GCA S3 03

Operating System

SEMESTER III

BACHELOR OF COMPUTER APPLICATION

BLOCK 2



KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta,	Gauhati University
Prof. (Retd.) Pranhari Talukdar,	Gauhati University
Dr. Jyotiprokash Goswami	Assam Engineering College

Course Coordinator

Dr. Sanjib Kr. Kalita,	KKHSOU
Dr. Tapashi Kashyap Das,	KKHSOU
Sruti Sruba Bharali	KKHSOU

SLM Preparation Team

Units	Contributor
9	Dr. Khurshid Alam Borbora, IDOL, Gauhati University
10	Sruti Sruba Bharali, KKHSOU
11	Dr. Pranab Das, Assam Don Bosco University
12	Swapnil Gogoi, IDOL, Gauhati University
13	Swapnil Gogoi, IDOL, Gauhati University
14	Dr. Pranab Das, Assam Don Bosco University

Editorial Team

Content Dr. Sanjib Kr Kalita, KKHSOU

Language (Retd). Prof & HOD English, Robin Goswami, Cotton College

Structure, Format & Graphics: Sruti Sruba Bharali, KKHSOU

June, 2018

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

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

Headquarters : Patgaon, Rani Gate, Guwahati - 781017

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

The University acknowledges with thanks the financial support provided by the **Distance Education Bureau**, **UGC** for the preparation of this study material.

BLOCK INTRODUCTION

This is the *Second Block* of the course “Operating System”. After completing this block you will find yourself at a moderate level of expertise in how operating system works.

This block comprises the following six units:

Unit - 9 introduces you to the different types of memory management.

Unit - 10 concentrates on segementation and paging. Different page replacement algorithms like First In First Out, Least Recenty Used and Optimal page replacement algorithm are discussed in this unit.

Unit - 11 gives us an introduction to file system. Operations on files, sequential and random access are discussed in this unit.

Unit - 12 introduces us to the basics of I/O management system. Basic principles of I/O hardware, software, devices are described in this unit.

Unit - 13 deals with operating system security and protection. Concepts like authentication, protection and crytptography are discussed in this unit.

Unit - 14 deals with multiprocessor operating system. Functional characteristics of multiprocessor OS are discussed in this unit.

The structure of Block 2 is as follows:

UNIT 9 : Memory Management

UNIT 10 : Segmentation and Paging

UNIT 11 : File System

UNIT 12 : I/O Management

UNIT 13 : Security and Protection

UNIT 14 : Introduction to Multiprocessor Systems

BACHELOR OF COMPUTER APPLICATION

OPERATING SYSTEM

DETAILED SYLLABUS

Block-2

		Pages
Unit-9 :	**Memory Management** Memory management requirements, memory partitioning, Fixed and Variable size partitioning, Swapping, Virtual Memory.	**5 – 16**
Unit-10 :	**Segmentation and Paging** Basic Concept of Segmentation, Segmentation Architecture, Address Translation in Segmentation, Basic Concepts of Paging, Page Tables, Address Translation in Paging, Logical and Physical Address, Page replacement algorithms: - First in First Out, Optimal, LRU	**17 – 31**
Unit-11 :	**File System** File Structure, File naming, File types(directory, regular, device), Sequential access and Random access files, File attributes, Operations on file, Directory Structure- Hierarchical directory structure, Path name (relative and absolute), Operation on directories, File System Implementation Techniques.	**32 – 48**
Unit-12 :	**I/O Management** Basic principles of I/O Hardware, I/O Devices, Device controllers, DMA, Principles of I/O Software, Interrupt Handlers, Device Drivers, Device Independent I/O Software	**49 – 60**
Unit-13 :	**Security and Protection** Security threats and goals, Authentication, Protection and Access control, Formal model of protection, Cryptography.	**61 – 72**
Unit-14 :	**Introduction to Multiprocessor Systems** Multiprocessor Introduction, Types of Multiprocessor Operating Systems, Functional Characteristics of Multiprocessor OS, Multiprocessor Synchronization	**73 – 92**

UNIT 9 : MEMORY MANAGEMENT

UNIT STRUCTURE

9.1 Learning Objectives
9.2 Introduction
9.3 Memory Management
9.4 Memory Management Requirements
    9.4.1 Relocation
    9.4.2 Protection
9.5 Memory Partioning
    9.5.1 Fixed Partioning
9.6 Swapping
9.7 Virtual Memory
9.8 Let Us Sum Up
9.9 Further Reading
9.10 Answers To Check Your Progress
9.11 Model Questions

9.1 LEARNING OBJECTIVES

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

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

9.2 INTRODUCTION

In the previous units of block 1, you were introduced to concepts of process, threads and inter process communication. Topics like semaphores, process scheduling and deadlocks were also discussed in detail.

This is the first unit of this block and in this unit, we will introduce concepts like memory management requirements, memory parttioning and swapping. Memory management schemes like fixed partitioning and placement algorithms are discussed in this unit. An introduction to virtual memory is

also included in this unit. Concepts of segmentation and paging will be covered in the next few units.

9.3 MEMORY MANAGEMENT

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.

9.4 MEMORY MANAGEMENT REQUIREMENTS

While surveying the various mechanisms and policies associated with memory management, it is needed 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.

9.4.1 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 difficult to say when it is next swapped 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 issues related to addressing, as shown in Fig. 9.1. The figure 9.1 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. 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.

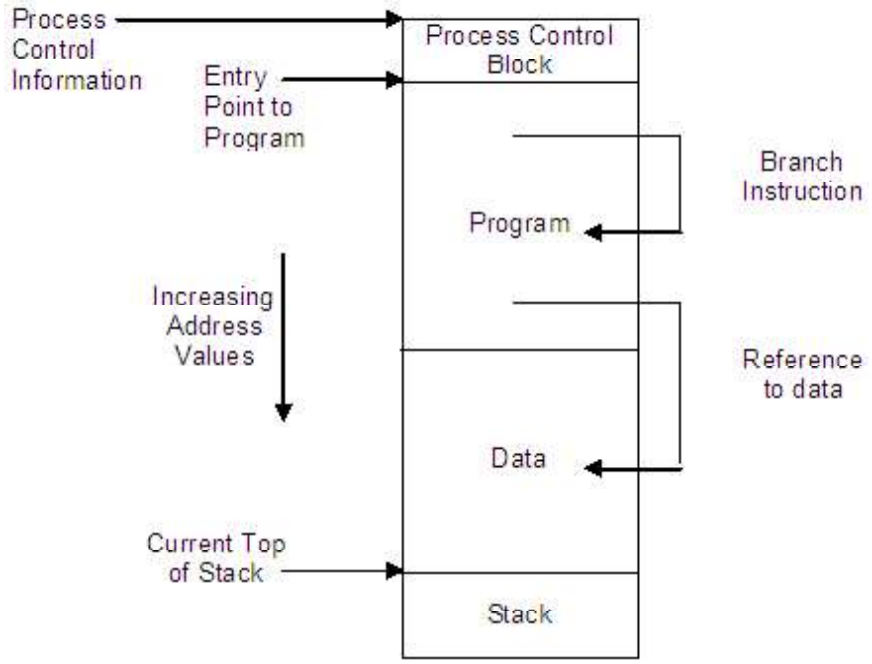


Fig 9.1 Adressing Requirements for a Process

9.4.2 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.

9.5 MEMORY PARTITIONING

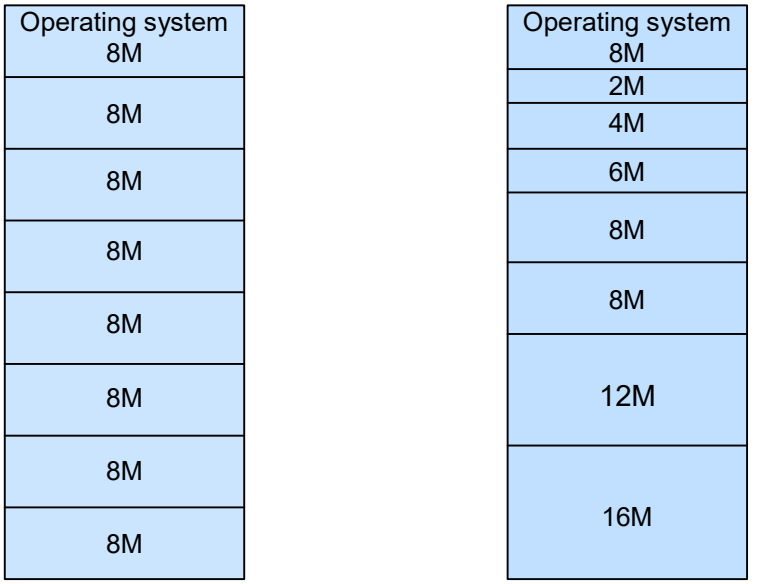
The principal operation of memory management is to bring processes into main memory for execution by the processor. In almost all modern multiprogramming systems, this involves a sophisticated scheme known as virtual memory. But before we can look at virtual memory techniques, we must prepare the ground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

9.5.1 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 9.2 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. 9.2 : 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 MB; yet it occupies an 8 MB 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. 9.2 b). In this example, programs as large as 16 MB can be accommodated without overlays. Partitions smaller than 8 MB 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. The process to swap out is a scheduling decision.

There are two possible ways to assign processes to partitions with unequal-size partitions. The simplest way is to assign each process to the smallest partition within which it will fit. In this case, a scheduling queue is needed for each partition, to hold swapped-out processes destined for that partition (Fig. 9.3a).

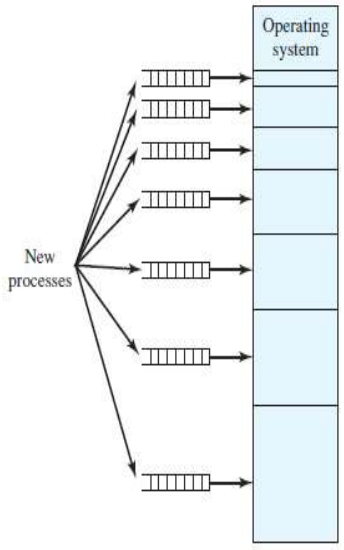


Fig. 9.3 (a) : One process queue per partition

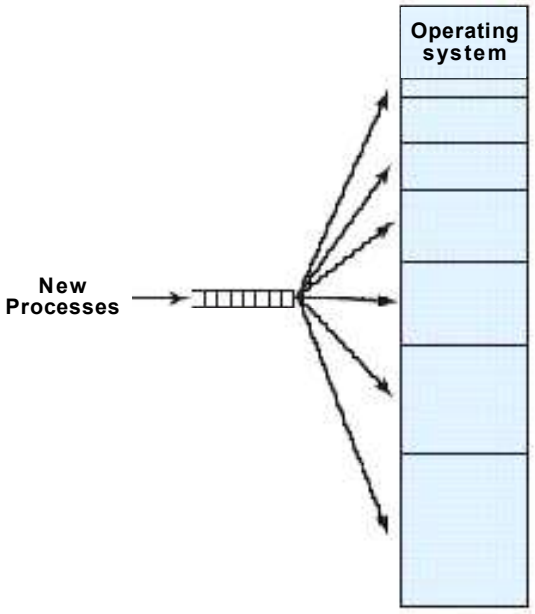


Fig. 9.3 (b) : Memory Assignment for Fixed Partioning

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 Figure 9.3 (b), 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 as in Figure 9.3 (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 obsolete 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).

9.6 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 (Figure 9.4).

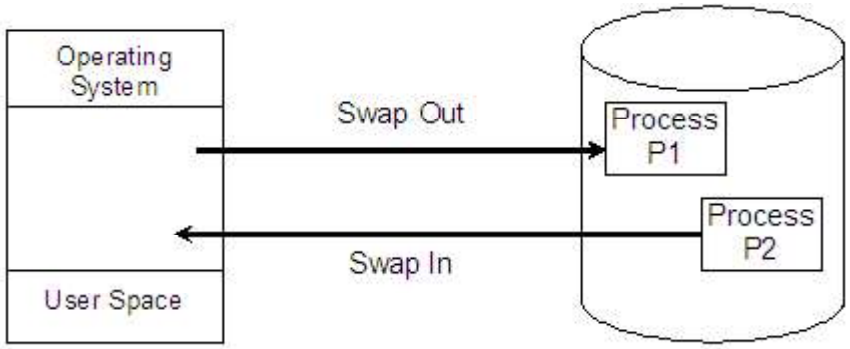


Fig 9.4 Swapping of two processes using a disk as a backing store

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 load and execute the higher-priority process. After 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. 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.

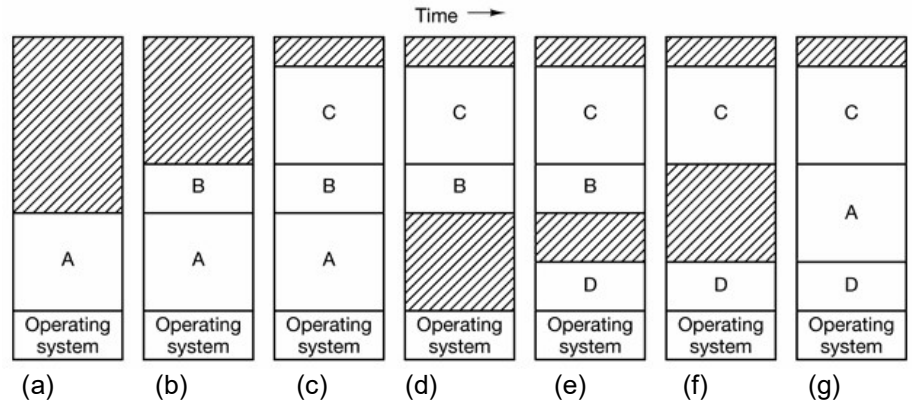


Fig 9.5 : Memory allocation changes

In figure 9.5, the shaded regions are unused memory.The operation of a swapping system is illustrated here. Initially only process *A* is in memory. Then processes *B* and *C* are created or swapped in from disk. In Fig. 9.5(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.

9.7 VIRTUAL MEMORY

Let us now move to the concept of virtual memory. It is a storage allocation scheme in which secondary memory can be thought of as 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. Some terminologies associated with virtual memory are :

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.



CHECK YOUR PROGRESS

Q1. 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.

d) 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.

Q2. 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) 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.



9.8 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 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.



9.9 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.

2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice Hall.

9.10 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1. (a) True, (b) False, (c) True, (e) True.

Answer to Q2. (a) (iv), (b) (i), (c) (iii)

9.11 MODEL QUESTIONS

Q1. Explain the concept of relocation?

Q2. What is memory partitioning? Describe a memory partioning technique.

Q3. What is virtual memory?

Q4. Define the terms – Virtual Address, Virtual Address Space, Address Space, Real Address.

Q5. Describe the process of swapping.

Q6. Differentiate between fixed equal and unequal size partitioning.

UNIT 10 : SEGMENTATION AND PAGING

UNIT STRUCTURE

10.1 Learning Objectives
10.2 Introduction
10.3 Segmentation
    10.3.1 Segmentation Architecture
10.4 Paging
10.5 Page Replacement
    10.5.1First In First Out
    10.5.2 Optimal
    10.5.3 Least Recently Used (LRU)
10.6 Let Us Sum Up
10.7 Further Reading
10.8 Answers to Check Your Progress
10.9 Model Questions

10.1 LEARNING OBJECTIVES

After going through this unit you will be able to:

- learn the concept of segmentation
- describe segmentation architecture
- define paging and page tables
- describe address translation in segmentation and paging
- describe different page replacement algorithms

10.2 INTRODUCTION

In the previous units, you have been introduced to the concept of memory management. Continuous memory allocation and different types of partioning has been covered in the previous unit in addition to equal and unequal size fixed partitioning.

In this unit you will be introduced to the concepts of segmentation and paging. The architecture and address translation for both segmentation and paging are introduced in this unit. Different page replacement algorithms are also covered in this unit.

Once you are familiar with the different memory management techniques in this unit, then in the next unit you will be able to know the basics of file systems. Sequential and random access of files and different operations on files will be covered in the next units.

10.3 SEGMENTATION

We already have an idea of memory management in computer systems including different memory management schemes like fixed size partitioning and variable size partitioning. In this section we will present a new memory management scheme known as segmentation.

Segmentation is a variable sized memory management scheme that views the different parts of a program as segments. Let us consider any computer program. The whole program can be viewed as different parts such as variables, data, functions and procedures, and header files. Each of these sub-parts of a program can be considered as separate segments and can be stored separately. For example, a C program can be divided into different segments as shown in Figure 10.1. We can store these different parts or segments of the program separately. The order of storage of these segments is not of importance.

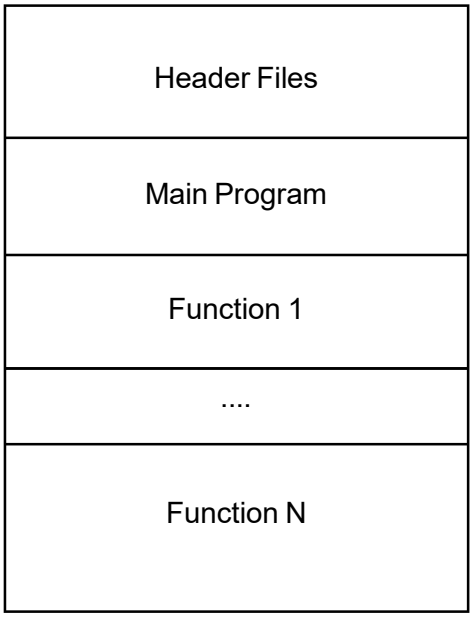


Fig 10.1: Different segments in a C program

10.3.1 Segmentation Architecture

Segmentation supports the user's view of memory. The logical address space is a collection of segments. Each of these segments has a segment number and length of the segment. Segments are referred by a segment number. A logical address has two parts - the segment number and an offset as depicted in figure 10.2. The offset gives the position within that segment.

The logical address needs to be translated into physical address of the computer memory. A mapping table is used to translate the logical address to physical address. This table is also known as segment table. The segment table has two parts - base address and limit address. The base address contains the starting address of the segment in the physical memory. The limit address on the other hand contains the length of the segment.

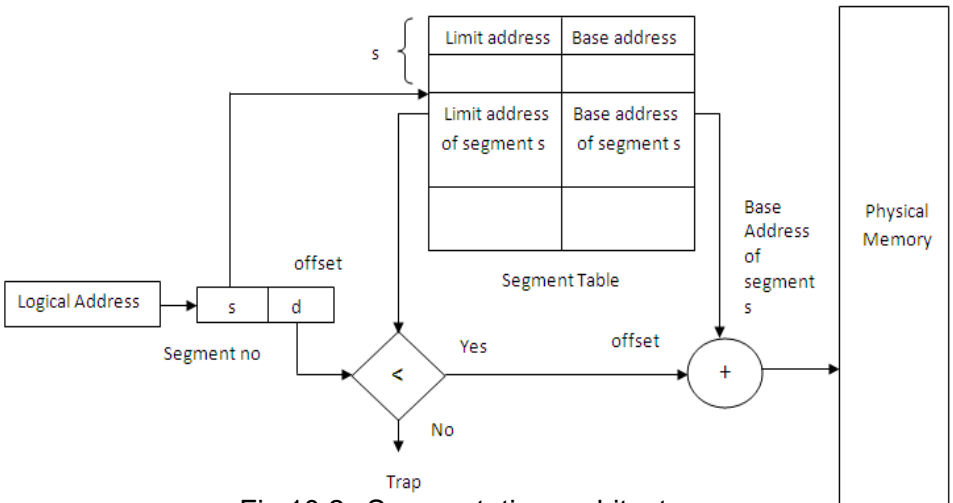


Fig 10.2 : Segmentation architecture

Address translation

The logical address consists of two parts - a segment number (s), and an offset into that segment (d). The segment number is used as an index to get the starting address from the segment table. The offset (d) is the displacement within the segment (s). The segment number provided in the logical address matches with the index of the segment table. Then the offset is checked if it is within the limit that is specified for that segment in the segment table. If the offset is greater than the limit specified for that segment then it generates a trap. But, if the offset is within the limit specified for that segment then the offset is added to the base address for that segment to get the physical address. This can be understood clearly from the segmentation architecture figure shown in Figure 10.2.

Let us look at an example to understand the concept of segmentation. In figure 10.3 we have five segments numbered from 0 to 4. The base address and the limit address of the segments are given in the segment table. The physical storage of the segments is also shown. Now, for example, the logical address (2, 600) means that the segment number is 2 and the offset is 600. The offset is then compared with the limit address given for segment 2 in the segment table. Since the offset 600 is less than the limit 1200 for segment 2 so the offset is added to the starting or base address for that segment: (600+2200=2800). The physical address for the logical address (2, 600) is 2800. Now, let us try to map another logical address (4, 120) to its physical address. To map (4, 120), first the offset 120 will be compared with the limit of segment number 4 which is 700. Since the offset is within the limit so, the base address 7800 will be added to the offset (120+7800=7920) to get the physical location of that address. Let us try to map another logical address (3, 660) to its physical address. To map (3, 660), first the offset 660 will be compared with the limit of segment number 3 which is 600. Since the offset is more than the limit of segment 3 so a trap will be generated when we want to access the physical memory for this logical address.

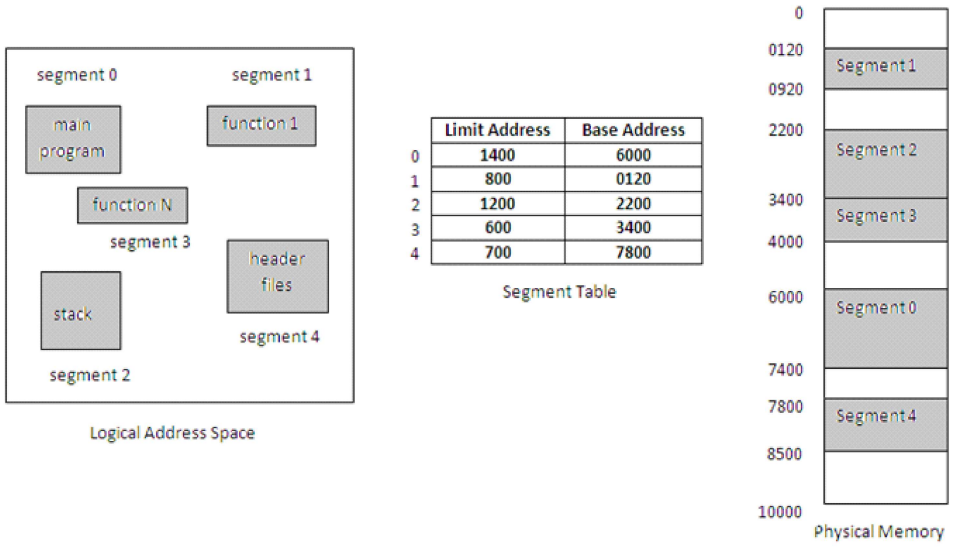


Fig 10.3: Segmentation example

External and Internal Fragmentation

Internal fragmentation is not present in segmentation since all the segments are allocated physical memory according to their segment sizes. But external fragmentation can be found. There are two solutions to external fragmentation. The first solution is to use compaction i.e. shuffling the free memory into one compact memory. The second solution to external fragmentation can be the

use of virtual memory addressing by combining segmentation with paging. We will learn about paging memory management scheme in the next section.

10.4 PAGING

Paging is a fixed size memory management scheme. The physical memory space is divided into fixed sized blocks called *frames*. The logical memory space is also divided into same fixed sized blocks called *pages*. When a process needs to be executed, the pages are loaded into any available memory frames in the physical memory space. Since all pages are of same size, it means that any page can be fitted into any available frame location at the physical memory. The allocation of the pages for any process does not need to be contiguous. The physical memory frames are numbered and the number of the frame allocated to each page of the process is stored in a page table. Each process is associated with a page table that stores the frame number at which each page of the process is stored at. The frame size and the page size in paging is generally a power of 2, like 512 bytes, 1 MB, 4 MB etc. The frame size is selected as power of 2 because it makes translation of logical to physical address easy.

As in segmentation, in paging also the logical address consists of two parts– ***a page number (p) and a page offset (d)***. The page number (p) is used as the index to the page table. The page table contains the starting address of the frames of the process in the physical memory. The offset (d) is combined to this starting address of the frame (f) to get the desired location in physical memory. This can be seen from figure 10.4.

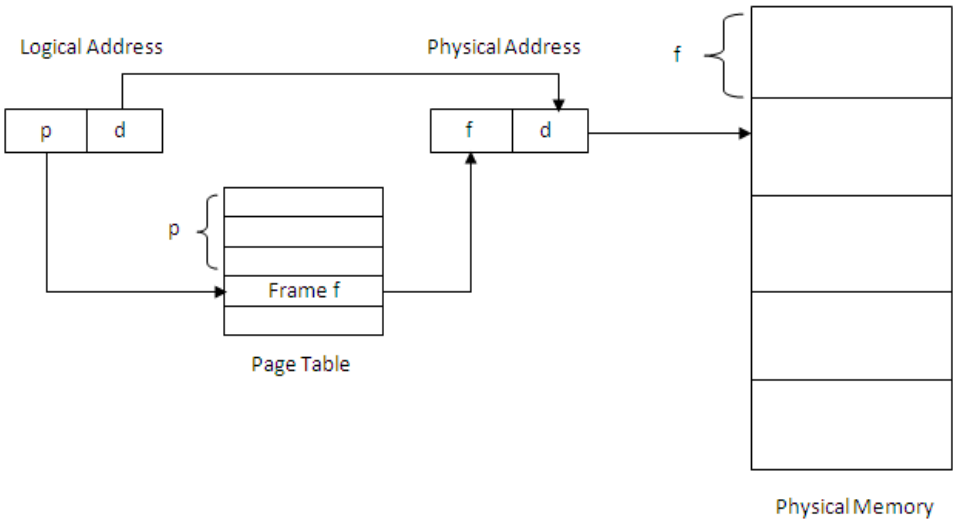


Fig 10.4 : Paging architecture

Let us look an example to understand the concept of paging as stated in figure 10.5. The logical address space of a process P is divided into four pages. All the four pages of the process have been allocated frames in the physical memory. The frames are not allocated in a sequential order ; rather any free frame can be allocated to any page of the process. In this example, page 0 is allocated to frame 5, page 1 is allocated frame 2, page 2 is allocated frame 4 and page 3 is allocated frame 1. It can be seen in the figure that the physical memory holds the allocated pages of the process.

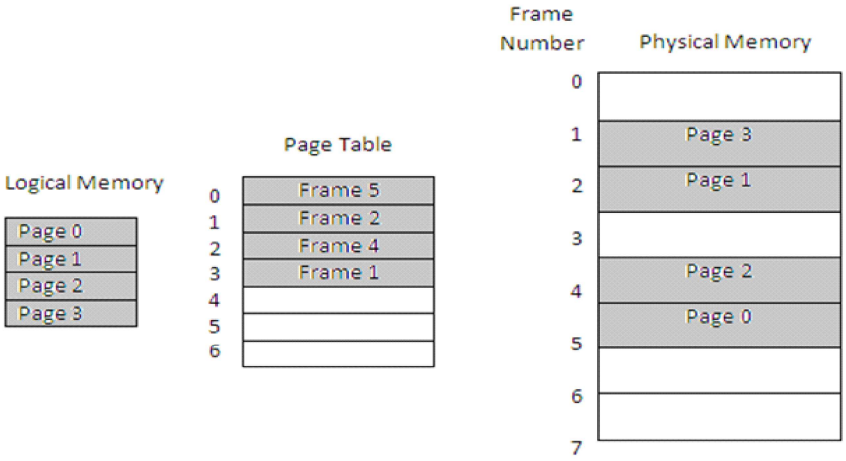


Fig 10.5 : Paging example

External and Internal Fragmentation

External fragmentation is not present in paging memory management scheme since any frame from the physical memory can be allocated to any page. However, internal fragmentation can be present in some cases. For any process, the length of the last page of that process may be less than the frame size that is assigned to it. In this case, internal fragmentation may arise. For example, suppose a frame and page size of 2 MB is considered. Let us consider a process P1 of size 7 MB. The process will be divided into four pages and will need four frames in physical memory for storage. The first three pages cover the whole length of the frames, but the last page will only cover 1 MB size of the frame. So, in the frame allocated to the last page of process P1, a size of 1 MB will be left unused and which cannot be allocated to any other process as a result internal fragmentation will occur.



CHECK YOUR PROGRESS

Q1. Does segmentation suffer from internal and external fragmentation?

Q2. Define paging.

10.5 PAGE REPLACEMENT

Till now, we have learnt that a process is divided into fixed size pages and these pages are loaded into frames of same sizes in the physical memory. If all the pages of a particular process are allocated physical memory, then the process can be executed without any hurdle. But, sometimes there may not be free frames to load all the pages of a particular process. When a page is requested to be executed but the page has not been allocated any frame in physical memory then a page fault occurs.

The page for which page fault occurs needs to be allocated a frame in physical memory. To allocate a frame to this page in the physical memory, we will need to replace this page with an existing page since there are no free frames available. To do this, a page replacement strategy is needed. There are many page replacement algorithms which can be used to replace the existing page with the desired page. Some of the popular page replacement algorithms are:

- First In First Out
- Optimal Page Replacement
- Not Recently Used
- Second Chance
- Least Recently Used

Let us look at some page replacement algorithms in detail in the next few sub-sections.

10.5.1 First In First Out

First In First Out (FIFO) is one of the most simple page replacement algorithm that is used in paging. The FIFO page replacement algorithm follows the principle that the page that was first allocated a frame in the physical memory will be replaced first. In other words, the page that has been in the physical memory for the longest time is the one chosen to be replaced. Let us look an example to understand the concept of First In First Out page replacement algorithm.

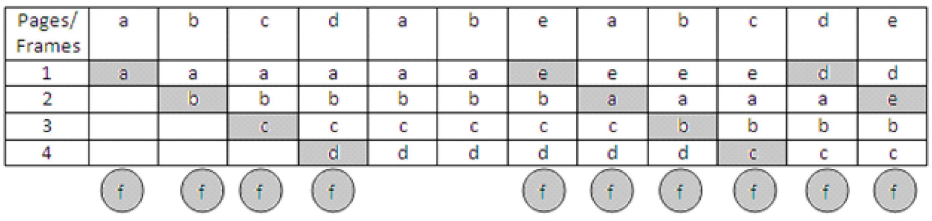


Fig 10.6 : FIFO page replacement example

A FIFO page replacement example has been given in figure 10.6. A process P is considered in this example and has five pages - a, b, c, d and e. Let us suppose that the physical memory has a size of only four frames and all the four frames are free at the beginning. The order in which the five pages of process P are executed is given in the example. The first page to be accessed is "a". Since this page is not found in the frames a page fault is generated and page "a" is loaded into the first free frame i.e., frame 1, of the physical memory. The next page to be accessed is page "b". Page "b" is not found in the physical memory so a page fault is generated and page "b" is loaded into frame "2" of the physical memory. Similarly, when pages "c", "d" are accessed they are not found in the physical memory and page fault is generated. So, page "c" and "d" are loaded into the remaining free frames, frames 3, and 4 of the physical memory. The next page to be accessed again is "a", but since page "a" has already been loaded into frame 1, so we do not need to load it again. This is also true for the next page to be accessed again "b", which is already loaded into frame 2. The next page to be accessed is page "e", which is not present in the physical memory. This causes a page fault. Now, at this point, if we had any free frames then page "e" could have been easily loaded into any free frame. But none of the four frames are free at this point so page replacement is needed to be done. According to FIFO policy, the page that has been in the memory for the longest time period needs to be replaced. The page that has been longest in the memory is page "a", hence page "a" has been replaced with page "e". The next page to be accessed is page "a", which is now no longer present in the physical memory. Hence, a page fault is generated. Since no free frames are available, page replacement needs to be done. According to FIFO algorithm, page "b" has been in the memory for the longest period so page "b" will be replaced with page "a". Now, again the next page to be accessed is page "b", which has just been replaced. So, a page fault will occur and again the page which has been in the memory for longest period will be replaced. So, page "c" will be replaced by page "a". Similarly, when we try to access page "c" next another page fault will occur and page "d" will be replaced with page "c". The next page to be accessed now is page "d", which is not in the physical memory, so a page fault will occur and using FIFO policy page "e" will be replaced by page "d". The last page to be accessed is page "e". Since page "e" is no longer in the physical memory, page fault occurs. Using FIFO algorithm, page "a", which has been in the memory for the longest time is replaced by page "e".

Page replacement algorithms with lesser number of page faults are preferred.

The aim of every page replacement algorithm is to limit the number of page faults. The number of page faults when we use First In First Out algorithm depends on the order in which the pages are accessed. In this example, the total number of page faults when we used FIFO page replacement algorithm is ten (10). Let us now look at other page replacement algorithms.

10.5.2 Optimal Page Replacement

Optimal page replacement algorithm is as the name suggests the best page replacement algorithm. Optimal algorithm is also known as Belady's page replacement algorithm. In optimal page replacement algorithm, the page whose next use is the farthest in the future is the one to be replaced. In other words, the page that is not going to be used for the longest time in future should be the one to be replaced. Let us look at an example to understand this concept.

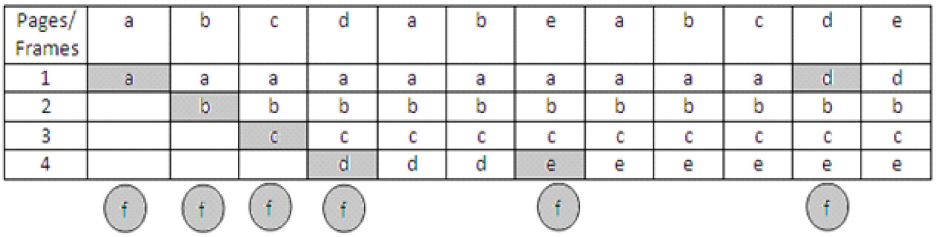


Fig 10.7 : Optimal page replacement example

An optimal page replacement example has been shown in figure 10.7. The same problem was described earlier in example 10.6 using FIFO algorithm. Optimal page replacement algorithm is applied to the problem now as shown in figure 10.7. The process P has five pages - a, b, c, d and e and the physical memory has a size of only four frames and all the four frames in the physical memory are free at the beginning. The order in which the five pages of process P are executed is same as in the earlier FIFO example as shown in figure 10.6.

The first page to be accessed is "a". Since this page is not found in the frames a page fault is generated and page "a" is loaded into the first free frame, frame 1, of the physical memory. The next page to be accessed is page "b". Page "b" is not found in the physical memory so a page fault is generated and page "b" is loaded into frame "2" of the physical memory.

Similarly, when pages "c", "d" are accessed they are not found in the physical memory and page fault is generated. So, page "c" and "d" are loaded into the remaining free frames, frames 3, and 4 of the physical memory. The next page to be accessed again is "a", but since page "a" has already been loaded into frame 1, so we do not need to load it again. Similarly, the next page to be accesses again is "b", which is also already loaded in frame "2". The next page to be accessed is "e", which is not present in any frames of the physical memory. So, a page fault occurs and the page to be replaced will be decided using optimal page replacement policy. According to page replacement strategy, the page whose use is farthest in the future is the one to be replaced. If we look at the next order of pages to be accessed, it can be seen that page "d" is the page out of the four pages in physical memory which needs to accessed last. So, page "d" is replaced with page "e". The next page to be accessed is "a", which is already loaded into the physical memory in frame "1". The next two pages to be accessed "b" and "c" are also already loaded into the physical memory in frame "2" and "3". The next page to be accessed is page "d" which is not present in the physical memory. So, a page fault is generated and a page has to be replaced in the physical memory. Since the next page to be accessed is "e" any page other than "e" can be replaced. In this case, we have replaced page "a" with page "d". The next page to be accessed is "e" which is already present in the physical memory.

In this example, the total number of page faults when we use optimal page replacement algorithm is six (6). We can see that the number of page faults is less when we use optimal page replacement algorithm in place of FIFO. Optimal page replacement algorithm is better than FIFO page replacement algorithm but there is a problem. The problem with optimal page replacement algorithm is that it is unrealistic. We cannot know which pages will be used and in what order the pages will be used in the future. So, it is difficult to predict which page should be replaced. But, in cases where we know the order in which pages are to be accessed in the future, we can use optimal page replacement algorithm.

10.5.3 Least Recently Used

Least recently used (LRU) page replacement algorithm is another popularly used page replacement algorithm. Least recently used algorithm can be said to be opposite of optimal page replacement algorithm. In optimal page

replacement algorithm we look at the future to decide which page should be replaced. Whereas, in least recently used page replacement algorithm, we look at the past to decide the page which has to be replaced. In least recently used page replacement algorithm, the page whose last use was the farthest in the past is the one to be replaced. In other words, the page that we have not used for the longest time in the past should be the one to be replaced. In this strategy, it is considered that pages have frequently used in the past may be used frequently again in the future. Let us look at an example to understand this concept.

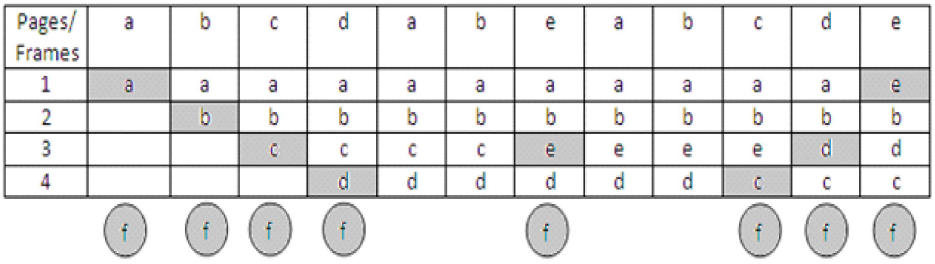


Fig 10.8 : Least recently used page replacement example

A least recently used page replacement example has been shown in figure 10.8. The same problem was described earlier in example 10.6 using FIFO and in 10.7 using optimal page replacement algorithm. Least recently used page replacement algorithm is applied to the problem now as shown in figure 10.8. The process P has five pages - a, b, c, d and e and the physical memory has a size of only four frames and all the four frames in the physical memory are free at the beginning. The order in which the five pages of process P are executed is same as in the earlier two examples as shown in figure 10.6 and in 10.7. The first page to be accessed is "a". Since this page is not found in the frames a page fault is generated and page "a" is loaded into the first free frame, frame 1, of the physical memory. The next page to be accessed is page "b". Page "b" is not found in the physical memory so a page fault is generated and page "b" is loaded into frame "2" of the physical memory. Similarly, when pages "c", "d" are accessed they are not found in the physical memory and page fault is generated. So, page "c" and "d" are loaded into the remaining free frames, frames 3, and 4 of the physical memory.. The next page to be accessed is "a", but since page "a" has already been loaded into frame 1, so we do not need to load it again. Similarly, the next page to be accessed is page "b" which is already loaded in frame "2" of physical memory so we do not need to load it again. The next page to be accessed is page "e" which is not present in the physical memory. This causes a page fault and

the one of the pages in the frames needs to be replaced with "e". According to the least recently used policy, the page that was used last will be the one to be replaced. The last page to be used was page "c" so page "c" will be replaced with page "e". The next two pages to be accessed are page "a" and "b" which are already present in the memory. The next page to be accessed is page is page "c" which is not present in any frame hence a page fault is generated. According to least recently used strategy, page "d" in frame "4" is replaced with page "c". The next page to be accessed is page "d" which has just been removed from the frames in physical memory. This causes a page fault and according to least recently used algorithm page "e" will be replaced with page "d". The last page to be accessed is page "e" which has again just been removed from the physical memory causing a page fault. Page "a" will be replaced with page "e" in the physical memory.

In this example, the total number of page faults when we used optimal page replacement algorithm is eight (8). We can see that the number of page faults is more when we use optimal page replacement algorithm but less when using FIFO.



CHECK YOUR PROGRESS

Q3. Define page fault.
Q4. How does LRU page replacement algorithm choose a replacement page?
Q5. What is the difficulty in using optimal page replacement algorithm



10.6 LET US SUM UP

- Segmentation is a variable sized memory management scheme that views the different parts of a program as segments. The whole program can be viewed as different parts such as variables, data, functions and procedures, and header files. Each of these sub-parts of a program can be considered as separate segments and can be stored separately.
- The logical address needs to be translated into physical address of the computer memory. A mapping table is used to translate the logical address to physical address. This table is also known as segment table.

- The logical address consists of two parts - a segment number (s), and an offset into that segment (d). The segment number is used as an index to get the starting address from the segment table. The offset (d) is the displacement within the segment (s).
- Internal fragmentation is not present in segmentation memory management scheme since all the segments are allocated physical memory according to their segment sizes. But external fragmentation can be found in segmentation.
- Paging is a fixed size memory management scheme. The physical memory space is divided into fixed sized blocks called frames. The logical memory space is also divided into same fixed sized blocks called pages.
- In paging also the logical address consists of two parts - a page number (p) and a page offset (d). The page number (p) is used as the index to the page table. The page table contains the starting address of the frames of the process in the physical memory. The offset (d) is combined to this starting address of the frame (f) to get the desired location in physical memory.
- External fragmentation is not present in paging memory management scheme since any frame from the physical memory can be allocated to any page. However, internal fragmentation can be present in some cases. For any process, the length of the last page of that process may be less than the frame size that is assigned to it. In this case, internal fragmentation may arise.
- When a page is requested to be executed but that page has not been allocated any frame in physical memory then a page fault occurs.
- To allocate a frame to this page in the physical memory, we will need to replace this page with an existing page since there are no free frames available. To do this, a page replacement strategy is needed.
- The FIFO page replacement algorithm follows the principle that the page that was first allocated a frame in the physical memory will be replaced first.
- In optimal page replacement algorithm, the page whose next use is the farthest in the future is the one to be replaced. In other words, the page that is not going to be used for the longest time in future should be the one to be replaced.

- In least recently used page replacement algorithm, the page whose last use was the farthest in the past is the one to be replaced. In other words, the page that we have not used for the longest time in the past should be the one to be replaced.



10.7 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.
2. Dhamdhere, D. M. (2006). Operating Systems: A Concept-based Approach, 2E. Tata McGraw-Hill Education.
3. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice-Hall.



10.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: Segmentation suffers from external fragmentation. But internal fragmentation is not present is segmentation memory management technique.

Answer to Q2: Paging is a fixed size memory management scheme. The physical memory space is divided into fixed sized blocks called frames. The logical memory space is also divided into same fixed sized blocks called pages. When a process needs to be executed, the pages are loaded into any available memory frames in the physical memory space.

Answer to Q3: If all the pages of a particular process are allocated physical memory, then the process can be executed without any hurdles. But, sometimes there may be not be enough free frames to load all the pages of a particular process. When a page is requested to be executed but that page has not been allocated any frame in physical memory then a page fault occurs.

Answer to Q4: In least recently used page replacement algorithm, the page whose last use was the farthest in the past is the one to be replaced. In other words, the page that we have not used for the longest time in the past should be the one to be replaced.

Answer to Q5: The problem with optimal page replacement algorithm is that it is unrealistic. We cannot know which pages will be used and in what order the pages will be used in the future.



10.9 MODEL QUESTIONS

Q1. Define segmentation.
Q2. Describe how of segmentation works.
Q3. Define paging.
Q4. Why do we need page replacement?
Q5. Define page fault.
Q6. Describe the process of paging.
Q7. Describe the FIFO page replacement algorithm with example.
Q8. Describe the LRU page replacement algorithm with example.
Q9. Describe the Optimal page replacement algorithm with example.
Q10. Which page replacement algorithm is best among FIFO, Optimal and LRU.
Q11. Find the number of page faults for
   i. FIFO
   ii. LRU
   iii. Optimal

   Suppose there 3 frames in the physical memory and the order of pages are:
   [P, Q, R, S, P, T, R, Q, P, S, T, P, Q]

******

UNIT 11 : FILE SYSTEM

UNIT STRUCTURE

11.1 Learning Objectives
11.2 Introduction
11.3 File System
  11.3.1 File and File Naming
11.4 File Types
  11.4.1 Regular Files
  11.4.2 Directory Files
  11.4.3 Device Files
11.5 File Attributes
11.6 Operations on File
11.7 Hierarchical Directory Structure
  11.7.1 Single Level Directory
  11.7.2 Two Level Directory
  11.7.3 Tree Structured Directory
  11.7.4 Acyclic Graph Directories
11.8 Operation on Directories
11.9 File System Implementation Techniques
11.10 Let Us Sum Up
11.11 Further Reading
11.12 Answers to Check Your Progress
11.13 Model Questions

11.1 LEARNING OBJECTIVES

After going through this unit, you will able to :

- define a file and identify its attributes
- describe different 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

11.2 INTRODUCTION

In the previous units, we have learnt about different memory managment schemes like fixed and variable size partitioning, segmentation and paging. We have also learnt about page faults and different page replacement algorithms. In this unit, we will learn about the concepts of file system. We will learn about the different types of files like regular, directory and device files. The hierarchical directory structure has also been discussed in detail in this unit. In the next unit, we will explore the concepts related to input and output management.

11.3 FILE SYSTEM

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 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.

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 to 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.

11.3.1 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.

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 based on the operating system.

- **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 as 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 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, DOS, some words might also be reserved and can not be used as filenames. For example, DOS Device file:CON, PRN, AUX, CLOCKS, NUL, COM0 to COM9, LPT0 toLPT9.

11.4 FILE TYPES

Till this point, we have understood 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 from that of an executable file. An executable file must have a specific structure to be executed. 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. The following are some different types of files.

11.4.1 Regular Files

A simple definition of a regular file would be that it is a one dimensional arrangement 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 represented as a stream of bytes within an EOF condition. The EOF condition indicates the End of File. 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 (LINUX OS) 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.

11.4.2 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. 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.

11.4.3 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. The files in **/dev** folder are used to ensure that we can access hardware such as the printer, cdrom, network etc. If you look at the way Linux OS 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.

11.5 FILE ATTRIBUTES

A **file attribute** is a metadata that describes or is associated with a computer file. For example, an operating system often keeps track of date for a file that is created and last modified, as well as the file’s size and extension.

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 gives the operating system and application software more information about the file and how it is intended to be used.

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: A read only file is a file that is not possible to delete or modify.

Hidden: If the file is marked as hidden then under normal circumstances it is invisible. There are some commands to convert hidden file to normal and normal file to hidden.

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.

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 files and 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.

11.6 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. Let us 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 file where 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 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 hard disk, monitor etc.

Most of the file operations mentioned above 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**. The per process table keeps track of all files that a process has open. 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 does 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.

11.7 HIERARCHICAL DIRECTORY STRUCTURE AND PATH NAMES

In this section we will be discussing some of the logical structures of a directory. A directory is a location to store subdirectories and files.

11.7.1 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 structure of single level directory is depicted in Figure 11.1.

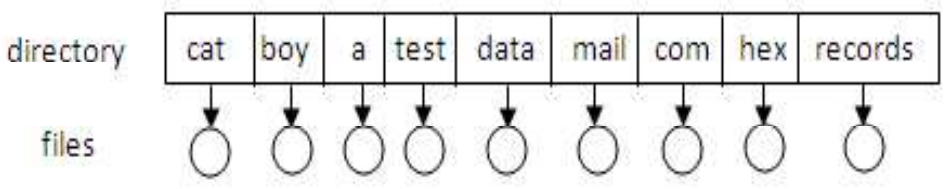


Fig 11.1 : One level directory

A single level directory has significant limitations, however when the number of files increases or when the system has 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.

11.7.2 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. It has been depicted in figure 11.2.

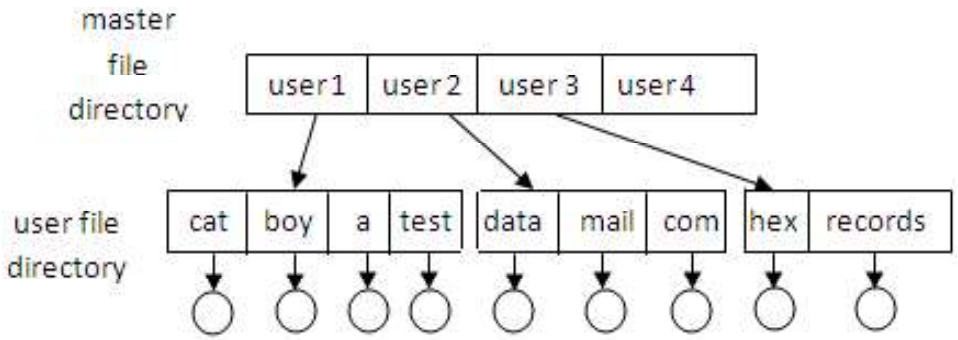


Fig 11.2 : Two level directory

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 is 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.

11.7.3 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. The tree has a root directory. Every file in the system has a unique path name. A **path name** is the path from thr root. A path specifies a unique location in a file system. In other words, it points to a 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 characters are most commonly used the slash ("/"), the backslash character ("\"), or colon (":"), though some operating systems may use a different delimiter. 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.

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. 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. A tree structured directory structure is given in Figure 11.3.

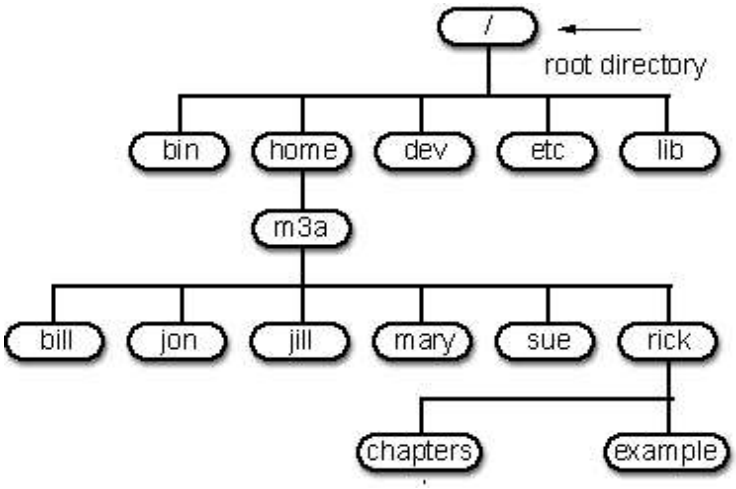


Figure 11.3: Tree structed directory

11.7.4 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 acyclic structure,

this 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. The acyclic graph directories structure has been depicted in Figure 11.4.

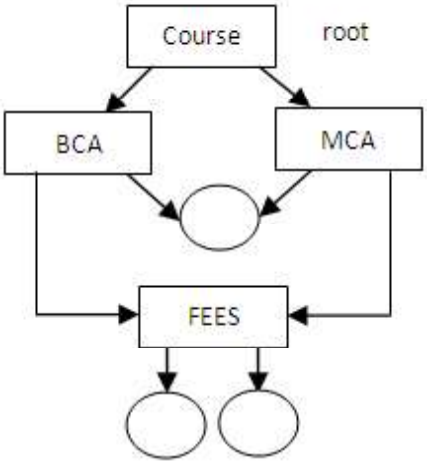


Fig 11.4 Acyclic graph directory

11.8 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.

11.9 FILE SYSTEM IMPLEMENTATION TECHNIQUES

The file system provides the mechanism for online storage and access of file contents, including data and programs. It 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. Figure 11.5 shows 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 controller, which interfaces the I/O to the rest of the system.

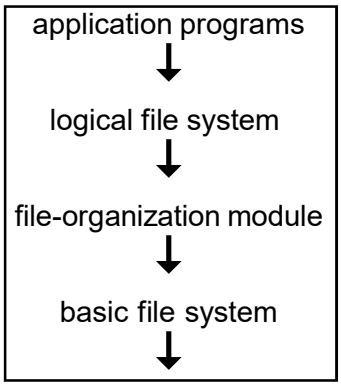


Figure 11.5 : Layered design of a file system

The basic file system needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address, 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

Q1. A file name consist of _____ part.

a) one
b) two
c) three
d) four

Q2. Which of the following can be used as filename in DOS?

a) CON
b) LPT10
c) PRN
d) AUX

Q3. Which of the following is not a file attribute?
a) System
b) Archive
c) Volume label
d) Type

Q4. 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

Q5. 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

Q6. FCB stands for
a) File System Control Block.
b) File Control Block.
c) File Count Block.
d) File Control Blog.



11.10 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 file** is a one dimensional arrangement 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 fil**e 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.



11.11 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Addison-wesley.
2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice Hall.



11.12 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1. b
Answer to Q2. b
Answer to Q3. d
Answer to Q4. d

Answer to Q5. a
Answer to Q6. b



11.13 MODEL QUESTIONS

Q1. Define a file. Discuss the rules for naming a file by taking a suitable example to explain.

Q2. What are reserved characters and words for a given operating system file naming?

Q3. What is file type? Compare 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 the attributes of a file.

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

Q8. What is the role of a directory structure? 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 the layered file system.

******

UNIT 12 : INPUT AND OUTPUT (I/O) MANAGEMENT

UNIT STRUCTURE

12.1 Learning Objectives

12.2 Introduction

12.3 I/O Management

12.3.1 Basic Principle of I/O Hardware

12.4 I/O Devices

12.5 Device Controllers

12.6 Direct Memory Access (DMA)

12.6.1. Working Principle of DMA

12.7 Interrupts and Interrupt Handler

12.8 Device Drivers

12.9 Let Us Sum Up

12.10 Further Reading

12.11 Answers to Check Your Progress

12.12 Model Questions

12.1 LEARNING OBJECTIVES

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

- learn how I/O management is done by OS
- describe the basic principle of I/O hardware
- describe the different types I/O management techniques.
- describe the device controllers, device drivers, etc.

12.2 INTRODUCTION

In the previous units we have studied how resources like processor and main memory are managed. We have also learnt about the different types of a files and file hiearchical structure. We shall now examine the I/O

management. Humans interact with machines by providing information through IO devices such as mouse, keyboard etc. Clearly, the management of all these devices can affect the throughput of a system. For this reason, input output management also becomes one of the primary responsibilities of an operating system. In this unit, we shall examine the role of operating systems in managing IO devices. In particular, we shall examine how the end use of the devices determines the way they are regulated for communication with either humans or with systems. In the next unit, we will explore the security and protection related issuse of operating system.

12.3 I/O MANAGEMENT

Let us first examine the context of input output in a computer system. We shall look at issues initially from the point of view of communication with a device. When we analyze device communication, we notice that communication is required at the following three levels:

- The need for a human to input information and receive output from a computer.
- The need for a device to input information and receive output from a computer.
- The need for computers to communicate (receive/send information) over networks.

The first kind of IO devices operates at good spread for humans to interact. These may be character-oriented devices like a keyboard or an event driven device like a mouse. Usually, human input using a keyboard will be a few key depressions at a time. This means that the communication is rarely more than a few bytes. Also, the mouse events can be encoded by a small amount of information (just a few bytes). Even though a human input is very small, it is stipulated that it is very important, and therefore requires an immediate response from the system. A communication which attempts to draw attention often requires the use of an interrupt mechanism or a programmed data mode of operation. Interrupt as well as programmed data mode of IO shall be dealt with in detail later in this unit.

The second kind of IO operation arises from devices like tapes and disk which have a very high character density. With these characteristics, it is not possible to regulate communication with devices on a character by character basis. The information transfer, therefore, is regulated in blocks of information. Additionally, sometimes this may require some kind of format

control to structure the information to suit the device and/or data characteristics. For instance, a disk drive differs from a line printer or an image scanner. For each of these devices, the format and structure of information is different. It is observed that the rate at which a device may provide data and the rates at which other device receive it may be considerably different. In spite of these differences, the OS should provide uniform and easy to use IO mechanisms. Usually, this is done by providing a buffer. The OS manages this buffer so as to be able to comply with the requirements of both the producer and consumer of data.

The third kind of IO requirements emanates from the need to negotiate system IO with the communications infrastructure. The system should be able to manage communications traffic across the network. This form of IO facilitates access to internet resources to support e-mail, file-transfer amongst machines or web applications. Additionally, now we have a large variety of options available as access devices. These access devices may be in the form of Personal Digital Assistant (PDA), or mobile phones which have infrared or wireless enabled communications. This rapidly evolving technology makes these forms of communications very challenging. However it can be said that most network cards are direct memory access (DMA) enabled to facilitate DMA mode of IO with communication infrastructure. We shall discuss DMA in later section. The second kind of devices operates over a much wider range. Printers operate at 1 to 2 KB per second, disks transfer at rates of 1 MB per second or more. The graphics devices fall between these two ranges while the graphics cards may in fact be even faster. The devices communicate with a machine using **a data bus** and a **device controller**. Essentially, all these devices communicate large data blocks. However, the communication with networks differs from the way the communication takes place with the block devices.

12.3.1 Basic Principle of I/O Hardware

Major components of a computer system are: CPU, memories (primary/ secondary), I/O system consists of I/O devices, device controllers and communication strategies.

I/O devices are generally of two types (i) Block devices and (ii) Character devices.

(i) Block devices are the I/O devices which are store information in fixed-sized blocks typical sizes : 128-1024 bytes.

(ii) Character devices are the I/O devices which delivers/accepts stream of characters.

- Device controllers, which connects physical device to system bus (Minicomputers, PCs)
  – Mainframes use a more complex model: Multiple buses and specialized I/O computers (I/O channels)
- Communication strategies can be categories as Memory-mapped I/O, controller registers , Direct Memory Access – DMA.

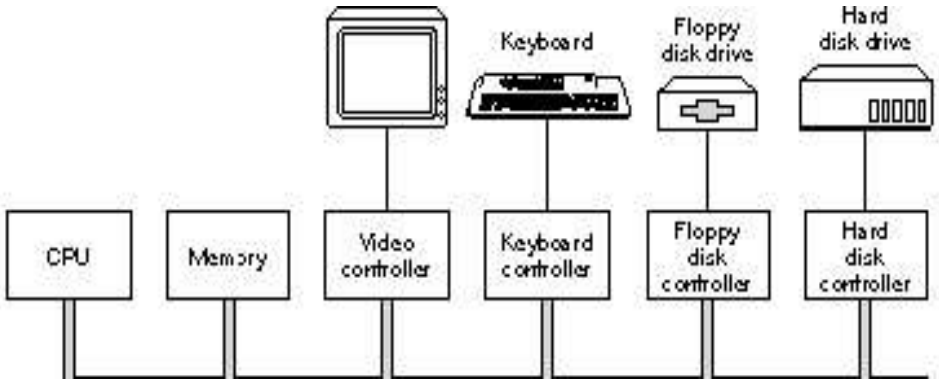


Fig. 12.1: Depicts the different I/O hardware along with their bus connection

We know that the role of the operating system in computer I/O is to manage and control I/O operations and I/O devices. Computer operates with different kinds of devices such as storage devices(disk, tapes), transmission devices (network cards, modems), and human –interface devices (screen, keyboard, mouse).

12.4 I/O DEVICES

Input/Output(I/O) devices are the devices through which we can provide data and instructions to the computer system. These devices can be categories as (i) *block devices*, (ii) *character devices*, (iii) *network devices*, and (iv) *clock and timers*.

- **Block devices:** It store information in fixed size blocks, each one with its own address which has following characteristics
  - ❍ common block sizes : 128 Bytes- 1024 Bytes
  - ❍ it is possible to read or write each block independently
  - ❍ example : disks

- **Character devices:** It delivers or accepts a stream of characters, without regard to any block structure having following charactersistics.
  - ❍ it is not addressable and does not have any seek operation
  - ❍ examples: terminals, line printers, paper tapes, punch cards, network interfaces, and most other devices that are not disk-like
- Clocks or memory mapped screens
- **Device drivers:** low-level software hiding the device specific details to the file system (deals with abstract block devices)

12.5 DEVICES CONTROLLERS

As we know that devices are the mechanical component of I/O unit. Device controllers can be defined as follows. It is electronic component of I/O unit (often printed circuit card), through which the operating system deals with. Communication between CPU and controllers may be two types they are (i) micro- and mini-computers: single-bus model (ii) mainframes: multiple buses and specialized I/O computers (I/O channels)

12.6 DIRECT MEMORY ACCESS (DMA)

The *Direct Memory Access* (DMA) technique allows computers to free up the processor by allowing each of the peripherals to directly access the memory. DMA is used for transferring data between the local memory and the main memory. Computers that have DMA can transfer data to and from devices with much less CPU overhead. Similarly a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time and allowing computation and data transfer concurrency.

Without DMA, the CPU is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU would initiate the transfer, do other operations while the transfer is in progress, and receive an interrupt from the DMA controller once the operation has been done.

12.6.1 Working Principle of DMA

DMA allows devices such as CPU, disk controller to transfer data without using the CPU for longer period. Otherwise, the CPU would have to copy each piece of data from the source to the destination, making CPU unavailable for other tasks. This situation is aggravated because access to I/O devices over a peripheral bus is generally slower than normal system RAM.

A DMA transfer copies a block of memory from one device to another. While the CPU initiates the transfer by issuing a DMA command, it does not execute it. A typical usage of DMA is copying a block of memory from system RAM to or from a buffer on the device. Such an operation does not stall the processor, which as a result can be scheduled to perform other tasks.

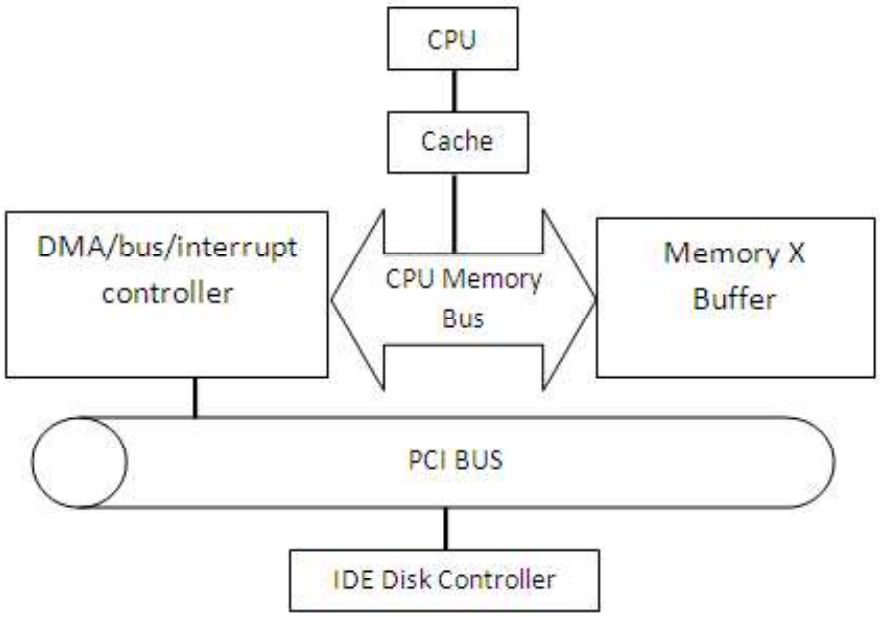


Fig. 12.2: Steps in a DMNA transfer

Data transfer through DMA follows the following steps:

1. Device driver is told to transfer disk data to buffer at address X.
2. Device driver tells disk controller to transfer C bytes from disk to buffer at address X.
3. Disk controlller initiates DMA transfer.
4. Disk controller sends each byte to SMA controller.
5. DMA controller transfer bytes to buffer X, increasing memory address and decreasing C until C=0.

6. When C=0, DMA interrupts CPU to signal transfer completion.

DMA can lead to cache coherency problems. Imagine a CPU equipped with a cache and an external memory that can be accessed directly by devices using DMA. When the CPU accesses location X in the memory, the current value will be stored in the cache. Subsequent operations on X will update the cached copy of X, but not the external memory version of X. If the cache is not flushed to the memory before the next time a device tries to access X, the device will receive a stale value of X.

12.7 INTERRUPTS AND INTERRUPT HANDLERS

A signal that informs a program that an event has occurred is called interrupt. When a program receives an interrupt signal, it takes a specified action (which can be to ignore the signal). Interrupt signals can cause a program to suspend itself temporarily.

The operating system may specify a set of instructions, called an interrupt handler, to be executed in response to each type of interrupt. This allows the operating system to gain control of the processor to manage system resources.

A processor may generate an interrupt as a result of executing a process's instruction(in which case it is often called a trap and is said to be synchronous with the operation of the process).

Interrupt signals can come from a variety of sources. For example, every keystroke generates an interrupt signal. Interrupts can also be generated by other devices, such as a printer, to indicate that some event has occurred. These are called hardware interrupts. Interrupt signals initiated by programs are called software interrupts. A software interrupt is also called a trap or an exception.

PCs support 256 types of software interrupts and 15 hardware interrupts. Each type of software interrupt is associated with an interrupt handler — a routine that takes control when the interrupt occurs. For example, when you press a key on your keyboard, this triggers a specific interrupt handler. The complete list of interrupts and associated interrupt handlers is stored in a table called the interrupt vector table, which resides in the first 1K of addressable memory.

When the user types a key, the console keyboard controller alters some

signals on the bus. This action notifies the CPU that something has changed inside the console keyboard controller. This condition causes the CPU to immediately start running the console keyboard controller's interrupt handler (I-6). The state of the interrupted process (either some other process or the idle loop) is saved so that the process can be returned to its original state as though it had never been interrupted in the first place.

12.8 DEVICE DRIVERS

A device driver, run as part of the kernel software, manages each of the device controllers on the system. It is a set of routines linked into the kernel, which are used as specific peripheral device for proper operations.

Purpose of a Device Driver

The purpose of a device driver is to handle requests made by the kernel with regard to a particular type of device. There is a well-defined and consistent interface for the kernel to make these requests. By isolating device-specific code in device drivers and by having a consistent interface to the kernel, adding a new device is easier.

Types of Device Drivers

A hardware device is a peripheral, such as a disk controller, tape controller, or network controller device. In general, there is one device driver for each type of hardware device. Device drivers can be classified as:

- Block device drivers
- Character device drivers (including terminal drivers)
- Network device drivers
- Pseudo device drivers

Block Device Driver

A block device driver is a driver that performs I/O operations using file system block-sized buffers from a buffer cache supplied by the kernel. The kernel also provides for the device driver support interfaces that copy data between the buffer cache and the address space of a process.
Block device drivers are particularly well-suited for disk drives, the most common block devices. For block devices, all I/O occurs through the buffer cache.

Character Device Driver

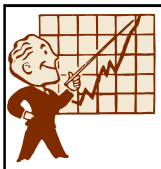
A character device driver does not handle I/O operation through the buffer cache, so it is not tied to a single approach for handling I/O. Character device driver are useful for devices such as a line printer that handles one character at a time. However, character drivers are not limited to performing I/O one character at a time (despite the name “character” driver). For example, tape drivers frequently perform I/O in 10K chunks.

Because of their flexibility in handling I/O, many drivers are character drivers. Line printers, interactive terminals, and graphics displays are examples of devices that require character device drivers.

A terminal device driver is actually a character device driver that handles I/O character processing for a variety of terminal devices. Like any character device, a terminal device can accept or supply a stream of data based on a request from a user process. It cannot be mounted as a file system and, therefore, does not use data caching.

Network Device Driver

A network device driver attaches a network subsystem to a network interface, prepares the network interface for operation, and governs the transmission and reception of network frames over the network interface.



CHECK YOUR PROGRESS

Q1. DMA stands for
- a) Data Memory Access
- b) Direct Memory Access
- c) Device Memory Access
- d) Disk Memeory Access

Q2. The I/O devices which are store information in fixed-sized blocks typical sizes: 128-1024 bytes are
- a) Block devices
- b) Character devices
- c) Bit devices
- d) Byte devices

Q3. A ——— is specialized software. It is specifically written to manage communication with a specified class of devices.

a) Device controller
b) Macro device
c) Device driver
d) None of these.

Q4. Which is not a device driver.

a) Block device drivers
b) Character device drivers (including terminal drivers)
c) Network device drivers
d) Operation drivers.

Q5. An interrupt is a

a) A signal informing a program that an event has occurred.
b) A program
c) A device driver
d) None of these

Q6. Which of the following not a character device.

a) Mouse, terminals
b) Line printers, punch cards
c) Paper tapes, network interfaces.
d) Disk.



12.9 LET US SUM UP

- Input output (IO) management becomes one of the primary responsibilities of an operating system. The role of operating systems in managing IO devices remarkable.
- The first kind of IO devices operate at rates good for humans to interact.
- The second kind of IO requirement arises from devices which have a very high character density such as tapes and disks.
- The third kind of IO requirements emanate from the need to negotiate system IO with the communications infrastructure like computer network system.
- I/O devices are generally two types (i) Block devices and (ii) Character devices.

- Block devices store information in fixed size blocks.
- Character device delivers or accepts a stream of characters, without regard to any block structure.
- A device driver is specialized software which is specifically written to manage communication with an identified class of devices.
- DMA is an essential feature of all modern computers, as it allows devices to transfer data without subjecting the CPU to a heavy overhead.
- A DMA transfer copies a block of memory from one device to another.
- DMA can lead to cache coherency problems.
- A key concept in the design of I/O software is known as device independence.
- Another important issue for I/O software is error handling.
- A device driver, run as part of the kernel software, manages each of the device controllers on the system.
- Device drivers can be classified as:
  1. Block device drivers
  2. Character device drivers (including terminal drivers)
  3. Network device drivers
  4. Pseudo device drivers
- The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software.
- Other functions of the device-independent software are given below:
  1. Uniform interfacing for device drivers
  2. Device naming.
  3. Device protection.
  4. Providing a device-independent block size
  5. Buffering.
  6. Storage allocation on block devices.
  7. Error reporting.



12.10 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.

2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice Hall.



12.11 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1. (b)

Answer to Q2. (a)

Answer to Q3. (c)

Answer to Q4. (d)

Answer to Q5. (a)

Answer to Q6. (d)



12.12 MODEL QUESTIONS

Q1. What is I/O management in operating system? Explain.

Q2. Explain the basic principle of I/O hardware and software.

Q3. What is DMA? Write the steps in DMA.

Q4. What is interrupt? What are the functions of interrupt handlers?

Q5. What are device drivers and device controllers? What are the different types of device drivers?

Q6. Explain the function of the device-independent I/O software.

UNIT 13 : SECURITY AND PROTECTION

UNIT STRUCTURE

13.1 Learning Objectives

13.2 Introduction

13.3 Security and Protection

13.4 Security Threats and Goals

13.5 Authentication

13.6 Protection

13.6.1 Protection Domains

13.7 Access Control

13.7.1 Benefits of Access Control

13.7.2 Access Control Lists

13.8 Cryptography

13.8.1 Symmetric Cryptography

13.8.2 Public-key Cryptography

13.9 Let Us Sum Up

13.10 Further Reading

13.11 Answers to Check Your Progress

13.12 Model Questions

13.1 LEARNING OBJECTIVES

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

- learn the basic concept of security and protection of OS
- describe why we need security and protection
- illustrate authentication and access control
- describe the concept of cryptography

13.2 INTRODUCTION

Now-a-days information access and delivery from and to a remote

location via internet is common. Many societal services like railway time-table or election results etc. are rendered through computers. The notion of electronic commerce has given fillip to provisioning commercial services as well. Most individuals use computers to store private information at home and critical professional information at work. In this kind of scenario, information is the key resource and needs to be protected.

In the previous unit, we have learnt about input output management of operating system. Now in this unit, we are going to learn about the security and protection mechanism of operating system.

13.3 SECURITY AND PROTECTION

The operating system being the system's resource regulator, must provide for security mechanisms. It must not only secure the information to protect the privacy but also prevent misuse of system resources. As the use of internet is increasing more and more companies are sharing their information system to their partners and suppliers. Therefore, it is essential to know which of the company's resources need protecting and to control system access and the user rights of the information system. The same is true when opening company access on the Internet. Moreover, in current time, employees are required to connect to information systems from virtually anywhere, so they are required to carry a part of the information system outside of the company's secure infrastructure.

13.4 SECURITY THREATS AND GOALS

The threat represents the type of action that is likely to harm, whereas vulnerability (sometimes called flaws or breaches) represents the level of exposure to threats in a particular context.
In order to secure a system, the potential threats must be identified and categorized to give an idea of how they work in order to know how to limit the risk of intrusion.

There are different security threats as follows:

Viruses and Worms

A worm is executable code that spreads by infecting files over a network. Worms rarely require any user action to propagate, nor do they need to be

attached to another program or file to spread. Once a virus or worm is released, it can spread rapidly, often infecting millions of computers worldwide within minutes or hours through internet.

A virus is a executable code which often sent as an attachment to an e-mail message or hidden in file such as audio clips, video clips and games-that attacks to or overwrites other files to replicate itself. Viruses can corrupt files, control applications or even erase a hard drive. There are different types of virus.

a) Boot sector virus: It is the type of virus which infects the boot sector of the computer's hard disk, allowing it to load with the operating system and potentially control the system.

b) Transient virus: This type of viruses attaches itself to a particular computer program. The virus is activated when the program is run and deactivated when the program is terminated.

c) Resident virus: Once loaded into the memory of a computer, resident virus operates until the computer is powered down.

d) Logic bomb: It is also a special type of virus which executes its code, when a given condition is met. An example of a logic bomb is a time bomb, which is activated when the clock on the computer matches a certain time or date.

Trojan Horse

A Trojan horse, is a program that appears to the user to perform a destructive function but, in fact, facilitates unauthorized access to the user's computer system. Unlike viruses, Trojan horses do not replicate themselves but they can be just as destructive. Additionally, they require interaction with a hacker to fulfil their purpose.

Trojan horses are designed to allow a hacker remote access to a target computer system. Once a Trojan horse has been installed on a target computer system it is possible for a hacker to access it remotely and perform operations. Operations which could be performed by a hacker on a target computer system using Trojan horse are as follows:

- Deletion of files
- Modification of files
- Uploading of files
- Downloading of files

- Installation of software (including other malware)
- Data Theft (e.g. passwords, security codes, credit card information)

So the objective of the Trojan horse would be to look for some patterns in the input programs and replace these with some trap door that will allow the attacker to attack the system at a later time.

Goals of Security

Information systems are generally defined by all of a company's data and the material and software resources that allow a company to store and circulate this data. Information systems are essential to companies and must be protected.

In general there are five goals of security:

- Integrity : guaranteeing that the data are those that they are believed to be
- Confidentiality : ensuring that only authorized individuals have access to the resources being exchanged
- Availability : guaranteeing the information system's proper operation
- Non-repudiation : guaranteeing that an operation cannot be denied
- Authentication : ensuring that only authorized individuals have access to the resources

13.5 AUTHENTICATION

First let us consider how a legitimate user establishes his identity to the system to access permitted resources. This is achieved typically by username/password pair. When the system finishes booting, the user is prompted for a username and then a password in succession. The password typed is not echoed to the screen for obvious reasons. Once the password is verified the user is given an interactive shell from where he can start issuing commands to the system. Clearly, choosing a clever password is important. Too simple a password would be an easy give away and too complex would be hard to remember. So how can we choose a nice password?

Choosing a good password: A malicious user usually aims to obtaining a complete control over the system. For this he must acquire the super user or root status. Usually, he attacks vulnerable points like using badly installed

software. A relatively easier way is to log in as a normal user and search the system for bugs to become super user. To do this, the attacker will have to have a valid user code and password combination to start with. Therefore, it is of utmost importance that all users on a system choose a password which is quite difficult to guess. The security of each individual user is closely related to the security of the whole system. Users often have no idea how a multi-user system works and do not realize that by choosing an easy to remember password, they indirectly make it possible for an attacker to manipulate the entire system. It is essential to educate the users well to avoid lackadaisical attitudes. For instance, if some one uses a certain facility for printing or reading some mails only, he may think that security is unimportant. The problem arises when someone assumes his identity. Therefore, the users should feel involved with the security of the system. It also means that it is important to notify the users of the security guidelines. Or at least make them understand why good passwords are essential.

13.6 PROTECTION

This section will concentrate on some technical ways that are used in operating system to protect files and other resources. All of these techniques makes a clear distinction between policy (whose data are to be protected) and mechanism (how the system enforces the policy). Our emphasis will be on the mechanism not the policy.

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.

13.6.1 Protection Domains

A computer system contains many objects which are very essential to be protected. These objects can be hardware(CPU, memory segments, disk drives or printers) or they can be software(processes, files, databases or semaphores)

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 make sense on a

semaphore. It is obvious that a way is needed to prohibit 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. For example, process A may be entitled to read, but not write.

Protection mechanisms is 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.

An important question is how the system keeps track of which object belongs to which domain. Conceptually, at least, one can envision a large matrix, with the rows being the domains and the columns being the objects. Each box lists the rights, if any, that the domain contains for the object.

Object	File1	File2	File3	File4	Printer1
D1	read	Read write			
D2			read	Read write execute	write
D3			Read write		write

Fig 13.1: A protection matrix

13.7 ACCESS CONTROL

Most access control mechanisms originate from a stated security policy. It is important to learn to design a security policy and offer suitable access mechanisms that can support security policies. Security policy models have evolved from many real-life operating scenarios. For instance, if we were to follow a rule of defense forces, we may resort to a hierarchy based policy. In such a policy, 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. For a more detailed discussion the reader may refer where there is a discussion on how to specify security policies as well.

As file systems became larger and computers process a larger share of business transactions, the need for data security becomes essential. Prior to computerization it was standard practice for all organization that confidential information to be locked away when not in use. Only authorized personnel were granted permissions to read, write, amend and delete such information.

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 relevant 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.

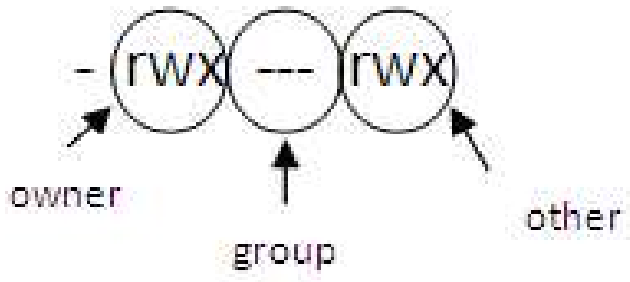
13.7.1 Benefits of Access Control

Access Control(AC) provides additional protection for server resources. It enables a strong defense-in-depth security practice while reducing the complexity and cost of managing access and reaching compliance.

The benefits of access control are :
1. Advanced policy management to enable efficient centralized management of security policies across the enterprise.
2. Policy-based compliance reporting of user entitlements and policy compliance.
3. Operating system hardening to reduce external security risks and ensure operating environment reliability.
4.Granular, high-integrity auditing for compliance fulfillment.

13.7.2 Access Control Lists

On Unix systems, file permissions define the accessibility of a file by an user. 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 rest of the world. 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. Example of file permission is:



13.8 CRYPTOGRAPHY

Cryptography is the science of information security . The word cryptography is originated from the Greek word “kryptos”, which means hidden. Cryptography focuses on encoding and decoding data so that it can be

interpreted only by the intended recipients.

Cryptographic techniques allow a sender to change the appearence of data so that an intruder cannot gain any 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 (active intruder).The art of breaking ciphers, called cryptanalysis, and the art devising them (cryptography) is collectively known as cryptology.

13.8.1 Symmetric Cryptography

In case of symmetric cryptography, the same secret key is used to encrypt and decrypt a message. Here the sender encrypts a message using the secret key, then sends the encrypted message to the intended recipient, then the recipent decrypts the message using the same secret key. A limitation of symmetric cryptography is that before two parties can communicate securely, they must find secure way to exchange the secret key. Symmetric cryptography is also known as secret-key cryptography.

13.8.2 Public-key Cryptography

In Public-key cryptography, two inversely related keys are used: a public key and a private key. The private key is kept secret by its owner and the public key is freely distributed. If the public key encrypts a message, only corresponding private key can decrypt it and vice versa. Each party in a transaction possesses both a public key and a private key. To transmit a message securely, the sender uses the receiver's public key to encrypt the message. The receiver then decrypts the message, using his or her unique private key. Here the private key is kept secret, so the message cannot be read by anyone other than intended receiver.



CHECK YOUR PROGRESS

Q1. Which is not a computer security threat

a) viruses c) Trojan horse

b) worms d) time bomb

Q2. The file mode contains ———------ bits that determine access permissions to the file plus ———-----—special bits.

a) Nine , three b) Eight, three

c) Nine, four d) Seven, two

Q3. Five goals of security are

a) Availability, Integrity, Authentication, Non-replication, Portability

b) Integrity, Confidentiality, Atomicity, Non-repudiation, Authentication

c) Availability, Integrity, Authentication, Confidentiality, Non-repudiation

d) Integrity, Confidentiality, Availability, Non-repudiation, Encryption

Q4. A ————is executable code that spreads by infecting files over a network.

a) worm c) Trojan horse

b) antivirus d) none of these

Q5. A———— do not replicate themselves.

a) worm c) Trojan horse

b) virus d) none of these

Q6. A mechanism by which the authorized users of the system can be identified.

a) authentication b) protection

c) confidentiality d) none of these

Q7. The message to be encrypted, known as

a) the plain text b) the cipher text

c) the bold text d) none of these

Q8. The encrypted message is known as

a) the plain text b) the cipher text

c) the bold text d) none of these

Q9. Cryptography means
a) encryption and decryption b) encoding
c) decoding d) all the above

Q10. Public-key cryptography involves two keys
a) Public key and personal key b) Public key and private key
c) Private key and personal key d) Public key and protected key



13.9 LET US SUM UP

- Protection and security on information and resources in our computer system is one of the most important responsibility of Operating System.
- There are various security threats like viruses and worms, Trojan horse etc.
- There are several goals of security like : Integrity ,Confidentiality, Availability ,Non-repudiation ,Authentication.
- Authentication is a mechanism by which the authorized users of the system can be identified.
- Protection is the some of the detailed technical ways which are used in operating system to protect files and other resources.
- Access control mechanisms emanate from a stated security policy. It is important to learn to design a security policy and offer suitable access mechanisms that can support security policies.
- Cryptography is the science of information technology by which 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.



13.10 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.

2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice Hall.

13.11 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1. (d)
Answer to Q2. (a)
Answer to Q3. (c)
Answer to Q4. (a)
Answer to Q5. (c)
Answer to Q6. (a)
Answer to Q7. (a)
Answer to Q8. (b)
Answer to Q9. (d)
Answer to Q10. (b)



13.12 MODEL QUESTIONS

Q1. What is security and protection mechanism in operating system?

Q2. What are the security threats in operating system? Explain briefly.

Q3. What is authentication? Explain briefly.

Q4. What is access control? What are its benefits?

Q5. Write short notes on cryptography.

****

UNIT 14 : INTRODUCTION TO MULTIPROCESSOR SYSTEMS

UNIT STRUCTURE

14.1 Learning Objective

14.2 Introduction

14.3 Multiprocessor System and Interconnections

14.4 Types of Multiprocessor Operating System

- 14.4.1 CPU with its own Operating System
- 14.4.2 Master-Slave Multiprocessors
- 14.4.3 Symmetric Multiprocessors
- 14.4.4 Asymmetric Multiprocessors

14.5 Multiprocessor OS Functions and Requirements

14.6 Multiprocessor Synchronization

14.7 Let Us Sum Up

14.8 Further Reading

14.9 Answers to Check Your Progress

14.10 Model Questions

14.1 LEARNING OBJECTIVES

After going through this unit, you will able to :

- define multiprocessor computer
- describe interconnections among the multiprocessor computer
- learn about the functions of operating system
- describe how performance of the overall system is improved by synchronization

14.2 INTRODUCTION

In the previous unit, we have learned about the security and protection issues of operating system. The different types of virus and worms were also explained in detail. Topics like authentication, protection and access control were discussed in detail. In this unit, we will eplore the concept of

multiprocessor systems. Different types of multiprocessor systems are discussed in this unit along with their functions and requirements.

14.3 MULTIPROCESSOR SYSTEM AND INTERCONNECTIONS

Multiprocessing is a type of processing in which two or more processors work together to process more than one job 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 other. They share common data structures and a common system clock.

Some advantages of multiprocessor system are as follows:

- **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.

Interconnection networks make a major factor to differentiate modern multiprocessor architectures. They can be categorized according to a number of criteria such as topology, routing strategy and switching technique. Interconnection networks are build up of switching elements; topology is the pattern in which the individual switches are connected to other elements, like processors, memories and other switches.

Direct topologies connect each switch directly to a node, while in *indirect* topologies at least some of the switches connect to other switches.

Using switching technique as a criterion one can mention some classes:

- *circuit switching*, in which the entire path through the network is reserved before a message is transferred,
- *packet switching* with *virtual cut through*, in which a packet is forwarded immediately after it determines an appropriate switch output,
- *wormhole routing*, which relaxes requirements of completely buffering of blocked packets in a single switch, typical for *packet switching*. Wormhole routing is currently the most popular technique in commercial parallel machines.

Queuing technique is another important factor in switching network architectures, since it strongly influences the aggregated bandwidth of the network. In the simple *input queuing* technique the packets queue at the switch input awaiting the availability of the desired switch output; higher performance is offered by *output queuing* which in turn is difficult to implement. A solution is to form multiple queues at each switch input or to create a central buffer shared among all switch inputs and outputs.

The networks can be classified as *static* or *dynamic*. Static interconnection networks are mainly used in message-passing architectures. The following figure shows some topological arrangement of processors.

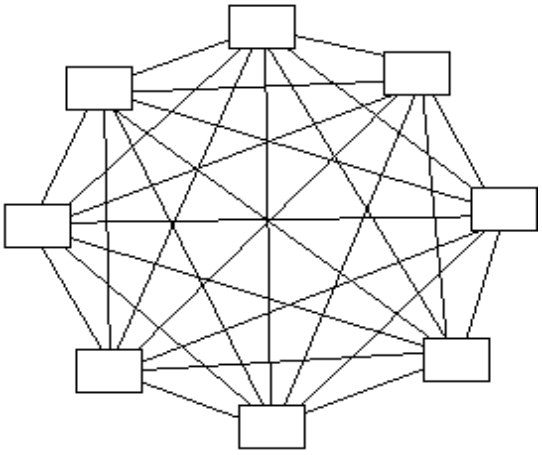


Fig 14.1 : Completely-connected network.

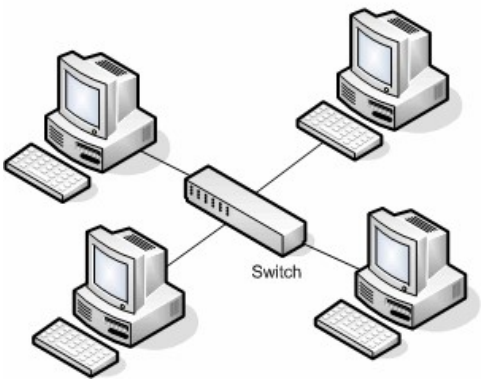


Fig 14.2: Star-connected network.

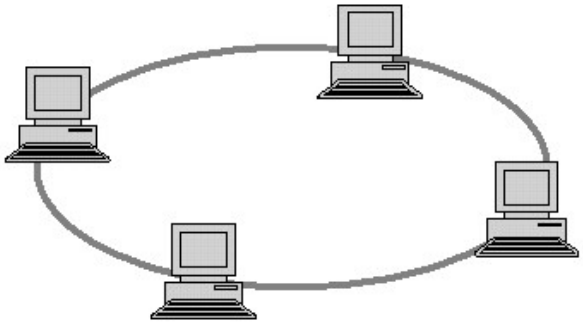


Fig 14.3: Linear array of ring of processors.

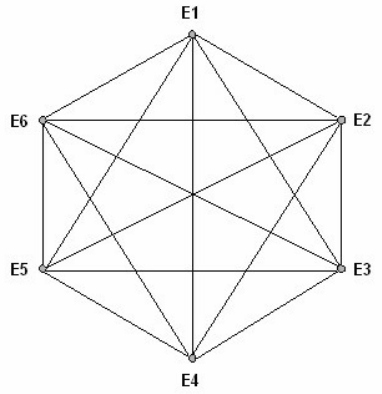


Fig 14.4: Mesh network (in 2- or 3D). Each processor has a direct link to four/six (in 2D/3D) neighbor processor. Extensions of this kind of networks are a warp around mesh or torus.

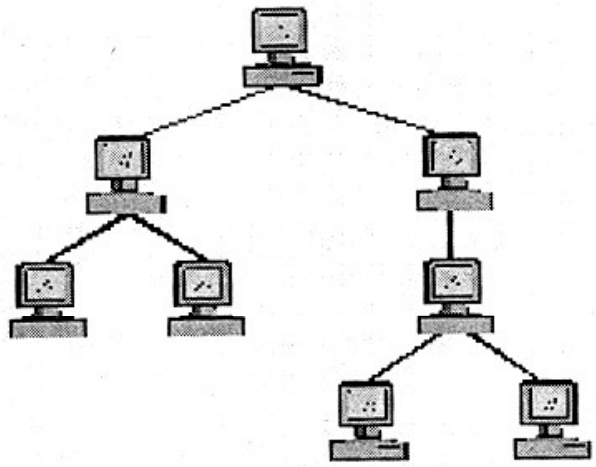


Fig 14.5 : Tree network of processors. Communication bottleneck likely to occour in large configurations can be alleviated by increasing the number of communication links for processors closer to the root, which results in the *fat-tree* topology, efficiently used in the TMC CM5 computer.

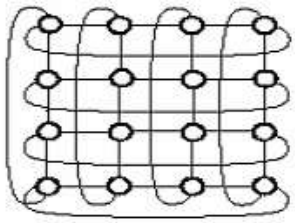


Fig 14.6: Hypercube network. Classically this is a multidimensional mesh of processors with exactly two processors in each dimension. An example of such a system is the Intel iPSC/860 computer

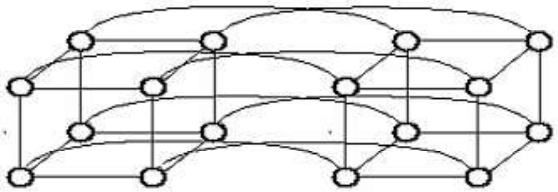


Fig 14.7: Processor system connected by a hypercube-type network.

Dynamics interconnection networks implement one of four main alternatives:

(a) *Bus-based networks* - the simplest and efficient solution when the cost and moderate number of processors are involved as shown in Table 1. Its main drawback is a bottleneck to the memory when number of processors becomes large and also a single point of failure. To overcome the problems, sometimes several parallel buses are incorporated . The classical example of such machine is the SGI Power Challenge computer with packet data bus.

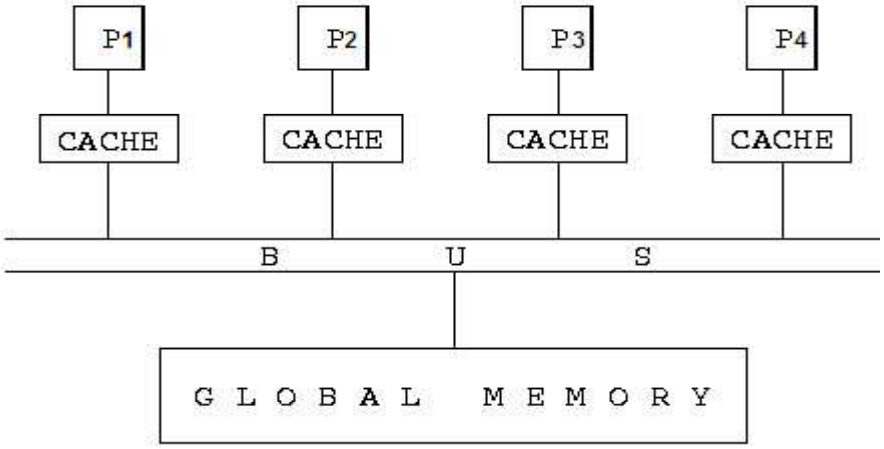


Fig 14.8 : Bus based network

Table 1 : Properties of various types of multiprocessor interconnections

Property	Bus	Crossbar	Multistage
Speed Cost	low low	high high	high moderate
Reliability Configurability	low high	high low	high moderate
Complexity	low	high	moderate

(b) ***Crossbar switching networks***, which employ a grid of switching elements. The network is nonblocking, since the connection of a processor to a memory bank does not block the connection of any other processor to any other memory bank. In spite of high speed, their use is limited, due to nonlinear complexity ($o(p^2)$, p - number of processors) and the cost (cf. Table 10). They are applied mostly in multiprocessor vector computers (like Cray YMP) and in multiprocessors with multilevel interconnections . One outstanding example is the Fujitsu VPP500 which incorporates 224x224 crossbar switch.

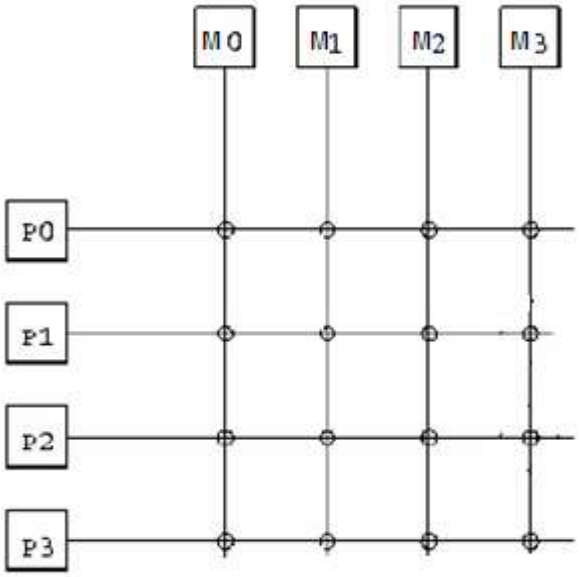


Fig 14.9: Crossbar switch

Multistage interconnection networks formulate the most advanced pure solution, which lies between the two extremes (Table 1). A typical example is the omega network, which consists of *log p* stages, where p is the number of inputs and outputs (usually number of processor and memory banks). Its

complexity is *O(p log p)*, less than for the crossbar switch. However, in the omega network some memory accesses can be blocked. Although machines of this kind of interconnections offer virtual global memory model of programming and ease of use they are still not much popular. Examples from the past cover BBN Butterfly.

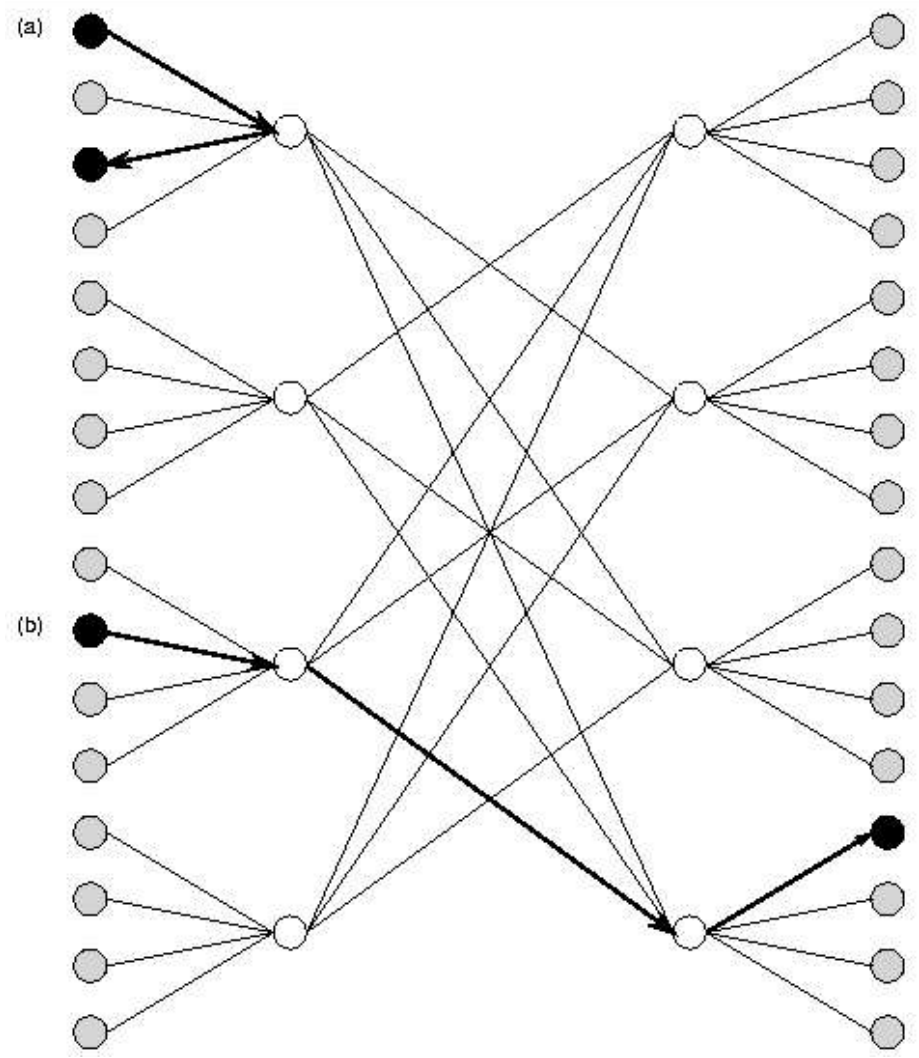


Fig 14.10: Communications in a bidirectional MIN. The communication indicated at (a) involves processors connected to the same crossbar; it takes just two hops and passes through a single switch. The communication at (b) takes three hops and passes through two switches.

Multilevel interconnection network seems to be a relatively recent development. The idea comes directly from clusters of computers and consists of two or more levels of connections with different aggregated bandwidths. Typical examples is Cray Origin2000.

14.4 TYPES OF MULTIPROCESSOR OPERATING SYSTEM

Multiprocessor operating systems are just regular operating systems. They handle system calls, do memory management, provide a file system, and manage I/O devices. Nevertheless, there are some areas in which they have unique features. These include process synchronization, resource management, and scheduling.

Let us now turn from multiprocessor hardware to multiprocessor software, in particular, multiprocessor operating systems. Various organizations are possible. Below we will study three of them.

14.4.1 CPU with its own Operating System

The simplest possible way to organize a multiprocessor operating system is to statistically divide memory into as many partitions as there are CPUs. Each CPU receives its own private memory and own private copy of the operating system. In there are *n* CPUs then operate as *n* independent computers. One obvious optimization is to allow all the CPUs to share the operating system code and make private copies of only the data, as shown in fig 14.11:

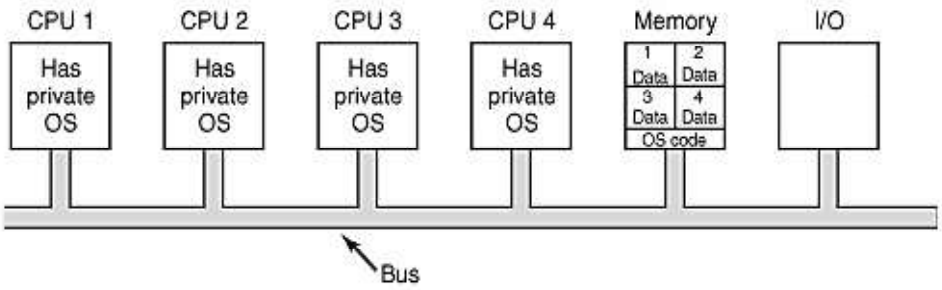


Fig 14.11: Partitioning multiprocessor memory among four CPUs, but sharing a single copy of the operating system code. The boxes marked Data are the operating system's private data for each CPU.

This scheme is still better than having *n* separate computers since it allows all the machines to share a set of resources such as disks and other I/O devices. It allows the memory to be shared flexibly. In addition, processes can efficiently communicate with one another.

It is worth explicitly mentioning *four* aspects of this design that may not be obvious. *First*, when a process makes a system call, the system call is

caught and handled on its own CPU using the data structures in that operating system's tables. *Second*, since each operating system has its own tables, it also has its own set of processes that it schedules by itself. There is no sharing of processes. If a user logs into CPU 1, all of his processes run on CPU 1. As a consequence, it can happen that CPU 1 is idle while CPU 2 is loaded with work. *Third*, there is no sharing of pages. It can happen that CPU 1 has pages to spare while CPU 2 is paging continuously. There is no way for CPU 2 to borrow some pages from CPU 1 since the memory allocation is fixed. *Fourth*, and worst, if the operating system maintains a buffer cache of recently used disk blocks, each operating system does this independently of the other ones. Thus it can happen that a certain disk block is present and dirty in multiple buffer caches at the same time, leading to inconsistent results. The only way to avoid this problem is to eliminate the buffer caches. Doing so is not hard, but it hurts performance considerably.

14.4.2 Master-Slave Multiprocessors

In master slave multiprocessor, one copy of the operating system and its tables are present on CPU 1. All system calls are redirected to CPU 1 for processing. CPU 1 may also run user processes if there is CPU time left over. This model is called **master-slave** since CPU 1 is the master and all the others are slaves as shown in Fig 14.12.

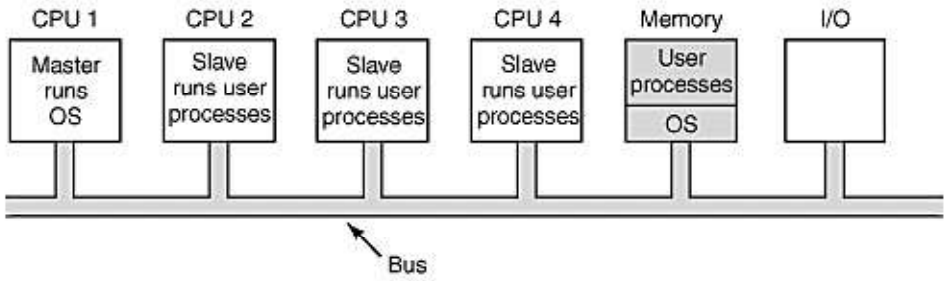


Fig 14.12 : A master-slave multiprocessor model.

There is a single data structure (e.g., one list or a set of prioritized lists) that keeps track of ready processes. When a CPU goes idle, it asks the operating system for a process to run. Thus it can never happen that one CPU is idle while another is overloaded. Similarly, pages can be allocated among all the processes dynamically and there is only one buffer cache, so inconsistencies never occur.

The problem with this model is that with many CPUs, the master will become a bottleneck. After all, it must handle all system calls from all CPUs. If, say, 10% of all time is spent handling system calls, then 10 CPUs will pretty much saturate the master, and with 20 CPUs it will be completely overloaded. Thus this model is simple and workable for small multiprocessors, but for large ones it fails.

14.4.3 Symmetric Multiprocessors

The third model is the **SMP** (**Symmetric MultiProcessor**), where one copy of the operating system in the memory, but any CPU can run it. When a system call is made, the CPU on which the system call was made traps to the kernel and processes the system call.

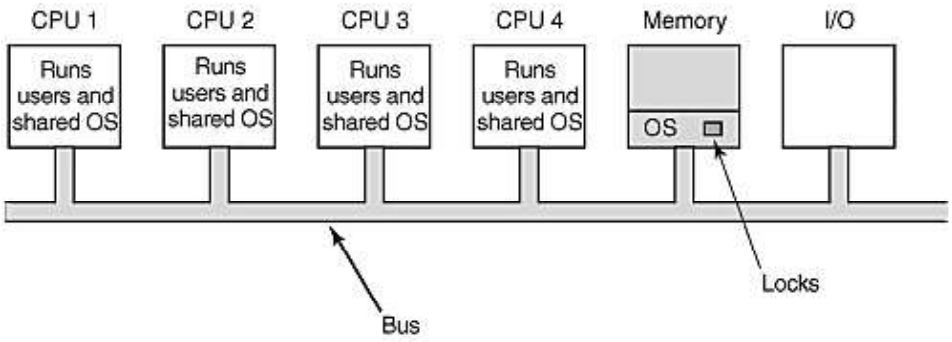


Fig 14.13: The SMP multiprocessor model.

This model balances processes and memory dynamically, since there is only one set of operating system tables. It also eliminates the master CPU bottleneck, since there is no master, but it introduces its own problems. In particular, if two or more CPUs are running operating system code at the same time, disaster will result. Imagine two CPUs simultaneously picking the same process to run or claiming the same free memory page. The simplest way around these problems is to associate a mutex (i.e., lock) with the operating system, making the whole system one big critical region. When a CPU wants to run operating system code, it must first acquire the mutex. If the mutex is locked, it just waits. In this way, any CPU can run the operating system, but only one at a time.

This model works, almost as badly as the master-slave model. Again, suppose that 10% of all run time is spent inside the operating system. With

20 CPUs, there will be long queues of CPUs waiting to get in. Fortunately, it is easy to improve. Many module of the operating system are independent of one another. For example, there is no problem with one CPU running the scheduler while another CPU is handling a file system call and a third one is processing a page fault.

This observation leads to splitting the operating system up into independent critical regions that do not interact with one another. Each critical region is protected by its own mutex, so only one CPU at a time can execute it. In this way, far more parallelism can be achieved. However, it may happen that some tables, such as the process table, are used by multiple critical regions. For example, the process table is needed for scheduling, but also for the fork system call and also for signal handling. Each table that may be used by multiple critical regions needs its own mutex. In this way, each critical region can be executed by only one CPU at a time and each critical table can be accessed by only one CPU at a time.

Furthermore, care must be taken to avoid deadlocks. If two critical regions both need table *A* and table *B*, and one of them claims *A* first and the other claims *B* first, sooner or later a deadlock will occur and nobody will know why. In theory, all the tables could be assigned integer values and all the critical regions could be required to acquire tables in increasing order. This strategy avoids deadlocks, but it requires the programmer to think very carefully which tables each critical region needs to make the requests in the right order.

14.4.4 Asymmetric Multiprocessors

Asymmetric multiprocessors and asymmetric multiprocessing (known as ASMP) were pioneered in 1970 by the Massachusetts Institute of Technology (MIT) and Digital Equipment Corporation. ASMP allows applications to run specific subtasks on processors separate from the “master” processor. ASMP computers are comprised of multiple physical processors that are unique, and thus not symmetrical. These processors are defined as either master or slave: master processors are more capable than slaves and are given full control over what the slave processors do. ASMP is not in use in the modern world as it was overtaken by symmetric multiprocessing (SMP),

which became common.

Although hardware-level ASMP may not be in use, the idea and logical process is still commonly used in applications that are multiprocessor intensive. Unlike SMP applications, which run their threads on multiple processors, ASMP applications will run on one processor but outsource smaller tasks to another. Although the system may physically be an SMP, the software is still able to use it as an ASMP by simply giving certain tasks to one processor and deeming it the “master”, and only outsourcing smaller tasks to “slave” processors.

Asymmetric hardware systems commonly dedicated individual processors to specific tasks. For example, one processor may be dedicated to disk operations, another to video operations, and the rest to standard processor tasks. These systems don’t have the flexibility to assign processes to the least-loaded CPU, unlike an SMP system.

Asymmetrical multiprocessors are defined by the characteristic that each processor is unique (non-symmetrical). It is common to have one processor that has access to the memory map as a whole, and other processors which simply act as slaves to the main or master processor. Usually, these slave processors will have their own memory which is not tied to the primary processors memory. Slave processors are required to exchange data with the main processor through a partitioned segment of memory that is allocated solely for the purpose of communication. Depending on the hardware in question, each processor may or may not be able to speak to other processors directly. For example, on a VAX (which contained four processors) slave processors could not talk to one another, and had to speak to the master processor instead.

Asymmetric multiprocessing (as opposed to asymmetrical multiprocessors) is the term that refers to software side ASMP. Just as one would refer to each unique die as a processor, in software each program or application is a process. ASMP for software means that all tasks/processes are unique (i.e non symmetrical). Thus a given task (such as your operating system or favorite game) would be assigned to a certain processor. In a more general context “a certain task not run on every processor”. It is common for application which use ASMP to work in the following way. The main processor will determine what work needs

to be done and will take the bulk of the load, from there it can create instances of the given task on other processors to complete work. Take a video rendering program, the main processor could run the application and the user interface, while offloading the rendering component to a slave processor. This type of action needs to be written into the software and is not decided at the hardware level. It is the programmers responsibility to determine what jobs should be completed by a given processor.

It must be noted that most applications will ONLY run on the master processor and that the slave processors can merely take on the role of completing tasks that the master processor asks. It is rare that an entire application will or can be run from a slave processor.

Differences between software ASMP & SMP

Symmetrical multiprocessing when referring to software, implies the exact opposite of ASMP. In regards to the operating system, SMP is able to spawn any process/task on any of the processors available. Because SMP systems have no master or slave processors, each logical unit is able to complete a given task. In an ASMP system, a certain processor may not be able to complete a task due to an inability to access the entire memory map, it must then be run from the master processor and given tasks by the master processor to complete. Once again it is up to the programmer to make sure the processors are being used to their maximum potential. It is a common conception that SMP programming is much simpler as any processor in the system can complete a given task, thus a programmer just needs to simply balance the workload between processors. in an ASMP environment, a programmer has to worry about whether a processor can complete a given task and how to make the processors communicate effectively to distribute tasks.

14.5 MULTIPROCESSOR OS FUNCTIONS AND REQUIREMENTS

Like the single processor computer, a multiprocessor operating system manges the available resources and augments the hardware functionality to form an abstraction that facilitates program execution and interaction

with users.Three basic types of resources that needs to be managed are:

1. Processor
2. Memory
3. I/O Devices

Processor scheduling is necessary for effective use of multiprocessors.The primary task of multiprocessor scheduler are to:

1 . Allocate processors to applications in a manner consistent with system design objectives.

2. Ensure effective use of processors allocated to an application.

This primarily affects the system throughput. Due to difficulties of automating the process, exploitation of parallelism within an application usually requires some degree of explicit programmer direction. Most of the support for explicit and automatic parallelism is provided by language translators and preprocessors. Since the operating system is unaware of the semantics of user computations, it usually cannot offer much direct help. However, the operating system should provide the mechanisms and facilities of foster parallelism and aid the efforts of compilers and programmers alike. The two primary facets of OS support for multiprocessing are (1) flexible and efficient interprocess and interprocessor synchronization mechanisms and (2) efficient creation and management of a large number of threads of activity, such as processes or threads. The later aspect is important because parallelism is often accomplished by splitting an application into separate , individually executable subtasks that may be allocated different processors. The following are some features of multiprocessor OS.

Allocation of Processing Resources

Processor allocation can be a significant problem in massively parallel systems. One way of keeping track of a large number of processes is to organize them in a logical hierarchy. The proposed hierarchy mimics similar to world organization, such as large companies and the military. Basically some of the processors are workers and others are managers.

The resulting system ,called wave scheduling, basically operates as follows. In the MICROS system, which pioneered this approach individual processors are dedicated to one task at a time. Each manager keeps tracks of the

number of worker that is available.Upon receiving a work request calling for R processors, the system must secure r or more processors, since some can fail.The work request can be submitted to a manager at any level of hierarchy. If the manager chosen does not have the sufficient number of workers, it calls on its boss for help. If the boss does not have enough help available, it can call on its boss. The process continues until a sufficient number of available processors is secured or the top of the hierarchy is reached. If the attempt to allocate processors fails at that level, the request may be delayed pending availability of an adequate amount of resources.

Support for multiprocessing

The operating system can support multiprocessing and parallelism by providing a mechanism for creation and management of large number of processes or threads. The process model can be viewed as providing a virtual processor abstraction. Each process has a state, a set of allocated resources such as files and memory, and connection to I/O devices. An application consisting of several cooperating processes represents a virtual multiprocessor that consists of virtual processors and resources that they share. The virtual multiprocessor abstraction is at a higher level than the hardware, because it includes the services of the operating system, such as virtual memory and file system. In a multiprocessor environment, true multiprocessing is possible by allocating a separate physical processor to each virtual processor (process). Access to shared resources is usually through the shared memory or by interprocess communication.

When threads are used, each application is implemented as a separate process , and its concurrent portions are coded as separate threads within the enclosing process. Threads belonging to a single process share the memory and all other resources acquired by the process. Given their close relationship and common goal of executing an application, threads belonging to a process usually have much closer interaction then thread that belong to different processes. Some OS provide specialized calls for efficient synchronization of threads that belong to same process. Communication is normally not an issue since threads share common memory. By having information about processes and threads, the scheduler can attempt to coschedule the related threads and thus reduce the probability of slowdowns caused by out of phase scheduling of closely coupled parties.

Memory Management in multiprocessor is highly dependent on the underlying architecture and interconnection scheme. Memory in loosely coupled system is handled independently, on a pre processor basis. In shared memory systems, the operating system should provide a flexible memory model that facilitates safe and efficient access to shared data structures and synchronization variables. Ideally a multiprocessor OS should provide hardware independent, unified model of shared memory to facilitate porting of applications between different multiprocessor environments. Shared memory may be simulated by means of message passing mechanism in loosely coupled systems. Conversely message passing mechanism can be implemented in tightly coupled systems by relying on physically shared memory. Such a unified model consisting of messages and shared memory provides a flexible model and a set of tools for application programmers. The duality of memory management and interprocess communication was exploited by designers of Macintosh operating system.

Device Management has received a little attention in a multiprocessor system to date. This is partly due to early focus on speedup in computer intensive applications that tend to execute for extended periods of time and do not generate much I/O after the initial loading. However as multiprocessors are applied to more balanced general-purpose applications, one can expect their I/O requirements to increase in portion with realized throughput and speedup. Work on disk arrays address some of these problems, but many other new and innovative techniques will be needed to obtain significantly increased performance using I/O devices whose individual speeds remain essentially unchanged. It is reasonable to expect that the application of multiprocesing techniques to I/O, such as stringing together a multitude of I/O devices and operating them in parallel, should yield good results.

14.6 MULTIPROCESSOR SYNCHRONIZATION

Designers of computer systems consistently strive for increased processing capacity with each product generation. Many different approaches have been adopted to achieve the computing speeds currently enjoyed by users of such systems. Increased system clock speeds, integrated circuit design advances, wider data paths, and various other technological developments

have all contributed to increasing the processing throughput of a single processor.

To further enhance computer capability, pipelined and parallel arrangements of multiple processing units have been pursued successfully. Parallel processing generally began with the use of “single instruction stream, multiple data stream” (SIMD) architectures, in which multiple processors perform identical operations on different data. In such a system, a single program line, or “thread,” of instructions is executed. More advanced “multiple instruction stream, multiple data stream” (MIMD) systems allow each processor to execute a completely diverse set of instructions, or a separate copy of the same set of instructions.

However, even in an MIMD system, some communication or cooperation between the various processors is typically required. One type of communication involves synchronizing two or more of the processors by requiring each of the processors to halt execution at a predetermined point in its execution thread , and then begin execution again at the same or another predetermined location . One or more such synchronization points may be employed depending on the general nature and specific requirements of the overall task to be performed.

Typically, a multiprocessing computer system supports the use of synchronization points by way of complicated, specialized and nonstandard hardware functions. For example, a dedicated hardware implementation of a specialized broadcast interrupt may be supplied to support the use of launch points so that a single processor may inform all other processors of a launch quickly and efficiently. However, other multiprocessor systems may not provide such hardware due to the design and implementation expense involved in supporting such a specialized hardware construct.



CHECK YOUR PROGRESS

Q1. State true of false :

a) Multiprocessor system is known as tightly coupled system.

b) Queking technique does not influences the bandwith of the network.

c) Dynamic networks are mainly used in message-passing technique.

d) In a tree network of processors, each processor has a direct link to 4/6 neighbour processors.

e) In a master-slave mulitiprocess system any one of the CPU has operating system and table.



14.7 LET US SUM UP

- Interconnection networks make a major factor to differentiate modern multiprocessor architectures.
- Switching technique as a criterion one can mention some classes:
  - *circuit switching*, in which the entire path through the network is reserved before a message is transferred,
  - *packet switching* with *virtual cut through*, in which a packet is forwarded immediately after it determines an appropriate switch output,
  - *wormhole routing*, which relaxes requirements of completely buffering of blocked packets in a single switch, typical for *packet switching*.
- Static Networks are :
  - completely-connected network.
  - star-connected network.
  - linear array or ring of processors.
  - mesh network
  - tree network of processors.
  - hypercube network.
- Dynamic Networks are :
  - bus-based networks
  - crossbar switching networks
  - multistage interconnection networks
  - multilevel interconnection network

- Asymmetric Multiprocessors-ASMP computers are comprised of multiple physical processors that are unique, and thus not symmetrical. These processors are defined as either master or slave
- Symmetric multiprocessor is able to spawn any process/task on any of the processors available. Because SMP systems have no master or slave processors, each logical unit is able to complete a given task
- Processor Scheduling-Allocate processor to applications in an efficient way
- Memory Management-In shared memory system OS should provide flexible memory model, that will provide access to shared data structure and synchronization
- I/O device management- Focuses on speedup for input output related applications
- Multiprocessor Synchronization is the synchronizing of two or more processors by requiring each of the processors to halt execution at a predetermined point in its execution thread, and then begin execution again at the same or another predetermined location.This also involves process synchronization.



14.8 FURTHER READING

1. Silberschatz, A., Galvin, P. B., Gagne, G., & Silberschatz, A. (1998). Operating system concepts (Vol. 4). Reading: Addison-wesley.

2. Tanenbaum, A. S., & Woodhull, A. S. (1987). Operating systems: design and implementation (Vol. 2). Englewood Cliffs, NJ: Prentice Hall.



14.9 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1. (a) T

(b) F

(c) F

(d) F

(e) T



14.10 MODEL QUESTIONS

Q1. What interconnection networks ? Explain using an example.

Q2. What are static networks ? Differentiate between completely connected and mesh network.

Q3. Briefly explain the various types of dynamic networks also explain how it is differs from static network ?

Q4. Explain master slave model ?

Q5 Differentiate between symmetric and asymmetric multiprocessors.

Q6. Explain processor scheduling in case of a multiprocessor system.

Q7. Discuss multiprocessor synchronization.

******