MCA11

KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Housefed Complex, Dispur, Guwahati - 781 006

Master of Computer Applications

OPERATING SYSTEM

CONTENTS

Subject Expert

Prof. Anjana Kakati Mahanta, Deptt. of Computer Science, Gauhati University
Prof. Jatindra Kr. Deka, Deptt. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati
Prof. Diganta Goswami, Deptt. of Computer Science and Engineering,
Indian Institute of Technology, Guwahati

Course Coordinator

Tapashi Kashyap Das, Assistant Professor, Computer Science, KKHSOU
Arabinda Saikia, Assistant Professor, Computer Science, KKHSOU

SLM Preparation Team

Units	Contributors
1, 2, 3, 4	**Jonalee Barman Kakoti** Lecturer, Deptt. of Business Administration NERIM, Guwahati, Assam
5, 8, 9	**Swapnanil Gogoi** Assistant Professor, IDOL, Gauhati University
6	**Khurshid Alam Barbora,** Assistant Professor, IDOL, Gauhati University
7,10,11	**Pranab Das** Assistant Professor, Don Bosco University, Guwahati, Assam

July 2012

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University.

The university acknowledges with thanks the financial support provided by the **Distance Education Council, New Delhi**, for the preparation of this study material.

Housefed Complex, Dispur, Guwahati- 781006; Web: www.kkhsou.net

COURSE INTRODUCTION

This is a course on **Operating System**. This course is designed as an introduction to the concepts and principles of operating system. An operating system (OS) is an interface between hardware and user; an operating system is responsible for the management and coordination of activities and the sharing of the resources of computer. This course will help you in gaining and understanding the basic concepts about OS.

Unit - 1 introduces basics of OS. This unit will help you to understand the evolution of OS along with its types.
Unit - 2 concentrates on the discussion about process including process models, process hierarchies and process states. In addition, threads are also discussed here.
Unit - 3 deals with the interprocess communication including the illustrations of race condition, critical section and mutual condition etc.
Unit - 4 concentrates on scheduling and illustrates the various scheduling algorithms.
Unit - 5 deals with the discussion of deadlock covering the deadlock avoidance techniques.
Unit - 6 explains the issues involved in the management of main memory and presents several memory management techniques which include virtual memory, paging, page replacement algorithms etc.
Unit - 7 deals with file system which cover the discussion of file, file types, file attributes, operation on files and directories etc. Different file accessing techniques like sequential and random access are also discussed in this unit.
Unit - 8 illustrates the principles of I/O management. Concept of DMA, interrupt handlers, device drivers are also discussed in this unit.
Unit - 9 concentrates on security and protection. Concept of cryptography is also covered in this unit.
Unit - 10 deals with the discussion of multiprocessor systems.
Unit - 11 concentrates on distributed operating system.

Each unit of this course includes some along-side boxes to help you know some of the difficult, unseen terms. Some “EXERCISES” have been included to help you apply your own thoughts. You may find some boxes marked with: “LET US KNOW”. These boxes will provide you with some additional interesting and relevant information. Again, you will get “CHECK YOUR PROGRESS” questions. These have been designed to make you self-check your progress of study. It will be helpful for you if you solve the problems put in these boxes immediately after you go through the sections of the units and then match your answers with “ ANSWERS TO CHECK YOUR PROGRESS” given at the end of each unit.

MASTER OF COMPUTER APPLICATIONS
Operating System
DETAILED SYLLABUS

Unit 1: Introduction to Operating System (Marks:10)
What is an operating system, batch system, mul;ti-programmed system, time-sharing system, personal computer operating system, parallel systems, distributed systems, real-time systems.

Unit 2: Processes (Marks:10)
Process(process models, process hierarchies, process states), Threads(what is thread and its use, design issues of thread).

Unit 3: Interprocess Communication (Marks:10)
What is interprocess communication, race conditions, critical-sections, mutual exclusion, solution to race condition, disabling interrupt, peterson's sloution, sleep & wake up (The Producer Consumer Problem) and Semaphores.

Unit 4: Scheduling (Marks:10)
Basic concepts, premitive and non-primitive scheduling, scheduling algorithms, types of scheduling - batch, interactive and real-time, goals of scheduling algorithms, first come first serve, shortest job first and round robin scheduling.

Unit 5: Deadlocks (Marks:10)
What is deadlock, principles of deadlock (deadlock conditions & modelling), deadlock detection, recovery & prevention, deadlock avoidance (banker's algorlithm)

Unit 6: Memory Management (Marks: 10)
Multiprogramming(with fixed partitions, relocation and protection).What is swapping and its basic concepts. Virtual Memory – Basic Concepts, Paging, Page Tables. Page replacement algorithms: - Optimal, Not Recently Used, First In First Out, Least Recently Used

Unit 7: File System (Marks: 10)
What is file, file naming, file types(directory, regular, device), sequential access and random access files, file attributes, operations on file, hierarchical directory structure, path name(relative and absolute), operation on directories. File System Implementation Techniques.

Unit 8: I/O Management (Marks:10)
Basic principles I/O Hardware, I/O Devices, Device controllers, DMA. Principles of I/O Software, its goals, Interrupt Handlers, Device Drivers, Device Independent I/O Software(its functions)

Unit 9: Security and Protection (Marks:8)
Security threats and goals, Authentication, Protection and Access control, Formal model of protection, Cryptography.

Unit 10: Multiprocessor Systems (Marks: 6)
Multiprocessor Interconnections, types of Multiprocessor Operating Systems, Multiprocessor OS Functions and Requirements, Multiprocessor Synchronization.

Unit 11: Distributed Operating Systems (Marks: 6)
Algorithms and Distributed Processing, Coping with Failures Models of Distributed systems, Remote procedure calls, distributed Shared Memory, Distributed File Systems.

UNIT1: INTRODUCTION TO OPERATING SYSTEM

UNIT STRUCTURE

1.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about operating system
- describe the types of operating systems

1.2 INTRODUCTION

Computer software can be divided into two types: system software and application software. The system software manages the operations of the computer itself and application programs are programs that are designed to help the user perform specific tasks. Operating system is the most vital component of the system software. An operating system is the system program that acts as an intermediary between the computer user and the computer hardware.

The purpose of the operating system is to provide an environment in which the user can execute programs in a convenient and efficient manner. The user interacts with the computer through the operating system in order to accomplish his task.

1.3 DEFINITION OF OPERATING SYSTEM

An operating system is the set of programs that manages the hardware and software resources of a computer. The operating system software lies nearest to the hardware and acts as the interface between the machine and the users of the computer. The user of a computer can interact with the system through the operating system without having to know the details of the hardware. It is the most important software in a computer without which the computer cannot run.

A computer system consists of hardware, system software, application software and users as shown in the figure below:

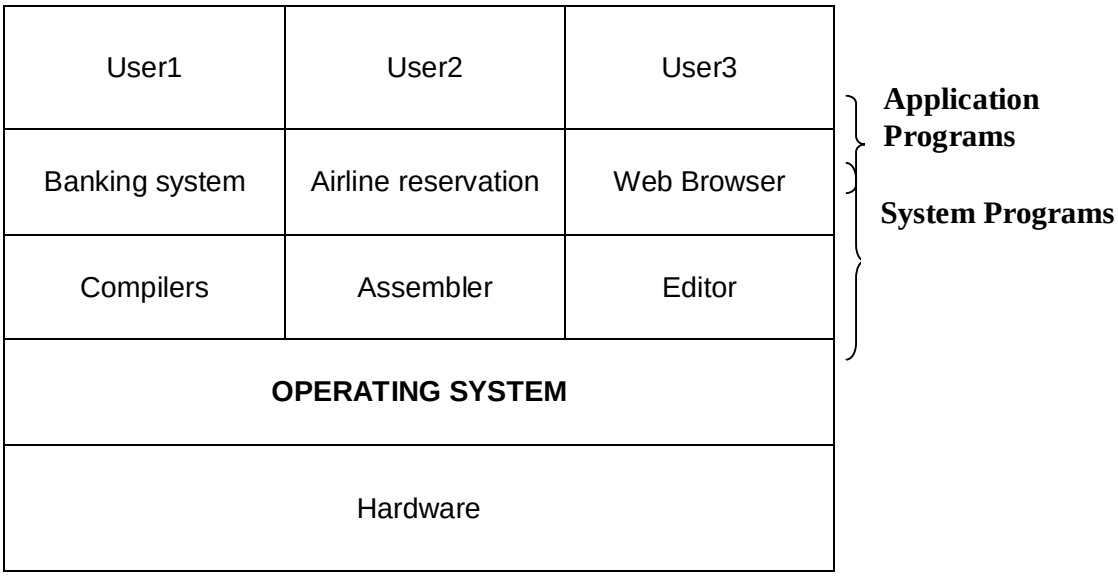


Figure 1: Abstract view of the components of a computer system

The operating system directly communicates with the hardware. It hides the complexity of the hardware and presents the programmers a nice and convenient

frame to work with. System Software like compiler, editors etc are on the top of the operating system and the application programs like word processor, spreadsheets are on the top of system programs.

The operating system performs varied functions in a computer system. These functions can be classified into the following three types:

1. The operating system acts as a resource manager: A computer has many resources ---- hardware and software such as CPU time, memory, input and output devices and files and so on. The operating system has to maintain a co-ordination among the peripherals. It manages all the available resources and allocates them to different users. The primary objective of an operating system is to make the computer system convenient to use and utilize the computer resources in an efficient manner. It keeps track of the status of each resource and decides who will have a control over the resources, for how long and when.
2. Establish a user interface. It provides the interface through which the computer users will interact with the computer system without the actual hardware knowledge of the system.
3. Execute and provide services to the application software. It provides the base for the application programs and services the programs.

Examples of operating systems: MS-DOS, LINUX, UNIX, WINDOWS, Android

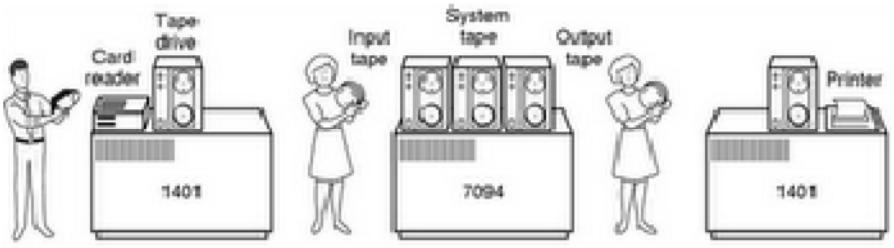
1.4 BATCH SYSTEM

In the early days, the computers were very large and kept in air conditioned rooms. Operators were there to run those computers.Whenever a programmer had to run a program, he used to first write down his program on a paper then punch it on the cards. The jobs in those days were given to the computer in the form of punch cards. Tape drives, card readers, printers were the available I/O devices at that time. The programmers then gave their cards to the computer operator in the input room.When the computer finished the task it was currently executing, the operator would take the printout of the output and keep it in the output room so that the respective programmer could take it. He would then go to

the input room to bring the next card and read it in the computer. Much of the computer time was wasted in this way in moving around the rooms.

A batch is a sequence of user jobs. The idea behind the batch system was to collect a tray full of jobs in the input room with similar requirements. The jobs are then read into a magnetic tape as a batch, using a small inexpensive computer. After that the magnetic tape was brought into the machine room and is mounted onto the tape drive of the machine. The output of each program was written on an output tape. The processing is carried out without any user intervention. The operating system automatically transferred the control from one job to the next.

When the whole batch finished processing, the output tape is removed and the input tape is replaced with the next batch of similar jobs. The operator takes the output tape, prints the outputs and keeps the outputs in the output room so that the appropriate programmers can take it. The whole process is shown in the figure below.



Batch processing operating systems are ideal in situations where:
- There are large amounts of data to be processed.
- Similar data needs to be processed.
- Similar processing is involved when executing the data.

Advantages of the batch systems were
- It allows sharing of computer resources among many users and programs.
- Improved system throughput and utilization.

In batch systems, the CPU is often idle as the speed of the I/O mechanical deviceswas much slower than the processor speed. The CPU had to wait for a card to be read or some output to be written on the magnetic tape for example.

Examples of batch processing systems include payroll systems and examination report card systems.

CHECK YOUR PROGRESS – 1

I. Fill in the blanks:

a. _____and_____ are the two types of computer software.

b. The _____ provides a platform on top of which the application programs can be build and run.

c. The Operating system acts as the interface between the _____ and the _____ the computer.

d. The operating system is a _____ software.

e. A _____ is a sequence of jobs.

f. The primary function of the _____ is to implement the jobs processing in a batch without user intervention.

g. Each job in the batch is _____ of other jobs in the batch.

h. Batch processing systems improve system _____ and _____.

i. In batch systems, the CPU is often idle due to the low speed of the _____.

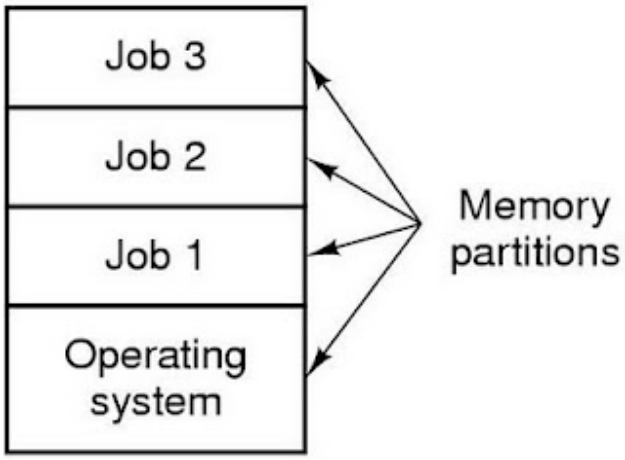
j. In the early days, _____ and _____ remained the primary input and output devices of a computer system respectively.

1.5 MULTIPROGRAMMED SYSTEMS

In batch systems, the CPU was often idle waiting for the I/O operations to finish thereby wasting the CPU time. One way of minimizing this inefficiency was to have a number of jobs available in the memory so that instead of sitting idle the CPU can execute instructions of one program while the I/O subsystem is busy with an I/O operation for another program or job.This is called multi-programming.

Multi-programming reduces the idle time of the CPU and increases the CPU utilization by organizing jobs so that the CPU always has a job to execute.

Some of the jobs that are ready to be executed by the CPU are kept in the memory from the ***job pool.***The job pool consists of all jobs that are on the disk ready to be executed and waiting to get the CPU time.



The operating system keeps several jobs in the memory as shown in the figure above. If many jobs are there in the job pool and the memory cannot hold all of them, then it is the task of the operating system to decide which jobs are to be brought into memory. This is called ***Job Scheduling***. When several jobs in the memory are ready to run at the same time, the operating system must chooses which job to run and the CPU time is given to that job. This is called ***CPU Scheduling***. The current job in the CPU would be executed until the job has to wait for some task, such as an I/O operation. The operating system then immediately switches to execute another job from the memory.When that job also needs to wait for some task, the CPU switches to another job and so on.

Eventually, when the first job finishes its task, for which it was waiting, it gets the CPU back. Multi-programming ensures that the CPU is always executing a job if there is a job to execute. At any moment, the CPU and the I/O subsystem are busy with different jobs. Thus both CPU and I/O subsystem access different areas of memory. This ensures that their activities would not interfere with one another. In principle the CPU and the I/O subsystem could operate on the same program in which case the program must explicitly synchronize their activities to ensure correctness.

Multi-programming thus increases the throughput of the system by efficiently using the CPU time. Throughput of a system is the number of programs processed by the system per unit time.

1.6 TIME-SHARING SYSTEMS

In multi-programming systems, the CPU is kept busy till there is at least one job in memory to be executed but it does not provide user interaction with the computer system. A user had no contact with his computation during its execution. Both the batch system and multi-programming systems provided poor user service.

Time sharing systems allow simultaneous access to a computer system by a number of independent users. Time-sharing is synonymous with multi-user. The CPU executes multiple jobs of the different users at the same time by switching among the jobs in relatively short intervals. The system switches from one user to the next so rapidly that the users are given the impression that the system is a single-user system, even though it is being shared by many users. The users in a time sharing system could provide inputs to a computation from a terminal and could also examine the output of the computation on the same terminal. The advent of time sharing provided ‘good’ response times to the computer users.

Time-sharing systems provided an interactive computing environment. The emphasis is given on good user response time rather than good utilization efficiency or throughput.

These systems are implemented using different scheduling algorithm techniques like, for example, round robin scheduling, time slicing algorithm etc.

1.7 PERSONAL COMPUTER OPERATING SYSTEM

A ***Personal computer*** is a general-purpose computer that is useful for the individuals. It may be a desktop computer or a laptop or a hand-held computer.PCs appeared in the 1970s. Earlier these PCs were neither multi-user nor multi-tasking. These systems aim for user convenience and responsiveness instead of maximizing CPU utilization. These systems run a version of Microsoft Windows operating system. MS-DOS operating system is also used.

Linux operating system recently has also gained much popularity. Mac OS X is the operating system that has been designed primarily for the Apple machines.

CHECK YOUR PROGRESS – 2

1. Say True or False:

a. If many user programs exist in the memory, the CPU can execute instructions of one program while the I/O subsystem is busy with an I/O operation of another program. This is multiprogramming.

b. The job pool consists of all the jobs on the disk that are ready to be executed by the CPU.

c. The task of the operating system to decide which jobs are to be brought into memory from the disk is called CPU Scheduling.

d. Multi-programming cannot ensure that the CPU is always executing a job if there is a job to execute in the memory.

e. The number of programs that can be processed by the system per unit time is called the throughput of the system.

f. There was almost no user interaction during the program execution in case of Batch systems and Multi-Programming systems.

g. Time-sharing systems allowed users to interact with computations during the program execution.
h. Time-sharing is synonymous with batch processing.
i. Time-sharing systems created an illusion that each user had a computer at his sole disposal.
j. The personal computer operating systems aim for maximizing CPU utilization.

1.8 PARALLEL SYSTEMS

Parallel systems are also known as multi-processor systems or tightly coupled systems. The processors share the computer bus, system clock and sometimes memory and other peripheral devices as shown in the figure below.

In parallel systems, the instructions of a program are divided among multiple processors with the objective of running the program in less amount of time.

The main advantages of a parallel system are as follows:

1. **Increased throughput**: More amount of work can be done in less amount of time.
2. **Economy of scale**: Sharing of the computer resources makes these systems more economic than using several single-processor systems.
3. **Increased reliability**: In these systems, if the functions are properly distributed among the many processors used then the failure of one

processor will not halt the system rather it will only slow down the system. The remaining processors will take a share of the work of that processor. This is called graceful degradation. The systems that are designed for graceful degradation are called fault-tolerant.

1.9 DISTRIBUTED SYSTEMS

A distributed system consists of a collection of autonomous computers, connected through a network. In these systems, the existence of the network is transparent to the users. The entire network appears like a uni-processor system to the user. A distributed operating system is a single operating system that manages resources on more than one computer system. The system enables the computers to co-ordinate their activities and to share the resources of the system so that the users perceive the system as a single system.

In a distributed system, there is sharing of information and services. The user would not know on which computer in the network the files are located, or in which machine in the network his program is running. The program may be running on a remote machine A and the files that the program uses might be in another remote machine B. it appears to the user that the program is running on his computer and the data and files is available locally. Distributed systems can allow vast numbers of users to access the same set of files reliably and efficiently. The possibility to add components to the system improves its availability, reliability, fault tolerance and performance.

Although distributed systems offer many advantages, they can be complex and difficult to implement and manage. These systems must be able to handle communication delays and reliability problems introduced by the network.

Examples of distributed systems are MIT's Chord operating system and the Amoeba operating system from the Vrije University (VU) in Amsterdam.

1.10 REAL-TIME SYSTEMS

The real time systems respond to events in real time. A real time system is a system which requires a timely response from a computer system so as to

prevent failures. The response is supplied within a certain bounded time period. A real time system can be used as a control device in a dedicated application. Sensors bring data to the computer. The computer analyses these data and adjusts the control to modify these data.

The real time systems have very tight time constraints. The processing must be done within the defined constraints. A real time operating system must guarantee this time requirement. A real time operating system thus helps to fulfill the worst – case response time requirements of an application. The worst-case response time of an application is the largest value of the response time for which it can still function correctly. The following facilities are provided by these systems for this purpose:

1. Multi-tasking within an application
2. Ability to define the priorities of tasks.
3. Deadline oriented scheduling
4. Programmer defined interrupts.

Process control systems, telecommunication systems etc are examples of applications that require the services of real time operating systems. A measurement from a petroleum refinery indicating that temperatures are too high might demand immediate attention to avert an explosion.

The telecommunication systems operate by transmitting bits on the communication lines. In order to receive an error-free message, every bit has to be received correctly.

In a flight simulation program, for example, the CRT display should change continuously as it would appear to the pilot in a real flight.

Real time systems can be **hard** and **soft real time** systems. The hard real time systems must meet the deadlines within the time specified.A hard real-time system must execute a set of concurrent real-time tasks in a way that all time-critical tasks meet their specified deadlines. Every task needs computational and data resources to complete the job. Missing deadline can result in a catastrophe. Examples include rocket propulsion control systems, flight response systems and nuclear reactors.

Soft real-time systems are used when missing a deadline occasionally is acceptable, such as when there is a limit on how long a user should wait to get a dial tone to place a phone call. That limit, if exceeded by few fractions of a second every once in a while, should pose no problem to the general availability of the feature.

CHECK YOUR PROGRESS – 3

1. Fill in the blanks:

i. Parallel systems are also known as _____ systems.

ii. The instructions of a program are divided among _____ in the parallel systems.

iii. Resources can be _____ in parallel systems.

iv. If a processor fails in a parallel system, then the work portion of that processor will be taken up by the rest of the processors. This is called _____.

v. A distributed system consists of a collection of autonomous_____, connected through a_____.

vi. A distributed operating system is a _____operating system that manages resources on more than one computer system.

vii. The real time systems respond to events in _____.

viii. The response in a real time system is supplied within a certain bounded _____.

ix. Real time systems can be _____ and _____ systems.

x. In a distributed system, there is sharing of _____ and _____.

1.11 LET US SUM UP

1. Computer software can be divided into two types: system software and application software.

2. The system software manages the operations of the computer itself and application programs are programs that are designed to help the user perform specific tasks.
3. An operating system is the system program that acts as an intermediary between the computer user and the computer hardware.
4. The operating system is the set of programs that manages the hardware and software resources of a computer system.
5. It hides the complexity of the hardware and presents the programmers a nice and convenient frame to work with.
6. The primary function of the Batch Processing System is to implement the processing of the jobs in a batch without requiring any intervention of the computer operator.
7. The transition from execution of one job to the next job in a batch system is automated.
8. In multiprogramming, the CPU instead of waiting for an I/O operation to finish can execute the instructions of another process in the main memory.
9. CPU utilization is increased with multiprogramming as it always executes a process, if there is any in the memory.
10. If many jobs are there in the job pool and the memory cannot hold all of them, then it is the task of the operating system to decide which jobs are to be brought into memory. This is called ***Job Scheduling.***
11. When several jobs in the memory are ready to run at the same time, the operating system must chooses which job to run and the CPU time is given to that job. This is called ***CPU Scheduling***.
12. Multi-programming thus increases the throughput of the system by efficiently using the CPU time.
13. In Time-sharing systems, multiple users can use the system simultaneously.
14. Time-sharing systems allow user interaction during the program execution.
15. A Personal computermay be a desktop computer or a laptop or a hand-held computer.

16. Parallel systems are also known as multi-processor systems or tightly coupled systems.
17. The instructions of a program are divided among multiple processors in parallel systems with the objective of running the program in less amount of time.
18. A distributed system consists of a collection of autonomous computers, connected through a network
19. The network is transparent to the users.
20. A distributed operating system is a single operating system that manages resources on more than one computer system.
21. Distributed systems can allow vast numbers of users to access the same set of files reliably and efficiently.
22. A real time system is a system which requires a timely response from a computer system so as to prevent failures.
23. The real time systems have very tight time constraints. The processing must be done within the defined constraints.
24. Real time systems can be hard and soft real time systems.

1.17 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS – 1

a. System software, application software
b. Operating system
c. Hardware, user
d. System
e. Batch
f. Batch processing system
g. Independent
h. Throughput, utilization
i. I/O mechanical devices
j. Card reader, printer

CHECK YOUR PROGRESS – 2

a. True
b. True
c. False
d. False
e. True
f. True
g. True
h. False
i. True
j. False

CHECK YOUR PROGRESS –3

i. tightly coupled
ii. multiple processors
iii. Shared
iv. Graceful degradation
v. computers , network
vi. Single
vii. real time
viii. time period
ix. Hard real time, soft real time
x. Information, services

1.18 FURTHER READINGS

1. *"*Systems Programming and Operating Systems*"*,*D M Dhamdhere*, Tata McGraw Hill.
2. "Operating System Design and Implementation", Andrew S Tanenbaum, Albert S Woodhull, Prentice-Hall India
3. "Operating System", H M Deitel, P J Deitel, D R Choffnes, Pearson Education

1.19 MODEL QUESTIONS

1. What is an operating system? Explain the functions of an operating system.
2. Explain the working of a batch system.
3. Explain the difference between multi-programming systems and Time-sharing systems.
4. What is a distributed operating system?
5. What is a real time system and what are the different types of real-time systems?

*****

UNIT 2:PROCESSES

UNIT STRUCTURE

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about processes
- describe different types of process models
- learn about various process states
- describe threads

2.2 INTRODUCTION

In the previous unit we have learnt the basic of operating system (OS) and also learn the different types of OS. The primary goal of an operating system is to make the computer convenient to use and also at the same time to efficiently use the available resources of the computer.

In this unit we will go through the detail about the role of “process” in the operating system. Execution of the user programs in a multiprogramming

computer system or a time-sharing computer system is represented by a set of processes.

A program written is a static entity. A process represents the execution of a program and is dynamic in nature. A process is also sequential in nature, that is, the instructions in the code are executed one after another. There may be many processes in the system, but at any moment, only one process will be actually executing the CPU. A process can be in a number of states at any instance of time. In this unit we are going to discuss some of the important topics about process.

2.3 PROCESS

A process is a program in execution. When a program is written and submitted to the operating system for execution, the program is converted into a process. A program is a passive entity whereas a process is an active entity. A process needs resources like CPU time, memory, files, I/O devices etc. to accomplish its task. These resources are generally given to the process at the time of its creation.

A process contains the following:

- An address space
- The program code, also known as the **text section**
- The data for the executing program
- The process stack
- The program counter (PC) which indicates the next instruction to be executed
- A set of general-purpose registers containing the current values.
- A set of operating system resources like open files, network connections etc.

Every process is named by a process number known as process ID or **PID** in short.

A single processor executes one or more instructions at a time, one after another. To allow the users to run several programs at once, the single processor computer systems can perform time sharing. Time sharing allows processes to switch between being executed and waiting to be executed. In most cases, this is

done very rapidly providing the illusion that several processes are executed ‘at once’. This is known as concurrency or multiprogramming.

2.4 PROCESS MODEL

In the process model, all the computer software is organized into sequential process or simply processes. A process is a program in execution. It includes the current value of the program counter, registers and variables.

In a multiprogramming environment it seems all the process are running parallel at the same time. In reality, this is not the case. The CPU can run at a time only one process. It switches from one process to another.

The Fig.2.1 (a) below shows a computer multiprogramming four programs in the memory. Fig.2.1 (b) shows the four programs running independently of each other. Each has their own program counter to keep track of which instruction to execute next. I the third figure, the graph of the processes over time is shown. It can be seen that the processes has made progress over the time but at any instance of time, only one process is actually running.

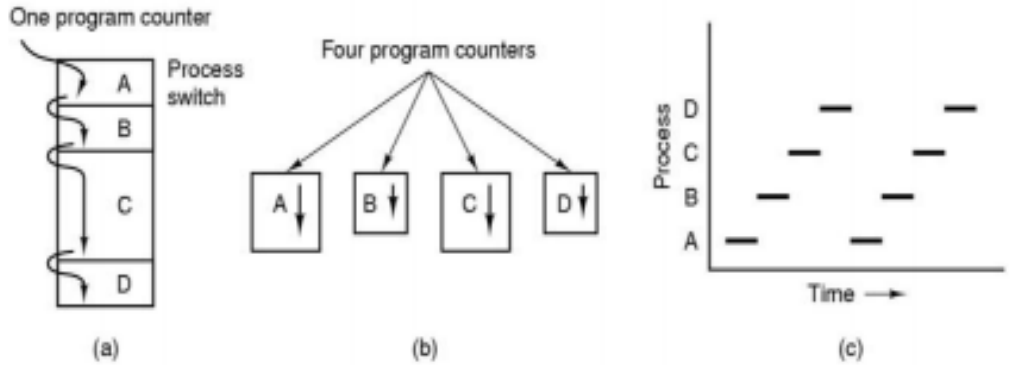


Fig.2.1

CHECK YOUR PROGRESS 1

1. Fill in the blanks:

a. A program when it is being executed is known as _____.

b. To indicate the next instruction in a program to be executed is given by _____.

c. The unique identifier of a process is the _____.

d. _____ allows processes to switch between being executed and waiting to be executed.

e. The process model organizes the software into _____.

f. In _____, the users are given the illusion that several processes are executed 'at once'.

g. A program is a _____ entity whereas a process is an _____ entity.

h. To accomplish its task, a process needs _____ like CPU time, memory, files, I/O devices.

i. The general-purpose registers store the _____ values of the program in execution.

j. At any instance of time, in a multi-programming environment, _____ number of program is actually running.

2.5 PROCESS HIERARCHIES

A process during its execution time may create several new other processes. Suppose Process A while execution created another process B. Then Process A is called the *Parent process* and Process B is called the *child process*. A parent process may have a number of child processes. Each of the new processes may create in turn other processes thereby forming a hierarchy of processes as shown in Fig.2.2.

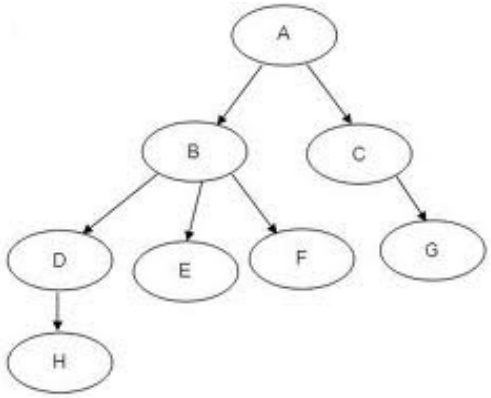


Fig.2.2: Process Hierarchy

The root process in the hierarchy of the processes is a special process created by the operating system during startup. In UNIX operating system, a process creates a new process by the ***fork*** system call. Each of the processes in UNIX has a unique identification number called process identifier or, in short, PID.

There are two different possibilities in terms of process execution concerning creating new processes:

a. The parent process continues to execute concurrently with its children,
b. The parent process waits until some or all of its children processes have terminated.

A process needs certain resources like CPU time, memory files etc. to accomplish its task. When a process creates child processes, either the parent and children processes share all resources or the children processes share only a subset of their parent's resources or the parent and children processes share no resources in common. If a child process is restricted to a subset of the parent resources, it may prevent any process from overloading the system by creating too many processes.

A parent process may terminate the execution of one of its children for the following reasons:

a. The child process has exceeded the usage of the resource that was allocated to it,
b. The task assigned to the child process is no longer required.

For example, in the figure of the process hierarchy above, if C issued a ***wait( )*** system call it would block until G finished. The wait command checks to see if the process G has been completed. When this command is executed no process is spawned.

2.6 PROCESS STATES

At any time in a multiprogramming environment, there are many processes in the system. Each of the process is in its own particular process state. Each of the processes may be in any of the following states:

1. New: When a process is created it is in this state. Here it awaits admission to the 'Ready' state. This admission will be approved by the scheduler.

2. Ready: A ready process is the process that has been loaded into the main memory and is awaiting execution on a CPU. There may be many ready processes at any one point of the system execution.

   A ready queue is used in CPU scheduling. As the CPU is capable of handling only one process at a time, the processes that are ready for the CPU are kept in a queue.

3. Running: The Running or Active process is a process which is currently executing on the CPU. From this state a process may
   i. Exceed its allocated time slice and be context switched out and send back to the Ready state, or
   ii. Indicate that it has finished and so send to the Terminated state, or
   iii. Block on some resource need for its execution (example, I/O resource)and so moved to the Waiting or Blocked state.

4. Waiting or Blocked: When a process is waiting for some event to occur such as an I/O operation, it will be removed from the CPU and will be in the Waiting state. It will remain there until the event for which it was waiting occurs, then after that it will return to the ready state.

5. Terminated: When a process has finished execution it enters this state.
   If a process is not removed from the memory after it has entered this state, this state may also be called as Zombie state.

The transition between the possible states of a process is shown in Fig.2.3:

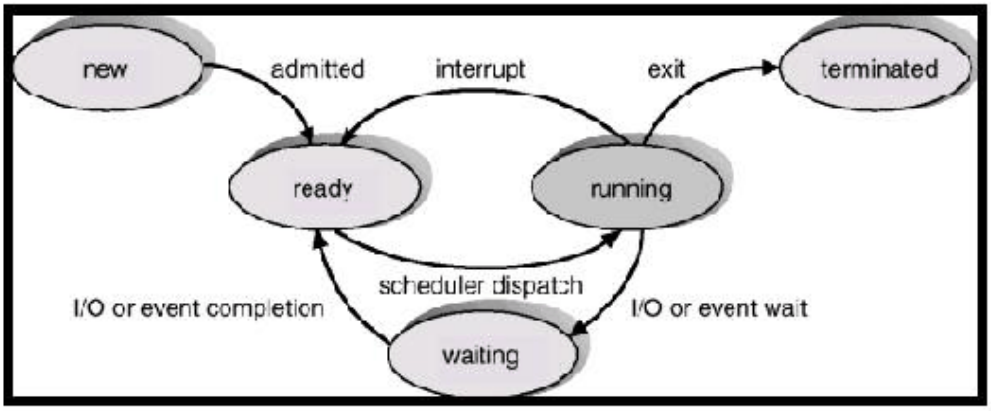


Fig.2.3

Apart from the above process states two additional states are available for processes in systems that support virtual memory. In both of these states, the processes are stored in the secondary memory, typically a hard disk.

1. Swapping Out and Ready (Suspended and Ready):
   A process may be swapped out that is removed from the main memory and placed in the virtual memory by the scheduler. From here it may be swapped back in to the Ready state, waiting to get the CPU time.
2. Swapping out and Waiting (Suspended and Waiting):
   Here a process may be swapped out from the main memory to the virtual memory and be waiting.

CHECK YOUR PROGRESS -2

1. Say 'True' or 'False'

i. A process can create other processes.

ii. A parent process may have only one child process.

iii. The exec() system call is used to create a new process.

iv. If the task assigned to the child process is no longer required, then the parent process can terminate the child process.

v. When the wait() command is executed no process is spawned.

vi. A ready process is the process that has been loaded into the main memory and is awaiting execution on a CPU.

vii. CPU scheduling uses Stack data structure to hold the ready processes.

viii. When a process exceeds its allocated time slice it gets terminated.

ix. A terminated process if not removed from the main memory is said to be in the Zombie state.

x. If a process during execution needs to wait for some I/O operation, it goes to the ready state.

2.7 PROCESS CONTROL BLOCK

The operating system maintains a table, an array of structures, called the process table or the **Process Control Block** (PCB). The process control block contains all the information about a process. It is also sometimes known as the Task Control Block. When a process is created the operating system also creates a process control block to maintain all the information about that process.

The following are some of the information that is stored in a process control block:

1. Process state: The state may be new, ready, running, waiting and so on.
2. Process number (PID)

3. Program counter (PC): It indicates the address of the next instruction in the process to be executed.
4. CPU Registers: They may vary in their number and type depending on the computer architecture. They include accumulators, index registers, stack pointers and other general-purpose registers.
5. Memory management information: This information may include the page tables, the segment tables and other memory related information depending on the memory system used in the computer system.
6. I/O status information: This includes the list of all the I/O devices that are allocated to the particular process, the list of the open files, etc.
7. Processor scheduling information: This includes the process priority and other scheduling information.
8. Accounting information: This information may include the amount of the CPU time used, the time limits and so on.

A Process Control Block is shown in the Fig.2.3:

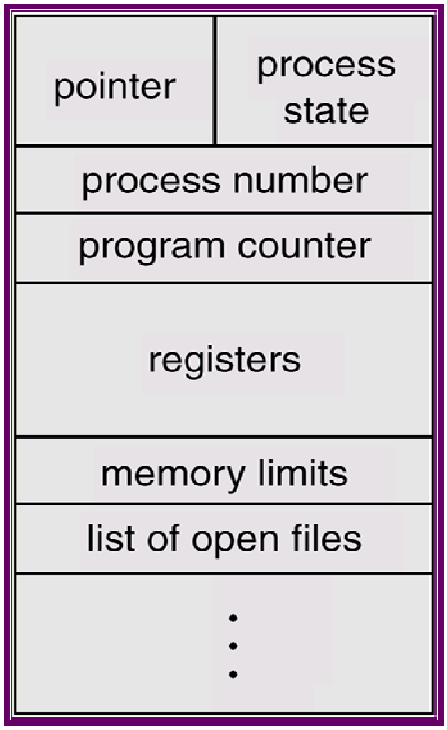


Fig.2.3 : A Process Control Block

2.8 THREADS AND ITS USE

The concept of a process and a thread is a different one, although a thread must execute in a process. From time to time, a process may be allocated different resources according to its requirement, that is, processes are used to group the resources together. Threads are the entities that are scheduled for execution on the CPU. Threads are similar to processes as they have a beginning, an end, a sequence as processes. However, a thread is not a process. It cannot run on its own but runs within a process. *A thread is a single sequential flow of control within a program.*Threads is also known as *lightweight processes* as they have some of the process properties. It is the smallest unit of processing that is scheduled by a scheduler in an operating system.

A traditional process has one thread in it. Multithreading refers to the ability of an operating system to support multiple threads of execution within a single process. a multithreaded program allows it to operate faster on computer systems that have multiple CPUs.

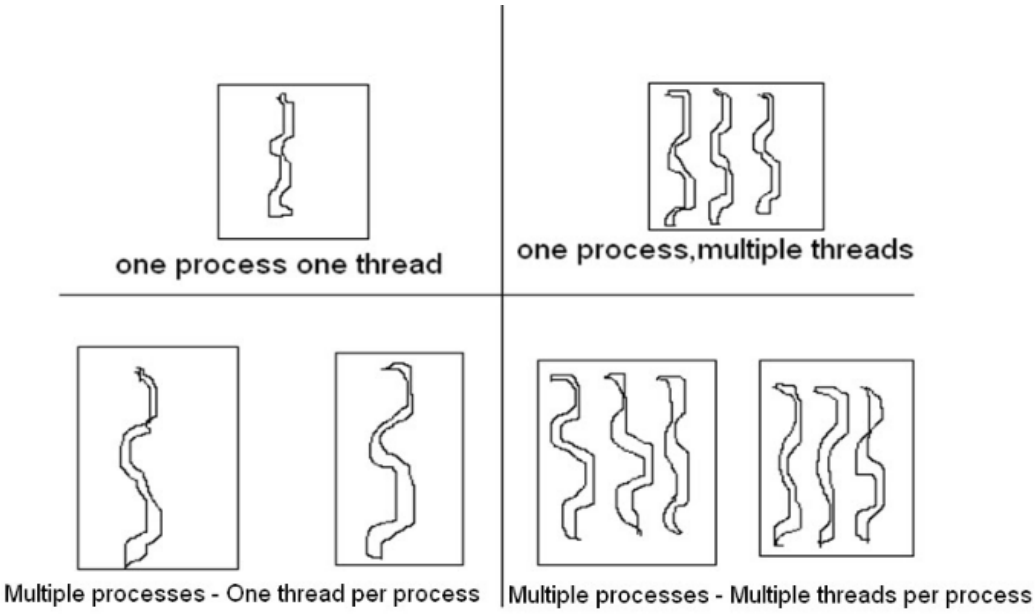


Fig.2.4: **Processes and Threads**

The figure above shows four different types of operating systems, which supports threads in different ways. The two arrangements shown in the left-half, of the figure above, are single-threaded approaches. The right-half in the figure shows the multithreaded approach.MS-DOS supports "a single user process and a single thread". UNIX, support "multiple user processes" but only support "one thread per process". A Java run-time environment is an example of a system of "one process with multiple threads". Windows supports "multiple processes - where each process has multiple threads".

Threads provide a way of allowing applications to maximize its usage of CPU resources in a system, usually in a multi-processor system. The key benefits of threads derive from the performance implications:

- It takes far less time to create a new thread in an existing process than to create a brand - new process.
- It takes less time to terminate a thread than a process.
- It takes less time to switch between two threads within the same process.

2.9 DESIGN ISSUES OF THREAD

A thread like a process also has a program counter, a register set, stack space etc. The program counter is required as threads, like processes, can be suspended and resumed. The registers are needed because when threads are suspended, their registers must be saved. Threads can also be in different states, running, ready or blocked state. All threads of a process share the state and resources of that process. They reside in the same address space and share the same data. When any data item is changed in the memory, the other threads can see the change when they access those data items. However, threads have their own stacks and their own CPU states.

The operating system treats each thread like a process. But there is a difference – it does not store the execution environment while switching between threads of

a process. Thus the resource state remains with the process while the execution state is associated with the thread of the process.

There are some similarity between processes and threads:

1. Like processes, threads share CPU and only one thread can be active (running) at a time.
2. Like processes, threads within processes execute sequentially.
3. Like processes, thread can also create children.
4. And like process, if one thread is blocked, another thread can run.

There are also differences between them. Unlike processes, threads are not independent of each other. Also threads are designed to assist one another, unlike processes. Processes may or may not assist one another as they may be created by different users.

There are basically two different approaches to the implementation of threads. They are as follows:

a. Kernel level Threads
b. User level Threads

A kernel level thread is created and processed by the kernel. These threads can make system calls to the operating system for their resource requirements. The thread that makes the system call can block for the completion of an I/O operation or grant of a resource. The disadvantage of such threads is that their state is smaller and also all switching is performed by the kernel.

In case of user level threads, a process manages its own threads. They are not known to the kernel. A user level thread is usually created by a threading library and scheduling is managed by the threading library itself. All user level threads belong to process that created them.

CHECK YOUR PROGRESS 3

Fill up the blanks:

a. All the information about a process is kept in a table called the _____. Process Control Block
b. The Process Control Block is created when a process is _____ by the operating system call. Created
c. The contents of the registers used by a process during execution must be saved in the _____. PCB
d. _____ are the entities that are scheduled for execution on the CPU. Threads
e. Threads are also known as _____. lightweight processes
f. Scheduling of the processes is done by a _____ in an operating system. Scheduler
g. _____ refers to the ability of an operating system to support multiple threads of execution within a single process. Multithreading
h. The MS-DOS operating system supports a single user process and _____ thread. single
i. Unlike processes, threads are not _____ of each other. Independent
j. Threads within the processes execute _____. sequentially

2.10 LET US SUM UP

1. A process is a program in execution.
2. A process needs resources like CPU time, memory, files, I/O devices etc to accomplish its task.
3. A process is identified in the system by a Process Identifier (PID).

4. To allow the users to run several programs at once, the single processor computer systems can perform time sharing.
5. Time sharing allows processes to switch between being executed and waiting to be executed.
6. In process model all the computer software is organized into sequential process or simply processes.
7. In a multiprogramming environment although it seems that all the process are running simultaneously, actually the CPU switches very rapidly from one process to another. At a time, only one process can have the CPU time.
8. A process during its execution time may create several new other processes.
9. The creating process is called the parent process and the created process is called the Child process.
10. . Each of the new processes may create in turn may create other processes thereby forming a process hierarchy.
11. In UNIX operating system, a process creates a new process by the ***fork*** system call.
12. A process needs certain resources like CPU time, memory files etc. to accomplish its task.
13. The states in which a process may be in are New, Ready, Running, Waiting and Terminated.
14. The Queue data structure is used for CPU scheduling.
15. If a process is not removed from the memory after it has entered the Terminated state, then that state may also be called as Zombie state.
16. Apart from the above mentioned process states two additional states are available for processes in systems that support virtual memory. They are Swapping Out and Ready (Suspended and Ready) and Swapping out and Waiting (Suspended and Waiting).
17. The operating system maintains a structure called the Process Control Block that contains all the information about a process.
18. The Program Counter indicates the address of the next instruction in the process to be executed.
19. A thread is a single sequential flow of control within a program.

20. Threads are also known as *lightweight processes* as they have some of the process properties.
21. A traditional process has one thread in it.
22. Multithreading refers to the ability of an operating system to support multiple threads of execution within a single process.
23. A thread like a process also has a program counter, a register set, stack space etc.
24. Like processes, Threads can also be in different states --- running, ready or blocked state.

2.11 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS 1

1.

a. Process
b. program counter
c. Process Identifier (PID)
d. Time sharing
e. Processes
f. multi-programming
g. passive, active
h. resources
i. current
j. one

CHECK YOUR PROGRESS 2

2. i. True
   ii. False
   iii. False
   iv. True
   v. True

vi. True
vii. False
viii. False
ix. True
x. False

CHECK YOUR PROGRESS 3

1. a. Process Control Block
   b. Created
   c. Process control Block
   d. threads
   e. lightweight processes
   f. Scheduler
   g. Multithreading
   h. single
   i. Independent
   j. sequentially

2.12 FURTHER READINGS

1. "Operating system concepts", Abraham Silberschatz, Peter Baer Galvin, Wiley
2. "Systems Programming and Operating Systems", D M Dhamdhere, Tata McGraw Hill
3. "Modern Operating Systems", A S Tanenbaum, Prentice-Hall

2.13 MODEL QUESTIONS

1. What are processes and what are the different states of a process?
2. What are threads and explain the differences between threads and multi-threads?
3. What is the purpose of a Process Control Block?

*****

UNIT 3: INTERPROCESS COMMUNICATION

UNIT STRUCTURE

3.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about interprocess communication
- get the concept of race conditions
- learn about mutual exclusion
- know different ways of achieving mutual exclusion
- learn the concept of semaphores

3.2 INTRODUCTION

In the previous unit we have already studied about the process concept. Processes can also communicate among themselves and this is called InterProcess Communication (IPC). In this unit we will discuss about interprocess communication and race conditions which occurs due to IPC. Also here we will discuss about the critical region and how to achieve mutual exclusion among the processes that are communicating with another.

3.3INTERPROCESS COMMUNICATION

The processes in a system frequently communicate with one another to exchange information. The method of communication between two or more processes is known as Interprocess Communication (IPC).The information exchange involves exchanging data items among the processes. The producers of the data items are called *senders* and the consumers of those data items are called *receivers*. The operating system assists the processes in setting up the communication among them. Most operating systems support some communication primitives and the processes execute these primitives to exchange information.

Interprocess communication may be direct or indirect. In indirect communication, processes make system callswhereas in direct communication, processes directly communicate by reading and modifying values in the common physical memory locations. As these memory locations ar mapped to the user space the processes do not need to make system calls to access the memory contents.

The interprocess communication, direct or indirect, may be modeled as schematics shown in Fig.3.1:

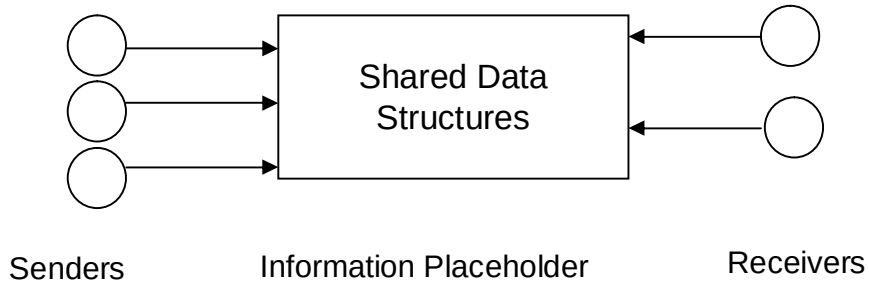


Fig.3.1:A typical interprocess communication facility

The information is structured into shared data or shared variables. Each shared variable has a unique name, address and atype. The variables can be accessed by the communication primitives. Processes communicate among themselves by manipulating the values of the shared variables andby executing operations supported by these variables.

Most operating systems also implement different synchronization schemes for the process coordination purposes.

3.4 RACE CONDITIONS

In most operating systems, when the processes are working together they may share some resources like storage that each of them can read and write. The shared storage may be a common file or somewhere in the main memory.

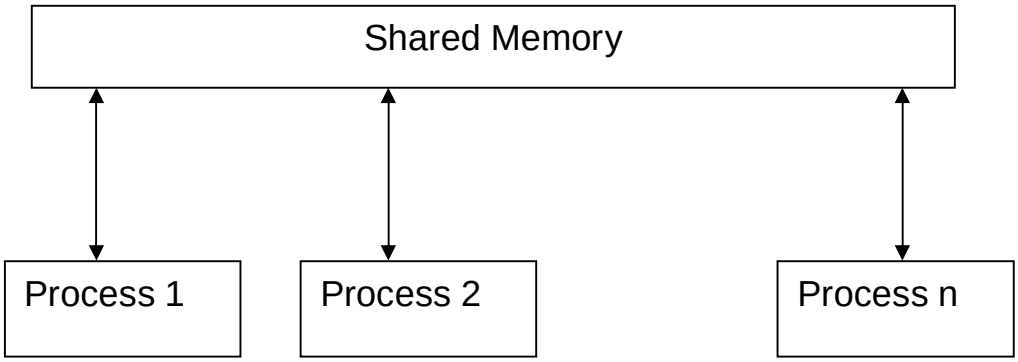


Fig.3.2

Now, let us take an example to show how the interprocess communication takes place. Let us consider the print spooler. When a process wants to print a file, the filename is entered in the spooler directory. A process called the printer daemon keeps on checking the spooler directory to find the filenames which are to be printed. If it finds any filename, it prints the file and removes the name of that file from the directory.

Suppose the spooler directory has a number of slots numbered as 0, 1, 2, ……..and each of the slot can hold a filename. Suppose also that there are two shared variable named as ***in*** and ***out***. The *out* variable points to the next file to be printed and the *in*variable points to the next free slot in the directory.

At a time suppose that the slots from 0 to 3 are empty, that is, the filenames in those slots are already printed. Slots from 4 to 6 are occupied by some filenames. Now suppose that both process A and process B wants to keep a file in the queue for printing. The situation is as shown in Fig.3.3 below:

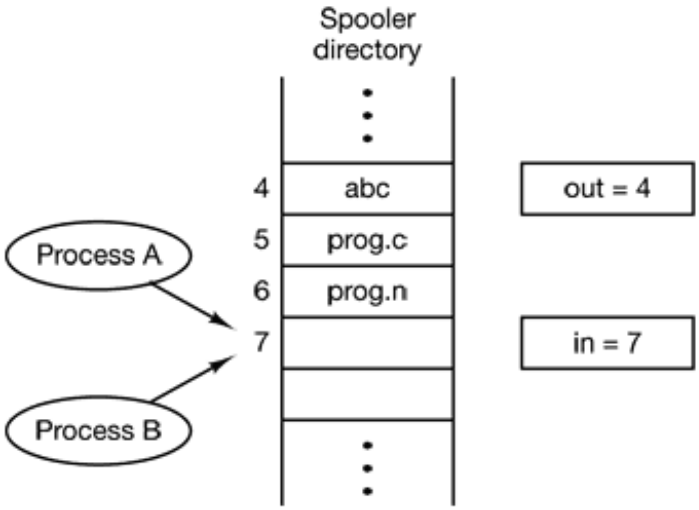


Fig.3.3

Process A reads the variable *in* and finds slot 7 to be free. It stores the value 7 in a local variable, say, next_free_slot. At that moment an interrupt occurs and the

operating system decides to run process B. As process B also wants to queue a file to print, it reads the variable in and finds 7 in it. It then stores the filename in slot number 7 and updated *in* to 8.

Eventually process A is starts execution again from the point it was stopped. It finds 7 in its variable next_free_slot and so stores its filename in slot number 7 thereby erasing the filename put there by process B. It then computes next_free_slot + 1, which is 8 and sets *in* to 8. The process B will never get any output.

Situations like this, where two or more processes are sharing some shared resources and the final result depends on who runs precisely when is called ***race condition***. In other words, a race condition occurs when the result of an operation depends on the ordering of when individual processes are run.

3.5 CRITICAL-SECTIONS

A critical section or critical region is a block of code where the shared resources (memory, files) are accessed. We have discussed about race conditions in the above section (Section 3.4). The race conditions should be recognized in the system and be avoided. To avoid race conditions, no two processes should be allowed to execute their critical sections at the same time. This desired state is called ***Mutual Exclusion***. Mutual Exclusion ensures that if one process is using a shared variable or file then all other processes should be excluded from doing the same thing. By managing the critical sections it can be ensured that the code in a critical section appears atomic with respect to the other processes in the system.

This can avoid the race conditions but this is not sufficient for having parallel processes cooperate correctly and efficiently. There are four conditions that must hold to have a good solution.

1. No two processes may be simultaneously inside their critical sections.
2. No assumptions may be made about the speeds or the number of CPUs.

3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region.

If the above conditions hold, the race conditions will be avoided.

CHECK YOUR PROGRESS 1

1. Fill in the blanks:

a. Method of communication between two or more processes in a system is known as _____.
b. The information exchange involves exchanging _____ among the processes.
c. In_____ , processes make system calls to access the values of the shared variables.
d. When a process wants to print a file, the filename is entered in the _____.
e. The _____ process keeps on checking the spooler directory to find filenames which are to be printed.
f. When two or more processes are sharing some shared resources and the final result depends on who runs precisely when called_____ is.
g. A _____ is a block of code where the shared resources (memory, files) are accessed. critical section
h. If process P_i is executing in its critical section, then no other processes can be executing in their critical sections. This is - _____.
i. No _____ may be made about the speeds or the number of CPUs.
j. Interprocess communication may be _____or_____ communication.

3.6 MUTUAL EXCLUSION

If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

- ***Implications:***
  - Critical sections should be focused and short.
  - The process should not get into an infinite loop in the critical region.
  - If a process somehow halts/waits in its critical section, it must not interfere with other processes.

3.7 SOLUTION TO RACE CONDITION

To Solve the Race Condition problem when a number of processes are competing to use a resource, only one process must be allowed to access the shared resource at a time. In other words two or more processes are prohibited from simultaneously or concurrently accessing a shared resource.

The critical section problem is how to ensure that at most one process is executing its critical section at a given time. Formally, the following requirements should be satisfied.

- Mutual Exclusion,
- Progress: if no process is in its critical region and there exist some other process that wants to enter its critical region, then one of those waiting processes should be allowed to enter.
- Bounded Waiting: there must be a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter the critical section and before that request is granted.

Informally, three important cases should be considered.

1. A process wants to enter its critical section and the other process is not in its critical section.

2. A process wants to enter its critical section and the other process is already in its critical section.
3. Both the processes want to enter their critical sections.

3.8 DISABLING INTERRUPT

The simplest mechanism to achieve mutual exclusion is to disable the interrupts in the system during the time a process is in its critical section. A process just after entering the critical section will disable all the interrupts and enables the interrupts immediately after exiting the critical section. The CPU switches from process to process as a result of the clock interrupt in the system. With interrupt disabled, the CPU will not switch among processes. Mutual exclusions is achievedbecause a process is not interrupted during the execution of its critical section and thus excludes all otherprocesses from entering their critical section.

This is an effective solution but it might lead to different problems. If the user processes are given the power to disable the interrupts, then sometime it may happen that after finishing the work in the critical section the user process might never turn on the interrupts again. In that case it could be the end of the system.

Moreover this solution will not work in a multi-processor system. Disabling interrupt affects only the CPU that will execute the disable instruction.

3.9 PETERSON'S SOLUTION

In 1981, G L Peterson discovered a simpler method to achieve mutual exclusion for two processes. The Peterson's algorithm is shown in the figure below. It is based on busy waiting. It is a software approach and does not require any special hardware.

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 – process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

This algorithm consists of two procedures written in ANSI C language. The procedures are *enter_region()* and *leave_region()*. The method requires two data itemsto be shared between two processes:

int turn

int interested[N], where N=2

- The variable turn indicates whose turn it is to enter the critical section.
- The interested array is used to indicate if a process is ready to enter the critical section.
    - Interested[i] = True implies that process $P_i$is ready.

Before entering its critical region (that is, using the shared variables)each of the process calls the *enter_region* procedure with its own process number, which may be 0 or 1, as the parameter. This call may cause the process to wait until it is safe to enter the critical region if another process is already inside it. The processcalls the *leave_region* procedure when finished with the shared resources to indicate that it is done and allow the other process to enter the critical region, if it wants to.

Let us see how this solution works. Initially no process is in its critical region. Process 0 then calls the *enter_region*procedure. It indicates that it is ready to enter the critical region by setting the array element *interested[0]* as True and set *turn* to 0. As process 1 is not interested to enter the critical section so Process 0 immediately enters. If process 1 now calls the *enter_region*it will have to wait until Process 0 leaves the region by calling the *leave_region* procedure and sets *interested[0]* to False.

Suppose that both the processes calls the *enter_region*almost simultaneously. Both will store their process numbers in the variable *turn.* Suppose process 1 stores last, so the value of *turn* will be 1. When both the processes execute the while statement, Process 0 will enter the critical region and Process 1 loops there and does not enter the region.

CHECK YOUR PROGRESS 2

1. Say True or False

a. If a process is executing inside its critical section, then other processes should also be allowed to enter their critical sections.
b. The process should not get into an infinite loop in the critical region.
c. Two or more processes are prohibited from simultaneously or concurrently accessing a shared resource.
d. The critical section problem is the problem of how to execute two processes in its critical section at a given time.
e. With interrupts disabled the CPU cannot switch from one process to another.
f. Disabling interrupt solution is best for multi-processor systems.
g. The Peterson's algorithm requires special hardware facility.
h. The problem with Peterson's solution is that it requires busy waiting.
i. The Peterson's method requires two data items to be shared between two processes.
j. There are two procedures in the Peterson's solution for mutual exclusion.

3.10 SLEEP & WAKE UP

The Peterson's solution discussed above is based on busy waiting. Busy waiting can be time-wasting. Processes waiting to enter their critical sections waste the CPU time checking to see if they can proceed. Also there can be sometimes unexpected effects. Suppose that there are two processes---- Process H with higher priority and process L with low priority. In scheduling a process with higher priority should be allowed to run when it is ready. But suppose here if Process Lis executing in is critical section and at that moment H becomes ready. H now begins busy waiting but since L was never scheduled while H is running, L never gets the chance to get out of the critical region, so H loops there forever. This situation is referred to as ***Priority Inversion Problem.***

A better solution to the mutual exclusion problem, which can be implemented with the addition of some new primitives, would be to block processes when they are denied access to their critical sections instead of wasting the CPU time. Two primitives, ***Sleep and Wakeup***, are often used to implement blocking in mutual exclusion.Sleep is a system call that allows the caller to block, that is, suspended until another process wakes it up. The Wakeup call has one parameter, the process that has to be awakened.

3.11 THE PRODUCER CONSUMER PROBLEM

The Producer Consumer problem is also known as the Bounded Buffer problem. Two processes share a common fixed-size buffer. The producer creates information and puts that information into the buffer and the consumer takes that information out of the buffer. The code for both producer and consumer is given below.

```
#define N 100                                   /* number of slots in the buffer */
int count = 0;                                  /* number of items in the buffer */

void producer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          item = produce_item();                /* generate next item */
          if (count == N) sleep();              /* if buffer is full, go to sleep */
          insert_item(item);                    /* put item in buffer */
          count = count + 1;                    /* increment count of items in buffer */
          if (count == 1) wakeup(consumer);     /* was buffer empty? */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* repeat forever */
          if (count == 0) sleep();              /* if buffer is empty, got to sleep */
          item = remove_item();                 /* take item out of buffer */
          count = count - 1;                    /* decrement count of items in buffer *
          if (count == N - 1) wakeup(producer); /* was buffer full? */
          consume_item(item);                   /* print item */
     }
}
```

There are some constraints in the producer-consumer problem:

- The consumer must wait for the producer to fill the buffer
- Producer must wait for the consumer to empty the buffer, when all buffer space is in use.
- Only one process must manipulate buffer pool at a time (mutual exclusion)

The trouble arises when the producer wants to put a new item in the buffer but the buffer is already full. The solution is for the producer to go to sleep and be awakened when the consumer has consumed one or more items from the buffer. Similarly if the consumer wants to remove an item from the buffer and finds the buffer empty. The solution for the consumer also is to go to sleep and be awakened when the producer has produced some items and had put the items in the buffer.

Suppose that the maximum number of items that he buffer can hold is N. the producer's code will first test to check if *count* is N. if so, the producer goes to sleep. If it is not N, the producer will add an item to the buffer and increment *count.* Similarly, the consumer's code will first check if *count* is 0. If so, then the consumer will go to sleep but if *count* is not 0, then it will remove an item from the buffer and decrement *count*. The producer's and consumer's code also checks if the other process should be sleeping and if they should not be sleeping then wakes it up.

Although the approach seems to be simple it can also lead to race conditions because the access to *count* is unconstrained. Let us consider a possible situation.

Suppose the buffer is empty and the consumer had checked *count* tofind it to be 0. At that instant, process switch occurs and the consumer is temporarily suspended. The producer is allowed to run by the scheduler. The producer enters an item in the buffer and increments *count*.Finding *count* to be 1, the producer calls the *wakeup* signal to wake up the consumer assuming that the consumer was asleep.

Unfortunately, the consumer was not yet logically asleep, so the wakeup signal was lost. When the consumer again gets the CPU to run, it will test the value of *count* it previously read and finding it to be 0, it will go to sleep. Sooner or later the producer will fill up the buffer and once full it will also go to sleep. This way both will sleep forever.

3.12 SEMAPHORES

The problem with implementing Sleep and Wakeup policy is the potential for losing the wakeup signals. Semaphores solve the problem of lost wakeups.

E W Dijkstra(1965) introduced a new variable type called Semaphore. It is an integer variable to count the number of wakeups that is saved for future use. A semaphore can have value 0 if no wakeups were saved or some positive value if

there were one or more wakeups that were pending. Two operations were proposed by Dijkstra on the semaphores ---- DOWN and UP (generalizations of Sleep and Wakeup respectively).

The DOWN operation checks the value of the semaphore. If the value is greater than 0, then it decrements the value (that is, it uses up one stored wakeup) and just continues. If the value is 0 then the process is put to sleep without completing the DOWN operation for the moment.

The UP operation increments a semaphore value.If one or more processes were sleeping on that semaphore, unable to complete its DOWN operation, the system chooses one of them, for examplerandomly, and is allowed to complete its DOWN operation. Thus after an UP operation on a semaphore with processes sleeping on it, the value of the semaphore will still be 0, but there will be one fewer process sleeping on it.

The checking of the semaphore value, changing it and possibly let it go to sleep or wake it up is done as a single, indivisible, atomic action. When an operation on a semaphore starts then no other process can access that semaphore until that operation has completed or blocked. This helps in solving the synchronization problems and avoiding the race conditions.

Solving the Producer-Consumer Problem using Semaphores:

In the Producer-Consumer problem, semaphores are used for two purposes:

- mutual exclusion and
- synchronization.

Semaphores solve the Producer-Consumer's lost wakeup problem as shown in the figure below:

```
#define N 100                                   /* number of slots in the buffer */
typedef int semaphore;                          /* semaphores are a special kind of int */
semaphore mutex = 1;                            /* controls access to critical region */
semaphore empty = N;                            /* counts empty buffer slots */
semaphore full = 0;                             /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE) {                             /* TRUE is the constant 1 */
          item = produce_item( );                /* generate something to put in buffer */
          down(&empty);                          /* decrement empty count */
          down(&mutex);                          /* enter critical region */
          insert_item(item);                     /* put new item in buffer */
          up(&mutex);                            /* leave critical region */
          up(&full);                             /* increment count of full slots */
     }
}


void consumer(void)
{
     int item;

     while (TRUE) {                             /* infinite loop */
          down(&full);                           /* decrement full count */
          down(&mutex);                          /* enter critical region */
          item = remove_item( );                 /* take item from buffer */
          up(&mutex);                            /* leave critical region */
          up(&empty);                            /* increment count of empty slots */
          consume_item(item);                    /* do something with the item */
     }
}
```

In the solution three semaphores are used: *full* for counting the number of slots in the buffer that are full, *empty*for counting the number of empty slots and *mutex*is used for mutual exclusion that is to ensure that the producer and consumer do not access the buffer at the same time. *Full* and *empty* semaphores are used for synchronization.

Initially, *full* is 0, *empty* is equal to the number of slots in the buffer and *mutex*is 1. The semaphores that are initialized to 1 and used by two or more processes to

ensure that at a time only one of the processes can enter the critical region are called ***Binary Semaphores***.

full is incremented and *empty* is decremented when a new item is added in the buffer slot.

CHECK YOUR PROGRESS 3

1. Fill up the blanks:
   a. Busy waiting can _____ the CPU time.
   b. A solution to the mutual exclusion problem would be to _____ when they are denied access to their critical sections instead of wasting the CPU time.
   c. _____ are often used to implement blocking in mutual exclusion.
   d. The Producer Consumer problem is also known as the _____ problem.
   e. In Producer Consumer problem the processes share a common _____ buffer.
   f. When the buffer is full the producer goes to_____ .
   g. Semaphores solve the problem of _____ signal.
   h. _____ are the two operations on the semaphores.
   i. _____ helps in solving the synchronization problems and avoiding the race conditions.
   j. The semaphores that are initialized to 1 and used by two or more processes to ensure that at a time only one of the processes can enter the critical region are called_____ .

3.13 LET US SUM UP

1. The method of communication between two or more processes is known as Interprocess Communication (IPC).
2. Interprocess communication may be direct or indirect. In indirect communication, processes make system calls whereas in direct communication, processes directly communicate by reading and modifying values in the common physical memory locations.
3. Processes communicate among themselves by manipulating the values of the shared variables and by executing operations supported by these variables.
4. A race condition occurs when the result of an operation depends on the ordering of when individual processes are run.
5. A critical section or critical region is a block of code where the shared resources (memory, files) are accessed.
6. To avoid race conditions, no two processes should be allowed to execute their critical sections at the same time. This is called ***Mutual Exclusion***.
7. The Peterson's algorithm is a software approach and is based on busy waiting.
8. Busy waiting can be time-wasting. Processes waiting to enter their critical sections waste the CPU time checking to see if they can proceed.
9. Two primitives, **Sleep and Wakeup**, are often used to implement blocking in mutual exclusion.
10. Sleep is a system call that allows the caller to block or suspend a process until another process wakes it up. The Wakeup call awakens a sleeping process.
11. The Producer Consumer problem is also known as the Bounded Buffer problem as the processes share a common fixed-size buffer.
12. The producer creates information and puts that information into the buffer and the consumer takes that information out of the buffer.
13. The problem with implementing Sleep and Wakeup policy is the potential for losing the wakeup signals.
14. Semaphores solve the problem of lost wakeups.
15. Semaphore is an integer variable.

16. Two operations were proposed by Dijkstra on the semaphores ---- DOWN and UP (generalizations of Sleep and Wakeup respectively).
17. The DOWN operation decrements a semaphore value and the Up operation increments a semaphore value.
18. Checking the semaphore value, changing it and possibly let it go to sleep or wake it up is done as a single, indivisible, atomic action.
19. Semaphores help in solving the synchronization problems and avoiding the race conditions.
20. The semaphores that are initialized to 1 and used by two or more processes to ensure that at a time only one of the processes can enter the critical region are called Binary Semaphores.

3.14 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS – 1

1. a. Interprocess Communication

b. data items

c. indirect communication

d. spooler directory

e. printer daemon

f. race condition

g. critical section

h. Mutual exclusion

i. assumptions

j. Direct, indirect

CHECK YOUR PROGRESS –2

1. a False

   b. True

   c. True

   d. False

   e. True

   f. False

   g. False

   h. True

   i. True

   j. True

CHECK YOUR PROGRESS –3

1. a Waste
   b. block processes
   c. Sleep and Wakeup
   d. Bounded Buffer
   e. fixed-size
   f. sleep
   g. lost wakeup
   h. DOWN and UP
   i. Semaphores
   j. Binary Semaphores

3.15 FURTHER READINGS

1. "Systems Programming and Operating Systems" ,*D M Dhamdhere*, Tata McGraw Hill.
2. "Operating System Design and Implementation", Andrew S Tanenbaum, Albert S Woodhull, Prentice-Hall India
3. "Operating System", H M Deitel, P J Deitel, D R Choffnes, Pearson Education

3.16 MODEL QUESTIONS

1. What is a race condition?
2. Discuss critical section.
3. What are the disadvantages of Disabling Interrupt?
4. Discuss Peterson's algorithm in achieving mutual exclusion using semaphores.

*****

UNIT 4: SCHEDULING

UNIT STRUCTURE

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- illustrate the basic concepts of CPU scheduling
- learn the types of scheduling
- describe the criteria for CPU scheduling
- explain the different scheduling algorithms

4.2 INTRODUCTION

CPU scheduling is the basis of multiprogramming. As we have already discussed in the previous unit that in a multi-programming environment there may be more than one process that may be running in the computer at the same time.

Processes communicate with one another, that is interprocess communication occurs among the different processes in the system.

CPU Scheduling plays a vital role in the basic framework of the operating system. It is the activity of determining which process request should be handled by the system next. Scheduling is important as it influences user service and the efficiency of the system. In this unit we will discuss in details about what is scheduling and what are the different scheduling algorithms that are available for CPU scheduling. Different system environments require different types of scheduling algorithms.

4.3 BASIC CONCEPTS

We have already learnt about multiprogramming systems in the Unit 1. In multiprogramming systems, multiple programs can run in the computer system simultaneously. The main purpose of having multiprogramming systems is to increase the CPU utilization by keeping the CPU as busy as possible.

In multiprogramming, several programs that are ready to be executed in the CPU are kept in the main memory at a time. In reality the CPU can execute only one program at a time. The running instance of a program is called a *process*.When a process is being executed in the CPU, the other ready processes are kept in a ready queue, waiting to get the CPU time. When the process that is running has to wait for some event, say an I/O operation, the operating system takes the CPU away from that process and gives the CPU time to another process in the queue. This happens in case of multiprogramming systems. In a simple computer system, the CPU sit idle until the I/O request is granted, thereby wasting its time.

The selecting of a process from among the ready processes in the memory and allocating the CPU to the selected process for execution is called ***CPU Scheduling.***

The selection process is carried out by a ***CPU Scheduler***. A scheduler is an operating system program or module that selects a job or process which is to be admitted next for execution. There are three types of schedulers.

a. Long-term Scheduler
b. Medium-term Scheduler
c. Short-term Scheduler

Long-term Scheduler (Job Scheduler):

The processes after their creation are kept in the secondary storage devices such as disks for processing later. The long-term scheduler or the Job Scheduler selects the jobs or processes from this job pool and loads them into the memory. The processes in the memory, as already mentioned, are kept in a ready queue.

Short-term Scheduler (CPU Scheduler):

The CPU scheduler selects a process from the ready queue in the memory and allocates the CPU to that process with the help of the ***dispatcher***. The dispatcher is the module that gives the control of the CPU to the process that has been selected by the short-term scheduler. This involves the following:

a. Switching context
b. Switching to user mode
c. Jumping to the proper location in the user program to restart that program.

The time that the dispatcher takes to stop one process and to start running another process is known as the *dispatch latency*. The method of selecting a process by the CPU scheduler depends on the type of the *Scheduling Algorithm* used by the system.

Medium-term Scheduler: This scheduler decides when and which process to swap in or swap out of the memory. This also controls the degree of multiprogramming.

4.4 PRIMITIVE AND NON-PRIMITIVE SCHEDULING

Scheduling is of two types depending upon how the processes are executed ----

a. Non-Preemptive Scheduling
b. Preemptive Scheduling

In non-preemptive scheduling, once the CPU is assigned to a process it cannot be taken away from the process until the completion of that process. Thus the process that is running currently in the processor has the control of the processor as well as the allocated resources. This means that even if a process with higher priority enters the system, the executing process cannot be forced to suspend. Context switch occurs only when the process is blocked or terminated.

In case of preemptive scheduling, if a process with higher priority enters the system, the process that is currently running can be suspended, even if it has not requested for any I/O operation or itsallocated time is not completed,and the higher priority process may be allowed to execute. This type of scheduling is advantageous for online or real-time processing where interactive users and high priority processes require immediate attention.

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting
2. Switches from running to ready
3. Switches from waiting to ready
4. terminates

Scheduling under 1 and 4 is non-preemptive. All other are preemptive scheduling.

Advantages of non-preemptive scheduling:

1. They are simple.
2. They cannot lead to race conditions.

The disadvantage is as follows:

1. They do not allow real multiprogramming

Advantage of Preemptive Scheduling is as :

1. Real multiprogramming is allowed here.

Disadvantages of Preemptive Scheduling are as :

1. They may lead to race conditions
2. They are complex.

4.5 SCHEDULING ALGORITHMS

When more than one process is runnable, the operating system must decide which process to run first. CPU scheduling is the process of selecting a process from the ready queue and assigning the CPU to it. The operating system program or module that makes this decision is called a **scheduler** and the algorithm that it uses is called **Scheduling Algorithm**.

The scheduling algorithms can be categorized based on preemptive scheduling algorithm and non-preemptive scheduling algorithms. Sometimes it is also categorized according to the system whether it is a batch system, interactive system or a real system.

CHECK YOUR PROGRESS 1

1. Fill in the blanks:

a. Multiprogramming systems _____ the CPU utilization.

b. The running instance of a program is called a_____.

c. _____ is the selection of a process from the ready queue and allocating the CPU time to it.

d. The operating system program that selects a job which is to be executed next is called a _____.

e. The _____ is the module that gives the control of the CPU to the process that has been selected by the short-term scheduler.

f. The two types of scheduling are _____ and _____.

g. The algorithm that the scheduler uses to schedule the processes is called _____.

h. In CPU scheduling, each process should get a _____ share of the CPU time.

i. The throughput of the system should be _____ in scheduling.

j. _____ scheduling is advantageous for online processing where interactive users and high priority processes require immediate attention.

4.6 TYPES OF SCHEDULING - BATCH, INTERACTIVE AND REAL-TIME

Different scheduling algorithms are needed in different environments with different goals, applications and different operating systems. There are basically three different environments as follows:

a. Batch
b. Interactive, and
c. Real

In case of Batch scheduling systems, there is almost no user interaction with the system. This type of scheduling algorithms may be preemptive or non-preemptive. The execution may also take long periods which are also acceptable. This approach reduces process switches and thus improves performance. Examples of batch scheduling systems are First Come First Served (FCFS), Shortest Job First (SJF) and Shortest Remaining Time Next. The FCFS and SJF algorithms are Non-Preemptive whereas the Shortest Remaining Time Next is a Preemptive version of the SJF algorithm.

In Interactive scheduling environments, the preemptive technique is essential to keep the process hogging the CPU and denying other services. The interactive

scheduling algorithm has to respond to the requests quickly.Examples of such type of scheduling are Round Robin algorithm, Priority based algorithm etc.

In some real-time scheduling algorithms, a task can be preempted if another task of higher priority becomes ready. In contrast, the execution of a non-preemptive task shouldbe completed without interruption, once it is started. The goal of real-time scheduling algorithms is to meet the deadlines set and not to lose any data.

4.7 GOALS OF SCHEDULING ALGORITHMS

There are many scheduling algorithms that are available and each of the algorithms has different properties. These properties help in determining the best algorithm for a process. In order to have a good algorithm we have to have an idea of what the algorithm should perform. The goal of any scheduling algorithm is to fulfill a number of criteria. The criteria that are used include the following:

1. CPU Utilization: The CPU must be kept busy all the time. CPU utilization ranges from 0% to 100%. Actually in real systems, it ranges from 40% (for light loaded systems) to 90% (for heavily used systems).
2. Throughput: The number of processes that are completed per unit time is known as the throughput of the system. A good scheduling algorithm is the one that has higher throughput.
3. Turnaround Time: The time interval between the submission of a process and its completion is known as the turnaround time. It is the sum of the time that is spent waiting to get into memory, waiting in the ready queue, executing on the CPU and performing the I/O operation.
4. Waiting time: Waiting time is the sum of the period that a process spends in the ready queue.
5. Response time: Response time is the time duration between the submission of a request and its first response.
6. Fairness: Makes sure that each process gets a fair share of the CPU.

A best CPU scheduling algorithm should have the following properties:

i. Maximum CPU utilization

ii. Maximum throughput
iii. Minimum turnaround time
iv. Minimum waiting time
v. Minimum response time.

Some goals of the scheduling algorithm under different circumstances are as follows:

All systems

Fairness - giving each process a fair share of the CPU
Policy enforcement - seeing that stated policy is carried out
Balance - keeping all parts of the system busy

Batch systems

Throughput - maximize jobs per hour
Turnaround time - minimize time between submission and terminat
CPU utilization - keep the CPU busy all the time

Interactive systems

Response time - respond to requests quickly
Proportionality - meet users' expectations

Real-time systems

Meeting deadlines - avoid losing data
Predictability - avoid quality degradation in multimedia systems

CHECK YOUR PROGRESS 2

1. Say True or False:
a. The same scheduling algorithm can be used in any environment.
b. In batch systems there is very low user-machine interaction.
c. The Shortest Job First algorithm is an example of Interactive scheduling system.
d. The preemptive version of Shortest Job First is the Shortest Remaining Time Next algorithm.
e. Priority Based scheduling is a non-preemptive scheduling technique.

f. Real-time algorithms must meet the deadlines within the time specified.
g. A good scheduling algorithm is the one that has higher throughput.
h. The number of processes that are completed per unit time is known as the Turnaround time.
i. Each process in the system must get a fair share of the CPU time.
j. The waiting time should be minimum in a scheduling algorithm.

4.8 FIRST COME FIRST SERVE

The First Come First Served (FCFS) Scheduling algorithm is the simplest algorithm. In this algorithm the set of the ready processes is kept in a FIFO (First-In-First-Out) queue. The CPU processes the jobs in the order they request the CPU. The process requesting the CPU first will be executed by the CPU first and so on. It is a non-preemptive algorithm. Thus the CPU executes the processes until their completion in the order their entering the FIFO queue.

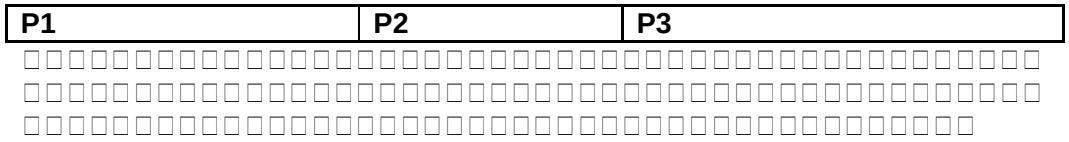
For example, consider the three processes P1, P2 and P3, whose CPU burst time is as follows:

Process	Burst Time
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: *P1*, *P2*, *P3.*

Burst time is the time required by the CPU to execute the process. The graphical representation of the process execution can be shown using a “Gantt Chart”.

The Gantt Chart for the schedule is:



To calculate the average waiting time and turn-around time for the processes,

Average Waiting time:

Waiting time for P1 = 0

" " " P2 = 24

" " " P3 = 27

Thus, the average waiting time for the processes are(0+24+27)/3 = 17

The average turn-around time is as :

Turn-around time for P1 = 24

Turn-around time for P2 = 27

Turn-around time for P3 = 30

Thus the average turn-around time is (24+27+30)/3 = 27

As the FCFS algorithm is a non-preemptive algorithm, the system throughput is less and therefore it gives poor performance. Also the short jobs have to wait for longer times when the CPU is allocated to longer jobs.

4.9 SHORTEST JOB FIRST

In the Shortest Job First algorithm, the scheduler selects the job which is the shortest, process it and then the next higher is processed and so on. This algorithm associates with each process the length of its next CPU burst. These lengths are used by the scheduler to schedule the process with the shortest time. If two processes are having the same CPU burst time then FCFS is used to break the tie. There are two different schemes:

1. Non-preemptive : once the CPU is given to a process it cannot be preempted until completion.
2. Preemptive :when a new process arrives having the CPU burst length less than the remaining time of the current executing process, then the

executing process is preempted and the new process is given the CPU time. This scheme is known as the *Shortest Remaining Time First*.

Consider the four processes, P1 P2, P3 and P4:

Process	Burst Time
P1	5
P2	10
P3	8
P4	3

In the order of their shortest CPU time or Burst time,

Process	Burst Time
P4	3
P1	5
P3	8
P2	10

SJF (non-preemptive). The Gantt Chart is as :

□□	□□	□□	□□

□□
□□
□□
□

Average waiting time = (0+3+8+16)/4 = 6.75.

Now, consider the same example as in the FCFS algorithm, where there were three processes P1, P2 and P3.

Process	Burst Time
P1	24
P2	3
P3	3

The Gantt Chart for the SFJ schedule is as :

P2	**P3**	**P1**

The processes P2 and P3 have the same CPU burst time, so suppose P2 arrived earlier to P3 then using FCFS algorithm P2 will be allowed to run first and then P3. P1 having the burst time more than P2 and P3 will be given the CPU time later.

The average waiting time = (0+3+6)/3 = 3

Comparing the average waiting time with that of the FCFS algorithm, it is seen that there is a considerable reduction in the average waiting time whenever the shorter jobs are processed first instead of using the FCFS algorithm. So, the SJF is an optimal algorithm in terms of minimizing the average waiting time for a given set of processes.

The main problem of this algorithm is to know the length of the burst time. Another problem is the starvation problem where the big jobs has to wait long to get the CPU time.

4.10 ROUND ROBIN SCHEDULING

This is one of the oldest, simplest and most widely used algorithms. It is a preemptive algorithm and is especially designed for time-sharing systemsin which the system needs to guarantee reasonable response times for interactive users. The CPU time in RR Scheduling is divided into small unit of time called time-slices or quantum. Each process gets a small quantum, usually 10 – 100 milliseconds. The ready processes are kept in a queue. If a process does not complete before its CPU time quantum expires, the CPU is preempted and given to the next waiting process in the queue. The preempted process is then added to the end of the queue.

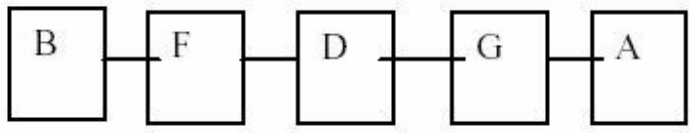


Fig (a) the list of runnable process

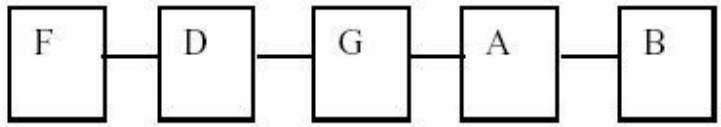


Fig (b) the list of runnable processes after B uses up its quantum

When there are processes waiting in the queue, no process can run for more than one time-slice. If a process completes before its quantum, it releases the CPU voluntarily.

Switching from one process to another requires a certain amount of time for doing the administration --- saving and loading registers, updating the tables etc. the interesting thing in Round Robin algorithm is the length of the quantum. Setting the quantum too short causes too many process switches and lowers the CPU efficiency, but setting it too long may cause poor response to short interactive requests.

CHECK YOUR PROGRESS 3

1. Fill in the blanks:

a. In FCFS scheduling algorithm, the set of the ready processes is kept in a _____.

b. The processes are executed in the _____ of their entry in the queue.

c. FCFS is a _____ algorithm.

d. The system performance in case of FCFS scheduling is _____.

e. _____ Algorithm associates with each process the length of its next CPU burst.

f. The preemptive version of SJF algorithm is also known as _____.

g. SJF is an optimal algorithm in terms of minimizing the _____time for a given set of processes.

h. The Round Robin algorithm is a _____ algorithm.

i. The CPU time in RR Scheduling is divided into_____.

j. Setting the quantum in RR Scheduling too short causes too many _____ and lowers the CPU efficiency.

4.11 LET US SUM UP

1. Multiprogramming systems increases the CPU utilization by keeping the CPU as busy as possible.
2. The running instance of a program is called a *process.*
3. The selecting of a process from among the ready processes in the memory and allocating the CPU to the selected process for execution is called CPU Scheduling.
4. A scheduler is an operating system program or module that selects a job or process which is to be admitted next for execution.
5. There are three types of schedulers.
   d. Long-term Scheduler
   e. Medium-term Scheduler
   f. Short-term Scheduler
6. The dispatcher is the module that gives the control of the CPU to the process that has been selected by the short-term scheduler.
7. Scheduling is of two types depending upon how the processes are executed ----
   c. Non-Preemptive Scheduling
   d. Preemptive Scheduling
8. In non-preemptive scheduling, once the CPU is assigned to a process it cannot be taken away from the process until the completion of that process

9. In preemptive scheduling, if a process with higher priority enters the system, the process that is currently running can be suspended, and the higher priority process may be allowed to execute.
10. The algorithm that the scheduler uses to select which process to run is called Scheduling Algorithm.
11. There are basically three different scheduling environments:
    d. Batch
    e. Interactive, and
    f. Real
12. First Come First Served (FCFS), Shortest Job First (SJF) and Shortest Remaining Time Next are examples of Batch Scheduling environment.
13. Round Robin algorithm, Priority based algorithms are examples of Interactive scheduling algorithms.
14. The First-Come-First-Served scheduling algorithm is a non-preemptive algorithm. The CPU processes the jobs in the order they request the CPU.
15. The Shortest Job First algorithm associates with each process the length of its next CPU burst.
16. If two processes in SJF are having the same CPU burst time then FCFS is used to break the tie among them.
17. There are two different schemes in SJF algorithm : Non-Preemptive and Preemptive.
18. The Round Robin algorithm is a preemptive algorithm and is especially designed for time-sharing systems.
19. The CPU time in RR Scheduling is divided into small unit of time called time-slices or quantum.
20. If a process does not complete before its CPU time quantum expires, the CPU is preempted and given to the next waiting process in the queue. The preempted process is then added to the end of the queue.

4.12 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS – 1

1. a.Increases

b. Process

c. Scheduling

d. Scheduler

e. Dispatcher

f. Preemptive, Non-preemptive

g. Scheduling Algorithm

h. Fair

i. Maximized

j. Preemptive

CHECK YOUR PROGRESS –2

1. a. False
   b. True
   c. False
   d. True
   e. False
   f. True
   g. True
   h. False
   i. True
   j. true

CHECK YOUR PROGRESS –3

1. a. Queue
   b. Order
   c. non-preemptive
   d. Poor
   e. Shortest Job First
   f. Shortest Remaining Time First
   g. average waiting
   h. Preemptive
   i. quantum
   j. process switches

4.13 FURTHER READINGS

1. *"*Systems Programming and Operating Systems*"* ,*D M Dhamdhere*, Tata McGraw Hill.
2. "Operating System Design and Implementation", Andrew S Tanenbaum, Albert S Woodhull, Prentice-Hall India
3. "Operating System", H M Deitel, P J Deitel, D R Choffnes, Pearson Education

4.14 MODEL QUESTIONS

1. Compare Preemptive scheduling and Non-preemptive scheduling
2. What are the different types of scheduling needed in different environments.
3. Explain the goals of scheduling
4. Explain the FCFS algorithm and the SJF algorithm. Compare the average waiting time of both the algorithms.

*****

UNIT 5: DEADLOCKS

UNIT STRUCTURE

5.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn the meaning of deadlock
- describe different principles of deadlock
- learn about deadlock detection and recovery
- describe deadlock prevention
- learn the procedure of avoiding deadlock
- illustrate Banker's algorithm for deadlock avoidance

5.2 INTRODUCTION

The resources of computer systems can only be used by one process at a time. For example CPU, memory, printers, tape drives etc. In a multiprogrammed system, the different resources are shared and used concurrently by several processes. It provides many benefits but

some additional complexities are introduced. One problem that arises in multiprogrammed system is deadlock. Deadlock is also a common problem in multiprocessing and distributed system. In this unit we are going to discuss deadlock which is defined as the permanent blocking of a set of processes which are waiting for a particular event that will not occur. Here we will discuss three techniques to eliminate deadlock which are (a) deadlock prevention, (b) deadlock avoidance, (c) deadlock detection and recovery.

5.3 INTRODUCTION TO DEADLOCK

The resources of computer system can be categories into two types which are preemptable and nonpreemptable. A preemptable resource is one that can be taken away from the process owning it with no ill effects. Memory is an example of a preemptable resource. On the other hand **nonpreemptable resources** cannot be taken away from its current owner without causing the computation to fail. Printer is an example of nonpreemptable resource.So preemptable resources can be used by more than one processes concurrently.

Now let us imagine that P1, P2 and P3 are three processes and R1, R2 and R3 are three resources. Now let us imagine a situation where

1) P1 is using R1 and waiting for R2
2) P2 is using R2 and waiting for R3
3) P3 is using R3 and waiting for R1

In this situation if the resources are nonpreemptable then all the three processes will be permanently in blocking state. This type of situation is called ***deadlock***. In case of preemptable resources, deadlock can be resolved by reallocating resources from one process to another. On the other hand, if R1, R2 and R3 are nonpreemptable resources then P1, P2 and P3 will be in permanently waiting state for their required resource. So in that case deadlock occurs.

A deadlock is a situation in which two or more processes are in permanently waiting state as each waiting for some event which will not occur.

5.4 PRINCIPLES OF DEADLOCK

In computer system, when a process is required to use a system resource then at first it requests the resource. If the resource is not available then the requesting process is forced to wait or automatically blocked till the requested resource becomes available. In some other systems, the request fails with an error code, and it is up to the calling process to wait a little while and try again. Now if the resource is available then the process can use the resource and after finishing the job the process releases the resource.

As we know, deadlocks that involve preemptable resources can usually be resolved by reallocating resources from one process to another. But in case of nonpreemptable resources it is not so simple. In the further discussion of deadlock, we will assume that when a resource request of a process is not successful then, it is put to sleep.

There are four conditions that must be hold for a deadlock occurrence stated by Coffman et al. (1971)

1. **Mutual exclusion condition:** Here each resource is allowed to be assigned to exactly one process at one time.
2. **Hold and wait condition:** In this condition, processes currently holding some resources are allowed to request new resources.
3. **No preemption condition:** Resources currently held by any process cannot be forcibly taken away from the process. Here the resources currently held by a process can be released by the process.

4. **Circular wait condition:** In this condition, two or more processes must be in a circular chain where each process is waiting for a resource held by the next member of the chain.

If any one of the above conditions is not satisfied then deadlock is not possible.

Now sometimes deadlock can be ignored if it is rarely occurred or it does not seriously affect the system.

In general, three techniques are used for dealing with deadlocks as given follows:

1. Detection and recovery: Deadlocks are detected and some strategies are applied to recover from it.
2. Deadlocks can be prevented by breaking one of the four conditions necessary to cause a deadlock.
3. Deadlocks can be avoided dynamically by careful resource allocation.

5.5 DEADLOCK DETECTION AND RECOVERY

In this technique, the operating system performs an algorithm to detect the occurrence of deadlock and then apply different types of recovery technique to eliminate the deadlock. The detection of deadlock is done by detecting the circular wait condition. Now we will discuss some ways to detect deadlock and some recovery techniques.

Deadlock Detection

At first we will consider the simplest situation where only one resource of each type exits. In this case, a resource allocation graph is constructed for different resource requests of different processes. If this graph contains one or more cycles, a deadlock exists. The

processes which are part of a cycle are deadlocked. Now if the graph does not have any cycle then it means that the system does not have any deadlock.

For example, let there are 5 processes(P1,P2,P3,P4,P5) and 7 resources(R1,R2,R3,R4,R5,R6,R7) in a system. Now which process is currently using which resources and which ones are currently being requested is given as follows:

1. Process P1 holds R1 and request for R2.
2. Process P2 holds R2 and request for R3.
3. Process P3 holds R3 and request for R5 and R6.
4. Process P4 holds R5 and request for R4 and R7.
5. Process P5 holds R4 and request for R2.

Now we can construct the resource graph shown in Fig. 5.1(a). This graph contains one cycle shown in Fig. 5.1(b). From this cycle, we can see that processes P2, P3, P4 and P5 are deadlocked. Here P1 is also deadlocked as it is waiting for the resource R2 but R2 is currently used by the process P2 which is deadlocked.

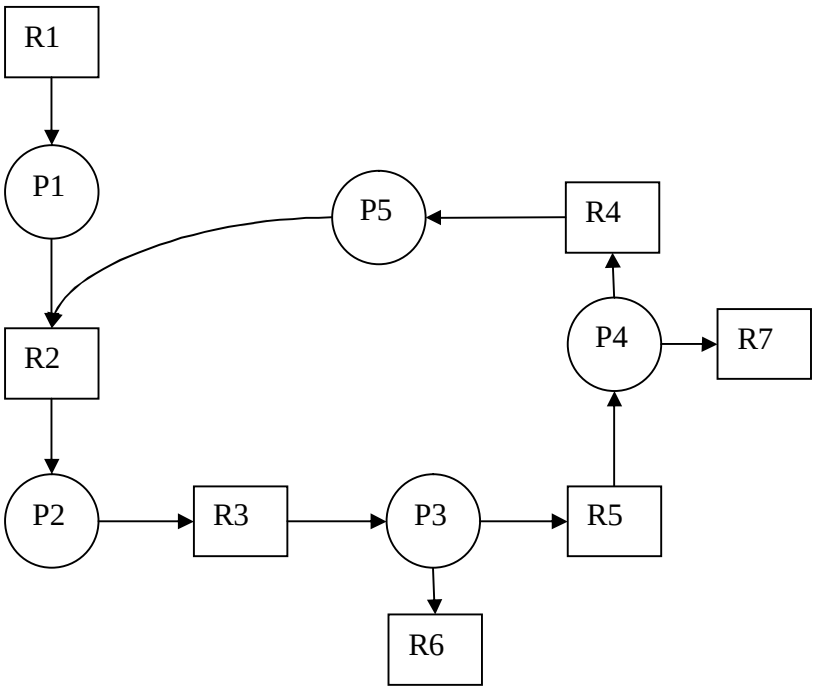


Fig.5.1 (a) A Resource graph

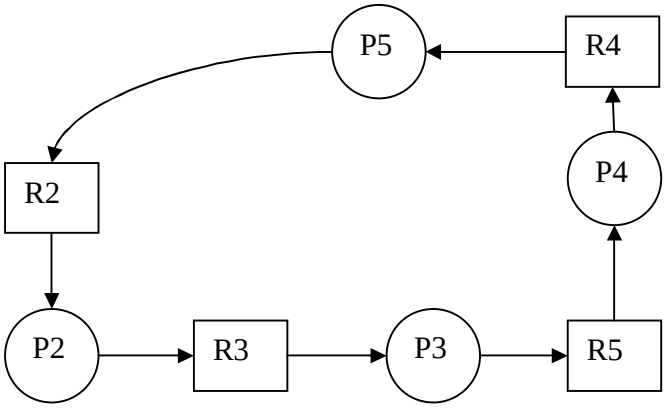


Fig.5.2 (b) A Cycle of 5.1(a)

Now there are different algorithms for detecting cycles in directed graphs are available. Here we will discuss an algorithm which uses

one data structure, L, a list of nodes. During the algorithm, arcs will be marked to indicate that they have already been inspected.

The algorithm operates by carrying out the following steps as specified:

1. Take an empty list , L
2. All the arcs in the resource graph are unmarked and make a node, N as current or starting node.
3. Add the current node to the list, L and check that the node now appears in *L* two times. If it does then it means that the graph contains a cycle and the algorithm terminates.
4. Now if there are any unmarked outgoing arcs from the current node then go to step5, otherwise go to step 6.
5. One unmarked outgoing arc is selected to be marked. Then the node at the end of the selected arc is set as the new current node and go to step 3.
6. It is the end of the path in the resource graph. Now remove the current node and go back to the previous node, which was a current node just before the removed one. Make this node as the current node, and go to step 3. But if this node is the starting node, N then it means that the graph does not contain any cycles and the algorithm terminates.

Now we will consider the situation where multiple resources of same types exist. In this more complex situation we have to use a different approach to detect deadlocks. Let there are n processes (P_1, P_2 ,P_3 ,…….P_n) competing for m classes of resources(E_1, E_2, E_3 ,…,E_m). Here if class 1 resource is printer then E_1=3 means that the system has three printers. So E gives the total number of resources in existence.

Now another data structure is used to give the total number of resources currently available for each class. Let A be the available

resource vector, with A_i giving the number of instances of resource class i that are currently available.

Now in this approach two matrices are required. Let matrix *C,* the current allocation matrix, and matrix *R*, the request matrix is used. The ith row of *C* gives the number of resources of each resource class currently held by the process P_i . Similarly, R[i][j] is the number of instances of resource class j requested by the process P_i .

Here initially each process is unmarked. Now as the algorithm progresses, processes will be marked, which means that they can use all the required resources and so able to complete their job. When the algorithm terminates, if there is any unmarked processes available then it will be deadlocked.

The deadlock detection algorithm can now be given, as follows.

1. A searching is done to find out an unmarked process, P_i, for which the i th row of matrix R is less than or equal to A.
2. If such a process is found, add the i th row of *C* to *A*, mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.

When the algorithm terminates we will find all the unmarked processes. If no unmarked process is found then no deadlock will occur.

Recovery from Deadlock

After the successful detection of deadlocks, we have to find a way to eliminate the deadlocks. So for this purpose, we will discuss some ways of recovering from deadlock.

1. In case of preemptable resources it is possible to temporarily take a resource away from its current owner and give it to another process. On the other hand, in case of nonpreemptable resource it is difficult or sometimes impossible to be shared among different processes concurrently. But in some cases it may be possible. For example, a printer can be taken away temporarily from its current owner which is currently suspended if the operator can collect all the sheets already printed and after the new owner of the printer finishes, the operator can put back all the printed sheets in the printer's output tray and the original process can be restarted. But this approach can not be used for all nonpreemptable resources and in every situation.

2. Arranging checkpoints for processes and rollback is a way to recover from a deadlock. Check pointing is a process by which the current state of a process is written to a file so that it can be restarted later. The checkpoint of a process contains the memory image, and the list of resources currently assigned to the process. Here new checkpoints should not overwrite the old ones but should be written to new files. Rollback means restart a process execution from a previously defined check point. So when a deadlock is detected then back up of each deadlock process is taken to some previously defined checkpoints and rollback of the deadlocked processes is done. The disadvantage of this approach is that after rollback the original deadlock may occur again.

3. A deadlock can be eliminated by killing one or more processes. In this process after a process is killed it may have the possibility that the other process will be able to continue. But if the deadlock is not eliminated then more processes will be killed repeatedly until the cycle is broken. Now the process to be killed is carefully chosen because if a process is killed but it is not holding any resource required by some other

process in the cycle then it will not help in breaking the deadlock. On the other hand sometimes, a process is killed but it is not in the cycle in order to release its resources which are required by the processes in the cycle. The process which can be restarted from the beginning with no ill effects is the best one to kill.

5.6 DEADLOCK PREVENTION

In the earlier discussion, we have found that there are four conditions stated by Coffman et al. (1971) which must be satisfied to cause a deadlock. If we can find a way to prevent at least one of these conditions to be satisfied, then deadlocks can be prevented. Now we will try to attack these four conditions one by one.

Mutual Exclusion Condition

In case of the mutual exclusion condition, if no resource were ever assigned exclusively to a single process then deadlock will never occur. In case of printer, by spooling, more than one processes can generate output at the same time. So here we can prevent deadlock for the printer. But all types of system resources can not be spooled.

Hold and Wait Condition

In case of hold and wait condition, if we can prevent processes that hold resources from waiting for more resources then we can prevent deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution. If everything is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are not available then nothing will be allocated and the process would just wait. But this solution has two problems. First one is that many processes do not know how many resources they will need until they have started running. The

other problem is that optimal use of resources will not be possible with this approach.

Now a different way to break the hold and wait condition can be applied. In this approach, a process must temporarily release all the resources it currently holds before requesting any other resource. Then it tries to get all the required resources together.

No Preemption Condition

It is difficult or impossible to break the third condition (no preemption condition). So it is not a good idea to remove forcibly all type of resources from a process currently holding it.

Circular Wait Condition

The circular wait condition can be break in different ways. One way is to have a restriction, according to which a process can use only a single resource at any moment and if it requires an another one then it must release the first one. But this approach is not acceptable for all situations. In some situations, a process requires more then one resources together to perform its job.

Another way to prevent the circular wait is to make an order of different resources. So for this purpose, a global numbering is done to all of the resources. In this approach processes can request resources in a numerical order. With this restriction, the circular wait condition can never happen. For example, if process A requests resource R_i and process B requests resource R_j where i and j are global numbers of the distinct resources R_i and R_j respectively. Now if $i > j$, then A is not allowed to request R_j. If $i < j$, then B is not allowed to request R_i.

Here, resources can be requested in increasing sequence and in some cases it can be done in decreasing sequence also.

In this approach, the main problem is to find an ordering of resources which will be satisfied in any situation.

5.7 DEADLOCK AVOIDENCE

In this approach, deadlock can be avoided by careful resource allocation to any process. So here the system will grant a resource request if and only if doing this does not lead to an unsafe state. For this purpose, the system need some information in advance like resources currently available, resources currently allocated to each process and resources that will be required and released by these processes in the future.

Now the algorithms for performing deadlock avoidance are based on the concept of safe states. The state of the system reflects the current allocation of resources to processes. A safe state is one in which there is at least one sequence of resource allocations to processes that does not result in a deadlock. But in case of an unsafe state, the resource requests of processes will lead to a deadlock.

In case of deadlock avoidance, it is not necessary to preempt and rollback processes but it requires some restrictions as given follows:

- The maximum resource requirement for each process must be stated in advance.
- The execution of the processes under consideration should not be dependent upon any synchronization requirements.
- There must be a fixed number of resources to allocate.
- Processes holding resources are not allowed to exit.

Now in the next section we will discuss a scheduling algorithm to avoid deadlock known as Banker's algorithm published by Dijkstra in 1965.

5.7.1 BANKER'S ALGORITHM

The banker's algorithm is based on the way a banker might deal with a group of customers in case of granting credits. In this algorithm, a check is made to see if granting a resource request will leads to a safe state or unsafe state. If it is unsafe state then, the request is postponed until later. On the other hand if it is safe state then the resource request can be granted.

The Banker's Algorithm for a Single Resource

Now let us consider a situation, there are five processes (P_1,P_2,P_3,P_4 and P_5) are competing for 15 resources of a single type in a system of single resource.

P_1	0	7
P_2	0	4
P_3	0	5
P_4	0	8
P_5	0	3

Fig.5.2 (a)

P_1	4	7
P_2	2	4
P_3	3	5
P_4	3	8
P_5	1	3

Fig.5.2(b) Safe state

P_1	4	7
P_2	2	4
P_3	3	5
P_4	4	8
P_5	1	3

Fig.5.2(c) Unsafe state

In Fig.5.2(a),Fig.5.2(b) and Fig.5.2(c) the second column gives the number of resources currently held by the processes and the third column gives the maximum number of resources required by the processes. In Fig.5.2 (a) number of resources currently available is 15. Now, Fig.5.2(b) is an example of safe state. Here the number of available resource is 2. Now granting a resource request to P_4 will lead to a unsafe state shown in Fig.5.2(c). Here number of available resource is one and the requirement of each process is more than one. So, all five processes will be deadlocked.

So when a resource request comes according to the banker's algorithm, the system always checks the number of available resources and the requirement of each process in the system. Now if the system has found that after granting the resource request, the

number of the available resources is not sufficient for any process and no process can complete their jobs then the resource request is postponed.

The Banker's Algorithm for Multiple Resources

Now the banker's algorithm can be generalized to be used in a system with multiple types of resources. For this purpose we have two matrices, matrix in Fig.5.3 (a) and matrix in Fig.5.3 (b). The matrix in Fig.5.3 (a) shows the number of resources of each type (R_1,R_2,R_3 and R_4) currently assigned to each of the five processes (P_1, P2, P3, P4 and P_5). The matrix in Fig.5.3 (b) shows the number of resources still required by each process in order to complete their jobs.

	R_1	R_2	R_3	R_4
P_1	2	1	1	0
P_2	1	0	2	1
P_3	0	1	1	2
P_4	1	1	2	0
P_5	0	2	1	2

Fig.5.3 (a) Resources assigned

	R_1	R_2	R_3	R_4
P_1	2	1	3	1
P_2	2	1	2	1
P_3	1	2	1	2
P_4	2	1	1	1
P_5	1	3	2	1

Fig.5.3 (b) Resources still required

Now the algorithm to check a state is safe or unsafe is given as follows:

1. Search for a row in the matrix in Fig.5.3 (b) where the number of resources of each type is smaller than or equal to the currently available resources of each type. If no such row exists then the system will be in unsafe state and no process can complete their jobs.
2. Now the process of the row found in step 1 requests all the resources it needs and the system is in a safe state and so mark that process as terminated. Add all its resources as currently available resources.
3. Repeat steps 1 and 2 until either all processes are marked terminated or until a deadlock occurs.

CHECK YOUR PROGRESS

1. Multiple choice questions:

(I). Deadlock occurred in

A. Multiprogrammed system
B. Multiprocessor system
C. Both A and B
D. None of the above

(II). Which is not necessary to be satisfied to cause a deadlock?

A. Circular wait condition
B. No preemption condition
C. Mutual exclusion condition
D. Resource allocation graph

(III). Which one is a technique to handle deadlock?

A. Deadlock prevention
B. Deadlock avoidance
C. Deadlock detection and recovery
D. All of the above

(IV). Banker's algorithm is used in

A. Deadlock prevention
B. Deadlock avoidance
C. Deadlock recovery
D. None of the above

(V). Killing of one or more processes may be done in case of

A. Deadlock prevention
B. Deadlock avoidance
C. Deadlock recovery
D. All of the above

2. Fill in the blanks:

(I) _____ graph is used in deadlock detection.
(II) Deadlock can be prevented by _____.
(III) Deadlock can be avoided by careful _____.
(IV) Deadlock can occur in case of _____ resources.
(V) Banker's algorithm is used to _____.

3. State whether the following statements are true or false:

(I). It is not necessary to satisfy the hold and wait condition to cause a deadlock.

(II). In case of preemptable resources, deadlock can be resolved by reallocating resources from one process to another.

(III). Banker's algorithm is used in deadlock prevention.
(IV). In case of multiple resources, Banker's algorithm cannot be applied.
(V). Printer is an example of preemptable resource.

5.8 LET US SUM UP

The summery of this unit is given as follows:

- Deadlock is a problem occurred in multiprogrammed systems defined as the permanent blocking of a set of processes which are waiting for a particular event that will not occur.

- There are four conditions that must be hold for a deadlock occurrence stated by Coffman et al. (1971)
    - **Mutual exclusion condition**. Each resource is allowed to be currently assigned to exactly one process.
    - **Hold and wait condition**. Processes currently holding resources are allowed to request new resources.
    - **No preemption condition**. Resources held by any process cannot be forcibly taken away from it. The processes can release the resources held by them.
    - **Circular wait condition**. There must be a circular chain of two or more processes where each process is waiting for a resource held by the next member of the chain.
- Deadlocks are detected using resource allocation graph and some strategies are applied to recover from it. Check pointing processes, killing deadlocked processes etc are examples of deadlock recovery strategies.
- Deadlocks can be prevented by breaking one of the four conditions necessary to cause a deadlock.
- Deadlocks can be avoided dynamically by careful resource allocation. Banker's algorithm is a way to avoid deadlocks.

5.9 ANSWERS TO CHECK YOUR PROGRESS

1. (I) .C , (II).D , (III).D , (IV). B , (V). C

2.(I) Resource allocation

(II) breaking one of the four conditions necessary to cause a deadlock.

(III) resource allocation to the processes

(IV) nonpreemptable

(V) avoid deadlock

3. (I) False , (II) True , (III) False, (IV) False , (V) False

5.9 FURTHER READINGS

1. Andrew S. Tanenbaum : Modern Operating Systems ,Prentice-Hall
2. Silberschatz, Galvin, Gagne: Operating System Concepts
3. Stallings: Operating Systems: Internals And Design Principles, Pearson Education India

5.10 MODEL QUESTIONS

1. What is Deadlock? Explain.
2. What do you mean by Reusable Resource and Consumable Resource? How are they related with Deadlock?
3. What are the conditions that must be present for Deadlock to occur? Explain in brief.
4. Explain Deadlock Modeling.
5. What is the basic difference between Deadlock Detection and Deadlock Avoidance?
6. Explain the Banker's Algorithm for Deadlock Avoidance.
7. Write short notes on:
   (i) Deadlock Modeling.
   (ii) Deadlock Recovery.
   (iii) Deadlock Prevention.
   (iv) Deadlock Avoidance.

UNIT - 6: MEMORY MANAGEMENT

UNIT STRUCTURE

6.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn various ways of organizing memory hardware.
- describe various memory-management techniques
- illustrate the concept of virtual memory, paging, page table
- learn different page replacement algorithms.

6.2 INTRODUCTION

Memory is an important resource that must be carefully managed. In a *uniprogramming system*, *main memory* is divided into two parts: one part for the *operating system* (*resident monitor, kernel*) and one part for the *program* currently being *executed*. In a *multiprogramming system*, the "*user*" part of *memory* must be further subdivided to accommodate *multiple processes*. The task of subdivision is carried out *dynamically* by the *operating system* and is known as *memory management*.

Memory management is primarily concerned with allocation of main memory to requesting processes. No process can ever run before a certain amount of memory is allocated to it. In this unit we will discuss several memory management schemes which vary from fixed partition to virtual memory, paging and page replacement algorithms etc.

6.3 BASIC HARDWARE

Main memory and the registers built into the processor itself are the only storage that the CPU can access directly. There are machine instructions that take memory addresses as arguments, but none that take disk addresses. Therefore, any instructions in execution, and any data being used by the instructions, must be in one of these direct-access storage devices. If the data are not in memory, they must be moved there before the CPU can operate on them.

Registers that are built into the CPU are generally accessible within one cycle of the CPU clock. Most CPUs can decode instructions and perform simple operations on register contents at the rate of one or more operations per clock tick. The same cannot be said of main memory, which is accessed via a transaction on the memory bus. Memory access may take many cycles of the CPU clock to complete, in which case the processor normally needs to stall, since it does not have the data required to complete the instruction that it is executing. This situation is intolerable because of the frequency of memory accesses. The remedy is to add fast memory between the CPU and main memory. A memory buffer used to accommodate a speed differential, called a cache.

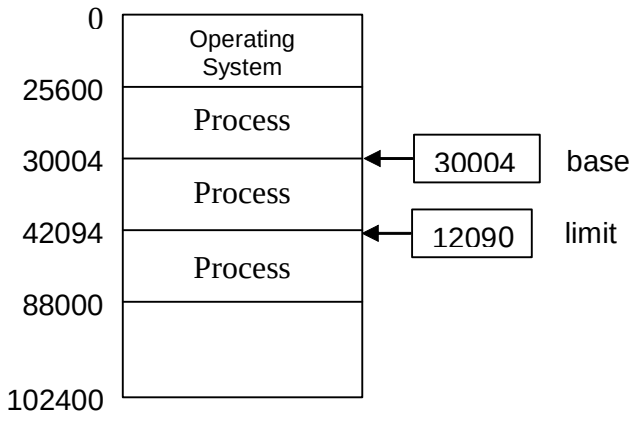


Fig. 6.1:

A base and a limit register define a logical address space. The first need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection by using two registers, usually a base and a limit, as illustrated in Fig. 6.1. The base register holds the smallest legal physical memory address; the limit register specifies the size of the range. For example, if the base register holds 30004 and limit register is 12090, then the program can legally access all addresses from 30004 through 42094 (inclusive).

6.4 MEMORY MANAGEMENT REQUIREMENTS

While surveying the various mechanisms and policies associated with memory management, it is indeed need to know the requirements that memory management is intended to satisfy. The requirements are:

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

Here, we will discuss only about Relocation and Protection.

6.4.1 RELOCATION

In a multiprogramming system, the available main memory is generally shared among a number of processes. Typically, it is not possible for the programmer to know in advance which other programs will be resident in main memory at the time of execution of his or her program. In addition, we would like to be able to swap active processes in and out of main memory to maximize processor utilization by providing a large pool of ready processes to execute. Once a program has been swapped out to disk, it would be quite limiting to declare that when it is next swapped back in, it must be placed in the same main memory region as before. Instead, we may need to relocate the process to a different area of memory.

Thus, we cannot know ahead of time where a program will be placed, and we must allow that the program may be moved about in main memory due to swapping. These facts raise some technical concerns related to addressing, as illustrated in Fig. 6.2. The figure depicts a process image. For simplicity, let us assume that the process image occupies a contiguous region of main memory. Clearly, the operating system will need to know the location of process control information and of the execution stack, as well as the entry point to begin execution of the program for this process. Because the operating system is managing memory and is responsible for bringing this process into main memory, these addresses are easy to come by. In addition, however, the processor must deal with memory references within the program. Branch instructions contain an address to reference the instruction to be executed next. Data reference instructions contain the address of the byte or word of data referenced. Somehow, the processor hardware and operating system software must be able to translate the memory references found in the code of the program into actual physical memory addresses, reflecting the current location of the program in main memory.

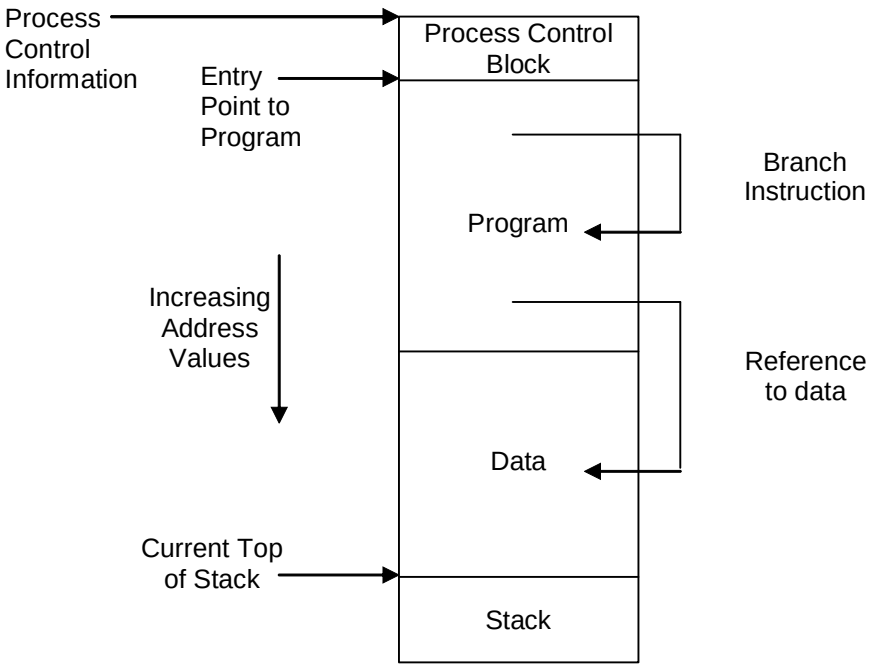


Fig. 6.2: Addressing Requirments for a Process

6.4.2 PROTECTION

Each process should be protected against unwanted interference by other processes, whether accidental or intentional. Thus, programs in other processes should not be able to reference memory locations in a process for reading or writing purposes without permission. In one sense, satisfaction of the relocation requirement increases the difficulty of satisfying the protection requirement. Because the location of a program in main memory is unpredictable, it is impossible to check absolute addresses at compile time to assure protection. Furthermore, most programming languages allow the dynamic calculation of addresses at run time (for example, by computing an array subscript or a pointer into a data structure). Hence all memory references generated by a process must be checked at run time to ensure that they refer only to the memory space allocated to that process. Fortunately, we shall see that mechanisms that support relocation also support the protection requirement.

Normally, a user process cannot access any portion of the operating system, neither program nor data. Again, usually a program in one process cannot branch to an instruction in another process. Without special arrangement, a program in one process cannot access the data area of another process. The processor must be able to abort such instructions at the point of execution.

Note that the memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software).This is because the operating system cannot anticipate all of the memory references that a program will make. Even if such anticipation were possible, it would be prohibitively time consuming to screen each program in advance for possible memory-reference violations. Thus, it is only possible to assess the permissibility of a memory reference (data access or branch) at the time of execution of the instruction making the reference. To accomplish this, the processor hardware must have that capability.

6.5 MEMORY PARTITIONING

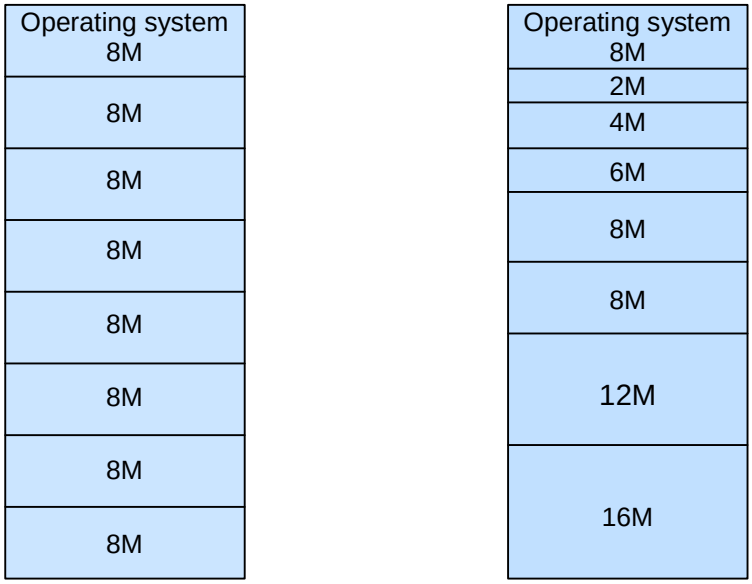
The principal operation of memory management is to bring processes into main memory for execution by the processor. In almost all modern multiprogramming systems, this involves a sophisticated scheme known as virtual memory. But before we can look at virtual memory techniques, we must prepare the ground by looking at simpler techniques that do not involve virtual memory. One of these techniques is Fixed Partioning.

6.5.1 FIXED PARTIONING

In most schemes for memory management, we can assume that the operating system occupies some fixed portion of main memory and that the rest of main memory is available for use by multiple processes. The simplest scheme for managing this available memory is to partition it into regions with fixed boundaries.

Partition Sizes:

Figure 6.3 shows examples of two alternatives for fixed partitioning. One possibility is to make use of equal-size partitions. In this case, any process whose size is less than or equal to the partition size can be loaded into any available partition. If all partitions are full and no process is in the Ready or Running state, the operating system can swap a process out of any of the partitions and load in another process, so that there is some work for the processor.



(a) Equal-size partitions (b) Unequal-size partitions

Fig. 6.3: Example of Fixed Partitioning of a 64-Mbyte

There are two difficulties with the use of equal-size fixed partitions:

- A program may be too big to fit into a partition. In this case, the programmer must design the program with the use of overlays so that only a portion of the program need be in main memory at any one time. When a module is needed that is not present, the user's program must load that module into the program's partition, overlaying whatever programs or data are there.
- Main memory utilization is extremely inefficient. Any program, no matter how small, occupies an entire partition. In our example, there may be a program whose length is less than 2 Mbytes; yet it occupies an 8-Mbyte partition whenever it is swapped in. This phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation.

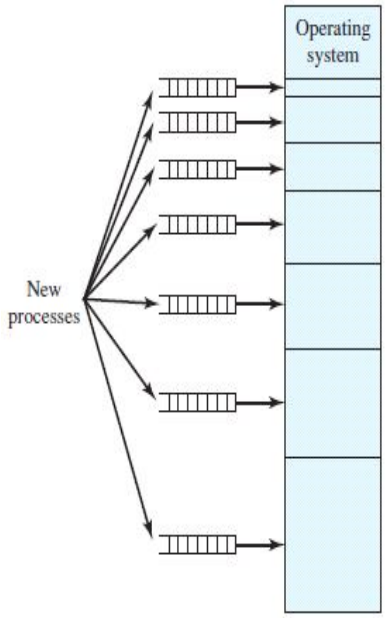
Both of these problems can be lessened, though not solved, by using unequalsize partitions (Fig. 6.3b). In this example, programs as large as 16 Mbytes can be accommodated without overlays. Partitions smaller than 8 Mbytes allow smaller programs to be accommodated with less internal fragmentation.

Placement Algorithm:

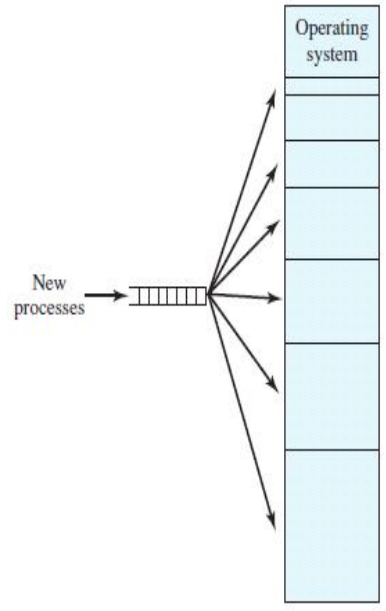
With equal-size partitions, the placement of processes in memory is trivial. As long as there is any available partition, a process can be loaded into that partition. Because all partitions are of equal size, it does not matter which

partition is used. If all partitions are occupied with processes that are not ready to run, then one of these processes must be swapped out to make room for a new process. Which one to swap out is a scheduling decision; this topic is explored in Part Four.

With unequal-size partitions, there are two possible ways to assign processes to partitions. The simplest way is to assign each process to the smallest partition within which it will fit.1 In this case, a scheduling queue is needed for each partition, to hold swapped-out processes destined for that partition (Fig. 6.4a).



(a) One process queue per partition



(b) Single queue

Fig. 6.4: Memory Assignment for Fixed Partioning

The advantage of this approach is that processes are always assigned in such a way as to minimize wasted memory within a partition (internal fragmentation).

Although this technique seems optimum from the point of view of an individual partition, it is not optimum from the point of view of the system as a whole. In Figure6.4b, for example, consider a case in which there are no processes with a size between 12 and 16M at a certain point in time. In that case, the 16M partition will remain unused, even though some smaller process could have been assigned to it. Thus, a preferable approach would be to employ a single queue for all processes (Figure 6.3(b).When it is time to load a process into main memory, the smallest available partition that will hold the process is selected. If all partitions are occupied, then a swapping decision must be made. Preference might be given to swapping out of the smallest partition that will hold the incoming process. It is also possible to consider other factors, such as priority, and a preference for swapping out blocked processes versus ready processes.

The use of unequal-size partitions provides a degree of flexibility to fixed partitioning. In addition, it can be said that fixed-partitioning schemes are relatively simple and require minimal operating system software and processing overhead. However, there are disadvantages:

- The number of partitions specified at system generation time limits the number of active (not suspended) processes in the system.
- Because partition sizes are preset at system generation time, small jobs will not utilize partition space efficiently. In an environment where the main storage requirement of all jobs is known beforehand, this may be reasonable, but in most cases, it is an inefficient technique.

The use of fixed partitioning is almost unknown today. One example of a successful operating system that did use this technique was an early IBM mainframe operating system, OS/MFT (Multiprogramming with a Fixed Number of Tasks).

6.6 SWAPPING

A process must be in memory to be executed. A process, however, can be swapped temporarily out of memory to a **backing store** and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the memory space that has been freed (Fig. 6.5).

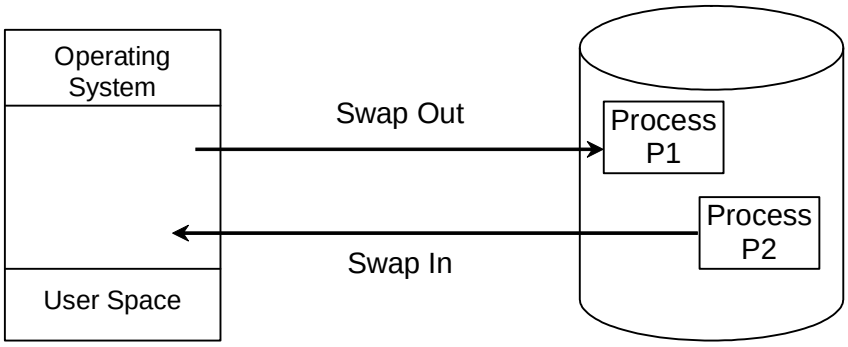


Fig. 6.5: Swapping of two processes using a disk as a backing store

In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. In addition, the quantum must be large enough to allow reasonable amounts of computing to be done between swaps.

A variant of this swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is sometimes called roll out, roll in.

Normally, a process that is swapped out will be swapped back into the same memory space it occupied previously. This restriction is dictated by the method of address binding. If binding is done at assembly or load time, then the process cannot be easily moved to a different location. If execution-time binding is being used, however, then a process can be swapped into a different memory space, because the physical addresses are computed during execution time.

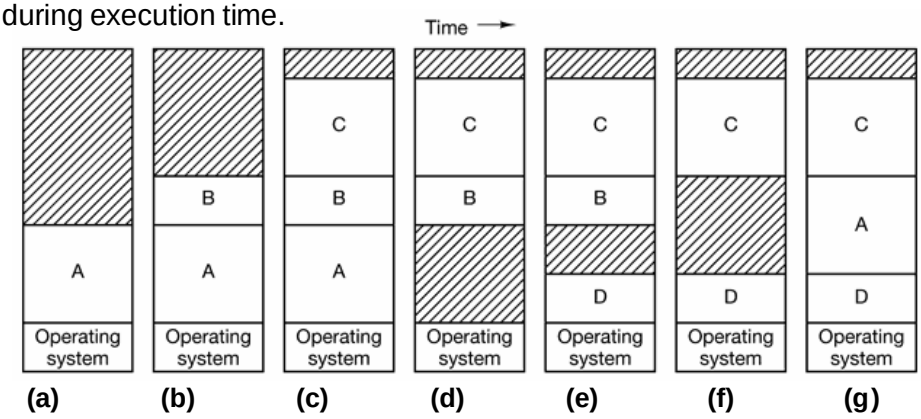


Figure 6.6: Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

The operation of a swapping system is illustrated in Fig. 6.6. Initially only process *A* is in memory. Then processes *B* and *C* are created or swapped in from disk. In Fig. 6.6(d) *A* is swapped out to disk. Then *D* comes in and *B* goes out. Finally *A* comes in again. Since *A* is now at a different location, addresses contained in it must be relocated, either by software when it is swapped in or (more likely) by hardware during program execution.

6.7 VIRTUAL MEMORY

Let's now move to the concept of virtual memory. It is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations. Before proceeding further let's know some terminologies.

Virtual Address: The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.

Virtual Address Space: The virtual storage assigned to a process.

Address Space: The range of memory addresses available to a process.

Real Address: The address of a storage location in main memory.

6.7.1 PAGING

Most virtual memory systems use a technique called paging. The use of paging to achieve virtual memory was first reported for the Atlas computer and soon came into widespread commercial use.

When a program uses an instruction like

```
MOV REG, 1000
```

it does this to copy the contents of memory address 1000 to REG (or vice versa, depending on the computer). Addresses can be generated using indexing, base registers, segment registers, and other ways.

These program-generated addresses are called virtual addresses and form the virtual address space. On computers without virtual memory, the virtual address is put directly onto the memory bus and causes the physical memory word with the same address to be read or written. When virtual memory is

used, the virtual addresses do not go directly to the memory bus. Instead, they go to an MMU (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses.

A very simple example of how this mapping works is shown in Fig. 6.7. In this example, we have a computer that can generate 16-bit addresses, from 0 up to 64K. These are the virtual addresses. This computer, however, has only 32 KB of physical memory, so although 64-KB programs can be written, they cannot be loaded into memory in their entirety and run. A complete copy of a program's core image, up to 64 KB, must be present on the disk, however, so that pieces can be brought in as needed.

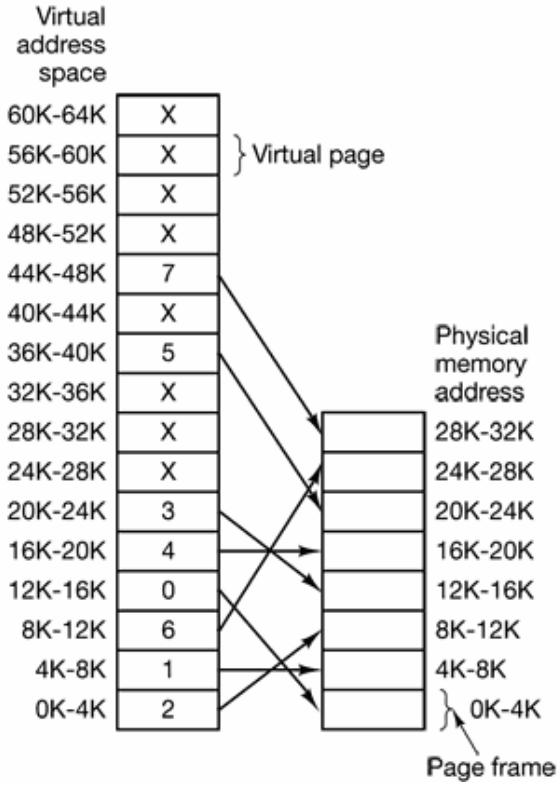


Fig.6.7: The relation between virtual addresses and physical memory addresses is given by the page table.

The virtual address space is divided up into units called pages. The corresponding units in the physical memory are called page frames. The pages and page frames are always the same size. In the above example they are 4 KB, but page sizes from 512 bytes to 64 KB have been used in real systems. With 64 KB of virtual address space and 32 KB of physical memory, we get 16 virtual pages and 8 page frames. Transfers between RAM and disk are always in units of a page.

When the program tries to access address 0, for example, using the instruction

MOV REG, 0

virtual address 0 is sent to the MMU. The MMU sees that this virtual address falls in page 0 (0 to 4095), which according to its mapping is page frame 2 (8192 to 12287). It thus transforms the address to 8192 and outputs address 8192 onto the bus. The memory knows nothing at all about the MMU and just sees a request for reading or writing address 8192, which it honors. Thus, the MMU has effectively mapped all virtual addresses between 0 and 4095 onto physical addresses 8192 to 12287.

Similarly, an instruction

MOV REG, 8192

is effectively transformed into

MOV REG, 24576

because virtual address 8192 is in virtual page 2 and this page is mapped onto physical page frame 6 (physical addresses 24576 to 28671). As a third example, virtual address 20500 is 20 bytes from the start of virtual page 5 (virtual addresses 20480 to 24575) and maps onto physical address 12288 + 20 = 12308.

By itself, this ability to map the 16 virtual pages onto any of the eight page frames by setting the MMU's map appropriately does not solve the problem that the virtual address space is larger than the physical memory. Since we have only eight physical page frames, only eight of the virtual pages in Fig. 6.7 are mapped onto physical memory. The others, shown as a cross in the figure, are not mapped. In the actual hardware, a Present/absent bit keeps track of which pages are physically present in memory.

What happens if the program tries to use an unmapped page, for example, by using the instruction

MOV REG, 32780

which is byte 12 within virtual page 8 (starting at 32768)? The MMU notices that the page is unmapped (indicated by a cross in the figure) and causes the CPU to trap to the operating system. This trap is called a page fault. The operating system picks a little-used page frame and writes its contents back to the disk. It then fetches the page just referenced into the page frame just freed, changes the map, and restarts the trapped instruction.

For example, if the operating system decided to evict page frame 1, it would load virtual page 8 at physical address 4K and make two changes to the MMU map. First, it would mark virtual page 1's entry as unmapped, to trap any future accesses to virtual addresses between 4K and 8K, Then it would replace the cross in virtual page 8's entry with a 1, so that when the trapped instruction is reexecuted, it will map virtual address 32780 onto physical address 4108.

Now let us look inside the MMU to see how it works and why we have chosen to use a page size that is a power of 2. In Fig. 6.8 we see an example of a virtual address, 8196 (0010000000000100 in binary), being mapped using the MMU map of Fig.6.7. The incoming 16-bit virtual address is split into a 4-bit page number and a 12-bit offset. With 4 bits for the page number, we can have 16 pages, and with 12 bits for the offset, we can address all 4096 bytes within a page.

The page number is used as an index into the page table, yielding the number of the page frame corresponding to that virtual page. If the *Present/absent* bit is 0, a trap to the operating system is caused. If the bit is 1, the page frame number found in the page table is copied to the high-order 3 bits of the output register, along with the 12-bit offset, which is copied unmodified from the incoming virtual address. Together they form a 15-bit physical address. The output register is then put onto the memory bus as the physical memory address.

6.7.2 PAGE TABLES

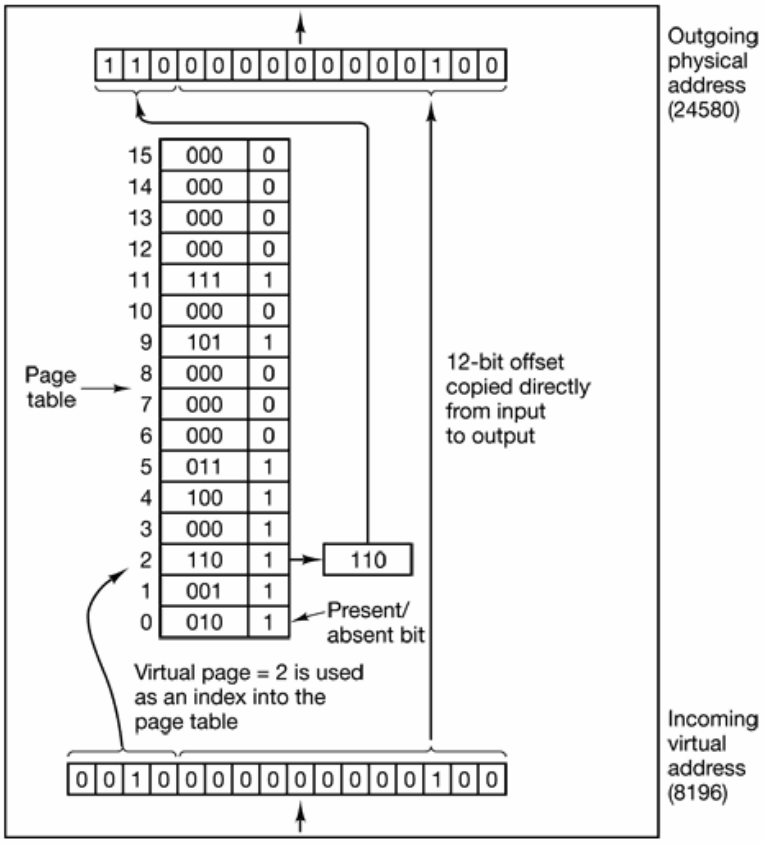


Fig. 6.8: The internal operation of the MMU with 16 4-KB pages

In the simplest case, the mapping of virtual addresses onto physical addresses is as we have just described it. The virtual address is split into a virtual page number (high-order bits) and an offset (low-order bits). For example, with a 16-bit address and a 4-KB page size, the upper 4 bits could specify one of the 16 virtual pages and the lower 12 bits would then specify the byte offset (0 to 4095) within the selected page. However a split with 3 or 5 or some other number of bits for the page is also possible. Different splits imply different page sizes.

The virtual page number is used as an index into the page table to find the entry for that virtual page. From the page table entry, the page frame number (it any) is found. The page frame number is attached to the high-order end of the offset, replacing the virtual page number, to form a physical address that can be sent to the memory.

The purpose of the page table is to map virtual pages onto page frames. Mathematically speaking, the page table is a function, with the virtual page number as argument and the physical frame number as result. Using the result of this function, the virtual page field in a virtual address can be replaced by a page frame field, thus forming a physical memory address.

Despite this simple description, two major issues must be faced:

1. The page table can be extremely large.
2. The mapping must be fast.

The first point follows from the fact that modern computers use virtual addresses of at least 32 bits. With, say, a 4-KB page size, a 32-bit address space has 1 million pages, and a 64-bit address space has more than you want to contemplate. With 1 million pages in the virtual address space, the page table must have 1 million entries. And remember that each process needs its own page table (because it has its own virtual address space).

The second point is a consequence of the fact that the virtual-to-physical mapping must be done on every memory reference. A typical instruction has an instruction word, and often a memory operand as well. Consequently, it is necessary to make 1, 2, or sometimes more page table references per instruction, If an instruction takes, say, 4 nsec, the page table lookup must be done in under 1 nsec to avoid becoming a major bottleneck.

The need for large, fast page mapping is a significant constraint on the way computers are built. Although the problem is most serious with top-of-the-line machines, it is also an issue at the low end as well, where cost and the

price/performance ratio are critical. In this section and the following ones, we will look at page table design in detail and show a number of hardware solutions that have been used in actual computers.

The simplest design (at least conceptually) is to have a single page table consisting of an array of fast hardware registers, with one entry for each virtual page, indexed by virtual page number, as shown in Fig.6.8. When a process is started up, the operating system loads the registers with the process' page table, taken from a copy kept in main memory. During process execution, no more memory references are needed for the page table. The advantages of this method are that it is straightforward and requires no memory references during mapping. A disadvantage is that it is potentially expensive (if the page table is large). Having to load the full page table at every context switch hurts performance.

At the other extreme, the page table can be entirely in main memory. All the hardware needs then a single register that points to the start of the page table. This design allows the memory map to be changed at a context switch by reloading one register. Of course, it has the disadvantage of requiring one or more memory references to read page table entries during the execution of each instruction. For this reason, this approach is rarely used in its most pure form, but below we will study some variations that have much better performance.

Structure of a Page Table Entry:

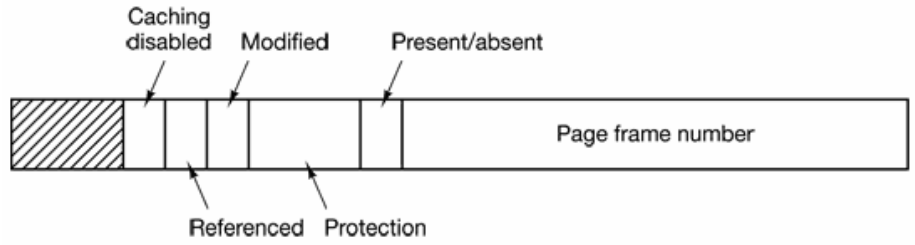


Fig. 6.9: A typical page table entry

The layout of a single page table entry is highly machine dependent, but the kind of information present is roughly the same from machine to machine. In Fig. 6.9 we give a sample page table entry. The size varies from computer to computer, but 32 bits is a common size.

The most important field is the *Page Frame Number*. The goal of the page mapping is to locate this value. Next to it we have the *Present/Absent* bit. If this bit is 1, the entry is valid and can be used. If it is 0, the virtual page to which the entry belongs is not currently in memory. Accessing a page table entry with this bit set to 0 causes a page fault.

The *Protection* bits tell what kinds of access are permitted. In the simplest form, this field contains 1 bit, with 0 for read/write and 1 for read only. A more sophisticated arrangement is having 3 bits, one bit each for enabling reading, writing, and executing the page.

The *Modified* and *Referenced* bits keep track of page usage. When a page is written to, the hardware automatically sets the *Modified* bit. This bit is of value when the operating system decides to reclaim a page frame. If the page in it has been modified (i.e., is “dirty”), it must be written back to the disk. If it has not been modified (i.e., is “clean”), it can just be abandoned, since the disk copy is still valid. The bit is sometimes called the dirty bit, since it reflects the page’s state.

The *Referenced* bit is set whenever a page is referenced, either for reading or writing. Its value is to help the operating system choose a page to evict when a page fault occurs.

6.8 PAGE REPLACEMENT ALGORITHMS

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date. If, however, the page has not been changed (e.g., it contains program text), the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted.

This is sometimes difficult to explain because several interrelated concepts are involved:

- How many page frames are to be allocated to each active process
- Whether the set of pages to be considered for replacement should be limited to those of the process that caused the page fault or encompass all the page frames in main memory
- Among the set of pages considered, which particular page should be selected for replacement

When page replacement is required, we must select the frames that are to be replaced. Designing appropriate algorithms to solve these problems is an important task, because disk I/O is so expensive.

There are certain basic algorithms that are used for the selection of a page to replace. Replacement algorithms that will discussed include

- Optimal
- Not Recently Used (NRU)
- Least Recently Used (LRU)
- First-In-First-Out (FIFO)

6.8.1 OPTIMAL

The optimal policy selects for replacement that page for which the time to the next reference is the longest. It can be shown that this policy results in the fewest number of page faults. Clearly, this policy is impossible to implement, because it would require the operating system to have perfect knowledge of future events. However, it does serve as a standard against which to judge real world algorithms.

Figure 6.10 gives an example of the optimal policy. The example assumes a fixed frame allocation (fixed resident set size) for this process of three frames. The execution of the process requires reference to five distinct pages. The page address stream formed by executing the program is

2 3 2 1 5 2 4 5 3 2 5 2

which means that the first page referenced is 2, the second page referenced is 3, and so on. The optimal policy produces three page faults after the frame allocation has been filled.

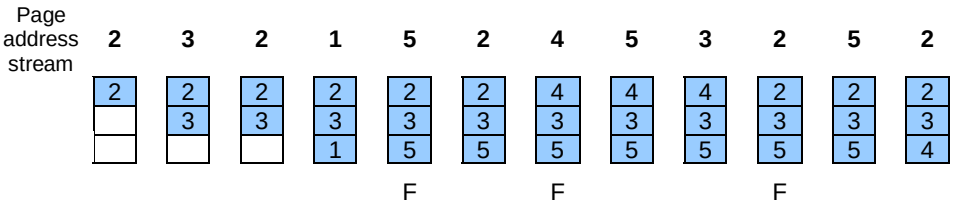


Fig. 6.10

6.8.2 NOT RECENTLY USED (NRU)

In order to allow the operating system to collect useful statistics about which pages are being used and which ones are not, most computers with virtual memory have two status bits associated with each page. *R* is set whenever the page is referenced (read or written). *M* is set when the page is written to (i.e., modified). The bits are contained in each page table entry, as shown in Fig.6.9. It is important to realize that these bits must be updated on every memory reference, so it is essential that they be set by the hardware. Once a bit has been set to 1, it stays 1 until the operating system resets it to 0 in software.

If the hardware does not have these bits, they can be simulated as follows. When a process is started up, all of its page table entries are marked as not in memory. As soon as any page is referenced, a page fault will occur. The operating system then sets the *R* bit (in its internal tables), changes the page table entry to point to the correct page, with mode READ ONLY, and restarts the instruction. If the page is subsequently written on, another page fault will occur, allowing the operating system to set the *M* bit and change the page's mode to READ/WRITE

The *R* and *M* bits can be used to build a simple paging algorithm as follows. When a process is started up, both page bits for all its pages are set to 0 by the operating system. Periodically (e.g., on each clock interrupt), the *R* bit is cleared, to distinguish pages that have not been referenced recently from those that have been.

When a page fault occurs, the operating system inspects all the pages and divides them into four categories based on the current values of their *R* and *M* bits:

Class 0: not referenced, not modified.
Class 1: not referenced, modified.
Class 2: referenced, not modified.
Class 3: referenced, modified.

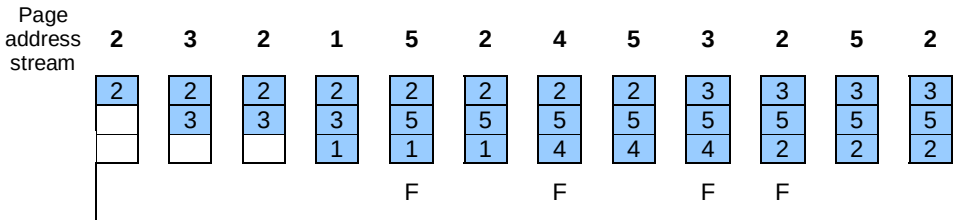
Although class 1 pages seem, at first glance, impossible, they occur when a class 3 page has its *R* bit cleared by a clock interrupt. Clock interrupts do not clear the *M* bit because this information is needed to know whether the page has to be rewritten to disk or not. Clearing *R* but not *M* leads to a class 1 page.

The NRU (Not Recently Used) algorithm removes a page at random from the lowest numbered nonempty class. Implicit in this algorithm is that it is

better to remove a modified page that has not been referenced in at least one dock tick (typically 20 msec) than a clean page that is in heavy use. The main attraction of NRU is that it is easy to understand, moderately efficient to implement, and gives a performance that, while certainly not optimal, may be adequate.

6.8.3 LEAST RECENTLY USED (LRU)

The least recently used (LRU) policy replaces the page in memory that has not been referenced for the longest time. By the principle of locality, this should be the page least likely to be referenced in the near future. And, in fact, the LRU policy does nearly as well as the optimal policy. The problem with this approach is the difficulty in implementation. One approach would be to tag each page with the time of its last reference; this would have to be done at each memory reference, both instruction and data. Even if the hardware would support such a scheme, the overhead would be tremendous. Alternatively, one could maintain a stack of page references, again an expensive prospect.



F = page fault occurring after the frame allocation is initially filled

Fig. 6.11

Fig.6.11 shows an example of the behavior of LRU, using the same page address stream as for the optimal policy example. In this example, there are four page faults.

6.8.4 FIRST-IN-FIRST-OUT (FIFO)

The first-in-first-out (FIFO) policy treats the page frames allocated to a process as a circular buffer, and pages are removed in round-robin style. All that is required is a pointer that circles through the page frames of the process. This is therefore one of the simplest page replacement policies to implement. The logic behind this choice, other than its simplicity, is that one is replacing the page that has been in memory the longest: A page fetched into memory a long time ago may have now fallen out of use. This reasoning

will often be wrong, because there will often be regions of program or data that are heavily used throughout the life of a program. Those pages will be repeatedly paged in and out by the FIFO algorithm.

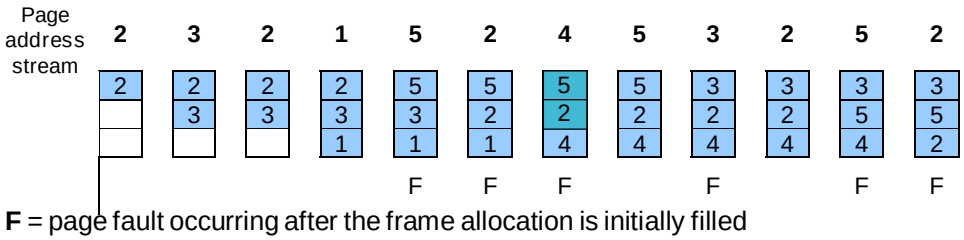


Fig. 6.12

Continuing the example in Figure 6.12, the FIFO policy results in six page faults. Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

CHECK YOUR PROGRESS

1. State whether the following statements are true(T) or false(F):

a) In a *uniprogramming system*, *main memory* is divided into two parts: one part for the *operating system* (*resident monitor, kernel*) and one part for the *program* currently being *executed*.

b) Registers that are built into the CPU are generally accessible within two cycle of the CPU clock.

c) Branch instructions contain an address to reference the instruction to be executed next.

d) The principal operation of memory management is to bring processes into main memory for execution by the processor.

e) Virtual Address is the address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.

f) The *Protection* bits tell what kinds of access are permitted.

g) Not Recently Used is not a Page Replacement Algorithm.

2. Answer the following by selecting the appropriate option:

a) The memory management requirements are

(i) Relocation.

(ii) Protection.

(iii) Sharing.

(iv) All of the above.

b) In a multiprogramming system, the available main memory is generally
   (i) shared among a number of processes.
   (ii) allocated to a single process among the processes.
   (iii) Both (i) and (ii).
   (iv) None of the above.

c) The Page Replacement Algorithms are
   (i) Optimal.
   (ii) Least Recently Used.
   (iii) First-in-First-out.
   (iv) All of the above.

d) Real Address means
   (i) The address assigned to a process.
   (ii) The range of memory addresses available to a process.
   (iii) The address of a storage location in main memory.
   (iv) None of the above.

e) *Protection* bits
   (i) tell what kinds of access are permitted.
   (ii) keep track of page usage.
   (iii) Both (i) and (ii).
   (iv) None of the above.

6.9 LET US SUM UP

One of the most important and complex tasks of an operating system is memory management. Memory management involves treating main memory as a resource to be allocated to and shared among a number of active processes. To use the processor and the I/O facilities efficiently, it is desirable to maintain as many processes in main memory as possible. In addition, it is desirable to free programmers from size restrictions in program development.

The basic tools of memory management are paging and segmentation (not included in this unit). With paging, each process is divided into relatively small, fixed-size pages.

The way to address both of these concerns is virtual memory. With virtual memory, all address references are logical references that are translated at run time to real addresses. This allows a process to be located anywhere

in main memory and for that location to change over time. Virtual memory also allows a process to be broken up into pieces. These pieces need not be contiguously located in main memory during execution and, indeed, it is not even necessary for all of the pieces of the process to be in main memory during execution.

Two basic approaches to providing virtual memory are paging and segmentation (not included in this unit). With paging, each process is divided into relatively small, fixed-size pages.

6. 10 ANSWERS TO CHECK YOUR PROGRESS

1. (a) True, (b) False, (c) True, (d) True,
   (e) True, (f) True, (g) False.

2. (a) (iv), (b) (i), (c) (iv), (d) (iii), (e) (i).

6.11 FURTHER READINGS

1. *"Operating Systems, Design and Implementations",*
   - Andrew. S. Tanenbaum & Albert S. Woodhull
   Prentice-Hall India
2. *"Operating System"*,
   - Deitel and Deitel, Choffnes
   Pearson Education

6.12 MODEL QUESTIONS

1. What do you mean by Relocation? Explain.
2. What is Memory Partitioning? Describe a Memory Portioning Technique.
3. What is Virtual Memory? Define the terms – Virtual Address, Virtual Address Space, Address Space, Real Address.
4. Draw and explain the structure of a typical Page Table entry.
5. What is/are the difference(s) between Not Recently Used and Least Recently Used Page Replacement Algorithms?

*****

UNIT- 7 FILE SYSTEM

UNIT STRUCTURE

7.1 LEARNING OBJECTIVES

After going through this unit, you will able to :

- define a file and identify its attributes
- understand type of files like regular, directory and device files
- describe the operations on files and directories
- define the structure of the directory system
- describe file system implementation techniques

7.2 INTRODUCTION

In computing, a file system is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. File systems may use a data storage device such as a hard disk or CD-ROM and involve maintaining the physical location of the files, they might provide access to data on a file server by acting as clients for a network protocol e.g.NFS, or they may be virtual and exist only as an access method for virtual data. It is distinguished from a directory service and registry.

Protocol :
Protocols are rules determining the format and transmission of data.

More formally, a file system is a special-purpose database for the storage, organization, manipulation, and retrieval of data.Most file systems make use of an underlying data storage device that offers access to an array of fixed-size blocks, sometimes called sectors. The file system software is responsible for organizing these sectors into files and directories, and keeping track of which sectors belong to which file and which are not being used. Most file systems address data in fixed-sized units called "clusters" or "blocks" which contain a certain number of disk sectors. This is the smallest amount of disk space that can be allocated to hold a file.

7.3 FILE AND FILE NAMING

File is a collection of data or information that has a name called the filename. Almost all information stored in a computer must be in a file. There are many different types of files: data files, text files ,program files, directory files, and so on. Different types of files store different types of information. For example, program files store programs, whereas text files store text.

String : An orderly arrangement of characters.

A **filename** is a special kind of string used to uniquely identify a file stored on the file system of a computer. Some operating systems also identify directories in the same way. Different operating systems impose different restrictions on length and allowed characters on filenames. A filename includes one or more of these components:

- **protocol** or **scheme** — access method e.g., http, ftp, file etc.
- **host** or **network-ID** — host name, IP address, domain name, or LAN network name
- **device** or **node** — port, socket, drive, root mountpoint, disc.
- **directory** or **path** — directory tree e.g., /usr/bin
- **file** — base name of the file
- **type** format or **extension** — indicates the content type of the file e.g., .txt, .exe, .COM, etc.
- **version** — revision number of the file

To refer to a file on a remote computer (host, server), the remote computer must be known. The remote computer name or address might be part of the file name, or it might be specified at the time a file system is “mounted”, in which case it won’t necessarily be part of the file name.In some systems, if a filename does not contain a path part, the file is assumed to be in the current working directory.

Many operating systems, including MS-DOS, Microsoft Windows, and VMS systems, allow a filename extension that consists of one or more characters following the last period in the filename, thus dividing the filename into two parts: the **basename** (the primary filename) and the **extension** (usually indicating the file type associated with a certain file format). On Unix-like systems, files will often have an extension for example *prog.c*, denoting the C-language source code of a program called "prog" ,but since the extension is not considered a separate part of the filename, a file on a Unix system which allows 14-character filenames, and with a filename which uses *"."* as an "extension separator" or "delimiter", could possibly have a filename such as *a.longxtension*

Within a single directory, filenames must be unique. Since filename also applies for subdirectories, it is also not possible to create equally named file and subdirectory entries in a single directory. However, two files in different directories may have the same name. In some operating systems, such as MS-DOS, Microsoft Windows, and classic Mac OS, upper-case letters and lower-case letters in file names are considered the same, so that, for example, the file names "MyName" and "myname" would be considered the same, and a directory could not contain a file with the name "MyName" and another file with the name "myname". The file systems in those operating systems are called "case-insensitive".

Reserved characters and words

Many operating systems prohibit control characters from appearing in file names. For example, DOS and early Windows systems require files to follow the 8.3 filename convention. Some operating systems prohibit some particular characters from appearing in file names:

In Windows the space and the period are not allowed as the final character of a filename. The period is allowed as the first character, but certain Windows applications, such as Windows Explorer, forbid creating or renaming such files despite this convention being used in Unix-like systems to describe hidden files and directories.

Some file systems on a given operating system especially file systems originally implemented on other operating systems, and particular applications on that operating system, may apply further restrictions and interpretations.

In Unix-like systems, MS-DOS, and Windows, the file names "." and ".." have special meanings current and parent directory respectively.

In addition, in Windows and DOS, some words might also be reserved and can not be used as filenames. For example, DOS Device file:CON, PRN, AUX, CLOCK$, NUL, COM0 to COM9, LPT0 toLPT9.

7.4 FILE TYPES

We have understood till this point that what is a file and the naming conventions for a file in different operating systems.Because files can have different purposes, they have different types. The file type is best identified by its file structure. For example a text file would have a very different structure than a file than can be executed. An executable file must have a specific structure to be able to be run. The file structure is used to determine its MIME type. The word MIME stands for multipurpose internet mail extension and is used as a standard to identify various file types.

Regular files

A simply definition of a regular file would be that it is a one dimensional assortment of bytes that are stored on a disk or other mass storage devices. A program that uses a file needs to know the structure of the file and needs to interpret the file contents. This is because no structure is imposed on the contents of a file by the operating system itself. This is a very powerful feature as it means that you could work with any file that you need to work on e.g. a DOS file. Files are presented to the application as a stream of bytes and then an EOF condition. However the EOF condition is not typed in, it is merely that the stream of bytes is as long as the file size is and then it is at the end. In other words it is actually a sort of offset that will happen when a program attempts to access an offset larger than the file size itself. There are many different types of regular files, text, binary, executable etc. When using the **file** command to establish a file type the command accesses the magic database. If you get a chance have a look at the magic file and see how many different types of files Linux could support. A regular file is referenced by an inode number.

Directory files

A simple definition of a directory is that it is a file that provides a mapping mechanism between the names of files and the files themselves. A directory is also called a file. Its purpose is really to make sure that there is a good structure maintained on the system - sort of like a good filing

system. The directory only holds inode numbers and filenames. Yet this is also vitally important as this is the only place where a filename is referenced by its inode. If you delete a file from a directory the entry in the list is zeroed and this is then called a shadow inode. The inode is then freed up.

Device files

A device file refers to a device driver and these are important to the kernel. The device drivers are written in C and compiled to make an object file and then placed as part of the kernel. Created a device file using the mknod command. The files in **/dev** are used to ensure that we can access hardware such as the printer, cdrom, network etc. If you look at the way Linux uses a device driver, it handles many of the functions that we could compare to the way DOS uses the BIOS. However the differences are often the reason why a piece of hardware that would work with a DOS related system will not work with a Unix or Linux related system. Here we can read and write directly to the device, so the user issues a system call to a device, the kernel performs a successful open on that device, if busy the read/write routine cannot operate, if not busy then reads or writes directly to that device. There are different types of device files:

Character device files - writes to and from the device a character at a time. Indicated by a “c” in the first field. Very little preliminary processing required by the kernel so the request is passed directly to the device.

A block device only receives a request once block buffering has taken place in the kernel. Indicated by a “b” in the first field. A filesystem is an example of a block buffering device. Block devices generally have an associated character device - for example if you format a diskette you would use the character device file name, if backing up to that diskette you would use the block device name where the blocking is handled in the kernel buffer cache.

Major and Minor Device Numbers

Each device is referenced by numbers, which are read when the kernel needs to use a device and subsequent device driver. These numbers are called the major and minor device numbers.The major device number refers to the device driver that should be used to access the device and it is a unique number, whereas the minor device number points to the device itself.

7.5 SEQUENTIAL ACCESS AND RANDOM ACCESS FILE

There are several ways that the information in the file can be accessed. Some systems provide only one access method for files. On other systems, many different access methods are supported, and choosing the right one for a particular application is a major design problem.

Reasons for introducing access methods

Without access methods, a programmer must write a special program for an I/O channel, a processor dedicated to control peripheral storage device access and data transfer to and from main memory. This processor requires programs written with special instructions, called Channel Command Words (CCWs). Programming those is a complex and arduous task. Channel programs are kicked off from mainframe programs by a "startIO" instruction issued by the operating system - this is usually front ended by the *Execute Channel Program* (EXCP) macro for application programmer convenience. This macro issues an supervisor call instruction that asks the operating system to issue the startIO on the application's behalf.

Access methods provide:

- Ease of programming - programmer would no longer deal with a specific device procedures, including error detection and recovery tactics in each and every program. A program designed to process a sequence of 80-character records would work no matter where the data are stored.
- Ease of hardware replacement - programmer would no longer alter a program when data should be migrated to newer model of storage device, provided it supports the same access methods.
- Easy shared data set access - an access method is a trusted program, that allows multiple programs to access the same file, while ensuring the basic data integrity.

We will be discussing two methods of accessing a file :

Sequential Access

Information in the file is processed in order, one record after the other. This is by far the most common mode of access of files. For example, computer editors usually access files in this fashion. A read operation reads

the next portion of the file and automatically advances the file pointer. Similarly, a write appends to the end of the file and the file pointer. Similarly, a write appends to the end of the file and the file pointer. Similarly, a write appends to the end of the end of the file and advances to the end of the newly written material ,the new end of file. Such a file can be reset to the beginning, and, on some systems, a program may be able to skip forward or backward n records, for some integer n. This scheme is known as sequential access to a file. Sequential access is based on a tape model of a file.

In data structures, a data structure is said to have sequential access if one can only visit the values it contains in one particular order. The canonical example is the linked list. Indexing into a list which has sequential access requires O(*k*) time, where *k* is the index. As a result, many algorithms such as quicksort and binary search degenerate into bad algorithms that are even less efficient than their naïve alternatives; these algorithms are impractical without random access. On the other hand, some algorithms, typically those which don't perform indexing, require only sequential access, such as mergesort, and so face no penalty.

Random access

In computer science, **random access** sometimes called **direct access** is the ability to access an arbitrary element of a sequence in equal time. The opposite is sequential access, where a remote element takes longer time to access. A typical illustration of this distinction is to compare an ancient scroll (sequential; all material prior to the data needed must be unrolled) and the book (random: can be immediately flipped open to any random page. A more modern example is a cassette tape is sequential you have to fast-forward through earlier songs to get to later ones and a compact disc is random access ,you can jump right to the track you want.

In data structures, random access implies the ability to access the *N*th entry in a list of numbers in constant time. Very few data structures can guarantee this, other than arrays and related structures like dynamic arrays. Random access is critical to many algorithms such as quicksort and binary search. Other data structures, such as linked lists, sacrifice random access to make for efficient inserts, deletes, or reordering of data.

7.6 FILE ATTRIBUTES

A **file attribute** is metadata that describes or is associated with a

computer file. For example, an operating system often keeps track of date a file was created and last modified, as well as the file's size and extension . File permissions are also kept track of. The user may attach other attributes themselves, such as comments or color labels.

Each file is stored in a directory, and uses a directory entry that describes its characteristics such as its name and size. The directory entry also contains a pointer to where the file is stored on disk. One of the characteristics stored for each file is a set of *file attributes* that give the operating system and application software more information about the file and how it is intended to be used.

The use of attributes is "voluntary". What this means is that any software program can look in the directory entry to discern the attributes of a file, and based on them, make intelligent decisions about how to treat the file. For example, a file management program's delete utility, seeing a file marked as a read-only system file, would be well-advised to at least warn the user before deleting it. However, it doesn't have to.

A file can have more than one attribute attached to it, although only certain combinations really make any sense. The attributes are stored in a single byte, with each bit of the byte representing a specific attribute ,actually, only six bits are used of the eight in the byte. Each bit that is set to a one means that the file has that attribute turned on. These are sometimes called *attribute bits* or *attribute flags*. The following are the attributes and the bits they use in the attribute byte:

Attribute	**Bit Code**
Read-Only	00000001
Hidden	00000010
System	00000100
Volume Label	00001000
Directory	00010000

Here is the more detailed description about the above attribute flags.

Read-Only: Most software, when seeing a file marked read-only, will refuse to delete or modify it. This is pretty straight-forward. For example, DOS will say "Access denied" if you try to delete a read-only file. On the other hand,

Windows Explorer will happily munch it.

Hidden: This one is pretty self-explanatory as well; if the file is marked hidden then under normal circumstances it is hidden from view. DOS will not display the file when you type “DIR” unless a special flag is used, as shown in the earlier example.

System: This flag is used to tag important files that are used by the system and should not be altered or removed from the disk. In essence, this is like a “more serious” read-only flag and is for the most part treated in this manner.

Volume Label: Every disk volume can be assigned an identifying label, either when it is formatted, or later through various tools such as the DOS command “LABEL”. The volume label is stored in the root directory as a file entry with the label attribute set.

Directory: This is the bit that differentiates between entries that describe files and those that describe subdirectories within the current directory. In theory you can convert a file to a directory by changing this bit.

Archive: This is a special bit that is used as a “communications link” between software applications that modify files, and those that are used for backup. Most backup software allows the user to do an incremental backup, which only selects for backup any files that have changed since the last backup. This bit is used for this purpose. When the backup software backs up (“archives”) the file, it clears the archive bit makes it zero. Any software that modifies the file subsequently, is supposed to set the archive bit. Then, the next time that the backup software is run, it knows by looking at the archive bits which files have been modified, and therefore which need to be backed up.

7.7 OPERATIONS ON FILE

A file is an abstract data type. To define a file properly, we need to consider the operations that can be performed on files. The operating system can provide system calls to create, write, read, reposition, delete and truncate files.Lets examine these basic operations.

Creating a file. Two steps are needed to create a file.First space in the file system must be found for the file.Second an entry for the new file must be made in the directory.

Writing a file. To write a file , we make a system call specifying both the name of the file and the information to be written to the file.Given the name of the file, the system searches the directoryto find the files location.The system must keep a write pointer to the location in the filewhere the next write is to take place.The write pointer must be updated whenever a write occurs.

Reading a file. To read a file , we make a system call that specifies the name of the file and where the next block of the file should be put. Again the directory is searched for the associated entry and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place , the read pointer is updated.Both read and write operations use current file position pointer for saving space and reducing system complexity.

Repositioning within a file. The directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value.Reposition within a file need not involve any actual I/O. This file operation is also known as file seek.

Deleting a file. To delete a file , we search the directory for the named file. having found the associated directory entry, we release all file space, so that it can be reused by other files and erase the directory entry.

Truncating a file. The user may want to erase the contents of a file but keep its attributes.This function allows allattributes to remain unchanged except for file length.

Other common operations include appending a new information to the end of a existing file and renaming an existing file.These primitive operation can be combined to perform other file operations. For instance we can create a copy of a file, or copy the file to another I/O device such as printer or display.

Most of the file operations mentioned involves searching the directory for the entry associated with the named file. To avoid this constant searching, many systems require that an open() system call be made before a file is first used actively.The OS keeps a small table called the **open file table** containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. Create and delete are system calls that work with closed rather than open

files.

The implementation of open() and close() operations is more complicated in an environment where several process may open the file at the same time.Typically the OS uses two levels of internal tables: a per process table and system wide table. Each entry in the per process table in turn points to a system wide open file table. The system wide table contains process independent information such as the location of the file on disk, access dates and file size.Once a file has been opened by one process, the system wide table includes an entry for file.When another process executes an open call ,a new entry is simply added to the process's open file table pointing to the appropriate entry in the system wide table.

Several pieces of information are associated with an open file.

File pointer.On system that do not include a file offset as a part of read() and write() system calls , the system must track the last read write location as a current file position pointer.

File open count. The file open counter tracks the number of opens and closes and reaches zero on the last close. The system then can remove the entry.

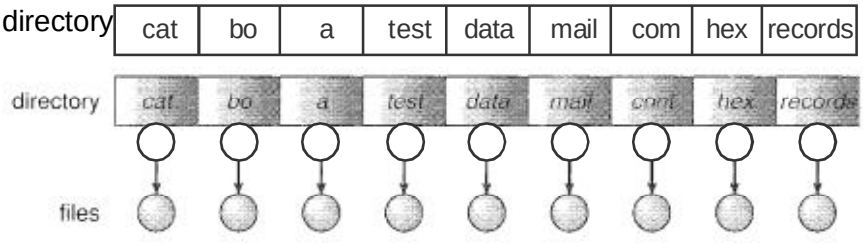
Disk location on the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

Access rights. This information is stored on the per process table so that OS can allow or deny subsequent I/O requests.

7.8 HIERARCHICAL DIRECTORY STRUCTURE AND PATH NAMES

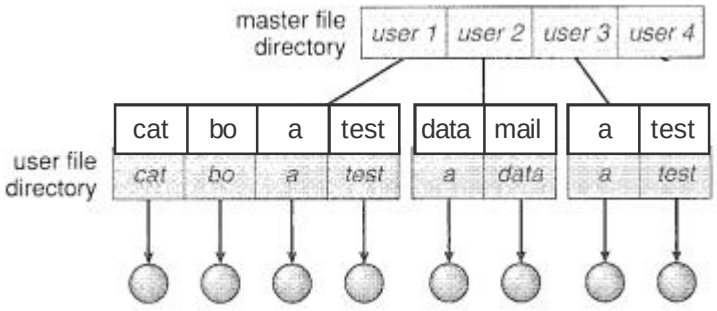
In this section we will be discussing some of the logical structures of a directory.

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.The diagram below shows the structure of single level directory



A single level directory has significant limitations, however when the number of files increases or when the system more than one user. Since all files are in the same directory, they must have unique names.Although filenames are generally selected to reflect the content of the file, they are often limited in length, complicating the task of making file names unique. MS-DOS allows only 11 character file names; UNIX in contrast allows 255 characters.

Two-Level Directory. In the two-level directory system, the system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users.The diagram below shows two-level directory structure.

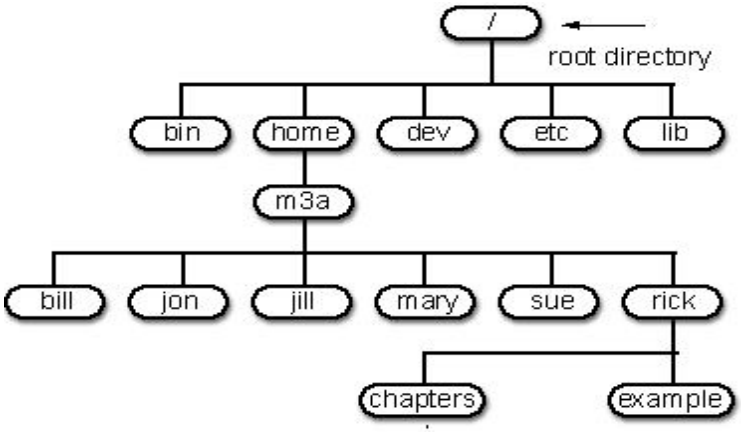


There are still problems with two-level directory structure. This structure effectively isolates one user from another. This is an advantage when the users are completely independent, but a disadvantage when the users want to cooperate on some task and access files of other users. Some systems simply do not allow local files to be accessed by other users.

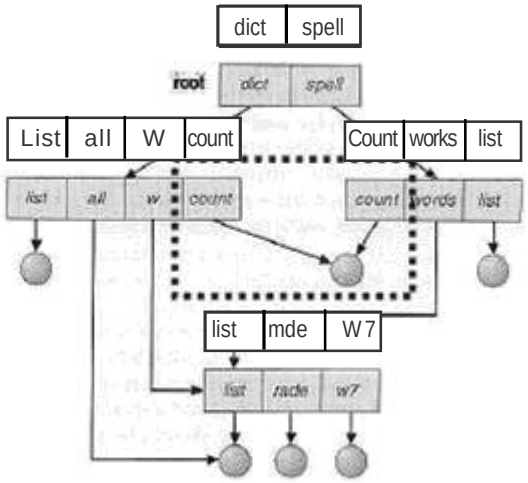
Tree-Structured Directories. In the tree-structured directory, the directory themselves are files. This leads to the possibility of having sub-directories that can contain files and sub-subdirectories.An interesting policy decision in a tree-structured directory structure is how to handle the deletion of a directory. If a directory is empty, its entry in its containing directory can simply be deleted. However, suppose the directory to be deleted id not empty, but contains several files, or possibly sub-directories. Some systems will not delete a directory unless it is empty.

Thus, to delete a directory, someone must first delete all the files in that directory. If these are any sub-directories, this procedure must be applied recursively to them, so that they can be deleted also. This approach may result in a insubstantial amount of work. An alternative approach is just to assume that, when a request is made to delete a directory, all of that directory's files and sub-directories are also to be deleted. The diagram is

given in the next page.



Acyclic Graph Directories. The acyclic directory structure is an extension of the tree-structured directory structure. In the tree-structured directory, files and directories starting from some fixed directory are owned by one particular user. In the acyclic structure, this prohibition is taken out and thus a directory or file under directory can be owned by several users.Here is the representation of Acyclic Graph Directory structure.



Path Names. A path, the general form of a filename or of a directory name, specifies a unique location in a file system. A path points to a file system location by following the directory tree hierarchy expressed in a string of characters in which path components are separated by a delimiting character, represent each directory. The delimiting character is most commonly the slash ("/"), the backslash character ("\"), or colon (":"), though some operating systems may use a different delimiter. Paths are used extensively in computer science to represent the directory and file

relationships common in modern operating systems, and are essential in the construction of Uniform Resource Locators (URLs).

Systems can use either absolute or relative paths. A full path or **absolute path** is a path that points to the same location on one file system regardless of the working directory or combined paths. It is usually written in reference to a root directory.

A **relative path** is a path relative to the working directory of the user or application, so the full absolute path may not need to be given.

7.9 OPERATION ON DIRECTORIES

The directory can be viewed as a symbol table that translates filename into their directory entries. When considering a particular directory, we need to keep in mind the operations that are to be performed on a directory.

Search for a file. Since files have symbolic names and similar names may indicate a relationship between files , we may want to be able to find all files whose names match a particular pattern.

Create a file. New files need to be created and added to the directory.

Delete a file. When a file is no longer needed, we should be able to remove it from the directory.

Rename a file. Because the name of a file represents its content to its users, we must be able to change the name when the contents or use of the file changes.

Traverse the file system. It is a good idea to save the contents and structure of the entire file system at regular intervals. This technique provides a backup copy in case of a system failure.

7.10 FILE SYSTEM IMPLEMENTATION TECHNIQUES

The file system provides the mechanism for online storage and access of file contents, including data and programs.The file system resides permanently on secondary storage, which is designed to hold a large amount of data permanently. These disks have two characteristics that make them a convenient medium for storage of multiple files:

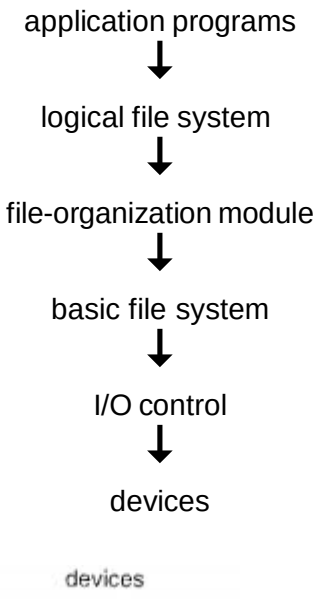
1. A disk can be rewritten in place, it is possible to read a block from the disk, modify the block and write it back into the same place.
2. From a disk , it is easy to access any file sequentially or randomly. And

switching from one file to another requires just moving the R/W head and waiting for the disk to rotate.

To provide efficient and convenient access to disk, the OS imposes one or more file systems to allow the data to be stored, located and retrieved easily. A file system poses two different design problems.The first problem is how the file system should look to the user. This task involves defining a file and its attributes, the operation allowed on files, and the directory structure for organizing the files. The second problem is creating algorithm and data structures to map the logical file system onto the physical secondary storage devices.

The file system itself is composed of many different levels. Below is the layered design of a file system.

Each level in the design uses the features of lower levels to create new features for use by higher levels. At the lowest level the **I/O control** , consists of device drivers and interrupt handlers to transfer information between main memory and disk system. A device driver can be thought of as a translator. Its input consists of high level commands and output consists of low level hardware specific instructions that are used by the hardware



devices

The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address, e.g drive1, cylinder 45, track 17 and sector 2.

The **file organization module** knows about files and their logical blocks as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file organization module can translate the logical block address to physical block address for the basic file system to transfer. Each file's logical block is numbered from 0 or 1 through N. Since the physical blocks containing the data do not match the logical numbers, a translation is needed to locate each block. It also includes the free space manager , which tracks unallocated blocks and provides these blocks to the file organization module when requested.

Finally the **logical file system** maintains the metadata information. Metadata includes all of the file system structure except the actual data. The logical file system maintains the directory structure to provide the file organization module with the information the latter needs, given a symbolic file name. It maintains file structure via file control blocks. A **file control block** contains information about the file, including ownership, permissions and location of the file contents.

When layered structure is used for file system implementation, duplication of code is minimized. The I/O control and sometimes the basic file system code can be used by multiple file systems. Each file system can then have its own logical file system and file organization modules. Many file system are in use today. For example most CD ROMS are written in the ISO 9660 format, a standard format agreed on by CD ROM manufacturers. UNIX uses UNIX file system (UFS) which is based on the Berkeley fast File System (FFS).

CHECK YOUR PROGRESS

1. DOS and Windows systems require files to follows
   a) 8.1 filename convention
   b) 8.2 filename convention
   c) 8.3 filename convention
   d) 8.4 filename convention
2. Which of the following can be used as filename in Window
   a) CON
   b) LPT10

c) PRN
d) AUX

3. Which of the following is not a file attribute
   a) System
   b) Archive
   c) Volume label
   d) Type

4. OS employs two tables , when many applications open same file at same time are:
   a) pre process & signal wide table
   b) post process & system wide table
   c) pre process & post wide table
   d) pre process & system wide table

5. In UNIX a single dot and double dot represents
   a) current and parent directory.
   b) parent and current directory
   c) child and parent directory
   d) parent and child directory

6. FCB stands for
   a) File System Control Block.
   b) File Control Block.
   c) File Count Block.
   d) File Control Blog.

7.11 LET US SUM UP

A collection of data or information that has a name is called the filename. There are many different types of **files**: data files, text files , program files, directory files, and so on.

A **filename** is a special kind of string used to uniquely identify a file stored on the file system of a computer.

Regular files is a one dimensional assortment of bytes that are stored on a disk or other mass storage devices.

Directory file is a file that provides a mapping mechanism between the names of files and the files (data blocks) themselves.

A **device file** refers to a device driver and these are important to the kernel.

Sequential Access: Information in the file is processed in order, one record after the other.

Random access sometimes called **direct access** is the ability to access an arbitrary element of a sequence in equal time.

File attribute is metadata that describes or is associated with a computer file.

Operations on File: Creating, writing, reading, repositioning, deleting, truncating

Operations on Directories: Search for a file, create a file, delete a file, list a directory, rename a file, traverse the file system

Single level directory: All file are contained in the same directory. Since all file are in the same directory they should have unique names.

Two level directory: In this each user has his own user file directory(UFD).The UFDs have similar structures, but each lists only the files of a single user.**Tree structured directories**: A tree is the most common directory structure. The tree has a root directory and every file in the system has a unique path. A directory contains a set of files or subdirectories.

To provide efficient and convenient access to disk, the operating system imposes one or more **file systems** to allow the data to be stored, located, and retrieved easily.

7.12 ANSWERS TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS

1. c, 2. b, 3. d, 4. d, 5. a , 6.b.

7.13 FURTHER READINGS

1. Modern Operating Systems by Andrew S. Tanenbaum
2. Guide to Operating Systems by Michael Palmer, Michael Walter
3. Operating Systems by H.M.Deitel
4. Operating System Concepts by Silberschatz and Galvin
5. Operating System Concepts, Seventh Edition, by John Wiley.

7.14 MODEL QUESTIONS

Q1. Define a File. Also discuss the rules for naming a file. Take a suitable example to explain.

Q2. What do you mean by reserving characters and words for a given operating system file naming?

Q3. What do you understand by File Type? Compare and contrast directory, regular and device files.

Q4. What are the different types of access methods for a file? Also mention why do we need different access methods?

Q5. Differentiate between Sequential and random access file.

Q6 Explain attribute of a file.

Q7. What is a file pointer? Briefly explain the various operation of a File.

Q8. What is role of a directory structure? And what are the operations that are being performed on a directory.

Q9. Differentiate between single level and two level directory structures.

Q10. Explain the concept of tree structured directory.

Q11. Explain layered File System.

*****

UNIT 8 : INPUT/OUTPUT MANAGEMENT

UNIT STRUCTURE

8.1 INTRODUCTION

One of the main function of an operating system is input/output management. The operating system controls the I/O devices, catch interrupts, and handle errors by writing some commands. It should also provide an interface between the I/O devices and the other parts of the system which is simple and easy to use. Humans interact with machines by providing information through IO devices and computer system provides as on-line services is essentially made available through specialized devices such as screen displays, printers, keyboards, mouse, etc. So, management of all these devices can affect the throughput of a system. In this unit we shall examine the role of operating systems in managing IO devices.

8.2 OBJECTIVES

After going through this unit, you will be able to :

- learn about the basic principles of I/O hardware
- describe types of I/O devices
- learn about device controller
- learn about direct memory access
- illustrate the principles of I/O software
- describe interrupt handler
- learn the concept of device driver
- learn about device independent I/O software
- describe user-space I/O Software

8.3 BASIC PRINCIPLES OF I/O HARDWARE

Today's computers contain various types of hardware .The operating system is called a resource manager because it manages the different hardware or resources of a computer. As the technology is growing rapidly, the hardware of computer system are changing day by day for which the computing speeds has increased. For this changing world of computer hardware, the operating system became more complex. Now the operating systems are written to be independent of a system's particular hardware configuration to provide improve extensibility .Input output devices are the one of the important resources of a computer system. The I/O system consists of I/O devices, Device controllers and communication strategies. Communication strategies can be categories as Memory-mapped I/O, controller registers, Direct Memory Access – DMA.

8.3.1 I/O DEVICES

The job of the Input output devices is to establish the communication between a computer, and the outside world i.e. a human, or another information processing system. There are different input output devices available for user like keyboard, mouse,

modems, network cards, disk etc. Input output devices are divided into two categories: block devices and character devices. The devices which are used to store information in some fixed-size blocks are called block devices. Here each block has its own address and it is possible to read or write each block independently of all the other ones. Common block size of block devices are range from 512 bytes to 32,768 bytes. Disk is a example of block device.

Devices which deliver or accept a stream of characters are called character devices. These devices do not have any block structure and seek operation. Character devices are not addressable. Examples of character devices are printers, network interfaces etc.

There are also some devices which are neither block devices nor character devices. For example, Clocks are the devices which only cause interrupts at well-defined intervals but it do not have any block structure. On the other hand clocks do not deliver or accept character streams.

8.3.2 DEVICE CONTROLLERS

A **device controller** is a part of a computer system. It is the electronic component of an I/O unit and the I/O device is the mechanical component of an I/O unit. The device controller is a printed circuit and usually has a connector on it. Any device connected to the computer is connected by a plug and socket, and the socket is connected to the connector of a device controller. The controller can be inserted into some slot on the computer's parentboard. Device controllers use binary and digital codes. Each device controller has a local buffer and a command register. It communicates with the CPU by interrupts. Many controllers can handle two, four or even eight identical devices. The operating system generally deals with the controller, not the device. The interface between the controller and the device is often a very low-level interface.

Device Controller plays an important role in order to operate the I/O device. The Device Controller receives the data from a connected device and stores it temporarily in some special purpose registers (i.e. local buffer) inside the controller. Then it communicates the data with a Device Driver. It also performs any error correction if necessary. For each device controller there is an equivalent device driver which is the standard interface through which the device controller communicates with the Operating Systems through Interrupts.

8.3.3 DIRECT MEMORY ACCESS

There are two methods to perform input/output between the CPU and peripherial devices in a computer which are Memory-mapped I/O and Port-mapped I/O. Each controller has used some control registers to communicate with the CPU. The operating system can write instructions into these registers to command the device to deliver data, accept data, switch on or off and perform some other action. On the other hand the operating system read these registers to understand about the device's state. By reading from these registers, the operating system can also learn, whether the device is prepared to accept a new command or not. Many devices have also a data buffer that the operating system can read and write.

In case of Memory-mapped I/O, the same address bus is used to address both memory and I/O devices, and the CPU instructions used to access the memory are also used for accessing devices. In order to accommodate the I/O devices, areas of the CPU's addressable space must be temporarily reserved for I/O. Each I/O device monitors the CPU's address bus and responds to any of the CPU's access of device-assigned address space, connecting the data bus to a desirable device's hardware register.

Port-mapped I/O uses a special class of CPU instructions specifically for performing I/O. I/O devices have a separate address

space from general memory, either accomplished by an extra “I/O” pin on the CPU’s physical interface, or an entire bus dedicated to I/O. Because the address space for I/O is isolated from that for main memory, this is sometimes referred to as isolated I/O.

Transferring data from I/O controller to other I/O devices wastes the CPU’s time. There is a feature of modern computers called Direct Memory Access (DMA) which allows certain hardware subsystems within the computer to access system memory for reading or writing independently of the central processing unit (CPU). DMA is also used for intra-chip data transfer in multi-core processors and for transferring data between the local memory and the main memory. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel.

The operating system can only use DMA if the hardware has a DMA controller. Sometimes the DMA controller is integrated into disk controllers and other controllers. In this case a separate DMA controller for each device is available. But in most cases, a single DMA controller is available (e.g., on the parentboard) for regulating transfers to multiple devices.

The DMA controller contains several registers which include a memory address register, a byte count register, and one or more .The control registers can be written and read by the CPU. The control registers specify the I/O port to use, the direction of the transfer i.e reading from the I/O device or writing to the I/O device, the transfer unit i.e byte at a time or word at a time, and the number of bytes to transfer in one burst.

When DMA is used the DMA controller need three information from the CPU which are the disk address of the block of data to be transferred, the memory address where the block is to go and the number of bytes to transfer. Now to initiate the DMA transfer, the CPU programs the DMA controller by setting its registers so that it knows the source, the destination and the amount of bytes of the

transfer. It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.

The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller .Here the wire used to send this read request is called DMA-request. Now the memory address to write to is on the bus' address lines so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle. When the write is complete, the disk controller sends an acknowledgement signal to the disk controller, also over the bus .Here the wire used to send this acknowledgment signal is called DMA-acknowledge .The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0 then the above processes repeated until the count reaches 0. At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete. When the DMA controller seizes the memory bus, the CPU is momentarily prevented from accessing main memory, although it can still access data items in its primary and secondary cache.

Most DMA controllers use physical memory addresses for their transfers. Using physical addresses requires the operating system to convert the virtual address of the intended memory buffer into a physical address and write this physical address into the DMA controller's address register. An alternative scheme used in a few DMA controllers is to write virtual addresses into the DMA controller instead. Then the DMA controller must use the Memory Management Unit (MMU) to have the virtual-to-physical translation done. Only in the case that the MMU is part of the memory rather than part of the CPU, can virtual addresses be put on the bus.

The disk first reads data into its internal buffer before DMA can start. Now the question is why the controller does need an

internal buffer. There are two reasons. First, by doing internal buffering, the disk controller can verify the checksum before starting a transfer. If the checksum is incorrect, an error is signaled and no transfer is done.

The second reason is that once a disk transfer has started, the bits keep arriving from the disk at a constant rate, whether the controller is ready for them or not. If the controller tried to write data directly to memory, it would have to go over the system bus for each word transferred. If the bus were busy due to some other device using it, the controller would have to wait. If the next disk word arrived before the previous one had been stored, the controller would have to store it somewhere. If the bus were very busy, the controller might end up storing quite a few words and having a lot of administration to do as well. When the block is buffered internally, the bus is not needed until the DMA begins, so the design of the controller is much simpler because the DMA transfer to memory is not time critical.

DMA is not useful for all computers. The main CPU is often far faster than the DMA controller and can do the job much faster. If the CPU is not heavily loaded with jobs then it is not beneficial to use the slow DMA controller instead of the fast CPU.

8.4 PRINCIPLES OF I/O SOFTWARE

Today's complex I/O hardware needs a good design of the I/O software so that the I/O operations could be done efficiently. The basic idea is to organize the software as a series of layers,with the lower ones concerned with hiding the peculiarities of the hardware from the upper ones, and the upper ones concerned with presenting nice, clean, regular interface to the users.

Goals of I/O software :

- **Device independence**–The I/O software should be designed in such a way so that it can access any I/O device without having

to specify the device in advance .In other words we can say an I/O software should be device independent. For example, a program should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.

- **Uniform naming**– The name of a file or a device should be a string or an integer and not depend on the device in any way. The addressing of all files and devices should be uniform.
- **Error handling**– Error handling should be done as close to the hardware as possible. If the controller discovers a read error, it should try to correct the error itself if it can. If it cannot, then the device driver should handle it. Many errors can be handled by repeating operations for example read errors can be occurred by specks of dust on the read head, and it can be removed by repeating the operation. Now if the lower layers are not able to handle the errors then the upper layers should be allowed to handle it.
- **Synchronous versus asynchronous transfers**– The I/O data transfers can be synchronous or asynchronous. In general, it is asynchronous. In asynchronous I/O transfers, the CPU starts the transfer and may switch to do something else until the interrupt arrives. In some cases synchronous transfer is needed. In synchronous or blocking transfer, after a read system call the program is automatically blocked until the I/O transfer is completed.
- **Buffering**– The data that come off a device cannot be stored directly in its final destination. These should be stored in some memory area before stored in the destination. This is called buffering. Buffering is important for I/O performance.
- **Sharable versus dedicated devices**– Sharable I/O devices can be used by multiple users at the same time. For example: disks. No problems are caused by multiple users having open files on the same disk at the same time. Other devices, such as

printers are dedicated devices which can be used by a single user at a particular time. When multiple user want to access dedicated devices then sometimes deadlock can occur. The operating system must be able to handle both shared and dedicated devices in a way that avoids problems.

The layers of I/O software are:

- Interrupt handlers
- Device drivers
- Device-independent operating system software
- User-level I/O software

8.4.1 INTERRUPT HANDLERS

Interrupt is a signal informing a program that an event has occurred. When a program receives an interrupt signal, it takes a specified action. Interrupt signals can cause a program to suspend itself temporarily to service the interrupt. They should be hidden in the operating system, so that as little of the operating system as possible knows about them. The best way to hide them is to block the driver after starting an I/O operation until the I/O has completed and the interrupt occurs. Interrupt handler is a routine or procedure that takes control when the interrupt occurs .The interrupt handler determines the cause of the interrupt performs the necessary action and executes a RETURN from interrupt instruction to return the CPU to the execution state prior to the interrupt. When the interrupt happens, the interrupt handler performs operations to handle the interrupt. Then it can unblock the driver that started it. So the driver that was previously blocked will now be able to run. This model works best if drivers are structured as kernel processes, with their own states, stacks, and program counters.

The interrupt mechanism is a complex job which need a great deal of work done by the operating system .The different steps of the interrupt mechanism are:

1. Save any registers including the Program Status Word(PSW) that have not already been saved by the interrupt hardware.
2. Set up a context for the interrupt service procedure. Doing this may involve setting up the Translation Lookaside Buffer(TLB), MMU and a page table.
3. Set up a stack for the interrupt service procedure.
4. Acknowledge the interrupt controller. If there is no centralized interrupt controller, reenable interrupts.
5. Copy the registers from where they were saved (possibly some stack) to the process table.
6. Run the interrupt service procedure. It will extract information from the interrupting device controller's registers.
7. Choose which process to run next. If the interrupt has caused some high-priority process that was blocked to become ready, it may be chosen to run now.
8. Set up the MMU context for the process to run next. Some TLB set up may also be needed.
9. Load the new process' registers, including its PSW.
10. Start running the new process.

8.4.2 DEVICE DRIVERS

A Device driver is a system program that controls a specific type of I/O devices attached to the computer. Each I/O device attached to a computer needs a device driver to perform its job. The device driver is generally written by the device's manufacturer and delivered along with the device. The modern operating systems already contain many device drivers but if later a new type of device need to be attached to the computer for which no appropriate device driver is available in the operating system , then an appropriate new device driver have to be installed. Each device driver normally handles one device type, or one class of closely related devices.

A device driver has several functions:

- The device driver accept read and write requests from the device-independent software and see that they are carried out.
- The device driver must initialize the device, if needed.
- The device driver may also need to manage the power requirements and log events of the device.
- The device driver may check if the device is currently in use. If it is currently in use then the request for the device will be queued for later processing. Now if the device is idle, then hardware status will be examined to see if the request can be handled now. If the request can be handled then a sequence of commands are issued to the device. The sequence of the commands is determined in the device driver depending on what has to be done. Then the driver starts writing them into the controller's device registers. Now if the controller can accept the command and is prepared to accept the next one, all the commands have been issued. In the next step the driver performs the operations. In some cases the device driver must wait until the controller does some work for it, so it blocks itself until the interrupt comes in to unblock it. Now the driver must check for errors. If there is no error then the driver may have data to pass to the device-independent software. Finally, the device driver performs some error reporting operations.

After responding a request by the device driver, if there are any other requests available then one of them can be selected and started. Otherwise the driver blocks and waiting for the next request.

Operating systems usually classify drivers into one of a small number of categories. The most common categories are the **block devices**, such as disks and the **character devices**, such as keyboards and printers. Most operating systems define a standard interface that all block drivers must support and a second standard

interface that all character drivers must support. These interfaces consist of a number of procedures that the rest of the operating system can call to get the driver to do work for it.

Drivers are not allowed to make system calls, but they often need to interact with the rest of the kernel. Usually, calls to certain kernel procedures are permitted.

8.4.3 DEVICE INDEPENDENT I/O SOFTWARE

One part of I/O software is device independent. The device-independent software performs the I/O functions that are common to all devices and provide a uniform interface to the user-level software. The basic functions done in the device-independent I/O software are:

- Uniform interfacing for device drivers
- Buffering
- Error reporting
- Allocating and releasing dedicated devices
- Providing a device-independent block size

Uniform Interfacing for Device Drivers : An uniform interfacing is maintained by the operating system to make all I/O devices and drivers look more-or-less the same. This uniform interfacing is necessary because if the different I/O devices are interfaced in different ways then every time a new device is added, the operating system must be modified.

An uniform interface between the device drivers and the rest of the operating system makes the installation of a new driver easier.

The device-independent software takes care of mapping symbolic device names onto the proper device driver. The i-node contains two numbers which are the major device number and the minor number. The major number is used to specify the appropriate driver and the minor device number is passed as a parameter to the driver to specify the unit to be read or written i.e. to access the

particular device.

Buffering : Buffering means to store data in a memory area called buffer while they are transferred between two devices or between a device and an application. Sometimes a single buffer is not sufficient for efficient data transfer. In such cases a second kernel buffer is used. If the first buffer is full the second buffer is used before the first one has been emptied. When the second buffer fills up, it is available to be copied to the user. While the second buffer is being copied to user space, the first one can be used for new data. In this way, the two buffers take turns: while one is being copied to user space, the other is accumulating new input. A buffering scheme like this is called double buffering.

Buffering is important for system performance. One reason to use buffering is to manage the problems occurred due to the speed mismatch between the producer and consumer of a data stream.

A second use of buffering is to manage the different data-transfer sizes between the devices. In computer networking, the buffers are used to store data stream and these messages are fragmented into small network packets. The packets are sent over the network, and the receiving side places them in a reassembly buffer to form the correct source data. So in such data transfers, different sizes of data do not create any problem because of buffering.

When an application is trying to write a buffer of data to disk, it will call a write system call. Now if after the write system call returns, the application changes the contents of the buffer then because of buffering, the version of the data written to disk is guaranteed to be the version at the time of the application system call, independent of any subsequent changes in the application's buffer. It is called copy semantics. In the given case, the operating system can guarantee copy semantics is by copying the application data into a kernel buffer before returning control to the application. The disk write is performed from the kernel buffer, so that

subsequent changes to the application buffer have no effect.

Buffering is also important on output.

Buffering is important but if data get buffered too many times then the performance of the system is degraded.

Error Reporting : The operating system must handle the I/O errors to perform the I/O operations efficiently. Many errors are device-specific and must be handled by the appropriate driver .The Error reporting is handled by the device independent I/O software.

In case of programming error, device independent I/O software returns an error code to the caller. For example when a process try to write to an input device like keyboard, mouse, scanner, etc. or reading from an output device like printer, plotter, etc. then an error code is sent to the process.

In case of actual I/O errors, the device driver is responsible for the error handling but if it can not able to solve the problem then it may pass the problem back up to device independent software .For example, trying to write a damaged disk block.

In case of some other errors which can not be handled in the given way, the system may have to display an error message and terminate. For example, a critical data structure like root directory, may have been destroyed.

Allocating and Releasing Dedicated Devices : Dedicated devices like printers can be used only by a single process at any given moment. The device independent I/O software of the operating system examine the requests for the dedicated devices and it accept these requests if the devices are available otherwise reject them . One way to handle these requests is to require processes to perform opens on the special files for devices directly. If the device is unavailable, the open fails. After using the dedicated devices, it releases it.

An alternative approach is to have special mechanisms for requesting and releasing dedicated devices. An attempt to acquire a device that is not available blocks the caller instead of failing.

Blocked processes are put on a queue. Sooner or later, the requested device becomes available and the first process on the queue is allowed to acquire it and continue execution.

Device-Independent Block Size : Different disks may have different sector sizes. It is the device-independent software's responsibility to hides this fact and to provide a uniform block size to higher layers. In this way, the higher layers only deal with abstract devices that all use the same logical block size, independent of the physical sector size. Now some character devices like modems deliver their data one byte at a time and there are some others devices like network interfaces which deliver their data in larger units. These differences may also be hidden by the device independent software.

8.4.4 USER-SPACE I/O SOFTWARE

User space I/O software is a small portion of the operating system run outside the kernel. It consists of libraries linked together with user programs. System calls, including the I/O system calls, are normally made by library procedures. The standard I/O library contains a number of procedures that involve I/O and all run as part of user programs. There are some user-level I/O software which are not consists of library procedures .For example, the spooling system. Spooling is a way of dealing with dedicated I/O devices like printers in a multiprogramming system.

CHECK YOUR PROGRESS

Q.1. Multiple choice questions :

I) Disk is a :

A. Character device B. Block device

C. Both A and B D. None of the above

II) Clock is a:

A. Character device B. Block device

C. Both A and B D. None of the above

III) DMA is not useful when :

A. CPU is not heavily loaded with jobs.

B. CPU is heavily loaded with jobs.

C. Machine dependent.

D. None of the above.

IV) Which is not a goal of I/O software :

A. Device dependence B. Error handling

C. Uniform naming D. Buffering

V) Which is a layer of I/O software :

A. Device controller B. Device driver

C. Interrupt handlers D. Both B and C

Q.2. Fill in the blanks :

I) A Device driver is a _____ that controls a specific type of I/O devices attached to the computer.

II) _____ devices can be used only by a single process at any given moment.

III) A device controller is the _____ component of an I/O unit.

IV) DMA is useful when the _____ is heavily loaded with jobs.

V) Device driver is responsible for _____.

Q.3. State whether the following statements are true or false:

I) DMA is faster than the CPU.

II) Device controller is a software component of the I/O unit.

III) Device driver is a hardware part of the I/O hardware unit.

IV) Interrupt is a signal informing a program that an event has occurred.

V) Printer is a examples of character devices.

8.5 LET US SUM UP

The summery of this unit is given as follows:

- Input output devices are divided into two categories: block devices and character devices. There are also some devices which are neither block devices nor character devices.
- A device controller is the electronic component of an I/O unit. It plays an important role in order to operate the I/O device.
- Direct memory access (DMA) is a feature of modern computer systems that allows certain hardware subsystems to access system memory for reading or writing independently of the central processing unit.
- Different objectives of designing I/O software are Device independence, Uniform naming, Error handling, Synchronous versus asynchronous transfers, Buffering and Sharable versus dedicated devices.
- Different layers of I/O software are Interrupt handlers, Device drivers, Device-independent operating system software and User-level I/O software.
- Interrupt handler is a routine or procedure that takes control when the interrupt occurs.
- A Device driver is a system program that controls a specific type of I/O devices attached to the computer system.
- One part of I/O software is device independent .The basic functions of this part are:
  1. Device independent I/O software maintains a uniform interfacing for different device drivers. The device independent software takes care of mapping symbolic device names onto the proper device driver
  2. Buffering
  3. Error reporting

4. Allocating and releasing dedicated devices
5. Device independent I/O software provides a device-independent block size for different disk or other devices to provide a uniform block size to higher layers.

- User space I/O software is a small portion of the operating system but it runs outside the kernel. It consists of libraries linked together with user programs.

8.6 ANSWER TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : I) B, II) D, III) A, IV) A, V) D

Ans. to Q. No. 2 : I) system program, II) Dedicated, III) electronic, IV) CPU, V) controlling specific type of I/O devices

Ans. to Q. No. 3 : I) False, II) False, III) False, IV) True, V) True

8.7 FURTHER READINGS

- Andrew S. Tanenbaum : *Modern Operating Systems*, Prentice-Hall
- Silberschatz, Galvin, Gagne: *Operating System Concepts*.
- Stallings: *Operating Systems: Internals And Design Principles*, Pearson Education India.

8.8 MODEL QUESTIONS

Q.1. Explain how I/O management is done by the operating system.
Q.2. Explain the working principles of Direct Memory Access.
Q.3. What are the different goals of I/O software?
Q.4. Explain the different layers of I/O software.
Q.5. What is device driver?
Q.6. Explain the basic functions of the device-independent I/O software.

UNIT: 9 SECURITY AND PROTECTION

UNIT STRUCTURE

9.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about the security and protection of operating system
- illustrate different security threats and goals
- describe the implementation of a security policy
- learn about authentication and access control
- learn about cryptography

9.2 INTRODUCTION TO SECURITY AND PROTECTION

Most individuals use computers to store private information at home and important professional information at work. Many companies possess valuable confidential information. This information can be technical, commercial, financial, legal etc. So file systems often contain information that is highly valuable to their users. It is the responsibility of operating system to provide efficient security mechanisms. It must not only secure the information to protect the privacy but also prevent misuse of system resources. In this unit we are going to learn about the security and protection mechanism of operating system. The term security is used to refer to all aspects of protecting a computer system from all types of possible problems such as physical damage, loss or corruption of data, loss of confidentiality etc.

9.3 SECURITY THREATS AND GOALS

The security threat is a type of action that may harm the system. Now the necessary actions to protect the system from any threat should be implemented. These implementations are not only technical solutions but also include user training, awareness and there should be some clearly defined rules.

There are four requirements of computer and network security given as follows:

- It requires confidentiality that means the information in a computer system can be accessible only by authorized users.
- It requires integrity that means unauthorized users should not be able to modify any data without the owner's permission. Modification of data includes writing new data, changing existing data, deleting and creating.
- It requires availability of the computer system assets to the authorized parties.
- It requires authenticity that means a computer system must be able to verify the identity of an authorized user.

In order to secure a system, the potential threats must be identified and some necessary action should be performed.

There are different types of threats available given as follows:

- **Interruption:** Loss of availability of an asset of the system is termed as physical threat or interruption. For examples destruction of a piece of hardware, such as a hard disk, the cutting of a communication line, or the disabling of the file management system.
- **Accidental Error:** There are different types of accidental errors may occur in any system that may be a security threat in some situation. For example: error made by programmers, incorrect data entry, wrong program run, errors in the calculation and presentation of some results, invalid updating of files etc. The backup and recovery procedures can be used as the solution for this problem.
- **Hardware or software errors** may also act like threats in some situations. For example: CPU malfunctions, program bugs, telecommunication errors etc.
- **Unauthorized access:**

  Interception: It means that an unauthorized party gain access to a system resource. This is an attack on confidentially. The unauthorized party could be a person, a program, or a computer.

  Modification: It means that an unauthorized party not only gains access to a system resource but also change the original settings of the resource. This is an attack on integrity. For example: change value in a data file, altering a program so that it performs differently etc.

  Fabrication: It means that an unauthorized party inserts counterfeit objects into the system. This is an attack of authenticity.

- **Malicious software:**

  Viruses: A virus is a small program which can invade a computer system by attaching itself to a legitimate program. It can also propagate by creating copies of itself which get attached to other programs. In addition to its reproductive activity, each virus will have some malicious effect, ranging from the trivial display of a message to serious corruption of the file system.

Worms: A worm is an independent process which spawns copies of itself. The effect of this is to clog up a system with spurious execution of these processes, preventing legitimate processes from running properly. In addition, like virus, the worm may perform some other destructive activity. Worms are usually associated with propagation through network systems.

Trojan horse: A Trojan horse is a program which ostensibly and even actually performs some useful legitimate function, but which also performs some other undesirable activity. A Trojan horse can be created by subtle modification of a normal program such as a compiler or text editor. When such a modified program is executed, the trojan horse code can perform any invasive, destructive or mischievous activity it chooses. This mechanism is very threatening for a number of reasons. When the contaminated program is executed, it will very likely have all the access privileges of the user, rendering this line of defense useless. Confidential files could be copied to an area normally accessible by the trojan horse or its activity could pass totally unseen by the system user and administrators. A trojan horse could also be the initiator of a virus invasion.

Logic bomb: The logic bomb is a program threat predating viruses and worms. It is a code that embedded in some legitimate program and set to execute when certain conditions are met. Examples of conditions that can be used as triggers for a logic bomb are the presence or absence of certain files, a particular day of the week or date, or a particular user running the application. Once a logic bomb is executed, it may alter or delete data or entire files, cause a machine halt, or do some other damage.

- **The Race Condition Attack:** A race condition occurs when two or more operations occur in an undefined manner. Specifically, the attacker attempts to change the state of the file system between two file system operations. Usually, the program expects these two operations to apply to the same file, or expects the information retrieved to be the same. If the file operations are not atomic, or do not reference the same file this cannot be guaranteed without proper care.

- **Fraudulent misuse:** One of the most serious problems affecting commercial computer systems is the threat of fraud being carried out by people working within the organization. One of the most sensitive areas in this respect is programming staff who by virtue of their role in the organization have the opportunity to produce programs which do more than the specification defines. Programmers would find it relatively easy to create Trojan horse programs, for instance. Another source of danger is higher level staff that is generally in position of trust and thereby have wide access to the system. There are no simple solutions to these problems, and the measures taken primarily involve procedural, supervisory and auditing controls.

Goals of Security:

In general there are five goals of security:

- **Integrity** **:** Data integrity is maintained. It is ensured that data is of high quality, correct, consistent and accessible. Verifying data integrity consists in determining if the data were changed during transmission.
- **Confidentiality** **:** It is ensured that only the authorized parties have access to the resources of the system.
- **Availability** **:** The goal of availability is to ensure that authorized parties can have the access to a service or resource.
- **Non-repudiation** **:** The non-repudiation is the guarantee that an operation cannot be denied at a later date.
- **Authentication** **:** The goal of authentication is to confirm a user's identity that means access control grants access to resources only to authorized individuals or parties.

9.4 AUTHENTICATION

When users try to log into a system then the operating system performed a process called user authentication to identify the authorized users. So, unauthorized access can be prevented by this process. But there are some people called hackers who are always trying to break the authentication mechanism. Now there are different types of authentication methods available as discussed as follows:

9.4.1 AUTHENTICATION USING PASSWORDS

The most widely used authentication method is to require the user to type a login name and a password. Now a central list of login name and corresponding password is maintained in this approach. The login name typed in is looked up in the list and the typed password is compared to the stored password. If they match, the login is allowed otherwise the login is rejected.

Now while a password is being typed in, the computer should not display the typed characters to keep them secret from other users available near the terminal. For example: In Windows 2000, an asterisk is displayed while each character is being typed, in UNIX, nothing at all is displayed while the password is being typed.

In older operating systems, the password file was kept on the disk in unencrypted form, but protected by the usual system protection mechanisms. But it is not a secured approach because too often many people have access the system. These may include system administrators, machine operators, maintenance personnel, programmers, management, and maybe even some other users.

A better approach is used in UNIX where the login program asks the user to type his name and password and the password is immediately “encrypted” by using it as a key to encrypt a fixed block of data. Effectively, a one-way function is being run, with the password as input and a function of the password as output. This process is not really encryption, but it can be used as encryption. The login program then reads the password file, which is just a series of ASCII lines, one per user, until it finds the line containing the user’s login name. If the encrypted password contained in this line matches the encrypted password just computed, the login is permitted, otherwise it is refused. The

advantage of this scheme is that no one, not even the super user, can look up any users' passwords because they are not stored in unencrypted form anywhere in the system.

For additional security, some modern versions of UNIX make the password file unreadable but provide a program to look up entries upon request.

Better password Security with strong password

A cracker always tries to guess passwords one at a time. So educating users about the need for strong passwords is very essential. Computer help can also be used for this purpose. Some computers have a program that generates some random words which does not have any meaning, such as ttdhdy, jjdghr etc. that can be used as passwords. The program that users call to install or change their password can also give a warning when a poor password is chosen. For example: some warnings are like the followings

1. Passwords should be a minimum of seven characters.
2. Passwords should contain both upper and lower case letters.
3. Passwords should contain at least one digit or special character.
4. Passwords should not be dictionary words, people's names, etc.

Some operating systems require users to change their passwords regularly, to limit the damage done if a password leaks out. But if users have to change their password too often then it may be possible that they start picking weak ones.

One-Time Password

One-time password is a password authentication approach where the user gets a book containing a list of passwords. Each login uses the next password in the list. If an intruder ever discovers a password then also it will not help him to break the system, since next time a different password must be used. It is suggested that the user try to avoid losing the password book.

Challenge-Response Authentication

In this authentication scheme, each new user provide a long list of questions and answers that are then stored on the server securely in encrypted form. At login, the

server asks one of them at random and checks the answer. To make this scheme practical, many question-answer pairs would be needed.

Another variation in this approach is the user has to pick an algorithm when signing up as a user, for example x^2. When the user logs in, the server sends the user an argument, say 10, in which case the user types 100. The algorithm can be different in the morning and afternoon, on different days of the week, and so on.

9.4.2 Authentication Using a Physical Object

The second method for authenticating users is to check for some physical object they have. Nowadays, the physical object used is often a plastic card that is inserted into a reader associated with the terminal or computer. Normally, the user must not only insert the card, but also type in a password, to prevent someone from using a lost or stolen card. For example: using a bank's ATM (Automated Teller Machine) starts out with the user logging in to the bank's computer via a remote terminal (the ATM machine) using a plastic card and a password.

Now the plastic cards come in two varieties: magnetic stripe cards and chip cards. Magnetic stripe cards hold about 140 bytes of information written on a piece of magnetic tape glued to the back of the card. This information can be read out by the terminal and sent to the central computer. Often the information contains the user's password so that the terminal can identify the user even if the link to the main computer is down. Typically the password is encrypted by a key known only to the bank.

Nowadays, smart cards are introduced for the authentication purpose which in general currently have a 4-MHz 8-bit CPU, 16 KB of ROM, 4 KB of EEPROM, 512 bytes of scratch RAM, and a 9600-bps communication channel to the reader. The cards are getting smarter in time, but are constrained in a variety of ways, including the depth of the chip and the width of the chip. Smart cards can be used to hold money, as do stored value cards, but with much better security and universality. The cards can be loaded with money at an ATM machine or at home over the telephone using a special reader supplied by the bank. When inserted into a merchant's reader, the user can authorize the card to deduct a certain amount of money from the card, causing the card to send a little encrypted message to the merchant. The merchant can later turn the message over to a bank to be credited for the amount paid.

The big advantage of smart cards over, say, credit or debit cards, is that they do not need an online connection to a bank.

One disadvantage of any fixed cryptographic protocol is that over the course of time it could be broken, rendering the smart card useless. One way to avoid this fate is to use the ROM on the card for a Java interpreter. The real cryptographic protocol is then downloaded onto the card as a Java binary program and run interpretively. In this way, as soon as one protocol is broken, a new one can be installed worldwide instantly. A disadvantage of this approach is that it makes an already slow card even slower. Another disadvantage of smart cards is that a lost or stolen one may be subject to a power analysis attack.

9.4.3 AUTHENTICATION USING BIOMETRICS

The third authentication method measures physical characteristics of the user. These are called biometrics. For example, voice, fingerprint etc. of a user.

In general a biometrics system has two parts: enrollment and identification. During enrollment, the user's characteristics are measured and the results digitized. Then significant features are extracted and stored in a record associated with the user. The record can be kept in a central database or stored on a smart card that the user carries around and inserts into a remote reader like an ATM machine.

The other part is identification. The user shows up and provides a login name. Then the system makes the measurement again. If the new values match the ones sampled at enrollment time, the login is accepted; otherwise it is rejected. The login name is needed because the measurements are not exact, so it is difficult to index them and then search the index. Also, two people may have the same characteristics, so requiring the measured characteristics to match those of a specific user is stronger than just requiring it to match those of any user.

The physical characteristic used for authentication purpose should have enough variability that the system can distinguish among many people with accuracy. For example hair color is not a good characteristic as too many people may have the same color. Also, the characteristic should not vary much over time. For example, a person's voice may be different due to a cold and a face may look different in different situation.

Finger length analysis can be used for authentication purpose. When this is used, each terminal has a device where the user inserts his hand, and the length of all his fingers is measured and checked against the database. But finger length measurements are not perfect however. The system can be attacked with hand molds made out of plaster of Paris or some other material.

Retinal pattern analysis is an popular biometric used in authentication purpose. Every person has a different pattern of retinal blood vessels, even identical twins. They can be accurately photographed by a camera, one meter from the subject, without the person even being aware of it. The amount of information in a retinal scan is much more than in a fingerprint, and it can be coded in about 256 bytes.

Signature analysis can also be used for authentication purpose. The user signs his name with a special pen connected to the terminal, and the computer compares it to a known specimen stored online or on a smart card. In this approach, the signature, the pen motions and pressure made while writing the signature are compared. A good forger may be able to copy the signature, but it is very difficult for him to maintain the exact pen motion and pressure while writing the signature same with the authenticated user.

Voice biometrics is also used in authentication process with minimal special hardware. Here a microphone is required to record the voice. Now software is used to identify the user of a recorded voice. Some systems just require the user to say a secret password, but these can be break by an cracker who can tape record passwords and play them back later. More advanced systems say something to the user and ask that it be repeated back and in this approach different texts are used for each login.

9.5 PROTECTION

Every operating system should have some protection mechanism to protect files and other resources. In some system, protection is enforced by a program called a reference monitor. Every time an access to a potentially protected resource is attempted, the system first asks the reference monitor to check its legality. The reference monitor then looks at its policy and makes a decision.

9.5.1 PROTECTION DOMAINS

A computer system contains many objects which are very essential to be protected. These objects can be hardware for example: CPU, memory, disk drives, printers etc. or they can be software for example: processes, files, databases, semaphores etc. Each object has a unique name by which it is referenced and finite set of operation that processes are allowed to carry out on it. READ and WRITE are operations appropriate to a file, UP and DOWN used on a semaphore. Now a way is required to prevent processes from accessing objects that they are not authorized to access. Furthermore, this mechanism must also make it possible to restrict processes to a subset of the legal operations when that is needed.

Protection mechanisms are useful to introduce the concept of a domain. A domain is a set of (object, right) pairs. Each pair specifies an object and some subset of the operations that can be performed on it. A right in this context means permission to perform one of the operations.

At every instant of time, each process runs in some protection domain. In other words, there is some collection of objects it can access, and for each object it has some set of rights. Processes can also switch from domain to domain during execution. The rules for domain switching are highly system dependent.

Now to keep track of which object belongs to which domain, a large matrix, with the rows being the domains and the columns being the objects is maintained. Here each box shows the rights, if any, that the domain contains for the object. For example a protection matrix is shown in figure 9.1.

Domain	File1	File2	File3	File4	Printer1
1	Read write			Read write	
2		read	read		write
3		write	Read write		Read write execute

Figure 9.1: A protection matrix.

9.6 ACCESS CONTROL

Every operating system should have an access control mechanisms for the resources and it should offer suitable access mechanisms that support the security policies. So here it is important to learn to design a security policy. Security policy models have evolved from many real life operating scenarios. For example in some cases, the access to resources shall be determined by associating ranks with users. This requires a security-related labeling on information to permit access. The access is regulated by examining the rank of the user in relation to the security label of the information being sought.

It is useful to make a distinction between the general problems involved in making certain that files are not read or modified by unauthorized personnel or by specific operating systems. Access control refers to the overall problem and can be categorized into:

1. Access control policies
2. Access control mechanisms

Access control policy is the protection of "whose data is to be protected from whom" and a protection mechanism is the manner by which the operating system enforces the access control policy.

Old Unix systems have at least one user with privileges to every part of the system. The system administrator requires these privileges in order to control the system on behalf of others. This means that the system administrator must have the highest level of security clearance. The privileges conferred to others should then be less than this. In general the less privilege the less chance that accidents will lead to damage or security breaches.

A computer system contains many objects that need to be protected, for example hardware and software. An object has a unique name by which it is referenced and a set of operations that can be carried out on it. For example, read and write are pertinent to a file. Mechanisms are required to:

- Protect an object from other processes accessing the object,

- Restrict processes to a subset of legal operations when needed.

One way to achieve this goal of protection is by the use of a domain, which is a set consisting of tuples of (object, rights) pairs. Each pair specifies an object and some subset of the operations that can be performed on that object. The objects in each domain can have one of three access rights: read, write and execute. An object that is a member of multiple domains can have different rights in each domain to which it is a member of.

9.6.1 BENEFITS OF ACCESS CONTROL

Access Control provides additional protection for server resources, which complements the native Windows operating system model and enables a strong defense-in-depth security practice while reducing the complexity and cost of managing access and reaching compliance.

Access Control achieves these goals through:

1. Fine-grained access control and segregation of duties to prevent internal access abuses.
2. Advanced policy management to enable efficient centralized management of security policies across the enterprise.
3. Policy-based compliance reporting of user entitlements and policy compliance.
4. Operating system hardening to reduce external security risks and ensure operating environment reliability.
5. Granular, high-integrity auditing for compliance fulfillment.

9.6.2 ACCESS CONTROL LISTS

On Unix systems, file permissions define the file mode as well. The file mode contains nine bits that determine access permissions to the file plus three special bits. This mechanism allows to define access permissions for three classes of users: the file owner, the owning group, and the other users. These permission bits are modified using

the chmod utility. The main advantage of this mechanism is its simplicity. With a couple of bits, many permission scenarios can be modeled. However, there often is a need to specify relatively fine-grained access permissions.

Access Control Lists support more fine grained permissions. Arbitrary users and groups can be granted or denied access in addition to the three traditional classes of users. The three classes of users can be regarded as three entries of an Access Control List. Additional entries can be added that define the permissions which the specific users or groups are granted.

9.7 CRYPTOGRAPHY

In ancient Greek the word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology has been used for thousands of years to safeguard military and diplomatic communications. The cryptology is divided into two separate parts which are cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the former's work by breaking his systems.

Cryptographic techniques allow a sender to disguise data so that an intruder can gain no information from the intercepted data. The receiver, of course must be able to recover the original data from the disguised data.

The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the cipher text, is then transmitted, often by messenger or radio. We assume that enemy or intruder, hears and accurately copies down the complete cipher text. However, unlike the intended recipient, he/she does not know what the decryption key is and so cannot decrypt the cipher text easily. Sometimes the intruder can not only listen to the communication channel but can also record messages and play them back later, inject his own messages, or modify legitimate messages before they get to the receiver. A fundamental rule of cryptography is that one must assume that the cryptanalysis knows the methods used for encryption and decryption.

Main goals of Cryptography:

The main goals of modern cryptography system are user authentication, data integrity, data origin authentication, non-repudiation of origin and data confidentiality.

The purpose of cryptography is to take a message or file, called the plain text, and encrypt it into the cipher text in such a way that only authorized people know how to convert it back to the plaintext.

Secret-Key Cryptography:

Many cryptographic systems have the property that given the encryption key it is easy to find the decryption key, and vice versa. Such systems are called secret-key cryptography or symmetric-key cryptography. For example there is a system called monoalphabetic substitution where each letter of the plain text is replaced by a different letter to construct the cipher text.

Now consider all '*A's* are replaced by 'L's, all '*B'*s are replaced by 'Q's, all '*C'*s are replaced by 'Z's and so on.

Plain text: ABBCABAC
Cipher text: LQQZLQLZ

Here the key will be the 26-letter string corresponding to the full alphabet. The decryption key tells how to get back from the cipher text to the plaintext.

But in this system if the amount of ciphertext is small then the cipher can be broken easily. The basic attack takes advantage of the statistical properties of natural languages. In English, for example, e is the most common letter followed by t, o, a, n, i, etc. The most common two letter combinations, called digrams, are th, in, er, re, etc. Using this kind of information, breaking the cipher is easy.

So monoalphabetic substitution ciphers should not be used. There are other symmetric key algorithms which are relatively secure if the keys are long enough. For serious security, probably 1024-bit keys should be used, giving a search space of $2^{1024} \approx 2 \times 10^{308}$ keys.

These systems are efficient because the amount of computation required to encrypt or decrypt a message is manageable. But here the sender and receiver must both be in possession of the shared secret key.

Public-Key Cryptography:

Public-key cryptography system has the property that distinct keys are used for encryption and decryption and that given a well-chosen encryption key, it is virtually impossible to discover the corresponding decryption key. Under these circumstances, the encryption key can be made public and only the private decryption key kept secret. So here the drawback of Secret-Key Cryptography is removed.

A public key system called RSA exploits the fact that multiplying big numbers is much easier for a computer to do than factoring big numbers, especially when all arithmetic is done using modulo arithmetic and all the numbers involved have hundreds of digits.

The way public-key cryptography works is that everyone picks a (public key, private key) pair and publishes the public key. The public key is the encryption key and the private key is the decryption key. To send a secret message to a user a correspondent encrypts the message with the receiver's public key. Since only the receiver has the private key, only the receiver can decrypt the message.

The main problem with public-key cryptography is that it is a thousand times slower than symmetric cryptography

Digital Signatures:

A digital signature is a mathematical scheme to verify the authenticity of a digital document. So with a valid digital signature, a recipient of a digital document can be confirmed about the originality of the document. Digital signatures are used for software distribution, financial transactions, and in other cases where it is important to detect altering or tampering of digital documents.Digital signatures make it possible to sign email messages and other digital documents in such a way that they cannot be repudiated by the sender later.

To implement digital signature, at first the document is run through a one-way hashing algorithm. The hashing function typically produces a fixed-length result independent of the original document size. The most popular hashing functions used are MD5 (Message Digest), which produces a 16-byte result and SHA (Secure Hash Algorithm), which produces a 20-byte result.

The next step assumes the use of public-key cryptography. The document owner then applies his private key to the hash to get a value called the signature block, is appended to the document and sent to the receiver.

When the document and hash arrive, the receiver first computes the hash of the document using MD5 or SHA, as agreed upon in advance. The receiver than applies the sender's public key to the signature block to encrypt the decrypted hash and getting the hash back. If the computed hash does not match the hash from the signature block, the document, the signature block, or both have been tampered with or altered by accident. The value of this scheme is that it applies public-key cryptography only to a relatively small piece of data.

CHECK YOUR PROGRESS

1. Multiple choice questions:

(I)Which is not a security threat?

A. Interruption
B. Viruses
C. Interception
D. None of the above

(II) Interception means

A. Unauthorized party gain access to a system resource
B. Unauthorized party alter the original settings of a resource
C. Unauthorized party insert counterfeit object into the system
D. Both A and B

(III) Which is not a goal of security?

A. Logic bomb.
B. Integrity.
C. Availability.
D. Authentication.

(IV) Monoalphabetic substitution is type of

A. Symmetric-key cryptography.
B. Public-key cryptography.
C. Digital signature.
D. Both A and B.

(V) The hashing function used in digital signature is

A. MD5
B. SHA
C. Monoalphabetic substitution
D. Both A and B

2. Fill in the blanks:

I. MD5 produces a _____ byte result.

II. In case of _____ passwords, the user gets a book containing a list of passwords.

III. _____ means that only the authorized parties have access to the resources of the system.

IV. The cryptographic systems which have the property that given the encryption key it is easy to find the decryption key is called _____ .

V. _____ is a mathematical scheme to verify the authenticity of a digital document.

3. State whether the following statements are true or false:

I. In case of public key cryptography, the systems have the property that given the encryption key it is easy to find the decryption key, and vice versa.

II. In case of challenge-response authentication, each new user provides a long list of questions and answers that are stored on the server securely.

III. The non-repudiation is the guarantee that an operation cannot be denied at a later date.

IV. The logic bomb is a program threat predating viruses and worms.

V. Loss of availability of an asset of the system is termed as interception.

9.8 LET US SUM UP

The summery of this unit is given as follows:

- The security threat is a type of action that may harm the system.
- There are four requirements of computer and network security given as follows: confidentiality, integrity, availability and authenticity.
- The different types of security threats are interruption, accidental error, hardware or software errors, unauthorized access, malicious software, the race condition attack, fraudulent misuse.
- The goals of security are data integrity, confidentiality, availability**,** non-repudiation, and authentication.
- User authentication is a process performed by the operating system to identify the authorized users.
- The different types of user authentication process are authentication using passwords, authentication using a physical object and authentication using biometrics.
- Every operating system should have some protection mechanism to protect files and other resources. The concept of domains is introduced in protection mechanisms. A domain is a set of (object, right) pairs.
- The cryptology is divided into two separate parts which are cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the former's work by breaking his systems.
- Secret-key cryptography or symmetric-key cryptography have the property that given the encryption key it is easy to find the decryption key, and vice versa.
- Public-key cryptography system has the property that distinct keys are used for encryption and decryption.
- A digital signature is a mathematical scheme to verify the authenticity of a digital document.

9.9 ANSWERS TO CHECK YOUR PROGRESS

1. (I) D, (II) A, (III) A, (IV) A, (V) D

2. (I) 16 byte
   (II) One-time
   (III) Confidentiality
   (IV) Secret-key cryptography
   (V) Digital signature

3. (I) False, (II) True, (III) True, (IV) True , (V) False

9.10 FURTHER READINGS

1. Andrew S. Tanenbaum : Modern Operating Systems ,Prentice-Hall
2. Silberschatz, Galvin, Gagne: Operating System Concepts
3. Stallings: Operating Systems: Internals And Design Principles, Pearson Education India

9.11 MODEL QUESTIONS

1. Explain different types of security threats and goals.
2. What is user authentication? Explain different types of user authentication scheme.
3. Explain the concept of cryptography.
4. What is digital signature?

UNIT 10 : MULTIPROCESSOR SYSTEMS

UNIT STRUCTURE

10.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- learn about multiprocessors
- learn about the interconnections in multiprocessor
- describe the most common types of multiprocessor operating systems
- describe different multiprocessor operating system functions and requirements
- learn how multiprocessor synchronization is done

10.2 INTRODUCTION

In the previous unit we discussed about security, protection and cryptography. In this unit we begin the study with the basic concepts and categories of multiprocessor systems.We then discuss the different architecture of multiprocessor interconnection, including Bus-oriented System, Crossbar-connected System, Hyper cubes, Multistage Switch-based System. Then depending upon the control structure and organisation

we discuss the three basic types of multiprocessor operating system. Also functional capabilities and requirements which are often required in an operating system for a multiprogrammed computer. And finally multiprocessor synchronization that facilitates parallel program execution and read/write sharing of data for concurrent processing. In the next unit there is detailed discussion on distributed operating system.

BASICS OF MULTIPROCESSOR SYSTEM : A multiprocessor system has two or more processors or in other words we can say that in a Multiprocessor system there are more than one processor that works simultaneously. The main goal of these systems is to improve the overall performance of the system.

Multiprocessing is a type of processing in which two or more processors work together to process more than one program simultaneously. It allows the system to do more work in a shorter period of time. UNIX is one of the most widely used multiprocessing systems.

Multiprocessor system is also known as parallel system or tightly-coupled system. It means that multiple processors are tied together in some manner. Generally, the processors are in close communication with each

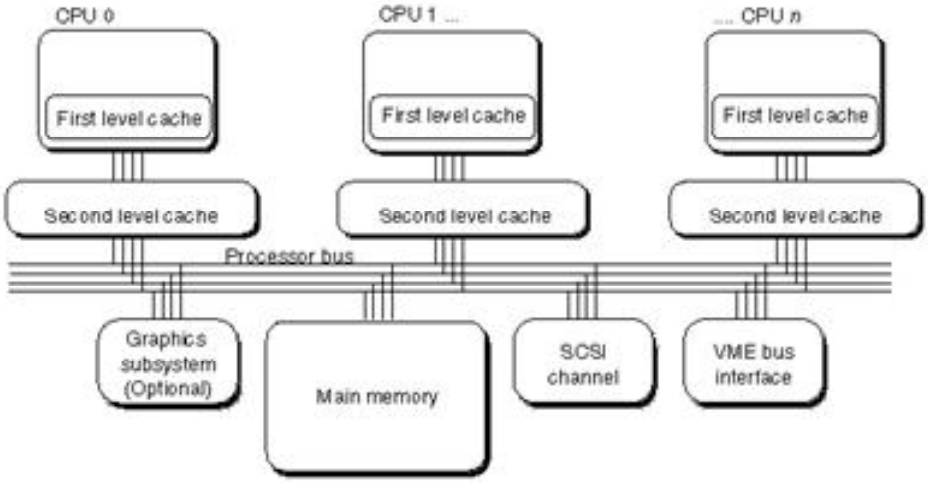


Fig 10.1 : Symmetric Multiprocessor Architecture

Advantages of Multiprocessor Systems :

- **Reduced Cost :** multiple processors share the same resources. Separate power ;supply or mother board for each chip is not

required. This reduces the cost.

- **Increased Reliability :** The reliability of system is also increased. The failure of one processor does not affect the other processors though it will slow down the machine. Several mechanisms are required to achieve increased reliability. If a processor fails, a job running on that processor also fails. The system must be able to reschedule the failed job or to alert the user that the job was not successfully completed.
- **Increased Throughput :** An increase in the number of processes completes the work in less time. It is important to note that doubling the number of processors does not halve the time to complete a job. It is due to the overhead in communication between processors and contention for shared resources etc.

CATEGORIES OF MULTIPROCESSOR SYSTEM : There are two main types of parallel processing :

1. Symmetric Multiprocessing
2. Asymmetric Multiprocessing

Symmetric Multiprocessing : A method of processing in which multiple processors work together on the same task is called symmetric multiprocessing. Each processor runs an identical copy of the same operating system. Symmetric multiprocessing treats all processors equally. I/O can be processed on any processor. The processors communicate with each other as needed. It enables many processes to be run at once without degrading performance. Each CPU is separate, it is possible for one CPU to be idle while another CPU is overloaded.

Symmetric multiprocessing is easier to implement in operating systems. It is the method mostly used by the operating systems that support multiple processors. The most common multiprocessor systems today use SMP architecture.

Asymmetric Multiprocessing : A multiprocessing technique in which individual processors are dedicated to particular tasks such as running the operating system or performing user requests is called asymmetric multi-processing. It has one master processor and remainder processors are called slave. The master distributes tasks among the slaves. I/O is usually

done by the master only.

An example is disk management where the responsibility of managing disk activities is not handled by main processor. It is performed by a microprocessor that has its own memory. It only performs disk management tasks such as how to move the head of disk to read data from disk into memory etc.

Virtually all modern operating systems including Windows NT, Solaris, Digital Unix, OS/2 and linux provide support for SMP. The difference between symmetric and asymmetric multiprocessing may be the result of either hardware or software. Special hardware can differentiate the multiple processors or the software can be written to allow only one master and multiple slaves. For instance Sun's operating system SunOS version 4 provides asymmetric multiprocessing whereas Solaris 2 is symmetric on the same hardware. As microprocessors become less expensive and more powerful, additional operating system functions are off loaded to slave processors.

Processor coupling : Another way in which multiprocessor system can be categorized is by processor coupling

Tightly-coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory or may participate in a memory hierarchy with both local and shared memory . The IBM p690 Regatta is an example of a high end SMP system. Intel Xeon processors dominated the multiprocessor market for business PCs. Both ranges of processors had their own onboard cache but provided access to shared memoryprocessor coupling.

Chip multiprocessors, also known as multi-core computing, involves more than one processor placed on a single chip and can be thought of the most extreme form of tightly-coupled multiprocessing. Mainframe systems with multiple processors are often tightly-coupled.

Loosely-coupled multiprocessor systems often referred to as clusters are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system (Gigabit Ethernet). A Linux Beowulf cluster is an example of a

loosely-coupled system.

Tightly-coupled systems perform better and are physically smaller than loosely-coupled systems, but have historically required greater initial investments and may depreciate rapidly; nodes in a loosely-coupled system are usually inexpensive commodity computers and can be recycled as independent machines upon retirement from the cluster.

Power consumption is also a consideration. Tightly-coupled systems tend to be much more energy efficient than clusters. This is because considerable economy can be realized by designing components to work together from the beginning in tightly-coupled systems, whereas loosely-coupled systems use components that were not necessarily intended specifically for use in such systems.

As with general multiprocessor systems, performance gained by the use of a multicore processor depends very much on the software algorithms and implementation. Many typical applications, however do not realize speedup factors.The parallelization of software is a significant ongoing topic of research. one of the goals of conventional processor scheduling is to keep execution units busy by assigning each processor a thread to run. A simplified model of a multiprocessor system consists of a single below.

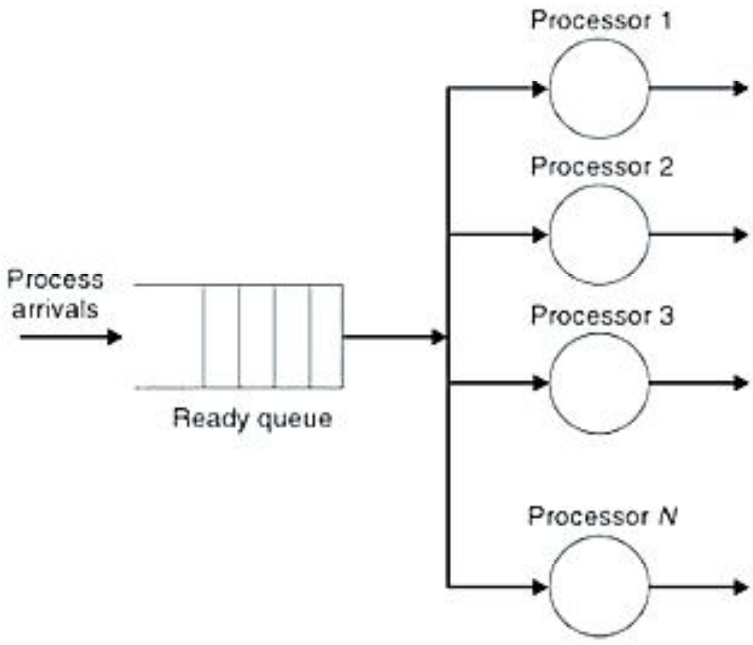


Fig. 10.2 : Multiple processors and a single queue

The model represents several servers to service arriving processes.An arriving process joins the ready queue. it examines the available server queue; if there is an available idle server,it signals the server to start service.if there are no servers the processes waits in the queue. Another model of a system with N processors, each one with its own ready queue is given below.Only difference is that an arriving process joins the short

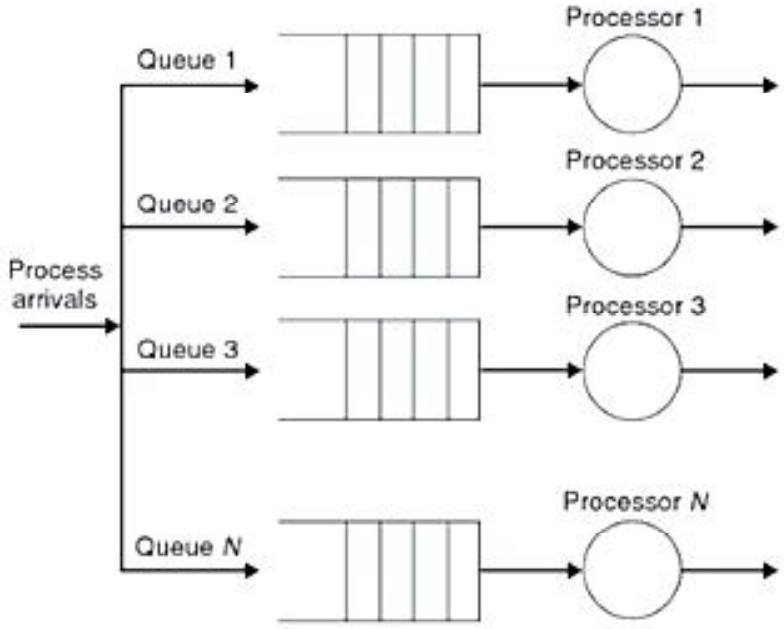


Fig. 10.3 : Multiple processors and multiple queue

10.3 MULTIPROCESSOR INTERCONNECTIONS

As learnt above, a multiprocessor can speed-up and can improve throughput of the computer system architecturally. The whole architecture of multiprocessor is based on the principle of parallel processing, which needs to synchronize, after completing a stage of computation, to exchange data. For this the multiprocessor system needs an efficient mechanism to communicate. This section outlines the different architecture of multiprocessor interconnection, including:

- Bus-oriented System

- Crossbar-connected System
- Hyper cubes
- Multistage Switch-based System.

Bus-oriented System : Figure 3 illustrates the typical architecture of a bus oriented system. As indicated, processors and memory are connected by a common bus. Communication between processors (P1, P2, P3 and P4) and with globally shared memory is possible over a shared bus. Other then illustrated many different schemes of bus-oriented system are also possible, such as:

1) Individual processors may or may not have private/cache memory.
2) Individual processors may or may not attach with input/output devices.
3) Input/output devices may be attached to shared bus.
4) Shared memory implemented in the form of multiple physical

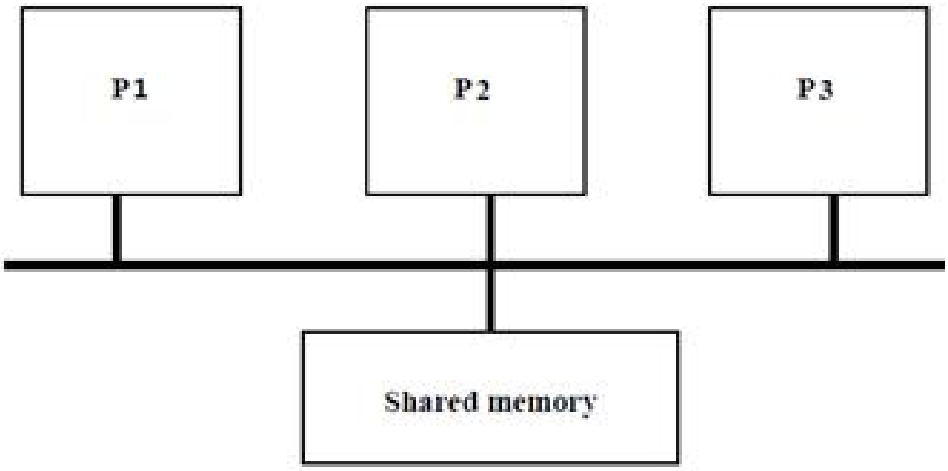


Fig. 10.4 : Bus oriented multiprocessor interconnection

The above architecture gives rise to a problem of contention at two points, one is shared bus itself and the other is shared memory. Employing private/cache memory in either of two ways, explained below, the problem of contention could be reduced;

- with shared memory; and

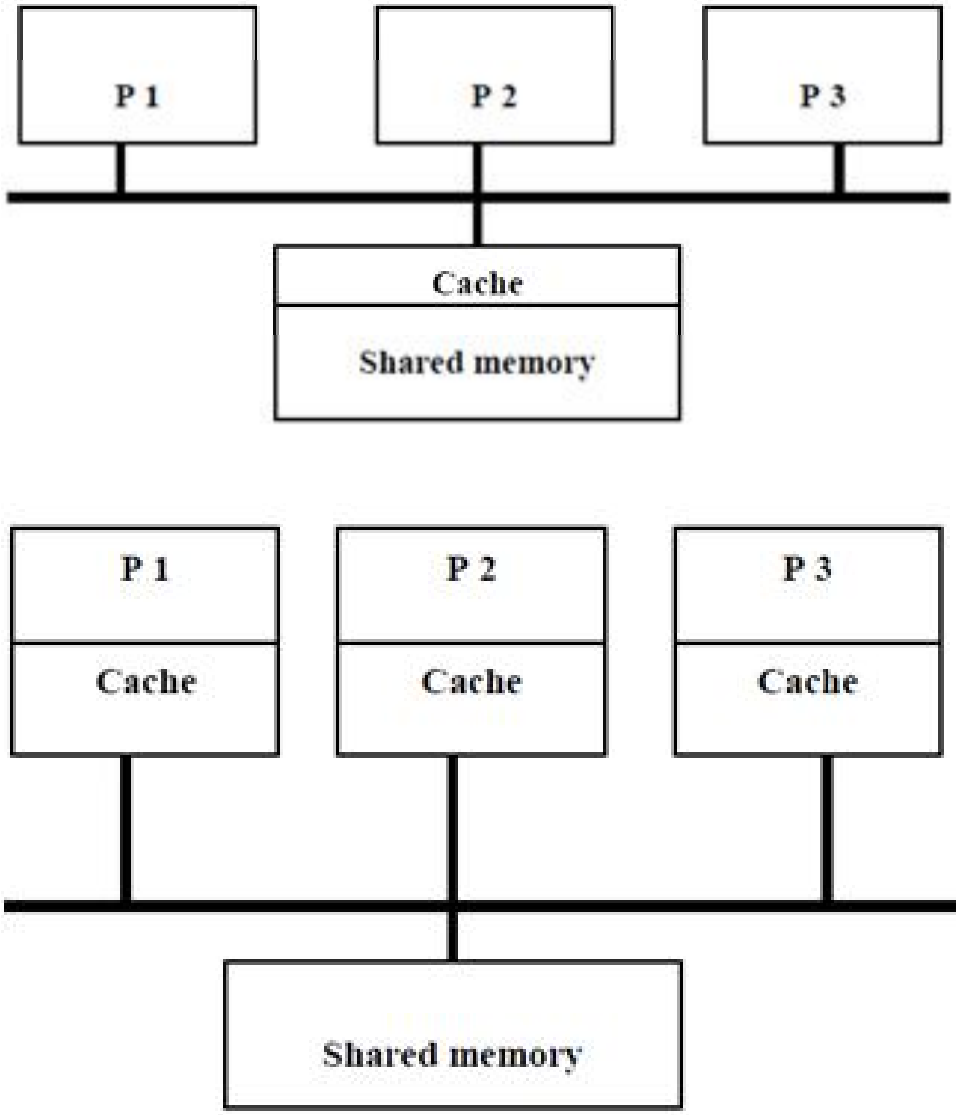


Fig. 10.5 : With cache associated with each individual processor

The second approach where the cache is associated with each individual processor is the most popular approach because it reduces contention more effectively. Cache attached with processor an capture many of the local memory references for example, with a cache of 90% hit ratio, a processor on average needs to access the shared memory for 1 out of 10 memory references because 9 references are already captured by private memory of processor. In this case where memory access is uniformly distributed a 90% cache hit ratio can support the shared bus to speed-up

10 times more than the process without cache. The negative aspect of such an arrangement arises when in the presence of multiple cache the shared writable data are cached. In this case Cache Coherence is maintained to consistency between multiple physical copies of a single logical datum with each other in the presence of update activity. Yes, the cache coherence can be maintained by attaching additional hardware or by including some specialised protocols designed for the same but unfortunately this special arrangement will increase the bus traffic and thus reduce the benefit that processor caches are designed to provide.

When clients in a system, particularly CPUs in a multiprocessing system, maintain caches of a common memory resource, problems arise. Some techniques which can be employed to decrease the impact of bus and memory saturation in bus-oriented system.

1) Wider Bus Technique: As suggested by its name a bus is made wider so that more bytes can be transferred in a single bus cycle. In other words, a wider parallel bus increases the bandwidth by transferring more bytes in a single bus cycle. The need of bus transaction arises when lost or missed blocks are to fetch into his processor cache. In this transaction many system supports, block-level reads and writes of main memory. In the similar way, a missing block can be transferred from the main memory to his processor cache only by a single main-memory (block) read action. The advantage of block level access to memory over word-oriented busses is the amortization of overhead addressing, acquisition and arbitration over a large number of items in a block. Moreover, it also enjoyed specialised burst bus cycles, which makes their transport more efficient.
2) Split Request/Reply Protocols: The memory request and reply are split into two individual works and are treated as separate bus transactions. As soon as a processor requests a block, the bus released to other user, meanwhile it takes for memory to fetch and assemble the related group of items.

The bus-oriented system is the simplest architecture of

multiprocessor system. In this way it is believed that in a tightly coupled system this organisation can support on the order of 10 processors. However, the two contention points (the bus and the shared memory) limit the scalability of the system.

Crossbar-Connected System : Crossbar-connected System is a grid structure of processor and memory modules. The every cross point of grid structure is attached with switch. By looking at the Figure it seems to be a contention free architecture of multiprocessing system, it shows simultaneous access between processor and memory modules as N number of processors are provided with N number of memory modules. Thus each processor accesses a different memory module. Crossbar needs N^2 switches for fully connected network between processors and memory. Processors may or may not have their private memories. In the later case the system supports uniform-memory-access

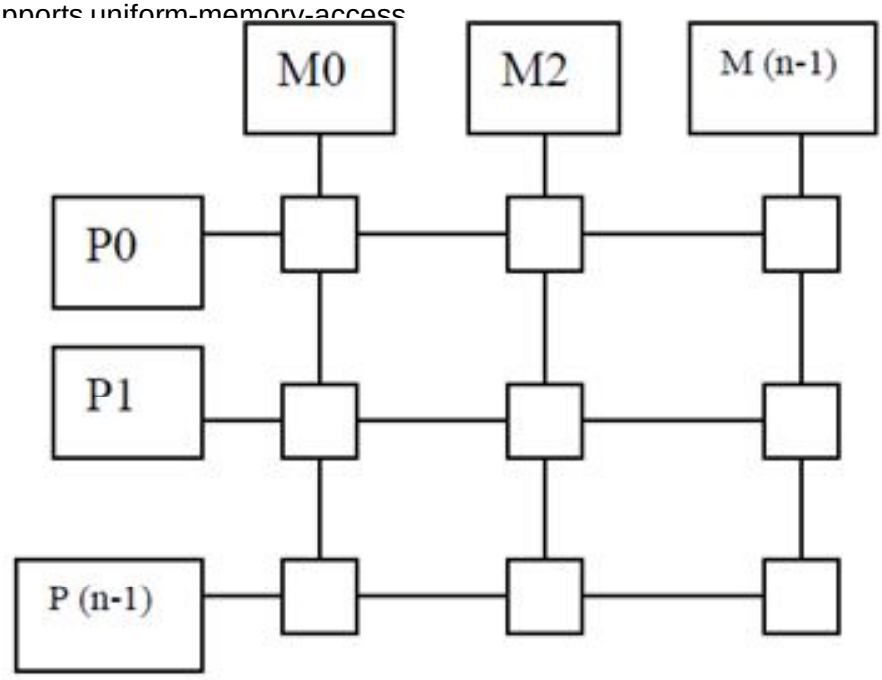


Fig. 10.6 : Crossbar connected system

Where P = processor, M = Memory Module and Switch

But Unfortunately this system also faces the problem of contention when,

More than two processors (P1 and P2) attempt to access the one

memory module (M1) at the same time. In this condition, contention can be avoided by making any one processor (P1) deferred until the other one (P2) finishes this work or left the same memory module (M1) free for processor P1.

More than two processor attempts to access the same memory module. This problem cannot be solved by above-mentioned solution.Thus in crossbar connection system the problem of contention occurs on at memory neither at processor nor at interconnection networks level. The solution of this problem includes quadratic growth of its switches as well as its complexity level. Not only this, it will make the whole system expensive and also limit the scalability of the system.

Hypercubes System : Hypercube base architecture is not an old concept of multiprocessor system. This architecture has some advantages over other architectures of multiprocessing system. As the Figure 7 indicates, the system topology can support large number of processors by providing increasing interconnections with increasing complexity. In an n-degree hypercube architecture, we have:

1) 2^n nodes (Total number of processors)
2) Nodes are arranged in n-dimensional cube, i.e. each node is connected to n number of nodes.
3) Each node is assigned with a unique address which lies between 0 to2^n –1
4) The adjacent nodes (n-1) are differing in 1 bit and the nth node is having maximum ‘n’ internode distance.

Let us take an example of 3-degree hypercube to understand the above structure:

1) 3-degree hypercube will have 2^n nodes i.e., $2^3 = 8$ nodes
2) Nodes are arranged in 3-dimensional cube, that is, each node is connected to 3 number of nodes.
3) Each node is assigned with a unique address, which lies be tween 0 to 7 (2^n –1), i.e., 000, 001, 010, 011, 100, 101, 110, 111
4) Two adjacent nodes differing in 1 bit (001, 010) and the 3rd (nth

) node is stance (100).

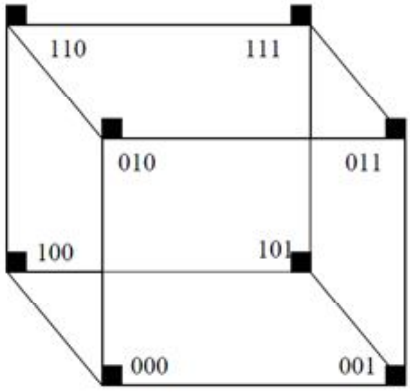


Fig. 10.7 : Hypercube System

Hypercube provide a good basis for scalable system because its communication length grows logarithmically with the number of nodes. It provides a bi-directional communication between two processors. It is usually used in loosely coupled multiprocessor system because the transfer of data between two processors goes through several intermediate processors. The longest internodes delay is n-degree. To increase the input/output bandwidth the input/output devices can be attached with every node (processor).

Multistage Switch Based system: Multistage Switch Based System permits simultaneous connection between several input-output pairs. It consists of several stages of switches which provide multistage inter-connection network. Many switches are used in a multi stage network. Each box is an interchange device with two inputs and two outputs as depicted in figure 7.

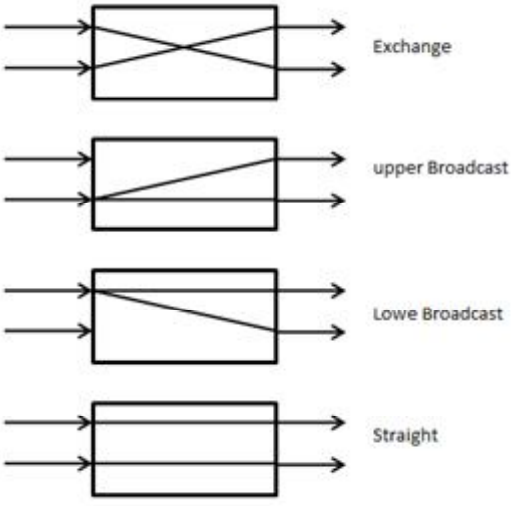
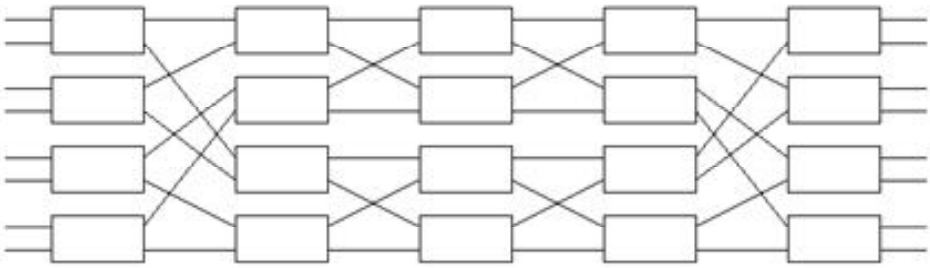


Fig. 10.8 : A two by two switching box and its four interconnection states

A N input-output connections contains K= $\log_2 N$ stages of N/2 switches at each stage. In simple words, N*N processor-memory interconnection network requires $\log_2 N$ switches.

For example, an 8×8 Benes network (i.e. with N = 8) is shown below; it has $2\log_2 8 - 1 = 5$ stages, each containing N/2 = 4 2×2 crossbar switches, and it uses a total of $N\log_2 N - N/2 = 20$ 2×2 crossbar switches. The central three stages consist of two smaller 4×4 Benes networks, while in the center stage, each 2×2 crossbar switch may itself be regarded as a 2×2 Benes



This network can connect any processor to any memory module by making appropriate connection of each of the 'K' stages. The binary address of processor and memory gives binary string routing path between module pairs of input and output. the routing path id decided by on the basis of destination binary address, that the sender includes with each request for connection. Various combinations of paths between sources to destination are possible by using different switch function (straight, swap, copy, etc.)

In multistage switch based system all inputs are connected to all outputs in such a way that no two-processor attempt to access the same memory at the same time. But the problem of contention, at a switch, arises when some memory modules are contested by some fixed processor. In this situation only one request is allowed to access and rest of the requests are dropped. The processor whose requests were dropped can retry the request or if buffers are attached with each switch the rejected request is forwarded by buffer automatically for transmission. This Multistage

interconnection networks also called store-and-forward networks.

CHECK YOUR PROGRESS

Q.1. Symmetric multiprocessing is :

a) A method of processing in which multiple processors work together on the same task.

b) A method of processing in which multiple processors work together on the different task.

c) A method of processing in which one processor work different task.

d) None of the above

Q.2. Multiple standalone single processor commodity computers interconnected via a high speed communication system is

a) Tightly coupled system b) Loosely coupled system

c) Shared system d) Attached procesor system

Q.3. Total nuber of processors in an 4-degree hypercube architecture

a) 14 b)15 c)16 d)17

Q.4. Split Request/Reply Protocols

a) data is split and placed in multiple busses

b) a processor requests a block, the bus released to other user

c) Bus issue request is split into more than 1 request

d) All of the above

Q.5. Benes network is an example of

a) Bus oriented system

b) Crossbar connected system

c) Hyper cubes

d) Multistage Switch-based System.

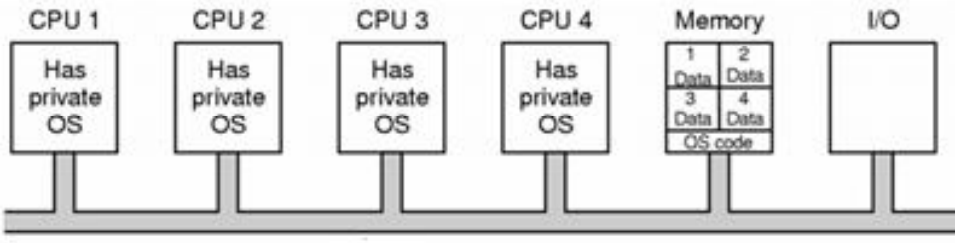
10.4 TYPES OF MULTIPROCESSOR OPERATING SYSTEMS

The multiprocessor operating systems are complex in comparison to multiprograms on an uniprocessor operating system because the operating system in multiprocessor being able to support multiple asynchronous tasks which execute concurrently.

Therefore, it must be able to support the concurrent execution of multiple tasks to increase processors performance.

Depending upon the control structure and its organisation the three basic types of multiprocessor operating system are:

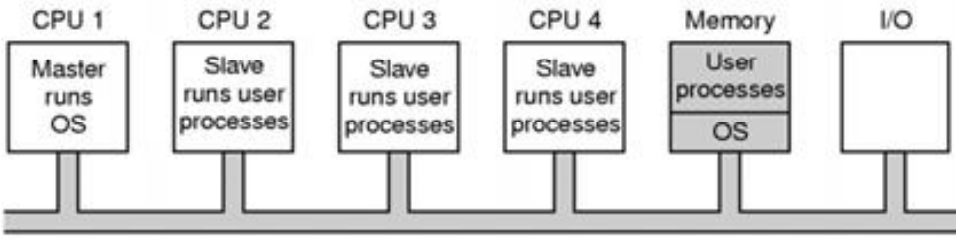
1) Separate supervisor
2) Master-slave
3) Symmetric Supervision



In separate supervisor system each process behaves independently. Each system has its own operating system which manages local input/ output devices, file system and memory well as keeps its own copy of kernel, supervisor and data structures, whereas some common data structures also exist for communication between processors. The access protection is maintained, between processor, by using some synchronization mechanism like semaphores. Such architecture will face the following problems:

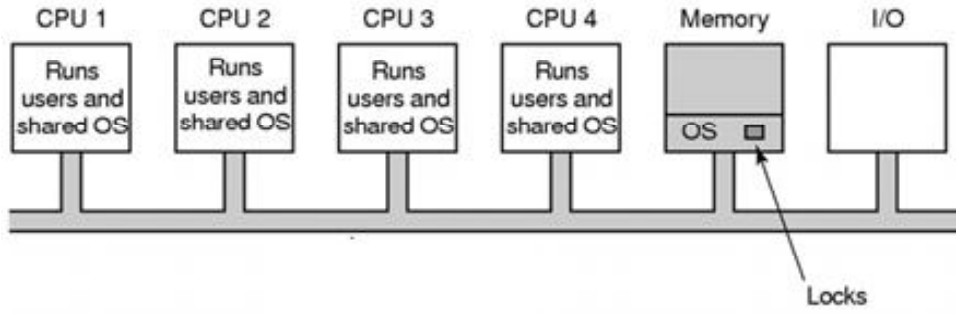
1) Little coupling among processors.
2) Parallel execution of single task.
3) During process failure it degrades.
4) Inefficient configuration as the problem of replication arises between supervisor/kernel/data structure code and each

processor.



In master-slave, out of many processors one processor behaves as a master whereas others behave as slaves. The master processor is dedicated to executing the operating system. It works as scheduler and controller over slave processors. It schedules the work and also controls the activity of the slaves. Therefore, usually data structures are stored in its private memory. Slave processors are often identified and work only as a schedulable pool of resources, in other words, the slave processors execute application programmes.

This arrangement allows the parallel execution of a single task by allocating several subtasks to multiple processors concurrently. Since the operating system is executed by only master processors this system is relatively simple to develop and efficient to use. Limited scalability is the main limitation of this system, because the master processor become a bottleneck and will consequently fail to fully utilise slave processors.



In symmetric organisation all processors configuration are identical. All processors are autonomous and are treated equally. To make all the processors functionally identical, all the resources are pooled and are available to them. This operating system is also symmetric as any processor may execute it. In other words there is one copy of kernel that can be

executed by all processors concurrently. To that end, the whole process is needed to be controlled for proper interlocks for accessing scarce data structure and pooled resources.

The simplest way to achieve this is to treat the entire operating system as a critical section and allow only one processor to execute the operating system at one time. This method is called 'floating master' method because in spite of the presence of many processors only one operating system exists. The processor that executes the operating system has a special role and acts as a master. As the operating system is not bound to any specific processor, therefore, it floats from one processor to another.

Parallel execution of different applications is achieved by maintaining a queue of ready processors in shared memory. Processor allocation is then reduced to assigning the first ready process to first available processor until either all processors are busy or the queue is emptied. Therefore, each idled processor fetches the next work item from the queue.

10.5 MULTIPROCESSOR OS FUNCTIONS AND REQUIREMENTS

The functional capabilities which are often required in an operating system for a multiprogrammed computer include the resource allocation and management schemes, memory and data set protection, prevention from system deadlocks and abnormal program termination or exception handling. In addition to these capabilities multiprocessor system also need techniques for efficient utilization of resources and hence must provide input output and processor load balancing schemes. The operating system must also be capable of providing system reconfiguration schemes to support graceful degradation.These extra capabilities and the nature of the multiprocessor environment places a much heavier burden on the operating system to support automatically the exploitation of parallelism in the hardware and the programs being executed.

A multiprocessor operating system manages all the available resources and schedule functionality to form an abstraction it will facilitates

programme execution and interaction with users.

A processor is one of the important and basic types of resources that need to be manage. For effective use of multiprocessors the processor scheduling is necessary.

Processors scheduling undertakes the following tasks:

1) Allocation of processors among applications in such a manner that will be consistent with system design objectives. It affects the system throughput. Throughput can be improved by co-scheduling several applications together, thus availing fewer processors to each.
2) Ensure efficient use of processors allocation to an application. This primarily affects the speedup of the system.

The above two tasks are somehow conflicting each other because maximum speedup needs dedication of a large proportion of a systems processors to a single application which will decrease throughput of the system. Due to the difficulties of automating the process, the need for explicit programmer direction arises to some degree. Generally the language translators and preprocessors provide support for explicit and automatic parallelism. The two primary facets of OS support for multiprocessing are:

i) Flexible and efficient interprocess and interprocessor synchronization mechanism, and
ii) Efficient creation and management of a large number of threads of activity, such as processes or threads.

The latter aspect is important because parallelism is often accomplished by splitting an application into separate, individually executable subtasks that may be allocated to different processors.

The Memory management is the second basic type of resource that needs to be managed. In multiprocessors system memory management is highly dependent on the architecture and inter-connection scheme.

- In loosely coupled systems memory is usually handled independently on a pre-processor basis whereas in multiprocessor system shared memory may be simulated by

means of a message passing mechanism.

- In shared memory systems the operating system should provide a flexible memory model that facilitates safe and efficient access to share data structures and synchronization variables.

A multiprocessor operating system should provide a hardware independent, unified model of shared memory to facilitate porting of applications between different multiprocessor environments. The designers of the mach operating system exploited the duality of memory management and inter-process communication.

The third basic resource is Device Management but it has received little attention in multiprocessor systems to date, because earlier the main focus point is speedup of compute intensive application, which generally do not generate much input/output after the initial loading. Now, multiprocessors are applied for more balanced general-purpose applications, therefore, the input/output requirement increases in proportion with the realised throughput and speed.

10.6 MULTIPROCESSOR SYNCHRONIZATION

Multiprocessor system facilitates parallel program execution and read/write sharing of data and thus may cause the processors to concurrently access location in the shared memory. Therefore, a correct and reliable mechanism is needed to serialize this access. This is called synchronization mechanism. The mechanism should make access to a shared data structure appear atomic with respect to each other. In this section, we introduce some new mechanism and techniques suitable for synchronization in multiprocessors.

Test-and-Set : The test-and-set instruction automatically reads and modifies the contents of a memory location in one memory cycle. It is as follows:

```
function test-and-set (var m: Boolean); Boolean;
begin
    test-and set: = m;
    m:= true
```

end;

The test-and-set instruction returns the current value of variable m (memory location) and sets it to true. This instruction can be used to implement P and V operations (Primitives) on a binary semaphore, S, in the following way (S is implemented as a memory location):

P(S): while Test-and-Set (S) do nothing;

V(S): S:=false;

Initially, S is set to false. When a P(S) operation is executed for the first time, test-and-set(S) returns a false value (and sets S to true) and the "while" loop of the P(S) operation terminates. All subsequent executions of P(S) keep looping because S is true until a V(S) operation is executed.

Compare-and-Swap : The compare and swap instruction is used in the optimistic synchronization of concurrent updates to a memory location. This instruction is defined as follows (r1and r2 are two registers of a processor and m is a memory location):

function test-and-set (var m: Boolean); Boolean;

var temp: integer;

begin

temp:=m;

if temp = r1 then {m:= r2;z:=1}

else {r1:= temp; z:=0}

end;

If the contents of r1 and m are identical, this instruction assigns the contents of r2 to m and sets z to 1. Otherwise, it assigns the contents of m to r1 and set z to 0. Variable z is a flag that indicates the success of the execution. This instruction can be used to synchronize concurrent access to a shared variable.

Fetch-and-Add : The fetch and add instruction is a multiple operation memory access instruction that automatically adds a constant to a memory location and returns the previous contents of the memory location. This instruction is defined as follows:

function Fetch-and-add (m: integer; c: integer);

Var temp: integer;

```
begin
    temp:= m;
    m:= m + c;
    return (temp)
end;
```

This instruction is executed by the hardware placed in the interconnection network not by the hardware present in the memory modules. When several processors concurrently execute a fetch-and-add instruction on the same memory location, these instructions are combined in the network and are executed by the network in the following way:

- A single increment, which is the sum of the increments of all these instructions, is added to the memory location.
- A single value is returned by the network to each of the processors, which is an arbitrary serialisation of the execution of the individual instructions.
- If a number of processors simultaneously perform fetch-and-add instructions on the same memory location, the net result is as if these instructions were executed serially in some unpredictable order.

The fetch-and-add instruction is powerful and it allows the implementation of P and V operations on a general semaphore, S, in the following manner:

P(S): while (Fetch-add-Add(S, -1)< 0) do

```
begin
    Fetch-and-Add (S, 1);
    while (S<0) do nothing;
end;
```

The outer “while-do” statement ensures that only one processor succeeds in decrementing S to 0 when multiple processors try to decrement variable S. All the unsuccessful processors add 1 back to S and try again to decrement it. The “while-do” statement forces an unsuccessful processor to wait (before retrying) until S is greater then 0.

V(S): Fetch-and-Add (S, 1).

CHECK YOUR PROGRESS

Q.6. The system that has its own operating system which manages local input/output devices, file system and memory.

a) Separate supervisor b) Master-slave

c) Symmetric Supervision d) None of the above

Q.7. The concept of 'floating master' method means

a) Each processors having its own OS.

b) All processors executes the OS .

c) Only one OS for many processors .

d) One out of many processors executes the OS.

Q.8. Which of these is not a multiprocessor operating system function.

a) Memory and data set protection

b) prevention from system deadlocks

c) database management

d) exception handling.

Q.9. The instruction that is executed by the hardware placed in the interconnection network for synchronization is

a) Swap and Add b) Compare and Swap

c) Test and Set d) Fetch and Add

Q.10. For clock synchronization,

a) there must be an external authoritative time source for internal synchronization;

b) internally clocks should also be externally synchronized;

c) if a set of processes P is synchronized externally within a bound D, it is also internally synchronized with the same bound D;

d) need to synchronize that computer's clock with some external authoritative source of time (external clock synchronization)

10.7 LET US SUM UP

- A multiprocessor system has two or more processors or in other words we can say that in a Multiprocessor system there are more than one processor that works simultaneously.
- A method of processing in which multiple processors work together on the same task is called symmetric multiprocessing.
- A multiprocessing technique in which individual processors are dedicated to particular tasks such as running the operating system or performing user requests is called asymmetric multiprocessing.
- Tightly-coupled multiprocessor systems contain multiple CPUs that are connected at the bus level. These CPUs may have access to a central shared memory or may participate in a memory hierarchy with both local and shared memory .
- Loosely-coupled multiprocessor systems often referred to as clusters are based on multiple standalone single or dual processor commodity computers interconnected via a high speed communication system.
- In Bus oriented system, processors and memory are connected by a common bus. Communication between processors and with globally shared memory is possible over a shared bus.
- Crossbar-connected System is a grid structure of processor and memory modules. The every cross point of grid structure is attached with switch.
- Hyper cube provides a bi-directional communication between two processors. It is usually used in loosely coupled multiprocessor system.
- Multistage Switch Based System permits simultaneous connection between several input-output pairs. It consists of several stages of switches which provide multistage interconnection network.

  Types of Multiprocessor Operating System
- Separate supervisor system each process behaves independently. Each system has its own operating system which manages local input/output devices, file system and memory.

- In master-slave, out of many processors one processor behaves as a master whereas others behave as slaves. The master processor is dedicated to executing the operating system.
- Symmetric organisation all processors configuration are identical. All processors are autonomous and are treated equally. To make all the processors functionally identical, all the resources are pooled and are available to them.
- A multiprocessor operating system manages all the available resources and schedule functionality to form an abstraction it will facilitates programme execution and interaction with users.

10.8 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : a)

Ans. to Q. No. 2 : b)

Ans. to Q. No. 3 : c)

Ans. to Q. No. 4 : b)

Ans. to Q. No. 5 : d)

Ans. to Q. No. 6 : a)

Ans. to Q. No. 7 : d)

Ans. to Q. No. 8 : c)

Ans. to Q. No. 9 : d)

Ans. to Q. No. 10 : d)

10.9 FURTHER READINGS

- Singhal Mukesh and Shivaratri G. Niranjan, *Advanced Concepts in Operating Systems*, TMGH, 2003, New Delhi.
- Hwang, K. and F Briggs, *Multiprocessors Systems Architectures*, McGraw-Hill, New York.
- Milenkovic, M., *Operating Systems: Concepts and Design*, TMGH, New Delhi.

- Silberschatz, A., J. Peterson, and D. Gavin, *Operating Systems Concepts*, 3rd ed., Addison Wesley, New Delhi.

10.10 MODEL QUESTIONS

Q.1. Discuss Multiprocessor Synchronization.

Q.2. Differentiate between Symmetric and Asymmetric multiprocessors.

Q.3. What do you understand by a interconnection network ? Explain using an example.

Q.4. What is the difference between symmetric and asymmetric multiprocessing?

Q.5. Explain Master Slave Model .

Q.6. Explain processor scheduling in case of a multiprocessor system.

Q.7. What is the difference between a loosely coupled system and a tightly coupled system? Give examples.

UNIT 11 : DISTRIBUTED OPERATING SYSTEMS

UNIT STRUCTURE

11.1 LEARNING OBJECTIVES

After going through this unit, you will be able to :

- learn about distributed operating system
- describe distributed algorithm and its uses
- learn about how distributed processing is done
- describe the models of distributed operating systems
- learn how to cope with the failure models of distributed systems
- describe remote procedure calls
- illustrate distributed shared memory
- describe how distributed file system works

11.2 INTRODUCTION

In the previous unit we have discussed how multiprocessor system helps in parallel processing .The types of multiprocessor and multiprocessor operarting system. With the advent of computer networks, in which many computers are linked together and are able to communicate with one another,

distributed computing became feasible. In this unit we start of with the basics of distributed operating system. Then we discuss the distributed algorithms and the standard problems being solved by distributed algorithms. Proposed models for distributed systems like Architectural Models,Interaction Models, Fault Models. The interprocess communication in distributed system including remote procedure calls. Also the shared memory concepts by which program processes can exchange data more quickly than by reading and writing using the regular operating system services.And finally the file system for file and directory services of individual servers ..

Basics of Distributed Operating System : A system that distributes the computation among several physical processors is called distributed system. It is known as loosely-coupled. It means that the processors do not share memory; data structures or system clock. Each processor has its own local memory. It can only communicate with other processors through a communication line usually over a network. The processors in distributed system may vary in size, speed and function.

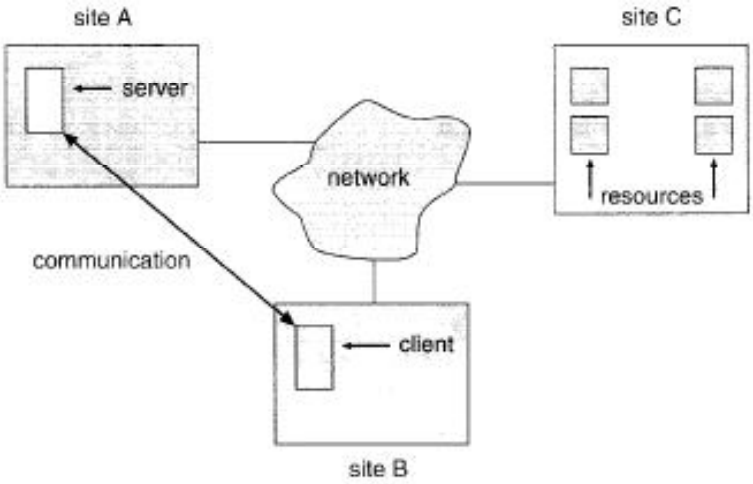


Fig. 11.1 : A Distributed System

The main advantages of building distributed systems are as follows:

1. Resource Sharing: Each computer in the distributed system may have specific resources. Other machines may utilize its resources while being part of the distributed system.
2. Computation Speedup: If a computation can be split up into sections, it is possible to give different sections to different

machines. Communication is usually fairly expensive since the machines may not be physically close to each other.

3. Reliability:It provides more reliability for different jobs. If one system fails, the other sites may be able to continue processing.

Distributed Operating System : A distributed operating system (OS) is an operating system is a generalization of a traditional operating system. A distributed OS provides the essential services and functionality required of an OS, adding attributes and particular configurations to allow it to support additional requirements such as increased scale and availability. To a user, a distributed OS works in a manner similar to a single-node, monolithic operating system. That is, although it consists of multiple nodes, it appears to users and applications as a single-node.

Distributed operating systems must handle all the usual problems of operating systems, such as deadlock,synchronization etc. Important systems concerns unique to the distributed case are workload sharing, which attempts to take advantage of access to multiple computers to complete jobs faster; task migration, which supports workload sharing by efficiently moving jobs among machines; and automatic task replication at different sites for greater reliability.

Example of distributed OS : Amoeba is an open source microkernel-based distributed operating system developed by Andrew S. Tanenbaum and others at the Vrije University. It was was originally developed in Python.

Distributed Vs Network Operating Systems : A distributed operating system is one where all of computers that are connected can share in tasks that need to be done. So one could have one or more programs that are actually running on someone else's computer. This way your computer is not bogged down by trying to do everything itself.

A network operating system is used to describe a set up where computers are connected and can share some resources either with each other, or a central server. They generally don't openly share all of the workload, although the server can and does provide many services for the other computers on the network.

11.3 ALGORITHMS AND DISTRIBUTED PROCESSING

Distributed processing refers to any of a variety of computer systems that use more than one computer, or processor, to run an application. This includes parallel processing, in which a single computer uses more than one CPU to execute programs. More often, however, distributed processing refers to local-area networks designed so that a single program can run simultaneously at various sites. Most distributed processing systems contain sophisticated software that detects idle CPUs on the network and parcels out programs to utilize them.

Another form of distributed processing involves distributed databases, databases in which the data is stored across two or more computer systems. The database system keeps track of where the data is so that the distributed nature of the database is not apparent to users.

In its most general sense, an algorithm is any set of detailed instructions which results in a predictable end-state from a known beginning. A distributed algorithm is an algorithm designed to run on computer hardware constructed from interconnected processors.

Distributed algorithms are necessary in distributed systems in infrastructure, basic services, and application , services to implement distributed process and transaction rules to ensure correct behaviour across distributed systems,Distributed algorithms are used in various application areas of distributed computing, such as telecommunications, scientific computing, distributed information processing, and real-time process control.

Distributed algorithms are typically executed concurrently, with separate parts of the algorithm being run simultaneously on independent processors, and having limited information about what the other parts of the algorithm are doing. One of the major challenges in developing and implementing distributed algorithms is successfully coordinating the behavior of the independent parts of the algorithm in the face of processor failures and unreliable communications links. The choice of an appropriate distributed algorithm to solve a given problem depends on both the characteristics of the problem, and characteristics of the system the algorithm will run on

such as the type and probability of processor or link failures, the kind of inter-process communication that can be performed, and the level of timing synchronization between separate processes.

Some of the Constraints on Distributed Algorithms:

1. requirement to have good scaling properties
2. no global consistent knowledge
3. no global clock time
4. problems of establishing consistency
5. need to cope with failures/reconfigurations
6. need to avoid distributed deadlocks

Standard problems solved by distributed algorithms include:

1) Atomic commit : An atomic commit is an operation where a set of distinct changes is applied as a single operation. If the atomic commit succeeds, it means that all the changes have been applied. If there is a failure before the atomic commit can be completed, the “commit” is aborted and no changes will be applied.

Algorithms for solving the atomic commit protocol include the two-phase commit protocol and the three-phase commit protocol.

2) Consensus : Consensus algorithms try to solve the problem of a number of processes agreeing on a common decision.More precisely, a Consensus protocol must satisfy the four formal properties below.

Termination : every correct process decides some value.

Validity : if all processes propose the same value v, then every correct process decides v.

Integrity : every correct process decides at most one value, and if it decides some value v, then v must have been proposed by some process.

Agreement : if a correct process decides v, then every correct process decides v.

3) **Reliable Broadcast :** Reliable broadcast is a communication primitive in distributed systems. A reliable broadcast is defined by the following properties:

   Validity : if a correct process sends a message, then some correct process will eventually deliver that message

   Agreement : if a correct process delivers a message, then all correct processes eventually deliver that message

   Integrity : every correct process delivers the same message at most once and only if that message has been sent by a process

4) **Replication :** It is one of the oldest and most important topics in the overall area of distributed systems. Replication is the process of sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility. It could be data replication if the same data is stored on multiple storage devices, or computation replication if the same computing task is executed many times. A computational task is typically replicated in space, i.e. executed on separate devices, or it could be replicated in time, if it is executed repeatedly on a single device.

   Whether one replicates data or computation, the objective is to have some group of processes that handle incoming events. If we replicate data, these processes are passive and operate only to maintain the stored data, reply to read requests, and apply updates. When we replicate computation, the usual goal is to provide fault-tolerance. For example, a replicated service might be used to control a telephone switch, with the objective of ensuring that even if the primary controller fails, the backup can take over its functions. But the underlying needs are the same in both cases: by ensuring that the replicas see the same events in equivalent orders, they stay in consistent states and hence any replica can respond to queries.

Replication models in distributed systems : A number of widely cited models exist for data replication, each having its own properties and performance:

1. Transactional replication : This is the model for replicating transactional data, for example a database or some other form of transactional storage structure. The one-copy serializability model is employed in this case, which defines legal outcomes of a transaction on replicated data in accordance with the overall ACID properties that transactional systems seek to guarantee.
2. State machine replication : This model assumes that replicated process is a deterministic finite automaton and that atomic broadcast of every event is possible. It is based on a distributed computing problem called distributed consensus and has a great deal in common with the transactional replication model.
3. Virtual synchrony : This computational model is used when a group of processes cooperate to replicate in-memory data or to coordinate actions. The model defines a distributed entity called a process group. A process can join a group, and is provided with a checkpoint containing the current state of the data replicated by group members. Processes can then send multicasts to the group and will see incoming multicasts in the identical order. Membership changes are handled as a special multicast that delivers a new membership view to the processes in the group.

4) Mutual exclusion : Mutual exclusion, refers to the problem of ensuring that no two processes or threads can be in their critical section at the same time. Here, a critical section refers to a period of time when the process accesses a shared resource, such as shared memory.

5) Resource allocation : Resource allocation is used to assign the available resources in an economic way. Resource allocation is the scheduling of activities and the resources required by those activities while taking into consideration both the resource

availability. Resource allocation may be decided by using computer programs applied to a specific domain to automatically and dynamically distribute resources to applicants.This is especially common in electronic devices dedicated to routing and communication. For example, channel allocation in wireless communication may be decided by a base transceiver station using an appropriate algorithm.

CHECK YOUR PROGRESS

Q.1. Which environment considers memory, process, and device and file management from a global viewpoint?

a) Distributed Operating System (DOS)
b) Network Operating System (NOS)
c) Multiprogramming Operating System (MOS)
d) None of the above

Q.2. A distributed system is

a) a parallel system with shared memory;
b) a collection of independent computers that appears to its users as a single coherent system.
c) a system that has physically distributed memory and a logically shared memory;
d) a system in which each processor must have the same architecture.

Q.3. In distributed systems, there is no global clock because

a) there is no need to have it for distributed applications;
b) it is too easy to accurately synchronize all the local clocks in DS;
c) it is a direct consequence of the fact that the only communication is message passing through a network;
d) none of the above.

Q.4. Network operating system

a) is used for managing multiprocessors and homogeneous multicomputers;

b) is used mainly to provide single system image of a distributed system;

c) is used for heterogeneous mutlicomputer systems;

d) must have middleware to share resources across the networked computer system.

Q.5. Which of the following is a distributed OS.

a) Windows NT b) Amoeba

c) Windows Vista d) UNIX

Q6. Which problems are not solved by distributed algorithms

a) Distributed search b) Mutual exclusion

c) Binary search d) Reliable Broadcast

11.4 COPING WITH FAILURES MODELS OF DISTRIBUTED SYSTEMS

We will start by discussing the following models that has been proposed so far for distributed systems.These are :

1. Architectural Models
2. Interaction Models
3. Fault Models

Architectural Model : It deals with the responsibilities distributed between system components and the way these components are placed. It has two types :

1) Client-server model : The system is structured as a set of processes, called servers, that offer services to the users, called clients. The client-server model is usually based on a simple request/reply protocol, implemented with send/receive primitives or using remote procedure calls.The client sends are request message to the server asking for some service.The server does the work and returns a result or an error code if the work could

not be performed.A server can itself request services from other servers; thus, in this new relation, the server itself acts like a client.

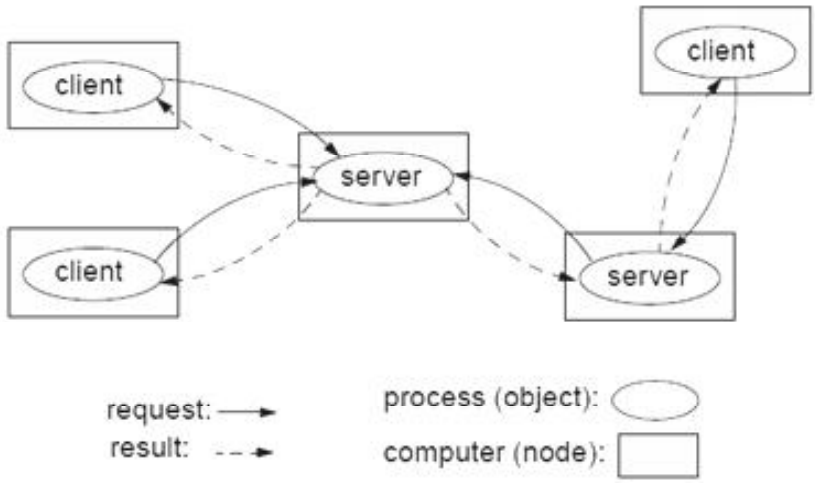


Fig. 11.2 : Client-Server Model

2) **Peer-to-Peer :** Some problems with client-server is Centralization of service, Poor scaling ,limitations- capacity of server, bandwidth of network connecting the server. Peer-to-Peer tries to solve some of the above .It distributes shared resources widely share computing and communication loads.

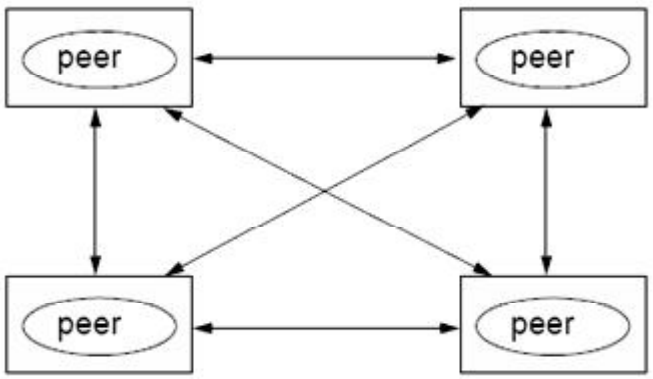


Fig. 11.3 : Peer-to-Peer

Variations of the above two are:

- Proxy server
- Mobile code
- Mobile agents
- Network computers

- Thin clients
- Mobile devices

A proxy server provides copies (replications) of resources which are managed by other servers.Proxy servers are typically used as caches for web resources. They maintain a cache of recently visited web pages or other resources.When a request is issued by a client, the proxy server is first checked, if the requested object (information item) is available there. Proxy servers can be located at each client, or can be shared by several clients.The purpose is to increase performance and availability, by avoiding frequent accesses to remote servers.

Mobile code: code that is sent from one computer to another and run at the destination.Advantage being remote invocations are replaced by local ones.Typical example are Java applets.

Mobile agent: a running program that travels from one computer to another carrying out a task on someone's behalf. A mobile agent is a complete program, code + data, that can work (relatively) independently. The mobile agent can invoke local resources/data.

Network computers : Do not store locally operating system or application code. All code is loaded from the servers and run locally on the network computer. Advantages:

- The network computer can be simpler, with limited capacity; it does not need even a local hard disk (if there exists one it is used to cache data or code).
- Users can log in from any computer.
- No user effort for software management/administration.

Thin Client : The thin client is a further step, beyond the network computer. Thin clients do not download code (operating system or application) from the server to run it locally. All code is run on the server, in parallel for several clients.The thin client only runs the user interface.Advantage is that of network computers but the computer at the user side is even simpler (cheaper).

Mobile devices : These are hardware, computing components that move (together with their software) between physical locations.This is

opposed to software agents, which are software components that migrate.Both clients and servers can be mobile (clients more frequently).

Interaction Model : This model discusses how do we handle time. Are there time limits on process execution, message delivery, and clock drifts. One of the most fundamental aspects of distributed systems is the distinction between the asynchronous and the synchronous formal models -- not to be confused with the entirely different issues of: messaging that is synchronous (send-wait) vs. asynchronous; or the occurrence of asynchronous events or state transitions.

The asynchronous model of distributed systems has no bounds on execution entity , execution latencies –- i.e., arbitrarily long (but finite) times may occur between execution steps.Message transmission latencies –- i.e., a message may be received an arbitrarily long time after it was sent and clock drift rates– i.e., process's local clocks may be arbitrarily unsynchronized

In other words, the asynchronous distributed system model makes no assumptions about the time intervals involved in any behaviors.

The synchronous model of distributed systems, conversely, has a' priori known upper and lower bounds on these quantities.Each process has a bounded time between its execution steps.Each message is transmitted over a channel and received in a bounded time.

Process's local clocks may drift either from each other or from global physical time only by a bounded rate .

It might seem that any implemented distributed system could be considered synchronous by presuming unrealistically tiny lower bounds and huge upper bounds on these three quantities. But the intent of having a system model is to facilitate reasoning about properties of the system -- if those presumed bounds are not deterministic reasoning based on them can not be deterministic, which may or may not be acceptable in any given circumstances. In particular, meeting hard deadlines requires reasoning based on deterministic worst case latencies, and doing so for multi-node behaviors requires that the system be synchronous.

The asynchronous model is the general case, and any proofs of its properties also apply to the synchronous model, but the converse is not true.

However, a number of important properties (such as distributed agreement) have been proven impossible even under very weak conditions in the asynchronous model, but are readily achievable in the synchronous model.

Neither the asynchronous nor the synchronous models are pragmatic for the vast majority of actual implemented distributed systems. Both are extreme vertices in a space of distributed system models. Distributed system models elsewhere in that space are partially synchronous along some dimension(s). Examples include (but are not limited to): a system may have either bounded process step latencies and unbounded communication latencies, or vice versa; a system may be asynchronous with synchronous subsystems; a system may change from one model to another at different times.

Fault Models : A variety of failure models have been proposed in connection with distributed systems. All are based on assigning responsibility for faulty behavior to the system's components like processors and communications channels. It is the faulty components that we count, and not occurrences of faulty behavior.Here we say that a system being t-fault tolerant when ,that system will continue satisfying its specification provided that no more than t of its components is faulty.

In classical work on fault-tolerant computing systems, it is the occurrences of faulty behavior that are counted. Statistical measures of reliability and availability, like MTBF (mean-time-between-failures) and probability of failure over a given interval, are deduced from estimates of elapsed time between fault occurrences. Such characterizations are important to users of a system, but there are real advantages to describing the fault tolerance of a system in terms of the maximum number of faulty components that can be tolerated over some interval . Thinking of a system to be t-fault tolerant is actually the measure of the fault tolerance supported by the system's architecture, in contrast to fault tolerance achieved simply by using reliable components.

Fault tolerance of a system will depend on the reliability of the components used in constructing that system ,in particular, the probability that there will be more than t failures during the operating interval of interest.

Thus, once t has been chosen, it is not difficult to derive the more traditional statistical measures of reliability.

We simply compute the probabilities of having various configurations of 0 through t faulty components. So, no expressive power is lost by counting faulty components.

Some care is required in defining failure models, however, when it is the faulty components that are being counted. For example, consider a fault that leads to a message loss. We could attribute this fault to the sender, the receiver, or the channel. Message loss due to signal corruption from electrical noise should be blamed on the channel, but message loss due to buffer overflow at the receiver should be blamed on the receiver. Moreover, since replication is the only way to tolerate faulty components, the architecture and cost of implementing a t-fault tolerant system very much depends on exactly how fault occurrences are being attributed to components. Incorrect attribution leads to an inaccurate distributed system model, erroneous conclusions about system architecture are sure to follow.

A faulty component exhibits behavior consistent with some failure model being assumed. Failure models commonly found in the distributed systems literature include:

Failstop : A processor fails by halting. Once it halts, the processor remains in that state. The fact that a processor has failed is detectable by other processors.

Crash : A processor fails by halting. Once it halts, the processor remains in that state. The fact that a processor has failed may not be detectable by other processors.

Crash+Link : A processor fails by halting. Once it halts, the processor remains in that state. A link fails by losing some messages, but does not delay, duplicate, or corrupt messages

Receive-Omission : A processor fails by receiving only a subset of the messages that have been sent to it or by halting and remaining halted.

Send-Omission : A processor fails by transmitting only a subset of the messages that it actually attempts to send or by halting and remaining halted.

General Omission : A processor fails by receiving only a subset of the messages that have been sent to it, by transmitting only a subset of the messages that it actually attempts send, and/or by halting and remaining halted.

Byzantine Failures : A processor fails by exhibiting arbitrary behavior.

Failstop failures are the least disruptive, because processors never perform erroneous actions and failures are detectable. Other processors can safely perform actions on behalf of a faulty failstop processor.

Unless the system is synchronous, it is not possible to distinguish between a processor that is executing very slowly and one that has halted due to a crash failure. Yet, the ability to make this distinction can be important. A processor that has crashed can take no further action, but a processor that is merely slow can. Other processors can safely perform actions on behalf of a crashed processor, but not on behalf of a slow one, because subsequent actions by the slow processor might not be consistent with actions performed on its behalf by others. Thus, crash failures in asynchronous systems are harder to deal with than failstop failures. In synchronous systems, however, the crash and failstop models are equivalent.

The next four failure models that is Crash+Link, Receive-Omission, Send-Omission, and General Omission , all deal with message loss, each modeling a different cause for the loss and attributing the loss to a different component. Finally, Byzantine failures are the most disruptive. A system that can tolerate Byzantine failures can tolerate anything.

Now certain questions come to our mind as to , why not define a failure model corresponding to memory disruptions or misbehavior of the processor's arithmetic-logic unit . The first reason brings us back to the two fundamental questions: The feasibility and cost of solving certain fundamental problems is known to differ across the seven failure models discussed above. A second reason that these Failure models involving the contents of memory or the functioning of an ALU, for example, concern internal details of the processor abstraction. A good model encourages suppression of irrelevant details.

Which Model When? : The dilemma faced by practitioners is that of deciding between models when building a system. Should we assume that processes are asynchronous or synchronous, failstop or Byzantine? The answer depends on how the model is being used. One way to regard a model is as an interface definition—a set of assumptions that programmers can make about the behavior of system components. Programs are written to work correctly assuming the actual system behaves as prescribed by the model. And, when system behavior is not consistent with the model, then no guarantees can be made.

For example, a system designed assuming that Byzantine failures are possible can tolerate anything. Assuming Byzantine failures is prudent in mission critical systems, because the cost of system failure is usually great, so there is considerable incentive to reduce the risk of a system failure. For most applications, however, it suffices to assume a more benign failure model. In those rare circumstances where the system does not satisfy the model, we must be prepared for the system to violate its specification.

Lastly, the various models can and should be regarded as limiting cases. The behavior of a real system is bounded by our models. Thus, understanding the feasibility and costs associated with solving problems in these models, can give us insight into the feasibility and cost of solving a problem in some given real system whose behavior lies between the models.

11.5 REMOTE PROCEDURE CALLS

Many distributed systems have been based on explicit message exchange between processes. However, the procedures send and receive do not conceal communication, which is important to achieve access transparency in distributed systems. This problem has long been known, but little was done about it until a paper by Birrell and Nelson (1984) introduced a completely different way of handling communication.

It is an Inter-process communication technology that allows a computer program to cause a procedure to execute in another another computer on a shared network without the programmer explicitly coding the details for this remote interaction. Remote procedure call (RPC) is a

powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports. RPC makes the client/server model of computing more powerful and easier to program.

Conventional Procedure Call : To understand how RPC works, it is important first to fully understand how a conventional (i.e., single machine) procedure call works. Consider a call in C like count = read(fd, buf, nbytes); where *fd* is an integer indicating a file, *buf* is an array of characters into which data are read, and *nbytes* is another integer telling how many bytes to read. If the call is made from the main program, the stack will be as shown in Fig. 11.4(a) before the call. To make the call, the caller pushes the parameters onto the stack in order, last one first, as shown in Fig. 11.4(b). After read has finished running, it puts the return value in a register, removes the return address, and transfers control back to the caller. The caller then removes the parameters from the stack, returning it to the original state.

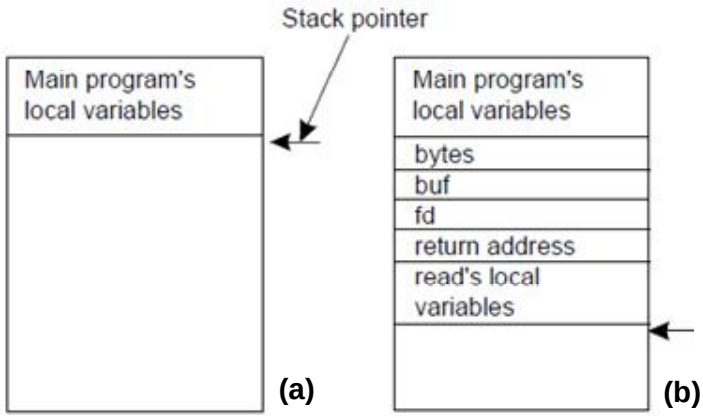


Fig.11.4 : (a) Parameter passing in a local procedure call: the stack before the call to read.

(b) The stack while the called procedure is active.

Client and Server Stubs : The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In other words, we want RPC to be transparent—the calling procedure should not be aware that the called procedure is executing on a different machine or vice versa. Suppose that a program needs to read some data from a file. The programmer puts a call to read in the code to get the data. In a traditional (single-processor) system, the read routine is extracted from the library by the linker and inserted into the object program. It is a short procedure, which is generally implemented by calling an equivalent read system call. In other words, the read procedure is a kind of interface between the user code and the local operating system.

Even though read does a system call, it is called in the usual way, by pushingthe parameters onto the stack, as shown in Fig. 11.4 (b). Thus the programmer does not know that read is actually doing something fishy. RPC achieves its transparency in an analogous way. When read is actually a remote procedure (e.g., one that will run on the file server's machine), a different version of read, called a client stub, is put into the library. Like the original one, it, too, is called using the calling sequence of Fig. 11.4 (b). Also like the original one, it too, does a call to the local operating system. Only unlike the original one, it does not ask the operating system to give it data. Instead, it packs the parameters into a message and requests that message to be sent to the server as illustrated in Fig. 11.5.

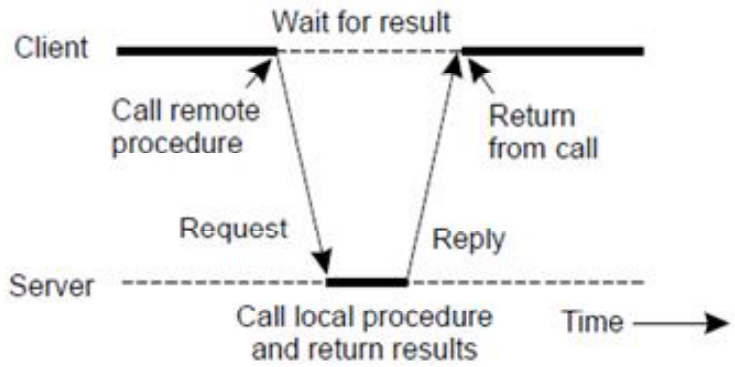


Fig. 11.5 : Principle of RPC between a client and server program

Following the call to send, the client stub calls receive, blocking itself until the reply comes back.

When the message arrives at the server, the server's operating system passesit up to a server stub. A server stub is the server-side equivalent of a client stub: it is a piece of code that transforms requests coming in over the network into local procedure calls. Typically the server stub will have called receive and be blocked waiting for incoming messages. The server stub unpacks the parameters from the message and then calls the server procedure in the usual way (i.e., as in Fig. 2).From the server's point of view, it is as though it is being called directly by the client—the parameters and return address are all on the stack where they belong and nothing seems unusual. The server performs its work and then returns the result to the caller in the usual way. For example, in the case of read, the server will fill the buffer, pointed to by the second parameter, with the data. This bufferwill be internal to the server stub.

When the server stub gets control back after the call has completed, it packs the result (the buffer) in a message and calls send to return it to the client. After that, the server stub usually does a call to receive again, to wait for the next incoming request.

When the message gets back to the client machine, the client's operating system sees that it is addressed to the client process (or actually the client stub, but the operating system cannot see the difference). The message is copied to the waiting buffer and the client process unblocked. The client stub inspects the message, unpacks the result, copies it to its caller, and returns in the usual way. When the caller gets control following the call to read, all it knows is that its data are available. It has no idea that the work was done remotely instead of by the local operating system. This blissful ignorance on the part of the client is the beauty of the whole scheme. As far as it is concerned, remote services are accessed by making ordinary (i.e., local) procedure calls, not by calling send and receive. All the details of the message passing are hidden away in the two library procedures, just as the details of actually making system calls are hidden away in traditional libraries.

To summarize, a remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.

2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

The net effect of all these steps is to convert the local call by the client procedure to the client stub, to a local call to the server procedure without either client or server being aware of the intermediate steps.

CHECK YOUR PROGRESS

Q.7. Byzantine Failures means

a) processor fails by receiving only a subset of the messages
b) processor fails by exhibiting arbitrary behavior
c) fails by transmitting only a subset of the messages
d) A processor fails by halting

Q.8. RPC is uniquely identified by the triple :

a) program number, version number , function number
b) procedure number, program number, version number
c) program number, version number , precedence number
d) procedure number, program number, vision number

Q.9. In making remote procedure call (RPC),

a) the parameters can be passed by reference and pointer values;
b) the parameters can only be passed by value;

c) the parameters can be any objects;

d) the parameters can only be object references.

Q.10. Which of these are not failure model in distributed system,

a) Receive-Omission b) Send-Omission

c) General Omission d) Fault Omission

Q.11. A mobile agent

a) is a complete program, code and data, that can work independently.

b) is sent from one computer to another and run at the destination.

c) is a program code , that can work independently.

d) is a part of program code and data , that can work independently.

11.6 DISTRIBUTED SHARED MEMORY

Let us begin the discussion by understanding what do we mean by shared memory. Shared memory is a method by which program processes can exchange data more quickly than by reading and writing using the regular operating system services. For example, a client process may have data to pass to a server process that the server process is to modify and return to the client. Ordinarily, this would require the client writing to an output file using the buffers of the operating system and the server then reading that file as input from the buffers to its own work space. Using a designated area of shared memory, the data can be made directly accessible to both processes without having to use the system services.

In computer hardware, shared memory refers to a large block of random access memory that can be accessed by several different central processing units in a multiple-processor computer system.

A shared memory system is relatively easy to program since all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location. However it has two complications:

1) CPU-to-memory connection becomes a bottleneck. Shared memory computers can not scale very well. Most of them have ten or fewer processors.
2) Cache coherence: Whenever one cache is updated with information that may be used by other processors, the change needs to be reflected to the other processors, otherwise the different processors will be working with incoherent data.

In computer software, shared memory is either a method of inter-process communication i.e. a way of exchanging data between programs running at the same time. One process will create an area in RAM which other processes can access, or a method of conserving memory space by directing accesses to what would ordinarily be copies of a piece of data to a single instance instead, by using virtual memory mappings.

The alternatives to shared memory are distributed memory and distributed shared memory, each having a similar set of issues.

Distributed Shared Memory (DSM), also known as a distributed global address space (DGAS), is a term in computer science that refers to a wide class of software and hardware implementations, in which each node of a cluster has access to a large shared memory in addition to each node's limited non-shared private memory.

Software DSM systems can be implemented within an operating system, or as a programming library. Software DSM systems implemented in the operating system can be thought of as extensions of the underlying virtual memory architecture. Such systems are transparent to the developer; which means that the underlying distributed memory is completely hidden from the users. In contrast, Software DSM systems implemented at the library or language level are not transparent and developers usually have to program differently. However, these systems offer a more portable approach to DSM system implementation.

Software DSM systems also have the flexibility to organize the shared memory region in different ways. The page based approach organizes shared memory into pages of fixed size. In contrast, the object based approach organizes the shared memory region as an abstract space for

storing shareable objects of variable sizes. Other commonly seen implementation uses tuple space, in which unit of sharing is a tuple.

Shared memory architecture may involve separating memory into shared parts distributed amongst nodes and main memory; or distributing all memory between nodes. A coherence protocol, chosen in accordance with a consistency model, maintains memory coherence.

11.7 DISTRIBUTED FILE SYSTEMS

A distributed file system is a client/server-based application that allows clients to access and process data stored on the server as if it were on their own computer. When a user accesses a file on the server, the server sends the user a copy of the file, which is cached on the user's computer while the data is being processed and is then returned to the server.

Ideally, a distributed file system organizes file and directory services of individual servers into a global directory in such a way that remote data access is not location-specific but is identical from any client. All files are accessible to all users of the global file system and organization is hierarchical and directory-based.

The file service is the specification of what file system offers to its clients. It describes what parameters they take and what action they perform.

A file server is a process that runs on some machine and helps implement the file service. A system may have one file server or several. It only knows that whenever there is a procedure specified in the file service, the required work is performed and the results are returned.

Since more than one client may access the same data simultaneously, the server must have a mechanism in place to organize updates so that the client always receives the most current version of data and that data conflicts do not arise. Distributed file systems typically use file or database replication to protect against data access failures. Sun Microsystems' Network File System (NFS), Novell NetWare, Microsoft's Distributed File System, and IBM/Transarc's DFS are some examples of distributed file systems.

Distributed File System Design : A distributed file system typically has two reasonably distinct components: the File service interface and the Directory service interface.

File Service Interface can be split into two types, depending on whether they support an upload/ download model or a remote access model. In the upload/download model, the file service provide read and write operations.The read file operation transfers the requested file from one of the file servers to the requesting client. The write operation transfers file from client to the file server. Here is the diagram showing the upload/ download model:

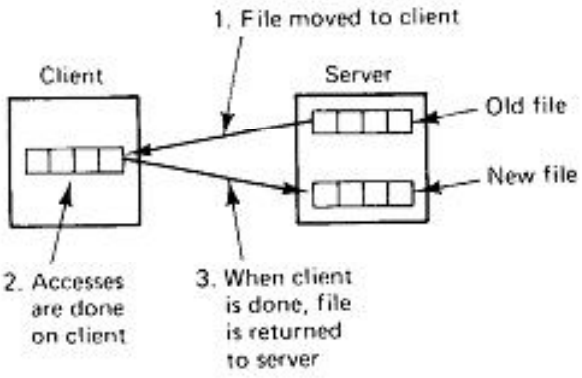


Fig. 11.6 (a) : A upload/download model

In the remote access model , the file service provides a large number of operations for opening and closing files, reading and writing parts of files, moving around within files, examining and changing file attributes and so on. It has the advantage of not requiring much space on the clients as well as eliminating the need to pull in entire files when only small pieces are needed. Below is the diagram showing this model.

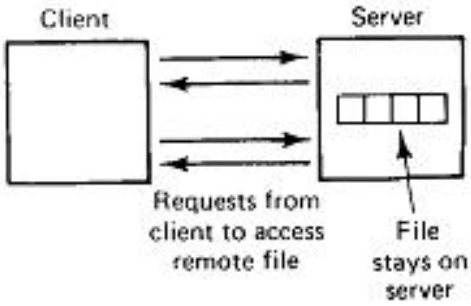


Fig. 11.6 (b) : Remote access model

Directory Server Interface defines some alphabet and syntax for composing file and directory names. All distributed systems allow directories to contain subdirectories, to make it possible for users to group related files together.This results in hierarchical file system. Two of the directory systems are given below

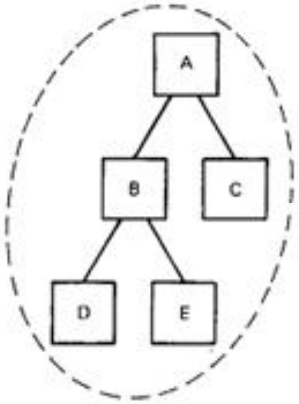


Fig. 11.7 (a) : A direcory tree contained on one machine

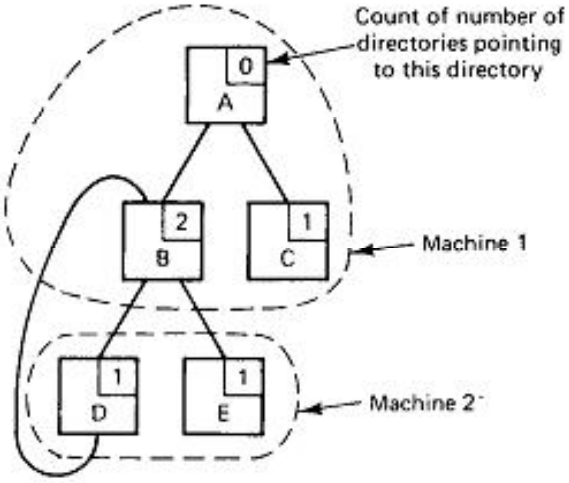


Fig. 11.7 (b) : Adirectory graph on two machines

In fig. 11.7 (a) a link to the directory can be removed only when the directory pointed to is empty. In fig. 11.7 (b) this can happen only if other link exists.This problem is serious in case of a distributed system compared to a centralized system. Further issues include wheteher or not all machines should have the same view of the directory hierarchy in a distributed system. To understand this let us take an example to show that a path /D/E/x is valid on one machine, is also valid on other different directory hierarchy. For this we will take a example:

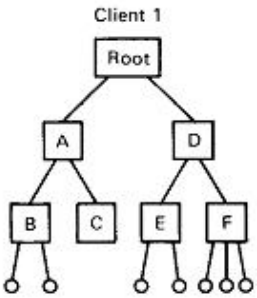


Fig. 11.8 (a) : square are directories and circles are files

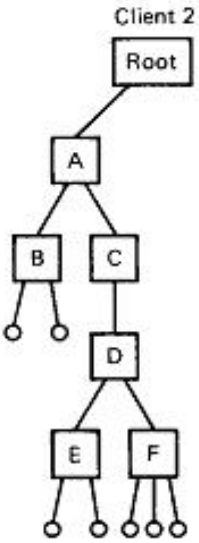


Fig. 11.8 (b) : Square are directories and circles are files

In the above fig. 11.8 (a) and fig. 11.8 (b) it is clearly visible that the path /D/E/x is valid on one but not on the other, thus requiring the same view of the directory hierarchy.

The distributed filesystem should have location transparency and location independence.The first one location transparency, means that the path name gives no hint as to where the file is located. A path like /server1/ dir1/dir2/x tells that the file is located on server 1, but does not tell where it is located. Location independence means files can be moved from one server to another without changing their names. Most distributed system use some form of two level naming. Files have symbolic names such as sample.txt , for use by people, but they can also have internal binary names for use by the system itself.

Semantics of File sharing : There are four ways of dealing with shared files in distributed systems. These are UNIX semantics– Every

operation on a file is instantly visible to all processes. Immutable files -- No updates are possible, simplifies sharing and replication.Transactions -- All changes have the all or nothing property.Session semantics -- No changes are visible to other processes until the file is closed.

CHECK YOUR PROGRESS

Q.12. Which is not the way of dealing with shared files in distributed systems

a) Immutable files b) Transactions
c) Session semantics d) Atomic semantics

Q.13. Achieving mutual exclusion,

a) is only required in distributed systems, and is never needed in single-processor systems;
b) is primarily used in distributed systems to enhance the performance;
c) is necessary for a process to read or write certain shared data;
d) none of the above.

Q.14. Distributed OS works on the _____ principle.

a) File Foundation b) Single system image
c) Multi system image d) Networking image

Q.15. Which of the following is a distributed file system.

a) NTFS b) FAT16 c) FAT32 d) NFS

Q.16. Location transparency

a) allows the movement of resources and clients within a system without affecting the operations of users or programs;
b) enables local and remote resources to be accessed using identical operations;
c) hides whether a resource is in memory or on disk;
d) enables resources to be accessed without knowledge of their location.

11.8 LET US SUM UP

- Distributed system is a collection of independent computers that appear to the users of the system as a single computer.
- A distributed algorithm is an algorithm designed to run on computer hardware constructed from interconnected processors. Distributed algorithms are used in various application areas of distributed computing, such as telecommunications, scientific computing, distributed information processing.
- Distributed processing refers to any of a variety of computer systems that use more than one computer, or processor, to run an application. This includes parallel processing, in which a single computer uses more than one CPU to execute programs.

  Models of distributed systems :
  - Architectural Models
  - Interaction Models
  - Fault Models
- Synchronous Distributed Systems : Lower and upper bounds on execution time of processes can be set. Transmitted messages are received within a known bounded time.
- Asynchronous Distributed Systems : No bound on process execution time nothing can be assumed about speed, load, reliability of computers. No bound on message transmission delays nothing can be assumed about speed, load, reliability of interconnections
- Remote Procedure Call is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or local procedure calling, so that the called procedure need not exist in the same address space as the calling procedure.
- Shared memory refers to a large block of random access memory that can be accessed by several different central processing units (CPUs) in a multiple-processor computer system.

- Distributed File System Design :
  - File Service Interface it provides two major operations, read file and write file
  - Directory Server Interface defines some alphabet and syntax for composing file and directory names.

11.9 ANSWERS TO CHECK YOUR PROGRESS

Ans. to Q. No. 1 : a)
Ans. to Q. No. 2 : c)
Ans. to Q. No. 3 : c)
Ans. to Q. No. 4 : c)
Ans. to Q. No. 5 : b)
Ans. to Q. No. 6 : c)
Ans. to Q. No. 7 : b)
Ans. to Q. No. 8 : b)
Ans. to Q. No. 9 : c
Ans. to Q. No. 10 : d
Ans. to Q. No. 11 : a)
Ans. to Q. No. 12 : d)
Ans. to Q. No. 13 : c)
Ans. to Q. No. 14 : b)
Ans. to Q. No. 15 : d)
Ans. to Q. No. 16 : d)

11.10 FURTHER READINGS

- *Distributed operating Systems* By Tanenbaum.
- *Distributed Systems: Principles and Paradigms* by Andrew S. Tanenbaum and Maarten van Steen.
- *Distributed Operating Systems: Concepts and Design* By Sinha, Pradeep.

- *Distributed Operating Systems And Algorithm Analysis* by Chow Randy.

11.11 MODEL QUESTIONS

Q.1. What is the definition of a distributed system according to DS? What are the three major consequences of this definition of distrib uted systems?

Q.2. What is the purpose of an architectural model?

Q.3. Discuss in brief the problems solved by a distributed algorithms.

Q.4. List the main forms of transparency.Explain the difference between access transparency and location transparency.Give examples of both.

Q.5. Explain how a distributed algorithm is different from a general algorithm?

Q.6. Discuss the Consensus algorithms.

Q.7. Explain the difference between architectural and fundamental models.

Q.8. What are the main aspects captured in the fundamental models?

Q9.What are the three main classes of failures in distributed systems?

Q.10. What does it mean to mask a failure?

Q.11. Explain the main techniques to handle threats

Q.12. Explain the difference between a security model and a threat model.