 4.4 Paging – Basic Method, Hardware support, Protection, Shared Pages
   - 4.5 Segmentation – Basic Concept, Hardware
   - 4.6 Virtual Memory Management – Demand Paging, Performance of Demand Paging, Page Replacement – FIFO, Optimal, LRU, Second Chance Algorithm
   - 4.7 Thrashing – Cause of Thrashing, Working-Set Model

5. File System **[10 Hrs]**

5.1 File Concept, File Attribute, File Operations, File Types, File Structure

5.2 Access Methods - Sequential Access Method, Direct Access Method, Other Access Methods

5.3 Directory Overview, Single Level Directory, Two Level Directory, Tree Structure Directory, Acyclic Graph Directory, General Graph Directory

5.4 File System Structure and Implementation - Partitions and Mounting, Virtual File Systems

5.5 Allocation Methods - Contiguous Allocation, Linked Allocation, Indexed Allocation

5.6 Free Space Management – Bit Vector, Linked list, Grouping, Counting, Space Map

6. Disk Scheduling **[06 Hrs]**

6.1 Overview

6.2 Disk Structure

6.3 Disk Scheduling, FCFS Scheduling, SSTF Scheduling, SCAN Scheduling, C-SCAN Scheduling, LOOK Scheduling

6.4 Disk Management

■■■

Contents ...

1.	Process Scheduling and Multithreaded Programming	1.1 – 1.68
2.	Process Synchronization	2.1 – 2.22
3.	Deadlocks	3.1 – 3.44
4.	Memory Management	4.1 – 4.62
5.	File System	5.1 – 5.32
6.	Disk Scheduling	6.1 – 6.26

■■■

1... Process Scheduling and Multithreaded Programming

Learning Objectives ...

- To study importance of Process Scheduling and different types of Schedulers.
- To learn various operations on Processes.
- To understand methods of Inter Process Communication.
- To understand the benefits of Multithreaded Programming.
- To learn the concept of Multi-core Programming.
- To study different Multithreaded Models.
- To understand concept of CPU Scheduling.
- To learn evaluation criteria for selecting a CPU-Scheduling Algorithm for a particular System.
- To study various CPU Scheduling Algorithms like FCFS, SJF, RR, etc.

1.1 INTRODUCTION [W-18]

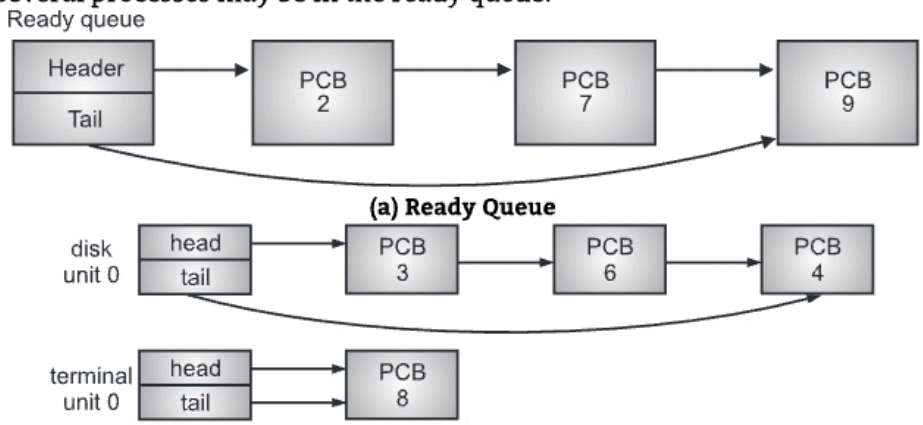
- We know that a process is program in execution. A process is defined as "an entity which represents the basic unit of work to be implemented in the system".
- The current activity of a process is known as its state. As a process executes, it changes state. The process state is an indicator of the nature of the current activity in a process. The process can be any one of states like New (Dormant), Ready, Active, Waiting and Halted.
- For creating process, operating system maintains a process table. Process table contains an array of structures, with one entry per process. So there is a separate Process Control Block (PCB) for each process. Information in a PCB is updated during the transition of process states. PCB includes Pointer, Process state, Process ID, Process priority, Register, Program counter, I/O status information, Accounting information and Memory limits.

1.2 PROCESS SCHEDULING

- The objective of multiprogramming is to keep to CPU busy for maximum time and achieve maximum CPU utilization.
- In uniprocessor system, only one process is there and any time it may get the attention of the CPU. But in multiprogramming as only one process may get attention of the CPU at a time, other processes have to wait. So the process scheduling is necessary and it is done by operating system. The method of selecting a process to be allocated to CPU is called Process Scheduling.
- When two or more processes compete for the CPU at the same time then choice has to be made which process to allocate the CPU next. This procedure of determining the next process to be executed on the CPU is called process scheduling and the module of operating system that makes this decision is called the Scheduler.

1.2.1 Scheduling Queues

- In multiprogramming, several processes are there in ready or waiting state. These processes form a queue. The various queues maintained by operating system are Job Queue, Ready Queue, and Device Queue.
  1. **Job Queue**: As the process enters the system; it is put into a job queue. This queue consists of all processes in the system.
  2. **Ready Queue:** The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ready queue. This queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCBs in the list. Each PCB includes a pointer field that points to the next PCB in the ready queue in Fig. 1.1 (a).
  3. **Device Queue:** A process may have an input/output request, and the device requested may be busy. In such case, the input/output request in maintained is the device queue. It contains all those processes that are waiting for a particular I/O device. Each device has its own device queue. In case of dedicated devices, the device queue will never have more processes in it. In case of sharable devices, several processes may be in the ready queue.



(b) I/O Device Queue

Fig. 1.1: Scheduling Queues

- CPU scheduling is represented by a queuing diagram as shown in Fig. 1.2. In this figure, queues are represented as rectangles and resources are represented as circles. Directed arrows represent the flow of the processes in the system.

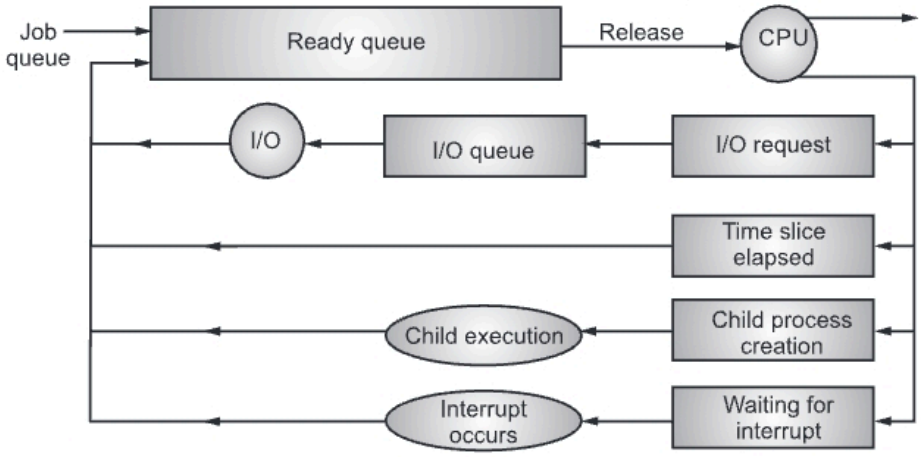


Fig. 1.2: Queuing diagram for CPU Scheduling

- A process that enters the system, it is placed in ready queue. It waits there for the CPU. When the process is executing on the CPU, many events occur like:
  1. The process have I/O request, and then it placed in I/O queue.
  2. The process can create a new sub process and wait for finish that process.
  3. If the interrupt occurs, then the process is put back in the ready queue.
  4. After the CPU is utilized by the process, it may finish its execution and leave the system. At that time it is removed from all queues and has its PCB and resources deallocated.
- The process scheduling must perform the following functions:
  1. It keeps track of the process states (either running, ready or waiting) of all the processes. This is done by the traffic controller.
  2. It selects process from ready queue for execution. This task is done by the scheduler.
  3. It allocates the CPU to process.
  4. When running process requires, any I/O resources or an interrupt occurs or it exceeds its time quantum then processor releases the process.

1.2.2 Schedulers [W-18]

- When a process is created, it moves from one scheduling queue to other. The operating system must select processes from these queues in some way. The job of the appropriate scheduler is to select the process for execution.
- Schedulers are special system softwares which handles process scheduling in various ways. Scheduler's main task is to select the jobs to be submitted into the system and to decide which process to run.

- There are three different types of scheduler as shown in Fig. 1.3 namely long-term scheduler, medium-term scheduler and short-term scheduler.

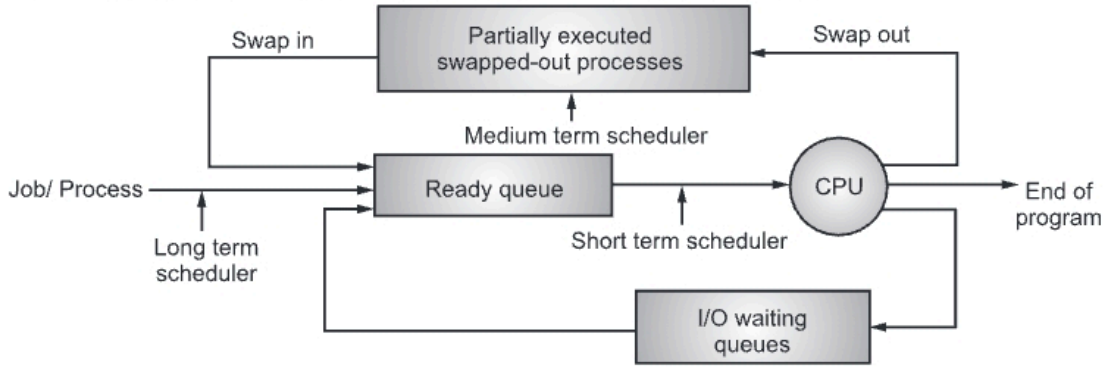


Fig. 1.3: Types of Schedulers

- The types of schedulers are explained below:

1. Long-term Scheduler (Job Scheduler):

- The long-term scheduler selects the job or process to be executed from job pool on a secondary storage device and loads them into memory for execution. The long-term scheduler executes less frequently.
- The long-term scheduler is invoked when the process leaves the system. Because of the longer duration between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.
- It also provides a good process mix of I/O bound and CPU bound processes to the short-term scheduler. The I/O bound process means the process requires more I/O than CPU and CPU bound process means the process require more CPU than I/O.
- If all processes are CPU bound, the I/O waiting queue will almost be empty, and if all the processes are I/O bound, then ready queue will almost be empty, and the short-term scheduler will have little to do. Hence, the system with the best performance will have a combination of CPU bound and I/O bound processes.
- The above tasks are summarized as follows:
  - The long-term scheduler communicates directly with the job pool.
  - It selects number of jobs from the pool and loads them in the memory for execution.
  - Long-term scheduler controls degree of multiprogramming. (Degree of multiprogramming is number of programs that can reside in the memory of the computer at a time).
  - It needs to be invoked only when a job leaves the system. Therefore, it is invoked less frequently as compared to short-term scheduler.
  - It has the responsibility of selecting proper mix of input/output bound and CPU bound jobs, so that CPU utilization is good.

2. Short-term Scheduler (CPU Scheduler):

- Short-term scheduler selects a job from ready queue and submits it to CPU. As the short-term scheduler selects only one job at a time, it is invoked very frequently.
- In case of input/output bound jobs as the ready queue is almost empty, short-term scheduler has very less work to do.
- The systems like time sharing do not have long-term scheduler. The jobs are placed directly in the ready queue for the short-term scheduler. But such some systems have an additional type of scheduler called as medium-term scheduler.

3. Medium-term Scheduler:

- Following are the situations in which medium-term scheduler is required:
  - Sometimes, it is required to remove some number of jobs from memory temporarily and reduce degree of multiprogramming. (Degree of multiprogramming may have to be changed to have a proper job mix of input/output bound and CPU bound jobs to increase CPU utilization).
  - Sometimes, while executing the program, it is found that, memory requirements of the program are changed. Therefore, the job is to be removed from the memory temporarily. Later, the process can be reintroduced into memory, and its execution can be continued where it left off.
- Swapping is term associated with the medium-term scheduler by which processes are temporarily removed and then brought back to the ready queue.
- The dispatcher is the actual component that gives control of the CPU to the process selected by short-term scheduler.

Table 1.1: Comparison between Schedulers

Sr. No.	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.	It controls the degree of multiprogramming.	It provides lesser control over degree of multiprogramming.	It reduces the degree of multiprogramming.
3.	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.
4.	Speed is lesser than short term scheduler.	Speed is fastest among other two.	Speed is in between both short and long term scheduler.
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.

1.2.3 Context Switch

- Switching the CPU from one process to another process requires saving the state of old process and loading the saved state of new process. This task is known as Context Switch.
- Context of a process represented in the PCB. The context of the interrupted old process must require saving the state.
- In simple words, CPU switching from one process to another process is called as context switch.

When Context Switch Occurs?

- Whenever, an interrupt arrives, the CPU must do a state-save of the currently running process, and then switch into kernel mode to handle the interrupt, and then do a state-restore of the interrupted process.
- Similarly, a context switch occurs when the time slice for one process has expired and a new process is to be loaded from the ready queue. This will be instigated by a timer interrupt, which will then cause the current process's state to be saved and the new process's state to be restored.
- Saving and restoring states involves saving and restoring all of the registers and program counter(s), as well as the process control blocks described above.
- Context switch time is overhead; the system does no useful work while switching.
- Context switch times vary from machine to machine, depending on the memory speed, number of CPU registers, and the existence of special instructions. It highly depends on the hardware support. It ranges from 1 to 1000 microseconds.
- Some CPU provides multiple sets of registers (e.g. sun UltraSPARC). A context switch simply involves changing the pointer to the current register set. If active processes are more than register sets, the system resorts to copying register data to and from memory.
- Fig. 1.4 shows the concept of Context Switch.

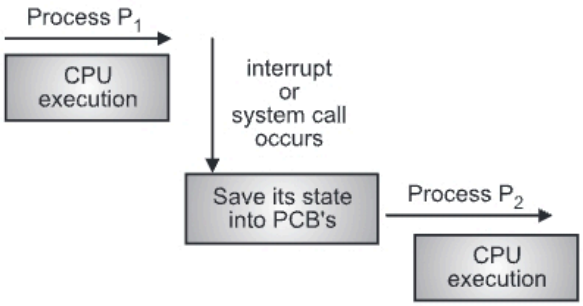


Fig. 1.4: Concept of Context Switch

- When process P_1 is executing and system call from operating system occurs (e.g. I/O request) then status of process P_1 is saved into different registers and process is send to fulfil the outstanding request.
- At that time, operating system process P_2 for execution. When process P_2 requires I/O then again operating system switches to P_1 from where it was left off. This switching time from one process to other is called **Context Switch.**

1.3 OPERATIONS ON PROCESS

- In the system, the processes are created and deleted dynamically. There are two operations provided by operating system on processes:
  1. Process creation.
  2. Process termination.

1.3.1 Process Creation

- When the process is executes, it may create several new processes, via a create process system call. The creating process is called a Parent process and the new process is called Child process.
- If there are many processes created then they all are children processes of parent process. The child process in turn, can create further child processes.
- It is a tree structure as shown in Fig. 1.5.

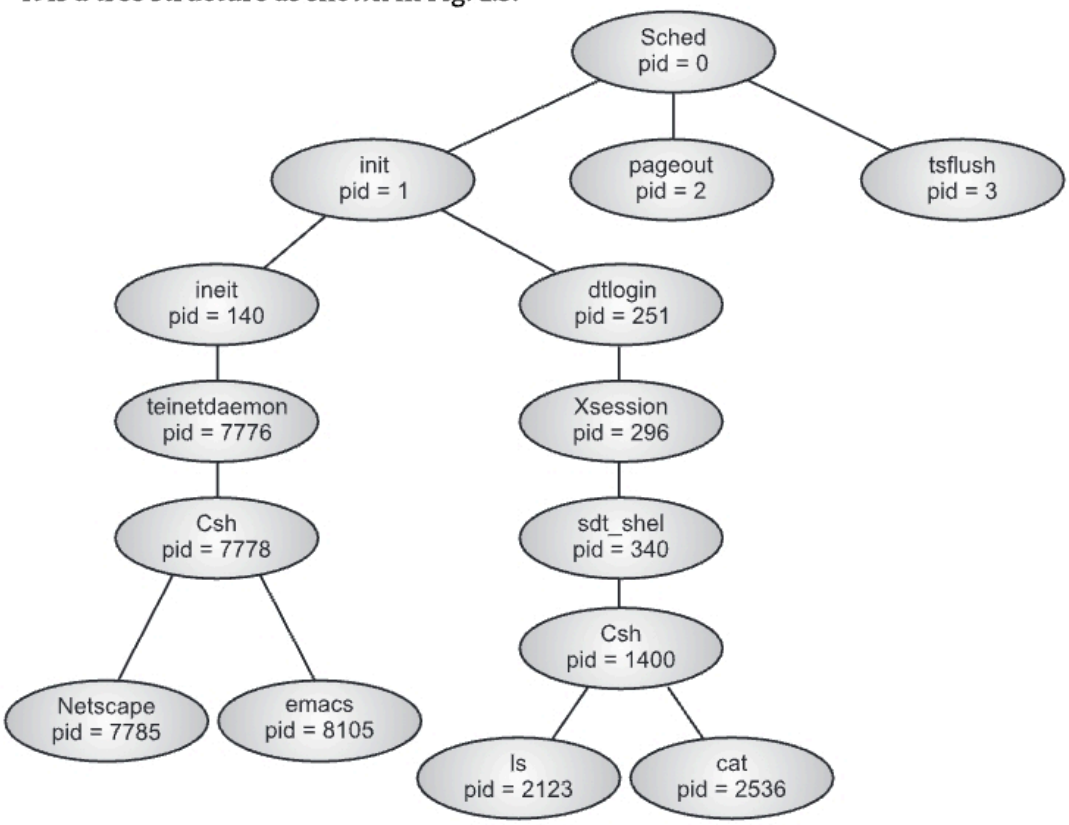


Fig. 1.5: A Tree of Processes in UNIX

- Each process is given an integer identifier, termed its process identifier or PID. The parent PID (PPID) is also stored for each process.
- On typical UNIX systems, the process scheduler is termed **Sched** and is given PID 0. The first thing it does at system startup time is to launch **init**, which gives that process PID 1. Then **init** launches all system daemons and user logins, and becomes the ultimate parent of all other processes.

- On UNIX, the command `$>ps– e1` will list complete information for all processes currently active in the system.
- When a process creates a sub process, the sub process obtains the resources from its parent or from the operating system directly.
- There are two options for the parent process after creating the child:
  1. **Wait for the child process to terminate before proceeding.** The parent makes wait( ) system call, for either a specific child or for any child, which causes the parent process to block until the wait( ) system call returns. UNIX shells normally wait for their children to complete before issuing a new prompt.
  2. **Run concurrently with the child, continuing to process without waiting.** This is the operation seen when a UNIX shell runs a process as a background task. It is also possible for the parent to run for a while, and then wait for the child later, which might occur in a sort of a parallel processing operation.

Address space:

- An address space is a range of valid addresses in memory that are available for a program or process. That is, it is the memory that a program or process can access. The memory can be either physical or virtual and is used for executing instructions and storing data.
- Two possibilities for the address space of the child relative to the parent:
  1. The child may be an exact duplicate of the parent, sharing the same program and data segments in memory. Each will have their own PCB, including program counter, registers, and PID. This is the behavior of the fork system call in UNIX.
  2. The child process may have a new program loaded into its address space, with all new code and data segments. This is the behavior of the spawn system calls in Windows. UNIX systems implement this as a second step, using the exec system call.

Example:

- The UNIX operating system follows a certain procedure to achieve above implementations as shown in Fig. 1.6.
- There are 3 distinct phases in mechanism of process creation and uses 3 system calls: fork(), exec() and wait().

Process related System Calls:

1. Fork() System call:

- In UNIX, each process has a unique integer number called process identifier or process-id. A child is created by the **Fork** system call. The fork operation is used to split a sequence of instructions into two concurrently executable instruction sequences. The parent process can be easily communicated with new process because the new process consists of a copy of the address space of the original process. From the instruction after the fork call, the parent and child processes continue execution.

- When the child process is created, the return code of the fork call is zero and when the child is returned to the parent, the return code is non-zero. Fork usually returns the identification of the child to the parent process and the parent can use that identifier to identify the child. The relation between processes created by fork is executed from a single segment of code and child usually initially obtains a copy of the variables of its parent.

2. Wait() System call:

- The parent then executes wait system call to wait for the child process to complete its execution.

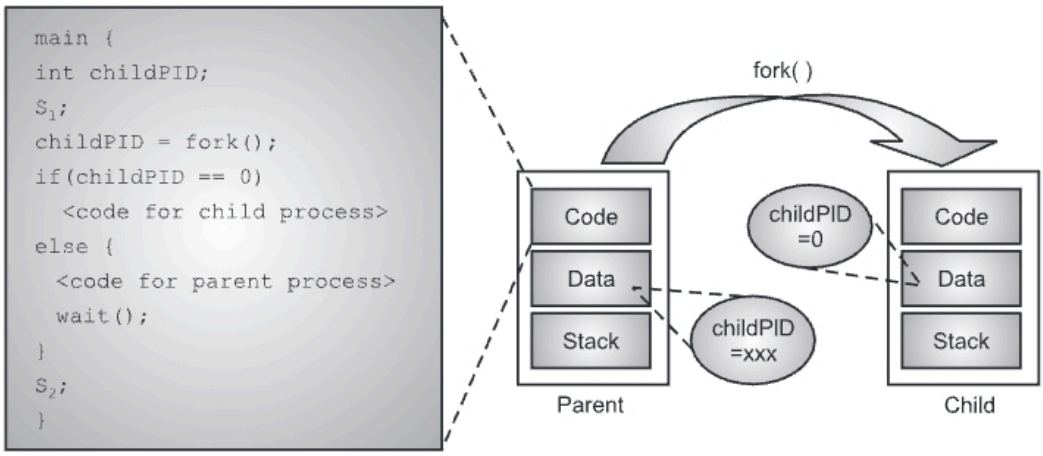


Fig. 1.6: Fork() System Call

- Fig. 1.7 below shows the **fork** and **exec** process on a UNIX system.

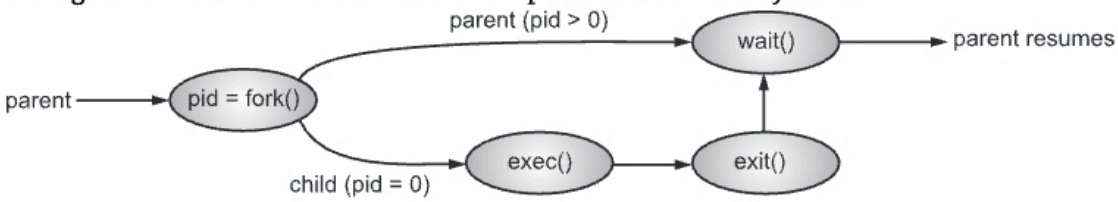


Fig. 1.7: A separate process using the UNIX fork() System Call

3. Exec() System Call:

- The exec system call is used to execute a file which is residing in an active process. When exec is called the previous executable file is replaced and new file is executed.
- By using exec system call, we will replace the old file or program from the process with a new file or program. The entire content of the process is replaced with a new program.
- The new program is loaded into the same process space. The current process is just turned into a new process and hence the process id PID is not changed, this is because we are not creating a new process we are just replacing a process with another process in exec.

- Exec system call is a collection of functions and in C programming language, the standard names for these functions are as follows:
  1. **Execl:** l is for the command line arguments passed a list to the function.
  2. **Execle:** It is an array of pointers that points to environment variables and is passed explicitly to the newly loaded process.
  3. **Execlp:** p is the path environment variable which helps to find the file passed as an argument to be loaded into process.
  4. **Execv:** v is for the command line arguments. These are passed as an array of pointers to the function.

Example:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   printf("PID of example.c = %d\n", getpid());
   char *args[] = {"Hello", "C", "Programming", NULL};
   execv("./hello", args);
   printf("Back to example.c");
   return 0;
}
```

4. Execlp() System call:

- The Execlp system call is used after a Fork call by one of the two processes to replace the processes memory space with a new program. This call loads a binary file into memory and starts its execution. So two processes can be easily communicates with each other.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
   int pid;
   /* fork a child process */
   pid = fork();
   if (pid< 0)
   { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0)
   {  /* child process */
      execlp("/bin/wc","wc",NULL);
   }
```

```
    else
    {   /* parent process */
        /* parent will wait for the child to complete */
        wait(NULL);
        printf("Child Complete");
    }
    return 0;
}
```

5. Nice() System call:

- Using Nice() system call, we can change the priority of the process in multi-tasking system. The new priority number is added to the already existing value.

Example:

```
#include <stdio.h>
main()
{
    int pid, retnice
    print("press DEL to stop process\n");
    pid = fork();
    for(::)
    {
        if(pid==0)
        {
            retnice = nice (- 5);
            print("child gets higher CPU priority %d \n", retnice);
            sleep(1);
        }
        else
        {
            retnice = nice(4);
            print("Parent gets lower CPU priority %d \n",retnice);
            sleep(1);
        }
    }
}
```

6. Sigcld() System call:

- When the process is terminated it should be removed from the process table and shell should know that the process has died. Whenever process is died, it intimates the parent process by signal SIGCLD.

Example:

```
#include <stdio.h>
#include <signal.h>
void abc()
int pid;
main()
{
   pid = fork();
   if(pid = = 0)
   sleep(1);
   else
   {
      signal(SIGCLD, abc);   //signal function to handle SIGCLD signal
      for(i = 0; i < 1000; i++)
         print(%d, i)
         print("parent existing \n");
   }
}
void abc()
{
   print("child died \n");
   getchar();
}
```

1.3.2 Process Termination [S-19]

- After execution ends, the process is terminated. The operating system terminates the process using Exit() system call typically returning int.
- This init is passed along to the parent if it is doing a wait( ), and is typically zero on successful completion and some non-zero code in the event of problems.

child code:

```
int exitCode;
exit(exitCode ); //return exitCode; has the same effect when executed from
main( )
```

parent code:

```
pid_t pid;
int status
pid = wait( &status );   // pid indicates which child exited.
```

- When the process is terminated, all the resources such as memory, I/O files are deallocated or free by the operating system. The process termination status and execution times are returned to the parent if the parent is waiting for the child to terminate, or eventually returned to init if the process becomes an orphan.

- Processes which are trying to terminate but which cannot because their parent is not waiting for them are termed zombies. These are eventually inherited by init as orphans and killed off. Modern UNIX shells do not produce as many orphans and zombies as older systems used to.
- Another system call is **Abort**. Usually, this call can be invoked by only the parent of the process that is to be terminated.
- There are many reasons for terminating the execution of child process by parent such as:
  1. The child has crossed the limit of usage of the resources it has been allocated.
  2. The child process no longer required.
  3. In response to a KILL command or other unhandled process interrupts.
  4. If the parent process terminates then operating system does not allows a child to continue execution.
- A process terminates normally or abnormally when a process is terminates, then all its children must be terminated. This is called Cascading Termination.

1.4 INTERPROCESS COMMUNICATION

- When multiple processes run on a computer system concurrently and more than one process requires CPU at the same time, then it becomes essential to select any one process to which the CPU can be allocated. To serve this purpose, scheduling is required.
- Moreover, the multiple processes running on a system also need to intercommunicate in order to reciprocate some data or information. This kind of intercommunication between several processes is referred to as InterProcess Communication (IPC).

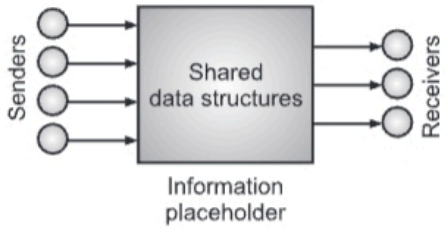


Fig. 1.8: A typical InterProcess Communication Facility

- Operating system provides co-operating processes to communicate with each other via an Inter Process Communication (IPC) facility (See Fig. 1.8).
- The concurrent executing processes are two types:
  1. **Independent Processes:** They cannot affect or be affected by other executing processes.
  2. **Co-operating Processes:** They can affect or be affected by other executing processes in the system.

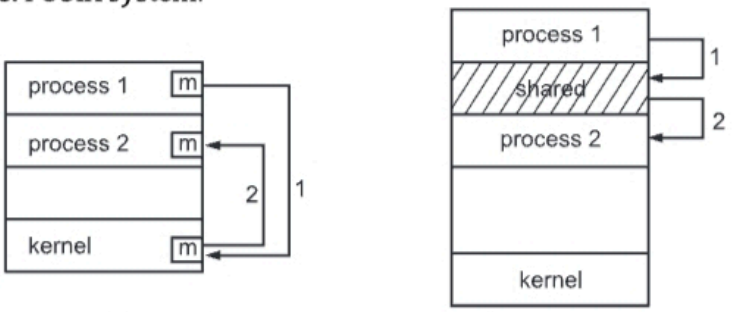
1.4.1 Co-operating Processes

- Co-operating processes can share the data with other processes. Co-operating processes requires InterProcess Communication (IPC) mechanisms.
- The co-operating process also called as Interacting Processes.

Reasons for requiring cooperating processes:

- The co-operation between the processes is required because of the following reasons:
  1. **Information Sharing:** The same information (e.g. a shared file) can be shared by many processes.
  2. **Computation Speedup:** In multiprocessing system, the same task is divided into number of subtasks. These subtasks run parallel to speed up the execution.
  3. **Modularity:** To construct a system in modular way, the system functions are divided into separate processes or threads and these processes wants to communicate with each other.
  4. **Convenience:** It is convenient if user wants to do parallel tasks (e.g. editing, printing, compiling) at same time.
- The co-operating processes require some mechanism to exchange data or pass information to each other. One such mechanism is InterProcess Communication (IPC). This is a very useful facility provided by the operating system that allows the processes running on a single system to communicate with each other.
- Two basic communication models for providing IPC are shared memory systems and message passing systems:
  1. In **Shared Memory Model**, memory is shared by cooperating processes. Shared Memory is faster once it is set up, because no system calls are required and access occurs at normal memory speeds. However it is more complicated to set up, and does not work as well across multiple computers. Shared memory is generally preferable when large amounts of information must be shared quickly on the same computer.

     Example: POSIX system.



(a) Message Passing **(b) Shared Memory**

Fig. 1.9: InterProcess Communication Models

  2. In **Message Passing Model**, co-operating process are communicating by means of message exchange. Message passing requires system calls for every message transfer, and is therefore slower, but it is simpler to set up and works well across multiple computers. Message passing is generally preferable when the amount and/or frequency of data transfers are small, or when multiple computers are involved.

     Example: Mach system. Mach is a micro-kernel architecture used within Mac OS X.

1.4.2 Shared Memory Systems [W-18]

- In these systems, the communicating processes must establish a region of shared memory. The region is established by the process that is creating the shared memory region in its address space.
- The processes that want to communicate using shared memory region must attach to their address space.
- The processes can exchange the information by reading and writing data in the shared area. Let us consider Producer – Consumer problem which arises in interprocess communication.

Producer – Consumer Problem:

- A process is a program in execution. It is an active entity. A sequential process may be in one of the following four states.
  1. **Running** : Instructions are being executed.
  2. **Blocked** : Process waiting for some event to occur.
  3. **Ready** : Process is waiting to be assigned to a processor.
  4. **Deadlocked** : Process is waiting for some event that will never occur.
- There are two categories of the processes which are commonly found in an operating system a producer process and a consumer process.
- For example, a print program produces characters that are consumed by printer driver. A compiler may produce assembly code, which is consumed by an assembler. The assembler in turn, may produce object modules, which are consumed by the loader.
- To allow producer and consumer to run concurrently, we must provide a pool of buffers that will be filled by producer and emptied by consumer. Producer can produce in one buffer and consumer can consume from other buffer.
- The unbound-buffer producers/consumers problem places no limit on the number of buffers. (This is practically difficult to achieve).
- The bounded-buffer problem assumes that there are some fixed n numbers of buffers. Two pointers in and out are used. They are initialized to zero.
- Producer will first check if buffer is not full, then it will produce an item, add it and increment the in pointer. Consumer will check if pool is empty. Condition is (in = out).

The Producer code will be,

```
repeat
   produce an item in next_item
   while in + 1 mod n = out do skip;
   (because buffer is full).
      buffer[in] = next_item
   in:=  in + 1  mod n
until false;
```

The Consumer code will be,

```
repeat
        while in = out do skip;
        next _ item _ consume:= buffer[out]
        out:= out + 1 mod n;
        consume the item in next_item_consume
until false ;
```

- (Recall this solution is nothing but a circular queue of size n; in and out are rear and front respectively, producer and consumer are same as add-queue() and del-queue ( )).
- If producer and consumer want to share some common data, then care must be taken so that simultaneous access to same data does not lead to wrong execution or computation.
- For example, if counter is a common integer one process may add value to it and the other process which is interested in previous value of count will get this new updated value.

1.4.3 Message Passing Systems

- Message passing mechanism is used for communication of processes without sharing the same address space.
- Message passing provides two operations:
  1. send.
  2. receive.
- Messages can be either fixed size or variable size. If the message is fixed size, then system implementation is straight forward. If the message is variable size, then system level implementation is complex.

Communication Link:

- Established between a pair of processes only if both processes have a shared mailbox.
- A link can be associated with more than two processes.
- Each pair of processes can have number of links.

- To communicate processes, the communication link must exist between processes. Following are several methods to implement a link logically.
  - Direct or indirect communication.
  - Synchronous or asynchronous communication.
  - Automatic or explicit buffering.

1. **Direct Communication:**

- The process that wants to communicate must explicitly name the sender or recipient.
- The send() and receive() primitives are as follows:
  - send (P1, message) → send a message to process P1.
  - receive (P2, message) → receive a message from process P2.

- This is symmetric scheme. Properties of a communication link in symmetric scheme:
  (i) Automatically establish between every pair of processes.
  (ii) Associated with exactly two processes.
  (iii) Only one link between two processes.
- Properties of a communication link in asymmetric scheme:
  (i) receive (id, message) → receive message from any process.

2. Indirect Communication:

- In this communication, the messages are sent to and receive from mailboxes or ports. Multiple processes can share the same mailbox or boxes. Only one process can read any given message in a mailbox.
- Initially, the process that creates the mailbox is the owner and is the only one allowed to read mail in the mailbox, although this privilege may be transferred.
- The OS must provide system calls to create and delete mailboxes, and to send and receive messages to/from mailboxes. Each mailbox has unique identifier.
- The send and receive primitives are as follows:
  (i) send (A, message) → send a message to mailbox A.
  (ii) receive (A, message) → receive a message from mailbox A.

3. Synchronization:

- Different combination of send() and receive() are possible. The message passing is either blocking (synchronous) or non-blocking (asynchronous).
  (i) Blocking send: The sending process is blocked until the message is received by receiving process or by the mailbox.
  (ii) Non-blocking send: The sending process send a message and operation resumes.
  (iii) Blocking receive: The receiver block until message is available.
  (iv) Non-blocking receive: The receiver block can receive message.

4. Buffering:

- The messages are stored in a queue. These queues can be implemented for following capacity configurations:
  (i) Zero capacity: This is blocking send stage. The queue has a maximum length of zero.
  (ii) Bounded capacity: The queue has finite length and the maximum no. of messages is stored in a queue equal to length of queue. Sender can continue execution without waiting, if queue is not full. If queue is full, sender must block till queue will have sufficient space.
  (iii) Unbounded capacity: The queue length is infinite. The sender never blocks and any no. of messages can receive and send.

1.5 MULTITHREADED PROGRAMMING

1.5.1 Overview of Thread

- A thread is defined as, the fundamental unit of CPU utilization.

OR

- A thread is a program execution that uses the resources of a process.
- A thread is also called as light weight process. It is also defined as a "unit of concurrency within a process and had access to the entire code and data parts of the process". Thus, thread of the same process can share their code and data with one another.
- The items that the operating system must store that are unique to each thread are:
  1. Thread ID
  2. Saved registers, stack pointer, instruction pointer
  3. Stack (local variables, temporary variables, return addresses)
  4. Signal mask, and Priority (Scheduling information)

Table 1.2: Difference between Process and Thread

Sr. No.	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, 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.

1.5.2 Types of Threads [W-18]

- Practically, threads can be implemented at following two levels in operation system:
  1. The threads implemented at kernel level are known as kernel threads. The kernel threads are entirely handled by the operating system scheduler. An application programmer has no direct control over kernel threads.

2. The threads implemented at the user level are known as user threads. The API for handling user threads is provided by a thread library. The thread library maps the user threads to the kernel threads. A library which provides the programmers with an Application Programming interface (API) for thread creation and management is referred to as a thread library.

1.5.2.1 Kernel Level Threads

- In this case, thread management done by the Kernel. There is no thread management code in the application area.
- Kernel threads are supported directly by the operating system. 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 thread is created using create-thread call. The kernel creates a new thread and assigns it an id. The call returns with the id of the thread.
- A Thread Control Block (TCB) is needed for each thread and scheduler use thread control block for scheduling.
- The Fig. 1.10 shows the Kernel Level Threads.
- For example, Windows XP, Solaris, Linux, Mac OS X.

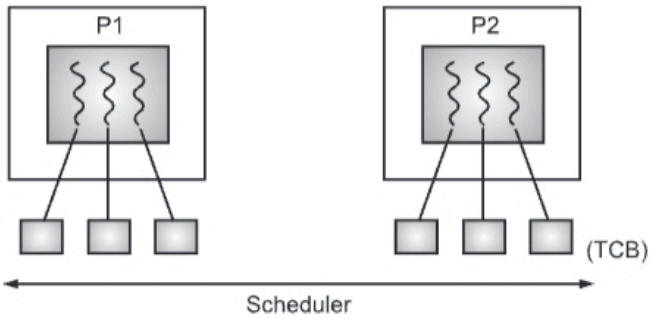


Fig. 1.10: Kernel Level Threads

Advantages:

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

Disadvantages:

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

1.5.2.2 User Level Threads

- In this case, application manages 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 begins with a single thread and begins running in that thread. A process does not create a thread using a **create thread** call.
- The process manages its own threads. The **id** of the user-level thread is not known to the kernel. Thread management is performed through a thread library and no burden on application of programmer.
- User threads are supported above the kernel and are managed without kernel support. The Fig. 1.11 shows the user level threads.

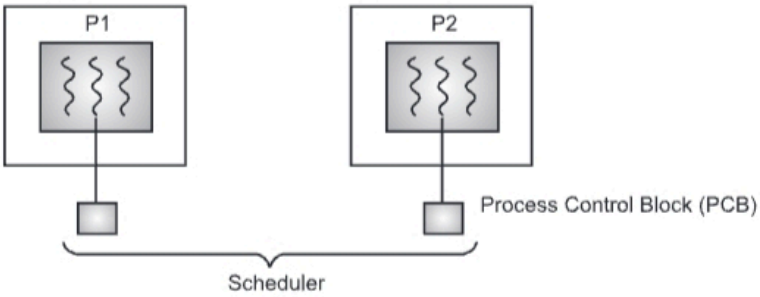


Fig. 1.11: User Level Threads

Advantages:

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

Disadvantages:

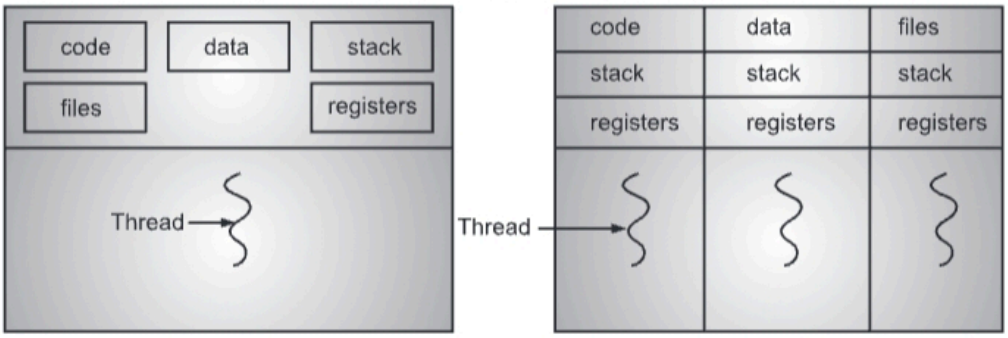
1. In a typical operating system, most system calls are blocking. If a thread were to block in a system call, it would block the process containing the thread. In effect, all threads of the process would get blocked.
2. Multithreaded application cannot take advantage of multiprocessing.

Table 1.3: Difference between User Level Thread and Kernel Level Thread

Sr. No.	User Level Threads	Kernel Level Threads
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.	Multithreaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be Multithreaded.

1.5.3 Multithreaded Programming-Overview [S-19]

- A traditional or heavyweight process has a single thread of control comprises a single thread of control i.e., it can execute one task at a time and thus, is referred to as a single threaded process.
- If a process has multiple threads of control, it can perform more than one task at a time, such a process is known as multithreaded process.
- The Fig. 1.12 shows the Single threaded and Multithreaded processes.



(a) Single-threaded Process **(b) Multithreaded Process**

Fig. 1.12: Single Threaded and Multithreaded Processes

- Operating system supports multiple threads of execution within a single process. Most operating system kernels are multithreaded. Example, **Solaris** operating system is a multithreaded; it creates a set of thread for interrupt handling. **Linux** uses set of thread for managing the amount of free memory. WINDOWS XP, Mac OS X are support multithreaded models. In many situations, multithreaded is preferable than single threaded (DOS is single threaded).

Example:

- Consider a web server. The web-server has to process request from many clients. Clients can request for the web pages or images or audio-video clips etc. If server runs as a single process (threaded) then for each client request, it has to create a separate process to service that request. It is time consuming. Therefore, multithreaded processes are used by server.
- For each client request, server is creating new thread rather than creating process. The multithreaded processes are also used by remote procedure call server.
- When server receives a message from client, the remote server services the message using separate thread. The Fig. 1.13 shows the Multithreaded Server Architecture.

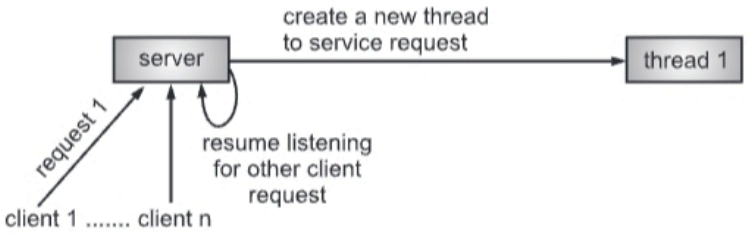


Fig. 1.13: Multithreaded Server Architecture

- When client 1 request for a service, server creates a new thread to service that request and then resume listening for other client request.
- The operating system also treats threads exactly like processes except for one difference that it does not save and restore the execution environment while switching between threads.
- Uses of threads in a single-user multiprocessing system:
  1. Foreground to background work
  2. Asynchronous processing
  3. Speed of execution
  4. Modular program structure

Benefits of Multithreaded Programming:

- Various benefits of multithreaded programming are listed below:
  1. **Responsiveness:** Thread management is performed through a thread library. This library provides function to create and terminate threads, to synchronize their activities and to permit them to make request to the operating system. So multithreading is an interactive application, allow user interaction in one thread while some other operation is performing with another thread. The program is continued to run even if part of it is blocked.
  2. **Resource sharing:** Threads share the memory and the resources of the process to which they belong. Less address space is required.
  3. **Economy:** Threads does not save and restore the execution environment, so context switch time is less. Separate processes are not need to create for each request.
  4. **Scalability:** In multiprocessor systems, threads are running in parallel on different processes, so multithreading is beneficial.

1.5.4 Multithreading Models

- There are different types of relationship between user- and kernel-level threads, each resulting in a specific multithreading model.
- The three common ways to establish the relationship between user thread and kernel thread i.e., Many-to-One, One-to-One and Many-to-Many.

1.5.4.1 Many-to-One Model

- In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers i.e., Many-to-One model maps many user-level threads to one kernel-level thread.
- The number of Kernel threads may be specific to either a particular application or a particular machine.
- Example, Green threads for Solaris.
- Fig. 1.14 shows Many-to-One model concept.

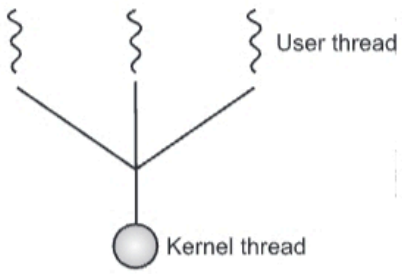


Fig. 1.14: Many-to-One Model

Advantages:

1. Thread management is done by thread library in user space.
2. It is efficient because thread management is done by thread library.

Disadvantages:

1. The disadvantage is entire process will block if a thread makes blocking system call.
2. Multiple threads are not able to run in parallel since only one thread can be accessed by kernel at a time.

1.5.4.2 One-to-One Model

- There is One-to-One relationship of user level thread to the kernel level thread i.e. each user threads maps to a kernel thread.
- The Fig. 1.15 shows One-to-One model.

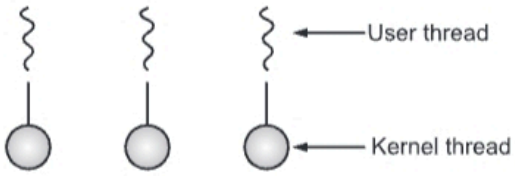


Fig. 1.15: One-to-One Model

- Example, LINUX, OS/2, Windows NT and Windows 2000.

Advantages:

1. This model provides more concurrency than the many to one model.
2. When one thread makes a blocking call, the entire process will not block and concurrent execution of thread is possible.
3. It supports multiple threads to execute in parallel on microprocessors.

Disadvantages:

1. The disadvantage is, for each user thread, the kernel thread is required.
2. Creating a kernel thread is overhead and performance of application degrades.

1.5.4.3 Many-to-Many Model

- In this model, many user level threads multiplex to the Kernel thread of smaller or equal numbers.
- The number of Kernel threads may be specific to either a particular application or a particular machine.
- Example, IRIX, HP-VX and Tru 64 UNIX.
- The Fig. 1.16 shows Many-to-Many model which maps many user held threads to a smaller or equal number of kernel threads.

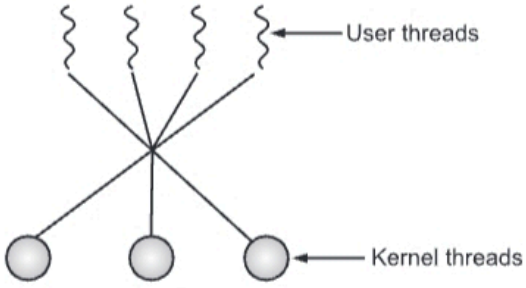


Fig. 1.16: Many-to-Many Model

Advantages:

1. Developers can create any number of threads and corresponding kernel threads can run in parallel on a multiprocessor.
2. Process will not block, since a thread performs a blocking system call, the kernel can schedule another thread for execution.
3. For large number of threads, many-to-many model is efficient than one-to-one.
4. For greater concurrency, this model is efficient than many-to-one.
5. The variation of this model is sometimes called two-level model.

Disadvantages:

1. Multiple threads of kernel are an overhead for OS.
2. Low performance.

1.6 BASIC SCHEDULING CONCEPTS

- In batch operating system, the batch of jobs is submitted to system sequentially. The CPU executes each job sequentially. The other job or process has to wait until the CPU is free and can be rescheduled.
- In multiprogramming system, many programs are present in memory. The CPU executes one program at a time. If that program requires I/O then operating system switches the CPU to another. At that time switching time is requires to switch among the processes.
- The other process has to wait; the operating system takes the CPU away from that process. This cycle is continued.
- In this manner, processes are executed concurrently. Here, CPU is primary resource used and its scheduling is very important.

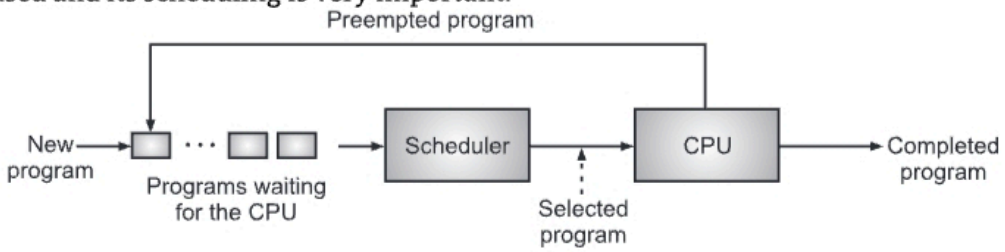


Fig. 1.17: A Schematic of CPU Scheduling

- Fig. 1.17 shows, a Schematic for CPU scheduling. Several programs wait for allocation of the CPU. The scheduler selects one of these programs for execution on the CPU. A preempted program is added to the set of programs waiting for the CPU.
- The main objectives of Scheduling are:
  1. Enforcement of fairness in allocating resources to processes.
  2. Enforcement of priorities to processes.
  3. Make best use of available system resources.
  4. Give preference to processes holding key resources.
  5. Give preference to processes exhibiting good behavior.
  6. Degrade gracefully under heavy loads.

1.6.1 CPU and I/O Burst Cycle

- The success of CPU scheduling depends on the following observed property of processes: the process execution is cycle of CPU execution and I/O wait.
- Processes alternate back and forth between these two states as shown in Fig. 1.18.
- A CPU burst performing calculations and an I/O burst waiting for data transfer in and out of the system.
- When the process is allocated to CPU after selecting from ready queue; it starts with CPU burst. (i.e. time for process is allocated to CPU). The CPU burst is followed by I/O burst when process required I/O. Again CPU burst (when I/O is finished), then I/O burst and so on.
- The last CPU burst will end with a system request to terminate execution. The duration of CPU burst vary from process to process or computer to computer. If the program is I/O bound, then it has short CPU burst, when the program or process is CPU bound then it has very short I/O burst.
- This distribution is important in selecting appropriate CPU scheduling algorithms discussed in further section.

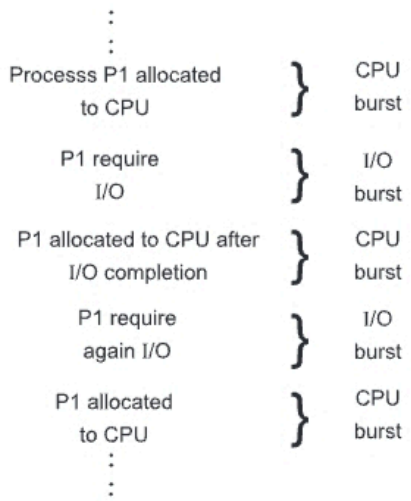


Fig. 1.18: Alternating sequence of CPU and I/O Bursts

1.6.2 CPU Scheduler

- A scheduler is an operating system module (program) that selects a job which is to be admitted next for execution.
- CPU scheduling is the procedure of selecting a process and allocating the processor to the selected process for execution.
- CPU scheduler or short-term scheduler selects the process from ready queue allocates to CPU for execution. This selection of process is not in FIFO order but it depends on CPU scheduling algorithms.

1.6.3 Preemptive and Non-preemptive Scheduling

- The scheduling schemes are of two types:
  1. Preemptive scheduling.
  2. Non-preemptive scheduling.
- When a process switches from running state to the ready state (e.g. when interrupt occurs) or switches from waiting to ready; then it is **preemptive scheduling** scheme. The preemptive means the process is pre-empted or interrupted.

- Example of preemptive scheduling are: Round robin scheduling, Priority scheduling, and Shortest Job First(SJF) scheduling
- When a CPU allocates a process and releases the process after completion or the process switches from running state to waiting state (request for I/O) then it is **non-preemptive scheduling** scheme.
- Preemptive scheduling allows a process to be interrupted in the middle of its execution, taking the CPU away and allocating it to another process. While Non-preemptive scheduling ensures that a process leaves control of the CPU only when it finishes with its current CPU burst.

Advantage:

- It provides an excellent mechanism where the importance of every task may be precisely defined.

Disadvantages of Preemptive Scheduling:

1. The scheduling cost is increases.
2. When two processors are sharing same data. One process is updating the data and at that time the pre-emption occurs then it will affect on the other process which is reading same data.
3. Pre-emption also has an affect on the design of the operating system kernel i.e. the important kernel data may be changed.

Applications:

1. Windows 95 and subsequent versions of Windows used preemptive scheduling.
2. Non-preemptive scheduling (cooperative scheduling) is used only on certain hardware platforms because it does not required special hardware like timer.

1.6.4 Dispatcher [W-18]

- Dispatcher is a component which involves in the CPU scheduling. The dispatcher is the module that actually gives control of the CPU to the process selected by the short-term scheduler.
- The CPU scheduler only selects a process to be executed next on the CPU but it cannot assign CPU to the selected process. The function of setting up the execution of the selected process on the CPU is performed by another module of the operating system, known as **dispatcher**.
- The following are functions performed by dispatcher:
  1. Loading the register of the process.
  2. Switching operating system to the user mode.
  3. Restart the program by jumping to the proper location in the user program.
- The dispatcher needs to be as fast as possible, as it is run on every context switch.
- The time taken by dispatcher to stop one process and start another process to run is called **Dispatch Latency Time.**

1.7 SCHEDULING CRITERIA [W-18]

- Scheduling algorithm is one which selects some job from ready queue based on some criteria and submits it to the CPU.
- There are many different CPU Scheduling algorithms are available. These CPU scheduling algorithms have different properties and may favour one class of processes over another.
- To choose a particular algorithm for a particular situation, we must consider the properties of the various algorithms. For comparing CPU - scheduling algorithms many criteria have been suggested.
- The characteristics used for comparison can make a substantial difference in the determination of the best algorithm.
- The **Scheduling Criteria** are listed below:

1. **CPU Utilization:** CPU utilization is defined as the percentage of time the CPU is busy in executing processes. CPU time is the most expensive resource of computer system. So keep the CPU and other resources as busy as possible. This utilization may range from 0% to 100%. Less the time CPU is idle, more is its utilization. For higher utilization, CPU must be kept as busy as possible, that is, there must be some process running at all times.
2. **Throughput:** Throughput is defined as, the total number of processes that a system can execute per unit of time. If throughput is more then the algorithm is better.
3. **Turnaround Time:** For a process, the important measure is how long it takes to execute that process. Turnaround time is defined as the amount of time that has rolled by from the time of creation to the termination or a process. To put it differently, it is the difference between the time a process enters the system and the time it exits the system. In other words, it is the amount of time to execute a particular process from its entry time. Smaller the turnaround time better is the algorithm. It is the sum of job service time and waiting time.
4. **Waiting time:** Waiting time is defined as the time spent by a process while waiting is the ready queue. More the waiting time, more is the turnaround time. A good system/algorithm should have the aim to have a small waiting time.
5. **Response time:** Response time is the time interval from submission of the job, to time when the first response was produced. This response is not the complete output. By the time first response is produced, computer may start computing next result. This major criterion is used because sometimes turnaround time is delayed because of slow output device and algorithm is not judged properly. To guarantee that all users get good service, we may want to minimize the maximum response time. Response time is defined as the time elapsed between the moment when a user initiates a request and the instant when the system starts responding to this request.

6. **Balanced utilization:** It is the percentage of time all the system resources are busy. It considers not only the CPU utilization but the utilization of I/O devices, memory, and all other resources.

- In order to maximize CPU utilization and throughput, to minimize turnaround time, waiting time and response time, we need to optimize the average measure using scheduling algorithms.

1.8 SCHEDULING ALGORITHMS

- CPU scheduling is the process of selecting a process from the ready queue and assigning the CPU to it. CPU scheduling algorithms decides which of the processes in the ready queue is to be allocated to the CPU.
- The algorithm used by the scheduler to carry out the selection of a process for execution is known as **Scheduling Algorithm**.
- A number of scheduling algorithms are available for CPU scheduling. Some of them are:
  1. FCFS (First Come First Serve)
  2. SJF (Shortest Job First)
     - Preemptive
     - Non-preemptive
  3. Priority scheduling
  4. RR (Round Robin) scheduling
  5. Multi-level queues scheduling
  6. Multi-level feedback queues scheduling
- To find out performance of the scheduling algorithms, we can use Gantt charts. Gantt chart is nothing but a chart use to describe a schedule. It will represent the sequence in which the jobs will be scheduled by the CPU.

1.8.1 FCFS (First Come First Serve) Scheduling

- The simplest among all is the FCFS scheduling algorithm.
- The process that requests the CPU first, is allocated CPU first. Jobs are processed in the order of their arrival in the ready queue. It can be implemented with FIFO (First In First Out) queue.
- The FCFS scheduling algorithm is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it wants to release the CPU, either by terminating or by requesting I/O.
- In time-sharing system it is not useful because process will hold the CPU until it finishes or changes a state to wait state.
- Performance of FCFS is often very poor because a process with a long CPU burst will hold up other processes. Moreover, it can affect overall throughput since I/O on processes in the waiting state may complete; while the CPU bound process is still running.

- Average waiting time for FCFS algorithm is not minimal, and it also varies substantially if the process CPU burst time vary greatly.

Example:

Jobs	Burst Time	Arrival Time
1	2	1
2	3	0
3	4	2

- Above given is the CPU burst time and arrival time of each job present in the ready queue.
- To calculate turnaround time and wait time, we will first draw the Gantt chart.

Jobs	Turnaround time	Wait time
1	$(5-1)=4$	$3-1=2$
2	$(3-0)=3$	$0-0=0$
3	$(9-2)=7$	$5-2=3$
	14	**5**

Average turnaround $= \frac{14}{3} = 4.66$

Average wait time $= \frac{5}{3} = 1.66$

- Note that, turnaround time is the difference between time of completion of a job and its arrival time.
- Wait time is time for which job was waiting in the ready queue i.e. difference of starting time of job and its arrival time.
- If arrival time is not given in the same example then arrival of each job is assumed to be zero.

Gantt chart would be:

Jobs	Turnaround Time	Wait Time
1	$(2-0)=2$	$(0-0)=0$
2	$(5-0)=5$	$(2-0)=2$
3	$(9-0)=9$	$(5-0)=5$
	16	**7**

Average turnaround $= \frac{16}{3} = 5.33$

Average wait time $= \frac{7}{3} = 2.33$

Advantages:

1. FCFS is easier to understand and implement as processes are simply to be added at the end and removed from the front of queue. No process from in between the queue is required to be accessed.
2. FCFS is well suited for batch systems where the longer time periods for each process arc often acceptable.

Disadvantages:

1. Average waiting time is very large.
2. FCFS is not an attractive alternative on its own for a single processor system.
3. Another difficulty with FCFS tends to favor processor bound. Processes over I/O bound processes and may result in inefficient use of both the processor and the I/O devices.

1.8.2 SJF (Shortest Job First) Scheduling [W-18, S-19]

- In SJF algorithm, the process with the shortest expected processing time is assigned to CPU. Hence, the name is shortest job first.
- Jobs or processes are processed in the ascending order of their CPU burst times. Every time the job with smallest CPU burst-time is selected from the ready queue.
- If the two processes having same CPU burst Time then they will be scheduled according to FCFS algorithm performance of this algorithm is very good in comparison to FCFS.
- The SJF scheduling algorithm is probably optimal; it gives minimal average waiting time for a given set of processes. By moving short process before a long one, the waiting time of short process decreases. Consequently, average waiting time reduces.

Drawbacks:

1. As it is optimal algorithm it cannot be implemented in short-term CPU scheduling.
2. This algorithm expects jobs to be submitted with their burst time. Many times it is practically not possible that one will predict the correct burst time of a job.
3. Aging is another problem where big jobs are waiting for long-time in the CPU.

- SJF can be evaluated in two different manners.
  1. **Non-preemptive SJF:** In this method, if CPU is executing one job, it is not stopped in between before completion.
  2. **Preemptive SJF:** In this method, while CPU is executing a job, if a new job arrives with smaller burst time, then the current job is pre-empted (sent back to ready queue) and the new job is executed. It is also called Shortest Remaining Time First (SRTF).

Example:

Jobs	Burst Time	Arrival Time
1	4	0
2	1	1
3	2	2
4	1	3

1. Non-preemptive SJF:

- At ø time unit only one job (J1) is in the ready queue. So we must start with J1.
- J2 and J4 has same burst times, therefore by applying FCFS we will consider J2 before J4.
- As it is non-preemptive, J1 will not be pre-empted before completion.
- Gantt chart would be:

Jobs	Turnaround Time	Wait Time
1	(4 – 0) = 4	(0 – 0) = 0
2	(5 – 1) = 4	(4 – 1) = 3
3	(8 – 2) = 6	(6 – 2) = 4
4	(6 – 3) = 3	(5 – 3) = 2
	17	**9**

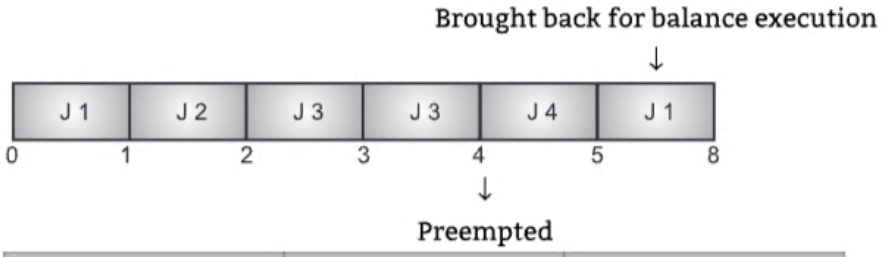
Average turnaround time = $\frac{17}{4}$ = 4.25.

Average wait time = $\frac{9}{4}$ = 2.25.

2. Preemptive SJF:

- At zero time units, there is only one job in the ready queue. Therefore we must start with J1. At one time unit a new job J2 arrives with smaller burst time therefore J1 will be pre-empted and J2 will complete its execution.
- At two time units J3 arrives and CPU starts executing it. J3 has a burst time of two units. After one unit of it is over (i.e. at 3 time units) J4 arrives with 1 as burst time.
- Now if we compare balance burst time of J3 (i.e. 2 – 1) and burst time of J4 (i.e. 1) both is exactly same. Applying FCFS, J3 will continue. At the end J4 will be executed and then balance of J1.

- Gantt chart would be:



Jobs	Turnaround Time	Wait Time
1	(8 – 0) = 8	(0 – 0) + (5 – 1) = 4
2	(2 – 1) = 1	(1 – 1) = 0
3	(4 – 2) = 2	(2 – 2) = 0
4	(5 – 3) = 2	(4 – 3) = 1
	13	**5**

Average turnaround time = $\frac{13}{4}$ = 3.25.

Average wait time = $\frac{5}{4}$ = 1.25.

- Little explanation about Wait Time and Turnaround Time:
  - In the above mentioned example, Job J1 started its execution at zero time units but as it was preempted, its execution was over at 8 time units. Therefore, turnaround time for J1 is 8 – 0 = 8.
  - J1 is arrived at zero time units and immediately started executing. Therefore, wait time for J1 is (0 – 0) = 0.
  - But when J1 was preempted, it was again waiting in the ready queue for CPU. This time (5 – 1) if added to previous wait time.

  ∴ Total wait time of J1 = (0 – 0) + (5 – 1) = 4.

Advantages:

1. Overall performance is significantly improve in terms of response time.
2. SJF algorithm eliminates the variance in waiting and turnaround times.

Disadvantages:

1. There is a risk of starvation of longer processes.
2. It is very difficult to know the length of next CPU burst.

1.8.3 Priority Scheduling

- Along with arrival time and the burst time, sometimes some more information is submitted along with jobs i.e. the priority of the job.
- While processing, the jobs are scheduled in the descending order of their priorities i.e. first the highest priority job is considered and at the end the lowest priority job is considered.

- Equal priority processes are scheduled with FCFS algorithm.
- SJF algorithm is a special case of Priority scheduling algorithm where priority is the predicted next CPU burst time.
- Priorities are roughly categorized into internal and external priorities.
  1. **Internal priorities** are based on burst time, memory requirements, number of open files, etc. measurable quantities.
  2. **External priorities** are human created e.g. seniority, influence of any person, etc.
- The priority scheduling can be either preemptive or non-preemptive. When process enters into the ready queue, its priority is compared with the priority of the currently running process.
- In a preemptive priority scheduling algorithm, at any time, the CPU is allocated one process and if the priority of the newly arrived process is higher than the priority of the currently running process, the process which was allocated is interrupted and return to the queue. The higher priority job is started for execution.
- A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.
- A problem with priority scheduling algorithm is indefinite blocking or starvation. Here, the low priority process may be indefinitely locked out by the higher priority processes. In general, completion of a process within finite time of its creation cannot be guaranteed with this scheduling policy. A solution to the problem of starvation of low priority processes is aging.
- Aging is a process in which the priority of each process is gradually increased after the process spends a certain amount of time in the system. Even a process with an initial priority 0, get the higher priority in the system and it would be executed.

Example:

Jobs	Burst Time	Priority
1	4	4
2	6	1 (Highest)
3	2	3
4	3	5 (Lowest)

Note:
- If there are more than one jobs with same priority apply FCFS.
- If along with priorities highest and lowest is not specified, please do not forget to make your own assumption and specify it before you solve the problem.

Gantt chart for the above example would be:

Arrival of all jobs assumed to be zero.

Jobs	Turnaround Time	Wait Time
1	(12 – 0) = 12	(8 – 0) = 8
2	(6 – 0) = 6	(0 – 0) = 0
3	(8 – 0) = 8	(6 – 0) = 6
4	(17 – 0) = 17	(12 – 0) = 12
	43	26

Average turnaround time = $\frac{43}{4}$ = 10.75.

Average wait time = $\frac{26}{4}$ = 6.5.

Advantages:

1. Important processes are never made to wait because of the execution of less important processes.

Disadvantages:

1. It suffers from the problem of starvation of lower priority processes, since the continuous arrival of higher priority processes will prevent lower priority processes indefinitely from acquiring the CPU.

1.8.4 Round Robin (RR) Scheduling

- Mostly, this scheduling algorithm is used for Time sharing system. The process time is divided into **time slices** or time quantum.
- The process from ready queue is selected in FCFS order. Each one allocates its time slice. If the process is not finished within time slice then again it is placed at the end of the ready queue. When the other processes are waiting in ready queue, the allocated process is not taking more than one time slice.
- The RR scheduler picks the first process from the ready queue, sets a timer to interrupt after one time quantum and dispatches the process.
- In this a fixed time slot is given to each process. After this time slot is over CPU is given to the next process, this takes place in the circular manner.
- Round robin scheduling is a preemptive version of First-Come, First-Served scheduling.
- Processes are dispatched in a first-in-first-out sequence but each process is allowed to run for only a limited amount of time.
- Most important thing for this algorithm is proper selection of time quantum. If time quantum is too small, then most of the time is wasted in context switching (time required switching from one job to other, this includes time to preserve the status of current job and give new job to CPU). If time quantum is too big, then processing is almost FCFS.
- Proper selection of time quantum gives equal priority to all jobs.

- At the end of each time quantum context switch is required to save old process status in registers and load new one. So the time quantum should be large with respect to the context switch time. If time quantum is less, then context switch time is more.
- Turnaround time also depends on the time quantum. It can be improved if most jobs finish their next CPU burst in a single time quantum. For example, given three jobs of 10 time slice each and a quantum of 1 time unit, the average turnaround time = 29. If the time quantum is 10, then the average turn-around time drops to 20. If context switch is added in, then for smaller time quantum we get more turnaround time.

Example:

Jobs	CPU burst time
1	3
2	4
3	5

Time quantum = 2

If time quantum is not given, you are allowed to assume some value and specify your assumption.

Gantt chart would be,

Arrival time of all jobs is considered to be 0 (zero).

Jobs	Turnaround Time
J1	(7 – 0) = 7 = 7
J2	(9 – 0) = 9 = 9
J3	(12 – 0) = 12
	= 28

Jobs	Wait time
J1	(0 – 0) + (6 – 2) = 4
J2	(2 – 0) + (7 – 4) = 5 = 5
J3	(4 – 0) + (9 – 6) = 7
	= 16

$\therefore$ Average turnaround time = $\frac{28}{3} = 9.33$

$\therefore$ Average wait time = $\frac{16}{3} = 5.33$

- While proceeding to next job, current job is brought back to the ready queue. All this time when job is in ready queue is added to the wait time of the job.
- If arrival time is given, user must take care of not entering a job into the Gantt chart before it arrives e.g. if arrival time of job Ji is 4 time units, then Ji should not be there onto the Gantt chart for 0 to 4 time units.

Advantages:

1. Round robin is particularly effective in a general purpose time sharing system or transaction processing system.
2. It is efficient for time sharing systems where the CPU time is divided among the competing processes.
3. RR increases the fairness among the processes.

Disadvantages:

1. If there is a mix of processor bound and I/O bound processes then processor bound process tend to receive an unfair portion of processor time, which result in poor performance for I/O bound processes, inefficient use of I/O devices and an increase in the variance of response time.
2. In RR the processes (even the short processes) may take long time to execute. This decreases the system throughput.
3. RR requires some extra hardware support, such as a timer, to cause interruption after each time out.

1.8.5 Multilevel Queue (MLQ) Scheduling

- A Multilevel Queue Scheduling (MLQ) algorithm partitions the ready queue into separate queues. The processes are permanently assigned to one queue, depending upon their properties such as the size of the memory or the type of the process.
- In MLQ algorithm:
  - All jobs are divided into different categories viz. system jobs, interactive, batched (non-interactive), editing, etc.
  - Ready queue is partitioned into number of sections, one for each of the category present.
  - Each such queue may have separate/different scheduling algorithm.
  - Each queue has jobs with some common property.
  - Priorities are specified for the queue.
  - One by one all queues are processed in descending order of priority.
  - Jobs assigned to a queue remain permanently in the same queue. They are not allowed to move from one queue to other.
  - Until and unless all jobs from high priority queue are not over, scheduling for low priority does not start.
  - Sometimes a fixed time slot is given to each job.
  - While scheduling a particular job, if any high priority job entered the queue, then current job is pre-empted. Therefore, this scheduling is also called as fixed priority preemptive scheduling.

Example:

- Let us consider an example of a multilevel queue-scheduling algorithm with five queues:
  1. System Processes
  2. Interactive Processes
  3. Interactive Editing Processes
  4. Batch Processes
  5. Student Processes
- Each queue has absolute priority over lower-priority queues. No process in the batch queue, for example, could run unless the queues for system processes, interactive processes, and interactive editing processes were all empty. If an interactive editing process entered the ready queue while a batch process was running, the batch process will be pre-empted.

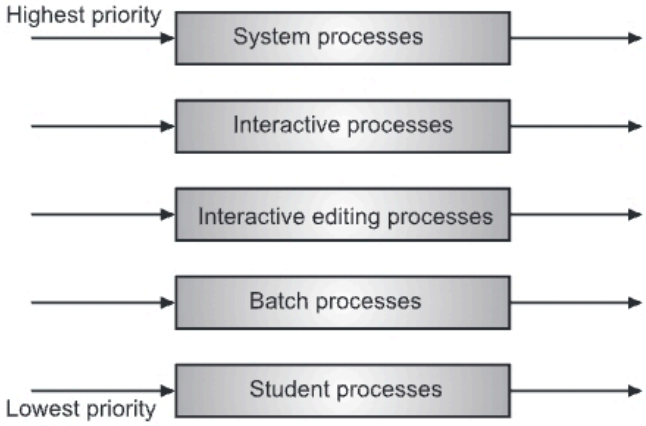


Fig. 1.19: Multilevel Queue Scheduling

- In this case, if there are no processes on the higher priority queue only then the processes on the low priority queues will run. For Example: Once processes on the system queue, the Interactive queue, and Interactive editing queue become empty, only then the processes on the batch queue will run.
- The Description of the processes in the above diagram is as follows:
  - **System Process:** The Operating system itself has its own process to run and is termed as System Process.
  - **Interactive Process:** The Interactive Process is a process in which there should be the same kind of interaction (basically an online game).
  - **Batch Processes:** Batch processing is basically a technique in the Operating system that collects the programs and data together in the form of the batch before the processing starts.
  - **Student Process:** The system process always gets the highest priority while the student processes always get the lowest priority.
- In an operating system, there are many processes, in order to obtain the result we cannot put all processes in a queue; thus this process is solved by multilevel queue scheduling.

Advantage:

1. In MLQ, the processes are permanently assigned to their respective queues and do not move between queues. This results in low scheduling overhead.

Disadvantage:

1. The processes in lower priority queues may have to starve for CPU in case processes are continuously arriving in higher priority queues.

1.8.6 Multilevel Feedback Queue Scheduling

- The Multilevel Feedback Queue Scheduling is an improved/enhanced version of multilevel queue scheduling algorithm.
- In this scheme, processes can move between different queues. The various processes are separated in different queues on the basis of their CPU burst times. The top priority queue is given smallest CPU time quantum.
- If the quantum expires before the process terminates, it is then placed at the back of the next lower queue. Again, if it does not complete, it is put to the last priority queue. The processes in this queue run on FCFS scheduling.

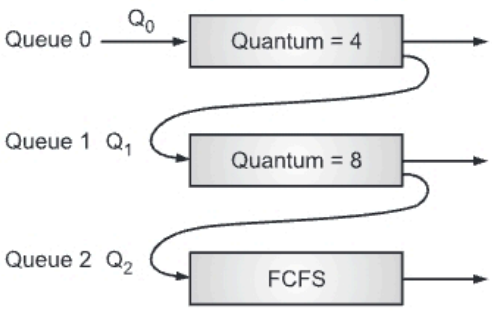


Fig. 1.20: Multilevel Feedback Queue

- If a process uses too much CPU time, it will be moved to a lower priority queue. This leaves I/O bound and interactive processes in the higher priority queues. If a process waits for long time in a lower priority queue, then it is moved to a higher priority queue. This form of aging prevents starvation.
- As shown in Fig. 1.20, different queues are maintained with different fixed time slots.
- All jobs are first placed into the first queue (queue = 0). Each job in Q_0 will be given a time slot of say 4 time units. If a job could not finish within this slot, it will be moved to the tail of next queue i.e. queue 1. After all jobs from queue are processed and queue 0 is empty, processing for queue 1 will start. All jobs from queue 1 are processed from its head to tail. They all are again given a fixed time slot of say 8 units each.
- Jobs not been able to complete in this slot are shifted to tail of queue 2. Queue 2 is processed on FCFS basis, after queue 1 is empty. In short, the highest priority is given to the jobs with smallest CPU burst time. Jobs with very big CPU burst time always sink to the last queue, queue 2 in this example. In queue 2 they will be processed on FCFS basis for the balance of their burst time.
- This is also a preemptive algorithm e.g. while processing job from queue 1, if a job arrives in queue 0, job in queue 0 will pre-empt job in queue 1.

Parameters:

- The below are the parameters for a multilevel feedback queue scheduler:
  - The number of queues.
  - The scheduling algorithm for each queue.
  - The method used to determine when to upgrade a process to a higher-priority queue.
  - The method used to determine when to demote a process to a lower priority queue.
  - The method used to determine which queue a process will enter when that process needs service.

Advantages:

1. It prevents starvation by moving a lower priority process to a higher priority queue if it has been waiting for too long.
2. It is fair to I/O bound (short) processes as these processes need not wait too long and are executed quickly.

Disadvantages:

1. The turnaround time for long processes may increase significantly.
2. It is the most complex scheduling algorithm.
3. In this algorithm, moving the processes between queues causes a number of context switches which results in an increased overhead.

1.9 THREAD SCHEDULING

- Many computers have only one CPU, so threads must share the CPU with other threads. The execution of multiple threads on a single CPU, in some order, is called scheduling.
- A process is a program that performs a single thread of execution. For example, if a process is running a word processor program, a single thread of instructions is being executed.
- This single thread of control allows the process to perform only one task at a time. Nowadays many modern operating systems have extended the process concept to allow a process to have multiple threads of execution.
- Multithreading allows the execution of multiple parts of a program at the same time. These parts are known as threads and are lightweight processes available within the process.
- A thread is a flow of execution through the process code, with its own program counter, system registers and stack.
- A thread is a path of execution within a process. A process can contain multiple threads.
- Threads are implemented using User Level Threads (user managed threads) and Kernel Level Threads (operating system managed threads acting on kernel).

- On operating systems that support them, it is kernel-level threads - not processes - that are being scheduled by the operating system.
- User-level threads are managed by a thread library, and the kernel is unaware of them. To run on a CPU, user-level threads must ultimately be mapped to an associated kernel-level thread, although this mapping may be indirect and may use a Light-Weight Process (LWP).
- In this section, we explore scheduling issues involving user-level and kernel-level threads and offer specific examples of scheduling for Pthreads.

Contention Scope:

- The word contention here refers to the competition or fight among the User level threads to access the kernel resources. Thus, this control defines the extent to which contention takes place.
- One distinction between user-level and kernel-level threads lies in how they are scheduled.
- The thread library schedules user-level threads to run on an available LightWeight Process (LWP), a scheme known as Process-Contention Scope (PCS), since competition for the CPU takes place among threads belonging to the same process.
- When we say the thread library schedules user threads onto available LWPs, we do not mean that the thread is actually running on a CPU; this would require the operating system to schedule the kernel thread onto a physical CPU.
- To decide which kernel thread to schedule onto a CPU, the kernel uses System Contention Scope (SCS).
- Competition for the CPU with SCS scheduling takes place among all threads in the system. Systems using the one-to-one model (such as Windows XP, Solaris 9, and Linux) schedule threads using only SCS.
- Typically, PCS is done according to priority—the scheduler selects the runnable thread with the highest priority to run. User-level thread priorities are set by the programmer and are not adjusted by the thread library, although some thread libraries may allow the programmer to change the priority of a thread.

Pthread Scheduling:

- Now, we highlight the POSIX Pthread API that allows specifying either PCS or SCS during thread creation.
- POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system.
- Pthread identifies the following contention scope values:
  - PTHREAD_SCOPE_PROCESS schedules threads using PCS scheduling.
  - PTHREAD_SCOPE_SYSTEM schedules threads using SCS scheduling.
- On systems implementing the many-to-many model, the PTHREAD_SCOPE_PROCESS policy schedules user-level threads onto available LWPs. The number of LWPs is maintained by the thread library, perhaps using scheduler activations.

- The PTHREAD_SCOPE_SYSTEM scheduling policy will create and bind an LWP for each user-level thread on many-to-many systems, effectively mapping threads using the one-to-one policy.

Solved Problems

Problem 1: Consider the following set of processes that arrive in the ready queue at the same time.

Processes	CPU Time
P1	8
P2	2
P3	1
P4	10
P5	3

Find out turnaround time and wait time for each of the process using FCFS (First Come First Served), SJF (Shortest Job First) and Round Robin (Quantum = 1).

Solution: Arrival time of all jobs assumed to be 0 **FCFS**.

Gantt Chart:

Process	Turnaround Time	Wait Time
P1	(8 – 0) = 8	(0 – 0) = 0
P2	(10 – 0) = 10	(8 – 0) = 8
P3	(11 – 0) = 11	(10 – 0) = 10
P4	(21 – 0) = 21	(11 – 0) = 11
P5	(24 – 0) = 24	(21 – 0) = 21

SJF: Preemptive or non-preemptive is not specified. Therefore assumed to be non-preemptive.

Gantt chart:

Processes	Turnaround Time	Wait Time
P1	(14 – 0) = 14	(6 – 0) = 6
P2	(3 – 0) = 3	(1 – 0) = 1
P3	(1 – 0) = 1	(0 – 0) = 0
P4	(24 – 0) = 24	(14 – 0) = 14
P5	(6 – 0) = 6	(3 – 0) = 3

Round Robin with quantum 1.

Gantt Chart:

P1	P2	P3	P4	P5	P1	P2	P4	P5	P1	P4	P5	P1	P4	P1	P4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

P1	P4	P1	P4	P1	P4	P4	P4

16 17 18 19 20 21 22 23 24

Processes	Turnaround Time
P1	(21 – 0) = 21
P2	(7 – 0) = 7
P3	(3 – 0) = 3
P4	(24 – 0) = 24
P5	(12 – 0) = 12

Processes	Wait Time
P1	(0 – 0) + (5 – 1) + (9 – 6) + (12 – 10) + (14 – 13) + (16 – 15) + (18 – 17) + (20 – 19) = 13
P2	(1 – 0) + (6 – 2) = 5
P3	(2 – 0) = 2
P4	(3 – 0) + (7 – 4) + (10 – 8) + (13 – 11) + (15 – 14) + (17 – 16) + (19 – 18) + (21 – 20) = 14
P5	(4 – 0) + (8 – 5) + (11 – 9) = 9

Problem 2: Consider snapshot of the system:

Job	Arrival time	Burst time
1	0	8
2	1	4
3	2	9
4	3	5

Compute average turnaround time using preemptive SJF and non-preemptive SJF.

Solution: Preemptive SJF.

Job 1	Job 2	Job 4	Job 1	Job 3

0 1 5 10 17 26

Average turnaround time:

Job	Completion time - Arrival time
1	(17 – 0) = 17
2	(5 – 1) = 4
3	(26 – 2) = 24
4	(10 – 3) = 7

$$\text{Average turnaround time} = \frac{17 + 4 + 24 + 7}{4}$$

$$= \frac{52}{4} = 13$$

Non-preemptive SJF:

$$\text{Average turnaround time} = \frac{(8-0)+(12-1)+(17-2)+(26-3)}{4}$$

$$= \frac{8+11+15+23}{4} = \frac{57}{4}$$

$$= \mathbf{14.25}$$

Problem 3: The following series of processes with the given estimated run times arrive in the ready queue in the order shown. For the FCFS and SJF scheduling policies, calculate the waiting time and wait time/run time ratio of each process. Comment on the results.

Job	Estimated runtime
1	10
2	50
3	2
4	100
5	5

Solution:

(a) FCFS: If the jobs arrive shortly one after the other in the order Job 1, Job 2, Job 3, Job 4, Job 5 the result shown in the following Gantt chart:

The waiting time is 0 time units for Job 1, 10 time units for Job 2, 60 time units for Job 3, 62 time units for Job 4, and 162 time units for Job 5.

Thus, the wait time/runtime ratio for job1 = 0,

The wait time/runtime ratio for Job 2 = $\frac{10}{50}$ = 0.2

The wait time/runtime ratio for Job 3 = $\frac{60}{2}$ = 30

The wait time/runtime ratio for Job 4 = $\frac{62}{100}$ = 0.62

The wait time/runtime ratio for Job 5 = $\frac{162}{5}$ = 32.4

(b) SJF: The result of the above example, using SJF scheduling is shown in the following Gantt chart:

The waiting time is 7 for Job 1, 17 time unit for Job 2, 0 time unit for Job 3, 67 time unit for Job 4, and 2 time unit for Job 5.

Thus, **wait time/runtime ratio**,

$$\text{for Job 1} = \frac{7}{10} = 0.7$$

$$\text{for Job 2} = \frac{17}{50} = 0.34$$

$$\text{for Job 3} = \frac{0}{2} = 0$$

$$\text{for Job 4} = \frac{67}{100} = 0.67$$

$$\text{for Job 5} = \frac{2}{5} = 0.4$$

From the above result, it is found that SJF is an optional scheduling algorithm in terms of minimizing the average waiting time of a given work load.

Problem 4: Five jobs arrive at time 0, in the order given.

Jobs	Burst Time
1	10
2	29
3	3
4	7
5	12

Considering FCFS, SJF and RR (quantum = 10) scheduling algorithms for this set of jobs, which algorithms would give the minimum average waiting time?

Solution:

(a) For FCFS, we would execute the jobs as:

Jobs	Waiting time
1	0
2	10
3	39
4	42
5	49
	140

The average waiting time is then,

$$\frac{140}{5} = 28$$

(b) For SJF (non-preemptive) we execute the jobs as:

Jobs	Waiting time
1	10
2	32
3	0
4	3
5	20
	65

The average waiting time is, $\frac{65}{5} = 13$

(c) With RR (quantum = 10), we start job 2, but pre-empt it after 10 time units, putting it in the back of the queue.

Jobs	Waiting time
1	0
2	32
3	20
4	23
5	40
	115

The average waiting time = $\frac{115}{5} = 23$

Problem 5: Consider the following snapshot.

Process	Burst time	Arrival time
P1	5	1
P2	3	0
P3	2	2
P4	4	3
P5	2	13

Compute average turnaround time and average waiting time with RR algorithm with time slice = 2.

Solution: Gantt chart:

$$\text{Average turnaround time} = \frac{(16-1)+(9-0)+(6-2)+(13-3)+(15-13)}{5}$$

$$= \frac{15+9+4+10+2}{5} = \frac{40}{5} = 8$$

$$\text{Average waiting time} = \frac{(15-4-1)+(8-2-0)+(4-2)+(11-2-3)+(13-13)}{5}$$

$$= \frac{10+6+2+6+0}{5}$$

$$= \frac{24}{5} = 4.8$$

Problem 6: Consider the following set of processes with CPU burst time given in milliseconds.

Process	Burst time	Arrival time	Priority
P1	8	0	4
P2	6	1	6
P3	7	3	3
P4	9	3	1 (highest)

Illustrate the evaluation of these process using non-preemptive SJF and priority preemptive CPU scheduling algorithm. Also calculate average waiting time.

Solution: 1. Non-preemptive SJF:

Gantt Chart:

$$\text{Average turnaround time} = \frac{(8-0)+(14-1)+(21-3)+(30-3)}{4}$$

$$= \frac{8+13+18+27}{4}$$

$$= \frac{66}{4} = 16.5$$

$$\text{Average waiting time} = \frac{(0-0)+(8-1)+(14-3)+(21-3)}{4}$$

$$= \frac{0+7+11+18}{4} = \frac{36}{4} = 9$$

2. Priority preemptive algorithm:

Gantt chart:

$$\text{Average turnaround time} = \frac{(24-0)+(30-1)+(19-3)+(12-3)}{4}$$

$$= \frac{24+29+16+9}{4}$$

$$= \frac{78}{4} = 19.5$$

$$\text{Average waiting time} = \frac{(19-3-0)+(24-1)+(12-3)+(3-3)}{4}$$

$$= \frac{16+23+9+0}{4}$$

$$= \frac{48}{4} = 12$$

Problem 7: Consider the following set of process with CPU burst time given in milliseconds.

Process	Burst time	Arrival time
P1	5	1
P2	3	0
P3	2	2
P4	4	3
P5	8	2

Illustrate the execution of these process using FCFS and preemptive SJF. Calculate average turnaround time and average waiting time.

Solution: 1. FCFS:

$$\text{Average turnaround time} = \frac{(8-1)+(3-0)+(10-2)+(22+3)+(18-2)}{5}$$

$$= \frac{7+3+8+19+16}{5}$$

$$= \frac{53}{5} = 10.6$$

$$\text{Average waiting time} = \frac{(3-1)+(0-0)+(8-2)+(18-3)+(10-2)}{5}$$

$$= \frac{2+0+6+15+8}{5}$$

$$= \frac{31}{5} = 6.2$$

2. Preemptive SJF:

Gantt chart:

$$\text{Average turnaround time} = \frac{(14-1)+(3-0)+(5-2)+(9-3)+(22-2)}{5}$$

$$= \frac{45}{5} = 9$$

$$\text{Average waiting time} = \frac{(9-1)+(0-0)+(3-2)+(5-3)+(14-2)}{5}$$

$$= \frac{23}{5} = 4.6$$

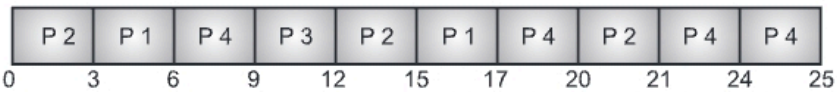
Problem 8: Consider the following set of process with CPU burst time given in milliseconds.

Process	Burst time	Arrival time
P1	1	5
P2	0	7
P3	3	3
P4	2	10

Illustrate the execution of these process using RR (with time quantum = 3 ms) and shortest remaining time first. Calculate average turnaround time and average waiting time.

Solution: 1. RR CPU Scheduling:

Gantt chart:



$$\text{Average turnaround time} = \frac{(17-1)+(21-0)+(12-3)+(25-2)}{4}$$

$$= \frac{16+21+9+23}{4}$$

$$= \frac{69}{4} = 17.25$$

$$\text{Average waiting time} = \frac{(15-3-1)+(20-6-0)+(9-3)+(21-6-2)}{4}$$

$$= \frac{11+14+6+13}{4}$$

$$= \frac{44}{4} = 11$$

2. SJF or SRTF: (Preemptive)

Gantt chart: Preemptive SJF is also called shortest remaining time first.

$$\text{Average turnaround time} = \frac{(6-1)+(15-0)+(9-3)+(25-2)}{4}$$

$$= \frac{5+15+6+23}{4} = \frac{49}{4} = 12.25$$

$$\text{Average waiting time} = \frac{(1-1)+(9-1-0)+(6-3)+(15-2)}{4}$$

$$= \frac{0+8+3+13}{4} = \frac{24}{4} = 6$$

Problem 9: Consider the following set of process with CPU burst time given in milliseconds.

Process	Burst time	Arrival time	Priority
P1	2	0	3
P2	3	1	1
P3	4	2	4 (Highest)
P4	3	3	2

Illustrate the execution of these process using non-preemptive SJF and non-preemptive priority. Calculate average turnaround time and average waiting time.

Solution: 1. Non-preemptive SJF:

Gantt Chart:

$$\text{Average turnaround time} = \frac{(2-0)+(5-1)+(12-2)+(8-3)}{4}$$

$$= \frac{2+4+10+5}{4} = \frac{21}{4} = 5.25$$

$$\text{Average waiting time} = \frac{(0-0)+(2-1)+(8-2)+(5-3)}{4}$$

$$= \frac{9}{4} = 2.25$$

2. Non-preemptive priority:

Gantt chart:

$$\text{Average turnaround time} = \frac{(2-0)+(12-1)+(6-2)+(9-3)}{4}$$

$$= \frac{2+11+4+6}{4} = \frac{23}{4} = 5.75$$

$$\text{Average waiting time} = \frac{(0-0)+(9-1)+(2-2)+(6-3)}{4}$$

$$= \frac{8+0+3}{4}$$

$$= \frac{11}{4} = 2.75$$

Problem 10: Consider the following set of process with CPU burst time given in milliseconds.

Process	Burst time	Arrival time
J1	1	5
J2	0	7
J3	3	3
J4	2	10

Compute average time around time using RR (time quantum = 2) and shortest remaining time first.

Solution: 1. RR:

Gantt chart:

$$\text{Average turnaround time} = \frac{(18-1)+(19-0)+(15-3)+(25-2)}{4}$$

$$= \frac{17+19+12+23}{4} = \frac{71}{4} = 17.75$$

2. Shortest remaining time first:

Gantt chart:

$$\text{Average turnaround time} = \frac{(6-1)+(15-0)+(9-3)+(25-2)}{4}$$

$$= \frac{5+15+6+23}{4}$$

$$= \frac{49}{4} = 12.25$$

Problem 11: Consider the following snapshot of a system:

Process	CPU Burst time	Arrival time
P1	5	3
P2	2	0
P3	2	4
P4	3	5

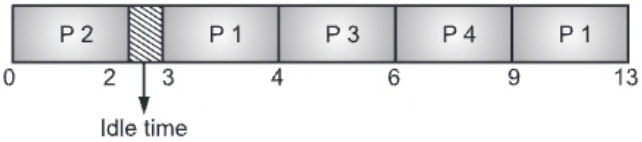
Draw the Gantt chart and find average waiting time for the following scheduling algorithms:

(i) Preemptive SJF.

(ii) Round Robin (time quantum = 2).

Solution:

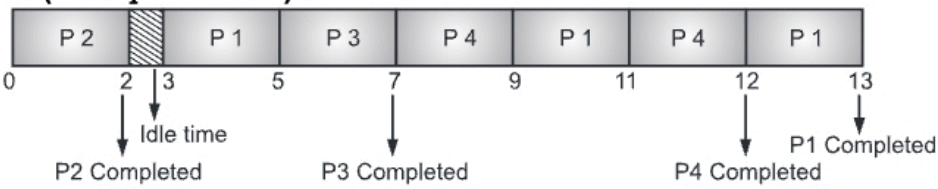
(i) Preemptive SJF: Gantt chart



Waiting time:

Process	Waiting time
P1	(3 – 3) + (9 – 4) = 5 +
P2	(0 – 0) = 0 +
P3	(4 – 4) = 0 +
P4	(6 – 5) = 1
Total waiting time	= 6
Average waiting time	= 6/4 = 1.5

(ii) RR (Time quantum = 2): Gantt chart



Waiting time:

Process	Waiting time
P1	(3 – 3) + (9 – 5) + (12 –11) = 5 +
P2	(0 – 0) = 0 +
P3	(5 – 4) = 1 +
P4	(7 – 5) + (11 – 9) = 4
Total waiting time	= 10
Average waiting time	= 10/14 = 2.5

Problem 12: Consider the following snapshot of a system:

Process	Burst time	Arrival time
P_1	5	1
P_2	3	0
P_3	2	2
P_4	4	3
P_5	2	13

Compute the average turnaround time and average waiting time using:

(i) SJF (Non-preemptive).

(ii) Round Robin (Time Quantum = 2).

Solution:

(i) SJF (Non-preemptive):

$$\text{Turnaround time} = (14-3)+(3-0)+(5-2)+(9-3)+(16-13)$$

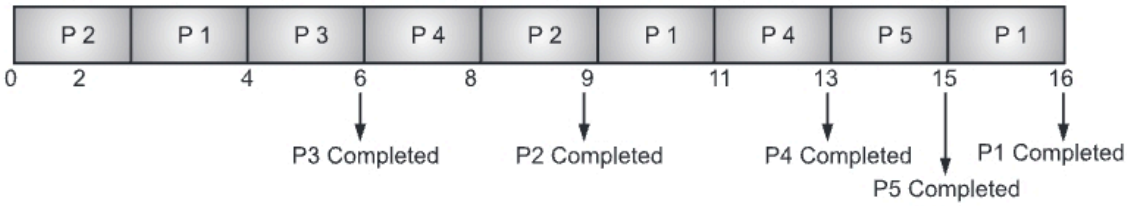
$$\text{Avg. TT} = \frac{(11)+(3)+(3)+(6)+(3)}{5} = \frac{26}{5} = 5.2$$

$$\text{Waiting time} = \frac{(9-1)+(0-0)+(3-2)+(5-3)+(14-13)}{5}$$

$$= \frac{8+1+2+1}{5}$$

$$\text{Avg. waiting time} = 2.4$$

(ii) Round Robin (Time quantum = 2):



$$\text{Average Turnaround TT} = \frac{(16-1)+(9-0)+(6-2)+(13-3)+(15-13)}{5}$$

$$= \frac{15+9+4+10+2}{5}$$

$$\text{Average TT} = 8$$

$$\text{Average Waiting Time} = \frac{[(15-11)+(9-4)+(2-1)]+[(8-2)+0] + [4-2]+[(11-6)+(6-3)+[13-13]}{5}$$

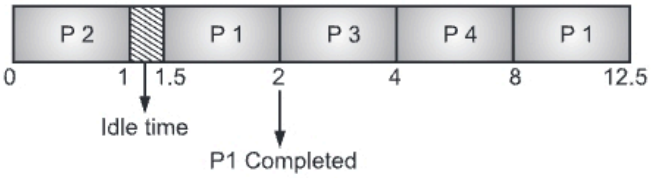
$$\text{Average Waiting Time} = \frac{10+6+2+8+0}{5} = 5.2$$

Problem 13: Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

Process	Burst time	Arrival time
P1	5	1.5
P2	1	0
P3	2	2
P4	4	3

Illustrate the execution of these processes using preemptive SJF CPU scheduling algorithm. Calculate average waiting time and average turnaround time. Give the contents of Gantt chart.

Solution: Preemptive SJF:



$$\text{Average Turn around time} = \frac{(12.5-1.5)+(1-0)+(4-2)+(8-3)}{4}$$

$$= \frac{19}{4} = 4.75$$

$$\text{Average waiting time} = \frac{[(8-2)+(1.5-1.5)]+(0-0)+(2-2)+(4-3)}{4}$$

$$= \frac{7}{4}$$

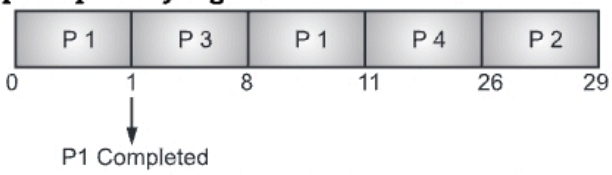
$$= 1.75$$

Problem 14: Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

Process	Burst time	Arrival time	Priority
P1	4	0	3
P2	3	2	1 (lowest)
P3	7	1	4 (highest)
P4	15	3	2

Illustrate the execution of these processes using preemptive priority algorithm. Draw Gantt chart and calculate average turnaround time and waiting time.

Solution: Preemptive priority algorithm:



$$\text{Average Turnaround time} = \frac{(11-0)+(29-2)+(8-1)+(26-3)}{4}$$

$$= \frac{68}{4}$$

$$= 17$$

$$\text{Average waiting time} = \frac{[(8-1)+(0-0)]+(26-0)+(1-1)+(11-3)}{4}$$

$$= \frac{41}{4}$$

$$= 10.25$$

Problem 15: Consider the following set of processes, with the length of CPU burst time and arrival time in milliseconds.

Process	Burst time	Arrival time
P1	4	2
P2	6	0
P3	2	1

Illustrate the execution of these processes using Round Robin (RR) CPU scheduling algorithm (quantum = 3 milliseconds). Calculate average waiting time and average turnaround time. Give the contents of Gantt chart.

Solution: Round Robin Algorithm:

$$\text{Average turnaround time} = \frac{(10-2)+(12-0)+(4-1)}{3}$$

$$= \frac{23}{3} = 7.6$$

$$\text{Average waiting time} = \frac{[(8-6)+(4-2)] + [(10-8)+(6-2)+0] + (2-1)}{3}$$

$$= \frac{4+6+1}{3} = \frac{11}{3} = 3.66$$

Problem 16: Consider the following snapshot of the system.

Process	Burst time	Priority	Arrival time
P1	10	3	0
P2	5	0 (high)	4
P3	2	1	3
P4	16	2	5
P5	8	4 (low)	2

Schedule the above set of processes according to:

(i) Non-preemptive priority scheduling algorithm.

(ii) Preemptive priority scheduling algorithm.

Draw proper Gantt chart and find average turnaround time and waiting time.

Solution: (i) Non-preemptive priority:

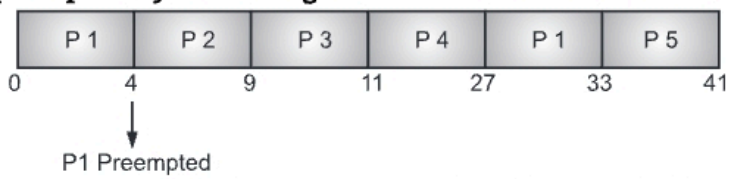
$$\text{Average Turn around time} = \frac{(10-0)+(15-4)+(17-3)+(33-5)+(41-2)}{5}$$

$$= \frac{102}{5} = 20.4$$

$$\text{Average waiting time} = \frac{[(0-0)]+(10-4)+(15-3)+(17-4)+(33-2)}{5}$$

$$= \frac{62}{5} = 12.4$$

(ii) Preemptive priority scheduling:



$$\text{Average turnaround time} = \frac{(33-0)+(9-4)+(11-3)+(27-5)+(41-2)}{5}$$

$$= \frac{107}{5} = 21.4$$

$$\text{Average waiting time} = \frac{[(27-4)+(0-0)+(4-4)+(9-3)+(11-5)+(33-2)]}{5}$$

$$= \frac{66}{5} = 13.2$$

Problem 17: Consider the following set of processes with the length of CPU burst time and arrival time in milliseconds:

Process	Burst time	Arrival time	Priority
P_1	5	1	1 (h)
P_2	6	0	2
P_3	2	1	1
P_4	4	0	3 (*l*)

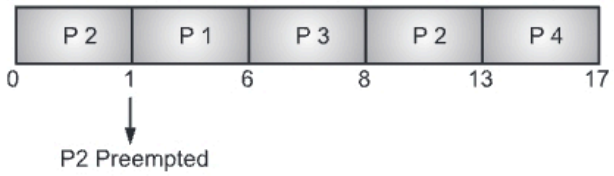
(a) Illustrate the execution of these processes using preemptive priority and FCFS scheduling algorithm.

(b) Calculate waiting time and turnaround time for each process and calculate average waiting time and average turnaround time.

(c) Give the contents of Gantt chart.

Solution:

(i) Preemptive priority algorithm:



$$\text{Average turnaround time} = \frac{(6-1)+(13-0)+(8-1)+(17-0)}{4}$$

$$= \frac{42}{4} = 10.5$$

$$\text{Average waiting time} = \frac{(1-1)+[(8-1)+0]+(6-1)+(13-0)}{4}$$

$$= \frac{25}{4} = 6.25$$

(ii) FCFS algorithm:

$$\text{Average turnaround time} = \frac{(15-1)+(6-0)+(17-1)+(10-0)}{4}$$

$$= \frac{46}{4} = 11.5$$

$$\text{Average waiting time} = \frac{(10-1)+(0-0)+(15-1)+(6-0)}{4}$$

$$= \frac{29}{4} = 7$$

Summary

- The method of selecting a process to be allocated to CPU is called Process Scheduling.
- In multiprogramming, several processes are there in ready or waiting state. These processes form a queue. The various queues maintained by operating system are: Job Queue, Ready Queue and Device Queue.
- When a process is created, it moves from one scheduling queue to other. The job of the appropriate scheduler is to select the process for execution. There are three different types of scheduler long-term scheduler, medium-term scheduler and short-term scheduler.
- The long-term scheduler selects the job or process to be executed from job pool on a secondary storage device and loads them into memory for execution. The long-term scheduler executes less frequently. It controls degree of multi-programming.
- Short-term scheduler selects a job from ready queue and submits it to CPU. As the short-term scheduler selects only one job at a time, it is invoked very frequently.
- Swapping is term associated with the medium-term scheduler by which processes are temporarily removed and then brought back to the ready queue.
- Switching the CPU from one process to another process requires saving the state of old process and loading the saved state of new process. This task is known as Context Switch.
- There are two operations provided by operating system on processes: Process creation and Process termination.

- When a process creates a sub process, the sub process obtains the resources from its parent or from the operating system directly.
- In UNIX, each process has a unique integer number called process identifier or process-id. A child is created by the FORK system call. The FORK operation is used to split a sequence of instructions into two concurrently executable instruction sequences. The parent process can be easily communicated with new process because the new process consists of a copy of the address space of the original process.
- After execution ends, the process is terminated. The operating system terminates the process using EXIT system call typically returning int.
- The concurrent executing processes are two types i.e., Independent processes, Co-operating processes.
- Independent processes cannot affect or be affected by other executing processes.
- Co-operating processes can affect or be affected by other executing processes in the system.
- Co-operating processes can share the data with other processes. Cooperating processes require some type of inter-process communication, which is most commonly one of two types: Shared memory and Message passing.
- In shared memory model, memory is shared by cooperating processes. Shared Memory is faster. Once it is set up, because no system calls are required and access occurs at normal memory speeds. Shared memory is generally preferable when large amounts of information must be shared quickly on the same computer. Example, POSIX system.
- In message passing model, cooperating process are communicating by means of message exchange Message Passing requires system calls for every message transfer. Message passing is generally preferable when the amount and/or frequency of data transfers is small, or when multiple computers are involved. Example – Mach system.
- A thread is a flow of execution through the process code, with its own program counter, system registers and stack.
- The multiple processes running on a system also need to intercommunicate in order to reciprocate some data or information. This kind/type of intercommunication between several processes is referred to as InterProcess Communication (IPC).
- A thread is also called as Light -Weight process. It is used to utilize CPU more effectively.
- A process may be multithreaded, where the same program contains multiple concurrent threads of execution. If the processes are multithreaded, it can perform more than one task at a time.
- Operating system supports multiple threads of execution within a single process. That operating system kernel is multithreaded.
- In a single core system, one processor runs multiple threads. The threads are executed in interleaved manner, because processor can run only one thread at a time.

- In a multicore system, more than one processor is in the system. The threads are executed in parallel manner. Each processor can run one thread at a time.
- The threads are implemented using two approaches: Kernel level threads and User level threads.
- In kernel level threads, 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.
- In user level threads, application manages 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.
- There are three common ways to establish the relationship between user thread and kernel thread: Many-to-One, One-to-One and Many-to-Many.
- Many-to-One model maps many user-level threads to one kernel-level thread.
- In One-to-One model, there is one to one relationship of user level thread to the kernel level thread.
- In Many-to-Many model, many user level threads multiplex to the Kernel thread of smaller or equal numbers.
- In multi-programming environment, many processes are in memory. The problem of determining when processors should be assigned and to which processes is called processor scheduling or CPU scheduling.
- The success of CPU scheduling depends on the following observed property of processes: The process execution is cycle of CPU execution and I/O wait. Processes alternate back and forth between these two states.
- CPU scheduler or short-term scheduler selects the process from ready queue allocates to CPU for execution.
- The scheduling schemes are of two types: Preemptive scheduling and Non-preemptive scheduling.
- When a CPU allocates a process and releases the process after completion or the process switches from running state to waiting state (request for I/O) then it is non-preemptive scheduling scheme.
- When a process switches from running state to the ready state (e.g. when interrupt occurs) or switches from waiting to ready; then it is preemptive scheduling scheme. The preemptive means the process is pre-empted or interrupted.
- The dispatcher is the module that actually gives control of the CPU to the process selected by the short-term scheduler.
- The time taken by dispatcher to stop one process and start another process to run is called dispatch latency time.
- Scheduling algorithm is one which selects some job from ready queue based on some criteria and submits it to the CPU.
- Scheduling criteria are CPU Utilization, Throughput, Turnaround time, Waiting time and Response time.

- In order to maximize CPU utilization and throughput, to minimize turnaround time, waiting time and response time, we need to optimize the average measure using scheduling algorithms.
- The different scheduling algorithms are: FCFS (first come first served), SJF (shortest job first), Priority, RR (Round Robin), Multi-level queues and Multi-level feedback queues.
- In FCFS scheduling, the process that requests the CPU first, is allocated the CPU first. Thus, the name First-Come-First-Served. The implementation of FCFS is easily managed with a FIFO queue. FCFS scheduling algorithm is non-preemptive.
- In SJF, the process with the least estimated execution time is selected from the ready queue for execution. It associates with each process, the length of its next CPU burst. SJF algorithm can be preemptive or non-preemptive.
- In priority scheduling, a priority is associated with all processes.
- Processes are executed in sequence according to their priority. CPU is allocated to the process with highest priority. Priority scheduling can be preemptive or non-preemptive.
- In Round Robin scheduling, processes are dispatched in FIFO but are given a small amount of CPU time known as Time Quantum or Time Slice. If a process does not complete before its time slice expires, the CPU is preempted and is given to the next process in the ready queue. Round Robin scheduling is always preemptive as no process is allocated the CPU for more than given time quantum.
- Multi-Level Queue scheduling classifies the processes according to their types. Multi-Level Feedback Queue scheduling is an enhancement of MLQ. In this scheme, processes can move between different queues. The various processes are separated in different queues on the basis of their CPU burst times.
- The process scheduler schedules only the kernel threads.
- User threads are mapped to kernel threads by the thread library - The OS (and in particular the scheduler) is unaware of them.

Check Your Understanding

1. A _____ is a program in execution.
   (a) system program (b) application program
   (c) process (d) None of the Above
2. The current activity of a process is known as its _____. It is an indicator of the nature of the current activity in the process.
   (a) state (b) execution
   (c) both (a) & (b) (d) None of the Above
3. The operating system maintains the information about each process in a record or a data structure called as _____.
   (a) Process descriptor (b) Process control block
   (c) Process table (d) All of the Above

4. In Unix, which system call creates the new process?
   (a) new (b) create
   (c) fork (d) None of the Above
5. What is the ready state of a process?
   (a) When process is unable to run until some task has been completed.
   (b) When process is scheduled to run after some execution.
   (c) When process is using the CPU.
   (d) All of the Above
6. When two or more processes compete for the CPU at the same time then choice has to be made which process to allocate the CPU next and this procedure of determining the next process to be executed on the CPU is called _____ and the module of operating system that makes this decision is called as the _____.
   (a) CPU scheduling; Dispatcher
   (b) Process scheduling; Scheduler
   (c) Both (a) & (b)
   (d) None of the Above
7. Which of the following is not the state of a process?
   (a) New (b) Running
   (c) Waiting (d) Old
8. Which of the following do not belong to queues for processes?
   (a) Device Queue (b) Ready Queue
   (c) Job Queue (d) All of the Above
9. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the _____ queue.
   (a) Device (b) Ready
   (c) Job (d) All of the Above
10. What is a long-term scheduler?
    (a) It selects which process has to be brought into the ready queue.
    (b) It selects which process has to be executed next and allocates CPU.
    (c) It selects which process to remove from memory by swapping.
    (d) None of the Above
11. What is a short-term scheduler?
    (a) It selects which process has to be brought into the ready queue.
    (b) It selects which process has to be executed next and allocates CPU.
    (c) It selects which process to remove from memory by swapping.
    (d) None of the Above

12. What is a medium-term scheduler?
    (a) It selects which process has to be brought into the ready queue
    (b) It selects which process has to be executed next and allocates CPU
    (c) It selects which process to remove from memory by swapping
    (d) None of the Above
13. CPU switching from one process to another process is called as _____.
    (a) CPU switch (b) Dispatcher switch
    (c) Context switch (d) None of the Above
14. The multiple processes running on a system also need to intercommunicate in order to reciprocate some data or information. This kind of intercommunication between several processes is referred to as _____.
    (a) InterProcess Communication (IPC)
    (b) InterThread Communication (ITC)
    (c) InterCluster Communication (ICC)
    (d) None of the Above
15. A _____ is the fundamental unit of CPU utilization which is a program execution that uses the resources of a process.
    (a) process (b) thread
    (c) task (d) All of the Above
16. A thread is also called as _____.
    (a) Heavy Weight Process (HWP) (b) Light Weight Process (LWP)
    (c) Both (a) & (b) (d) None of the Above
17. In a _____ system, more than one processors are in the system and the threads are executed in parallel manner.
    (a) multicore (b) multithreaded
    (c) single core (a) None of the Above
18. The thread implemented at kernel level are known as _____ threads while the thread implemented at the user level are known as _____ threads.
    (a) process; cluster (b) single; multi
    (c) kernel; user (d) All of the Above
19. The model in which one kernel thread is mapped to many user-level threads is called as _____.
    (a) One to Many model (b) Many to One model
    (c) Many to Many model (d) One to One model
20. The model in which one user-level thread is mapped to many kernel level threads is called as _____.
    (a) One to Many model (b) Many to One model
    (c) Many to Many model (d) One to One model

21. The portion of the process scheduler in an operating system that dispatches processes is concerned with _____.
    (a) assigning ready processes to CPU
    (b) assigning ready processes to waiting queue
    (c) assigning running processes to blocked queue
    (d) All of the Above
22. In _____ model, one to one relationship of user level thread to the kernel level thread i.e. each user threads maps to a kernel thread.
    (a) One to Many (b) Many to One
    (c) Many to Many (d) One to One
23. Which of the following operating system not uses the kernel level thread?
    (a) Windows XP (b) MS-DOS
    (c) Linux (d) Solaris
24. A process can be _____.
    (a) single threaded (b) multithreaded
    (c) Both (a) and (b) (d) None of the Above
25. When more than one process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the_____, and algorithm it uses is called the scheduling _____.
    (a) dispatcher; algorithm (b) scheduler; algorithm
    (c) Both (a) and (b) (d) None of the Above
26. _____ scheduling is the process of selecting a process and allocating the processor to the selected process for execution.
    (a) CPU (b) Process
    (c) Both (a) and (b) (d) None of the Above
27. CPU scheduling is the basis of _____.
    (a) multi-processor systems
    (b) clustered systems
    (c) multiprogramming operating systems
    (d) None of the Above
28. The two steps of a process execution are _____.
    (a) I/O & OS Burst (b) CPU & I/O Burst
    (c) Memory & I/O Burst (d) OS & Memory Burst
29. The strategy of making processes that are logically runnable to be temporarily suspended is called as _____.
    (a) Non preemptive scheduling (b) Shortest job first
    (c) Preemptive scheduling (d) None of the Above

30. In the following cases non – preemptive scheduling occurs _____.
    (a) When a process goes from the running state to the waiting state
    (b) When a process switches from the running state to the ready state
    (c) When a process switches from the waiting state to the ready state
    (d) All of the Above
31. The FIFO algorithm _____.
    (a) first executes the job that came in last in the queue
    (b) first executes the job that needs minimal processor
    (c) first executes the job that has maximum processor needs
    (d) first executes the job that came in first in the queue
32. Waiting time is _____.
    (a) the total time spent in the ready queue
    (b) the total time in the blocked and waiting queues
    (c) the total time spent in the running queue
    (d) the total time from the completion till the submission of a process
33. Response time is _____.
    (a) the total time taken from the submission time till the completion time
    (b) the total time taken from submission time till the response is output
    (c) the total time taken from the submission time till the first response is produced
    (d) None of these
34. Turnaround time is _____.
    (a) the total waiting time for a process to finish execution
    (b) the total time spent in the running queue
    (c) the total time spent in the ready queue
    (d) the total time from the completion till the submission of a process
35. In priority scheduling algorithm, when a process arrives at the ready queue, its priority is compared with the priority of _____.
    (a) currently running process (b) all process
    (c) parent process (d) All of the Above
36. In multilevel feedback scheduling algorithm _____.
    (a) processes are not classified into groups
    (b) a process can move to a different classified ready queue
    (c) classification of ready queue is permanent
    (d) None of the Above
37. Which module gives control of the CPU to the process selected by the short-term scheduler?
    (a) Dispatcher (b) Interrupt
    (c) Scheduler (d) All of the Above

38. The interval from the time of submission of a process to the time of completion is termed as _____.
    (a) waiting time (b) response time
    (c) turnaround time (d) throughput
39. The time required to submit the process to CPU and finish it, is called _____.
    (a) turnaround time (b) waiting time
    (c) throughput (d) response time
40. _____ algorithm is non-preemptive only.
    (a) SJF (b) priority
    (c) RR (d) FCFS
41. _____ gives control of the CPU to the process selected by the short-term scheduler.
    (a) scheduler
    (b) dispatcher
    (c) context switch
    (d) None of the Above

Answers

1. (c)	2. (a)	3. (d)	4. (c)	5. (b)	6. (b)	7. (d)	8. (c)	9. (b)	10. (a)
11. (c)	12. (c)	13. (c)	14. (a)	15. (b)	16. (b)	17. (a)	18. (c)	19. (b)	20. (a)
21. (a)	22. (d)	23. (b)	24. (c)	25. (b)	26. (a)	27. (c)	28. (b)	29. (c)	30. (a)
31. (d)	32. (a)	33. (c)	34. (d)	35. (a)	36. (b)	37. (a)	38. (c)	39. (a)	40. (d)
41. (b)									

Practice Questions

Q.I Answer the following questions in short.

1. What is process? Define it.
2. What is scheduler? Enlist its type.
3. What is process scheduling?
4. What is IPC?
5. "Any executable file on a disk is called as a process" True/False – Justify.
6. State any two examples of One-to-One model.
7. Which resources are used when a thread is created?

Q.II Answer the following questions.

1. What is meant by process management? Which activities carried by process management?
2. What is meant by CPU scheduler? Explain the criteria for CPU scheduling.

3. Describe context switch diagrammatically,
4. Explain operations on processes with example.
5. Explain scheduling queue in detail.
6. What is co-operating process? Explain in brief two fundamental models of interprocess communication.
7. State the role of medium-term process scheduler.
8. Define short term scheduler with queuing diagram.
9. Compare user-level and kernel level threads.
10. Explain many-to-one model of multithreading
11. Define thread. Enlist its types.
12. Compare process and thread.
13. What are the benefits of multithreading programming? Explain any one of Multithreading models.
14. Explain the CPU scheduling algorithms with example: (i) RR (ii) SJF (iii) FCFS.
15. With the help of diagram describe CPU-I/O burst cycle.
16. Describe the term dispatcher in detail.
17. What is meant by Multilevel Queue (MLQ) Scheduling?
18. What is priority scheduling? Explain with example.
19. Write short note on: Multilevel feedback queue scheduling.
20. What is Preemptive and Non-preemptive scheduling? Compare them.
21. Consider the following snapshot of a system.

Job	Arrival Time	CPU Burst Time
1	0	7
2	1	2
3	2	5
4	3	4

Compute turnaround time using RR with quantum 3 and SJF (non-preemptive).

22. Consider following snapshot of the system.

Job	Arrival Time	CPU Burst Time
1	0.0	8
3	0.5	5
3	1.0	2

Compute average turnaround time and average wait time using (1) FCFS, (2) SJF (preemptive).

23. Assume that the following jobs are to be executed with one processor.

Job	Burst time	Priority	Arrival time
1	5	2	0
2	2	4 (lowest)	3
3	1	3	3
4	3	1 (highest)	4

Give the turnaround time for each job using

(i) Preemptive shortest job first algorithm.

(ii) Non-preemptive priority algorithm.

Assume that you have the following jobs to be executed with one processor.

Job	Arrival Time	Burst Time
1	0	12
2	2	3
3	5	8
4	5	5
5	7	6

Calculate the average turnaround fine and total wait time using preemptive SJF scheduling algorithm.

24. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

(i) The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5 all at time 0.

(ii) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-preemptive (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

(iii) What is the turnaround time of each process for each of the scheduling algorithms in part (i)?

(iv) What is the waiting time of each process for each of the scheduling algorithms in part (i)?

(v) Which of the schedules in part (i) results in the minimal average waiting time (over all processes).

25. Consider the following set of process with CPU burst time given in milliseconds.

Process	Burst time	Arrival time
P1	1	5
P2	0	7
P3	3	3
P4	2	10

Illustrate average time around time using

(i) RR (time quantum = 3)

(ii) Shortest remaining time first.

Q.III Define the terms.

1. CPU utilization
2. Response time
3. Turnaround time.
4. Dispatch latency time
5. Thread scheduling
6. Long-term scheduler
7. Message Passing model

Previous Exam Questions

Winter 2018

1. _____ scheduler loads the job from secondary storage to memory. **[1 M]**

(i) Long-term (ii) Short-term

(iii) Mid-term (iv) All of these

Ans. (i)

2. What is a dispatcher? **[2 M]**

Ans. Refer to Section 1.6.4.

3. List different types of scheduler. Explain any two types in detail. **[4 M]**

Ans. Refer to Section 1.2.2.

4. Consider the following set of processes with CPU burst time given in milliseconds: **[5 M]**

Process	Burst time	Arrival time
P1	5	1
P2	3	0
P3	2	2
P4	4	3
P5	8	2

Illustrate the execution of these processes using preemptive SJF. Calculate average turn around time and average waiting time.

Ans. Refer to Section 1.8.2.

5. Explain producer-consumer problem in detail. **[3 M]**

Ans. Refer to Section 1.4.2.

6. Define the following terms: **[2 M]**

(i) Response time

(ii) Waiting time

Ans. Refer to Section 1.7.

7. Write a short note on Process Control Block. **[4 M]**

Ans. Refer to Section 1.1.

8. State and explain different types of threads. **[3 M]**

Ans. Refer to Section 1.5.2.

Summer 2019

1. _____ is a module that gives control of the CPU to the process selected by the short-term scheduler. **[1 M]**

(a) Compiler (b) Dispatcher

(c) Processor (d) None of above

Ans. (b)

2. State the benefits of multithreaded programming. **[4 M]**

Ans. Refer to Section 1.5.3.

3. Consider the following set of processes, with the length of the CPU burst time given in :

A	3	2	4
B	3	0	2
C	6	4	1 (highest)
D	4	1	3

Compute the average turn around time and average waiting time for Non-preemptive shortest job first algorithm. **[4 M]**

Ans. Refer to Section 1.8.2.

4. Write a short note on process termination. **[3 M]**

Ans. Refer to Section 1.3.2.

5. Explain the different issues related to message passing system. **[4 M]**

Ans. Refer to Section 1.4.3.

■■■

2... Process Synchronization

Learning Objectives ...

- To understand problems in execution of Co-operating Processes that shares a Logical Address Space.
- To introduce the Critical Section Problem whose solutions can be used to ensure the consistency of Shared Data.
- To explore several tools like Semaphore to solve Process Synchronization Problems.
- To study Classic Problems of Synchronization such as the Bounded buffer problem, the Reader writer problem, the Dining philosopher problem.

2.1 BACKGROUND

- Concurrent execution of the statements means simultaneous or parallel execution of more than statements. Concurrent access often need access to shared data and shared resources.
- If there is no controlled access to shared data, it may result in data inconsistency. Maintaining data consistency requires mechanism to ensure the orderly execution of co-operating processes.
- A cooperating process is one that can affect or be affected by other processes in the system. Cooperating processes can either directly share a logical address space or be allowed to share data through files or messages.
- In this chapter, we would discuss the general issue of concurrency both within a single process and concurrency across processes.
- Process synchronization means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.
- Process synchronization also referred to as process co-ordination.

2.1.1 InterProcess Communication

- As processes frequently need to communicate with other processes therefore, there is a need for a well-structured communication, without using interrupts among processes.

- Consider producer - consumer problem in which two processes share a fixed-size buffer.
- One process produces information and puts it in the buffer, while the other process consumes information from the buffer.
- These processes do not take turns accessing the buffer, they both work concurrently. Herein lays the problem.
- What happens if the producer tries to put an item into a full buffer? What happens if the consumer tries to take an item from an empty buffer? In order to synchronize these processes, we will block the producer when the buffer is full, and we will block the consumer when the buffer is empty.

```
process producer
{
   while(true)
   {
      while(count == BUFFER_SIZE);
      ++count;
      buffer[in] = item;
      in = (in + 1) % BUFFER_SIZE;
   }
}
process consumer
{
   while(true)
   {
      while (count == 0);
      --count;
      item = buffer[out];
      out = (out - 1) % BUFFER_SIZE;
   }
}
```

- Assume `count` = 5 and both producer and consumer execute the statements `++count` and `--count`.
- What will be the Result? `count` could be set to 4, 5, or 6 (but only 5 is correct).
- Although both Producer and Consumer routines are correct separately, they may not function correctly when executed concurrently.
- We would arrive at this inconsistent state because we allowed both processes to manipulate the variable counter concurrently.

2.1.2 Race Condition [S-19]

- A situation, where several processes execute concurrently sharing some data and the result of execution depends on the particular order in which the access to shared data takes place is called race condition. Consider a simple example as shown in Fig. 2.1.

- Assume that two processes are being run, each process occasionally requests that line be printed on the single printer. Depending on scheduling of process P1 or P2, all the printout of process P1 may precede or follow the printout of process P2. But most likely the printout of each will be interspersed on the printer paper if they are interleaved.
- **Definition:** "A situation where multiple processes access and manipulate the same data concurrently and the outcome of the execution depends on the order in which the instructions execute".

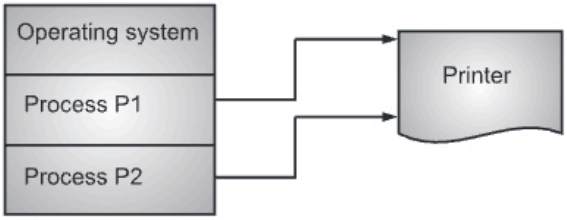


Fig. 2.1: Race Condition

- To guard against the race condition, we need to ensure that only one process at a time can be manipulating the common resource.
- To obtain this, we need some form of process synchronization.
- The process which completes its task should release the resource and the process which requires the resources and resources are not available (currently in use) would automatically switch to wait state.

2.1.3 Problems with Concurrency

- The processes that coexist in the memory at some time are called Concurrent processes. The concurrent processes may either be independent or cooperating.
- Concurrency is the interleaving of processes in time to give the appearance of simultaneous execution. The processes are said to be concurrent if they exist at the same time. Concurrent processes can function completely independent of one another, or they can be asynchronous, which means that they occasionally require synchronization and cooperation.
- Since, processes frequently need to communicate with other processes. Therefore, there is a need for well-structured communication mechanisms, without using interrupts, among processes. In operating systems, processes that are working together share some common storage (main memory, file, etc.) that each process can read and write.
- When two or more processes are reading or writing some shared data and the final result depends on who runs precisely when, are called race conditions.
- A race condition occurs when multiple processes read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.
- For example, consider two processes P1 and P2 share the global variable A. In the execution, process P1 updates variable A to the value 2 and at some point in its execution P2 updates A to the value 3. These two tasks are in a race to write variable A. In this, the process that updates last determines the final value of A.
- If two process P3 and P4 shares two global variables A and B with initial values A = 2 and B = 3. While in the execution, process P3 executes the assignment A = A + B and

process P4 executes the assignment B = A + B. Both the processes update different variables. The final values of the two variables depend on the order in which the two processes execute these two assignments. If process P3 executes its assignment statement first, then the final values are A = 5 and B = 8. If process P4 executes its assignment statement first, then the final values are A = 7 and B = 5.

- To avoid race conditions and inconsistent results, one must identify codes in critical sections in each process. The part of the program where the shared memory is accessed is called the critical section.
- Concurrency results in following issues:
  1. Problems with the sharing of global resources.
  2. Problems in allocating resources.
  3. It can result in programming issues because multiple processes update the same resource which is difficult to reproduce.
- Parallelism also introduces the issue that different processors may run at different speeds, but again this problem is mirrored in concurrency because different processes progress at different rates.
- Management and Design issues for concurrency are as follows:
  1. The OS must be able to keep track of the various processes.
  2. The OS must protect the data and physical resources of each process against unintended, interference by other processes.
  3. Functioning of a process and the output it produces must be independent of the speed at which its execution is carried out.
  4. The OS must allocate and deallocate various resources for each active process.

2.2 CRITICAL SECTION PROBLEM [S-19]

- The key to preventing trouble involved in shared storage is find some way to prohibit more than one process from reading and writing the shared data simultaneously. That part of the program where the shared memory is accessed is called the critical section.
- To avoid race conditions and flawed results, one must identify codes in critical sections in each process.

Definition Critical Section:

- Consider a system consisting of n co- operating processes {P1, ..., Pn}. Each process has a segment of code, called a Critical Section, in which the process may be changing common variables, updating a table, writing a file, and so on.

Definition Critical Section Problem:

- The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section.
- The execution of the critical section must be mutually exclusive. While one process is in its critical section, other process should not be allowed to enter its own critical section.

- Every process should take permission of operating system to enter its own critical section. There should be an exit point to the critical section. The balance code after the critical section is called as remainder section.
- The solution to critical section problem must satisfy following requirements:
  1. **Mutual exclusion:** When one process is in its critical section, other process should not be allowed to enter its critical section.
  2. **Progress:** If no process is in critical section and, there are some processes requesting to enter their critical sections, then all requesting processes except the one which is in its remainder section can participate in the selection decision. (The processes which are in their remainder section, have recently finished with their critical section. So the processes again should not be immediately given the chance to enter their critical section.)
  3. **Bounded waiting:** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
- Assume that each process executes at a nonzero speed. No assumption concerning relative speed of the n processes. There are a number of software and hardware solutions to solve critical section problem.

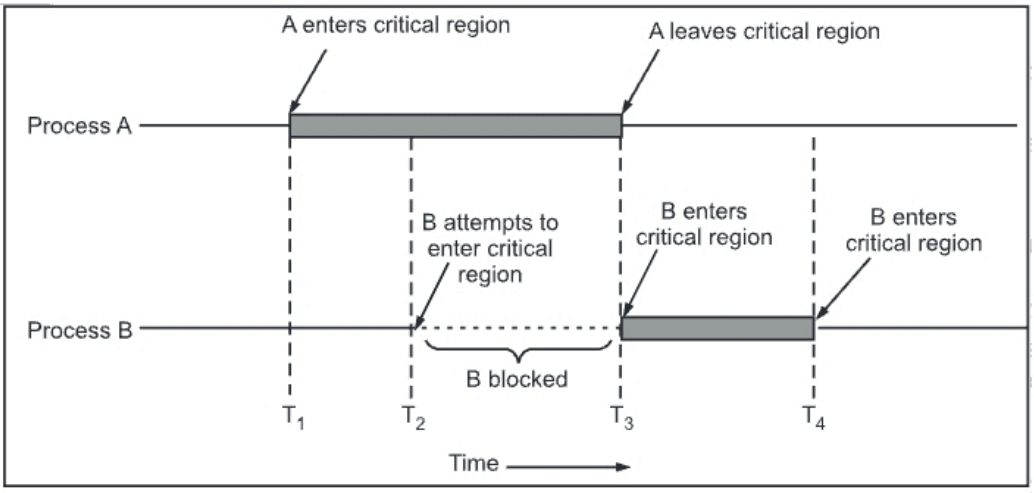


Fig. 2.2: Mutual Exclusion by using Critical Section

- The **General structure** of process Pi is as follows:

```
do
{
    Entry section
        Critical section
    Exit section
        Remainder section
} while(TRUE);
```

Approaches to handle Critical section:

- Two general approaches are used to handle critical sections in operating system:
  1. Preemptive kernels.
  2. Non-preemptive kernels.
- A preemptive kernel allows a process to be preempted while it is running in the kernel mode.
- A non-preemptive kernel does not allow a process running in kernel mode to be preempted. A kernel mode process will run until it exits kernel mode, blocks or voluntarily yields control of the CPU.
- Obviously, a non-preemptive kernel is essentially free from race conditions on kernel data structures, as only one process is active in the kernel at a time.
- We cannot say the same about preemptive kernels, so they must be carefully designed to ensure that shared kernel data are free from race conditions.
- Preemptive kernels are especially difficult to design for SMP architectures, since in these environments it is possible for two kernel mode processes to run simultaneously on different processors.
- Preemptive kernel is preferred over a non-preemptive one. A preemptive kernel is more suitable for real-time programming, as it will allow real-time process to preempt a process currently running in the kernel.
- Furthermore, a preemptive kernel may be more responsive, since there is less risk that a kernel-mode process will run for an arbitrarily long period before relinquishing the processor to waiting processes. But, this effect can be minimized by designing kernel code that does not behave in this way.
- Windows XP and Windows 2000 are non-preemptive kernels and the traditional UNIX kernel. Linux 2.6 onwards changed to preemptive kernel.

2.2.1 Peterson's Solution (For Two Processes)

- It is restricted to two processes that alternate execution between their critical and remainder section.
- Assumption that the LOAD and STORE instructions are atomic; that is, cannot be interrupted.
- The two processes Pi and Pj share two variables:
  - `int turn;`
  - `Boolean flag[2];`
- The variable `turn` indicates whose turn ~~it~~ is to enter the critical section. The `flag` array is used to indicate if a process is ready to enter the critical section. `flag[i] = true` implies that process Pi is ready.

```
while(true)
    {
        flag[i] = TRUE;
        turn = j;
        while(flag[j] && turn == j);
```

```
        //Critical section
        flag[i] = FALSE;
        //Remainder section
    }
```

Explanation:

- Turn is a common integer initialized to one or zero. (It can take values 0 or 1 only). This solution is for two processes Pi and Pj. Process Pi can enter its critical section if `turn = i`. Process Pj can enter its critical section if `turn =j`.
- To enter the critical section, process Pi first sets `flag[i]` to true and then sets turn to value j so that Pj can enter the critical section. If both processes try to enter at the same time, turn will be set to both i and j at the same time. Only one of these assignments will last; the other will occur but will be overwritten immediately. The eventual value of turn decides which of the two processes are allowed to enter its critical section first. This solution satisfies three requirements of critical section problem.
- **Mutual Exclusion:**
  - Each Pi enters its critical region only if either `flag[j]==false` or `turn==i`.
  - Each Pi enters its critical region with `flag[i]==true`.
  - Both Pi and Pj cannot enter their critical region at the same time.
- **Progress:**
  - Pi can be stuck only if either `flag[j]==true` and `turn==j`
- **Bounded Waiting:**
  - Pi will enter the critical region after at most one entry by Pj.
  - A process will not wait longer than one turn for entrance to the critical section. After giving priority to the other process, this process will run to completion and set its flag to false, thereby allowing the other process to enter critical section.

2.2.2 Bakery Algorithm

- Bakery algorithm devised by 'Lesile Lamport'. This is one of the solutions to the mutual exclusion problem for the general case of N process.
- This algorithm uses the concept which is used in bakeries to serve customers. When a customer enters in a bakery, he will receive a token in ascending order of number.
- While providing the service, the customer with smallest token number is served first.
  1. It is generalization for n processes.
  2. Each process has an id. Process ids are ordered.
  3. Before entering its critical section, a process receives a number. The holder of the smallest number enters its critical section.

Data structures:

- The common data structures are:

```
var choosing: array[0…n-1] of boolean,
    number: array[0…n-1] of integer;
```

- The structure for process Pi is as follows:

```
Repeat
   choosing[i] = true;
   number[i] = max(number[0],…, number[n-1])+1;
   choosing[i] = false;

for j:= 0 to n-1 do
begin
   while choosing[j] do skip;
   while ((number[j]≠0) && ((number[j],j) < (number[i], i) ))
   do skip;
end;
   //Critical section
number[i] =0;
   //Remainder section
Until false;
```

Explanation:

- Processes can enter into their critical section on FCFS (First Come First Serve) basis.
- The number of value of the process willing to enter is set to `max` of all plus 1.
- While one process is in critical section, other is not allowed to enter.
- After critical section of a process is over, its number value is set to zero.
- This algorithm does not ensure, if two processes are given same number or not. They will also be treated on FCFS basis.

2.3 SEMAPHORES [W-18, S-19]

- Semaphore is introduced by Dijkstra in 1965. In the design of operating system, it is required to allow concurrent execution of a collection of cooperating sequential processes.
- The two processes can cooperate through the means of signals, so that a process can be held at a specific point, until it receives a specific signal. For this purpose a special tool is used called **Semaphore.**
- The various hardware-based solutions to the critical region problem are complicated for application programmers to use.
- It can be solved using synchronization tool, called a semaphore that does not require busy waiting. It restricts the sequence or order of execution of the statements.

Definition:

- A semaphore is a protected variable, apart from initialization whose value can be accessed and altered only by the two atomic operations Wait() (or P) and Signal() (or V).

- The `signal()` & `wait()` operations are as follows and shown in Fig. 2.3.

```
wait(S)
{
   while (S <= 0);
   // no-operation
   S--;
}
signal(S)
{
   S++;
}
```

- **Wait (S):** It decrements the value of semaphore variable S. As soon as it becomes negative, wait operation completes. It must be executed indivisibly.
- **Signal (S):** It increments the value of semaphore variable S, as an indivisible operation.
- When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. It is less complicated to use.

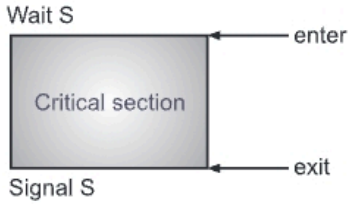


Fig. 2.3:Basic concept of Semaphore

2.3.1 Usage

- Semaphore can be of two types: Binary semaphore and Counting semaphore.

1. **Binary Semaphore:**

- Binary semaphore can range only between 0 and 1. Binary semaphore is also known as mutex lock that provides mutual exclusion.
- We can use binary semaphores to deal with critical section problem for multiple processes.
- The n processes share a semaphore mutex initialized to 1. Each process Pi is organized as shown below:

```
do
{
   Wait (mutex)
   //Critical section
   Signal (mutex)
   //Remainder section
}
while(TRUE);
```

2. **Counting Semaphore:**

- The value of counting semaphore can range over an unrestricted domain. It can be used to control access to a given resource consisting of finite number of instances.

- The semaphore is initialized to number of resources available. Each process that wishes to use resource performs wait() operation. When process releases resource, it performs a signal().
- Counting Semaphore S can be implemented in terms of binary semaphore.
- Following data structures used:

```
var    S1: binary-semaphore;
       S2: binary-semaphore;
       S3: binary-semaphore;
       C: integer;
```

- Initialization:

```
       S1 = S3 =1;
       S2 = 0;
       C = initial value of semaphore S;
```

- **Wait Operation can be defined as follows:**

```
wait(S3);
wait(S1);
C:= C-1;
if (C < 0)
{
   signal (S1);
   wait(S2);
}
else signal (S1);
signal (S3);
```

- **Signal Operation can be defined as follows:**

```
wait(S1);
C = C + 1;
if (C <= 0)
   signal (S2);
   signal (S1);
```

- Semaphore is also used to solve many synchronization problems. For example, consider process P1 and P2 are concurrently running and share a common semaphore synch initialized to zero. Consider statements of P1 and P2.

Process P1	Process P2
S1; signal (synch);	wait (synch); S2;

- The process P2 will execute S2 only after the P1 executes S1 and signal (synch).

Explanation:

- synch is initialized to zero.
- If S1 is executed first then signal (synch) operation will take place.
- By signal () operation synch = 1.
- Before S2 is executed wait (synch) operation will take place.
- This operation check value of synch <= 0.
- But as synch = 1.
- Next statement synch: = synch-1 will be executed.
- So S2 can be executed after S1 is executed.

Now consider the reverse situation:

- Let start with synch = 0
- If S2 is to be executed, then wait (synch) will be executed.
- Now as synch is 0, while synch<=0 condition is true and do skip will allow to continue.
- Therefore S2 cannot be started. S2 will have to wait till synch > 0 and that will be made only after S1 completes its execution.
- P2 cannot be started before P1 completed its execution. Actually processes which are made to wait, by operation wait must start after operation signal is executed.

2.3.2 Implementation

- The critical aspect of semaphore is that they are executed automatically. We must guarantee that no two processes can execute wait() and signal() operations on the semaphore at the same time. This is critical section problem.
- While one process is in its critical section, another process that tries to enter its critical section must loop continuously in the entry code. This continuous looping is problem in multiprogramming environment.
- In multiprogramming environment, only one CPU is shared among processes. A process which is waiting utilizes the CPU only to execute wait loop which is nothing but wastage of CPU cycles. This is called as problem of busy waiting.
- To overcome need of busy waiting; wait() and signal() operations are modified. With each semaphore, there is an associated waiting queue.
- Each entry in a waiting queue has two data items: value (of type integer) and pointer to next record in the list.
- C structure of semaphore is as follows:

```
typedef struct
{
   int value;
   struct process *list;
}semaphore;
```

Explanation:

- synch is initialized to zero.
- If S1 is executed first then signal (synch) operation will take place.
- By signal () operation synch = 1.
- Before S2 is executed wait (synch) operation will take place.
- This operation check value of synch <= 0.
- But as synch = 1.
- Next statement synch: = synch-1 will be executed.
- So S2 can be executed after S1 is executed.

Now consider the reverse situation:

- Let start with synch = 0
- If S2 is to be executed, then wait (synch) will be executed.
- Now as synch is 0, while synch<=0 condition is true and do skip will allow to continue.
- Therefore S2 cannot be started. S2 will have to wait till synch > 0 and that will be made only after S1 completes its execution.
- P2 cannot be started before P1 completed its execution. Actually processes which are made to wait, by operation wait must start after operation signal is executed.

2.3.2 Implementation

- The critical aspect of semaphore is that they are executed automatically. We must guarantee that no two processes can execute wait() and signal() operations on the semaphore at the same time. This is critical section problem.
- While one process is in its critical section, another process that tries to enter its critical section must loop continuously in the entry code. This continuous looping is problem in multiprogramming environment.
- In multiprogramming environment, only one CPU is shared among processes. A process which is waiting utilizes the CPU only to execute wait loop which is nothing but wastage of CPU cycles. This is called as problem of busy waiting.
- To overcome need of busy waiting; wait() and signal() operations are modified. With each semaphore, there is an associated waiting queue.
- Each entry in a waiting queue has two data items: value (of type integer) and pointer to next record in the list.
- C structure of semaphore is as follows:

```
typedef struct
{
   int value;
   struct process *list;
}semaphore;
```

- Two operations:
  1. **block:** Place the process invoking the operation on the appropriate waiting queue.
  2. **wakeup:** Remove one of processes in the waiting queue and place it in the ready queue.
- A process which is waiting is blocked and added to list of blocked processes.
- The process which is blocked by operation wait(), will be restarted by wakeup operation which is included in signal() operation.

Implementation of Wait:

```
wait(semaphore *s)
{
    s → value--;
    if (s → value < 0)
    {
        add this process to waiting queue
        block();
    }
}
```

Implementation of Signal:

```
Signal(semaphore *s)
{
    s→value++;
    if (s → value < = 0)
    {
        remove a process P from the waiting queue
        wakeup(P);
    }
}
```

- In a uniprocessor environment (only one CPU), critical section problem can be solved by inhibiting interrupts during execution of wait() and signal() (P or V) operation. So only currently running process executes until interrupts are re-enabled.
- In a multiprocessor environment, inhibiting interrupts will not work. Commands from different processes may be interleaved in some arbitrary way. If the hardware does not support any special instructions, we can provide any of the correct software solution for the critical –section problem.

2.4 CLASSIC PROBLEMS OF SYNCHRONIZATION

- In this section, number of synchronization problems can be solved by using semaphores.

2.4.1 The Bounded Buffer Problem

- Bounded buffer problem is commonly used to illustrate the power of synchronization primitives.

Producer and Consumer Problem:

Producer:

- Creates item and adds to the buffer.
- Do not want to overflow the buffer.

Consumer:

- Removes items from buffer (consumes it).
- Do not want to get ahead of producer.

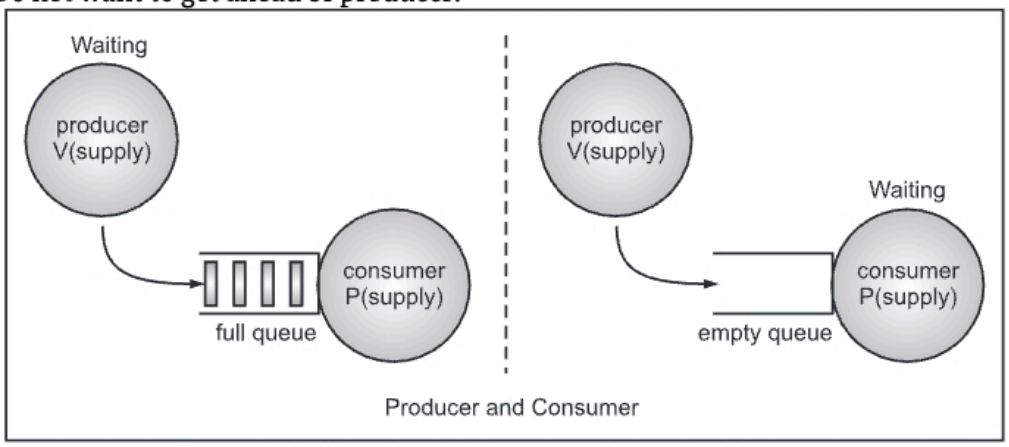


Fig. 2.4: Producer and Consumer Problem

- Bounded buffer assumes that there is a fixed buffer size.
- Consumer waits for new item, producer waits if buffer is full.
- Producers and consumers are much like Unix pipes.
- The bounded buffer problem can be handled using semaphores. `mutex` semaphore provides mutual exclusion. `empty` and `full` semaphores count number of empty and full buffers respectively.

```
Initialize Semaphore full: = 0,
Semaphore empty: =n
Semaphore mutex: = 1
```

Producer	Consumer

`while(true)`
`{`
`  …`
`  produce an item in nextp`
`  …`
`  wait (empty);`
`  wait (mutex);`
`  …`
`  add nextp to buffer`
`  …`
`  signal (mutex);`
`  signal (full);`
`}`
	`while(true)`
	`{`
	`  wait (full );`
	`  wait (mutex);`
	`  …`
	`  remove an item from buffer to nextc`
	`  …`
	`  signal (mutex);`
	`  signal (empty);`
	`  …`
	`  consume the next item in nextc`
	`  …`
	`}`

Explanation:

1. While one process is producing or consuming, `mutex` value is made `0` and after producing and consuming is over; it is set back to 1. Therefore, mutual exclusion is preserved.
2. For producer, if `empty=0`, then process cannot continue. It is made to wait and after producing is over; full is incremented.
3. For consumer, exactly reverse is the case. If `full=0`; then process cannot continue. At the end, empty is incremented.

2.4.2 The Reader Writer Problem

- Reader writer is another classical problem in concurrent programming. It basically resolves around a number of processes using a shared global data structure.
- A data set is shared among a number of concurrent processes. Some processes may want to read the data, whereas others may want to write the data.
- These two types of processes are referred as:
  1. **Readers**: These processes only read the data set; they do not perform any updates.
  2. **Writers**: These processes can both read and write.
- The Reader Writer problem allows multiple readers to read at the same time; but only one single writer can exclusively access the shared data at the same time. Such control can be achieved using semaphores.

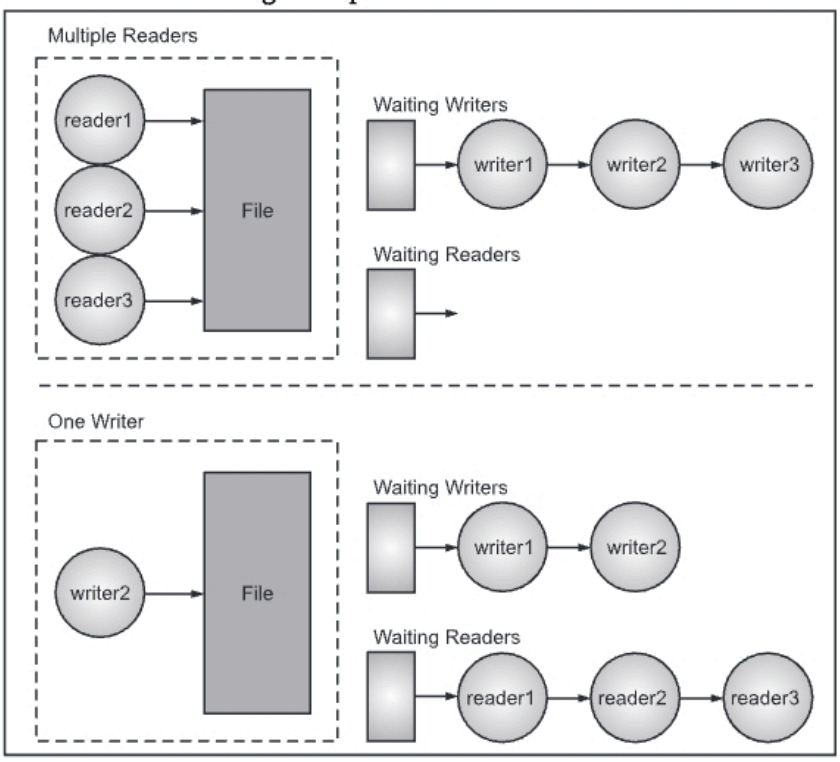


Fig. 2.5: Reader Writer Problem

- In the following code, `mutex` and `write` are common semaphores shared by all processes. `readcount` is a common integer which keeps count of concurrent reader processes.

```
Semaphore mutex=1
Semaphore write=1
integer readcount = 0
```

Reader	Writer

`while(true)`
`{`
`  wait (mutex)`
`  readcount ++;`
`  if (readcount == 1)`
`  wait (write);`
`  signal (mutex)`
`  // reading data is performed`
`  wait (mutex);`
`  readcount --;`
`  if (readcount == 0)`
`  signal (write);`
`  signal (mutex);`
`}`
	`while(true)`
	`{`
	`  wait (write);`
	`  // writing data is performed`
	`  signal (write);`
	`}`

Explanation:

Case 1: More than one process is reading

- We start with mutex =1 and write = 1 and readcount = 0.
  Suppose A is a process that wants to read. Process will execute following steps:
  1. wait (mutex) → mutex=0.
  2. readcount: = readcount +1 => readcount=1
     One process is reading
  3. If readcount = 1 then wait(write)
  4. signal (mutex) => mutex=1
  5. Reading of process A will begin
- Now some other process B want to read then it will also execute same set of steps:
  1. wait(mutex)=> mutex=0(Since Process A has set mutex to 1)
  2. readcount = readcount +1 => readcount=2
  3. If readcount = 1 then wait(write)this condition does not hold.
  4. signal(mutex) => mutex=1
  5. Reading of Process B start

  Such any number of processes can read simultaneously.

Case 2: Process A is reading and Process B is trying to write

- Process A will execute reader structure by which mutex=0 , write=0 and readcount=1 and process A will start reading.
- Now while process A is reading, if process B is trying to write, B will execute following steps:
  1. wait(write)
  2. Process can not proceed because write is made 0 by Process A, therefore Process B is blocked.

Case 3: Process A is writing and Process B is trying to write

- We start with write =1
- Process A will execute following steps:
  1. wait(write) =>write=0
  2. Writing started.
- Now suppose Process B also wants to write, it will execute following steps:
  1. wait(write)
  2. Process B is blocked because write value is made 0 by process A.

Case 4: Process A is writing and Process B is trying to read

- We start with write =1, mutex=1 and readcount =0
- If process A is writing:
  1. wait(write) => write=0
  2. Writing started
- Now process B will execute following steps to start reading:
  1. wait(mutex) =>mutex =0.
  2. readcount=readcount+1 => readcount=1
  3. If readcount =1 then wait(write)
- Process B is blocked because write value made 0 by process A.
- After reading or writing of process is over, mutex and write values are again set to proper values.

2.4.3 The Dining Philosopher Problem

- It is popular and classic synchronization problem of concurrency control. The problem can be stated as follows:
  - Consider five philosophers spend their lives alternating between thinking and eating.
  - They are seated around a circular table. In the center of table is a bowl of rice, and table is laid with five single chopsticks.
  - Each philosopher has access to the chopsticks at her left and right. In order to eat, a philosopher must be in possession of both chopsticks.
  - A philosopher may only pick up one chopstick at a time. Each philosopher attempts to pick up the left chopstick and then the right chopstick.
  - When a philosopher finished eating, he puts down both of chopsticks on the table and start thinking.

- Since the philosophers are sharing chopsticks, it is not possible for all of them to be eating at the same time.
- Now consider each philosopher as process and chopsticks are as resources. It is a simple presentation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner.

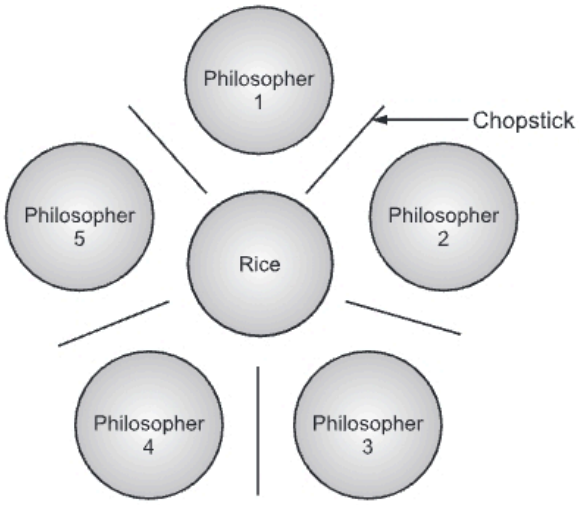


Fig. 2.6: Dining Philosopher Problem

- This problem can be solved as follows:

Shared Data:

- Bowl of rice (data set).
- Semaphore chopstick [5] initialized to 1.

The Structure of Philosopher is:

```
while (true)
{
   wait( chopstick[i] );
   wait( chopstick[ (i + 1) % 5] );
      // eat
   signal ( chopstick[i] );
   signal (chopstick[ (i + 1) % 5] );
      // think
}
```

- Although this solution guarantees that no two neighbours are eating simultaneously i.e. mutual exclusion, but it is not deadlock free. The deadlock can be handled by following solutions:
  1. Allow most four philosophers to be sitting simultaneously at the table.
  2. Allow a philosopher to pick up her chopsticks only if both chopsticks are available.
  3. Odd numbered philosopher picks up first her left chopsticks and then her right chopstick, whereas an even numbered philosopher picks up her right chopstick and then her left chopstick.

Summary

- A co-operating process is one that can affect or be affected by other processes in the system.
- Co-operating processes can either directly share a logical address space or be allowed to share data through files or messages.

- Race conditions occur in a situation where multiple processes access and manipulate the same data concurrently and the outcome of the execution depends on the order in which the instructions execute.
- To guard against the race condition, we need to ensure that only one process at a time can be manipulating the common resource. To obtain this, we need some form of process synchronization.
- To avoid race conditions and flawed results, one must identify codes in Critical Sections in each thread.
- The critical section problem is used to design a set of protocols which can ensure that the race condition among the processes will never arise.
- Consider a system consisting of n co-operating processes $\{P_1, \ldots, P_n\}$. Each process has a segment of code, called a critical section, in which the process may be changing common variable, updating a table, writing a file, and so on.
- The solution to critical section problem must satisfy following requirements: i.e., mutual exclusion, progress and bounded wait.
- Two general approaches are used to handle critical sections in operating system: Preemptive kernels and Non-preemptive kernels.
- Preemptive kernel is preferred over a non-preemptive one.
- Semaphore is introduced by Dijkstra in 1965. The various hardware-based solutions to the critical region problem are complicated for application programmers to use. It can be solved using synchronization tool, called a semaphore that does not require busy waiting.
- A semaphore is a protected variable, apart from initialization whose value can be accessed and altered only by the two atomic operations Wait() (i.e. P) and Signal() (or V).
- Semaphore can be of two type's binary semaphore and counting semaphore.
- Binary semaphore can range only between 0 and 1. Binary semaphore is also known as mutex lock that provides mutual exclusion.
- The value of counting semaphore can range over an unrestricted domain.
- Number of synchronization problems like bounded buffer problem, the reader writer problem, the dining philosopher problem etc. can be solved by using semaphores.

Check Your Understanding

1. _____ means sharing system resources by processes in such a way that, concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.
   (a) Process synchronization (b) Thread synchronization
   (c) Process virtualization (d) All of the Above
2. When several processes access the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called as _____.
   (a) Dynamic condition (b) Race condition
   (c) Essential condition (d) Critical condition

3. If a process is executing in its critical section, then no other processes can be executing in their critical section. This condition is called as _____.
   (a) Mutual exclusion (b) Critical exclusion
   (c) Synchronous exclusion (d) Asynchronous exclusion
4. Which one of the following is a synchronization tool?
   (a) thread (b) socket
   (c) semaphore (d) None of the Above
5. Process synchronization can be done on _____.
   (a) hardware level (b) software level
   (c) Both (a) and (b) (d) None of the Above
6. _____ are used for signaling among processes and can be readily used to enforce a mutual exclusion discipline.
   (a) Semaphores (b) Messages
   (c) Monitors (d) None of the Above
7. Semaphores provide a primitive yet powerful and flexible tool for enforcing mutual exclusion and for coordinating processes called as _____ .
   (a) Binary semaphore (b) Monitor
   (c) Counting semaphore (d) None of the Above
8. Concurrent access to shared data may result in _____.
   (a) data consistency (b) data insecurity
   (c) data inconsistency (d) None of the Above
9. The shared data (resources) is called _____ and the portion of the program, that uses this critical resource, is called as _____.
   (a) Critical resource; Critical section (b) Process resource; Process section
   (c) Thread resource; Thread section (d) Logical resource; Logical section
10. In the Bakery algorithm, to solve the critical section problem _____.
   (a) each process is put into a queue and picked up in an ordered manner
   (b) each process receives a number (may or may not be unique) and the one with the lowest number is served next
   (c) each process gets a unique number and the one with the highest number is served next
   (d) each process gets a unique number and the one with the lowest number is served next

11. Bounded waiting indicates that there exists a bound on the number of times a process is allowed to enter its critical section _____.
    (a) after a process has made a request to enter its critical section and before the request is granted
    (b) when another process is in its critical section
    (c) before a process has made a request to enter its critical section
    (d) When a process has not made a request
12. The bounded buffer problem is also known as _____.
    (a) The Dining Writer problem (b) The Producer-Consumer problem
    (c) The Reader-Writer problem (d) The Dining Philosopher problem
13. The Dining Philosopher problem is _____.
    (a) Page Fault free solution (b) Starvation free solution
    (c) Deadlock free solution (d) All of the above
14. In _____ semaphore, the integer value can range over an unrestricted domain.
    (a) binary (b) counting
    (c) bounded (d) unbounded
15. Semaphores are mostly used to implement:
    (a) System protection (b) System Calls
    (c) IPC mechanisms (d) None of the above
16. The Dining Philosopher problem will occur in case of _____.
    (a) 4 Philosophers and 5 chopsticks (b) 5 Philosophers and 5 chopsticks
    (c) 6 Philosophers and 5 chopsticks (d) 3 Philosophers and 5 chopsticks
17. Basically the signal operation of the semaphore works on the basic _____ system call.
    (a) wakeup() (b) getup()
    (c) start() (d) continue()
18. The two types of semaphore are _____.
    (a) Mutex and Counting (b) Binary and Counting
    (c) Binary and Decimal (d) Counting and Octal
19. In the bounded buffer problem _____.
    (a) There is only one buffer
    (b) There are n buffers (n being greater than one but finite)
    (c) There are infinite buffers
    (d) The buffer size is bounded

20. If no process is suspended, the signal operation _____.
    (a) Puts the system into a deadlock state
    (b) Suspends some default process execution
    (c) Nothing happens
    (d) The output is unpredictable

Answers

1. (a)	2. (b)	3. (a)	4. (c)	5. (c)	6. (a)	7. (b)	8. (c)	9. (a)	10. (b)
11. (a)	12. (b)	13. (c)	14. (b)	15. (c)	16. (b)	17. (a)	18. (b)	19. (b)	20. (c)

Practice Questions

Q.I Answer the following questions in short.

1. What is process synchronization?
2. What is critical section problem?
3. What is purpose of semaphores?
4. What is Bounded buffer problem?
5. Which two standard atomic operations can access semaphore value?

Q.II Answer the following questions.

1. Define the term semaphore. Enlist its types in detail.
2. Which are drawbacks of solution of critical section problem?
3. Write a short note on: (i) Semaphore (ii) Race condition.
4. Which are the types of semaphore? Explain with its usage and implementation.
5. How semaphore can be used to solve the Dining Philosopher problem of concurrency control?
6. Which three requirements must be satisfied while designing a solution to the critical section problem? Explain each in detail.
7. Describe Peterson's solution to solve critical section problem.
8. Explain the role of wait( ) and signal( ) operations used in semaphores.
9. Discuss the following problems in concurrent programming and explain the use of semaphore in these problems:
   (i) The producer-consumer problem
   (ii) The reader-writer problem
   (iii) The Dining Philosopher problem
10. Describe Critical Section problem in detail.

Q.III Define the terms.

1. Race condition.
2. Bounded wait.
3. Critical section.
4. Binary semaphore.
5. Counting Semaphore.

Previous Exam Questions

Winter 2018

1. Write a short note on semaphore. **[4 M]**

Ans. Refer to Section 2.3.

Summer 2019

1. What is a semaphore? Discuss its types. **[4 M]**

Ans. Refer to Section 2.3.

2. Discuss the requirements of the critical problem solution. **[3 M]**

Ans. Refer to Section 2.2.

3. What is race condition?

Ans. Refer to Section 2.1.2. **[1 M]**

■■■

3... Deadlocks

Learning Objectives...

- To learn about the concept system model.
- To understand concept of Deadlock and conditions for Deadlock to occur.
- To learn methods for handling Deadlocks like Deadlock Prevention, Deadlock Avoidance and Deadlock Detection.
- To know how to recover from Deadlock.

3.1 BACKGROUND

- In multiprogramming environment, Process A needs the resource which is held by other Process B. On the other hand, the other Process B also needs the resource held by the Process A. If both processes are waiting for each other to release the resource, then the situation is called a deadlock as shown in Fig. 3.1.
- None of the processes can run, none of them can release any resources, and none of them can be awakened. It is important to note that the number of processes, kind of resources possessed and requested are unimportant.

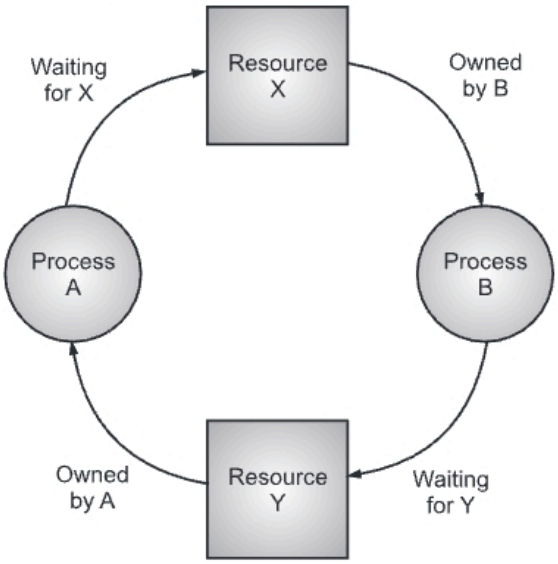


Fig. 3.1: Situation of Deadlock

- The resources may be either physical or logical. Examples of physical resources are Printers, Tape Drivers, Memory Space, and CPU Cycles. Examples of logical resources are Files, Semaphores and Monitors.
- A deadlock is a situation in which some processes wait for each other's actions indefinitely. It can arise in a synchronization situa-tion, e.g., when processes wait for other processes to send them messages, or to release resources that they need. In real life, dead-locks can arise when two persons wait for phone calls from one another.
- In this chapter, we would discuss the technique for dealing with deadlocks. These techniques are deadlock prevention, deadlock avoidance and deadlock detection and recovery.

3.2 SYSTEM MODEL

- Any system consists of finite number of resources which are distributed among the processes. For example, if a system has 5 printers, then the resource is printer and it has 5 instances.
- If a Process P requires an instance of a resource type printer, then it is allocated to Process P (if free) to satisfy the request.
- If the instances of printer are not free; means all the printers are allocated to other processes, then Process P has to wait until any other process release it. So a process must request a resource before using it, and must always release the resource after using it.
- A process may request any number of resources to carry out its task. A process cannot request the number of resources greater than the number of resources present in the system. For example, if the system having 3 printers and the process requests 4 printers, then request is not granted.
- Under the normal mode of operation, any resource is utilized by a process in following manner:
  - **Request:** A process request for a resource. If it is available, it is immediately granted; otherwise the process must wait for it.
  - **Use:** After a request is granted, process uses the resource.
  - **Release:** The Process releases the resource after it is used.
- The following Fig. 3.2 shows the above sequence.
  1. Process P1 is requesting resource R1.
  2. If Resource R1 is free, then request made by P1 is granted. Process P1 use resource R1.
  3. Process P1 releases resource R1 and may be used by P2.

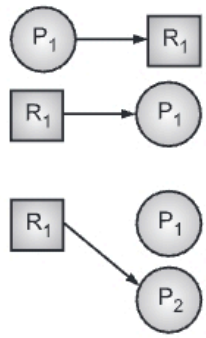


Fig. 3.2: System Model

Definition of Deadlock:

- A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set.
- For example, there are two tape drives, in some systems. There are two processes, each holding one more tape. Now each of the process needs one more tape drive. Now both the processes are waiting for each other to release a tape drive. This is a deadlock situation.

3.3 DEADLOCK CHARACTERIZATION

- In a deadlock, processes never finish executing and system resources are tied-up, preventing other jobs from starting.
- Before we discuss the various methods for dealing with the deadlock problems, we shall describe features that characterize deadlocks.

3.3.1 Necessary Conditions [S-19]

- Coffman (1971) identified **four conditions** that must hold simultaneously for there to be a deadlock.
- These conditions are explained below:

1. **Mutual Exclusion Condition**: The resources involved are non-shareable. Only one process at a time can use a resource.

   Explanation: At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. **Hold and Wait Condition:** a process holding at least one resource is waiting to acquire additional resources held by other processes.

   Explanation: There must be a process that is holding a resource already allocated to it; while waiting for additional resources that are currently being held by other processes.

3. **No-Preemptive Condition:** Resources already allocated to a process cannot be preempted.

   Explanation: Resources cannot be removed from the processes until its completion or released voluntarily by the process holding it.

4. **Circular Wait Condition:** The processes in the system form a circular list or chain; where each process in the list is waiting for a resource held by the next process in the list.

Explanation:

- A set $\{P_0, P_1, ..., P_n\}$ of waiting processes must exist such that P_0 is waiting for resource held by P_1, P_1 is waiting for a resource held by P_2, ..., P_{n-1} is waiting for a resource held by P_n and P_n is waiting for a resource held by P_0.
- It is necessary to understand that all four conditions must be satisfied simultaneously for a deadlock to occur. If any one of them does not occur, a deadlock can be avoided.

- A deadlock is a situation where a process or a set of processes is blocked, waiting for some resource that is held by some other waiting processes. Consider an example of the traffic on road as shown in Fig. 3.3.

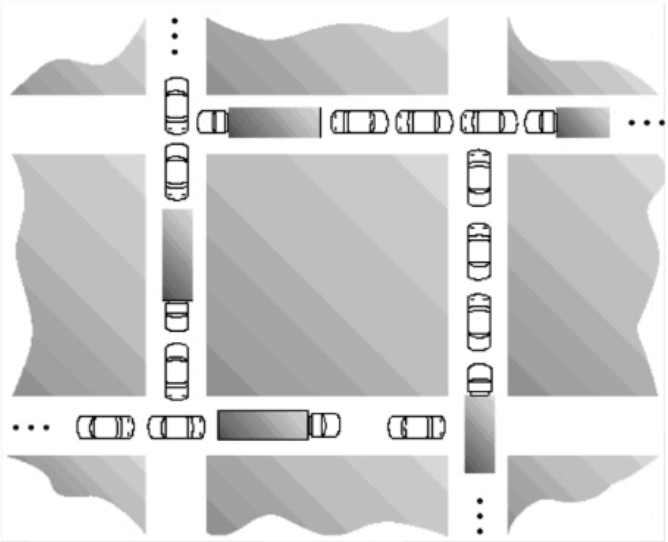


Fig. 3.3: Traffic Deadlock

- Consider each section of the street as a resource.
  1. **Mutual exclusion** condition applies, since only one vehicle can be on a section of the street at a time.
  2. **Hold-and-wait** condition applies, since each vehicle is occupying a section of the street, and waiting to move on to the next section of the street.
  3. **No-preemptive** condition applies, since a section of the street that is occupied by a vehicle cannot be taken away from it.
  4. **Circular wait** condition applies, since each vehicle is waiting on the next vehicle to move. That is, each vehicle in the traffic is waiting for a section of street held by the next vehicle in the traffic.
- The simple rule to avoid traffic deadlock is that a vehicle should only enter an intersection if it is assured that it will not have to stop inside the intersection.
- It is not possible to have a deadlock involving only one single process. The deadlock involves a circular "hold-and-wait" condition between two or more processes, so "one" process cannot hold a resource, yet be waiting for another resource that it is holding.
- In addition, deadlock is not possible between two threads in a process, because it is the process that holds resources, not the thread that is, each thread has access to the resources held by the process.

3.3.2 Resource Allocation Graph [S-19]

- It is nothing but directed graph, which is used to describe resources allocated to a particular process in a system.

- The graph is made up of:
  - A set of processes $P = \{P_1, P_2,...,P_n\}$. This set contains all the processes in the system.
  - A set of resources $R = \{R_1, R_2, ..., R_n\}$. This contains all the resources present in the system.
  - A set of directed edges.
  - A directed edge from a process P_i to a resource R_j. ($P_i \rightarrow R_j$) is called as a Request Edge.
  - A directed edge from a resource R_j to a process P_i ($R_j \rightarrow P_i$) is called as Allocation Edge or an Assignment Edge.

Fig. 3.4: Edges in Resource Allocation Graph

Conventions:

1. Process is represented by circle.
2. Resource is represented by rectangle.
3. Number of resources of similar type (instances) is represented by dots.
4. Process may be holding a resource or may be requesting for a resource.
5. When process P_i requests an instance of resource type R_j, a request edge is inserted in the resource-allocation graph.
6. When this request can be fulfilled, the request edge is instantly transformed to an assignment edge.
7. When the process no longer needs access to the resource, it releases the resource; as result, assignment edge is removed.
8. If the graph contains no cycle, then no process in the system is deadlocked.
9. If graph contains cycle, then deadlock may exist.

- The resource allocation graph as shown in Fig. 3.5 depicts the following situation:
  - $P = (P_1, P_2, P_3)$
  - $R = (R_1, R_2, R_3)$
  - E = Request Edge = $\{P_1 \rightarrow R_1, P_2 \rightarrow R_3\}$
  - Assignment Edge = $\{R_1 \rightarrow P_2, R_3 \rightarrow P_3, R_2 \rightarrow P_1, R_2 \rightarrow P_2\}$

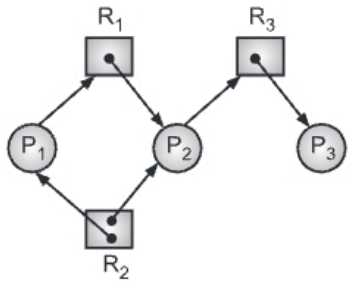


Fig. 3.5: Resource Allocation Graph

Process States:

1. Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
2. Process P_2 is holding an instance of resource type R_1 and is waiting for an instance of resource type R_3.
3. Process P_3 is holding an instance of resource type R_3.

- As graph contains no cycle, system is deadlock free.
- Consider Fig. 3.6 (a) in which Process P_3 requests an instance of resource type R_2. At this point, two minimal cycles exist in the system:
  $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
  $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$

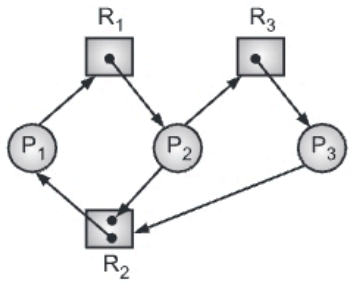


Fig. 3.6 (a): Resource Allocation Graph with Deadlock

- Processes P_1, P_2 and P_3 are deadlocked. Process P_2 is waiting for the resource R_3, which is held by process P_3. Process P_3 is waiting for either process P_1 or P_2 to release resource R_2. Also, process P_1 is waiting for process P_2 to release resource R_1. So system is in deadlock.
- Consider resource allocation graph shown in Fig. 3.6 (b) which contains cycle, $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_1$. However there is no deadlock. Process P_3 may release the resource R_2. Then that resource R_2 can allocated to P_2, breaking the cycle.

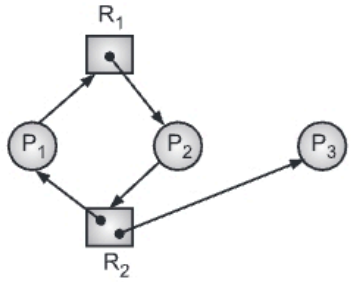


Fig. 3.6 (b): Resource Allocation Graph with Cycle but no Deadlock

- If graph contains cycle, then check the following:
  - If there is one instance per resource type, then there is a deadlock.
  - If resource type has multiple instances, then there may or may not be deadlock.
- There are following two common methods for dealing with the deadlock situation:
  1. Don't allow the system to enter a deadlock state.
  2. Allow the system to enter a deadlock state and recover.
- The deadlock handling techniques are deadlock prevention, deadlock avoidance, and deadlock detection and recovery.
- Deadlock prevention algorithm disallows one of four necessary conditions for deadlock.
- Deadlock avoidance algorithm do not grant resource request if resource-allocation have potential have led to deadlock.
- Deadlock Detection and Recovery algorithm always grants resource request when possible.
  It periodically checks for a deadlock. If deadlock exists, recover from it.
- The deadlock prevention and deadlock avoidance approaches ensure that deadlocks cannot occur, by not allowing the conditions for deadlocks to hold simultaneously.

3.4 METHODS FOR HANDLING DEADLOCK

3.4.1 Deadlock Prevention

- We have already discussed in Section 3.2 each of four necessary conditions hold, for occurrence of deadlock. A deadlock can be prevented if either of four conditions is prevented from taking place.
- By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.
- Let us consider each of them separately.

1. **Mutual Exclusion:** The mutual exclusion condition must hold for non-sharable resources. That is, several processes cannot simultaneously share a single resource. This condition is difficult to eliminate because some resources, such as the tape drive and printer, are inherently non-shareable. Note that shareable resources like read-only-file do not require mutually exclusive access and thus cannot be involved in deadlock. Processes never need to wait for a sharable resource. So try to use maximum resources in sharable mode.
2. **Hold and Wait:** In order to prevent this condition to hold, we must guarantee that, a process holding some resources does not request for more resources. This can be achieved by either of holding following protocols:
   (i) The first protocol is that a process request be granted all of the resources it needs at once, prior to execution. This strategy requires that all of the resources a process will need must be requested at once. The system must grant resources on "all or none" basis. If the complete set of resources needed by a process is not currently available, then the process must wait until the complete set is available. While the process waits, however, it may not hold any resources. Thus the "wait for" condition is denied and deadlocks simply cannot occur.

   Drawbacks:

   (a) This strategy can lead to serious waste of resources. For example, a program requiring ten tape drives must request and receive all ten drives before it begins executing. If the program needs only one tape drive to begin execution and then does not need the remaining tape drives for several hours. Then substantial computer resources (9 tape drives) will sit idle for several hours.
   (b) It can cause indefinite postponement (starvation). Since not all the required resources may become available at once.
   (c) As well as process must be able to specify all the future requirements of the resources. This may not be possible.

   (ii) Second protocol is to disallow a process from requesting resources whenever it has previously allocated resources. A process will first request the source it requires, use it and release it. Then the process will request for second source it need, use it then release it and so on. By this solution also hold and wait conditions does not take place. But drawback of this method is a process may have to wait for a long time, in case of popular resource. A process may suffer starvation.

3. **No Preemption:** This condition can also be avoided by either of two solutions:
   (i) In the first solution, if process P_1 is holding some resource and waiting for few more resources. If the resources for which P_1 is waiting are currently not available, then P_1 release all its resources. All released resources are added to request list of process P_1. Now P_1will restart only when it will get all the resource from its request list (i.e. previously allocated and new resources together). Thus no preemption is avoided when P_1 is preempted.
   (ii) In the second solution, if a process P, holding some resources and waiting for some additional resources. If they are available then given to process P. If they are held by other process, that is waiting for few more resources. Then this other process will be preempted and the resources will be given to Process P. In simple words, in first solution, Process P itself forcefully preempted and in second solution, it forces other process to preempt.

   In both ways, the forceful preemption of process may lead to an inconsistent state of system. When a process release resources, the process may lose all its work to that point. One serious consequence of both strategies is the possibility of indefinite postponement (starvation). A process might be held off indefinitely as it repeatedly requests and releases the same resources.
4. **Circular Wait**: The circular wait can be denied by imposing a total ordering on all of the resource types than forcing. This strategy impose that each process requests resources in a numerical order (increasing or decreasing) of enumeration. If several instances of same resource type are needed, a single request for all of them is issued. With this rule, the resource allocation graph can never have a cycle. For example, provide a global numbering of all the resources, as shown below:

Number	Resource
1	Card reader
2	Printer
3	Plotter
4	Tape drive
5	Card punch

   (i) Now the rule is this; processes can request resources whenever they want to, but all requests must be made in numerical order.
   (ii) A process may request first printer and then a tape drive (order: 2, 4), but it may not request first a plotter and then a printer (order: 3, 2). The problem with this strategy is that it may be impossible to find an ordering that satisfies everyone.

3.5 DEADLOCK AVOIDANCE

- If detail information about the processes and resources is available, then it is possible to avoid deadlock. For example, which process will require which resources, possibly in what sequence etc. This information may help to decide the sequence in which the processes can be executed to avoid deadlock.

- Each request can be analyzed on the basis of number of resources currently available, currently allocated and future request which may come from other process. From this information, system can decide whether or not a process should wait.
- The deadlock avoidance algorithm dynamically examines the resource-allocation state from the available information to ensure that there is no circular wait. The resource-allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

3.5.1 Safe State

- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.
- A system is in a safe state; if there exist a safe sequence.
- **Definition of Safe Sequence**: A sequence of processes <P_1, P_2, ..., P_n> is a safe sequence for current allocation state if, for each process P_i, the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j have finished with j <i.
- If resources that P_i needs are not available, then P_i can wait until all P_j have finished. When P_j terminates, P_i will obtain its resources and completes the task. When P_i terminates, P_{i+1} obtain its resources and so on. If no such sequence exists, then system is in unsafe state. A system without safe sequence is unsafe.

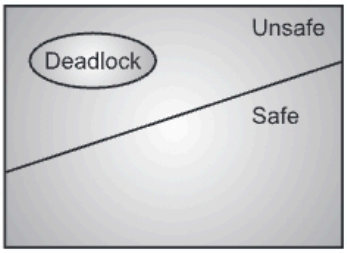


Fig. 3.7: Safe, Unsafe and Deadlock State Spaces

- A safe state is not deadlocked state. Conversely deadlocked state is unsafe state. Thus unsafe system may lead to deadlock as shown in Fig. 3.7.

3.5.2 Resource-Allocation Graph Algorithm

- This algorithm uses a variant of Resource Allocation Graph for single instance of resource type.
- In this algorithm, apart from all set and edges from resource allocation graph, one more edge is used. This edge is called as **a claim edge**. A claim edge (P_i, R_j) indicates that process P_i may request resource R_j, sometime in future.
- A claim edge is represented by a dashed line. All the claim edges must appear in graph before P_i starts execution.
- When process P_i requests for a resource R_j, then the claim edge<P_i, R_j>is converted to request edge <P_i, R_j> (dashed line converted to regular line).
- A resource is granted only when, conversion of request edge <P_i, R_j> to assignment edge <R_j, P_i> does not form any cycle in the graph.
- If graph contains cycle, then system is in unsafe state.
- Consider the following example, in Fig.3.8, Processes P_1 and P_2 claim for resources before execution.

- In Fig. 3.8 (a), Process P_1 requests Resource R_A . So claim edge <P_1, R_A > is converted into request edge <P_1, R_A >.
- In Fig. 3.8 (b), request made by Process P_1 for Resource R_A is granted. So request edge <P_1, R_A> is converted into assignment edge <R_A, P_1>.
- In Fig. 3.8 (c), Process P_2 requests Resource R_B. So claim edge <P_2, R_B> is converted into request edge <P_2, R_B>.
- In Fig. 3.8 (d), if request made by Process P_2 for Resource R_B is granted, it forms cycle in the graph. So allocation is denied, even if R_B is available.

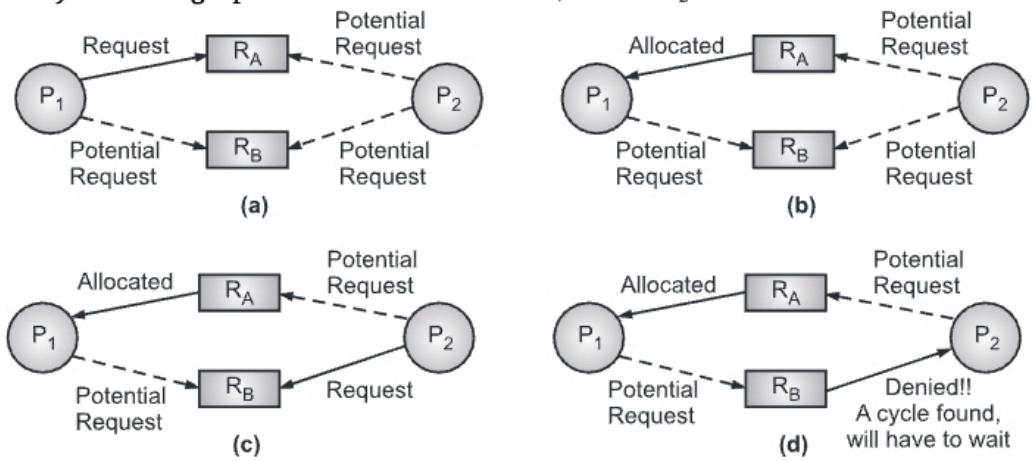


Fig. 3.8: Resource Allocation Graph for Deadlock Avoidance

3.5.3 Banker's Algorithm [W-18, S-19]

- The avoidance algorithm used for several instances of the resources is called as Banker's algorithm developed by Dijkstra (1965).
- It has been named as Banker's algorithm, because this algorithm could be used in a banking system to ensure that the bank never allocates its available cash in such way that it can no longer satisfy the requirement of all customers (means available cash is always > customer requirement).
- Each process must a priori claim for maximum use. When a process requests a resource, it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.
- Several data structures are used to implement Banker's algorithm. Let n be number of processes and m be number of resource types.

1. **Available:** A vector of length m indicating the number of available resources of each type. If Available [j] = k means there are k instances of resource type R_j available.
2. **Max:** A n $\times$ m matrix defining the maximum demand of each process. If Max [i, j] = k, then process P_i may request at most k instances of resource type R_j.
3. **Allocation:** A n $\times$ m matrix defining number of resources of each type currently allocated to each process. If Allocation [i,j] = k then process P_i is currently allocated k instances of resource type R_j.

4. **Need:** A $n \times m$ matrix indicating the remaining resource need of each process. If Need [i, j] = k, then process P_i may need k more instances resource type R_j, in order to complete its task.

3.5.3.1 Safety Algorithm

- Let **Work** and **Finish** be vectors of length m and n, respectively.
  1. Initialize Work = Available
     Finish [i] = false for i = 0, 1, ..., n – 1.
  2. Find such that both:
     (i) Finish [i] = = false
     (ii) $Need_i \leq$ Work
     If no such i exists, go to step 4.
  3. Work = Work + $Allocation_i$, Finish[i] = true, go to step 2.
  4. If Finish [i] == true for all i, then the system is in a safe state.
- This algorithm requires order of $m \times n^2$ operation to check whether system is in safe state.

3.5.3.2 Resource – Request Algorithm

- This algorithm determines if requests can be safely granted.
- Let $Request_i$ be request vector for process P_i. If $Request_i$ [j] = k then process P_i wants k instances of resource type R_j.
  1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since Process has exceeded its maximum claim.
  2. If $Request_i \leq$ Available, go to step 3.Otherwise P_i must wait, since resources are not available.
  3. Pretend to allocate requested resources to P_i by modifying the state as follows:
     Available = Available–$Request_i$;
     $Allocation_i = Allocation_i + Request_i$;
     $Need_i = Need_i - Request_i$
     If safe state$\Rightarrow$ the resources are allocated to P_i.
     If unsafe state $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored.
- **Example:** Consider the following snapshot of system, A, B, C and D are the resource type.

	Allocation			
	A	B	C	D
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

	MAX			
	A	B	C	D
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

Available			
A	B	C	D
1	5	2	0

- Answer the following questions using Banker's Algorithm:
  1. What are the contents of need array?
  2. Is the system in safe state? If yes give the safe sequence.
  3. If a request from process P_i arrives for (0,4,2,0) can it be granted immediately?

  Solution:

  1. The content of need array is as follows:

     Need[i][j] = Max[i][j] – Allocation[i][j]

	Max			
	A	**B**	**C**	**D**
P_0	0	0	1	2
P_1	1	7	5	0
P_2	2	3	5	6
P_3	0	6	5	2
P_4	0	6	5	6

–

	Allocation			
	A	**B**	**C**	**D**
P_0	0	0	1	2
P_1	1	0	0	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

=

	Need			
	A	**B**	**C**	**D**
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

  2. To check whether system is in safe state, Banker's safety algorithm is used.

Initialize Finish = { False, False, False, False, False,} Work = Available = {1,5,2,0}
Let i=0 Finish[0] == False Check Need(P_0) <= Work {0,0,0,0} <= {1,5,2,0}=>true Therefore, Process P_0 can be granted required resources. So Work = Work + Allocation[P_0] (After P_0 finishes its task, it will release resources) Work = {1,5,2,0} + {0,0,1,2} = {1,5,3,2} Finish = { True, False, False, False, False} Safe Sequence = {P_0}
Let i=1 Finish[1] == False Check Need(P_1) <= Work {0,7,5,0} <= {1,5,3,2} =>False Therefore, Process P_1can not be given resources. P_1 must wait.
Let i=2 Finish[2] == False Check Need(P_2) <= Work {1,0,0,2} <= {1,5,3,2} =>true Therefore, Process P_2 can be granted required resources. So Work = Work + Allocation[P_2] (After P_2 finishes its task, it will release resources) Work = {1,5,3,2} + {1,3,5,4} = {2,8,8,6} Finish = { True, False, True, False, False} Safe Sequence ={P_0,P_2}

Let i=3 Finish[3] == False Check Need(P_3) <= Work {0,0,2,0} <= {2,8,8,6}=>true Therefore, Process P_3 can be granted required resources. So Work = Work + Allocation[P3] (After P_3 finishes its task, it will release resources) Work = {2,8,8,6} + {0,6,3,2} = {2,14,11,8} Finish = { True, False, True, True, False} Safe Sequence ={P_0,P_2,P_3}
Let i=4 Finish[4] == False Check Need(P_4) <= Work {0,6,4,2} <= {2,14,11,8}=>true Therefore, Process P_4 can be granted required resources. So Work = Work + Allocation[P_4] (After P_4 finishes its task, it will release resources) Work = {2,14,11,8} + {0,0,1,4} = {2,14,12,12} Finish = { True, False, True, True, True} Safe Sequence ={P_0,P_2,P_3,P_4}
Again Check Finish[1] == False Check Need(P_1) <= Work {0,7,5,0} <= {2,14,12,12}=>true Therefore, Process P_1 can be granted required resources. So Work = Work + Allocation[P_1] (After P_1 finishes its task, it will release resources) Work = {2,14,12,12} + {1,0,0,0} = {3,14,12,12} Finish = { True, True, True, True, True} Safe Sequence = {P_0,P_2,P_3,P_4,P_1} Yes, system is in safe state.

3. If request from process P_1 arrives for {0,4,2,0), can it granted immediately?

   Resource – request algorithm is used to check request can be granted immediately.

   (i) Request(P_1) < = Need(P_1) => {0,4,2,0} <= {0,7,5,0} =>True

   (ii) Request(P_1) < = Available(P_1) => {0,4,2,0} <= {1,5,2,0}=>True

   (iii) Then system pretends to have allocated the requested resources to process P_1 by modifying the state as follows:

   Available = Available–Request(P_1);

   $Allocation_i$ = $Allocation_i$ + Request(P_1);

   Need(P_1) = Need(P_1) – Request(P_1)

	Allocation			
	A	B	C	D
P_0	0	0	1	2
P_1	1	4	2	0
P_2	1	3	5	4
P_3	0	6	3	2
P_4	0	0	1	4

	Need			
	A	B	C	D
P_0	0	0	0	0
P_1	0	3	3	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

Available			
A	B	C	D
1	1	0	0

- Now again check whether system is in safe state, by using Banker's safety algorithm and find safe sequence.

Initialize Finish = { False, False, False, False, False}
Work = Available = {1,1,0,0}
Let i=0
Finish[0] == False
Check Need(P_0) <= Work
{0,0,0,0} <= {1,1,0,0} =>true
Therefore, Process P_0 can be granted required resources.
So Work = Work + Allocation[P_0] (After P_0 finishes its task, it will release resources)
Work = {1,1,0,0} + {0,0,1,2} = {1,1,1,2}
Finish = { True, False, False, False, False}
Safe Sequence = {P_0}

Let i=1
Finish[1] == False
Check Need(P_1) <= Work
{0,3,3,0} <= {1,1,1,2} => False
Therefore, Process P_1can not be given resources. P_1 must wait.

Let i=2
Finish[2] == False
Check Need(P_2) <= Work
{1,0,0,2} <= {1,1,1,2} => true
Therefore, Process P_2 can be granted required resources.
So Work = Work + Allocation[P_2] (After P_2 finishes its task, it will release resources)
Work = {1,1,1,2} + {1,3,5,4} = {2,4,6,6}
Finish = { True, False, True, False, False}
Safe Sequence = {P_0,P_2}

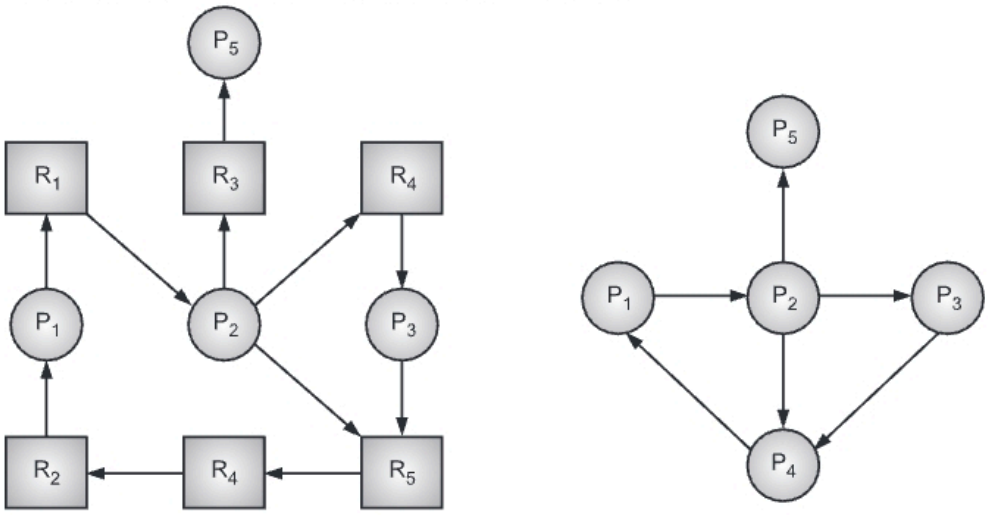
Let i=3 Finish[3] == False Check Need(P_3) <= Work {0,0,2,0} <= {2,4,6,6} =>true Therefore, Process P_3 can be granted required resources. So Work = Work + Allocation[P_3] (After P_3 finishes its task, it will release resources) Work = {2,4,6,6} + {0,6,3,2} = {2,10,9,8} Finish = { True, False, True, True, False} Safe Sequence = {P_0,P_2,P_3}
Let i=4,Finish[4] == False Check Need(P_4) <= Work {0,6,4,2} <= {2,10,9,8}=>true Therefore, Process P_4 can be granted required resources. So Work = Work + Allocation[P_4] (After P_4 finishes its task, it will release resources) Work = {2,10,9,8} + {0,0,1,4}= {2,10,10,12} Finish = { True, False, True, True, True} Safe Sequence ={P_0,P_2,P_3,P_4}
Again Check Finish[1] == False Check Need(P_1) <= Work {0,3,3,0} <= {2,10,10,12} => true Therefore, Process P1 can be granted required resources. So Work = Work + Allocation[P_1] (After P_1 finishes its task, it will release resources) Work = {2,10,10,12} + {1,4,2,0} = {3,14,12,12} Finish = { True, True, True, True, True} Safe Sequence = {P_0,P_2,P_3,P_4,P_1} Yes, system is in safe state. So, request of Process P_1 can be granted immediately.

3.6 DEADLOCK DETECTION

- Deadlock detection detects a deadlock by checking whether all conditions necessary for a deadlock hold simultaneously.
- The systems that do not implement algorithms for deadlock prevention or avoidance must implement an algorithm for deadlock detection and recovery.
- The system must maintain information about:
  1. Process status: resource allocated and requested by a process.
  2. An algorithm to detect the deadlock.
- There are two algorithms i.e. Single instance of resource type and Several instances of a resource type.

3.6.1 Single Instance of Resource Type

- If all resources have single instance in a system, then deadlock detection algorithm uses variant of resource allocation graph, known as wait-for-graph.
- This graph contains only processes, resources are removed. An edge from process P_i to process P_j exists in wait-for-graph if and only if the corresponding resource allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource type R_q. If wait-for-graph contains cycle, then deadlock exists in system.
- To detect a deadlock, system needs to maintain wait-for-graph and periodically invoke an algorithm that searches for cycle in the graph. The algorithm requires an order of n^2 operations where n is number of vertices.



(a) Resource Allocation Graph **(b) Wait-for-graph**

Fig. 3.9

- In the Fig.3.9, Wait-for-graph is shown corresponding to Resource Allocation Graph. In Wait-for-graph:

  Edge $P_1 \rightarrow P_2$ exists since there is corresponding edge from $P_1 \rightarrow R_1$ and $R_1 \rightarrow P_2$
  Edge $P_2 \rightarrow P_3$ exists since there is corresponding edge from $P_2 \rightarrow R_4$ and $R_4 \rightarrow P_3$
  Edge $P_3 \rightarrow P_4$ exists since there is corresponding edge from $P_3 \rightarrow R_5$ and $R_5 \rightarrow P_4$
  Edge $P_4 \rightarrow P_1$ exists since there is corresponding edge from $P_4 \rightarrow R_2$ and $R_2 \rightarrow P_1$
  Edge $P_2 \rightarrow P_5$ exists since there is corresponding edge from $P_2 \rightarrow R_3$ and $R_3 \rightarrow P_5$
- As there is cycle $P_1 \rightarrow P_2 \rightarrow P_4 \rightarrow P_1$ in wait-for-graph, Process P_1,P_2 and P_4 are deadlocked.

3.6.2 Several Instances of the Resource Type

- The deadlock detection algorithm uses same data structures as Banker's algorithm:
  - **Available** = A vector of length m indicating number of available resources of each type.
  - **Allocation** = An $n \times m$ matrix indicating the number of resources of each type currently allocated to each process.
  - **Request** = A $n \times m$ matrix indicating the current request of each process. If Request[i][j] = k then process P_i is requesting k more instances of resource type R_j.

Algorithm:

Step 1: Let Work and Finish be vectors of length m and n, respectively.

(a) Initialize Work = Available.

(b) For i = 1, 2, ..., n, if $Allocation_i \neq 0$, then Finish[i] = false; otherwise, Finish[i] = true.

Step 2: Find an index i such that both:

(a) Finish[i] == false

(b) $Request_i \leq Work$

If no such i exists, go to Step 4.

Step 3: Work = Work + $Allocation_i$, Finish[i] = true go to Step 2.

Step 4: If Finish[i] == false, for some i, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if Finish[i] == false, then P_i is deadlocked.

- Algorithm requires an order of $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example: Consider the system with 3 resources types A, B and C with 7,2,6 instances respectively. Consider the following snapshot:

Process	Allocation		
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2

Process	Request		
	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

Total Resources		
A	B	C
7	2	6

Answer the following questions:

1. What are the contents of Available array?
2. Is there any deadlock?

Solution:

The content of Available array is as follows:

Available = Total Resources – Total Allocation

Total Allocation = {7,2,6}

Process	Allocation		
	A	B	C
P_0	0	1	0
P_1	2	0	0
P_2	3	0	3
P_3	2	1	1
P_4	0	0	2
	7	2	6

Available = {7, 2,6} – {7,2,6} = {0,0,0}

Now execute the deadlock detection algorithm.

Initialize Finish = {False, False, False, False, False} Work = Available = {0,0,0}
Let i=0 Check Finish[0] == False Check Request(P_0) <= Work {0,0,0} <= {0,0,0}=>true So Work = Work + Allocation[P0] Work = {0,0,0} + {0,1,0} = {0,1,0} Finish = { True, False, False, False, False}
Let i=1 Finish[1] == False Check Request(P_1) <= Work {2,0,2} <= {0,1,0} =>False Finish = { True, False, False, False, False} Let i=2; Finish[2] == False Check Request(P_2) <= Work {3,0,3} <= {0,1,0} => False Finish = { True, False, False, False, False}
Let i=3; Finish[3] == False Check Request(P_3) <= Work {2,1,1} <= {0,1,0} =>False Finish = { True, False, False, False, False}
Let i=4; Finish[4] == False Check Request(P_4) <= Work {0,0,2} <= {0,1,0} =>False Finish = { True, False, False, False, False} As Finish[i] == False for process P_1, P_2, P_3 and P_4, system is in deadlock state. Process P_1, P_2, P_3 and P_4 are deadlocked.

3.6.3 Deadlock Detection Usage Algorithm

- When should we invoke the detection algorithm depends on:
  1. How often is a deadlock likely to occur?
  2. How many processes will be affected by deadlock when it happens?
- If deadlocks occur frequently, then the algorithm should be invoked frequently.

- Deadlocks only occur when some process makes a request which cannot be granted, (if this request completes a chain of waiting processes).
  1. **Extreme:** Invoke the algorithm every time a request is denied
  2. **Alternative**: Invoke the algorithm at less frequent time intervals:
     - Once per hour.
     - Whenever CPU utilization < 40%.
     - Disadvantage: cannot determine exactly which process 'caused' the deadlock.

3.7 RECOVERY FROM DEADLOCK [W-18]

- When deadlock detection algorithm detects a deadlock in system, some recovery scheme must be used to recover the system from deadlock. The recovery scheme may use either of the following approaches:

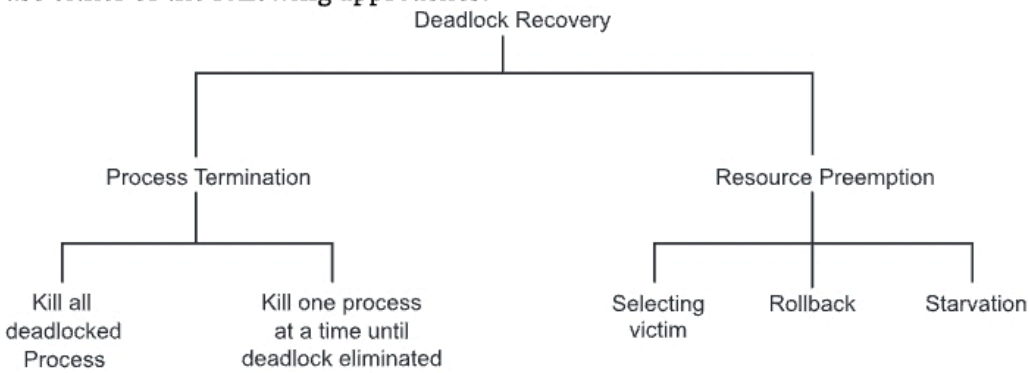


Fig. 3.10: Deadlock Recovery Approaches

- The Fig. 3.10 shows the different ways for breaking a deadlock.
  - The first option in deadlock recovery is to kill one or more processes in order to break the circular wait.
  - The next option is to preempt some resources from one or more of the deadlocked processes.

3.7.1 Process Termination

- The process can be terminated by two ways:
  1. **Kill all deadlocked processes:** This scheme is very expensive way of breaking a deadlock.
  2. **Kill one process at a time until the deadlock is eliminated:** This scheme involves lot of overhead, since after killing each process; deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.
- While killing the process one must take care, because partial completion of the process may leave the system in inconsistent state.
- Many factors are involving in taking the decision of the process selection:
  1. Priority of the process.
  2. Cost involved in the killing the process (how much the process has completed and how much time it will need to completion?)
  3. Which resources and what type of resources are held by the process?
  4. How many total processes will be involved in rollback?

3.7.2 Resource Pre-emption

- In this method successively, some resources are preempted and given to other processes unless a deadlock is broken.
- This method requires three major decisions to be taken.
  1. **Selecting a victim:** May be selected based on following criteria:
     (i) **Priority:** Resources of low priority processes will be preempted.
     (ii) **Own cost:** Process that has completed very less of its execution will preempt its resources.
     (iii) **Cost affecting other processes**: How many maximum processes can restart their execution if resources of a particular process P are preempted?
  2. **Rollback:** The process, from which resource or resources are preempted, will not be able to continue execution such process is rollbacked partially or completely, to some safe state.
  3. **Starvation:** A process may be held for long time waiting for its resources; then process is said to be starved. So care must be taken so that some process is not selected again and again as victim. To achieve this, the cost factor may contain the number of rollbacks of the process. This count will also be considered while selecting the victim.

3.8 COMPARISON BETWEEN DEADLOCK AND STARVATION [W-18]

Table 3.1: Difference between Deadlock and Starvation

Sr. No.	Deadlock	Starvation
1.	A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set.	A process may be held for long time waiting for its resources.
2.	All four conditions-Mutual Exclusion, Hold and Wait, No Preemption and circular wait, occur simultaneously, and then deadlock occurs.	Due to higher priority of process, lower priority process selected as victim again and again and has to wait for long time. It is also known as indefinite post-ponement or or indefinite blocking or starvation.
3.	It is not possible to have a deadlock involving only one single process.	It is possible to have a starvation involving only one single process.
4.	If a deadlock occurs, system can not progress further since two or more processes may be blocked.	If a starvation occurs, system can progress with other processes.
5.	To avoid a deadlock, one of the method can be implemented: • Deadlock Prevention • Deadlock Avoidance • Deadlock detection and recovery	To avoid starvation: • Priority of process is increased periodically (Aging). • Count of number of rollback is kept so that same process should not be selected as victim again and again.

- Let us summarize three different techniques for handling a Deadlock:

Table 3.2: Summary of different Deadlock handling Techniques

Factors	Deadlock Prevention	Deadlock Avoidance	Deadlock Detection and Recovery
Method	This method ensures that at least one of four conditions should not be held.	This method does not grant a resource request if it leads to deadlock.	This method allows entering a deadlock but implements algorithm which invoked periodically to detect deadlock and recover from it.
Handling of process's requests	It prevents a deadlock by restraining how requests can be made. These restrains ensure that one of conditions can not occur.	The system requires a prior information regarding overall potential use of each resource for each process. Then system dynamically considers the every request and decides whether it is safe to grant request to a process.	Process's request can be satisfied by: Process preemption or Resource Preemption of other process.
Advantages	• No preemption necessary • Not required runtime information • Useful for processes that performs single burst of activity	• No preemption necessary • Proper resource utilization • Good system throughput	• Every request of process granted. • Facilitates online handling
Disadvantages	• Low Resource utilization • Starvation • low system throughput • Disallow incremental resource requests	• Future resource requirement must be known. • Sometimes process's requests are not granted, though resources are available. Since it may lead to a deadlock in future.	• Overhead includes run-time cost of maintaining necessary information. • Execution deadlock detection algorithm periodically incurs. overhead in computation time. • Losses inherent in recovery from deadlock.

Solved Problems

Problem 1: Consider a system with 5 processes {P_0, P_1, P_2, P_3, P_4} and four resources types {A, B, C, D}. There are 3 instances of type A, 14 instances of type B, 12 instances of type C and 12 instances of type D. The allocation and maximum demand matrices are as follows:

	Allocation			
	A	**B**	**C**	**D**
P_0	0	6	3	2
P_1	0	0	1	2
P_2	1	0	0	0
P_3	1	3	5	4
P_4	0	0	1	4

	Max			
	A	**B**	**C**	**D**
P_0	0	6	5	2
P_1	0	0	1	2
P_2	0	7	5	0
P_3	2	3	5	6
P_4	0	6	5	6

Answer the following questions having Banker's algorithm.

1. What are the contents of need array?
2. Is a system in a safe state?
3. Is the request from process P_4 arrives for (0, 0, 4, 1) can the request be immediately granted?

Solution:

Total Resources				–	Total Allocation				= Available
A	B	C	D		A	B	C	D	= A B C D
3	14	12	12	–	2	9	10	12	= 1 5 2 0

1. Need = Max – Allocation

	Need					Max				–	Allocation			
	A	**B**	**C**	**D**		**A**	**B**	**C**	**D**		**A**	**B**	**C**	**D**
P_0	0	0	2	0		0	6	5	2		0	6	3	2
P_1	0	0	0	0	=	0	0	1	2	–	0	0	1	2
P_2	0	7	5	0		1	7	5	0		1	0	0	0
P_3	1	0	0	2		2	3	5	6		1	3	5	4
P_4	0	6	4	2		0	6	5	6		0	0	1	4

2. Now using Banker's Safety Algorithm.

Work = Available = {1, 5, 2, 0}
Finish = {F, F, F, F, F}

(i) Let i = 0, Finish[i] = F,

Need (P_0) < = Work ⇒ {0, 0, 2, 0} < = {1, 5, 2, 0}
So work = Work + Allocation
= {1, 5, 2, 0} + {0, 6, 3, 2}
Work = {1, 11, 5, 2}

(After P_0 finishes, it will release resources)

Finish = {T, F, F, F, F}
Safe sequence = {P_0}

(ii) Let i = 1, Finish[1] = F

$$\text{Need } (P_1) <= \text{Work} \Rightarrow \{0, 0, 0, 0\} <= \{1, 11, 5, 2\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{1, 11, 5, 2\} + \{0, 0, 1, 2\}$$
$$\text{Work} = \{1, 11, 6, 4\}$$

(After P_1 finishes, it will release resources)

$$\text{Finish} = \{T, T, F, F, F\}$$
$$\text{Safe sequence} = \{P_0, P_1\}$$

(iii) Let i = 2, Finish[2] = F

$$\text{Need } (P_2) <= \text{Work} \Rightarrow \{0, 7, 5, 0\} <= \{1, 11, 6, 4\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$=\{1, 11, 6, 4\} + \{1, 0, 0, 0\}$$
$$\text{Work} = \{2, 11, 6, 4\}$$

(After P_2 finishes, it will release resources)

$$\text{Finish} = \{T, T, T, F, F\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2\}$$

(iv) Let i = 3, Finish[3] = F

$$\text{Need } (P_3) <= \text{Work} \Rightarrow \{1, 0, 0, 2\} <= \{2, 11, 6, 4\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{2, 11, 6, 4\} + \{1, 3, 5, 4\}$$
$$\text{Work} = \{3, 14, 11, 8\}$$
$$\text{Finish} = \{T, T, T, T, F\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2, P_3\}$$

(v) Let i = 4, Finish[4] = F

$$\text{Need } (P_4) <= \text{Work} \Rightarrow \{0, 6, 4, 2\} <= \{3, 14, 11, 8\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$\Rightarrow \{3, 14, 11, 8\} + \{0, 0, 1, 4\}$$
$$= \{3, 14, 12, 12\}$$
$$\text{Finish} = \{T, T, T, T, T\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2, P_3, P_4\}$$

$\therefore$ System is in safe state.

3. If request (0, 0, 4, 1) from P_4 arrives. According to Resource Request Algorithm,

(i) $\text{Request } (P_4) = \text{Need } 4 \Rightarrow$

$\{0, 0, 4, 1\} <= \{0, 0, 6, 4\} \Rightarrow$ True go to Step 2

(ii) Check if $\text{Request } (P_4) <= \text{Available}$

$\{0, 0, 4, 1\} <= \{1, 5, 2, 0\} \Rightarrow$ No

So P_4 has to wait since resources are not available.

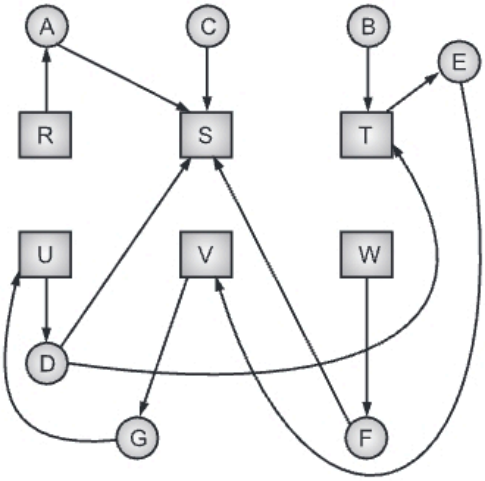
$\therefore$ Request cannot be granted immediately.

Problem 2: Consider a system with 7 processes A through G and six types of resources R through W with one resource for each type. Resource ownership is as follows:

A holds R and wants S
B holds nothing but wants T
C holds nothing but wants S
D holds U and wants S and T
E holds T and wants V
F holds W and wants S
G holds V and wants U

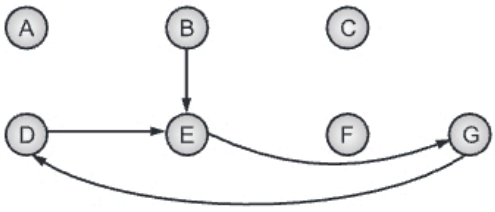
Is the system deadlocked, and if so, which processes are involved?

Solution: Resource Allocation graph is as follows:



(a) Allocation graph

Wait – for – Graph for above resource allocation graph is:



(b) Resource allocation graph

Fig. 3.11

As cycle exists in graph B → E – G – D; processes D, E, G are involved in a deadlock.

Problem 3: Consider given snapshot of system. A system has 5 processes and 3 types of resources A, B, C.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Answer the following questions using Banker's Algorithm:

1. What are the contents of matrix need?

2. Is the system in safe state?
3. If request from process P_1 arrives as (1, 0, 2) can the request be granted immediately?

Solution:

1. The contents of Need Matrix,

   Need = Max – Allocation

	Need		
	A	**B**	**C**
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

2. Now using Banker's Safety Algorithm,

$$\text{Work} = \text{Available} = \{3, 3, 2\}$$
$$\text{Finish} = \{F, F, F, F, F\}$$

(i) Let i = 0, $\text{Finish}[0] = F$

$$\text{Need}(P_0) <= \text{Work} \Rightarrow \{7, 4, 3\} <= \{3, 3, 2\}$$

No, P_0 cannot be given resources. P_0 has to wait.

(ii) Let i = 1, $\text{Finish}[1] = F$

$$\text{Need}(P_1) <= \text{Work} \Rightarrow \{1, 2, 2\} <= \{3, 3, 2\}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (3, 3, 2) + (2, 0, 0)$$
$$= (5, 3, 2)$$

[P_1 releases resources after its condition]

$$\text{Finish} = \{F, T, F, F, F\}$$
$$\text{Safe sequence} = \{P_1\}$$

(iii) Let i = 2, $\text{Finish}[2] = F$

$$\text{Need}(P_2) <= \text{Work} \Rightarrow \{6, 0, 0\} <= \{5, 3, 2\}$$

No, P_2 has a wait.

(iv) Let i = 3 $\text{Finish}[3] = F$

$$\text{Need}(P_3) <= \text{Work} \Rightarrow \{0, 1, 1\} <= \{5, 3, 2\}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (5, 3, 2) + (2, 1, 1)$$
$$= (7, 4, 3)$$

After P_3 finishes, it releases resources

$$\text{Finish} = \{F, T, F, T, F\}$$
$$\text{Safe sequence} = \{P_1, P_3\}$$

(v) Let i = 4 Finish[4] = F

Need (P_4) <= Work ⇒ {4, 3, 1} <= {7, 4, 3} ⇒ Yes

∴ Work = Work + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

After P_4 finishes, it releases resources

Finish = {F, T, F, T, T}

Safe sequence = {P_1, P_3, P_4}

(vi) Let i = 0

Need (P_0) <= Work ⇒ {7, 4, 3} <= {7, 4, 5}

∴ Work = Work + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

After P_0 finishes, it releases resources

Finish = {T, T, F, T, T}

Safe sequence = {P_1, P_3, P_4,P_0}

(vii) Let i = 2

Need (P_2) <= Work ⇒ {6, 0, 0} <= {7, 5, 5}

∴ So work = Work + Allocation

= (7, 5, 5) + (3, 0, 2)

= (10, 5, 7)

Finish = {T, T, T, T, T}

Safe sequence = {P_1, P_3, P_4,P_0, P_2}

∴ Yes, system is in safe state.

3. If request from process P_1 arrives for (1, 0, 2), can it granted immediately?

According to Resource request algorithm:

(i) Request (P_1) <= Need (P_1)

(1, 0, 2) <= (1, 2, 2)

(ii) Request (P_1) <= Available

(1, 0, 2) <= (3, 3, 2) which is true.

Then system pretends to fulfill request, then modify resource allocation state as follows:

(i) Available = Available – Request (P_1)

(ii) Allocation (P_1) = Allocation (P_1) + Request (P_1)

(iii) Need (P_1) = Need (P_1) – Request (P_1)

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Now, check system is in safe state

Work = Available = {2, 3, 0}

Finish = {F, F, F, F, F}

(i) Let i = 0, Finish[0] = F

Need (P_0) < = Work ⇒ {7, 4, 3} < = {2, 3, 0}

No, P_0 has to wait for resources.

(ii) Let i = 1, Finish[1] = F

Need (P_1) < = Work ⇒ {0, 2, 0} < = {2, 3, 0}

∴ Work = Work + Allocation

= (2, 3, 0) + (3, 0, 2)

= (5, 3, 2)

After P_1 finishes, it releases resources

Finish = {F, T, F, F, F}

Safe sequence = {P_1}

(iii) Let i = 2, Finish[2] = F

Need (P_2) < = Work ⇒ {6, 0, 0} < = {5, 3, 2}

No, P_2 has to wait for resources.

(iv) Let i = 3, Finish[3] = F

Need (P_3) < = Work ⇒ {0, 1, 1} < = {5, 3, 2}

∴ Work = Work + Allocation

= (5, 3, 2) + (2, 1, 1)

= (7, 4, 3)

After P_3 finishes, it releases resources

Finish = {F, T, F, T, F}

Safe sequence = {P_1,P_3}

(v) Let i = 4, Finish[4] = F

Need (P_4) < = Work ⇒ {4, 3, 1} < = {7, 4, 3}

∴ Work = Work + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

After P_4 finishes, it releases resources

Finish = {F, T, F, T, T}

Safe sequence = {P_1,P_3, P_4}

(vi) Let i = 0, Finish[0] = F

Need (P_0) < = Work ⇒ {7, 4, 3} < = {7, 4, 5}

∴ Work = Work + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

After P_0 finishes, it releases resources

Finish = {T, T, F, T, T}

Safe sequence = $\{P_1, P_3, P_4, P_0\}$

(vii) Let i = 2, Finish[2] = F

Need (P_2) < = Work ⇒ {6, 0, 0} < = {7, 5, 5}

∴ Work = Work + Allocation

= (7, 5, 5) + (3, 0, 2)

= (10, 5, 7)

Safe sequence = $\{P_1, P_3, P_4, P_0, P_2\}$

Yes, system is in safe state. So request can be granted immediately.

Problem 4: Consider the following snapshot of a system:

Process	Allocation	Max.	Available
	A B C	A B C	A B C
P0	2 3 2	9 7 5	3 3 2
P1	4 0 0	5 2 2	
P2	5 0 4	1 1 0 4	
P3	4 3 3	4 4 4	
P4	2 2 4	6 5 5	

Answer the following questions using Banker's algorithm:

1. What is the content of Need Matrix?
2. Is the system in a safe state? If yes, give the safe sequence.

Solution:

1. Need = Max – Allocation

	Need			Max			Allocation		
	A	B	C	A	B	C	A	B	C
P_0	7	4	3	9	7	5	2	3	2
P_1	1	2	2	5	2	2	4	0	0
P_2	6	0	0	11	0	4	5	0	4
P_3	0	1	1	4	4	4	4	3	3
P_4	4	3	1	6	5	5	2	2	4

2. Now using Banker's Safety Algorithm.

$$\text{Work} = \text{Available} = \{3, 3, 2\}$$
$$\text{Finish} = \{F, F, F, F, F\}$$

(i) Let i = 0, $\text{Finish}[0] = F$

$$\text{Need } (P_0) <= \text{Work} \Rightarrow \{7, 4, 3\} <= \{3, 3, 2\}$$

No, P_0 cannot be given resources. P_0 has to wait.

(ii) Let i = 1, $\text{Finish}[1] = F$

$$\text{Need } (P_1) <= \text{Work} \Rightarrow \{1, 2, 2\} <= \{3, 3, 2\}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (3, 3, 2) + (2, 0, 0)$$
$$= (5, 3, 2)$$

[P_1 releases resources after its condition]

$$\text{Finish} = \{F, T, F, F, F\}$$
$$\text{Safe sequence} = \{P_1\}$$

(iii) Let i = 2, $\text{Finish}[2] = F$

$$\text{Need } (P_2) <= \text{Work} \Rightarrow \{6, 0, 0\} <= \{5, 3, 2\}$$

No, P_2 has a wait.

(iv) Let i = 3 $\text{Finish}[3] = F$

$$\text{Need } (P_3) <= \text{Work} \Rightarrow \{0, 1, 1\} <= \{5, 3, 2\}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (5, 3, 2) + (2, 1, 1)$$
$$= (7, 4, 3)$$

After P_3 finishes, it releases resources

$$\text{Finish} = \{F, T, F, T, F\}$$
$$\text{Safe sequence} = \{P_1, P_3\}$$

(v) Let i = 4 $\text{Finish}[4] = F$

$$\text{Need } (P_4) <= \text{Work} \Rightarrow \{4, 3, 1\} <= \{7, 4, 3\} \Rightarrow \text{Yes}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (7, 4, 3) + (0, 0, 2)$$
$$= (7, 4, 5)$$

After P_4 finishes, it releases resources

$$\text{Finish} = \{F, T, F, T, T\}$$
$$\text{Safe sequence} = \{P_1, P_3, P_4\}$$

(vi) Let i = 0

$$\text{Need } (P_0) <= \text{Work} \Rightarrow \{7, 4, 3\} <= \{7, 4, 5\}$$
$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (7, 4, 5) + (0, 1, 0)$$
$$= (7, 5, 5)$$

After P_0 finishes, it releases resources

$$\text{Finish} = \{T, T, F, T, T\}$$
$$\text{Safe sequence} = \{P_1, P_3, P_4, P_0\}$$

(vii) Let i = 2

$$\text{Need } (P_2) <= \text{Work} \Rightarrow \{6, 0, 0\} <= \{7, 5, 5\}$$

$$\therefore \quad \text{So work} = \text{Work} + \text{Allocation}$$

$$= (7, 5, 5) + (3, 0, 2)$$

$$= (10, 5, 7)$$

$$\text{Finish} = \{T, T, T, T, T\}$$

$$\text{Safe sequence} = \{P_1, P_3, P_4, P_0, P_2\}$$

$\therefore$ Yes, system is in safe state.

Problem 5: Consider a system with 5 processes $\{P_0, P_1, P_2, P_3, P_4\}$ and four resources types {A, B, C, D}. There are 3 instances of type A, 14 instances of type B, 12 instances of type C and 12 instances of type D. The allocation and maximum demand matrices are as follows:

	Allocation			
	A	**B**	**C**	**D**
P_0	0	6	3	2
P_1	0	0	1	2
P_2	1	0	0	0
P_3	1	3	5	4
P_4	0	0	1	4

	Max			
	A	**B**	**C**	**D**
P_0	0	6	5	2
P_1	0	0	1	2
P_2	0	7	5	0
P_3	2	3	5	6
P_4	0	6	5	6

Answer the following questions having Banker's algorithm.

1. What are the contents of need array?
2. Is a system in a safe state?
3. Is the request from process P_4 arrives for (0, 0, 4, 1) can the request be immediately granted?

Solution:

Total Resources				–	Total Allocation				=	Available			
A	B	C	D		A	B	C	D	=	A	B	C	D
3	14	12	12	–	2	9	10	12	=	1	5	2	0

1. Need = Max – Allocation

	Need			
	A	**B**	**C**	**D**
P_0	0	0	2	0
P_1	0	0	0	0
P_2	0	7	5	0
P_3	1	0	0	2
P_4	0	6	4	2

=

	Max			
	A	**B**	**C**	**D**
P_0	0	6	5	2
P_1	0	0	1	2
P_2	1	7	5	0
P_3	2	3	5	6
P_4	0	6	5	6

–

	Allocation			
	A	**B**	**C**	**D**
P_0	0	6	3	2
P_1	0	0	1	2
P_2	1	0	0	0
P_3	1	3	5	4
P_4	0	0	1	4

2. Now using Banker's Safety Algorithm.

$$\text{Work} = \text{Available} = \{1, 5, 2, 0\}$$
$$\text{Finish} = \{F, F, F, F, F\}$$

(i) Let i = 0, Finish[i] = F,

$$\text{Need } (P_0) <= \text{Work} \Rightarrow \{0, 0, 2, 0\} <= \{1, 5, 2, 0\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{1, 5, 2, 0\} + \{0, 6, 3, 2\}$$
$$\text{Work} = \{1, 11, 5, 2\}$$

(After P_0 finishes, it will release resources)

$$\text{Finish} = \{T, F, F, F, F\}$$
$$\text{Safe sequence} = \{P_0\}$$

(ii) Let i = 1, Finish[1] = F

$$\text{Need } (P_1) <= \text{Work} \Rightarrow \{0, 0, 0, 0\} <= \{1, 11, 5, 2\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{1, 11, 5, 2\} + \{0, 0, 1, 2\}$$
$$\text{Work} = \{1, 11, 6, 4\}$$

(After P_1 finishes, it will release resources)

$$\text{Finish} = \{T, T, F, F, F\}$$
$$\text{Safe sequence} = \{P_0, P_1\}$$

(iii) Let i = 2, Finish[2] = F

$$\text{Need } (P_2) <= \text{Work} \Rightarrow \{0, 7, 5, 0\} <= \{1, 11, 6, 4\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{1, 11, 6, 4\} + \{1, 0, 0, 0\}$$
$$\text{Work} = \{2, 11, 6, 4\}$$

(After P_2 finishes, it will release resources)

$$\text{Finish} = \{T, T, T, F, F\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2\}$$

(iv) Let i = 3, Finish[3] = F

$$\text{Need } (P_3) <= \text{Work} \Rightarrow \{1, 0, 0, 2\} <= \{2, 11, 6, 4\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$= \{2, 11, 6, 4\} + \{1, 3, 5, 4\}$$
$$\text{Work} = \{3, 14, 11, 8\}$$
$$\text{Finish} = \{T, T, T, T, F\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2, P_3\}$$

(v) Let i = 4, Finish[4] = F

$$\text{Need } (P_4) <= \text{Work} \Rightarrow \{0, 6, 4, 2\} <= \{3, 14, 11, 8\}$$
$$\text{So work} = \text{Work} + \text{Allocation}$$
$$\Rightarrow \{3, 14, 11, 8\} + \{0, 0, 1, 4\}$$
$$= \{3, 14, 12, 12\}$$
$$\text{Finish} = \{T, T, T, T, T\}$$
$$\text{Safe sequence} = \{P_0, P_1, P_2, P_3, P_4\}$$

$\therefore$ System is in safe state.

3. If request (0, 0, 4, 1) from P_4 arrives. According to Resource Request Algorithm,

   (i) Request (P_4) = Need 4 $\Rightarrow$

   $\{0, 0, 4, 1\} <= \{0, 0, 6, 4\} \Rightarrow$ True go to Step 2

   (ii) Check if Request (P_4) <= Available

   $\{0, 0, 4, 1\} <= \{1, 5, 2, 0\} \Rightarrow$ No

   So P_4 has to wait since resources are not available.

   $\therefore$ Request cannot be granted immediately.

Problem 6: Consider the following sets P, R and E:

$P = \{P_1, P_2, P_3\}$

$R = \{R_1, R_2, R_3, R_4\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1\}$

Also consider the following number of instances per resource type:

(i) One instance of resource type R_1 and R_3.

(ii) Two instances of resource type R_2.

(iii) Three instances of resource type R_4.

Construct the Resource - allocation graph for the above problem. Check whether the system is in the deadlock.

Solution:

$R = \{R_1, R_2, R_3, R_4\}$

$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1\}$

Resource instances:

- One instance of resource typeR_1
- Two instances of resource typeR_2
- One instance of resource typeR_3
- Three instances of resource typeR_4

Process States:

- Process P_1 is holding an instance of resource type R_2 and is waiting for an instance of resource type R_1.
- Process P_2 is holding an instance of R_1 and an instance of R_2 and is waiting for an instance ofR_3.

The resource allocation graph for above instance is as follows:

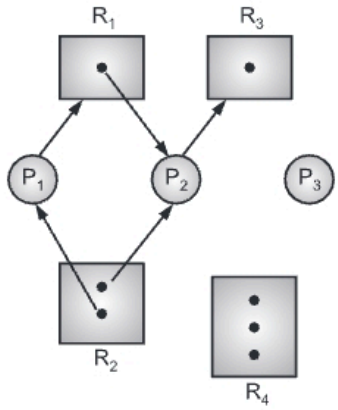


Fig. 3.12

Since, above resource allocation graph does not contain cycle, so no process in system is deadlocked. So there is no deadlock.

Problem 7: Consider given snapshot of system. A system has 5 processes and 3 types of resources A, B, C.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Answer the following questions using Banker's Algorithm:

1. What is the contents of matrix need?
2. Is the system in safe state?
3. If request from process P_1 arrives as (1, 0, 2) can the request be granted immediately?

Solution:

1. The contents of Need Matrix

   Need = Max – Allocation

	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

2. Now using Banker's Safety Algorithm,

$$\text{Work} = \text{Available} = \{3, 3, 2\}$$
$$\text{Finish} = \{F, F, F, F, F\}$$

(i) Let i = 0, $\text{Finish}[0] = F$

$$\text{Need}(P_0) <= \text{Work} \Rightarrow \{7, 4, 3\} <= \{3, 3, 2\}$$

No, P_0 cannot be given resources. P_0 has to wait.

(ii) Let i = 1, $\text{Finish}[1] = F$

$$\text{Need}(P_1) <= \text{Work} \Rightarrow \{1, 2, 2\} <= \{3, 3, 2\}$$

$$\therefore \quad \text{Work} = \text{Work} + \text{Allocation}$$
$$= (3, 3, 2) + (2, 0, 0)$$
$$= (5, 3, 2)$$

[P_1 releases resources after its condition]

$$\text{Finish} = \{F, T, F, F, F\}$$
$$\text{Safe sequence} = \{P_1\}$$

(iii) Let i = 2, Finish[2] = F

Need $(P_2) <=$ Work $\Rightarrow$ {6, 0, 0} <= {5, 3, 2}

No, P_2 has a wait.

(iv) Let i = 3, Finish[3] = F

Need $(P_3) <=$ Work $\Rightarrow$ {0, 1, 1} <= {5, 3, 2}

$\therefore$ Work = Work + Allocation

= (5, 3, 2) + (2, 1, 1)

= (7, 4, 3)

After P_3 finishes, it releases resources

Finish = {F, T, F, T, F}

Safe sequence = $\{P_1, P_3\}$

(v) Let i = 4, Finish[4] = F

Need $(P_4) <=$ Work $\Rightarrow$ {4, 3, 1} <= {7, 4, 3} $\Rightarrow$ Yes

$\therefore$ Work = Work + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

After P_4 finishes, it releases resources

Finish = {F, T, F, T, T}

Safe sequence = $\{P_1, P_3, P_4\}$

(vi) Let i = 0

Need $(P_0) <=$ Work $\Rightarrow$ {7, 4, 3} <= {7, 4, 5}

$\therefore$ Work = Work + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

After P_0 finishes, it releases resources

Finish = {T, T, F, T, T}

Safe sequence = $\{P_1, P_3, P_4, P_0\}$

(vii) Let i = 2

Need $(P_2) <=$ Work $\Rightarrow$ {6, 0, 0} <= {7, 5, 5}

$\therefore$ So work = Work + Allocation

= (7, 5, 5) + (3, 0, 2)

= (10, 5, 7)

Finish = {T, T, T, T, T}

Safe sequence = $\{P_1, P_3, P_4, P_0, P_2\}$

$\therefore$ Yes, system is in safe state.

3. If request from process P_1 arrives for (1, 0, 2), can it granted immediately?

According to Resource request algorithm:

(i) Request $(P_1) <=$ Need (P_1)

(1, 0, 2) <= (1, 2, 2)

(ii) Request $(P_1) <= $ Available

$(1, 0, 2) <= (3, 3, 2)$ which is true.

Then system pretends to fulfill request, then modify resource allocation state as follows:

(i) Available = Available – Request (P_1)

(ii) Allocation (P_1) = Allocation (P_1) + Request (P_1)

(iii) Need (P_1) = Need (P_1) – Request (P_1)

	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

Now, check system is in safe state

Work = Available = {2, 3, 0}

Finish = {F, F, F, F, F}

(i) Let i = 0, Finish[0] = F

Need $(P_0) <= $ Work $\Rightarrow$ $\{7, 4, 3\} <= \{2, 3, 0\}$

No, P_0 has to wait for resources.

(ii) Let i = 1, Finish[1] = F

Need $(P_1) <= $ Work $\Rightarrow$ $\{0, 2, 0\} <= \{2, 3, 0\}$

$\therefore$ Work = Work + Allocation

= (2, 3, 0) + (3, 0, 2)

= (5, 3, 2)

After P_1 finishes, it releases resources

Finish = {F, T, F, F, F}

Safe sequence = $\{P_1\}$

(iii) Let i = 2, Finish[2] = F

Need $(P_2) <= $ Work $\Rightarrow$ $\{6, 0, 0\} <= \{5, 3, 2\}$

No, P_2 has to wait for resources.

(iv) Let i = 3, Finish[3] = F

Need $(P_3) <= $ Work $\Rightarrow$ $\{0, 1, 1\} <= \{5, 3, 2\}$

$\therefore$ Work = Work + Allocation

= (5, 3, 2) + (2, 1, 1)

= (7, 4, 3)

After P_3 finishes, it releases resources

Finish = {F, T, F, T, F}

Safe sequence = $\{P_1, P_3\}$

(v) Let i = 4, Finish[4] = F

Need (P_4) < = Work ⇒ {4, 3, 1} < = {7, 4, 3}

∴ Work = Work + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

After P_4 finishes, it releases resources

Finish = {F, T, F, T, T}

Safe sequence = {P_1,P_3, P_4}

(vi) Let i = 0, Finish[0] = F

Need (P_0) < = Work ⇒ {7, 4, 3} < = {7, 4, 5}

∴ Work = Work + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

After P_0 finishes, it releases resources

Finish = {T, T, F, T, T}

Safe sequence = {P_1,P_3, P_4, P_0}

(vii) Let i = 2, Finish[2] = F

Need (P_2) < = Work ⇒ {6, 0, 0} < = {7, 5, 5}

∴ Work = Work + Allocation

= (7, 5, 5) + (3, 0, 2) = (10, 5, 7)

Safe sequence = {P_1,P_3, P_4, P_0, P_2}

Yes, system is in safe state. So request can be granted immediately

---

Problem 8: Consider a system with four processes P_1, P_2, P_3, P_4 and four resource types A, B, C, D with one instance of each type. Resource ownership is as follows:

P_1 holds A and wants C

P_2 holds B

P_3 holds D wants B

P_4 holds C wants D

Is system deadlocked?

(Draw resource-allocation graph and wait-for graph)

Solution:

R = {A, B, C, D}

E = {$P_1 \rightarrow C_1$, $P_4 \rightarrow D$, $P_3 \rightarrow B$, $A \rightarrow P_1$, $C \rightarrow P_4$, $D \rightarrow P_3$, $B \rightarrow P_2$}

Resource instances:

- One instance of resource type A,B,C,D

Process states:

- Process $P_1$is holding an instance of resource type A and is waiting for an instance of resource type C.
- Process P_2 is holding an instance of B.

- Process P_3 is holding an instance of resource type D and is waiting for an instance of resource type B.
- Process P_4 is holding an instance of resource type C and is waiting for an instance of resource type D.

The resource allocation graph for above instance is as follows:

Wait-for-Graph for above instance is as follows:

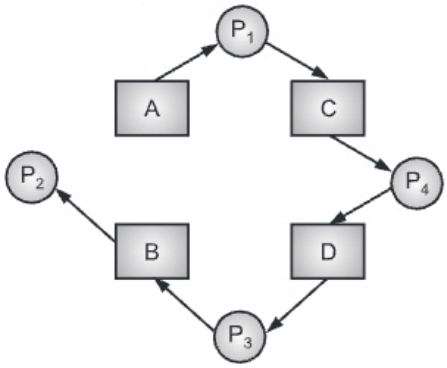


Fig. 3.13

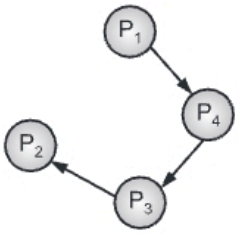


Fig. 3.14

As wait-for-Graph does not contain cycle, System has no deadlock.

Problem 9: Consider the following snapshot of the system:

Process	Allocation				Max			
	A	B	C	D	A	B	C	D
P0	0	3	2	4	6	5	4	4
P1	1	2	0	1	4	4	4	4
P2	0	0	0	0	0	0	1	2
P3	3	3	2	2	3	9	3	4
P4	1	4	3	2	2	5	3	3
P5	2	4	1	4	4	6	3	4

A system has total 10, 20, 13, 15 instances of resource types A, B, C, D respectively.

Answer the following using Banker's algorithm:

(i) What is content of need and available matrix?

(ii) Is the system in safe state?

(iii) If a request from process P1 arrives for (2, 2, 3, 3), can it be granted immediately?

Solution:

Total Resources				–	Total Allocation				=	Available			
A	B	C	D		A	B	C	D	=	A	B	C	D
10	20	12	15	–	7	16	8	13	=	3	4	4	2

1. Need = Max – Allocation

	Need					Max					Allocation			
	A	B	C	D		A	B	C	D		A	B	C	D
P_0	6	2	2	0		6	5	4	4		0	3	2	4
P_1	3	2	4	3	=	4	4	4	4	–	1	2	0	1
P_2	0	0	1	0		0	0	1	2		0	0	0	0
P_3	0	6	1	2		3	9	3	4		3	3	2	2
P_4	1	1	0	1		2	5	3	3		1	4	3	2
P5	2	2	2	1		4	6	3	5		2	4	1	4

2. Now using Banker's Safety Algorithm.

Work = Available = {3, 4, 4, 2}

Finish = {F, F, F, F, F, F}

(i) Let i = 0, Finish[i] = F,

Need (P_0) < = Work ⇒ {6, 2, 2, 0} < = {3, 4,4, 2}

No, P_0 has to wait for resources.

(ii) Let i = 1, Finish[1] = F

Need (P_1) < = Work ⇒ {3, 2, 4, 3} < = {3, 4, 4, 2}

No, P1has to wait for resources.

(iii) Let i = 2, Finish[2] = F

Need (P_2) < = Work ⇒ {0, 0, 1, 0} < = {3, 4, 4, 2}

So work = Work + Allocation

= {3, 4, 4, 2} + {0, 0, 0, 0}

Work = {3, 4, 4, 2}

(After P_2 finishes, it will release resources)

Finish = {F, F, T, F, F, F}

Safe sequence = {P_2}

(iv) Let i = 3, Finish[3] = F

Need (P_3) < = Work ⇒ {0,6,1, 2} < = {3, 4, 4, 2}

No, P_3 has to wait for resources.

(v) Let i = 4, Finish[4] = F

Need (P_4) < = Work ⇒ {1, 1, 0, 1} < = {3, 4, 4, 2}

So work = Work + Allocation

⇒ {3, 4, 4, 2} + { 1, 4, 3, 2}

= {4, 8, 7, 4}

(After P4 finishes, it will release resources)

Finish = {F, F, T, F, T, F}

Safe sequence = {P_2,P_4}

(vi) Let i = 5, Finish[5] = F

Need (P_5) < = Work ⇒ {2, 2, 2,1} < = {4, 8, 7, 4}

So work = Work + Allocation

⇒ {4, 8, 7, 4} + { 2, 4, 1, 4}

= {6, 12, 8, 8}

(After P_5 finishes, it will release resources)

Finish = {F, F, T, F, T, T}

Safe sequence = {P_2,P_4,P_5}

(vii) Let i = 0, Finish[0] = F

Need (P_0) < = Work ⇒ {6, 2, 2, 0} }< = {6, 12, 8, 8}

So work = Work + Allocation

⇒ {6, 12, 8, 8} + { 0, 3, 2, 4}

= {6, 15, 10, 12}

(After P_0 finishes, it will release resources)

Finish = {T, F, T, F, T,T}

Safe sequence = {P_2,P_4,P_5,P_0}

(viii) Let i = 1, Finish[1] = F

Need (P_1) < = Work ⇒ {3, 2, 4, 3} < = {6, 15, 10, 12}

So work = Work + Allocation

⇒ {6, 15, 10, 12} + { 1, 2, 0, 1}

= {7, 17, 10, 13}

(After P_1 finishes, it will release resources)

Finish = {T, T, T, F, T, T}

Safe sequence = {P_2,P_4,P_5, P_0, P_1}

(viii) Let i = 3, Finish[3] = F

Need (P_3) < = Work ⇒ {0, 6, 1, 2} < = {7, 17, 10, 13}

So work = Work + Allocation

⇒ {7, 17, 10, 13} + { 3, 3, 2, 2}

= {10, 20, 12, 15}

(After P_3 finishes, it will release resources)

Finish = {T, T, T, T, T, T}

Safe sequence = {P_2,P_4,P_5,P_0,P_1,P_3}

∴ Yes, system is in safe state.

If request (2, 2, 3, 3) from P_1 arrives, then according to Resource Request Algorithm,

(i) Request (P_1) <= Need 4 ⇒

(2, 2, 3, 3) < = {3, 2, 4, 3} ⇒ True go to Step 2

(ii) Check if Request (P_1) < = Available

(2, 2, 3, 3) < = {3, 4, 4, 2} ⇒ No

So P_1 has to wait since resources are not available.

∴ Request cannot be granted immediately.

Summary

- A set of processes is in a deadlock state when every process in the set is waiting for an event that can only be caused by another process in the set.
- For example, there are two tape drives, in some system. There are two processes, each holding one more tape. Now each of the process needs one more tape drive. Now both the processes are waiting for each other to release a tape drive. This is a deadlock situation.
- Under the normal mode of operation, any resource is utilized by a process in following manner:
  1. **Request:** A process request for a resource. If it is available, it is immediately granted; otherwise the process must wait for it.
  2. **Use:** After a request is granted, process uses the resource.
  3. **Release:** The Process releases the resource after it is used.
- Coffman identified four conditions that must hold simultaneously for there to be a deadlock are Mutual Exclusion Condition, Hold and Wait Condition, No-Preemptive Condition and Circular Wait Condition.
- Resource allocation graph is directed graph, which is used to describe resources allocated to a particular process in a system. It contains set of processes P, set of resources $R = \{R_1, R_2, ..., R_n\}$, request edge and allocation edge or an assignment edge.
- If Resource allocation graph contains cycle, then deadlock may exist.
- The deadlock handling techniques are deadlock prevention, deadlock avoidance, and deadlock detection and recovery.
- Deadlock prevention algorithm disallows one of four necessary conditions for deadlock.
- Deadlock avoidance algorithm do not grant resource request if resource-allocation have potential have led to deadlock.
- Deadlock Detection and Recovery algorithm always grants resource request when possible. It periodically checks for a deadlock. If deadlock exists, recover from it.
- A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock.
- A system is in a safe state; if there exists a safe sequence.
- Safe Sequence: A sequence of processes $<P_1, P_2, ..., P_n>$ is a safe sequence for current allocation state if, for each process Pi, the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j have finished with $j < i$.
- A system without safe sequence is unsafe.
- A safe state is not deadlocked state. Conversely deadlocked state is unsafe state. Thus unsafe system may lead to deadlock.
- The deadlock avoidance algorithm used for several instances of the resources is called as Banker's algorithm developed by Dijkstra.
- Deadlock detection algorithm is used to detect the deadlock. There are two algorithms:
- Single instance resource type uses wait-for-graph
- Several instances of a resource type uses data structure of Banker's algorithm.
- When deadlock detection algorithm detects a deadlock in system, some recovery scheme must be used to recover the system from deadlock.
- The recovery scheme may use either of the following approaches: Process Termination and Resource Preemption.

Check Your Understanding

1. A _____ is a situation/condition in which some processes wait for each other's actions indefinitely.
   (a) Deadlock (b) Starvation
   (c) Synchronization (d) All of the Above
2. Which of the following condition is required for deadlock to be possible?
   (a) Mutual exclusion
   (b) A process may hold allocated resources while awaiting assignment of other resources
   (c) No resource can be forcibly removed from a process holding it
   (d) All of the Above
3. Which one of the following is the deadlock avoidance algorithm?
   (a) FCFS algorithm (b) Round-robin algorithm
   (c) Banker's algorithm (d) Karn's algorithm
4. Which of the following is not necessary condition in deadlock?
   (a) Hold and Wait (b) Circular Wait
   (c) Mutual Exclusion (d) Safe State
5. In Resource-allocation graph, if there is edge from process to resource, then it is known as _____.
   (a) Claim edge (b) Request edge
   (c) Allocation edge (d) None of the Above
6. In Resource-allocation graph, if there is edge from resource to process, then it is known as _____.
   (a) Claim edge (b) Request edge
   (c) Allocation edge (d) Deallocation edge
7. Banker's safety Algorithm is used for _____.
   (a) Deadlock detection (b) Deadlock Prevention
   (c) Deadlock Avoidance (d) Recovery
8. For effective operating system, when to check for deadlock?
   (a) Every time a resource request is made.
   (b) At fixed time intervals.
   (c) Every time a resource request is made at fixed time intervals.
   (d) None of the Above
9. Wait-for-graph is used for _____ in the system.
   (a) Deadlock detection (b) Deadlock Prevention
   (c) Deadlock Avoidance (d) Recovery
10. A problem encountered in multitasking when a process is perpetually denied necessary resources is called as _____.
   (a) Deadlock (b) Starvation
   (c) Mutual exclusion (d) All of the Above

11. A system is in the safe state if _____.
    (a) the system can allocate resources to each process in some order and still avoid a deadlock
    (b) there exist a safe sequence
    (c) Both (a) & (b)
    (d) None of the Above
12. Deadlock prevention is a set of methods _____.
    (a) to ensure that all of the necessary conditions do not hold
    (b) to decide if the requested resources for a process have to be given or not
    (c) to ensure that at least one of the necessary conditions cannot hold
    (d) to recover from a deadlock
13. For effective operating system, when to check for deadlock?
    (a) Every time a resource request is made
    (b) At fixed time intervals
    (c) Both (a) & (b)
    (d) None of the Above
14. To avoid deadlock _____.
    (a) there must be a fixed number of resources to allocate
    (b) resource allocation must be done only once
    (c) all deadlocked processes must be aborted
    (d) inversion technique can be used
15. For a deadlock to arise, which of the following conditions must hold simultaneously?
    (a) Mutual exclusion (b) No preemption
    (c) Hold and wait (d) All of the Above

Answers

1. (a)	2. (d)	3. (c)	4. (d)	5. (b)	6. (c)	7. (d)	8. (c)	9. (a)	10. (b)
11. (a)	12. (c)	13. (c)	14. (a)	15. (d)					

Practice Questions

Q.I Answer the following questions in short.

1. What is deadlock? Define it.
2. What are different methods to handle a deadlock?
3. What is the role of Max and Need array used in Banker's Algorithm?
4. Define the term 'Rollback and select a victim' in the context of deadlock recovery.
5. 'Wait-for-graph is used for deadlock avoidance in the system.' True/False. Justify.

Q.II Answer the following questions.

1. Explain necessary conditions for a deadlock.
2. Explain deadlock avoidance technique for a single instance of each resource type.
3. With the help of example describe resource-allocation graph.

4. What is wait-for-graph? Explain with example.
5. Explain Bankers algorithm with example.
6. What is difference between deadlock and starvation?
7. Write a short note on: (i) Recovery from deadlock(ii) Deadlock prevention.
8. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock-free.
9. What is a wait-for-graph? Draw wait-for-graph for the following: (Fig. 3.15).

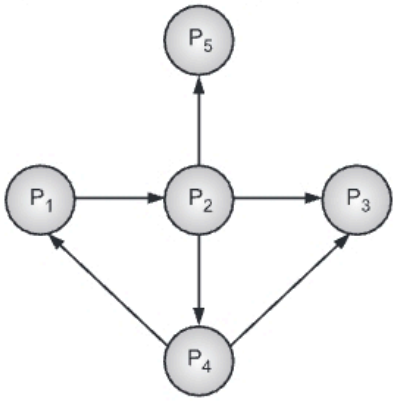


Fig. 3.15

Q.III Define the terms:

1. Starvation
2. Request Edge
3. Claim Edge
4. Allocation Edge
5. Rollback
6. Safe Sequence

Previous Exam Questions

Winter 2018

1. _____ is not a necessary condition in deadlock. **[1 M]**

(i) Safe state (ii) Mutual exclusion
(iii) Hold and wait (iv) Circular wait

Ans. (i).

2. The edge from process to resource in resource-allocation graph is called as _____. **[1 M]**

(i) Claim edge (ii) Allocation edge
(iii) Request edge (iv) None of these

Ans. (iii).

3. Explain deadlock recovery in detail. **[4 M]**

Ans. Refer to Section 3.7.

4. Differentiate between deadlock and starvation. **[4 M]**

Ans. Refer to Section 3.8.

5. Consider the given snapshot of the system, that has five processes and four resources: **[5 M]**

	Allocation			
	A	B	C	D
P_1	0	6	3	2
P_2	0	0	1	2
P_3	1	0	0	0
P_4	1	3	5	4
P_5	0	0	1	4

	Max			
	A	B	C	D
P_1	0	6	5	2
P_2	0	0	1	2
P_3	0	7	5	0
P_4	2	3	5	6
P_5	0	6	5	6

Available			
A	B	C	D
1	5	2	0

Answer the following questions using Banker's algorithm:

(i) What is the content of need matrix?

(ii) Is the system in safe state? If yes, give the safe sequence.

Ans. Refer to Section 3.5.3.

Summer 2019

1. A system is in safe state only if there exists a _____. **[1 M]**

(a) Continuous sequence

(b) Safe sequence

(c) None of above

Ans. (b).

2. State the necessary conditions for deadlock to occur. **[3 M]**

Ans. Refer to Section 3.3.1.

3. Discuss the symbols used for representation of resource allocation graph. **[3 M]**

Ans. Refer to Section 3.3.2.

4. Consider the following snapshot of the system. A system has 5 processes A through E and 4 resources type R_1 through R_4.

	Allocation			
	R_1	R_2	R_3	R_4
A	3	6	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Max			
R_1	R_2	R_3	R_4
4	1	1	1
0	2	1	2
4	2	1	0
1	1	1	1
2	1	1	0

Total			
R_1	R_2	R_3	R_4
6	3	4	

Answer the following question using Banker's algorithm:

If the system is in safe state give the safe sequence. **[4 M]**

Ans. Refer to Section 3.5.3.

■■■

4... Memory Management

Learning Objectives...

- To understand basic concepts of memory management like Physical and Logical Address Space, basic methods for memory utilization like Dynamic Loading, Dynamic Linking and sharing etc.
- To learn concept of Swapping.
- To know different memory management schemes like Contiguous Memory Allocation, Paging and Segmentation etc.
- To study Virtual Memory Management Techniques.
- To learn about thrashing.

4.1 INTRODUCTION [S-19]

- Main memory is storage area of quickly accessible data which is shared by CPU and I/O devices.
- Main memory is a large array of words or bytes. Each word or byte has its own address. For any program to be executed, it must be in main memory along with data.
- CPU reads instruction from main memory during Instruction Fetch Cycle and performs read and write operation in main memory during Data Fetch Cycle. Thus, it is storage device that CPU is able to address and access directly.
- To improve the utilization of the CPU and the speed of the computer's response to its users; we must keep several processes in memory; i.e. we must share memory. Thus, multiprogramming creates need of memory management.
- The main task of memory management is "efficient memory utilization".
- Operating system is responsible for the following activities in connection with memory management:
  1. To keep track of all memory locations- free or allocated. If allocated, to which process and how much.
  2. Deciding which processes and data to move into and out of memory.
  3. Allocating and de-allocating memory space as needed according to various memory management techniques and algorithms.

- There are various memory management schemes and techniques for managing memory. In selecting memory management scheme, various issues like hardware design, protection, binding of symbolic addresses to physical addresses, distinguishing logical and physical addresses, dynamic loading and linking etc. must be considered.

4.1.1 Basic Hardware

- Main memory and registers are only the storage that CPU can access directly. For any instruction in execution and any data being used by instruction must be in one of these storage devices.
- Registers are accessible generally in one clock of CPU. But main memory take more cycles of CPU to complete as CPU has to wait for data which is required for completing. To avoid this situation, a fast memory i.e. cache is added between main memory and CPU.
- In addition to speed of accessing memory, we must also ensure correct operation has to protect operating system from user access and user process from one another.
- This protection must be provided by the hardware. Two registers, base register and limit register can be used to provide protection as shown in Fig. 4.1.
- The base register holds the starting base address or smallest legal physical memory address and limit register specifies size of the range.

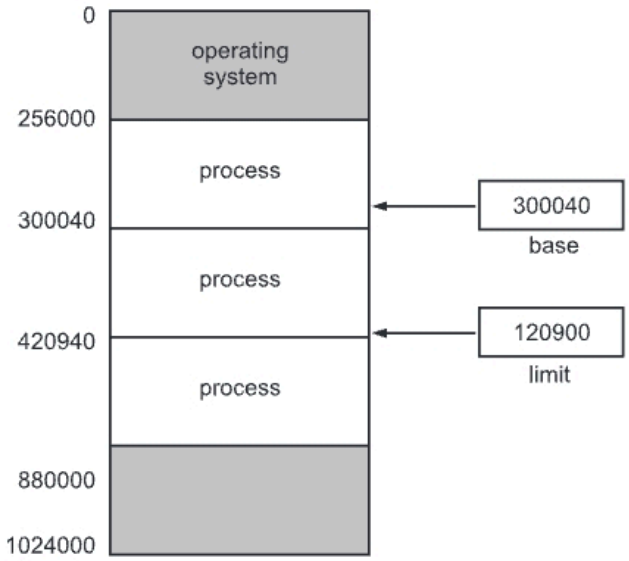


Fig. 4.1: A Base and A Limit Register define a Logical Address Space

- For example, if base register holds address 300040 and limit register contains range 120900 then process can access legal addresses from 300040 to 420940.
- Protection of memory space is accomplished by comparing every address generated by user's program with these registers.
- Any attempt made by user's program to access operating system's memory or other user's memory results in a trap to the operating system as shown in Fig. 4.2. Thus, this method prevents a user program from modifying code or data of operating system or other users.
- The base and limit registers can be loaded by operating system using special privileged instruction in kernel mode.

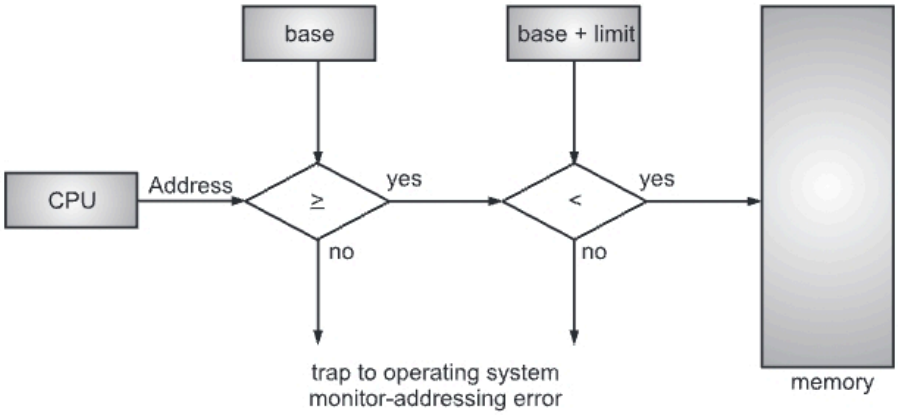


Fig. 4.2: Hardware Address Protection with Base and Limit Registers

4.1.2 Address Binding

- Generally, programs reside in executable form on the disk. For execution, program must be brought into memory and placed within process.
- So, to create an active process in memory, symbolic addresses generated by the program must be mapped into the process address space. This mapping is known as Address Binding. As process is executed, it accesses instructions and data from memory.
- Classically binding of instructions and data can be done at any step, (shown in Fig. 4.3) as follows:
    - **Compile Time:** If you know at compile time where the process will reside in memory, then absolute code can be generated. If starting location changes, then it will be necessary to recompile this code. In MS-DOS, .com files are bound at compile time.

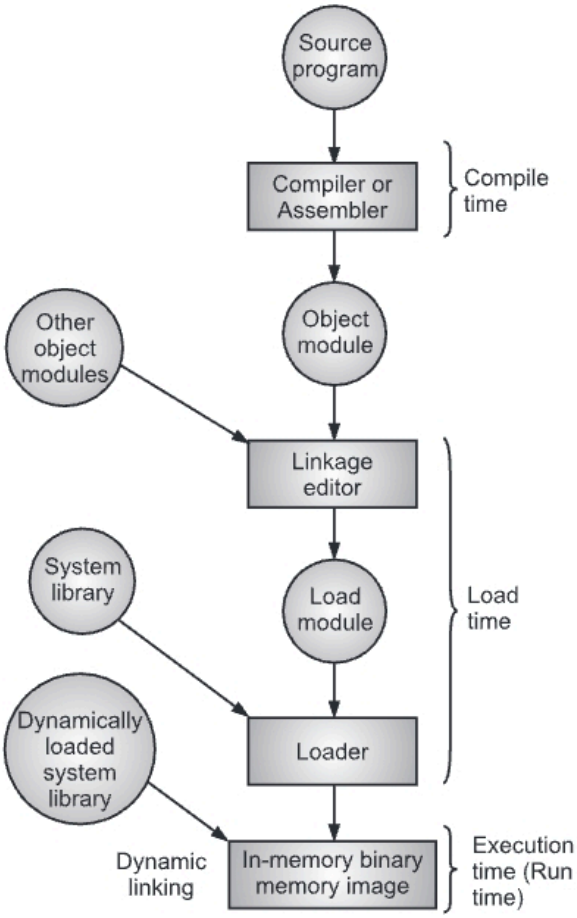


Fig. 4.3: Address Binding

- **Load Time:** If it is not known at compile time where the process will reside memory, then compiler generates relocatable code. Then binding is delayed until load time. If the starting address is changed, only reload the code to changed address.
- **Execution Time:** If the process can moved during its execution from one memory segment to another, then binding must be delayed until run time. Special hardware is needed to implement this scheme.

4.1.3 Logical Versus Physical Address Space

- An address generated by the CPU is commonly known as Logical Address or Virtual Address. The address seen by the memory unit that is one loaded into the memory-address- register of the memory is commonly referred as the Physical Address.
- The compile time and load time address binding generates the identical logical and physical addresses. However, the execution time address binding scheme generates differing logical and physical addresses as process moves from one memory segment to another.
- The set of all logical addresses generated by a program is known as **Logical Address** Space.
- The set of all physical addresses corresponding to these logical addresses is the **Physical Address** Space.
- Now, the run time mapping from virtual address to physical address is done by a hardware device known as Memory Management Unit (MMU).
- Different methods like contiguous allocation, paging, segmentation etc. can perform such mapping.

Example:

- The simplest memory mapping scheme is shown in the Fig. 4.4.

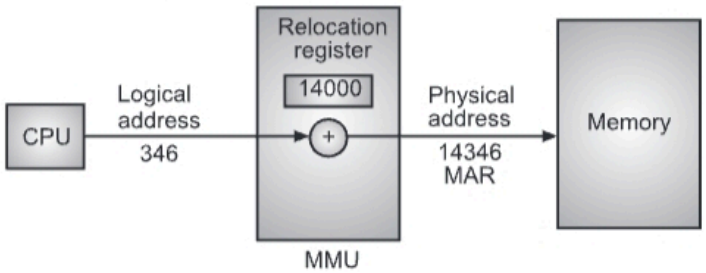


Fig. 4.4: Dynamic Relocation using Relocation Register

- Here, base register is known as Relocation register. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory.
- Let's understand this situation with the help of example: If the base register contains the value 14000, then an attempt by the user to address location 346 is mapped to location 14346.
- The user program never sees the real physical address space, it always deals with the Logical address.
- Logical address is in the range (0 to max) and Physical address is in the range (R to R+max) where R is the value of relocation register.
- As it is clear from the above text that user program supplies only logical addresses, these logical addresses must be mapped to physical address before they are used.

4.1.4 Dynamic Loading

- For process to execute, the entire program and data must be in physical memory. So the size of process is limited to size of physical memory.
- But all routines of program are not required at a time or program may contain some error routines which may be never called.
- To obtain better memory utilization, routines can be loaded in main memory as required. This is known as **Dynamic Loading**.
- Thus, in dynamic loading:
  - A routine is not loaded until it is called.
  - All routines are kept on disk in relocatable load format.
  - Only main routine is loaded into memory and is executed.
  - When routine needs to call another routine, the calling routine checks whether that routine has been loaded. If not, then relocating linking loader is called to load the desired routine and update program's address table.

Advantages of Dynamic Loading:

1. Unused routine is never loaded.
2. Better memory utilization.
3. Useful when large amount of code is needed to handle infrequently occurring cases.
4. Logical to physical address binding is at runtime.
5. No special support from the operating system is required, can be implemented through program design.

4.1.5 Dynamic Linking and Shared Libraries [S-19]

- In static linking, system language libraries are treated as like any other object module and are combined by the loader into binary program image.

4.1.5.1 Dynamic Linking

- In dynamic linking, linking is postponed until execution time. This feature is usually useful in system libraries such as language subroutine libraries.
- Without this facility, each program on system must include one copy of language library. With dynamic linking, a stub is included in the image for each library routine reference.
- The stub is a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not already present.
- When stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory. Either way, the stub replaces itself with address of the routine and executes the routine.
- Thus, the next time that particular code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.

Advantages:

1. All processes that use a language library execute only one copy of the library code in memory.
2. The above feature can be extended to library updates (such as bugs fixes).
3. A library may be replaced by a new version, and all programs that reference the library will automatically use the new version.
4. Without this support, such programs would need to be re-linked to gain access to the new library.

4.1.5.2 Shared Libraries

- More than one version of a library may be loaded into memory, and each program uses its version information to decide which copy of library to use. Major changes increments the version number and minor changes retain the same version number.
- Thus, only programs that are compiled with new library version are affected by the incompatible changes incorporated in it.
- Other programs linked before the new library was installed will continue using the older library. This system is also known as Shared Libraries.

4.1.6 Overlays [W-18]

- The entire program and data of process must be in physical memory for the process to execute. The size of a process is limited to the size of physical memory. So that a process can be larger than the amount of memory allocated to it a technique called Overlays is sometimes used.
- The idea of overlays is to keep in memory only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space that was occupied previously by instructions that are no longer needed.
- **For example:** consider two pass assembler. During Pass-1 it constructs a symbol table, and then during Pass-2 it generates machine language code. We may be able to partition such assembler into Pass-1 code, Pass-2 code, the symbol table and common support routines used by both Pass-1 and Pass-2. Assume the sizes of these components are as follows:

Pass-1	70K
Pass-2	80K
Symbol table	20K
Common support routines	30K

- To load everything at once we would require 200K of memory. If only 150K is available, we cannot run our process. Notice that Pass-1 and Pass-2 do not need to be in memory at the same time. Thus we define two overlays:
  - Overlay A is the symbol table, common routines and Pass-1.
  - Overlay B is the symbol table, common routines and Pass-2.

- We add on overlay driver (10K) and start with overlay A in memory. When we finish PASS-1, we jump to the overlay driver, which reads overlay B into memory, overwriting overlay A and then transfers control to PASS-2.

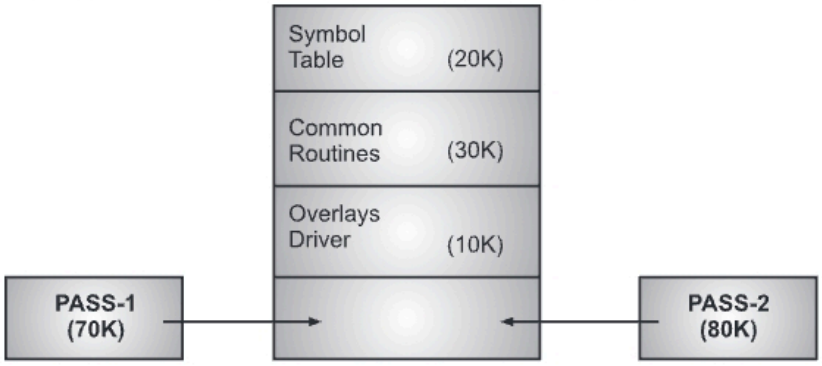


Fig. 4.5: Overlays for a Two-pass Assembler

- Overlay needs only 120K; whereas overlay B needs 130K we can now run our assembler in the 150K of memory.
- The code for overlay A and the code for overlay B are kept on disk as absolute memory images, and are read by the overlay driver as needed.
- Special relocation and linking algorithms are needed to construct the overlays. In dynamic loading overlays do no requires any support from the operating system. They can be implemented by the user with simple file structure.
- The programmer must design and program the overlay structure properly, requiring complete knowledge of the structure of the program, its code and its data structures, because the program is by definition, large, obtaining a sufficient understanding of the program, may be difficult for these reasons.
- The use of overlays is currently limited to microcomputers and other systems that have limited amounts of physical memory and that task hardware support for more advanced techniques. Some microcomputers compilers provide to the programmer support of overlays to make the task easier.

4.2 SWAPPING [W-18]

- The process to be executed must be in main memory. A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. This process is known as **Swapping.**
- Swapping requires backing store. Backing store is fast disk large enough to accommodate copies of all memory images for all users. It must provide direct access to these memory images as shown in Fig. 4.6.
- System maintains a ready queue of ready-to-run processes which have memory images on disk. When CPU scheduler decides to execute a process, dispatcher checks to see whether next process in the queue is in memory.
- If it is not, the dispatcher swaps out a process currently in memory and swaps in desired process along with context of process.
- If static binding is used, then process that swapped out will be swapped in same memory space it occupied previously.

- In case of dynamic binding, process can be easily moved in different locations. Variant part of swapping policy is roll out, roll in used for priority-based scheduling algorithms. Lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Context switch time is fairly high in swapping system. For example, we assume that the user process is 10MB, and the standard hard disk with a transfer rate of 40MB per second, actual transfer would to and from memory would take:

  10000KB/40000KB per sec = ¼ second = 250 milliseconds.
- Assume we expect 8 milliseconds of delay; each swap will take 258 milliseconds. And we need two swaps (in and out) therefore it takes 516 milliseconds. Thus major part of swap time is transfer time. Total transfer time is directly proportional to the amount of memory swapped.

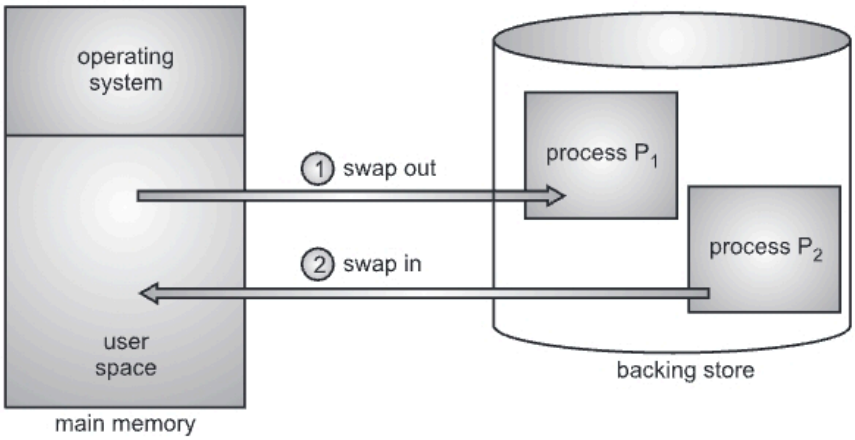


Fig. 4.6: Swapping of Two Processes using a Disk as a Backing Store

- Swapping is constrained by some factors like never swap a process with pending I/O or execute I/O operations.
- Because if we swap out a process which is executing or waiting I/O operations and swap in desired process, then I/O operation might attempt to use memory that belongs to desired process.
- Currently, standard swapping is used in few systems. It requires too much swapping and less execution. Modified versions of swapping are found on many systems i.e., UNIX, Linux, and Windows.

4.3 CONTIGUOUS MEMORY ALLOCATION

- Operating System allows accommodation of multiple processes in memory.
- Memory needs to be allocated efficiently in order to allow as many processes into memory as possible.
- If not enough processes are in memory, multiprogramming will not be effective and the CPU will be idle most of the time.

- Main memory usually divided into two partitions as shown in Fig. 4.7 i.e.,
  1. **Resident operating system**, usually held in low memory with interrupt vector.
  2. **User processes** held in high memory.
- Therefore, we need to discuss memory mapping and protection, contiguous memory allocation method.

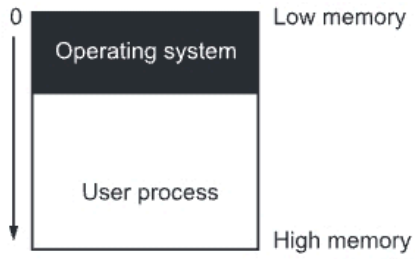


Fig. 4.7: Memory Partitions

4.3.1 Memory Mapping and Protection

- Relocation registers are used to protect user processes from each other, and from changing operating-system code and data.

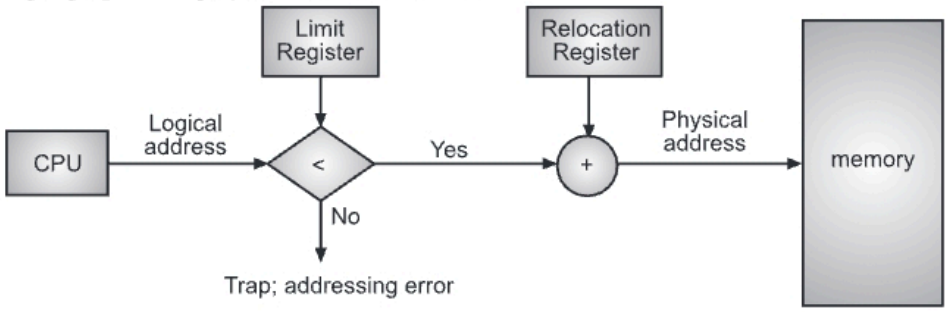


Fig. 4.8: Hardware support for Relocation and Limit Registers

- Relocation register contains value of smallest physical address. Limit register contains range of logical addresses – each logical address must be less than the limit register.
- MMU maps the logical address dynamically by adding the value in the relocation register to generate physical address which is sent to memory.
- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- The relocation register scheme provides an effective way to allow the operating system size to change dynamically. For example, Transient code such as device drivers are loaded as needed. So it is not kept in memory.
- With help of relocation register, this code can be loaded during program execution. So size of operating system changes dynamically.

4.3.2 Memory Allocation

- Memory can be partitioned by following two approaches:
  1. Single Partition Allocation.
  2. Multiple Partition Allocation.

4.3.2.1 Single Partition Allocation

- This is simplest method of memory management. The memory is divided in two sections one section for Operating System and second section for User Program.

- In order to provide a contiguous area of free storage for user program, operating system is loaded at one end (usually in low or bottom part).
- The relocation registers points to first location in user's partition. User's logical address is adjusted by hardware to produce physical address.
- Address binding is delayed until execution time.
- Relocation register value is static during program execution. Hence, all of the Operating system must be in memory.
- A new program (user process) is loaded only when operating system passes control to it. After receiving control, it starts running until its completion or termination due to I/O or some error.
- When this program is completed or terminated, the operating system may load another program for execution.
- Protection of operating system from user process is achieved through limit register. It is set to highest address occupied by operating system code.
- Memory address generated by user process to access certain memory location is compared with limit register. If address is more than limit register, then trap will be generated and permission is denied.

Advantages:

1. It is simplest method.
2. Sharing of data and code is very easy.

Disadvantages:

1. It only supports single process environment, for example MS-DOS (i.e. It does not support multiprogramming.
2. Less utilization of CPU. CPU will be remaining idle during I/O operation.
3. Less utilization of memory. Since only one program is residing at a time in memory, it may not occupy whole memory.

4.3.2.2 Multiple Partition Allocation

- In multiprogramming environment, several programs reside in main memory at a time. To support multiprogramming, main memory is divided into several partitions; each of which is allocated to a single process.
- The number of programs residing in memory will be bound by the number of partitions. Depending upon how and when partitions are created, there may be two types of approaches:
  1. Multiple contiguous fixed (static) partition allocation.
  2. Multiple contiguous variable (dynamic) partition allocation.

4.3.2.2.1 Multiple Contiguous Fixed (Static) Partition Allocation

- In this method, memory is divided into several partitions of fixed size that size never changes.
- Each partition method holds one process. This method is originally used in the IBM OS/360 operating system. It is also known as Multiprogramming with Fixed number of Tasks (MFT).

- Jobs are assigned to these fixed partitions by either of the following methods:
  1. There will be a separate ready queue for each of regions. A region will be big enough to satisfy memory requirement of all jobs in its own queue. In this scheme, a system will have to pre-pass over the common ready queue, in which all the jobs were initially placed. And then assign each of these jobs to the queue of appropriate region. Each queue is scheduled separately.

     Example: One queue for 2K, one queue for 6K, and a queue for 12K. When a job arrives then it is put on the appropriate queue if there is no memory partition available. If a job of 4K arrives and there is no partition of size 4K then it is put in the queue of larger partition (6K) as shown in Fig. 4.9.

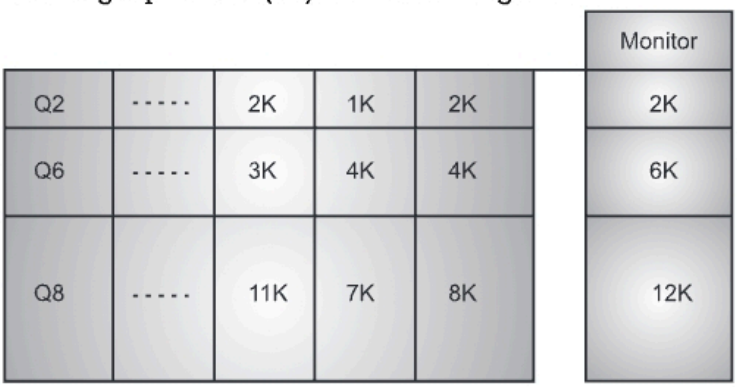


Fig. 4.9: MFT with separate queues for each partition

  2. In the other scheme, all the jobs are placed in only one queue. Whenever a partition is free, job selected by FCFS algorithm, Round Robin algorithm, or Priority algorithm is assigned to the partition as shown in following Fig. 4.10.

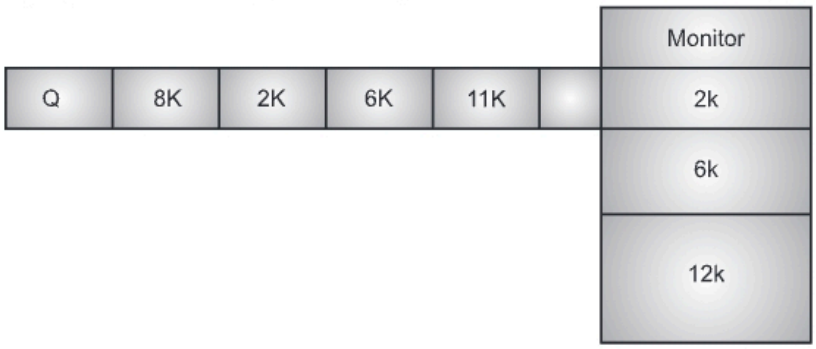


Fig. 4.10: MFT with one queue

- If one job is executing, and a high priority job from appropriate partition is swapped out /rolled out and another high priority job is swapped/rolled in for execution. After execution of this job, generally the swapped out job is brought back in the same region.

- Otherwise, that job will again have to undergo relocation and calculation of new address space for it. If after a memory is assigned to a job, the memory requirements of the job increase, then either of following situations takes place.
  - An error message is given to the user and a job is terminated.
  - Control returns to the user along with error and user decides either to modify or to exit.
  - System can:
    1. Swap out the job.
    2. Wait for lager region.
    3. Swap into lager region.
    4. Continue.
  - Overall performance of MFT depends upon correct selection of region size.
  - There are two major things to be selected and decided:
    1. How many regions to have?
    2. What will be the size of each region?
  - The decision regarding the size of the region is generally based on educated guess of the memory requirement of input programs.
  - For which one may have to study the kind and types of job which will be submitted to the system.

Different Placement approaches:

- Generally, the set of region is searched to determine which region is best to allocate. First fit, best fit and worst fit are most common strategies used to select a free region from the set of available regions.
  1. **First-fit:** The first-fit approach allocates the first free partition large enough to accommodate the process. Searching can start either at the beginning of the set of holes or where the previous first fit search ended.
  2. **Best-fit:** The best-fit approach allocates the smallest free partition that meets the requirements of the process. Under consideration we must search the entire list, unless the list is kept ordered by size. This strategy produces the smallest leftover partition.
  3. **Worst-fit:** The worst–fit approach allocates the largest free partition. Again, we must search the entire list, unless it is sorted by size. This strategy produced to largest leftover partition, which may be more useful than the smaller than leftover partition from a best – fit approach.

Advantages:

1. At the compile time we can blind address or only the address of each word can be given.
2. Algorithms needed to implement Fixed Partitioning are easy to implement.
3. Processing of Fixed Partitioning require lesser excess and indirect computational power.

Disadvantages:

1. Process can not demand more memory than allocated partition.
2. It does not support for dynamic relocation.
3. It does not support a system having dynamically data structures such as queue, stack and heap etc.
4. It suffers from internal fragmentation, (Fragmentation is discussed in next Section 4.3.3).

4.3.2.2.2 Multiple Contiguous Variable (Dynamic) Partition Allocation

- In this scheme, region or partition size is not fixed; it can vary dynamically. It creates partition according to requirements of process. It is also known as **Multiprogramming with Variable Number of Tasks (MVT)**.
- Operating system maintains a table indicating which parts of the memory are free and which are allocated.
- Whenever a job requests for the memory, the table indicating the current status of the memory is referred.
- If memory is available, then the job is assigned to partition according to job scheduling policy and the table is updated to reflect the new status.
- When the job terminates, it releases the memory occupied by it. Therefore, at any instance of time, the snapshot of memory will show blocks of allocated memory and free holes distributed all over the memory as shown in Fig. 4.11.

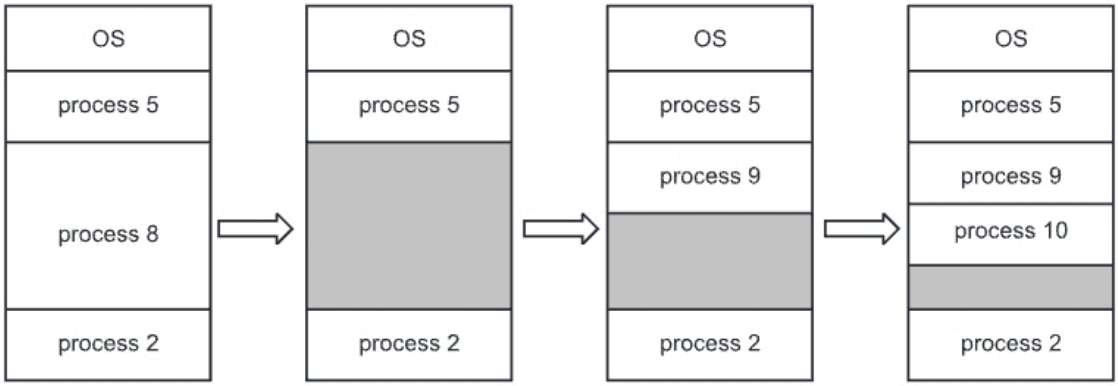


Fig. 4.11: Job Scheduling with MVT

Advantages:

1. It increases degree of multiprogramming.
2. It does not suffer from internal fragmentation.
3. Size of process can grow or shrink dynamically.

Disadvantages:

1. When user starts with, there is a very big hole of available memory. As job go on requesting and releasing the memory, this big hole is divided into small used blocks and unused holes. Such many holes are distributed in the memory, which

do not satisfy memory requirement of any job. They form the external fragmentation. The worst part with external fragmentation is wastage of memory.

2. Lot of time is consumed by the system to search all these holes, when a memory request arrives.

4.3.3 Fragmentation

- As jobs are allocated and de-allocated, memory is fragmented into small chunks. One of reason of fragmentation is improper selection of region size.

1. Internal Fragmentation:

- Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used is known as Internal fragmentation.
- For example, if the hole is the size of 20,000 bytes, suppose that next process requests 19,800 bytes. 200 bytes are lost. This is called internal fragmentation; i.e. memory that is internal to a partition but is not being used. The general approach to avoiding this problem is to break the physical memory into fixed size blocks and allocate memory in units based on block size.

2. External Fragmentation:

- As processes are loaded and removed, the free memory is broken into little pieces. Total memory space exists to satisfy a request, but it is not contiguous then it is known as External fragmentation.
- As shown in Fig. 4.12, first fit and best fit method suffers from external fragmentation.

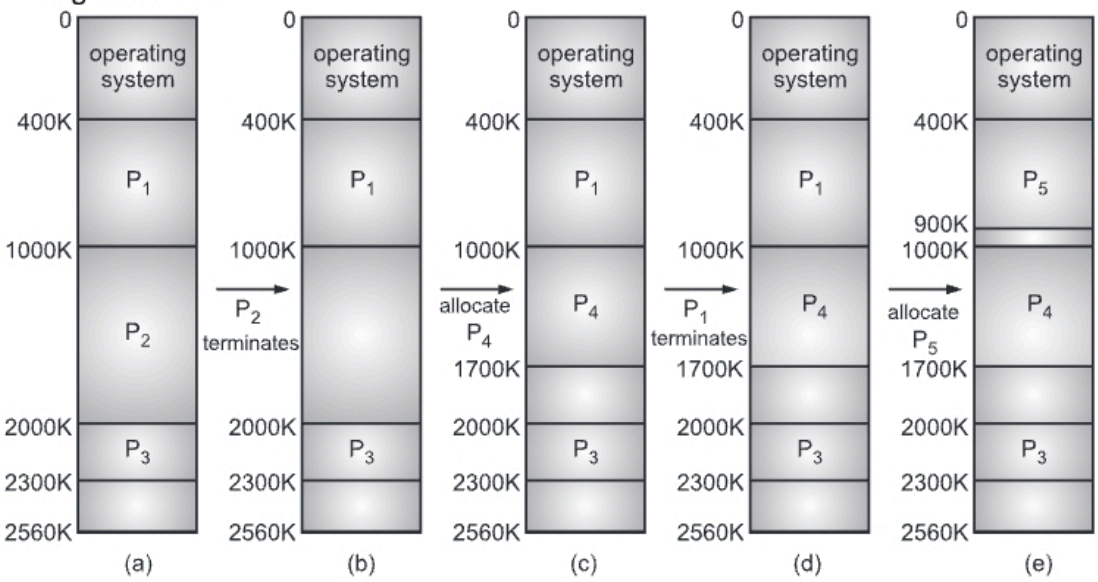


Fig. 4.12: External Fragmentation

- Depending on the total amount of memory storage and average process size, external fragmentation may be major or minor problem.

- Statistical analysis of first fit even with some optimization, given N allocated blocks, another 0.5N blocks will be lost to fragmentation.
- That is one-third of memory may be unusable! This property is known as the 50-percent rule.

Table 4.1: Difference between Internal Fragmentation and External Fragmentation [W-18]

Sr. No.	Internal Fragmentation	Internal Fragmentation
1.	It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space is allocated to the processes dynamically.
2.	In this fragmentation the fixed size partition is assigned to a process whose size is less than the size of the partition due to which the rest of the space in the partition becomes unusable.	In this fragmentation the memory space in the system can easily satisfy the requirement of the processes, but this available memory space is non-contiguous, so it cannot be utilized further.
3.	It mainly refers to the unused space in the partition that resides in the allocated region	It mainly refers to the unused blocks of the memory that are not contiguous and hence are unable to satisfy the requirements of the process.
4.	Internal fragmentation can be eliminated by allocating memory to processes dynamically.	External fragmentation can be eliminated by compaction technique.

4.3.3.1 How to remove External Fragmentation? [S-19]

- External fragmentation can be avoided by:
  1. **Compaction** is process of collecting all free holes and forms a single big free hole. It shuffles memory contents to place all free memory together in one large block. Compaction is possible only with dynamic relocation i.e. logical addresses be relocated dynamically, at execution time. If addresses are relocated only at load time, we can not compact storage. One must consider the cost of compaction while implementing compaction algorithm.
     - Simple compaction algorithm is to move all jobs towards one end of memory and all free holes to the other end of memory. This method is very expensive. As almost all jobs will change their address space, they will have to undergo relocation.
     - Variation to this algorithm says that, a free space will be created in middle of the memory. If swapping is a part of the system, then that can be combined with compaction e.g. a job to be shifted will be rolled out to a backing storage and later rolled into some different address space. In this method, actual code for compaction will be very less, as system already have code for roll-out and roll in.

- Frequency of compaction algorithm to be executed will vary system wise. One approach says, compaction should be called immediately after execution of job is over. Other approach may say, compaction must be called, only when it is required or may be after some regular interval of time. Compaction though is an extra overhead; it solves the problem of external fragmentation to maximum extent.

2. Another solution to external-fragmentation problem is **to permit the logical address space of the process to be non-contiguous**. Thus, allowing a process to be allocated physical memory wherever the space is available. Two complementary techniques achieve this solution: Paging and Segmentation.

- Let us compare MFT and MVT algorithm:

Table 4.2: Comparison of MFT and MVT Algorithm

Factors	MFT	MVT
Region Size	In this method, memory is divided into several partitions of fixed size that size never changes.	In this method, region or partition size is not fixed; it can vary dynamically.
Process Size	Can not grow at run time.	Can grow or shrink at run time.
Degree of multi-programming	Fixed (i.e. number of partitions).	Dynamic.
Memory utilization	Poor	Good
Fragmentation	Suffers from internal fragmentation.	Suffers from external fragmentation.
Overhead	Eliminates the overhead of run-time allocation of partitions.	Runtime overhead of allocation of partitions.
Example	IBM 360, DOS etc.	IBM OS.

4.4 PAGING

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous. It avoids external fragmentation.

4.4.1 Basic Method

- The basic method for implementation involves breaking physical memory into fixed sized blocks called frames and break logical memory into blocks of the same size called **Pages**.

- The hardware support for paging is illustrated in Fig. 4.13.

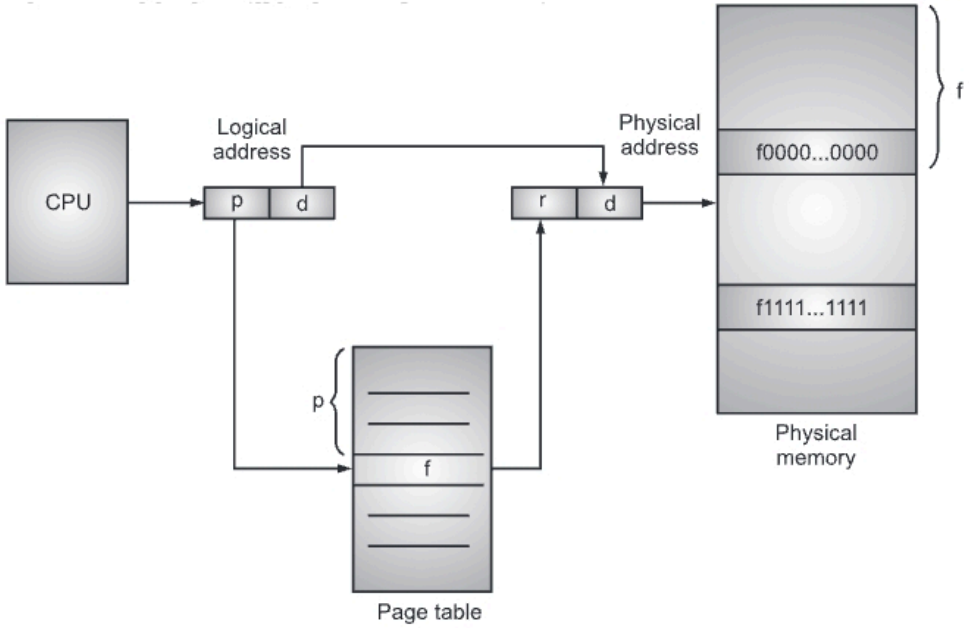


Fig. 4.13: Paging Hardware

- Every address generated by the CPU is divided into two parts namely, Page number(p) and Page offset(d). The **page number(p)** is used as an index into a Page Table. A page table is used to map a page on a frame.
- **Page table** has an entry for each page. The page table has the base address of each page in physical memory; which is combined with page offset(d) to define the physical memory address which is sent to memory unit.
- The paging model of memory is shown in Fig. 4.14.

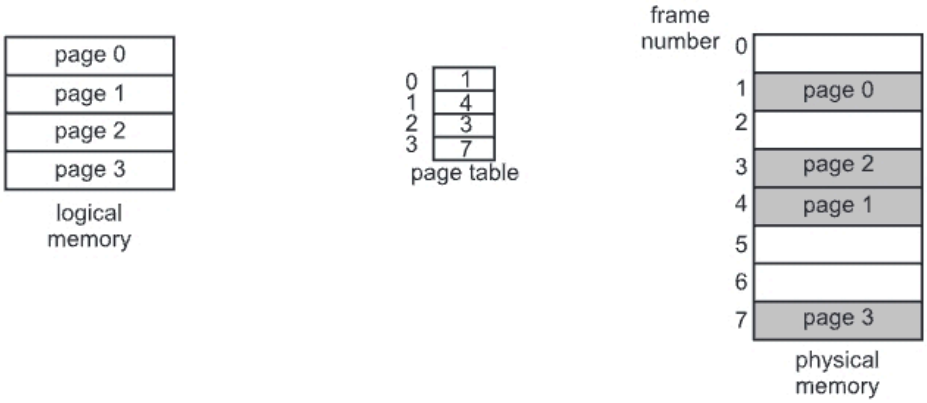
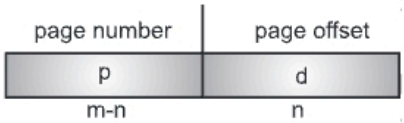


Fig. 4.14: Paging Model of Logical and Physical Memory

- Page size and frame is same and decided by hardware and is generally a power of 2 varying between 512 bytes and 16MB per page.

- Reason: If the size of logical address is 2^m and page size is 2^n, then the high-order m-n bits of a logical address designate the page number and n low-order bits designate the page offset. Thus logical address is as follows:

  Where, p is an index into page table and d is displacement within page.



- For example, consider the memory as shown in Fig. 4.15 using page size of 4 bytes and physical memory of 32 bytes (8 pages).
- Let us see how the user's view of memory can be mapped into physical memory by following steps:
  1. Find out Page number (p) = Logical address /page size (integer division).
  2. Find out Page offset (d) = Logical address % page size.
  3. Check if offset is smaller than page size otherwise; a trap is to be generated.
  4. Using page number (p) as an index into page table; obtain equivalent frame number.
  5. Find out the base address of the frame.

     Base address of a frame = (frame number *frame size) or (frame number * page size)

     Physical Address = Base address of a frame + Page offset.

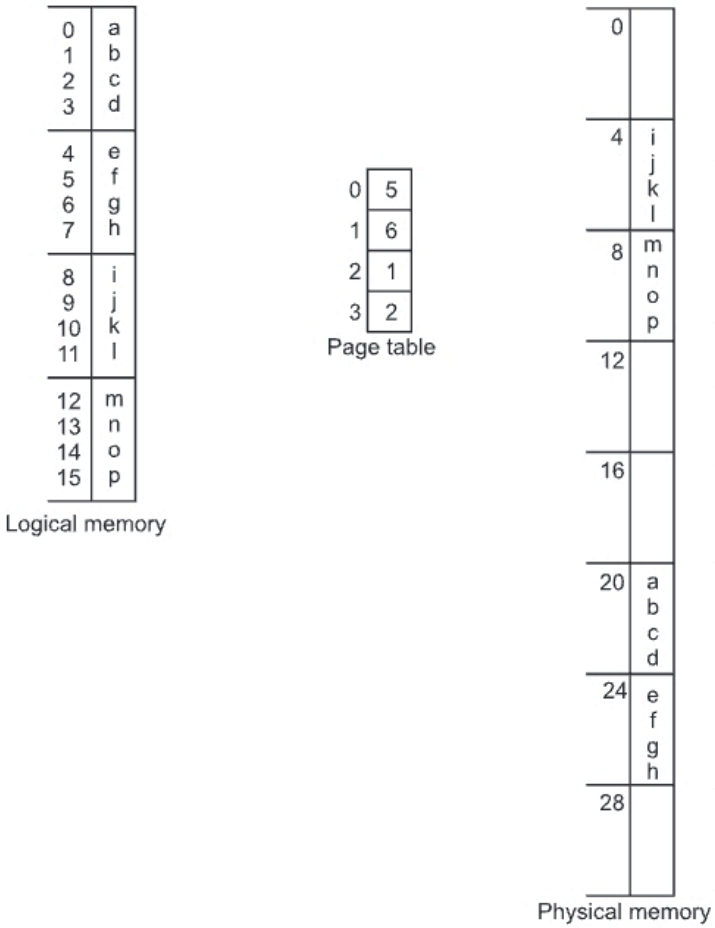


Fig. 4.15: Mapping of logical to physical memory

- Suppose Logical address 3 can be mapped to physical address 23 as follows:
  - Page number (p) = Logical address /page size => 3/4 = 0
  - Page offset (d) = Logical address % /page size => 3%4 = 3

- Check if offset is smaller than page size (i.e. 3 < 4) if yes, then find frame number using page number 0 in page table.
  - Frame number = 5
  - Base address of a frame = (frame number * page size) => 5 * 4 = 20
  - Physical Address = Base address of a frame + Page offset => 20+3=23
    - When process arrives in the system, its size is expressed in terms of number of pages.
    - Each page of the process needs one frame. If the process requires n pages, at least n frames are required in memory.
    - The first page of the process is loaded into the first frame listed on free-frame list, and the frame number is put into page table as shown in Fig. 4.16.
    - Operating system must be aware of which frames are allocated and which frames are free in physical memory. This information is kept in a data structure called a frame table.

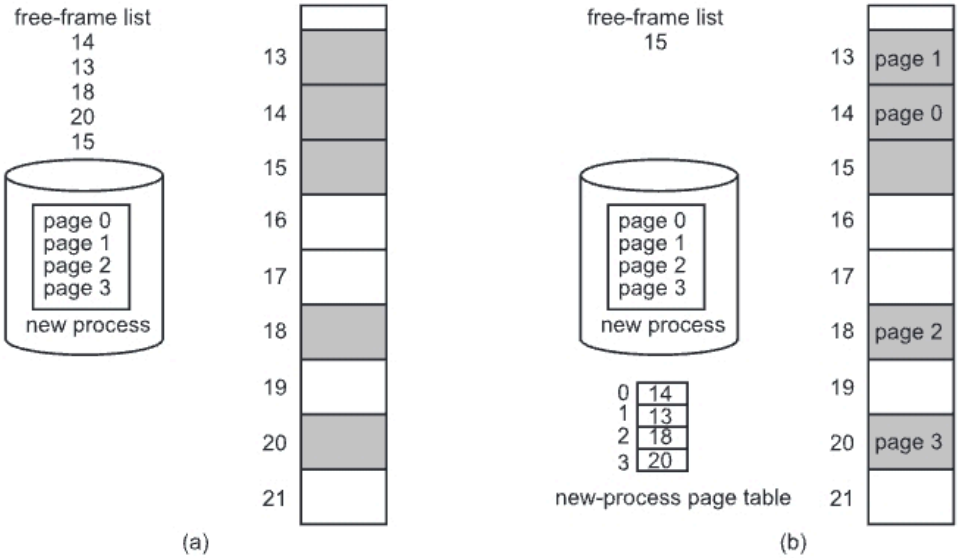


Fig. 4.16: Free Frames (a) Before Allocation (b) After Allocation

4.4.2 Hardware Support

- Every access to any of page has to go through the page table. Therefore, implementation of the page table directly affects the performance of this kind of memory management.
- Though there are different ways on which page table can be implemented, generally the method depends upon the size of the page table.

1. **Page Table Implemented as Dedicated Fast Registers:**
   - As registers are very fast, access to table is also very fast.
   - This is a costly scheme.
   - As a page grows, number of registers may not be available, neither affordable nor feasible.
   - Therefore scheme is useful only for small page tables.

2. **Page Table Implemented in the Main Memory:**
   - This scheme is used when page table is big.
   - Page table is stored in the main memory.
   - PTBR (Pointer To Base Register) points to the base address of page table.
   - In case, location of the page table is changed in the memory, only the contents of PTBR are changed.
   - This scheme gives reasonably slow access. The memory access is to be done twice, once for accessing page table for getting frame number and secondly for accessing actual frame.
3. **Page Table Implemented in Associative Registers:**
   - The above problem can be solved by using special, small, fast lookup hardware cache, called a Translation Look-Aside Buffer (TLB) or associative registers as shown in following Fig. 4.17.

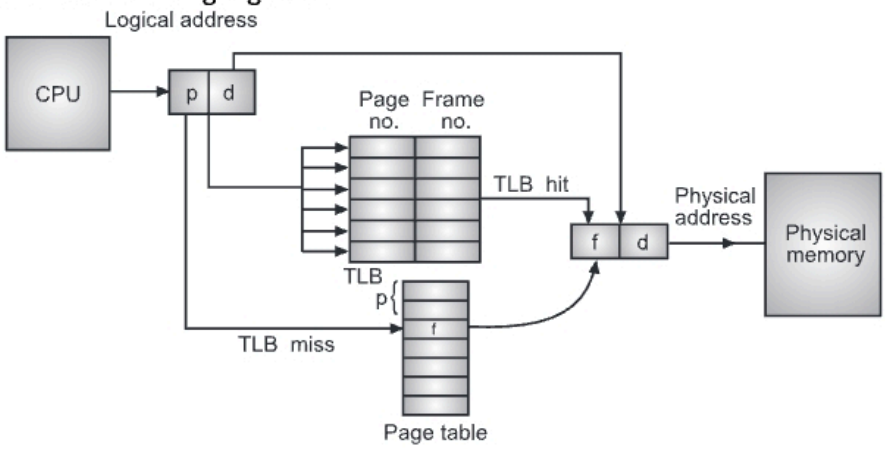


Fig. 4.17: Paging hardware with TLB

   - Each entry in the TLB consists of two parts: **Key** and **Value**. When associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned.
   - The TLB is used with page tables in the following way: The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If page number is found, its frame number is immediately available and is used to access memory.
   - If the page number is not in the TLB (TLB miss) a memory reference to the page table must be made. In addition, we add the page number and frame number into TLB.
   - If the TLB already full, the OS have to select one for replacement.
   - Some TLBs allow entries to be wire down, meaning that they cannot be removed from the TLB, for example kernel code.
   - The percentage of times that a particular page number is found in the TLB is called **Hit ratio**.

- An 80-percent hit ratio means that we find the desired page number in TLB 80 percent of time. If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then mapped memory access takes 120 nanoseconds when page is found in TLB. If TLB miss is there, then 220 nanoseconds will require for desired bytes. To find the effective memory access-time, weight each case by probability.
- Effective memory access time = $0.8 \times (100 + 20) + 0.2 \times (120 + 100) = 140$.
- If our hit ratio is 98%, the effective memory access time
  $= 0.98 \times (100 + 20) + 0.02 \times (120 + 100) = 122$.

4.4.3 Protection

- Memory protection in a paged environment is supported by protection bits associated with each frame. Normally, these bits are kept in page table.
- One bit can define a page to be read-write or read only. Every reference to memory goes through page table to find correct frame number. At same time, protection bits can be checked to verify that no writes are being made to a read only page. An attempt to write to a read only page causes hardware trap to the operating system.
- One more bit is attached to each entry in the page table: **a valid-invalid bit.** When bit is set to "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page. When bit is set to "invalid" indicates that the page is not in the process' logical address space. Illegal addresses are trapped by using this bit.
- Consider an example as shown in Fig. 4.18.
  - Address Space: 14 bits, (Address is from 0 to 16,383).
  - Page size: 2K (2048 words)
  - Program size: 0 to 10,468 (pages from 0 to 5).
  - Total number of pages that can be addressed: 16,383/2048 = 8.
  - Any attempt to generate an address in pages 6 or 7, will find that valid-invalid bit is set to invalid, trap will be generated.

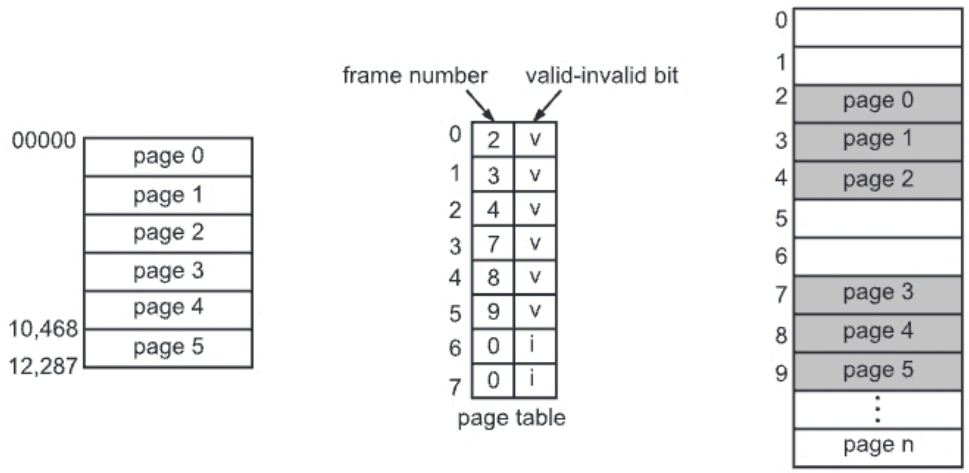


Fig. 4.18: Valid (v) or Invalid bit(i) in Page Table

4.4.4 Shared Pages

- An advantage of paging is the possible of sharing common code, especially time-sharing environment.
- For example, consider a server with 40 user using text editor (with 150k reentrant code and 50k data space) as shown in Fig. 4.19, we see three page editors with 50k each. Each process has its own data page. So we need (150k + 50K) × 40 = 8000KB memory space. Lot of memory wastage. But reentrant code or pure can be shared. Reentrant Code or pure code is non-self modifying code; it never changes during execution.
- Thus, two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold data for process executions.
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems, run-time libraries and database systems).
- Shared code must appear in same location in the logical address space of all processes. Thus, to support 40 users we need only 150k + 40 × 50k = 2150 KB memory space. Some operating systems implement shared memory using shared pages.

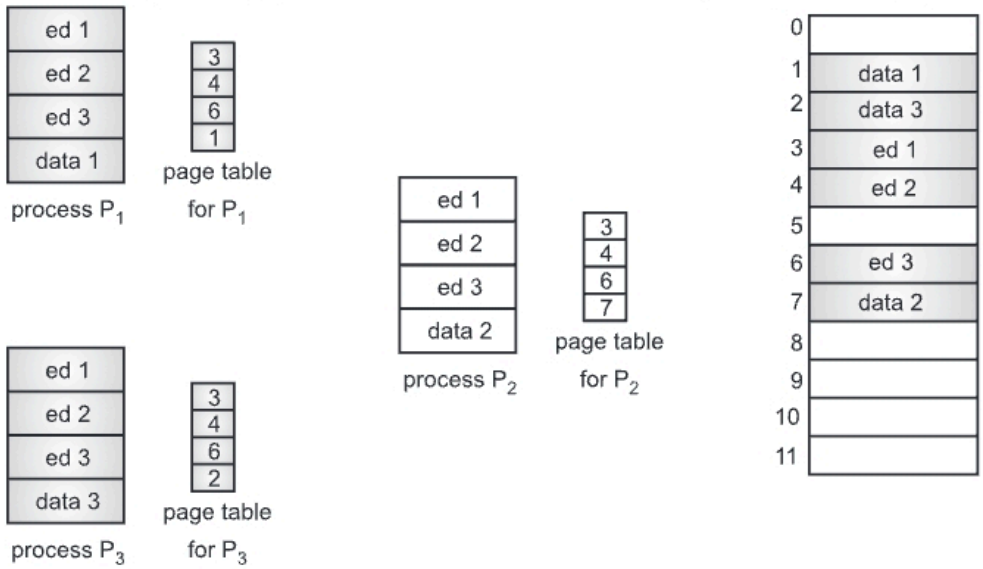


Fig. 4.19: Sharing of Code in a Paging Environment

Advantages:

1. It permits physical address space of process to be non-contiguous and support virtual memory concept.
2. It maintains clear separation between user's view of memory and actual physical memory.
3. It does not require any support for dynamic relocation because paging itself is a form of dynamic relocation. Every logical address is bound by paging hardware to physical address.

4. It does not suffer from external fragmentation. Any free frames can be allocated to process that it needs.
5. We can share common code, so memory space is properly utilized.

Disadvantages:

1. Paging suffers from internal fragmentation. Internal fragmentation especially on last page of process; average of 50% on last page.
2. Smaller frames create less fragmentation but increase number of frames and size of page table.
3. Overhead to maintain and update page table.
4. Paging hardware increases the cost of operating system.

4.5 SEGMENTATION

- As we have already seen, the user's view of memory is not same as actual physical memory in paging memory management scheme.
- The user's view is mapped onto physical memory which allows differentiation between physical and logical memory.
- Segmentation is memory management scheme that supports user's 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 etc. as shown in Fig. 4.20.
- A segment can he defined as, a logical grouping of instructions such as a subroutine, array or a data area. Every program (job) is a collection of these segments. Segmentation is a memory, management scheme which supports the programmer's view of memory.
- Thus, user prefers to view memory as collection of variable-sized segments, with no necessary ordering among segments.

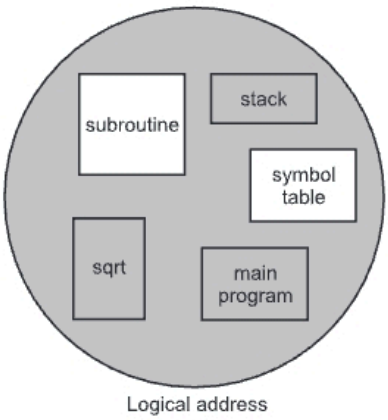


Fig. 4.20: User's View of Program

4.5.1 Basic Concept

- A logical address space is collection of segments. Each segment has a name (e.g., a unique number) and a length. The user specifies each address by two quantities: segment name and length.
- For simplicity, segments are numbered and referred by segment number rather than segment name. Thus, Logical address is a <segment number, offset>.
- Offset is to specify offset of the data from start of a particular segment. Physical address is found by looking into a segment table.

4.5.2 Segmentation Hardware

- Although user specifies two-dimensional address, but physical memory is one dimensional array. So two dimensional addresses should be mapped by one dimensional physical address. This mapping is done by Segment table.

- Each entry in segment table has a segment base and segment limit. The segment base contains the starting physical address where segment resides. Segment limit specifies the length of segment. The use of segment table is shown in Fig. 4.21.
- As logical address consists of two parts: a segment number(s) and an offset into that segment (d) the segment number is used and index to the segment table.
- The offset(d) of the logical address must be between 0 and segment limit. If it is not, we trap to the operating system. When offset is legal, it is added to segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

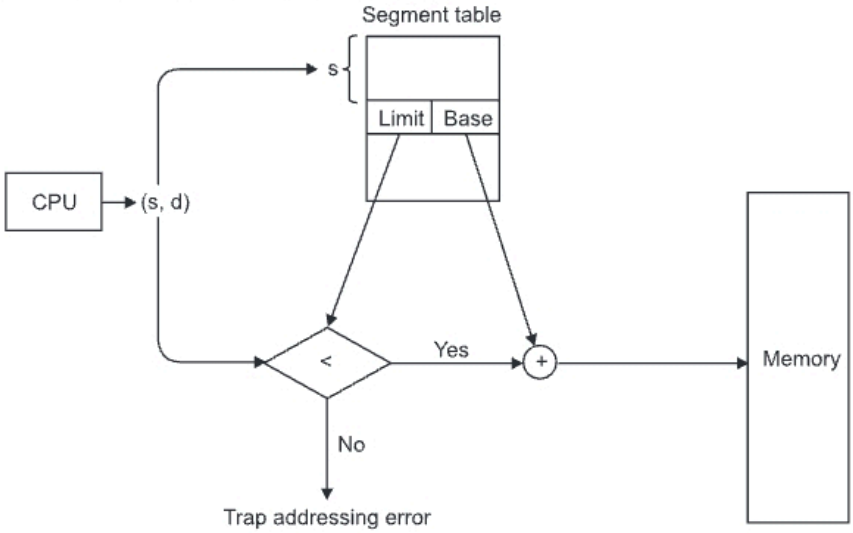


Fig. 4.21: Segmentation Hardware

Example:

- Consider an example, shown in Fig. 4.22. Segments are numbered from 0 through 4. The segments are stored in physical memory as shown. The segment table has separate entry for each segment.
- Let us see how to find physical address.
  1. Logical address is specified by a segment number(s) and an offset into that segment(d).
  2. Use segment number as an index into segment table to find base address of segment.
  3. If offset < segment limit then Physical address = Base address + offset; otherwise trap will be generated.
     - Suppose logical address is < 3,852>
     - Segment number = 3, offset = 852
     - Base Address of segment 3 = 3200
     - As offset < segment limit (i.e. 852 < 1100)
     - Physical Address= Base address + offset => 3200 + 852 => 4052

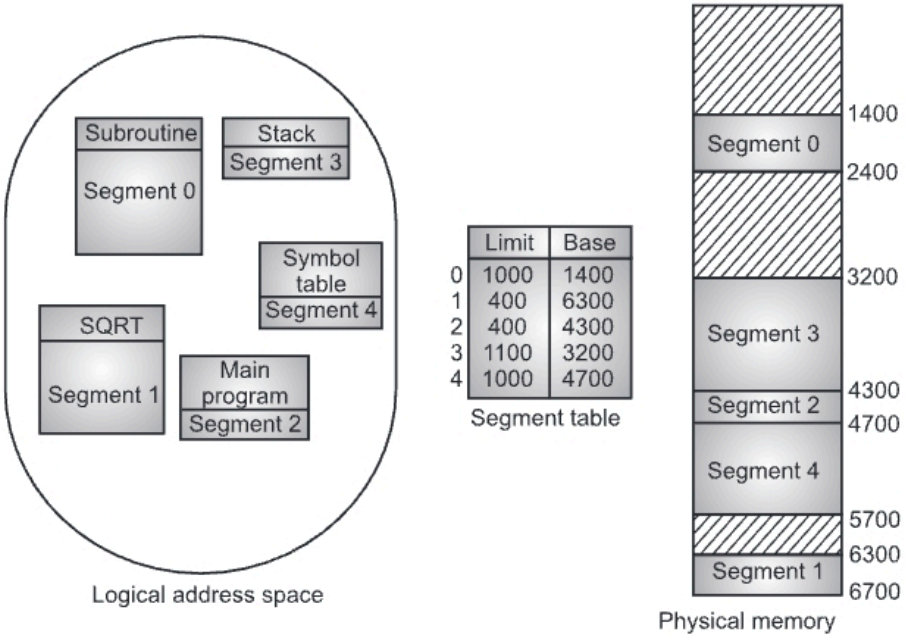


Fig. 4.22: Example of Segmentation

- Segment table can be implemented same as page table.

1. **Segment table can be stored in dedicated fast registers:**
   - As registers are very fast, access to table is also very fast.
   - This is a costly scheme.
   - As a segment number grows, number of registers may not be available, neither affordable nor feasible.
   - Therefore, scheme is useful only for small segment tables.
2. **Segment table is implemented in the Main Memory:**
   - This scheme is used when segment table is big.
   - Segment table is stored in the main memory.
   - STBR (Segment Table Base Register) points to base address of segment table.
   - In case, location of the segment table is changed from the memory, only the content of STBR is changed.

Protection of segments:

- Segments can be protected same as paging.
- Protection bits can be associated with each segment table entry.
- Protection bit can define a segment to be read-write or read only.
- Every reference to memory goes through segment table to find base address and limit.
- At same time, protection bits can be checked to verify that no writes are being made to a read only segment.
- An attempt to write to a read only segment causes hardware trap to the operating system.

Shared Segments:

- Segmentation permits sharing of code segments. When entries in the segment table of two different processes point to same physical location; segments are said to be shared.
- Shared segments are shown in Fig. 4.23.

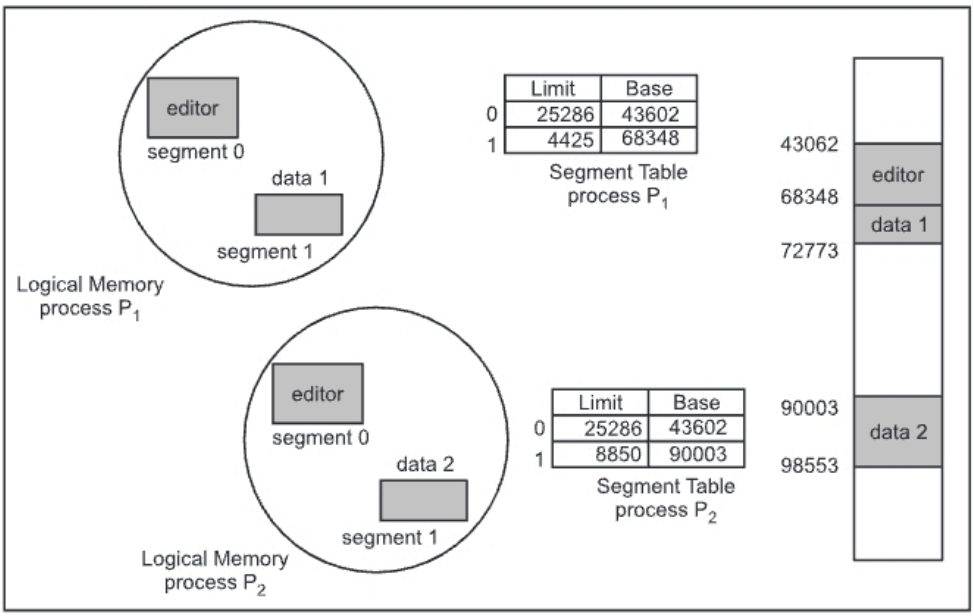


Fig. 4.23: Shared segments

- Sharing of module is easier than paging. Since segmentation is based on a logical division of memory rather than a physical one, segments of any size can be shared with only one entry in the segment tables of each user.
- With paging, there must be a common entry in the page tables for each page that is shared.

Advantages:

1. It permits physical address space of process to be non-contiguous and supports for virtual memory.
2. It supports user's view of memory and allows segments to grow.
3. It supports for dynamic relocation. Every logical address is bound to physical address by segment table.
4. In segment tables only one entry per actual segment as opposed to one per page in paging.
5. Average segment size > average page size. So less overhead in maintaing segment table.
6. We can share code or data by segmentation which is easier than paging, so memory space is properly utilized.

Disadvantages:

1. Segments suffer from external fragmentation. External fragmentation occurs due to dynamic allocation policies.
2. Overhead to maintain variable sized segments on secondary storage; problematic in case of swapping.
3. Cost of mapping by segmentation hardware is more as compared to paging.

- Let us compare contiguous and non-contiguous memory management scheme i.e., paging and segmentation on various factors:

Table 4.3: Comparison of Paging and Segmentation

Factors	Contiguous Memory Allocation	Paging	Segmentation
Memory allocation	In this, each process is contained in a single contiguous section of memory.	This scheme permits physical address space of process to be non-contiguous but make clear separation between logical memory and physical memory.	This scheme permits physical address space of process to be non-contiguous but supports user's view of memory.
Hardware support	Relocation and limit register.	Address mapping is done by page table.	Address mapping is done by segment table.
Dynamic relocation	Yes.	Paging itself is a form of dynamic relocation.	Yes, with help of Segment table.
Sharing of data or code	Not possible.	Possible. Sharing of reentrant code.	Possible, easier than paging.
Protection	By using limit register.	By Protection bits like r/w or r and valid-invalid bit attached to each entry of page table.	By Protection bits like r/w or r attached to each entry of segment table.
Fragmentation	Suffers from internal and external fragmentation.	Suffers from internal fragmentation.	Suffers from external fragmentation.
Performance	Time does not require for mapping of logical address to physical address. (Only comparing address with limit register).	If page table is implemented in associative registers, then mapping is faster.	If segment table is implemented in fast registers then mapping is faster.

4.6 VIRTUAL MEMORY MANAGEMENT

- **Virtual Memory** is a technique that allows the execution of process that is not completely in main memory.
- It abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- **Virtual address space** of a process refers to a logical view of how a process is stored in memory. A process begins at a certain logical address-say address 0 and exists in contiguous memory as shown in Fig. 4.24.
- As shown in Fig. 4.24, heap is allowed to grow upward direction in memory as it is used for dynamic memory allocation and stack is allowed to grow in downward direction through function calls.

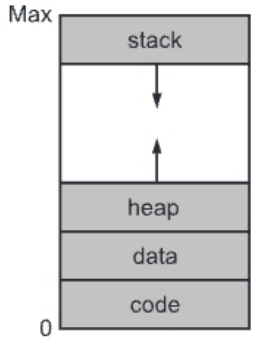


Fig. 4.24: Virtual Address Space

- The large blank space between heap and stack is virtual address space. Virtual address space includes holes that are known as **Sparse Address Spaces**.

4.6.1 Background

- Program to be executed must be in main memory. Program often have code to handle unusual error conditions. Since these errors rarely occur, so that code is almost never executed.
- Similarly, arrays lists and tables are often allocated more memory than actually required. Certain routines are not required at same time. But keeping entire program in main memory limits amount of physical memory and decreases CPU utilization and degree of multiprogramming. So there is a need to execute program that is only partially in memory (i.e. virtual memory technique).

Benefits:

- Virtual memory technique offer following benefits:
  1. A program would no longer be constrained by the amount of physical memory. So programmer can concentrated on the problem to be implemented.
  2. More programs can reside in memory at same time, so degree of multi-programming is increased. CPU utilization and throughput also increased.
  3. Less I/O would be needed to load or swap each user program.
  4. In addition to separating logical memory from physical memory, virtual memory allows files and memory to be shared.
  5. System libraries can be shared by several processes through mapping of the shared object into virtual address space.
  6. Similarly virtual memory enables processes to share memory.
  7. It provides an efficient mechanism for process creation.
- Virtual memory can be implemented via:
  1. Demand paging.
  2. Demand segmentation.

4.6.2 Demand Paging

- Demand paging policy or technique transfers memory pages instead of process to and from secondary storage. The entire process does not have to reside in main memory to execute and the kernel loads pages for a process on demand when the process references the pages.
- A demand paging is similar to paging system with swapping where processes reside in secondary memory as shown in Fig. 4.25.
- When we want to execute a process, swap from memory. Instead of swapping whole process, swapper brings only necessary pages into memory.
- A swapper is known as lazy swapper or pager who never swaps a page into memory unless that page will be needed.
- When process is to be swapped in, the pager guesses pages which will be used before process is swapped out again. Thus it decrease swap time and amount of physical memory needed.

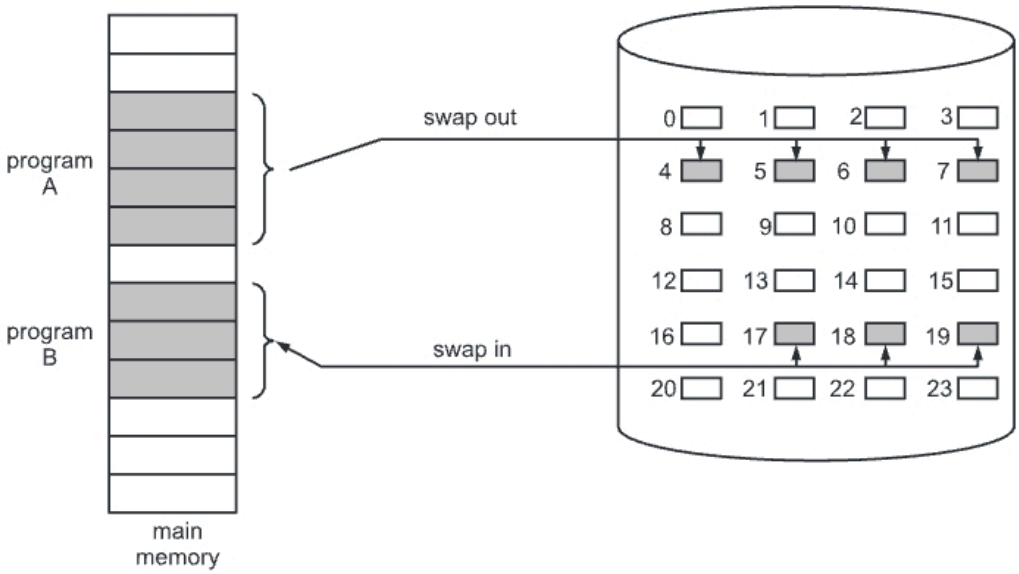


Fig. 4.25: Demand Paging

- The hardware to support demand paging is same as hardware for paging and swapping:
  1. **Page Table:** This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.
  2. **Secondary Memory:** It is usually high speed disk; also known as swap device. It holds those pages that are not in main memory.
- To distinguish between the pages that are in memory and the pages that are on the disk, valid-invalid bit is used.
- When this bit is set to "valid", the associated page is legal and in main memory. When this bit is set to "invalid", page may not be valid or page is valid but currently on the disk as shown in following Fig. 4.26.

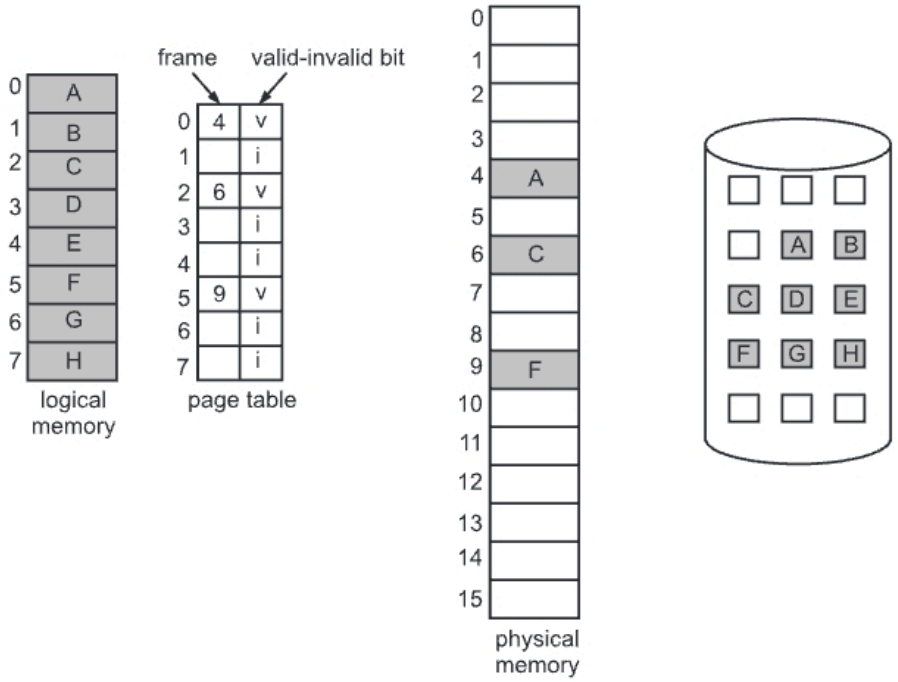


Fig. 4.26: Page Table with Valid-Invalid Bit

- **Page fault**: Whenever, process tries to access a page which is not into memory, then **page fault** occurs. The paging hardware, which translates the address through page table, will notice that invalid bit is set and cause trap to the operating system.

Procedure for handling page fault:

- The steps for handling page fault are as follows: (shown in Fig. 4.27).

  Step 1: Operating system first has to determine why page fault occurs due to missing page (valid memory reference) or out-of-range reference (invalid memory reference).

  Step 2: If invalid reference then terminate process. If valid reference, then causes a trap.

  Step 3: If valid page, but not yet brought in memory, then page is on backing store.

  Step 4: To bring in a missing page:

  - Suspend the process, saving user registers and process state.
  - Locate a free frame in memory.
  - Locate the missing page on backing store (its location on disk)
  - Bring in the missing page into memory.

  Step 5: When transfer completes, update page table by marking page to "valid".

  Step 6: Reset process's program counter in PCB so that when it starts the instruction will be re-executed. Change process's state from waiting state to ready state.

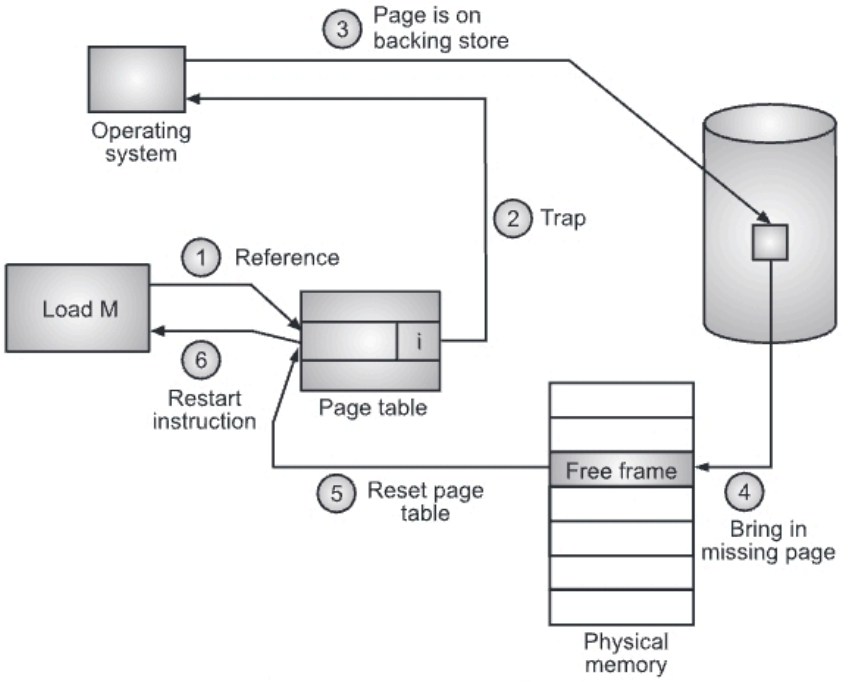


Fig. 4.27: Steps to handle a Page Fault

Pure Demand Paging:

- In demand paging, when process starts execution with no pages in memory, when operating system sets the instruction pointer to the first instruction of the process, process immediately faults for the page.
- After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. Thus, it never brings a page into memory until it is required.

4.6.3 Performance of Demand Paging

- Demand paging can significantly affect the performance of a system. For most computer systems, memory access time ranges from 10 to 200 nanoseconds.
- As long as we have no page faults, the effective access time is equal to memory access time.
- If page fault occurs, we must first read the relevant page from disk and access desired word.

  Let p be probability of page fault.
  If p = 0 then no page fault occurs.
  If p = 1, every reference is a fault.

  Effective Access Time (EAT) = (1 − p) × Memory access + p (Page Fault Overhead + Swap Page Out + Swap Page In + Restart Overhead)
- To compute the effective access time, we must know how much time is needed to service a page fault.
- Three major components affect the page fault service-time:
  1. Service the page-fault interrupts.
  2. Read in the page.
  3. Restart the process.

- Let us assume that Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds

$$\begin{aligned} EAT &= (1-p) \times 200 + p\ (8 \text{ milliseconds}) \\ &= (1-p) \times 200 + p \times 8{,}000{,}000 \\ &= 200 + p \times 7{,}999{,}800 \end{aligned}$$

- If one access out of 1,000 causes a page fault, then EAT = 8.2 microseconds. This is a slowdown by a factor of 40 because of demand paging. Thus effective access time is directly proportional to page-fault rate.
- Effective access time can be reduced by:
  1. Keeping page-fault rate low.
  2. Using better page replacement algorithm.
  3. Allocating more pages to a process.

4.6.4 Page Replacement

- In multiprogramming environment, it may happen that there is no free frame available and valid page fault occurs as shown in Fig. 4.28.
- The operating system could terminate the user process. But demand paging is an attempt to improve CPU utilization and throughput. Also users have feeling that all pages are in memory and large amount of memory available.
- To bring user page into memory, operating system swaps out some process by freeing its frames. The process of freeing frame is known as **Page replacement**.
- The frame which is selected for replacement is known as victim frame. The major issue is how to select victim frame, normally frame that is not currently being used and free it.

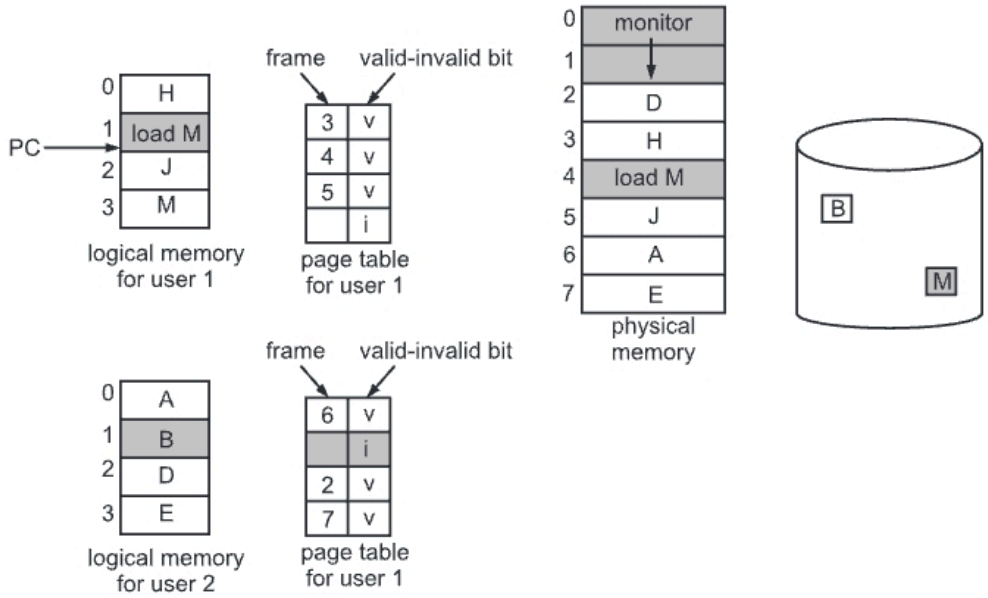


Fig. 4.28: Need for Page Replacement

- The page-fault service routine can be modified to include page replacement (shown in Fig. 4.29) as follows:
  - Find the location of the desired page on disk.
  - Find a free frame:
    - If there is a free frame, use it.
    - If there is no free frame, use a page replacement algorithm to select a victim frame.
    - Write the victim frame to the disk; change the page and frame tables accordingly.
  - Bring the desired page into the newly freed frame; update the page and frame tables.
  - Restart the process.

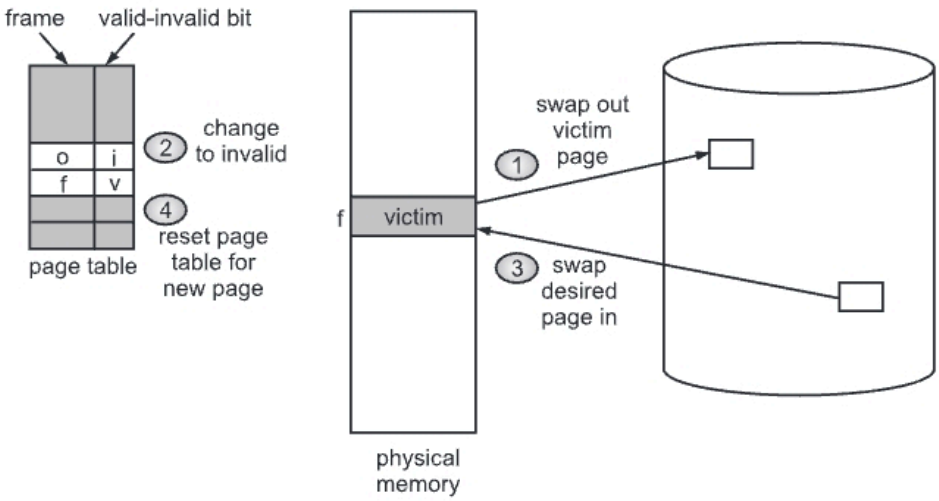


Fig. 4.29: Page Replacement

- If no free frames are free, two page transfers are required; first to free the victim frame and second to read the desired into frame. This overhead can be reduced by using **dirty bit** or modify bit.
- The dirty bit can be used as follows:
  1. Each page or frame has dirty bit associated with it in the hardware.
  2. Modify bit is set for a page, when any word has written to it after reading from secondary storage device.
  3. When we select a page for replacement, modify bit is checked.
  4. If the dirty bit is set, then page has been modified, so contents of page must be written to disk.
  5. If the dirty bit is not set, then page has not modified after reading from disk. i.e. contents of frame and copy of that page on the disk are same. So there is no need to swap out the page for purpose freeing frame. Only that particular frame is declared as free frame.

4.6.4.1 Page Replacement Algorithms [W-18, S-19]

- Every operating system uses one of the page replacement schemes from following schemes. If the page fault rate is less then page replacement algorithm will be better. The algorithm is evaluated on a string of memory references and is called as **Reference string**.
- One may generate the reference string as follows: Suppose the address sequences which are recorded are:

  0100, 0432, 0612, 0102, 0611, 0105, 0613, ... etc.

  with 100 words per page. One may divide each address by 100 to get the particular page number and the reference string will be:

  1, 4, 6, 1, 6, 1, 6, ... etc.

1. FIFO (First In First Out):

- This is one of the simplest page replacement algorithms. In this scheme, all the available frames are given to the pages from reference string serially i.e. first frame to first page, second frame to second page and so on.

	1	2	3	1	2	3	5	1	2
Page frame 0	1	1	1	4	4	4	5	5	5
Page frame 1		2	2	2	1	1	1	1	1
Page frame 2			3	3	3	2	2	2	2
Page faulted	Y	Y	Y	Y	Y	Y	Y	N	N

	3	4	5
Page frame 0	5	5	5
Page frame 1	3	3	3
Page frame 2	2	4	4
Page faulted	Y	Y	N

Total no. of page faults = 9

Fig. 4.30: FIFO Page Replacement Algorithm

- When all the available frames are over at that time, the first frame is selected as **victim frame** and at next time the second frame and so on.

For example: Consider the reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Number of frames are 3.

Explanation:

- Number of page frames given are 3, numbered page frame 0, page frame 1 and page frame 2. Reference string is scanned serially. If the page is not present in one of the three page frames, then a page fault is counted and a page is brought into the page frame.
- In the above example, first 3 pages 1, 2, 3 are brought in page frames 0, 1 and 2 respectively. When the page faulted for page 4, that time page frame containing page 1 is freed and 4 is brought in page frame 0.
- Page frame 0 was selected as victim frame because that was containing first of the first 3 pages. Next victim frame will be page frame 1 and so on.
- Total number of page faults in above example is 9. (For the pages 1, 2, 3, 4, 1, 2, 5, 3, 4).

Belady's Anomaly:

- One of the factor that affect page faults is the number of page frames available.
- Ideally, more the number of page frames, less should be the page fault rate. But practically, this is not found to be always true.

For example: We will again take the previous reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- Let the number of available page frames be 4.
- In above example, page fault rate is increased though number of page frames is increased. In the previous example, with the same reference string and 3 number of page frames, the page faults were 9.

	1	2	3	4	1	2	5	1
Page frame 0	1	1	1	1	1	1	5	5
Page frame 1		2	2	2	2	2	2	1
Page frame 2			3	3	3	3	3	3
Page frame 3				4	4	4	4	4
Page faulted	Y	Y	Y	Y	N	N	Y	Y

	2	3	4	5
Page frame 0	5	5	4	4
Page frame 1	1	1	1	5
Page frame 2	2	2	2	2
Page frame 3	4	3	3	3
Page faulted	Y	Y	Y	Y

Total no. of page faults = 10

Fig. 4.31: Example of Belady's Anomaly

- For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases. Such exception is known as Belady's Anomaly.

2. Optimal Replacement Algorithm:

- In this algorithm, the page that will not be required for longest period of time is replaced. Page fault rate in this case is less as compared to FIFO, hence the name optimal.
- With this algorithm, the page replacement decision requires complete reference string to be prepared before the execution of the algorithm starts because every page replacement needs to refer the reference string for the requirement of a particular page in future.
- Practically, this situation is difficult to achieve. Therefore, this algorithm is difficult to implement.

For example: Reference string is: 1, 3, 3, 2, 5, 4, 5, 4, 1, 4, 2, 2, 5. Number of page frames are 3.

	1	3	3	2	5	4	5	4
Page frame 0	1	1	1	1	1	1	1	1
Page frame 1		3	3	3	5	5	5	5
Page frame 2				2	2	4	4	4
Page faulted	Y	Y	N	Y	Y	Y	N	N

	1	4	2	2	5
Page frame 0	1	1	2	2	2
Page frame 1	5	5	5	5	5
Page frame 2	4	4	4	4	4
Page faulted	N	N	Y	N	N

Total no. of page faults = 6

Fig. 4.32: Optimal Page Replacement Algorithm

- If the same reference string is solved using FIFO, then 1, 3, 3, 2, 5, 4, 5, 4, 1, 4, 2, 2, 5.

	1	3	3	2	5	4	5	4	1	4	2
Page frame 0	1	1	1	1	5	5	5	5	5	5	5
Page frame 1		3	3	3	3	4	4	4	4	4	4
Page frame 2				2	2	2	2	2	1	1	1
Page faulted	Y	Y	N	Y	Y	Y	N	N	Y	N	Y

	2	5
Page frame 0	2	2
Page frame 1	4	5
Page frame 2	1	1
Page faulted	N	Y

Total no. of page faults = 8

Fig. 4.33: Optimal Page Replacement Algorithm using FIFO

Explanation (for optimal):

- First 3 pages 1, 3 and 2 are brought into frames 0, 1, and 2 respectively. To bring page number 5 in the memory, the reference string after this page is referred.
- Frame number 1 is selected as a victim frame because, it contains page number 3 which is farthest from page 5, in the future references of available pages. Similarly, while bringing page number 4, future reference after page number 4 is referred.
- The farthest page is page number 2, which is in page frame number 2. Therefore, page frame 2 becomes victim frame and 4 is brought in that frame.

3. LRU:

- LRU stands for Least Recently Used. In this algorithm, the algorithm refers to the past references of the pages. The one which is not used for longest time is selected for replacement.
- In simple words, optimal selects the page farthest in the right hand side direction from currently faulted page, and LRU selects the farthest page in the left hand side direction of currently faulted page.

	1	3	3	2	5	4	5	4	1
Page frame 0	1	1	1	1	5	5	5	5	5
Page frame 1		3	3	3	3	4	4	4	4
Page frame 2				2	2	2	2	2	1
Page faulted	Y	Y	N	Y	Y	Y	N	N	Y

	4	2	2	5
Page frame 0	5	2	2	2
Page frame 1	4	4	4	4
Page frame 2	1	1	1	5
Page faulted	N	Y	N	Y

Total no. of page faults = 8

Fig. 4.34: Least Recently Used (LRU) Algorithm

- This algorithm is easy to implement, as compared to optimal as no future knowledge of reference string is require.

  For example: Reference string: 1, 3, 3, 2, 5, 4, 5, 4, 1, 4, 2, 2, 5. Number of page frames = 3

Explanation:

- First 3 pages 1, 3 and 2 are brought into page frames 0, 1 and 2 respectively. When page fault for page 5 occurs, that time frame 0 is selected as victim frame.

- If you observe the reference string upto page 5, page number 2 is nearest to page 5 and page number 1 is farthest from page 5. The farthest page is the one which is least recently used. Therefore, page frame 0 is selected as victim frame (containing the farthest page) and 5th page is brought into it.
- To bring page number 4 in the memory, frame 1 is selected as victim frame, because it contains the page 3 farthest from page number 4 (farthest in the past) and so on.
- Implementation of true LRU needs hardware assistance. Either of the following methods can be used to implement LRU.

  (i) Counters: A time-of-use register is added to the page table. An internal logical clock counter is added to the CPU. Whenever a memory reference is made, the counter is incremented. Whenever a page reference is made, the current contents are copied to time-of-use register of respective page in the page table. The page with smallest time-of-use value is selected for the purpose of replacement. (The page with highest value of time-of-use is the one which is most recently used and the one with smallest value of time-of-use is the least recently used).

  (ii) Stack: A stack of page numbers is maintained. Whenever a reference to the page is made, it is brought onto the top of stack. At any moment, the page which is at the bottom of the stack, is the least recently used page and the frame containing this page is selected for the purpose of replacement.

  - Both the above mentioned algorithms cannot be implemented without hardware assistance.
  - Optimal and LRU both algorithms do not suffer Belady's Anomaly.

4. LRU Approximation using Reference Bit/Bits:

- Some systems do not provide hardware support for LRU and then there is restriction of using algorithms like FIFO. With the help of a reference bit a one may try to implement LRU without hardware support.
- A reference bit is attached with each page. Initially all the bits (one for each page) are cleared i.e. set to 0. After a reference to a page is made, it is turned to 1.
- From this, one could immediately find out, which pages are referenced and which are not. But then there was no way by which order of the reference to the pages can be found out. To overcome this problem, a reference byte is (8 bits) used for each page. All the 8 bits of the byte are set to 0.
- After a reference to the page is made, the least significant bit of the reference byte is set to 1. Each reference to the page set the bit from 0 to 1 from least significant position to most significant position, one bit at a time.

  For example: Page P1 has reference byte 00000000.
  After first reference to P1, the byte becomes 00000001
  After second reference to P1, the byte becomes 00000011 and so on.
- Now, it is easy to decide the order of reference of pages. Smaller the byte value, least recently the page is used.
- Only problem with this algorithm is that, at a particular instance, there may be more than one page with same bit pattern of reference byte.

5. Second Chance Algorithm:

- It is basically a FIFO algorithm. The only difference between FIFO and second chance is that, in FIFO, pages are replaced on First Come First Served Basis. In this algorithm, pages are given a chance on First Come First Served basis. Actual replacement may not be in FIFO manner.
- Reference bit of page is checked serially, if it is 0 then the page is replaced. If it is one, the bit is set to zero and current time is associated with page as its arrival time.
- Because of this, the particular page is given a second chance of replacement. This page will be replaced only after all pages are either replaced or given second chance by the system.
- In simple words, system imagine as if a list of pages is maintained. System starts with the first page in the list. If that page is having reference bit 0, then it is replaced. If not then this page is pushed to end of list and its bit is set to zero.
- System checks the second page, now either this page is replaced if reference bit is zero or pushed to the end of list if reference bit is 1 and its bit is set to 0.

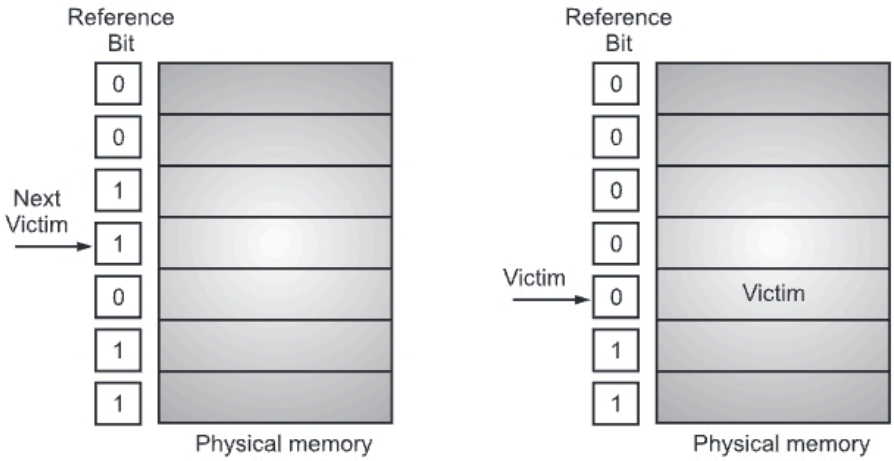


Fig. 4.35: Second Chance Algorithm

- In worst case, all the pages will be pushed to the end of the list by the system and again the first page along with reference bit 0 will appear at first position and will be selected for replacement.

6. LFU (Least Frequently Used):

- LFU page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again.
- Since, it was used heavily; it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

7. MFU (Most Frequently Used):

- The MFU page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

4.7 THRASHING

- A process that is spending more time paging than executing is said to be **Thrashing**. In other words, this process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.
- Initially when the CPU utilization is low, then the process scheduling mechanism loads multiple processes into the memory at the same time to increase the level of multiprogramming by allocating a limited amount of frames to each process.
- As the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the processes are waiting for pages. Hence, the scheduler loads more processes to increase CPU utilization, as this continues at a point of time the complete system comes to a stop.

4.7.1 Cause of Thrashing

- Thrashing occurs when a Process is spending more time in paging or Swapping activities rather than its execution. Thrashing can cause severe performance problems.
- Thrashing is caused due to under allocation of memory to a process. When a process has not been allocated as much frames as it needs to support its pages in active use, it causes a page fault.
- Thrashing phenomenon is shown in Fig. 4.36. CPU utilization is plotted against the degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached.
- If the degree of multiprogramming is increased even further, thrashing sets in and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.
- The effects of thrashing can be limited by using a local or priority replacement algorithm.

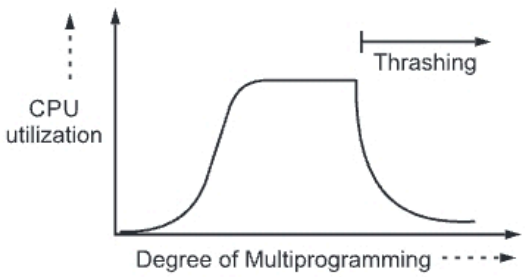


Fig. 4.36: Thrashing Phenomenon

- With local replacement, if one process starts thrashing, it cannot steal frames from another process and causes it to thrash also.
- In the next section, we will see how to eliminate Thrashing.

4.7.2 Working-Set Model

- A working-set model is a technique which is used to reduce the causes of thrashing in OS.
- Working-set model assumes that programs execute in localities and working-set is the set of pages in the current locality. A locality is a set of pages which are actively used together.
- Accordingly, each process should be allocated enough frames for its working-set. However, no fully accurate arrangement to find out the working set is available. So, it is a clumsy way of controlling thrashing.
- The working-set model uses a parameter W to define the working-set window. It examines the most recent page references in the window size W.
- The set of pages in the most recent W page references is the working-set as shown in Fig. 4.37.
- If a page is in active use, it will be in the working-set. If it is no longer being used, it will be dropped from the working-set W time units after its last reference. Thus, the working-set is an estimation of the program's locality of pages.

Example:

- Consider the sequence of memory references shown in Fig. 4.37. If W=12 memory references, then the working set at time t_1 is {1, 2, 3, 4, 5}. By time t_2, the working set has changed to {1, 3, 6, 8}.

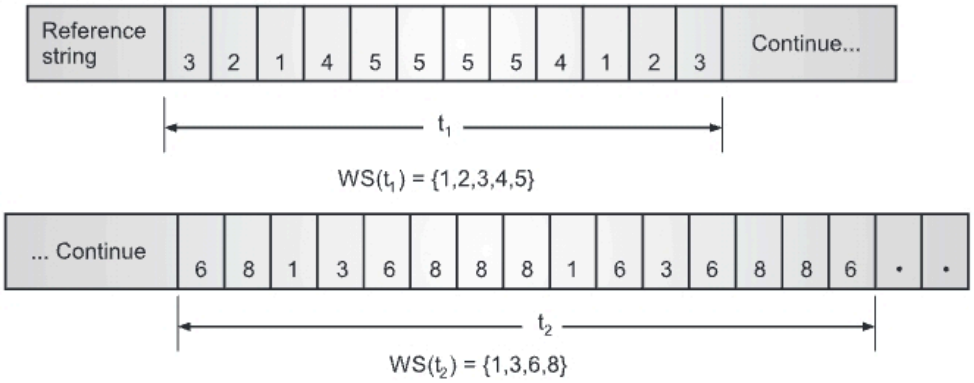


Fig. 4.37: Working-Set Model

- The correctness of the working-set depends upon the selection of W. If W is too small, it will not include the entire working-set; if W is too large, it may overlap several localities. In the extreme, if W is infinite, the working-set is the entire program.
- The most important property of the working set is its size. If the Working Window Set size, (WWS_i), for each process in the system, is considered as,

  $$D = WWS_i$$

  Where, D is the demand for frames. Each process is actively using the pages in its working set. Thus process i needs WWS_i frames. If the total demand is greater than the total number of available frames, thrashing will occur, since some processes will not have enough frames.

- The use of the working-set model is then quite simple and easy. The operating system monitors the working set of each process and allocates to it enough frames to provide it with its working set size, if there are enough extra frames available to provide it with its working set size, another process can be initiated.
- If the sum of the working set size increases, and exceed the total number of frames available, the operating system selects a process to suspend. Its pages are then written out and its frames are reallocated to other processes. The suspended process can be started again later.
- This working set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible. Thus, it tries to optimize CPU utilization.
- The difficulty with the working set model is keeping track of the working set.

Solved Problems

Problem 1: Consider the following page reference string,

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3

How many page faults will occur for following page replacement algorithms? Assuming 3 frames, all are initially empty.

1. FIFO
2. LRU

Solution: No. of frames = 3.

1. FIFO Method:

Reference String	1	2	3	4	2	1	5	6	2	1	2	3
Frame 1	1	1	1	4	4	4	4	6	6	6	6	3
Frame 2		2	2	2	2	1	1	1	2	2	2	2
Frame 3			3	3	3	3	5	5	5	1	1	1
Page Fault	1	2	3	4	X	1	5	6	2	1	X	3
	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	Y

Total No. of page faults by FIFO = 10.

2. LRU Method:

Reference String	1	2	3	4	2	1	5	6	2	1	2	3
Frame 1	1	1	1	4	4	4	5	5	5	1	1	1
Frame 2		2	2	2	2	2	2	6	6	6	6	3
Frame 3			3	3	3	1	1	1	2	2	2	2
Page Fault	1	2	3	4	X	1	5	6	2	1	X	3
	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N	Y

Total No. of page faults by LRU = 10.

Problem 2: Consider the following page reference string:

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7

How many page faults would occur for the following page replacement algorithms, assuming four frames? All frames are initially empty.

1. LRU
2. Optimal replacement

Solution:

1. LRU Page Replacement Method:

Reference String	1	2	3	4	5	3	4	1	6	7	8	7
Frame 1	1	1	1	1	5	5	5	5	6	6	6	6
Frame 2		2	2	2	2	2	2	1	1	1	1	1
Frame 3			3	3	3	3	3	3	3	7	7	7
Frame 4				4	4	4	4	4	4	4	8	8
Page Fault	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N

Total page faults by LRU = 9.

2. Optimal Page Replacement Method:

Reference String	1	2	3	4	5	3	4	1	6	7	8	7
Frame 1	1	1	1	1	1	1	1	1	6	6	6	6
Frame 2		2	2	2	5	5	5	5	5	7	7	7
Frame 3			3	3	3	3	3	3	3	3	8	8
Frame 4				4	4	4	4	4	4	4	4	4
Page Fault	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	N

Total page faults by Optimal = 8.

Problem 3: Consider the following segment table:

Segment	Base	Length
0	600	120
1	1200	350
2	75	85
3	1760	90
4	2510	680

What are the physical addresses for the following logical addresses?

1. 0, 125
2. 1, 310
3. 3, 88
4. 2, 77
5. 4, 444

Solution:

1. Logical address = 0, 255
   Segment No. = 0
   Offset = 255
   Base address of segment 0 = 600
   Since offset > Length (i.e. 255 > 120)
   Therefore invalid address and trap will be generated.
2. Logical address = 1, 310
   Segment No. = 1
   Offset = 310
   Base address of segment 1 = 1200
   Physical address = Base address + Offset
   = 1200 + 310
   = 1510
3. Logical address = 3, 88
   Segment No. = 3
   Offset = 88
   Since Offset < length, therefore address is valid
   Base Address of segment 3 = 1760
   Physical address = Base address + Offset
   = 1760 + 88
   = 1848
4. Logical address = 2, 77
   Segment No. = 2
   Offset = 77
   Base address = 75
   Physical address = 75 + 77
   = 152
5. Logical address = 4,444
   Segment No. = 4
   Offset = 444
   Base address = 2510
   Physical address = 2510 + 444
   = 2954

Problem 4: Consider a logical address space of four pages of 512 words each, mapped onto a physical memory of 16 frames.

1. How many bits are in virtual address?
2. How many bits are in physical address?

Solution:

$$\text{Logical address space} = 4 \text{ pages} = 2^2 \ (m = 2)$$

$$\text{Page size} = 512 = 2^9 \ (n = 9)$$

$$n = \text{offset} = 9 \text{ bits}$$

$\therefore$ Logical address or Virtual address= 9 + 2 = 11 bits

$\therefore$ 11 bits are in virtual address.

$$\text{Physical memory} = 16 \text{ frames} = 2^4$$

$$\text{Frame size} = 2^9 = 519 \text{ words}$$

$$\therefore \quad \text{Physical address bits} = 2^9 \times 2^4$$

$$= 2^{13}$$

$\therefore$ 13 bits are in physical address.

Problem 5: Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3

How many page faults would occur for the following page replacement algorithms assuming four frames? All frames are initially empty.

1. LRU
2. Optimal.

Solution:

1. LRU Replacement Method:

Reference String	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
Frame 1	1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6
Frame 2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Frame 3			3	3	3	3	5	5	5	5	5	3	3	3	3	3	3	3	3
Frame 4				4	4	4	4	6	6	6	6	6	7	7	7	7	1	7	7
Page Fault	Y	Y	Y	Y	N	N	Y	Y	N	N	N	Y	Y	Y	N	N	Y	N	N

Total page faults = 10.

2. Optimal Replacement Method:

Reference String	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3
Frame 1	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7	1	1	1
Frame 2		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Frame 3			3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Frame 4				4	4	4	5	6	6	6	6	6	6	6	6	6	6	6	6
Page Fault	Y	Y	Y	Y	N	N	Y	Y	N	N	N	N	Y	N	N	N	Y	N	N

Total page faults = 8.

Problem 6: Consider the following job queue:

Job	Memory	Time
1	80 k	9
2	110 k	4
3	20 k	18
4	60 k	5
5	40 k	10

Show the memory map of various stages by using MVT scheduling.

Assumption: Total memory is of 400 k and monitor of look and all jobs arrived at same time (0 ms).

Solution:

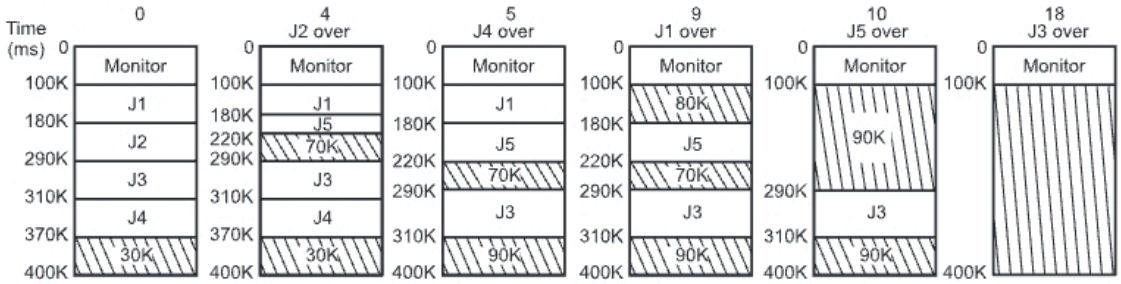


Fig. 4.38

Problem 7: Consider the following segment table:

Segment	Base	Length
0	363	500
1	1272	20
2	1675	1500
3	986	240
4	211	130

What are physical addresses for the following logical addresses?

1. 0, 425
2. 2, 500
3. 1, 150
4. 3, 285
5. 4, 125

Solution:

1. 0, 425

   Here offset is 425 and segment No. = 0.

   $\because$ Offset < Length

   $\therefore$ Address is valid

   Physical address = Base + Offset

   = 363 + 425

   = 788

2. 2, 500

Segment No. = 2
Offset = 500
∵ Offset < Length
∴ Address is valid
Physical address = Base + Offset
= 1675 + 500
= 2175

3. 1, 150

Segment No. = 1
Offset = 150
∵ Offset > Length, it will generate trap an addressing error

4. 3, 285

Segment No. = 3
Offset = 285
∵ Offset > Length, it will generate trap an addressing error

5. 4, 125

Segment No. = 4
Offset = 125
∵ Offset < Length, address is valid
Physical address = Base + Offset
= 211 + 125
= 336

Problem 8: Consider the physical memory of 15 frames with each frame of size 8 bytes. Consider page table entries as:

0	2
1	7
2	1
3	13
4	10
5	5

Map following logical addresses to their physical addresses:
(1) 5 (2) 40 (3) 23 (4) 36 (5) 68.

Solution: Page size = 8 bytes

1. Logical address = 5
Page No. = 5/8 ⇒ 0
Page offset = 5 % 8 = 5
Base address = Frame number × Frame size
= 2 × 8
= 16
Physical address = Base + Offset address
= 16 + 5
Physical address = 21

2. Logical address = 40
Page No. = 40/8 = 5
Page offset = 40 % 8 = 0
Frame No = 5
Base address = $5 \times 8 = 40$
Physical address = 40 + 0 = 40

3. Logical address = 23
Page No. = $23/8 \Rightarrow 2$
Page offset = $23 \% 8 \Rightarrow 7$
Frame No. = 1
Base address = $1 \times 8 = 8$
Physical address = $8 + 7 \Rightarrow 15$

4. Logical address = 36
Page No. = 36/8 = 4
Page offset = 36 % 8 = 4
Frame No. = 10
Base address = $10 \times 4 = 40$
Physical address = $40 + 4 \Rightarrow 44$

5. Logical address = 68
Page No. = 68/8 = 8
Page offset = 4

We can't find frame number since page 8 is not loaded in page table, so we can not find physical address.

Problem 9: Find the number of page faults for following string according to:

1. LRU
2. Optimal page replacement.

Consider reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. Assume 3 page frames.

Solution: 1. LRU Replacement Method:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	6	0	0	1	1	1	1	1	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Page Fault	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	N	Y	N	Y	N	N

Total page faults = 12

2. Optimal Replacement Method:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
Page Fault	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N	N	N	Y	N	N

Total page faults = 9.

Problem 10: Consider the following page reference string:

8, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2

How many pages faults would occur for the following page replacement algorithms, assuming three frames?

All frames are initially empty.

1. Optimal page replacement.
2. LRU replacement.

Solution:

1. Optimal replacement algorithm:

Reference string	8	0	1	2	0	3	0	4	2	3	0	3	2	1	2
Frame 1	8	8	8	2	2	2	2	2	2	2	2	2	2	2	2
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	1	1
Frame 3			1	1	1	3	3	3	3	3	3	3	3	3	3
Page Fault	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N

Total page faults = 8.

2. LRU replacement:

Reference string	8	0	1	2	0	3	0	4	2	3	0	3	2	1	2
Frame 1	8	8	8	2	2	2	2	4	4	4	0	0	0	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2
Page Fault	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	N

Total page faults = 10.

Problem 11: For the following page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 0, 1. How many page faults occur for the following page replacement algorithms, assuming three frames? All frames are initially empty.

1. LRU
2. Optimal page replacement.

Solution:

1. LRU:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	7	7	7
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	2	1
Page Fault	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	N	Y	Y	N	Y

Total page faults = 13.

2. Optimal Page Replacement:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1
Page Fault	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N

Total page faults = 9.

Problem 12: Consider the following page reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

How many page faults would occur for the following page replacement algorithms?

1. LRU
2. FIFO.

Assume three frames.

Solution:

1. LRU algorithm:

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Frame 1	1	1	1	4	4	4	5	5	5	3	3	3
Frame 2		2	2	2	1	1	1	1	1	1	4	4
Frame 3			3	3	3	2	2	2	2	2	2	5
Page Fault	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y

Total pages faults by LRU = 10.

2. FIFO algorithm:

Reference string	1	2	3	4	1	2	5	1	2	3	4	5
Frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Frame 2		2	2	2	1	1	1	1	1	3	3	3
Frame 3			3	3	3	2	2	2	2	2	4	4
Page Fault	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N

Total pages faults by FIFO = 9.

Problem 13: Consider the following page reference string:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2.

How many page faults would occur for the following page replacement algorithm?

1. LRU (Least Recently Used)
2. Optimal.

Assume three frames.

Solution:

1. LRU algorithm:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2
Page Fault	Y	Y	Y	Y	N	Y	N	Y	Y	Y	Y	N	N	Y	N

Total page faults = 10.

2. Optimal algorithm:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2
Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	1	1
Frame 3			1	1	1	3	3	3	3	3	3	3	3	3	3
Page Faults	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N

Total page faults by optimal = 8.

Problem 14: Consider a logical address space of 4 pages of 512 words each, mapped onto a physical memory of 16 frames.

1. How many bits are there in logical address?
2. How many bits are there in physical address?

Solution: Logical address space = 4 pages = 2^2 (m = 2)

Page size = 512 = 2^9 (n = 9)

n = offset = 9 bits

∴ Logical address = 9 + 2 = 11 bits

Physical memory = 16 Frames = 2^4

Frame size = 2^9 = 512 words

Physical address bits = $2^9 \times 2^4$

= 2^{13}

∴ Physical address = **13 bits**

Problem 15: Consider page reference string as follows:

7, 5, 6, 2, 9, 5, 7, 6, 2, 7, 6, 5, 2, 7, 2, 7, 8

Assume 3 frames. Find the number of page faults according to:

1. Optimal page replacement algorithm.
2. Least Recently Used (LRU) page replacement algorithm.

Solution:

1. Optimal page replacement algorithm:

Reference string	7	5	6	2	9	5	7	6	2	7	6	5	2	7	2	7	8
Frame 1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	8
Frame 2		5	5	5	5	5	5	5	2	2	2	2	2	2	2	2	2
Frame 3			6	2	9	9	9	6	6	6	6	5	5	5	5	5	5
Page Fault	Y	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	N	N	N	N	Y

Total pages faults = 9.

2. LRU algorithm:

Reference string	7	5	6	2	9	5	7	6	2	7	6	5	2	7	2	7	8
Frame 1	7	7	7	2	2	2	7	7	7	7	7	7	2	2	2	2	2
Frame 2		5	5	5	9	9	9	6	6	6	6	6	6	7	7	7	7
Frame 3			6	6	6	5	5	5	2	2	2	5	5	5	5	5	8
Page Fault	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y	N	N	Y

Total pages faults = 13.

Problem 16: Consider the following segment table:

Segment	Base	Limit
0	750	420
1	1780	535
2	3130	81
3	7070	70
4	6166	320

Map the following logical address to physical addresses. Consider the first leftmost digit as segment number.

1. 4666
2. 280
3. 0251
4. 1025
5. 3003

Solution:

1. **4666:**

   Segment No = 4, Offset = 666, Length of segment = 320

   Since, Offset > Length, it will generate trap addressing an error.

2. **280:**

   Segment No = 2, Offset = 80, Length of segment 2 = 81

   ∵ Offset < Length

   ∴ Address is valid

   ∴ Physical address = Base address + Offset
   = 3130 + 80
   = 3210

3. **0251:**

   Segment No = 0, Offset = 251, Length of segment 0 = 420

   ∵ Offset < Length

   ∴ Address is valid

   ∴ Physical address = Base address + Offset
   = 750 + 251
   = 1001

4. **1025:**

   Segment No = 1, Offset = 025, Length = 535

   ∵ Offset < Length

   ∴ Address is valid

   ∴ Physical address = Base address + Offset
   = 1780 + 025
   = 1805

5. **3003:**

   Segment No = 3, Offset = 003, Length = 70

   ∵ Offset < Length

   ∴ Address is valid

   ∴ Physical address = Base address + Offset
   = 7070 + 003
   = 7073

Problem 17: Consider the following page reference string:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 7, 0, 1.

How many page faults would occur for the following page replacement algorithm?

1. FIFO page replacement
2. Optimal page replacement.

Assume three frames.

Solution:

1. FIFO page replacement:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	7	7	7
Frame 2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	0	0
Frame 3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	1
Page Fault	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	N	N	Y	Y	N	Y	Y	Y

Total page faults = 15.

2. Optimal replacement:

Reference string	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	7	0	1
Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1
Page Fault	Y	Y	Y	Y	N	Y	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N

Total page faults = 9.

Problem 18: A system uses 3 page frames for storing process pages in main memory. It uses the First in First out (FIFO) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below?

4 , 7, 6, 1, 7, 6, 1, 2, 7, 2

Also calculate the hit ratio and miss ratio.

Solution: Total number of references = 10.

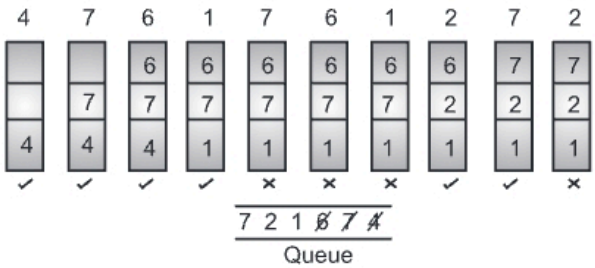


Fig. 4.39: Hit and Miss ratio

Here, Total number of page faults occurred = 6

Calculating Hit ratio:

Total number of page hits

= Total number of references – Total number of page misses or page faults

= 10 – 6

= 4

Thus, Hit ratio

$$= \frac{\text{Total number of page hits}}{\text{Total number of references}}$$

$$= \frac{4}{10}$$

= 0.4 or 40%

Calculating Miss ratio:

Total number of page misses or page faults = 6

Thus, Miss ratio

$$= \frac{\text{Total number of page misses}}{\text{Total number of references}}$$

$$= \frac{6}{10}$$

= 0.6 or 60%

Alternatively,

Miss ratio

= 1 – Hit ratio

= 1 – 0.4

= 0.6 or 60%

Summary

- Memory management algorithms for multiprogramming operating systems range from the simple single-user system approach to segmentation and paging.
- The most important determinant of the method used in a particular system is the hardware provided.
- We must ensure correct operation has to protect operating system from user access and user process from one another. This protection must be provided by the hardware. Two registers, base and limit register can be used to provide protection.
- The base register holds the starting base address or smallest legal physical memory address and limit register specifies size of the range.
- So to create active process in memory, symbolic addresses generated by program must be mapped into the process address space. This mapping is known as address binding.
- Address binding can be done at compile time, load time and execution time.
- An address generated by the CPU is commonly known as Logical Address or virtual address.

- The address seen by the memory unit that is one loaded into the memory- address-register of the memory is commonly referred as the Physical Address.
- The set of all logical addresses generated by a program is known as Logical Address Space.
- The set of all physical addresses corresponding to these logical addresses is Physical Address Space.
- Now, the run time mapping from virtual address to physical address is done by a hardware device known as Memory Management Unit (MMU).
- Simplest memory mapping is done using relocation register.
- To obtain better memory utilization, routines can be loaded in main memory as required. This is known as dynamic loading.
- In dynamic linking, linking is postponed until execution time.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. This process is known as swapping.
- Swapping requires backing store.
- Main memory usually divided into two partitions as 1) Resident operating system, usually held in low memory with interrupt vector 2) User processes then held in high memory.
- Memory can be partitioned by two approaches: Single partition allocation and Multiple partition allocation
- Single partition allocation is the simplest method of memory management. The memory is divided in two sections: one section for operating system and second section for user program.
- In multiple partition allocation, to support multiprogramming, main memory is divided into several partitions; each of which is allocated to a single process. The number of programs residing in memory will be bound by the number of partitions.
- In Multiple contiguous fixed (static) partition allocation (MFT) method, memory is divided into several partitions of fixed size that size never changes. Each partition method holds one process.
- In Multiple contiguous variable (dynamic) partition allocation (MVT) method, region or partition size is not fixed; it can vary dynamically. It creates partition according to requirements of process.
- One of reason of fragmentation is improper selection of region size.
- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used is known as internal fragmentation.
- External Fragmentation – as processes are loaded and removed, the free memory is broken into little pieces. Total memory space exists to satisfy a request, but it is not contiguous then it is known as external fragmentation.
- Compaction is process of collecting all free holes and forms a single big free hole. It shuffles memory contents to place all free memory together in one large block. Compaction is possible only with dynamic relocation.

- Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous. It avoids external fragmentation. The basic method for implementation involves breaking physical memory into fixed-sized blocks called frames and break logical memory into blocks of the same size called Pages.
- Every access to any of page has to go through the page table. Therefore, implementation of the page table directly affects the performance of this kind of memory management.
- The percentage of times that a particular page number is found in the TLB is called hit ratio.
- Memory protection in a paged environment is supported by protection bits associated with each frame.
- Reentrant Code or pure code is non-self modifying code; it never changes during execution.
- Paging suffers from internal fragmentation.
- Segmentation is memory management scheme that supports user's 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 etc.
- Each segment has a name (e.g., a unique number) and a length. The user specifies each address by two quantities: segment name and length. For simplicity, segments are numbered and referred by segment number rather than segment name.
- Two dimensional addresses should be mapped by one dimensional physical address. This mapping is done by Segment table. Each entry in segment table has a segment base and segment limit.
- Segments suffer from external fragmentation. External fragmentation occurs due to dynamic allocation policies.
- Virtual memory is a technique that allows the execution of process that is not completely in main memory.
- Virtual memory can be implemented via: Demand paging and Demand segmentation.
- Demand paging policy or technique transfers memory pages instead of process to and from secondary storage. The entire process does not have to reside in main memory to execute and the kernel loads pages for a process on demand when the process references the pages.
- Whenever process tries to access a page which is not into memory, then page fault occurs.
- To bring user page into memory, operating system swaps out some process by freeing its frames. The process of freeing frame is known as page replacement. The frame which is selected for replacement is known as victim frame.
- Page replacement algorithms are FIFO, Optimal, LRU and Second Chance.
- In FIFO scheme, all the available frames are given to the pages from reference string serially i.e. first frame to first page, second frame to second page and so on.

- In Optimal algorithm, the page that will not be required for longest period of time is replaced.
- In LRU algorithm, the algorithm refers to the past references of the pages. The one which is not used for longest time is selected for replacement.
- Second Chance is basically a FIFO algorithm. In this algorithm, pages are given a chance on First Come First Served basis. Actual replacement may not be in FIFO manner.
- Thrashing is high paging activity. A process is thrashing if it is spending more time paging than executing.

Check Your Understanding

1. Which one of the following is the address generated by CPU?
   (a) Absolute address (b) Logical address
   (c) Physical address (d) All of the Above
2. Run time mapping from virtual to physical address is done by _____.
   (a) CPU (b) File management
   (c) Memory management (d) None of the Above
3. A _____ is the address of an instruction or data as used by a program.
   (a) Logical address (b) Port address
   (c) Physical address (d) All of the Above
4. A _____ address is the effective memory address of an instruction or data that is obtained after the address binding has been done.
   (a) Logical (b) Physical
   (c) Port (d) None of the Above
5. The set of physical addresses in the main memory which a process actually occupies constitutes the _____ of the process.
   (a) Logical address space (b) Disk address space
   (c) Physical address space (d) None of the Above
6. The process of mapping logical addresses to real physical addresses in the memory is called as _____.
   (a) Address binding (b) Process linking
   (c) Address loading (d) All of the Above
7. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called as _____.
   (a) Fragmentation (b) Paging
   (c) Mapping (d) None of the Above

8. Operating System maintains the page table for _____.
   (a) each thread (b) each address
   (c) each process (d) All of the Above
9. Compaction is _____.
   (a) a technique for overcoming internal fragmentation
   (b) a paging technique
   (c) a technique for overcoming external fragmentation
   (d) a technique for overcoming fatal error
10. In contiguous memory allocation _____.
   (a) each process is contained in a single contiguous section of memory
   (b) all processes are contained in a single contiguous section of memory
   (c) the memory space is contiguous
   (d) None of the Above
11. The relocation register helps in _____.
   (a) providing more address space to processes
   (b) a different address space to processes
   (c) to protect the address spaces of processes
   (d) None of the Above
12. In fixed sized partition, the degree of multi programming is bounded by _____.
   (a) the number of partitions (b) the CPU utilization
   (c) the memory size (d) All of the Above
13. A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution this process is known as _____.
   (a) Linking (b) Loading
   (c) Swapping (d) All of the Above
14. Dynamic loading is _____.
   (a) loading multiple routines dynamically
   (b) loading a routine only when it is called
   (c) loading multiple routines randomly
   (d) None of the Above
15. The idea of overlays is to _____.
   (a) data that are needed at any given time
   (b) enable a process to be larger than the amount of memory allocated to it
   (c) keep in memory only those instructions
   (d) All of the Above

16. A page fault occurs when _____.
    (a) a page gives inconsistent data
    (b) a page cannot be accesses due to its absence from memory
    (c) a page is invisible
    (d) All of the Above
17. A process is thrashing if _____.
    (a) it spends a lot of time executing, rather than paging
    (b) it spends a lot of time paging, than executing
    (c) it has no memory allocated to it
    (d) None of the Above
18. A locality is _____.
    (a) a set of pages that are actively used together
    (b) a space in memory
    (c) an area near a set of processes
    (d) None of the Above
19. Logical memory is broken into blocks of the same size called as _____.
    (a) frames (b) pages
    (c) backing store (d) All of the Above
20. Every address generated by the CPU is divided into two parts _____.
    (a) frame bit & page number (b) page offset & frame bit
    (c) page number & page offset (d) frame offset & page offset
21. Memory protection in a paged environment is accomplished by _____.
    (a) protection algorithm with each page
    (b) restricted access rights to users
    (c) restriction on page visibility
    (d) protection bit with each page
22. _____ is the concept in which a process is copied into main memory from the secondary memory according to the requirement.
    (a) Paging (b) Demand paging
    (c) Segmentation (d) Swapping
23. Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used is known as _____ fragmentation.
    (a) Internal (b) External
    (c) Both (a) & (b) (d) None of the Above

24. In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases?
    (a) LRU (Least Recently Used) (b) OPT (Optimal Page Replacement)
    (c) MRU (Most Recently Used) (d) FIFO (First In First Out)
25. Which of the following page replacement algorithms suffers from Belady's anomaly?
    (a) FIFO (b) LRU
    (c) Optimal Page Replacement (d) Both LRU and FIFO

Answers

1. (b)	2. (c)	3. (a)	4. (b)	5. (c)	6. (a)	4. (b)	8. (c)	9. (c)	10. (a)
11. (c)	12. (a)	13. (c)	14. (b)	15. (d)	16. (b)	14. (b)	18. (a)	19. (b)	20. (c)
21. (d)	22. (b)	23. (a)	24. (d)	25. (a)					

Practice Questions

Q.I Answer the following questions in short.

1. What is memory management?
2. What is logical and physical address?
3. Explain the term logical vs physical address space.
4. What address binding?
5. What is swapping?
6. What is segmentation?
7. What is fragmentation?
8. What is meant by memory mapping?

Q.II Answer the following questions.

1. What is overlay? Explain with diagrammatically.
2. What is page fault? Explain the different steps in handling a page fault.
3. With the help of example explain following page replacement algorithms:
   (i) FIFO(ii) LRU (iii) LFU (iv) Optimal Page Replacement
4. With the help of diagram demand paging.
5. Describe paging diagrammatically.
6. What is thrashing? Explain causes of thrashing.
7. Write short note on: (i) Working set model (ii) Virtual memory (iii) Dynamic linking and Shared libraries (iv) Segmentation hardware.

Q.III Define the terms:

1. Demand Paging
2. Swapping
3. Page fault
4. Thrashing
5. Frames
6. Page table

Previous Exam Questions

Winter 2018

1. Differentiate between external and internal fragmentation. **[3 M]**

Ans. Refer to Section 4.3.3.

2. Define the term : Page fault **[1 M]**

Ans. Refer to Section 4.6.2.

3. Write a short note on swapping. **[3 M]**

Ans. Refer to Section 4.2.

4. Consider the following page reference string: **[4 M]**

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3

How many page faults will occur for LRU algorithm? Assume that number of frames is 3.

Ans. Refer to Section 4.6.4.1.

5. Explain overlays. **[3 M]**

Ans. Refer to Section 4.1.6.

6. Consider the following segment table: **[5 M]**

Segment	Base	Length
0	363	500
1	1272	20
2	1675	1500
3	986	240
4	211	130

Ans. Refer to 'Solved Problems' section.

7. What are the physical addresses for the following logical addresses? **[5 M]**

Segment	Offset
0	425
1	500
2	150
3	285
4	125

Ans. Refer to 'Solved Problems' section.

Summer 2019

1. A logical address is an address generated by _____. **[1 M]**

(a) Memory unit (b) Central processing unit

(c) Graphics processing unit (d) None of above

Ans. (b).

2. Explain the working of memory management unit in short. **[3 M]**

Ans. Refer to Section 4.1.

3. State the advantages and disadvantages of dynamic linking. **[4 M]**

Ans. Refer to Section 4.1.5.

4. What is compaction? **[1 M]**

Ans. Refer to Section 4.3.3.1.

5. Let the reference string be 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
How many page faults will occur for the following page replacement algorithms, assuming 4 frames for all initially empty:
(i) Optimal replacement (ii) LRU. **[5 M]**

Ans. Refer to Section 4.6.4.1.

6. Let the reference string be 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5. Use FIFO algorithm to find the number of page faults: **[4 M]**

(a) When there are 3 frames

(b) Where there are 4 frames.

Ans. Refer to Section 4.6.4.1.

■■■

5... File System

Learning Objectives...

- To understand concepts of File, File Attributes, Operations on Files and different types of Files.
- To study different File Access methods.
- To learn Directory Structure and its different types.
- To know File System structure.
- To explain various File Allocation methods.
- To explore different methods for Free Space Management.

5.1 INTRODUCTION

- The non-volatility of the memory enables the disks to store information indefinitely. This information can also be made available online all the time.
- Users think of all such information as files. It provides the mechanism for on-line storage of an access to both data and programs of operating system and all users of the computer system. OS provides support for such management through a file system.
- File system is the software which allows users and applications to organize and manage their files. So for most users, the file system is most visible portion of an operating system.
- The file system consist of two distinct parts namely a collection of files, each storing related data, and a directory structure, which organizes and provides information about all the files in the system.
- File management is one of the important functions of an operating system. The file system facilitates storage, access and retrieval of both data and programs of all the operating system and of the users.
- File management system is the system software that is concerned with file services such as accessing a file, directory maintenance, access control and others.
- A file is a collection of related records. A file system consists of logical structures and software routines that control access to files. It provides users with the functionality to create, modify, delete, and access files.

Drag from top and touch the back button to exit full screen.

- A file system organizes data in a structured hierarchical manner via the use of directories, which are containers for storing pointers to multiple files.

5.2 FILE CONCEPT

- Computer can store information on various storage media, such as magnetic disk, magnetic tapes, and optical disk.
- An operating system provides a uniform logical view of information storage. It abstracts physical properties of its storage devices to define a logical storage unit, the file.
- A file is a named collection of related information that is recorded on secondary storage. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files may be free form, such as text files, or may be formatted rigidly. In general, a file is a sequence of bits, bytes, lines or records, the meaning of which defined by the file's creator and user. Files are mapped by the operating system onto physical devices.
- To be able to use some commands on a file, operating system needs to know the structure of each file. For example, while printing a file; operating system prints garbage for a binary object file. It can be prevented if operating system has been told that the file is a binary object file.
- There are two major disadvantages in letting the operating system know the structure of the files:
  1. There are various types of the files and for each type, operating system requires separate program to understand that file, so size of operating system increases. It must have code to support all different types of files.
  2. Operating system will not be able to manage any new type of the file, until and unless code for the particular type is not added to the operating system.

5.2.1 File Attributes

- A file is named for the convenience of its human users, and is referred to by its name. A file's attributes vary from one operating system to another but typically consist of these:
  1. **Name:** The symbolic file name, usually a string of characters.
  2. **Identifier**: It is unique tag, usually a number that identifies the file within the file system.
  3. **Type:** This information is needed for system that supports different types of files.
  4. **Location:** It is pointer to a device and location of the file on that device.
  5. **Size:** The current size of the files (in bytes, word or blocks).
  6. **Protection:** Access-control information determines who can do reading, writing, executing and so on.

7. **Time, date, and user identification:** It is information of time and date of creation, last modification and last use. These data can be useful for protection, security, and usage monitoring.

- The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.

5.2.2 File Operations

- A file is an abstract data type. To define the file properly, we need to consider the operation that can be performed on files.
- The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

(a) Creating a File:

- Two steps are necessary to create a file which are:
  1. Find space for new file in the file system.
  2. Make an entry for the new file in the directory.

(b) Writing a File:

1. To write a file, name of the file and the information to be written to the file should be provided to system call.
2. Search the file name in the directory to find the file's location.
3. Place the write pointer to the location in the file where next write is to take place.
4. Update the write pointer whenever a write occurs.

(c) Reading a File:

- To read from a file,
  1. Specify the name of the file and where (in memory) the next block of the file should be put is given to system call.
  2. Search the file name in the directory.
  3. Place the read pointer to the location in the file where the next read is to take place.
  4. Update the read pointer whenever the read occurs.
- Because a process is usually either reading from or writing to a file, the current operation location can be kept as a per-process current-file-position pointer. Both the read and write operations use this same pointer, saving space and reducing system complexity.

(d) Repositioning within a File:

1. Search the file name in the directory.
2. Current-file-position pointer is repositioned to a given value.

- Repositioning within a file need not involve any actual I/O. This file operation is also known as a file seeks.

(e) Deleting a File:

- To delete a file,
  1. Search the directory for the named file.
  2. Release all the file space, so that it can be refused by the other files.
  3. Erase the directory entry.

(f) Truncating a File:

- The user may want to erase the contents of a file and then recreate it. This function allows all attributes to remain unchanged – except for file length. Length of file is set to zero and its file space is released.

(g) Searching a file:

- Most of the file operations mentioned involve searching the directory for the entry associated with the named file. To avoid this constant searching, many system require that an open() system call to be made before a file is first used actively.
- The operating system keep a small table, called the open-file table, containing information about files. When a file operation is requested, the file is specified via an index into this table, so no searching is required.

(h) Closing a file:

- When a file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table. Create and delete are system calls that work with closed rather than open files.
- The implementation of the open() and close() operations is more complicated in an environment where several processes may open the file at the same time.
- This may occur in a system where several different applications open the same file at the same time. The operating system uses two levels of internal table namely a per-process table and a system –wide table.
- The per-process table tracks all files that a process has open as shown in Fig. 5.1.
- It stores the information regarding the use of the file by the process. For example, the current file pointer for each file, access rights and accounting information etc. Each entry in the per-process table in turn points to a system-wide open–file table.

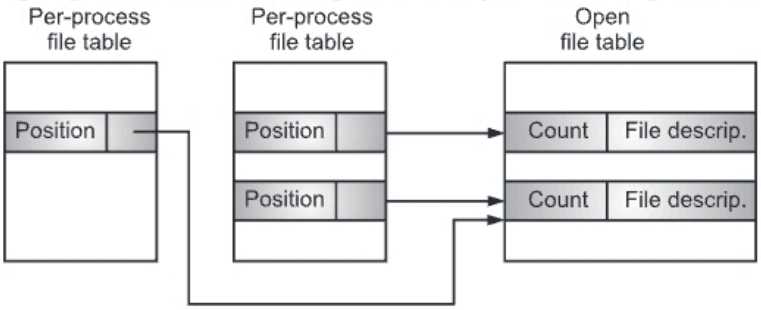


Fig. 5.1: OS Data Structure for Files

- It contains process- independent information such as the location of the file on disk, access dates and file size etc. Once file has been opened by process, this table includes an entry for the file.

Information associated with open file kept in open-file table:

- The following information associated with open file kept in open-file table is as follows:
  1. **File pointer:** On system that do not include a file offset as part of the read() and write() system calls, the system must track the last read – write location as a current file position pointer.
  2. **File-open count:** This counter tracks the number of opened and closed files by each process. It reaches zero on last close, then system can remove the entry from open-file table.
  3. **Disk location of the file:** Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.
  4. **Access rights:** Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O request.

5.2.3 File Types

- File type refers to the ability of the operating system to distinguish different types of file such as text files, source files and binary files etc.
- If an operating system recognizes the types of file, it can then operate on the file in reasonable ways. A common technique for implementing file types is to include the type as part of the file name.
- The file name is spilt into two parts – a name and an extension, usually separated by a period character (.). In this way, we can recognize the file type from name of file itself.

Table 5.1: Different file types and its functions.

File Type	Usual Extension	Function
executable	exe, com, bin or none	Ready-to-run machine-language program.
object	obj, o	Compiled, machine language, not linked.
source code	c, cc, java, pas, asm, a	Source code in various languages.
batch	bat, sh	Commands to the command interpreter.
text	txt, doc	Textual data, documents.
word processor	wp, tex, rtf, doc	Various word-processor formats.
library	lib, a, so, dll	Libraries of routines for programmers.
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing.
archive	arc, zip, tar	Related files grouped into one file, sometimes compressed, for archiving or storage.
multimedia	mpeg, mov, rm, mp3, avi	Binary file containing audio or A/V information.

- Operating system like MS-DOS and UNIX have the following types of files:
  1. **Ordinary Files:** These are the files that contain user information. These may have text, databases or executable program. The user can apply various operations on such files like add, modify, delete or even remove the entire file.
  2. **Directory Files:** These files contain list of file names and other information related to these files.
  3. **Special Files:** These files are also known as device files. These files represent physical device like disks, terminals, printers, networks, tape drive etc. These files are of two types:
     (i) **Character special files:** Data is handled character by character as in case of terminals or printers.
     (ii) **Block special files**: Data is handled in blocks as in the case of disks and tapes.

5.2.4 File Structure

- File structures is the data organization in Secondary Storage Device in such a way that minimize the access time and the storage space.
- A File structure is a combination of representations for data in files and of operations for accessing the data. It allows applications to read, write and modify data.
- A File Structure should be according to a required format that the operating system can understand.
- A file has a certain defined structure according to its type. For example.
  - **A text file:** This is a sequence of characters organized into lines.
  - **A source file:** This is a sequence of procedures and functions.
  - **An object file:** This is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

5.3 ACCESS METHODS

- Stored information in the file must be accessed and read into computer's memory. It can be accessed in several ways like sequentially, direct or by other methods.

5.3.1 Sequential Access Method

- This is the simplest access method. Information in the file is processed in order, one record after the other. A fixed format is used for records and key field uniquely identifies the record.
- Read and write make up the bulk of the operation on the file which always starts from the beginning of file.
  - **Read next** operation reads the next portion of the file and automatically advance file pointer, which tracks the I/O location.

- **Write next** operation appends to the end of the file and advances to the end of the newly written block(i.e. the new end of the file).
- **Rewinding file** reset file pointer to the beginning.

- Sequential access, which is depicted in Fig. 5.2, is based on a tape model of a file and works as well on sequential-access devices.

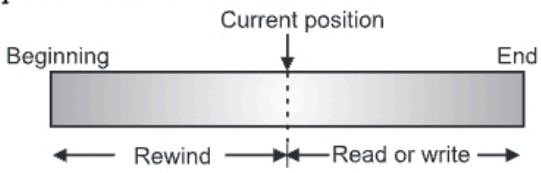


Fig. 5.2: Sequential Access File

Advantages:

1. It is simplest method of searching suitable for tape based systems.
2. If the application needs scanning of all records from file, then sequential access is best.

Disadvantages:

1. It is more time consuming since reading, writing and searching always starts from the beginning of the file.
2. Insertion and deletion operation take more time.

5.3.2 Direct Access Method

- Another method is direct access (or relative access) method. A file is made up of fixed length logical records that allow program to read and write record rapidly in any order.
- This method is based on a disk model of a file, since disk allow random access to any file block. For direct access, the file is viewed as a numbered sequence of block records. So the file operation must include the block number as parameter.
  - **read n**, where n is the block number, file pointer is placed at n^{th} block and reads data of that block.
  - **write n** operation writes data into n^{th} block.
- The block number provided by the user to the operating system is normally a relative block number which acts as an index relative to the beginning of the file.
- Thus, the first relative block of the file is 0; the next is 1, and so on, even though the actual absolute disk address of the block may be different.
- The use of relative block numbers allows the operating system to decide where the file should be placed and helps to prevent the user from accessing portions of the file system that may not be part of his/her file.

Advantages:

1. We can access any record randomly.
2. This method is more suitable for disk based systems.

5.3.3 Other Access Methods

Indexed Access Method:

- The basic form of index includes a record key and storage addresses for a record. To find a record when storage address is unknown, it is necessary to scan the records.
- An index is a separate file from master file. To find a specific record, the index is first searched to find the key of the record required.
- When it is found, corresponding storage address is noted and then the program accesses the record directly. This method uses sequential scan of index and direct accesses to appropriate record. This is fast way to search or access a file.
- There are two types of indexed access files:

1. **Indexed Non-Sequential Access Method:**
   - In this method, master file is not any specific order. There is one entry in index for every record in the master file.
   - When user wants to search any record, then operating system finds address using its key value from index table and move directly to that position.
2. **Indexed Sequential Access Method:**
   - In this, master file is sorted according to key values and index table contains only some key of the records.
   - When user wants to search any record, then operating system finds address using its key value from index table and apply sequential search to find out record from master table.
   - ISAM (Indexed Sequential Access Method) uses a small master index which points to parts of cylinder index. The cylinder index blocks point to track index and tracks index blocks point to the actual file as shown in Fig. 5.3.
   - The file is kept sorted on a defined key. To find a particular record, we first make a binary search of the master index, which provides the block number of the cylinder index.
   - Then make a binary search of the cylinder index, which provides a blocks number of track index. Again within a track, binary search is used to obtain a desire record.

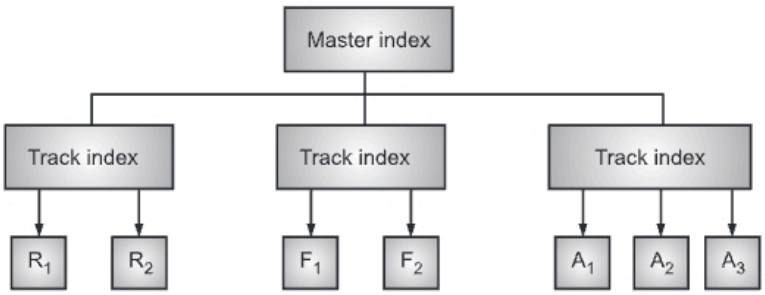


Fig. 5.3: Indexed Sequential Access Method

5.4 DIRECTORY OVERVIEW

- Computer system stores millions of files on disk.
- To manage all these data, we need to organize them. This organization involves use of directories.
- So, directory can be viewed as collection of nodes containing information about files as shown in Fig. 5.4.

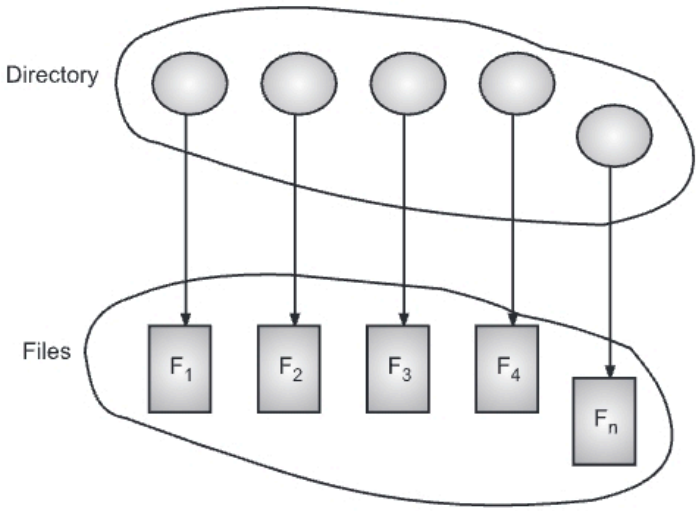


Fig. 5.4: Directory Structure

5.4.1 Storage Structure

- The device directory records information such as name, location, size and type for all files in file system. Volumes can also store multiple operating systems, allowing a system to boot and run more than one.
- The Fig. 5.5 shows typical file organization.

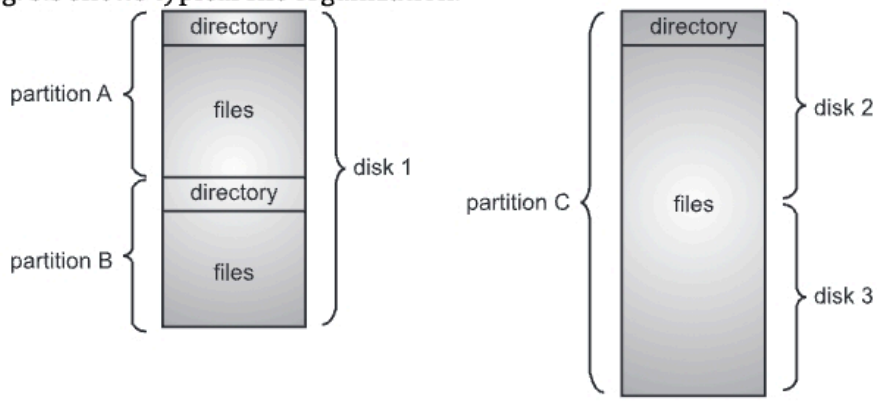


Fig. 5.5: A typical File System Organization

- The directory can be viewed as a symbol table that translates file names into their directory entries.

5.4.2 Operations done on a directory

- When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory:
  1. **Search for a file:** To find the entry for a particular file or find all files whose names match a particular pattern.

2. **Create a file:** New files need to be created and added to the directory.
3. **Delete a file:** To remove a file entry from directory when it is no longer needed.
4. **List a directory:** To list the files in a directory along with its information.
5. **Rename a file:** Renaming a file may also allow its position within the directory structure to be changed.
6. **Backup:** We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals.

- In the following sections, the most common schemes for defining the logical structure of a directory are described.

5.4.3 Single Level Directory

- The simplest directory structure is the single level directory. All files are contained in the same directory, which is easy to support and understand.

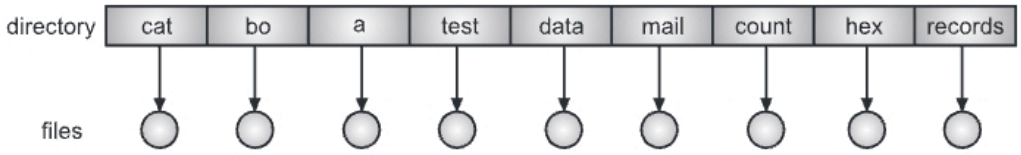


Fig. 5.6: Single level Directory

Advantages:

1. Simple to build.

Disadvantages:

1. As all files are in the same directory, they must have unique names.
2. It is difficult to remember the names of all the files as the number of files increases.
3. In multi-user system, many users may create files with same name.
4. Users can not create their own directory.

5.4.4 Two Level Directory

- The main drawback of single level directory is solved in the two-level directory structure. In this, each user has own User File Directory (UFD).
- The UFDs have similar structure, but each lists only the files of a single user. When a user job start or a user logs in, the system's Master File Directory (MFD) is searched.
- The MFD is indexed by user name or account number and each entry points to the UFD for the user. When a user refers to a particular file, only his UFD is searched.
- Different users may have files with the same name as long as all the file names within each UFD are unique.
- To create a file to user, the operating system searches only that user's UFD to verify whether another file of that name exists.

- To delete a file, the operating system limits its search to the local UFD; thus it cannot accidentally delete another user's file that has a same name.

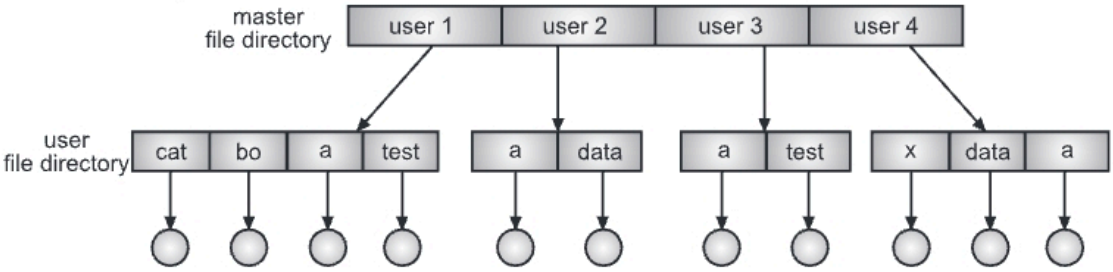


Fig. 5.7: Two level Directory Structure

Advantages:

1. It solves the file name collision problem by creating own user directory.
2. This method isolates one user from another and protects user's files.

Disadvantages:

1. As this structure isolates one user from another, sharing of tasks and access one another's files is not possible.
2. If access is to be permitted, user must have the ability to name a file in another user's directory (i.e. user must know pathname of the file desired.)
3. In case of system's files, which are required by all users, has to be copied in each users directory. It leads to wastage of memory.

5.4.5 Tree Structure Directory [S-19]

- The generalization of the two level directory structure is an arbitrary tree structure directory. This generalization allows users to create their own subdirectories and to organize their files accordingly.
- Tree structure directory is the most common directory structure. The tree has a root directory, and every file in the system has unique path name.
- A directory (or subdirectory) contains a set of files or subdirectories. It 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 sub directory (1).
- Special system calls are used to create, delete and change directories. Path names can be of two types i.e., absolute and relative.
  1. **An absolute path name** begins at the root and follows path down to the specified file, giving the directory names on the path.
  2. **A relative path name** defines a path from the current directory to that specified file.

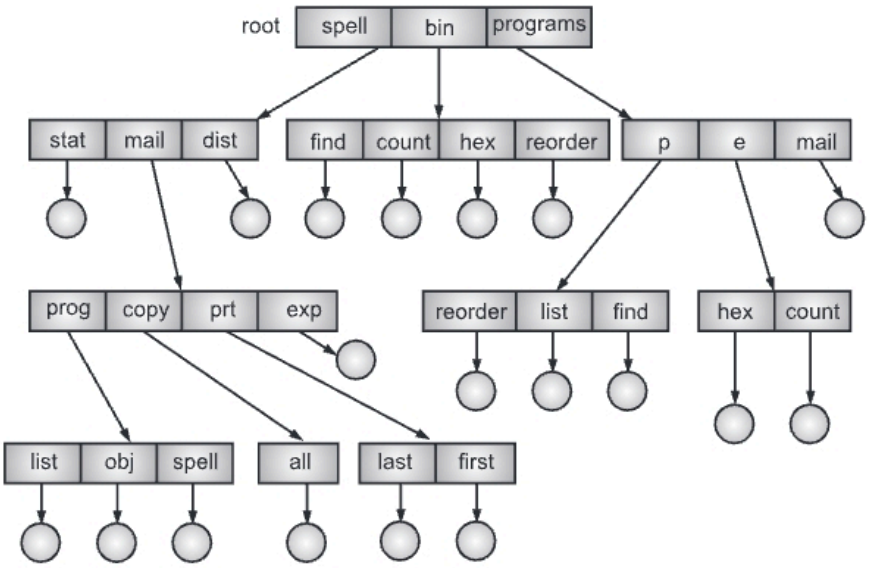


Fig. 5.8: Tree-structured file system(directory)

- For example, in the tree-structured file system of Fig. 5.8, if the current directory is root/spell/mail, then the relative path name prt/first/ refers to the same file as does the absolute path name root/spell/ mail/prt/ first.

Deletion of Files:

- If directory is empty, its entry in the directory that contains it can be deleted. But if the directory to be deleted is not empty, then two approaches are taken:
  1. Operating systems such as MS-DOS will not delete a directory unless it is empty. Thus, to delete directory, the user must first delete all the files in that directory. If sub-directories exist, then same procedure must applied recursively to them, so that they can be deleted. But it results in significant amount of work.
  2. The other approach used by UNIX rm command is to provide an option: when a request is made to delete a directory, all that directory and sub-directories are also to be deleted.

Advantages:

1. User can create directory as well as subdirectory.
2. Users can access the files of other users.

Disadvantages:

1. The tree structure can create duplicate copies of the files.
2. The users could not share files or directories.

5.4.6 Acyclic Graph Directory

- A tree structure prohibits the sharing of files or directories. Consider two programmers who are working on a joint project.

- The files associated with that project can be stored in a subdirectory, separating them from other projects and files of two programmers. But since both programmers are equally responsible for the project, both want the subdirectory to be in their own directories. So there is need that the common subdirectories should be shared.

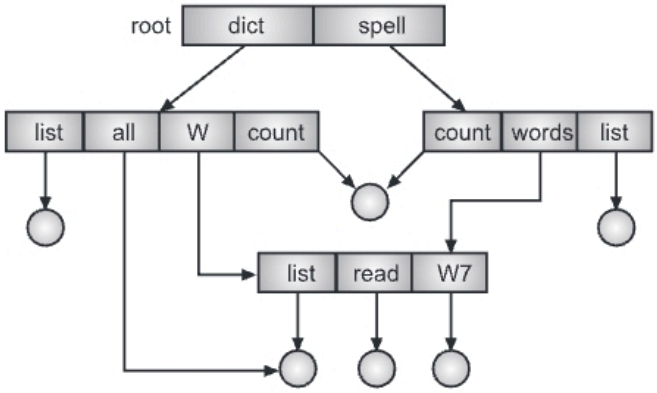


Fig. 5.9: Acyclic graph Directory Structure

- An **acyclic graph** that is, a graph with no cycles allows directories to share subdirectories and files as shown in following Fig. 5.9.
- The same file or subdirectory may be in two different directories. It is natural generalization of the tree- structured directory scheme.

Implementation of Shared Files:

- It is important to note that a shared file (or directory) is not the same as two copies of the file. With two copies, each programmer can view copy rather than the original one, but if programmer changes the file, the changes will not appear in other copies. So inconsistency is big problem with copies and memory wastage.
- With a shared file, only one actual file exists, so any changes made by one person are immediately visible to the other.
- Sharing is particularly important for subdirectories; a new file created by one person will automatically appear in all the shared subdirectories.
- But also with a single copy, several concurrent updates to a file may result in user obtaining incorrect information, and the file being left in an incorrect state.
- Shared files and subdirectories can be implemented in several ways:
  1. A common way is to create a new directory entry called a link. A link is effectively a pointer to another file or subdirectory. For example, a link may be implemented as an absolute or a relative path name. When a reference to a file is made, we search the directory. In the directory, entry is marked as link, and then the link is resolved by using that path name to locate the real file.
  2. Another common approach to implementing shared files is simply to duplicate all information about them in both sharing directories. Thus both entries are identical and equal.

Deletion of Shared File:

- Another problem involves deletion of shared files. One possibility is to remove the file whenever anyone deletes it, but this action may leave dangling pointer to the nonexistent file.

- To overcome this problem, a reference count is maintained for shared files. Whenever a user shares file, reference count is incremented and whenever a user deletes a file, it is decremented.
- When reference count becomes zero, at that time only the file is actually deleted.

Advantages:

1. It is more flexible than tree-structure directories.
2. It allows sharing of files and directories.
3. Implementation of algorithm is simple to traverse the graph.

Disadvantages:

1. It is difficult to ensure that there are no cycles.
2. Implementation of shared file is complicated.

5.4.7 General Graph Directory

- When we add links to the existing tree structure directories, the tree structure is destroyed, resulting in a simple graph directory as shown in Fig. 5.10. This scheme gives complete flexibility in sharing files.
- If cycles are allowed then algorithm should take care of the following:
  - Not allowing the reference to file to go in an infinite loop, because of cycle.
  - Garbage collection method should be implemented in case of deletion of shared files.
- Garbage collection involves traversing the entire files system, marking everything that can be accessed. Then, a second pass collects everything that is not marked onto a list of free space.

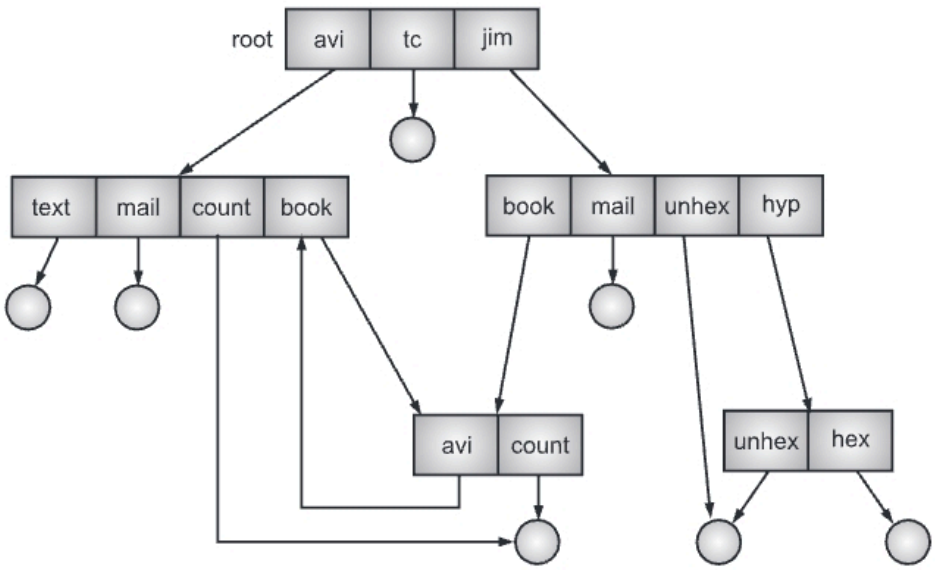


Fig. 5.10: General Graph Directory

Advantages:

1. Sharing of files.
2. Directories are more flexible than acyclic graph structure.

Disadvantages:

1. It requires separate algorithm for garbage collection.
2. Garbage collection for disk based file is time consuming.
3. While traversing, a poorly designed algorithm might result in an infinite loop continually searching through cycle and never terminating.

5.5 FILE SYSTEM STRUCTURE AND IMPLEMENTATION - PARTITIONS AND MOUNTING, VIRTUAL FILE SYSTEMS

- File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.
- The file system design problem is divided into two groups.
  1. The first group is dealing with how the file system should look to the user. It includes the identification of a file and its attributes, operations allowed on a file and the directory structure.
  2. Second group is defining algorithm and data structure used to map the logical file system onto the physical secondary storage devices.
- The file system is divided into many different levels as shown in Fig. 5.11. These are:
  (i) Application Programs (AP)
  (ii) Logical File System (LFS)
  (iii) File Organization Module (FOM)
  (iv) Basic File System (BFS)
  (v) Input/Output Control (IOC)
  (vi) Physical Devices.

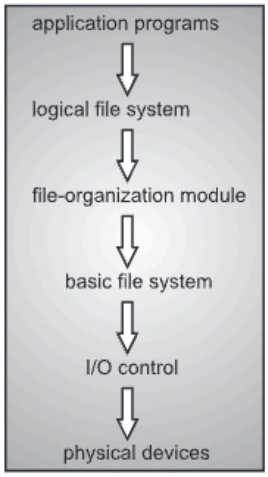


Fig. 5.11: Layered File System

- File systems organize storage on disk drives, and can be viewed as a layered design:
  - At the lowest layer are the **physical devices**, consisting of the magnetic media, motors & controls, and the electronics connected to them and controlling them. Modern disk put more and more of the electronic controls directly on the disk drive itself, leaving relatively little work for the disk controller card to perform.
  - **I/O Control** consists of **device drivers**, special software programs (often written in assembly) which communicate with the devices by reading and writing special codes directly to and from memory addresses corresponding to the controller card's registers. Each controller card (device) on a system has a different set of addresses (registers, a.k.a. **ports**) that it listens to, and a unique set of command codes and results codes that it understands.

- The **basic file system** level works directly with the device drivers in terms of retrieving and storing raw blocks of data, without any consideration for what is in each block. Depending on the system, blocks may be referred to with a single block number, (e.g. block # 234234), or with head-sector-cylinder combinations.
- The **file organization module** knows about files and their logical blocks, and how they map to physical blocks on the disk. In addition to translating from logical to physical blocks, the file organization module also maintains the list of free blocks, and allocates free blocks to files as needed.
- The **logical file system** deals with all of the meta data associated with a file (UID, GID, mode, dates, etc.), i.e. everything about the file except the data itself. This level manages the directory structure and the mapping of file names to **file control blocks, FCBs**, which contain all of the metadata as well as block number information for finding the data on the disk.

- The layered approach to file systems means that much of the code can be used uniformly for a wide variety of different file systems, and only certain layers need to be file system specific. Common file systems in use include the UNIX file system, UFS, the Berkeley Fast File System, FFS, Windows systems FAT, FAT32, NTFS, CD-ROM systems ISO 9660, and for Linux the extended file systems ext2 and ext3.

5.5.1 File-System Implementation

- File-system needs to maintain on-disk and in-memory structures, on-disk for data storage, in-memory for data access.
- File systems store several important data structures on the disk as follows:
  - A **boot-control bloc*k***, (per volume) the **boot block** in UNIX or the **partition boot sector** in Windows contains information about how to boot the system off of this disk. This will generally be the first sector of the volume if there is a bootable system loaded on that volume, or the block will be left vacant otherwise.
  - A **volume control block*,*** (per volume) the **master file table** in UNIX or the **superblock** in Windows, which contains information such as the partition table, number of blocks on each filesystem, and pointers to free blocks and free FCB blocks.

File permissions
File dates (create, access, write)
File owner, group, ACL
File size
File data blocks or pointers to file data blocks

Fig. 5.12: A Typical file-control block

- A directory structure (per file system), containing file names and pointers to corresponding FCBs. UNIX uses inode numbers, and NTFS uses a **master file table.**
- The **File Control Block, FCB,** (per file) containing details about ownership, size, permissions, dates, etc. UNIX stores this information in inodes, and NTFS in the master file table as a relational database structure.

- There are also several key data structures stored in memory as follows:
  - **An in-memory mount table**: Mount table storing file system mounts, mount points, file system types
  - An in-memory directory cache of recently accessed directory information.
  - **A system-wide open file table,** containing a copy of the FCB for every currently open file in the system, as well as some other related information.
  - **A per-process open file table,** containing a pointer to the system open file table as well as some other information. (For example the current file position pointer may be either here or in the system file table, depending on the implementation and whether the file is being shared or not.)
  - **I/O Memory Buffers**:This data structure holds file-system blocks while they are being read from or written to disk.
- Fig. 5.13 illustrates some of the interactions of file system components when files are created and/or used:
  - When a new file is created, a new FCB is allocated and filled out with important information regarding the new file. The appropriate directory is modified with the new file name and FCB information.
  - When a file is accessed during a program, the open( ) system call reads in the FCB information from disk, and stores it in the system-wide open file table. An entry is added to the per-process open file table referencing the system-wide table, and an index into the per-process table is returned by the open( ) system call. UNIX refers to this index as a **file descriptor**, and Windows refers to it as a **file handle**.
  - If another process already has a file open when a new request comes in for the same file, and it is sharable, then a counter in the system-wide table is incremented and the per-process table is adjusted to point to the existing entry in the system-wide table.
  - When a file is closed, the per-process table entry is freed, and the counter in the system-wide table is decremented. If that counter reaches zero, then the system wide table is also freed. Any data currently stored in memory cache for this file is written out to disk if necessary.

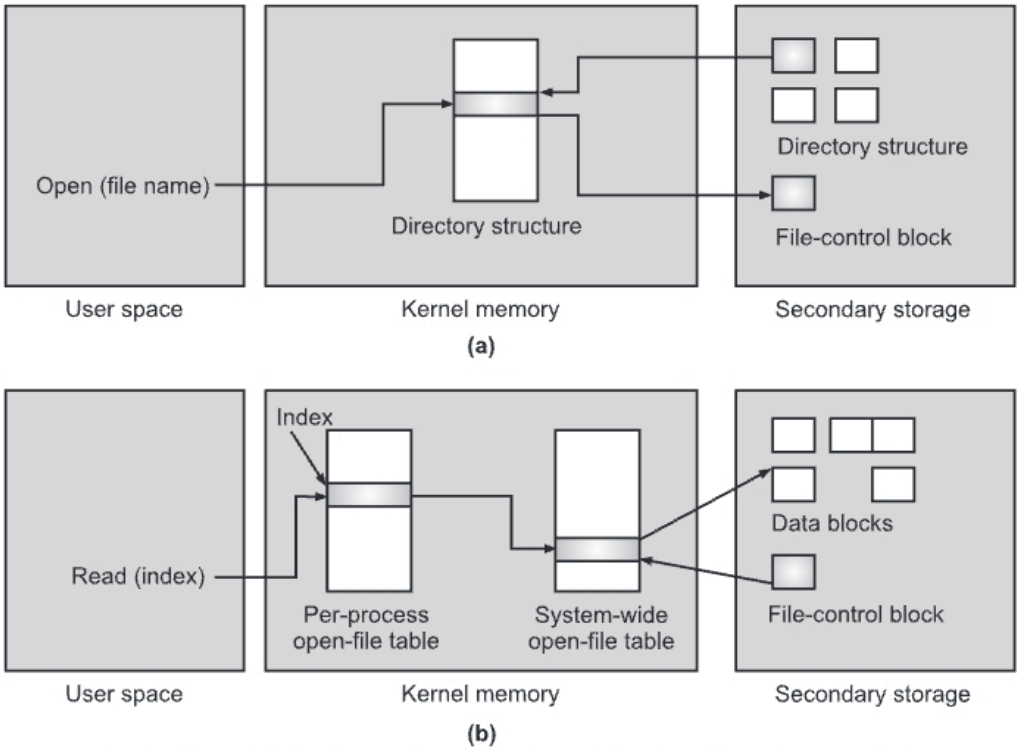


Fig. 5.13: In-memory file-system structures (a) File open (b) File read

5.5.2 Partitions and Mounting

- Physical disks are commonly divided into smaller units called **Partitions**. They can also be combined into larger units, but that is most commonly done for RAID installations.
- Partitions can either be used as raw devices (with no structure imposed upon them), or they can be formatted to hold a file system (i.e. populated with FCBs and initial directory structures as appropriate). Raw partitions are generally used for swap space, and may also be used for certain programs such as databases that choose to manage their own disk storage system. Partitions containing file systems can generally only be accessed using the file system structure by ordinary users, but can often be accessed as a raw device also by root.
- The boot block is accessed as part of a raw partition, by the boot program prior to any operating system being loaded. Modern boot programs understand multiple OS and file system formats, and can give the user a choice of which of several available systems to boot.
- The **root partition** contains the OS kernel and at least the key portions of the OS needed to complete the boot process. At boot time the root partition is mounted, and control is transferred from the boot program to the kernel found there.
- Continuing with the boot process, additional file systems get mounted, adding their information like mount points, types of file systems mounted, and an access path to the desired file system into the appropriate mount table structure.

साधु ने दिया मर्डरर को एक चांस!

- As a part of the mounting process the file systems may be checked for errors or inconsistencies, either because they are flagged as not having been closed properly the last time they were used, or just for general principals. File systems may be mounted either automatically or manually.
- In UNIX, a mount point is indicated by setting a flag in the in-memory copy of the inode, so all future references to that inode get re-directed to the root directory of the mounted file system.
- In UNIX, there is only one root file system that spans many disks. Different directories may actually exist on a different physical disk.
- To bind a directory to a physical device (like a partition or a CD-ROM) so that the device's file system can be read is called mounting the device.

5.5.3 Virtual File Systems

- A Virtual File System (VFS) is programming that forms an interface between an operating system's Kernel and a more concrete file system
- It manages the data storage and retrieval between the operating system and the storage sub-system.
- It gives applications access to different types of file systems and local and network storage devices.
- It separates file system generic operations from their implementation by defining a clean VFS interface. It is also based on a file-representation structure known as vnode, which contains a numerical designator needed to support network file systems.
- The VFS also has a 'lower' interface to the concrete file systems, which is labeled VFS interface. This interface consists of several dozen function calls that the VFS can make to each file system to get work done. VFS has two distinct interfaces: the upper one to the user processes and the lower one to the concrete file systems.
- VFS supports remote file systems using the **NFS (Network File System)** protocol.
- The VFS in Linux is based upon four key object types:
  - The **inode** object, representing an individual file. It contains metadata about a file (location, access mode, owners...)
  - The **file** object, representing an open file.
  - The **superblock** object, representing a filesystem. It defines the file system type, size, status, and other metadata.
  - The **dentry** object, representing a directory entry. It associates names to inodes, and the directory layout.
- Linux VFS provides a set of common functionalities for each filesystem, using function pointers accessed through a table. The same functionality is accessed through the same table position for all filesystem types, though the actual functions pointed to by the pointers may be filesystem-specific. Common operations provided include open( ), read( ), write( ), and mmap( ).

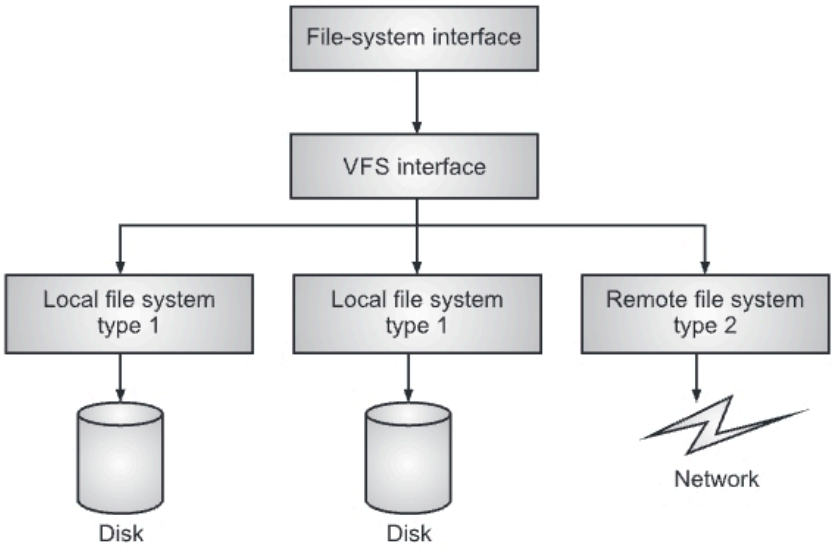


Fig. 5.14: Schematic view of a virtual file system

5.6 ALLOCATION METHODS [W-18]

- The direct-access nature of disks allows us flexibility in the implementation of files. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.
- Three major methods of allocating disk space are contiguous, linked and indexed. Each method has its advantages and disadvantages. More commonly, a system uses one method for all files within file system type.

5.6.1 Contiguous Allocation Method

- In this method, consecutive free blocks are allocated to a file. Directory contains name of the file, starting address and number of blocks.
- If the file is n blocks long and start at location b, then it requires blocks b, b + 1, b + n – 1 as shown in following Fig. 5.15.
- For sequential access of record, the file system must remember the disk address of the last block read so that next block is easily read.

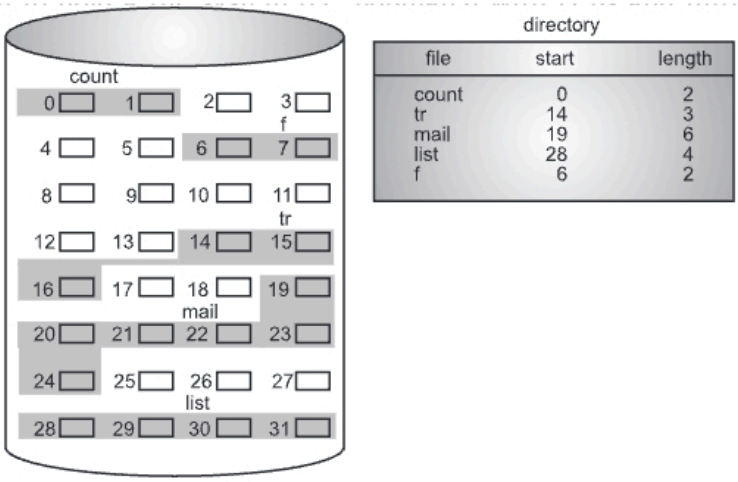


Fig. 5.15: Contiguous Allocation of Disk Space

- For direct access of block i of a file that starts at block b, we can directly access (b + i) block. Thus, this method supports sequential and direct access.
- It is used in IBM 370.

Advantages:

1. Supports both sequential and direct access.
2. Reading all blocks belonging to each file is very fast.

Disadvantages:

1. Difficulty in searching space for new files.
2. To allocate a file, one needs to know the size of the file in advance.
3. If new block are freed from the file, they will remain unallocated (because their consecutive blocks are not free) creating external fragmentation.
4. Problem is there in the cases where files grow after some days. In this case, either user program is terminated or a new big free space is found and the file is copied to it.

Dynamic Storage Allocation Problem in Contiguous Allocation:

- Disk space is nothing but large array of disk block. At any instance of time, few blocks are assigned to file and some blocks are freed by the file.
- At any moment, we may find that allocated as well as free blocks are spread all over the disk. The unallocated (free) blocks are also called as holes. There may be number of consecutive holes.
- Now whenever a user has a request of size n, either of the following method is used to search a hole big enough to satisfy the requirement of n units from the list of free holes.

1. **First fit:** In this case, as soon as the first hole (that is big enough) is encountered, searching is stopped and memory is allocated for creating a file.
   - Most simple and less time consuming method to search a hole.
   - Poor memory utilization.
   - It creates internal fragmentation.
2. **Best fit:** In this case, the entire list is searched for and the smallest hole, that satisfy the request is selected and is allocated for creating file.
   - Memory utilization is the best.
   - Internal fragmentation is less.
   - As compared to first and worst fit algorithm is more complex.
   - It is a time consuming process.
3. **Worst fit:** Again the entire list is scanned, and the biggest hole/block satisfying the requirement (i.e. biggest > n) is selected.
   - Time consuming.
   - Memory utilization is very poor.
   - Internal fragmentation is big problem.

Compaction:

- Compaction is used to improve memory utilization, which is poor because of external fragmentation. Compaction is process of collecting all free holes into one big hole.

5.6.2 Linked Allocation Method

- Linked allocation is essentially a disk-based version of the linked list. In this method, a linked list of the all block belonging to the file is maintained. These blocks may be scattered through the disk.

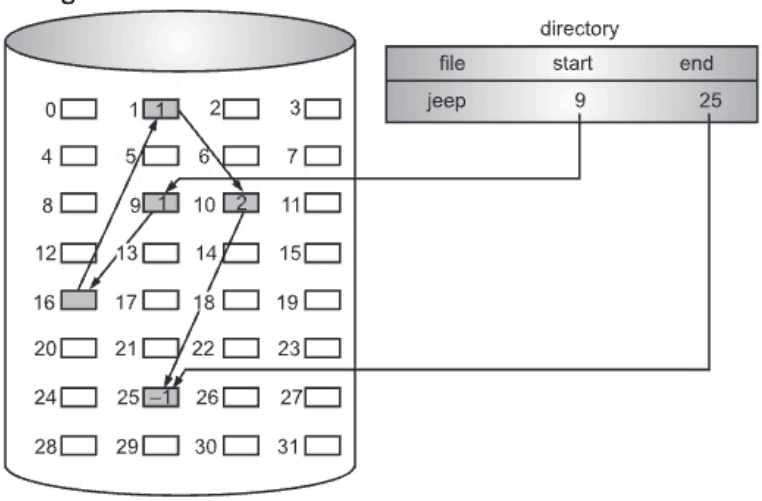


Fig. 5.16: Linked Allocation of Disk Space

- Directory entry contains the name of the file, address of starting block and the last block. If we want to read a file, we simply follow the pointers. If we want to write to a file, any free block is removed from free list and writes to it. This block is appended at the end of the file with pointer.
- It is used in DEC TOPS-10, Xerox Alto. The Fig. 5.16 shows file jeep of five blocks 9-16-1-10-25.

Advantages:

1. No external fragmentation because any free block is used to satisfy the request.
2. Files can grow any time.
3. No need to know the size of the file in advance.
4. No disk compaction.

Disadvantages:

1. Only sequential access is possible.
2. Memory is required for storing the pointers. Therefore, file requires slightly more space.
3. Reliability is big problem. Since disk blocks are linked by pointers, a single damaged pointer can make thousands of disk blocks inaccessible. So we can not access any portion of file or entire file.
4. Bug in an operating system may result in picking up wrong dangling pointers from the memory.

- A **File Allocation Table (FAT)** is a variation of Linked allocation. It uses a separate disk area to hold the links. This method doesn't use space in data blocks. Many pointers may remain in memory.
- A FAT file system is used by MS-DOS.

5.6.3 Indexed Allocation [W-18]

- One disadvantage with linked allocation method is that it does not support direct accessing since blocks are scattered all over the disk.
- This problem is solved by indexed allocation by placing all of the pointers together into an index block.
- In this method of allocation, an index block is there for each file. It contains addresses of all blocks belonging to that file sequentially e.g. i^{th} entry in the index block of a file will contain the address of ith block belonging to that file.
- A directory entry contains the name of the file and address of the index block. In case of large files, a single index block holds the addresses of all index blocks. These indexes may contain addresses of few more indexed blocks or addresses of actual block belonging to the file. This is called as level of indices.
- In the Fig. 5.17 Block number 19th contain address, 9th, 16th, 1st 10th and 25th blocks. Address of index block 19th is present in the directory entry.

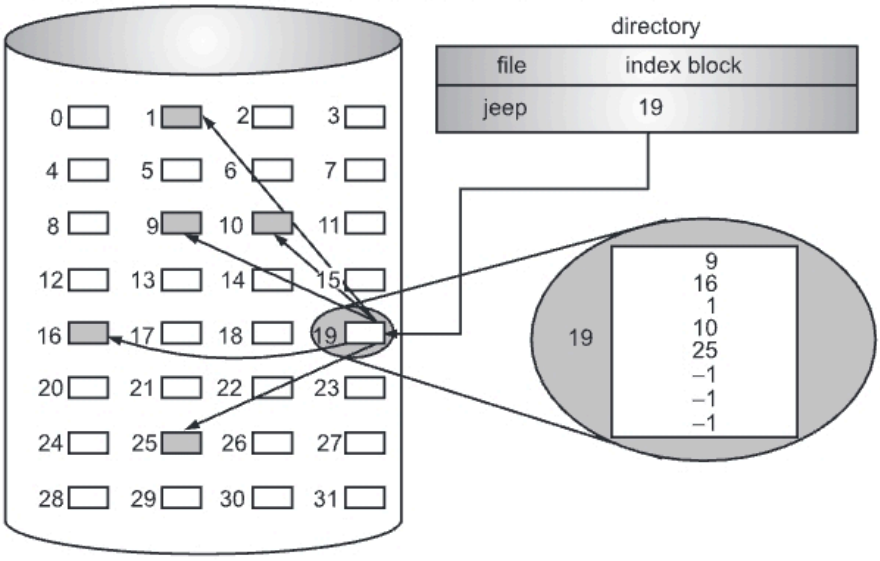


Fig. 5.17: Indexed Allocation of Disk Space

- It is used in DEC VMS, Nachos.

Advantages:

1. Direct access possible.
2. No need for user to know size of the file in advance.
3. No external fragmentation.

Disadvantages:

1. Overhead of index blocks is not feasible for very small file.
2. Overhead of index block is not feasible for very big file also. It is also very difficult to manage levels of indices.
3. File access time is increased because, for any operation of the file, index block has to be accessed, which increases input / output.
4. Keeping index in memory requires space.

- A question is of how big the index block should be, and how it should be implemented. There are several approaches:
  1. **Linked Scheme**: An index block is one disk block, which can be read and written in a single disk operation. The first index block contains some header information, the first N block addresses, and if necessary a pointer to additional linked index blocks.
  2. **Multi-Level Index**: The first index block contains a set of pointers to secondary index blocks, which in turn contain pointers to the actual data blocks.
  3. **Combined Scheme:** This is the scheme used in UNIX inodes, in which the first 12 or so data block pointers are stored directly in the inode, and then singly, doubly, and triply indirect pointers provide access to more data blocks as needed.
- Let us summarize briefly three allocation methods.

Table 5.2: Summary of Allocation Methods

Factors	Contiguous Allocation Method	Linked Allocation Method	Indexed Allocation Method
Allocation method	Array of disk blocks.	Linked list of disk blocks which are scattered throughout disk.	All pointers to blocks are placed together in one block known as index block.
Access method	Sequential and direct.	Only sequential	Sequential and direct.
Directory entry	File name, start location of file and size of file.	File name, start block and end block.	File name and address of index block.
File Size	Should be known in advance and size is fixed.	No need to know in advance and file size can grow at any time.	No need to know in advance and file size can grow at any time.
Fragmentation	Suffers from external and internal fragmentation.	No external fragmentation.	No external fragmentation.
Reliable	Yes	No, problem of dangling pointers.	Yes

5.7 FREE SPACE MANAGEMENT

- To store a file on the disk, operating system needs to know which blocks on the disk are free and which are not. The operating system maintains free space list to keep the track of free disk space.
- When a file is created, the required amount of the space is searched from free space list and allocated to the file. This space is then removed from the free space list.
- When a file is deleted, the space which was allocated to a file is freed and added to the free space list.

Free space management techniques:

- Following are some methods which manages free-space list information.

1. **Bit Map / Bit Vector** :
   - This is method of keeping track of allocated and unallocated blocks. Each block is represented by one bit.
   - If the block is free, the bit is set to 1 and if the block is allocated, the bit is set to 0.
   - For example, consider a disk where blocks 4, 5,8,9,10,12,13,15,18,19,25 are free, the free space bit map or bit vector is: **00011001110110100110000 01**.
2. **Linked List:**
   - Linked list of free blocks is maintained.
   - Pointer to first free block is preserved.
   - Fist free block contain address of the next free block and so on.
   - The following Fig. 5.18 shows linked list of free blocks in which pointer is at block 2 which is first free block. Block 2 would contain pointer to block 3 and so on.
   - However, this scheme is not efficient, since traversing of list is time consuming and requires reading of each block which requires substantial I/O.

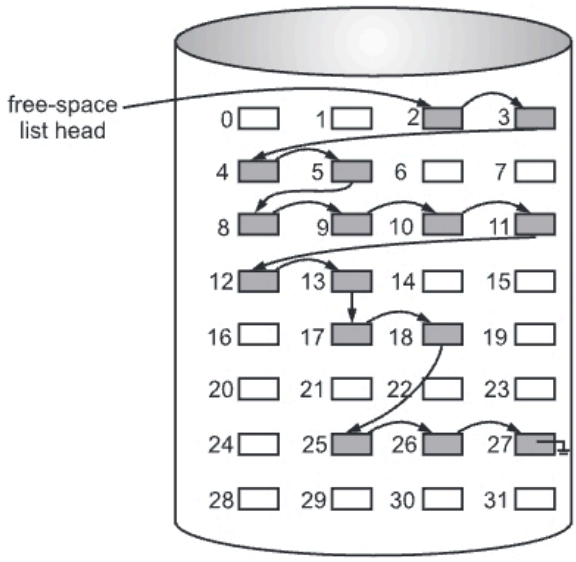


Fig. 5.18: Linked Free-space List on Disk

3. **Grouping:**
   - To resolve the problem occurred in linked free space list, the addresses of n free blocks in the first free block.
   - The first n-1 of these blocks are actually free.
   - The last block contains the disk addresses of other n free blocks and so on.
   - In this method, the addresses of a large number of free blocks can be found quickly.
4. **Counting:**
   - When many contiguous free blocks are present, rather than keeping a list of n free disk addresses, we can keep the address of first free block and count of the next n free contiguous blocks.
   - Each entry in the free space list then consists of a disk address and a count.
5. **Space Maps:**
   - Sun's ZFS file system was designed for HUGE numbers and sizes of files, directories, and even file systems.
   - The resulting data structures could be VERY inefficient if not implemented carefully. For example, freeing up a 1 GB file on a 1 TB file system could involve updating thousands of blocks of free list bit maps if the file was spread across the disk.
   - ZFS uses a combination of techniques, starting with dividing the disk up into (hundreds of) meta-slabs of a manageable size, each having their own space map.
   - Free blocks are managed using the counting technique, but rather than write the information to a table, it is recorded in a log-structured transaction record. Adjacent free blocks are also coalesced into a larger single free block.
   - An in-memory space map is constructed using a balanced tree data structure, constructed from the log data.
   - The combination of the in-memory tree and the on-disk log provide for very fast and efficient management of these very large files and free blocks.

Summary

- A file is a named collection of related information that is recorded on secondary storage. It is an abstract data type defined and implemented by the operating system. It is a sequence of logical records. A logical record may be a byte, a line (of fixed or variable length), or a more complex data item.
- Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.
- Files are mapped by the operating system onto physical devices.

- Early file systems were tape based. Each file was implemented by mapping it into its own reel of tape. It allows sequential access to data.
- When files are stored on Hard disk, it is called disk based system. It allows direct access to the data.
- Every file has attributes which vary from one operating system to another but typically consist of these: Name, Identifier, Type, Location, Size, Protection and Time, date, and user identification.
- Operations on file are create, open ,read ,write , repositioning file, delete file, truncate file close file etc.
- File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc.
- A common technique for implementing file types is to include the type as part of the file name.
- Stored information in the file must be accessed and read into computer's memory by following methods: sequentially, direct or by other methods like indexed sequential.
- A sequential access is that in which the records are accessed in some sequence i.e. the information in the file is processed in order, one record after the other.
- Random access file organization provides, accessing the records directly. Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- Indexed Sequential mechanism is built up on base of sequential access. An index is created for each file which contains pointers to various blocks. Index is searched sequentially and its pointer is used to access the file directly.
- Each device in a file system keeps a volume table of contents or a device directory listing the location of the files on the device. In addition, it is useful to create directories to allow files to be organized.
- The directory can be viewed as a symbol table that translates file names into their directory entries.
- In single level directory, all files are contained in the same directory, which is easy to support and understand. But in a multiuser system, it causes naming problems, since each file must have a unique name.
- A two level directory solves this problem by creating a separate directory for each user's files. The directory lists the files by name and includes the file's location on the disk, length, type, owner, time of creation, time of last use, and so on.
- The natural generalization of a two-level directory is a tree-structured directory. A tree-structured directory allows a user to create subdirectories to organize files.
- Acyclic graph directory structures enable users to share subdirectories and files but complicate searching and deletion.
- A general graph structure allows complete flexibility in the sharing of files and directories but sometimes requires garbage collection to recover unused disk space.
- File systems provide efficient and convenient access to the disk by allowing data to be stored, located, and retrieved easily.

- The file system is divided into many different levels as Application Programs (AP), Logical file system (LFS), File organization Module (FOM), Basic File System (BFS), Input/Output Control (IOC) and Devices.
- Three major methods of allocating disk space are contiguous, linked and indexed.
- In contiguous allocation method, consecutive free blocks are allocated to a file. Directory contains name of the file, starting address and number of blocks.
- In linked allocation method, a linked list of the all block belonging to the file is maintained. These blocks may be scattered through the disk. Directory entry contains the name of the file, address of starting block and the last block.
- In Indexed Allocation method, an index block is there for each file. It contains addresses of all blocks belonging to that file sequentially. A directory entry contains the name of the file and address of the index block.
- The operating system maintains free space list to keep the track of free disk space. When a file is created, the required amount of the space is searched from free space list and allocated to the file. When a file is deleted, the space which was allocated to a file is freed and added to the free space list.
- Free space management techniques are Bitmap, Linked List, Grouping, Counting and Space Map.

Check Your Understanding

1. A _____ is a collection of related records.
   (a) File (b) Directory
   (c) Disk (d) None of the Above
2. File system provides users with the functionality to _____ files.
   (a) create (b) access
   (c) delete (d) All of the Above
3. A file system organizes data in a structured hierarchical manner via the use of _____, which are containers for storing pointers to multiple files.
   (a) files (b) disks
   (c) directories (d) None of the Above
4. File type can be represented by _____.
   (a) file name (b) file extension
   (c) file identifier (d) None of the Above
5. In which type of allocation method each file occupy a set of contiguous block on the disk?
   (a) Contiguous allocation (b) Dynamic-storage allocation
   (c) Linked allocation (d) Indexed allocation

6. File attributes consist of _____.
   (a) name (b) type
   (c) identifier (d) All of the Above
7. In the sequential access method, information in the file is processed _____.
   (a) one disk after the other, record access doesn't matter
   (b) one record after the other
   (c) one text document after the other
   (d) None of the Above
8. For a direct access file _____.
   (a) there are restrictions on the order of reading and writing
   (b) there are no restrictions on the order of reading and writing
   (c) access is restricted permission wise
   (d) access is not restricted permission wise
9. In the Single level directory _____.
   (a) All files are contained in the same directory
   (b) All files are contained in different directories all at the same level
   (c) Depends on the operating system
   (d) None of the Above
10. In the Two level directory structure _____.
   (a) each user has his/her own user file directory
   (b) the system doesn't its own master file directory
   (c) Both (a) & (b)
   (d) None of the Above
11. In the Tree structured directories _____.
   (a) the tree has the stem directory (b) the tree has the leaf directory
   (c) the tree has the root directory (d) All of the Above
12. When two users keep a subdirectory in their own directories, the structure being referred to is _____.
   (a) tree structure (b) cyclic graph directory structure
   (c) acyclic graph directory (d) None of the Above
13. A tree structure of directory _____ the sharing of files and directories.
   (a) allows (b) may restrict
   (c) restricts (d) none of the mentioned
14. In Contiguous allocation _____.
   (a) each file must occupy a set of contiguous blocks on the disk
   (b) each file is a linked list of disk blocks
   (c) all the pointers to scattered blocks are placed together in one location
   (d) None of the Above

15. In Linked allocation _____.
    (a) each file must occupy a set of contiguous blocks on the disk
    (b) each file is a linked list of disk blocks
    (c) all the pointers to scattered blocks are placed together in one location
    (d) None of the Above
16. In Indexed allocation _____.
    (a) each file must occupy a set of contiguous blocks on the disk
    (b) each file is a linked list of disk blocks
    (c) all the pointers to scattered blocks are placed together in one location
    (d) None of the Above
17. _____ is method of keeping track of allocated and unallocated blocks and each block is represented by one bit. If the block is free, the bit is set to 1.
    (a) Bit Map (b) Bit Vector
    (c) Both (a) and (b) (d) None of the Above
18. The mounting of file system is _____.
    (a) to delete of a file system.
    (b) to create a file system.
    (c) to attach portion of the file system into a directory structure.
    (d) to remove the portion of the file system into a directory structure.
19. An "inode" in Virtual file system is _____.
    (a) Data (b) Files and Directories
    (c) Buffer (d) Logs
20. Superblock in Windows contains information about _____.
    (a) the partition table
    (b) number of blocks on each file system
    (c) pointers to free blocks and free FCB blocks
    (d) All of the mentioned

Answers

1. (a)	2. (d)	3. (c)	4. (b)	5. (a)	6. (d)	7. (b)	8. (b)	9. (a)	10. (a)
11. (c)	12. (c)	13. (c)	14. (a)	15. (b)	16. (c)	17. (c)	18.(c)	19.(b)	20.(d)

Practice Questions

Q.I Answer the following questions in short.

1. What is file system?
2. What is the purpose of file system?
3. What is directory?

4. Enlist attributes of files.
5. Which are different file access methods?
6. What is meant by file allocation?
7. Justify "Newly created directory will have two entries automatically in it"?
8. Write names of the free space management techniques.
9. What is virtual file system?

Q.II Answer the following questions.

1. Describe various directory structures in details.
2. Explain file operations in detail.
3. Explain the following file allocation methods with advantages and disadvantages: (i) Linked, (ii) Contiguous, (iii) Indexed.
4. Following is sequence of free blocks on a disk (size of blocks are given:
   120, 30, 60, 40, 50, 90, 70). If the disk place allocation request occurs request in following order; 40, 60, 58, 20, 90, 35. Find the allocation of free blocks to the above request using first fit and best fit strategy.
5. Compare linked, contiguous and indexed allocation.
6. What are the advantage and disadvantages of a tree structure directory?
7. Describe single level directory diagrammatically.
8. With the help of diagram describe two level directory structure.
9. Write a short note on: (i) Space map (ii) Grouping and Counting(iii) Acyclic graph directory (iv) File access methods.

Q.III Define the terms:

1. File
2. Directory
3. File Structure
4. Bit map
5. File allocation table

Previous Exam Questions

Winter 2018

1. List advantages and disadvantages of indexed allocation. **[3 M]**

Ans. Refer to Section 5.6.3.

Summer 2019

1. In a bit vector each block is represented by _____ if block is free and _____ if block is allocated. **[1 M]**

(a) 1, 0 (b) 0, 1
(c) 0, 0 (d) 1, 1

Ans. (a)

2. Differentiate between contiguous allocation and linked allocation method of disk space. **[5 M]**

Ans. Refer to Section 5.6.

3. Explain tree-structure directory structure. **[4 M]**

Ans. Refer to Section 5.4.3.

■■■

6... Disk Scheduling

Objectives...

- To learn about Disk Structure and understand the concept of disk scheduling.
- To study of Disk Scheduling Algorithms such as FCFS Scheduling, SSTF Scheduling, SCAN Scheduling, C-SCAN Scheduling, LOOK Scheduling.
- To understand the Disk Management.

6.1 OVERVIEW

- As we know, a process needs two types of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.
- One of the responsibilities of the OS is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth.
- **Disk scheduling** is done by operating systems to schedule I/O requests arriving for the disk. It is also known as I/O scheduling.
- Disk scheduling is important because:
  1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
  2. Two or more request may be far from each other so can result in greater disk arm movement.
  3. Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

6.2 DISK STRUCTURE

- In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.

Structure of a Magnetic Disk:

- A magnetic disk contains several platters. Each platter is divided into circular shaped tracks. The length of the tracks near the centre is less than the length of the tracks farther from the centre.

- Each track is further divided into sectors, as shown in the Fig. 6.1. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.
- Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

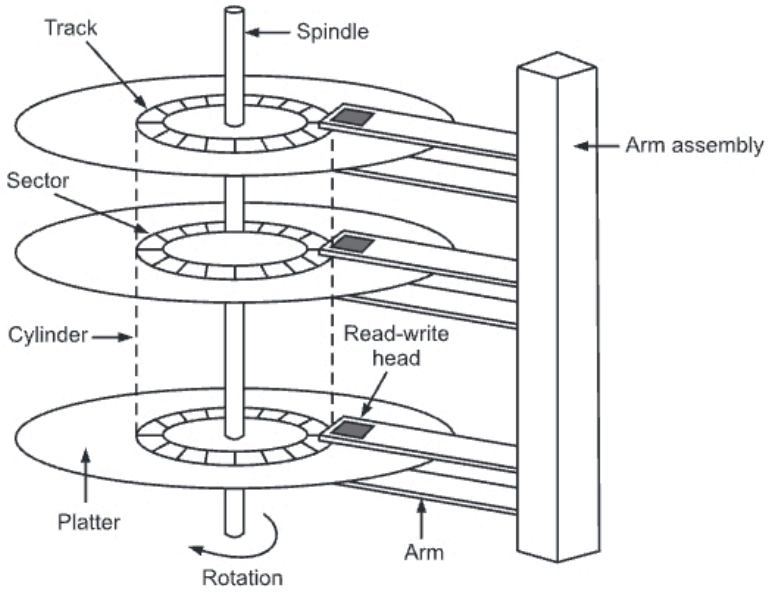


Fig. 6.1: Disk Structure

- The speed of the disk is measured as following two parts:
  - **Transfer Rate** is the rate at which the data moves from disk to the computer.
  - **Random Access Time** is the sum of the seek time and rotational latency.
    - The **seek time** is the time for the disk arm to move the head to the required cylinder containing the desired track.
    - The **rotational latency** is the additional time for the disk to rotate the desired sector to the disk head.
  - The **disk bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.
  - A disk drive is attached to a computer by a set of wires called an I/O bus. Several kinds of buses are available, including Advanced Technology Attachment (ATA), Serial ATA (SATA), eSATA, Universal Serial Bus (USB), and Fiber Channel (FC).
  - The data transfers on a bus are carried out by special electronic processors called controllers. The **host controller** is the controller at the computer end of the bus.
  - A **disk controller** is built into each disk drive. To perform a disk I/O operation, the computer places a command into the host controller, typically using memory-mapped I/O ports.

- Even though the disk is arranged as sectors and tracks physically; the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be 512 or 1024 bytes.
- The logical block is the smallest unit of transfer. The one-dimensional array of logical blocks is mapped into the sectors of the disk sequentially. In this way, each sector in the disk will have a logical address. Sector 0 is the first sector of the first track on the outermost cylinder.
- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

6.3 DISK SCHEDULING

- On a typical multiprogramming system, there will usually be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency.
- The OS is responsible for minimizing access time and maximizing bandwidth for disks. Access time has two major components : Seek time and Rotational latency.
- Whenever, a process needs I/O to or from the disk, it issues a system call to the operating system. The request specifies several pieces of information like whether operation is input or output, disk address, memory address for the transfer, number of sectors to be transferred etc.
- Requests are serviced immediately if the disk is not busy, if disk is busy then requests are queued and disk scheduling is used to choose the next request to service.

6.3.1 Disk Performance Parameters

- When a disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired tack and at the beginning of the desired sector on the track.
- Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed head system.
- On a movable-head system, the time taken to position the head at the track is known a seek time.
- Once, the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes to reach the beginning of the desired sector is known as rotational delay or rotational latency.
- The sum of the seek time, (for movable head system) and the rotational delay is termed as access time of the disk, the time it takes to get into appropriate position (track and sector) to read or write.
- Once the head is in position, the read or write operation is then performed as the sector moves under the head, and the data transfer takes place.

Performance parameters are as follows:

1. **Seek Time:**

- Seek time is the time required to move the disk arm to the required track. The seek time is approximated as,

$$T_s = m \times n + s$$

Where,
T_s = estimated seek time
n = number of tracks traversed
m = constant that depends on the disk drive
s = startup time

2. **Rotational Delay:**

- Disk drive generally rotates at 3600 rpm, i.e. to make one revolution it takes around 16.7 ms. Thus on the average, the rotational delay will be 8.3 ms.

3. **Transfer Time:**

- The transfer time to or from the disk depends on the rotational speed of the disk and it is estimated as,

$$T = \frac{b}{rN}$$

Where,
T = Transfer time
b = Number of bytes to be transferred
N = Numbers of bytes on a track
r = Rotational speed, in revolution per second.

4. **Access Time:**

Access Time = Seek Time + Rotational Delay + Transfer Rate

Thus, the total average access time can be expressed as,

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

where T_s is the average seek time.

5. **Disk Response Time:**

- Response Time is the average of time spent by a request waiting to perform its I/O operation. Average Response time is the response time of the all requests.
- Variance Response Time is the measure of how individual requests are serviced with respect to average response time.

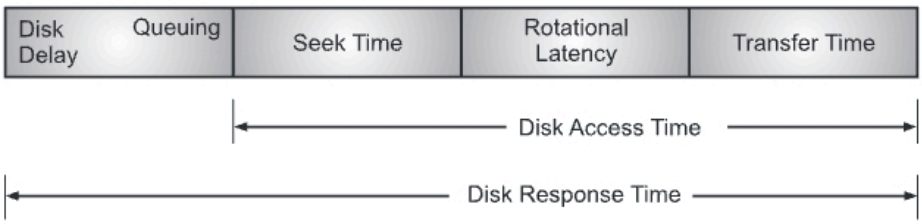


Fig. 6.2: Disk Performance Parameters

- Disks are potential bottleneck for system performances and storage system reliability.
- The disk access time is relatively higher than the time required to access data from main memory and performs CPU operation. Also, the disk drive contains some mechanical parts and it involves mechanical movement, so the failure rate is also high.
- The disk performance has been improving continuously; microprocessor performance has improved much more rapidly.
- A **disk array** is an arrangement of several disks, organized to increase performance and improve reliability of the resulting storage system. Performance is increased through data striping. Reliability is improved through redundancy.
- Disk arrays that implement a combination of data striping and redundancy are called Redundant Arrays of Independent Disks (RAID).

6.3.2 Scheduling Algorithms

- Disk scheduling is done by operating systems to schedule I/O requests arriving for disk.
- Disk scheduling is important because:
  1. Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
  2. Two or more request may be far from each other so can result in greater disk arm movement.
  3. Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.
- Disk scheduling is similar to process scheduling. The disk scheduling algorithm that gives minimum average seek time, minimum rotational latency and minimum variance response time is better.
- Some of the disk scheduling algorithms are described in following sections.

6.3.2.1 FCFS Scheduling

- The simplest form of disk scheduling is the First-Come, First-Served (FCFS) algorithm.
- In FCFS, the requests are addressed in the order they arrive in the disk queue.
- This algorithm is intrinsically fair to all processes, but it generally does not provide the fastest service.

Example:

- Consider a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199.
- All incoming requests are placed at the end of the queue. If the disk head is initially at cylinder 50, it will first move from 50 to 95, 95 then to 180, 34, 119, 11, 123, 62, and finally to 64.

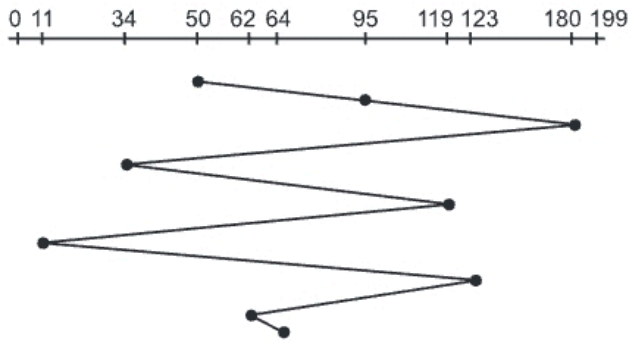


Fig. 6.3: FCFS Scheduling

- To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next.

Total head movement = (95 – 50) + (180 – 95) + (180 – 34) + (119 – 34) + (119 – 11) + (123 – 11) + (123 – 62) + (64 – 62)
= 45 + 85 + 146 + 85 + 108 + 112 + 61 + 2
= 644 tracks

- The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

Advantages:

1. Every request gets a fair chance.
2. No indefinite postponement.

Disadvantages:

1. Algorithm does not try to optimize seek time.
2. It may not provide the best possible service.

6.3.2.2 SSTF Scheduling

- The SSTF (Shortest Seek Time First) algorithm selects the request with the minimum seek time from the current head position. In other words, SSTF chooses the pending request closest to the current head position.
- SSTF scheduling is essentially a form of shortest-job-first (SJF) scheduling; and like SJF scheduling.
- So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time.
- As a result, the request near the disk arm will get executed first.

Example:

- Consider, a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199.

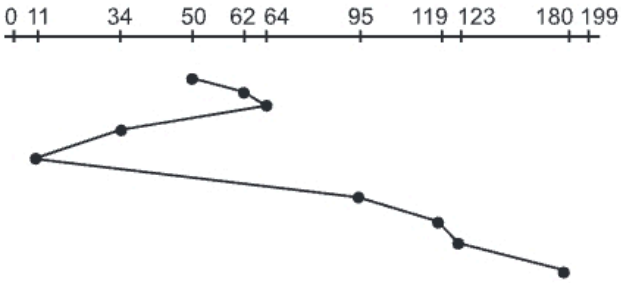


Fig. 6.4: SSTF Scheduling

- In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the processes are taken care of.
- For example, the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way.

Total head movement = (64 – 50) + (64 – 11) + (180 – 11) = 14 + 53 + 169 = 236 tracks

Advantages:

1. It decreases average response Time.
2. It increases throughput.

Disadvantages:

1. Overhead to calculate seek time in advance.
2. It can cause starvation for a request if it has higher seek time as compared to incoming requests.
3. High variance of response time as SSTF favours only some requests.

6.3.2.3 Scan Scheduling

- In SCAN algorithm, the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path.
- So, this algorithm works like an elevator and hence also known as Elevator algorithm. It moves the R/W head back-and-forth to the innermost and outermost track.
- As it scans the tracks from end to end, it process all the requests found in the direction it is headed.
- This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests.

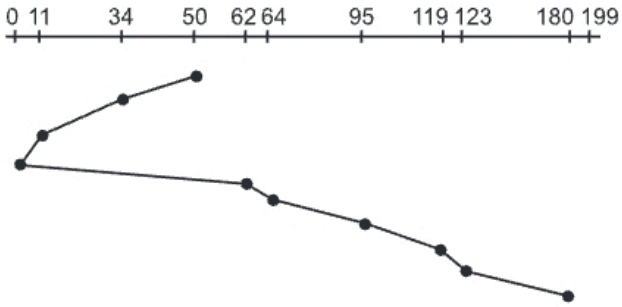


Fig. 6.5: SCAN Scheduling

Example:

- Consider a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199. Assume direction towards 0.

  Total head movement = (50 – 0) + (180 – 0) = 50 + 180 = 230 tracks

Advantages:

1. It gives high throughput.
2. It provides low variance of response time.
3. It provides average response time.
4. It does not lead to starvation.

Disadvantages:

1. Long waiting time for requests for locations just visited by disk arm.
2. It causes the head to move till the end of the disk even if there are no requests to be serviced.

6.3.2.4 C-SCAN Scheduling

- In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. These situations are avoided in C-SCAN algorithm.
- In C-SCAN, the head moves from one end of the disk to the other servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip. It treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

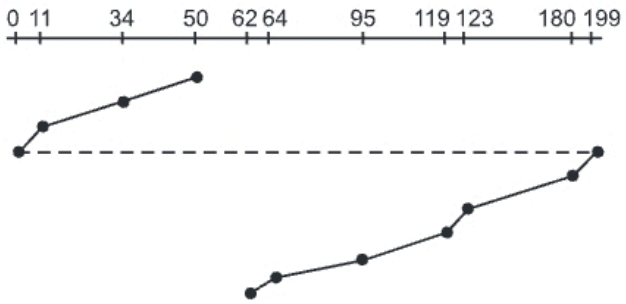


Fig. 6.6 C-SCAN Scheduling

Example:

- Consider a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199. Assume direction towards 0.

  Total head movement = (50 – 0) + (199 – 62) = 50 + 137 = 187 tracks

Advantages:

1. The waiting time for the cylinders just visited by the head is reduced as compared to the SCAN Algorithm.
2. It provides more uniform wait time compared to SCAN.
3. It provides better response time.

Disadvantages:

1. It causes more seek movements as compared to SCAN Algorithm.
2. It causes the head to move till the end of the disk even if there are no requests to be serviced.

6.3.2.5 LOOK Scheduling

- It is similar to the SCAN disk scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.
- The disk arm starts at the first I/O request on the disk, and moves toward the last I/O request on the other end, servicing requests until it gets to the other extreme I/O request on the disk, where the head movement is reversed and servicing continues.
- It moves in both directions until both last I/O requests; more inclined to serve the middle cylinder requests.

Example:

- Consider a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199. Assume direction towards 0.

  Total head movement

  = (50 – 11) + (180 – 11)

  = 39 + 169 = 208 tracks

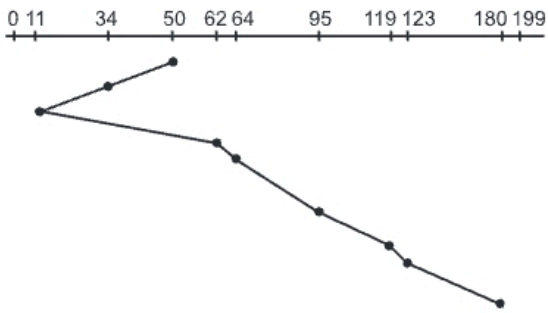


Fig. 6.7: LOOK Scheduling

Advantages:

- It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
- It provides better performance as compared to SCAN Algorithm.
- It does not lead to starvation.
- It provides low variance in response time and waiting time.

Disadvantages:

- There is an overhead of finding the end requests.
- It causes long waiting time for the cylinders just visited by the head. It doesn't provide uniform waiting time.

6.3.2.6 C-LOOK Scheduling

- C-LOOK is similar to C-SCAN disk scheduling algorithm.

- In C-LOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to necessary traversal to the end of the disk.
- In general, Circular versions are fairer but pay with a larger total seeks time. Scan versions have a larger total seek time than the corresponding Look versions.

Example:

- Consider a disk queue with requests for I/O to blocks on cylinders 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the cylinder 50 and the tail cylinder being at 199. Assume direction towards 0.

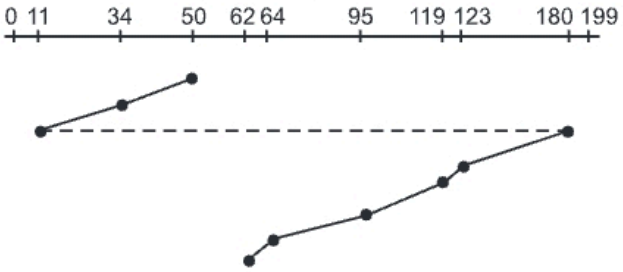


Fig. 6.8: C-Look Scheduling

Total head movement = (50 – 11) + (180 – 62) = 39 + 118 = 157 tracks

Advantages:

1. It does not cause the head to move till the ends of the disk when there are no requests to be serviced.
2. It reduces the waiting time for the cylinders just visited by the head.
3. It provides better performance as compared to LOOK Algorithm.
4. It does not lead to starvation.
5. It provides low variance in response time and waiting time.

Disadvantage:

1. There is an overhead of finding the end requests.

Performance of Scheduling Algorithms:

- When selecting a Disk Scheduling algorithm, performance depends on the number and types of requests. SSTF is common and has a natural appeal. SCAN gives better performance than FCFS and SSTF The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary. Either SCAN or C-LOOK is a reasonable choice for the default algorithm.

6.4 DISK MANAGEMENT

- The operating system is responsible for several aspects of disk management.

6.4.1 Disk Formatting

- A new magnetic disk is a blank slate. It is just platters of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting** (or **physical formatting).**

- **Low-level formatting** fills the disk with a special data structure for each sector. The data structure for a sector consists of a header, a data area, and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC).**
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps.
  1. The first step is to **partition** the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files.
  2. After partitioning, the second step is **logical formatting** (or creation of a file system). In this step, the operating system stores the initial file-system data structures onto the disk.

6.4.2 Boot Block

- When a computer is powered up or rebooted, it needs to have an initial program to run. This initial program is called the bootstrap program. It initializes all aspects of the system (i.e. from CPU registers to device controllers and the contents of main memory) and then starts the operating system.
- To do its job, the bootstrap program finds the operating system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating system execution.
- For most computers, the bootstrap is stored in read-only memory (ROM). This location is convenient because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And since ROM is read-only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips.
- For this reason, most systems store a tiny bootstrap loader program in the boot ROM, whose only job is to bring in a full bootstrap program from disk.
- The full bootstrap program can be changed easily. A new version is simply written onto the disk. The full bootstrap program is stored in a partition (at a fixed location on the disk) is called **the boot blocks**. A disk that has a boot partition is called **a boot disk or system disk.**

6.4.3 Bad Blocks

- Since disks have moving parts and small tolerances, they are susceptible to failure. Sometimes the failure is complete, and the disk needs to be replaced, and its contents restored from backup media to the new disk.
- More frequently, one or more sectors become defective. Most disks even come from the factory with bad blocks.

Methods to handle bad blocks:

- Depending on the disk and controller in use, these blocks are handled in a variety of ways.

Method 1: "Handled manually"

- If blocks go bad during normal operation, a special program must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: "sector sparing or forwarding"

- The controller maintains a list of bad blocks on the disk. The list is initialized during the low-level format at the factory and is updated over the life of the disk. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as **sector sparing** or **forwarding**.

Method 3: "sector slipping"

- In Sector slipping method, all sectors between the bad sector and the replacement sector are moved down by one, so that the linear progression of sector numbers can be maintained.
- For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.
- If the data on a bad block cannot be recovered, then a hard error has occurred. This error requires replacing the file(s) from backups, or rebuilding them from scratch.

Solved Problems

Problem 1: Consider Work Queue: 23, 89, 132, 42, 187 and schedule by algorithms:

1. FCFS (FIFO)
2. SSTF
3. SCAN
4. LOOK
5. C-SCAN
6. C-LOOK

There are 200 cylinders numbered from 0-199. The disk head starts at number 100.

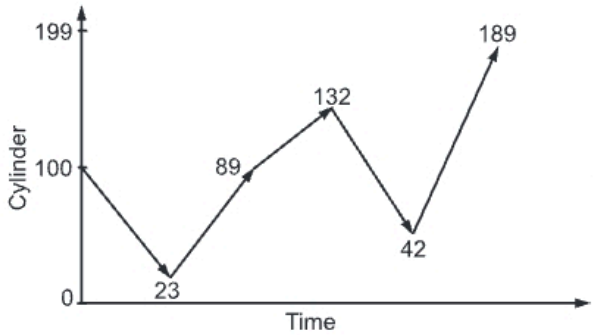


Fig. 6.9: FCFS scheduling

Solution:

1. **FCFS (FIFO):**

   Total time is estimated by total arm motion

   = (100 – 23) + (23 – 89) + (89 – 132) + (23 – 132) + (132 – 42) + (42 – 187)

   = 77 + 66 + 43 + 109 + 90 + 145

   = 530

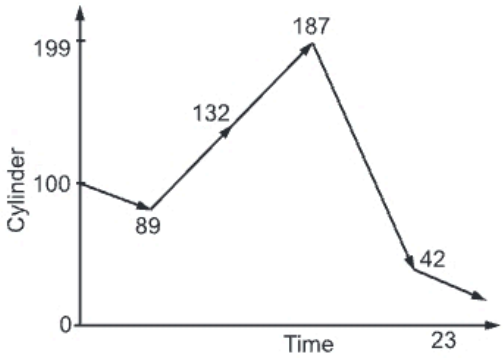


Fig. 6.10: SSTF scheduling

2. **SSTF:**

   = (100 – 89) + (89 – 132) + (132 – 187) + (187 – 42) + (42 – 23)

   = 11 + 43 + 55 + 145 + 19

   = 273

3. **SCAN:**

   Assume we are going inwards (i.e., towards 0).

   = (100 – 89) + (89 – 42) + (42 – 23) + (23 – 0) + + (0 – 132) + (132 – 187)

   = 11 + 47 + 19 + 23 + 132 + 55

   = 287

   Refer Fig. 6.11.

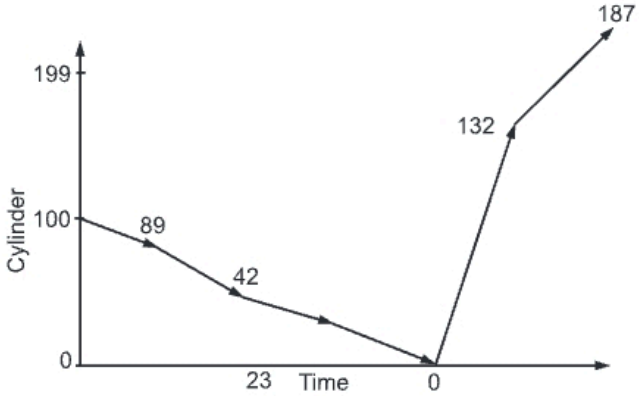


Fig. 6.11: SCAN scheduling

4. **LOOK:**

   = (100 – 89) + (89 – 42) + (42 – 23) + (23 – 132) + (132 – 187)

   = 11 + 47 + 19 + 109 + 55

   = 241

   Reduce variance compared to SCAN.Refer Fig. 6.12.

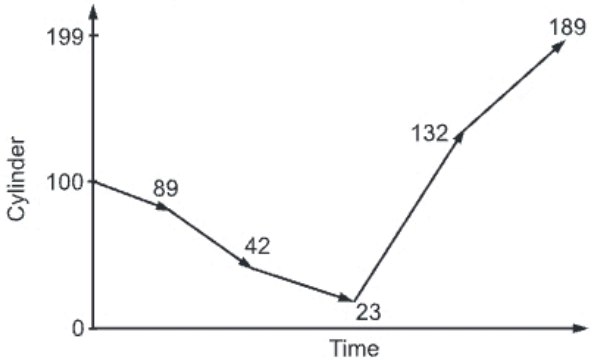


Fig. 6.12: LOOK scheduling

5. **C-SCAN:**
   = (100 – 89) + (89 – 42) + (42 – 23) + (23 – 0) + (0 – 199) + (199 – 187) + (187 – 132)
   = 11 + 47 + 19 + 23 + 199 + 12 + 55
   = 366
   Refer Fig. 6.13.

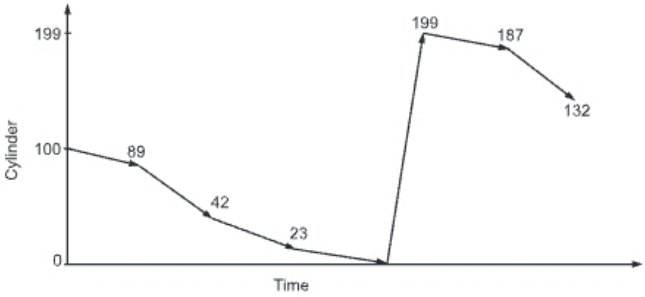


Fig. 6.13: C-SCAN scheduling

6. **C-LOOK:**
   = (100 – 89) + (89 – 42) + (42 – 23) + (23 – 187) + (187 – 132)
   = 11 + 47 + 19 + 164 + 55
   = 296
   Refer Fig. 6.14.

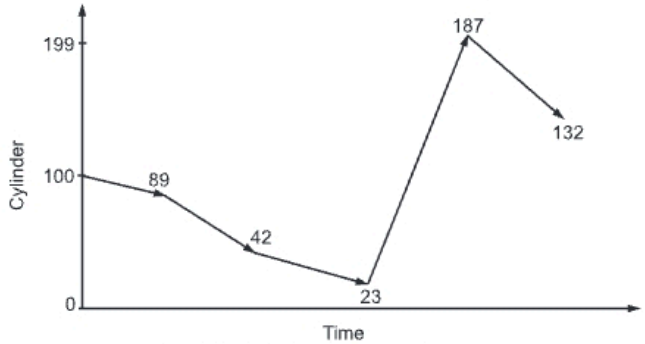


Fig. 6.14: C-LOOK scheduling

Problem 2: Consider a disk queue with requests for I/O to blocks on cylinders 82, 170, 43, 140, 24, 16, 190. The head is initially at cylinder number 50. Calculate total head movements incurred while servicing these requests using

1. FCFS (FIFO)
2. SSTF
3. SCAN
4. LOOK
5. C-SCAN
6. C-LOOK

Solution:

1. **FCFS:**

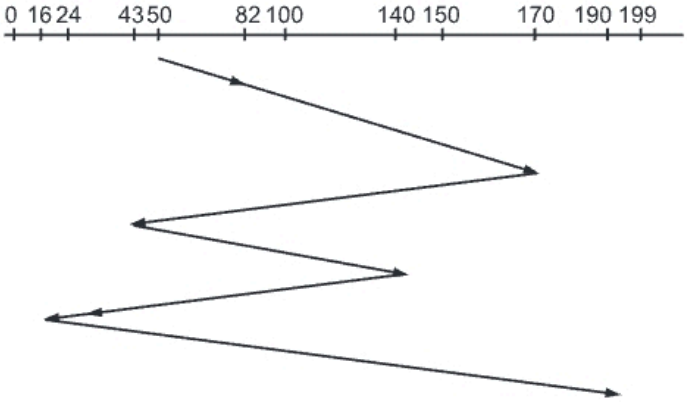


Fig. 6.15: FCFS scheduling

So, total seek time:

= (82 – 50) + (170 – 82) + (170 – 43) + (140 – 43) + (140 – 24) + (24 – 16) + (190 – 16)

= 642

2. **SSTF:**

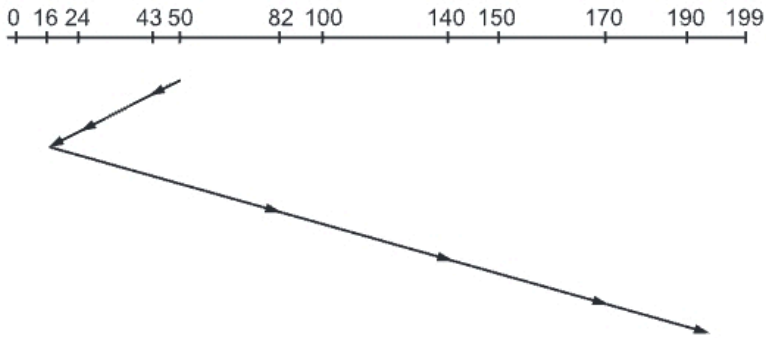


Fig. 6.16: SSTF scheduling

So, total seek time:

= (50 – 43) + (43 – 24) + (24 – 16) + (82 – 16) + (140 – 82) + (170 – 40) + (190 – 170)

= 208

3. **SCAN:** Suppose the requests to be addressed are: 82, 170, 43, 140, 24, 16, 190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

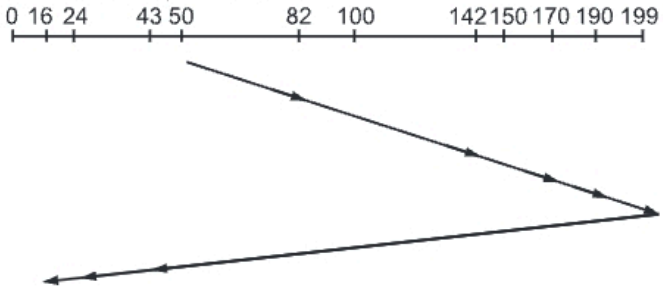


Fig. 6.17: SCAN scheduling

Therefore, the seek time is calculated as:

= (199 – 50) + (199 – 16) = 332

4. **C-SCAN**: Suppose the requests to be addressed are: 82, 170, 43, 140, 24, 16, 190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

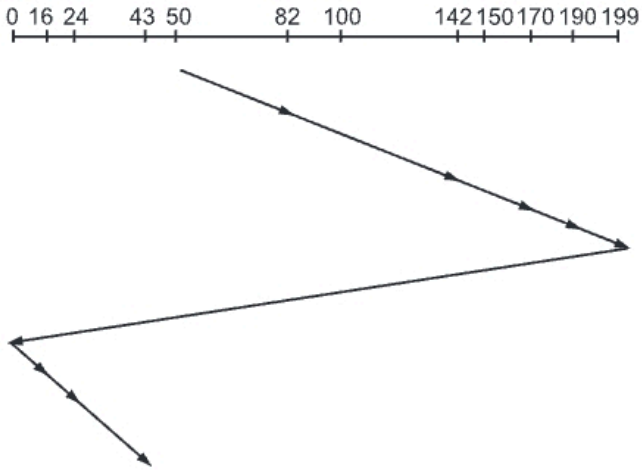


Fig. 6.18: C-SCAN scheduling

Seek time is calculated as:

= (199 – 50) + (199 – 0) + (43 – 0)

= 391

5. **LOOK:** Suppose the requests to be addressed are: 82, 170, 43, 140, 24, 16, 190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value".**

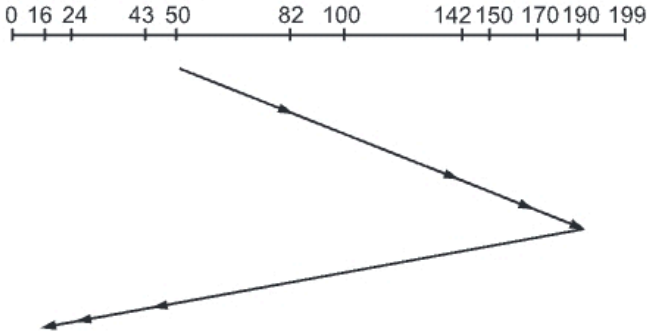


Fig. 6.19: LOOK scheduling

So, the seek time is calculated as:

= (190 – 50) + (190 – 16)

= 314

6. **C-LOOK:** Suppose the requests to be addressed are: 82, 170, 43, 140, 24, 16, 190. And the Read/Write arm is at 50, and it is also given that the disk arm should move **"towards the larger value"**.

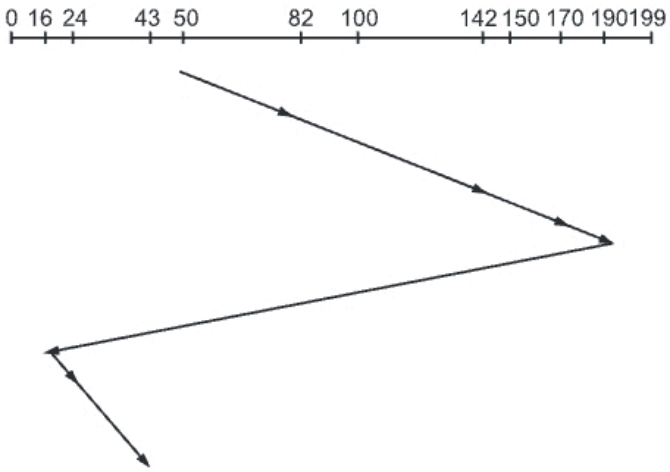


Fig. 6.20: C-LOOK scheduling

So, the seek time is calculated as:

= (190 – 50) + (190 – 16) + (43 – 16)

= 341

Problem 3: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The FCFS scheduling algorithm is used. The head is initially at cylinder number 53. The cylinders are numbered from 0 to 199. How many total head movement (in number of cylinders) incurred while servicing these requests?

Solution:

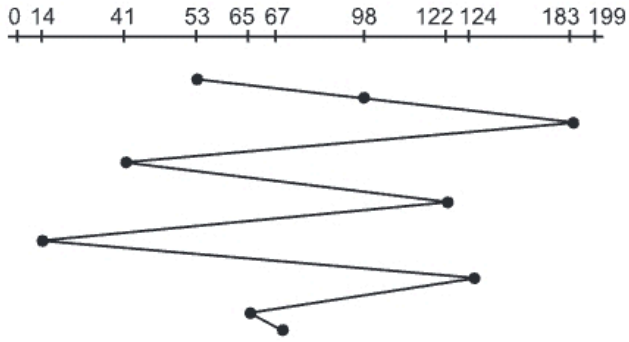


Fig. 6.21: FCFS scheduling

Total head movements incurred while servicing these requests are:

= (98 – 53) + (183 – 98) + (183 – 41) + (122 – 41) + (122 – 14) + (124 – 14) + (124 – 65) + (67 – 65)

= 45 + 85 + 142 + 81 + 108 + 110 + 59 + 2

= 632

Problem 4: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SSTF scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. How many total head movements (in number of cylinders) incurred while servicing these requests?

Solution:

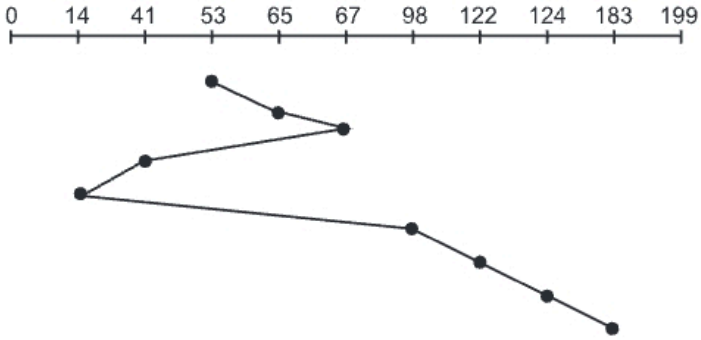


Fig. 6.22: SSTF scheduling

Total head movements incurred while servicing these requests are

= (65 − 53) + (67 − 65) + (67 − 41) + (41 − 14) + (98 − 14) + (122 − 98) + (124 − 122) + (183 − 124)

= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59

= 236

Problem 5: Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence: 4, 34, 10, 7, 19, 73, 2, 15, 6, 20.

Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1 ms to move from one cylinder to adjacent one and Shortest Seek Time First policy is used?

Solution:

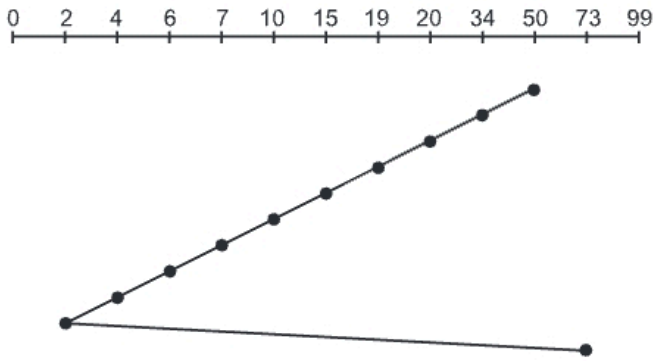


Fig. 6.23: SSTF scheduling

Total head movements incurred while servicing these requests

= (50 – 34) + (34 – 20) + (20 – 19) + (19 – 15) + (15 – 10) + (10 – 7) + (7 – 6) + (6 – 4) + (4 – 2) + (73 – 2)

= 16 + 14 + 1 + 4 + 5 + 3 + 1 + 2 + 2 + 71

= 119

Time taken for one head movement = 1 ms.

So, Time taken for 119 head movements

= 119 × 1 ms

= 119 ms

Problem 6: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. How many e total head movements (in number of cylinders) incurred while servicing these requests?

Solution:

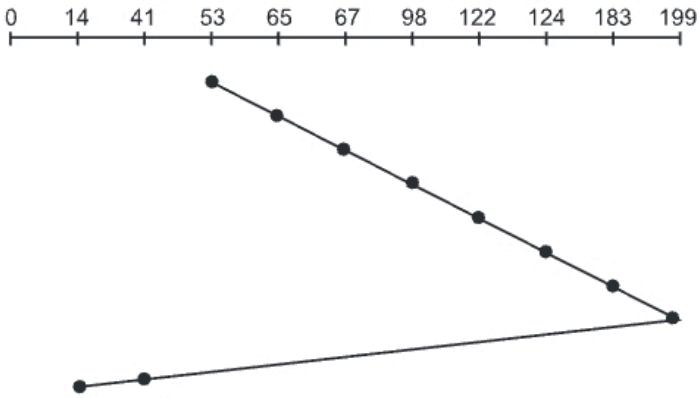


Fig. 6.24: SCAN scheduling

Total head movements incurred while servicing these requests are

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 41) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 158 + 27

= 331

Alternatively,

Total head movements incurred while servicing these requests

= (199 – 53) + (199 – 14)

= 146 + 185

= 331

Problem 7: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 1 99. How many total head movements (in number of cylinders) incurred while servicing these requests?

Solution:

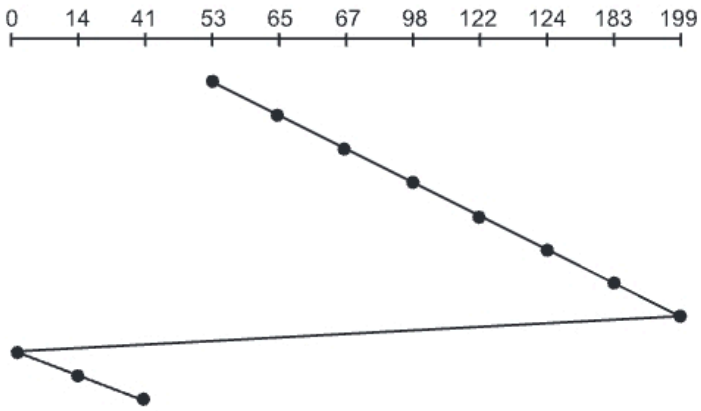


Fig. 6.25: C-SCAN scheduling

Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (199 – 183) + (199 – 0) + (14 – 0) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 27

= 386

Alternatively,

Total head movements incurred while servicing these requests

= (199 – 53) + (199 – 0) + (41 – 0)

= 146 + 199 + 41

= 386

Problem 8: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. How many total head movements (in number of cylinders) incurred while servicing these requests?

Solution:

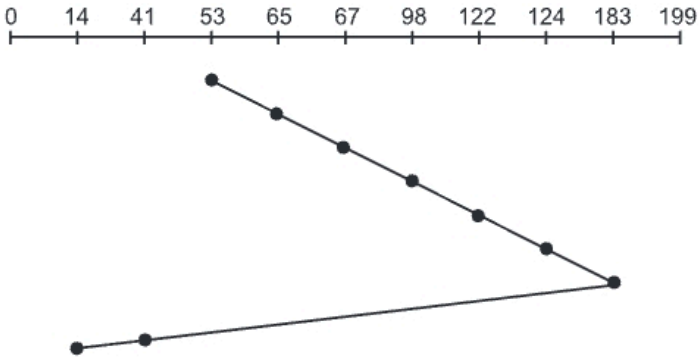


Fig. 6.26: LOOK scheduling

Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 41) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 142 + 27

= 299

Alternatively,

Total head movements incurred while servicing these requests

= (183 – 53) + (183 – 14)

= 130 + 169

= 299

Problem 9: Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. How many total head movements (in number of cylinders) incurred while servicing these requests?

Solution:

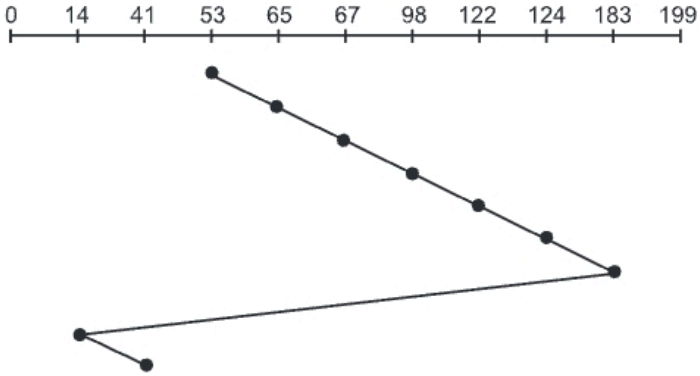


Fig. 6.27: C-LOOK scheduling

Total head movements incurred while servicing these requests

= (65 – 53) + (67 – 65) + (98 – 67) + (122 – 98) + (124 – 122) + (183 – 124) + (183 – 14) + (41 – 14)

= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 27

= 326

Alternatively,

Total head movements incurred while servicing these requests

= (183 – 53) + (183 – 14) + (41 – 14)

= 130 + 169 + 27

= 326

Problem 10: Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. How many total head movements (in number of cylinders) incurred while servicing these requests ?

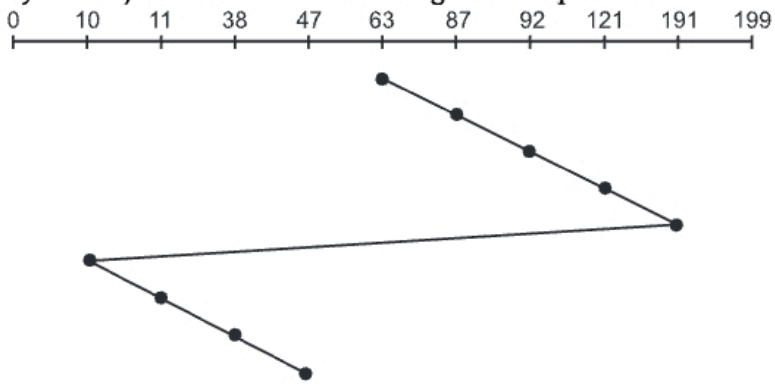


Fig. 6.28: C-LOOK scheduling

Solution: Total head movements incurred while servicing these requests

= (87 – 63) + (92 – 87) + (121 – 92) + (191 – 121) + (191 – 10) + (11 – 10) + (38 – 11) + (47 – 38)

= 24 + 5 + 29 + 70 + 181 + 1 + 27 + 9

= 346

Alternatively,

Total head movements incurred while servicing these requests

= (191 – 63) + (191 – 10) + (47 – 10)

= 128 + 181 + 37

= 346

Summary

- Disk scheduling is similar to process scheduling. The OS is responsible minimizing access time and maximizing bandwidth for disks.
- Disk Scheduling Algorithms are : FCFS, SSTF, SCAN, LOOK, C-SCAN, C-LOOK.
- The FCFS performs operations in order requested. No reordering of work queue since it processed disk requests according to its arrival. There is no starvation and all the requests are serviced but it doesn't provide fastest service.
- The Shortest Seek Time First (SSTF) selects the disk I/O request that requires the least movement of the disk access arm from its current position regardless of direction. It also reduces the seek time compared to FCFS but in this algorithm, I/O requests at the edges of the disk surface may get starved.
- The SCAN algorithm go from the outside to the inside servicing requests and then back from the outside to the inside servicing requests. It also reduces variance compared to SSTF.
- The Circular SCAN (C-SCAN) moves from one end of the disk to the other, servicing requests. When other end is reached, it immediately returns to the beginning of the disk, without servicing any requests. This algorithm treats the cylinders as a circular list that wraps around from the last cylinder to the first one. It also provides a more uniform wait time than SCAN.
- In LOOK scheduling algorithm, the arm goes only as far as the final request in each direction. The direction reverses immediately, without going all the way to the end of the disk.
- The Circular LOOK (C-LOOK) algorithm is similar to C-SCAN. The disk head also goes as far as the last request in its direction then reverses its direction immediately without first going all the way to the end of the disk.

Check Your Understanding

1. In _____ information is recorded magnetically on platters.
   (a) magnetic disks (b) electrical disks
   (c) assemblies (d) cylinders
2. The heads of the magnetic disk are attached to a _____ that moves all the heads as a unit.
   (a) spindle (b) disk arm
   (c) track (d) none of the mentioned
3. The set of tracks that are at one arm position make up a _____.
   (a) magnetic disks (b) electrical disks
   (c) assemblies (d) cylinders
4. The time taken to move the disk arm to the desired cylinder is called the _____.
   (a) Positioning Time (b) Random Access Time
   (c) Seek Time (d) Rotational Latency

5. The time taken for the desired sector to rotate to the disk head is called _____.
   (a) Positioning Time (b) Random Access Time
   (c) Seek Time (d) Rotational Latency
6. When the head damages the magnetic surface, it is known as _____.
   (a) Disk crash (b) Head crash
   (c) Magnetic damage (d) All of the mentioned
7. A floppy disk is designed to rotate _____ as compared to a hard disk drive.
   (a) faster (b) slower
   (c) at the same speed (d) None of the mentioned
8. The host controller is _____.
   (a) controller built at the end of each disk
   (b) controller at the computer end of the bus
   (c) controller built at the start of each disk
   (d) None of the mentioned
9. _____ controller sends the command placed into it via messages to the _____ controller.
   (a) host, host (b) disk, disk
   (c) host, disk (d) disk, host
10. The disk bandwidth is _____.
   (a) the total number of bytes transferred
   (b) total time between the first request for service and the completion on the last transfer
   (c) the total number of bytes transferred divided by the total time between the first request for service and the completion on the last transfer
   (d) None of the mentioned
11. SSTF algorithm, like SJF _____ of some requests.
   (a) may cause starvation (b) will cause starvation
   (c) does not cause starvation (d) causes aging
12. In the _____ algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests till the other end of the disk. At the other end, the direction is reversed and servicing continues.
   (a) LOOK (b) SCAN
   (c) C-SCAN (d) C-LOOK
13. In the _____ algorithm, the disk head moves from one end to the other servicing requests along the way. When the head reaches the other end, it immediately returns to the beginning of the disk without servicing any requests on the return trip.
   (a) LOOK (b) SCAN
   (c) C-SCAN (d) C-LOOK

14. In the _____ algorithm, the disk arm goes as far as the final request in each direction, and then reverses direction immediately without going to the end of the disk.
    (a) LOOK (b) SCAN
    (c) C-SCAN (d) C-LOOK
15. Consider a disk queue with requests for I/O to blocks on cylinders.
    98 183 37 122 14 124 65 67
    Considering SSTF (shortest seek time first) scheduling, the total number of head movements is, if the disk head is initially at 53 is?
    (a) 224 (b) 236
    (c) 245 (d) 240

Answers

1. (a)	2. (b)	3. (d)	4. (c)	5. (d)	6. (b)	7. (b)	8. (b)	9. (c)	10. (c)
11. (a)	12. (b)	13. (c)	14. (a)	15. (b)					

Practice Questions

Q.I Answer the following questions in short.

1. What is a disk scheduler?
2. What is host controller?
3. What are the different types of disk scheduling?
4. What is C-Scan scheduling?
5. What is C- look scheduling?

Q.II Answer the following questions.

1. Explain structure of disk along with diagram.
2. Explain the disk performance parameters.
3. Write advantages and disadvantages of FCFS, SSTF, SCAN and LOOK algorithms.
4. Describe the disk management in OS.
5. Explain SCAN disk scheduling algorithm with example.

Q.III Define the terms:

1. Access time
2. Rotational time
3. Seek time
4. Transfer rate
5. Disk controller
6. Response time

Previous Exam Questions

Winter 2018

1. Suppose the disk drive has 200 cylinders, numbered from 0 to 199. The current head position is 53, the queue of pending requests is: 98, 183, 37, 122, 14, 124, 65, 67.Calculate the total distance that the disk arm moves to satisfy all the pending requests for disk scheduling algorithm SCAN. Assume that the disk arm is moving towards 0. **[4 M]**

■■■