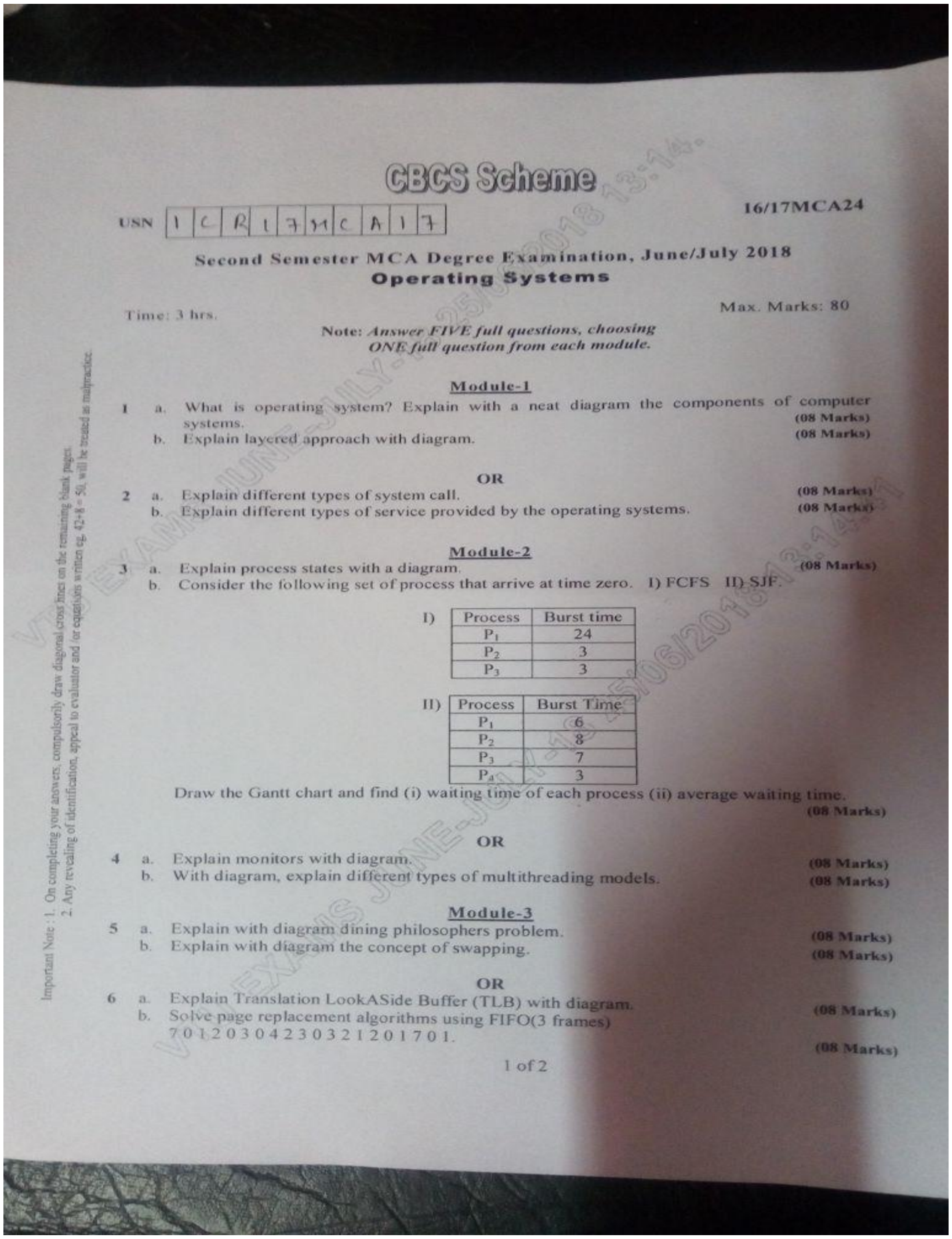


CBCS Scheme

USN 1 C R 1 7 M C A 1 7

16/17MCA24

Second Semester MCA Degree Examination, June/July 2018
Operating Systems

Time: 3 hrs. Max. Marks: 80

Note: *Answer FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. What is operating system? Explain with a neat diagram the components of computer systems. **(08 Marks)**

b. Explain layered approach with diagram. **(08 Marks)**

OR

2 a. Explain different types of system call. **(08 Marks)**

b. Explain different types of service provided by the operating systems. **(08 Marks)**

Module-2

3 a. Explain process states with a diagram. **(08 Marks)**

b. Consider the following set of process that arrive at time zero. I) FCFS II) SJF.

I)

Process	Burst time
P_1	24
P_2	3
P_3	3

II)

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Draw the Gantt chart and find (i) waiting time of each process (ii) average waiting time. **(08 Marks)**

OR

4 a. Explain monitors with diagram. **(08 Marks)**

b. With diagram, explain different types of multithreading models. **(08 Marks)**

Module-3

5 a. Explain with diagram dining philosophers problem. **(08 Marks)**

b. Explain with diagram the concept of swapping. **(08 Marks)**

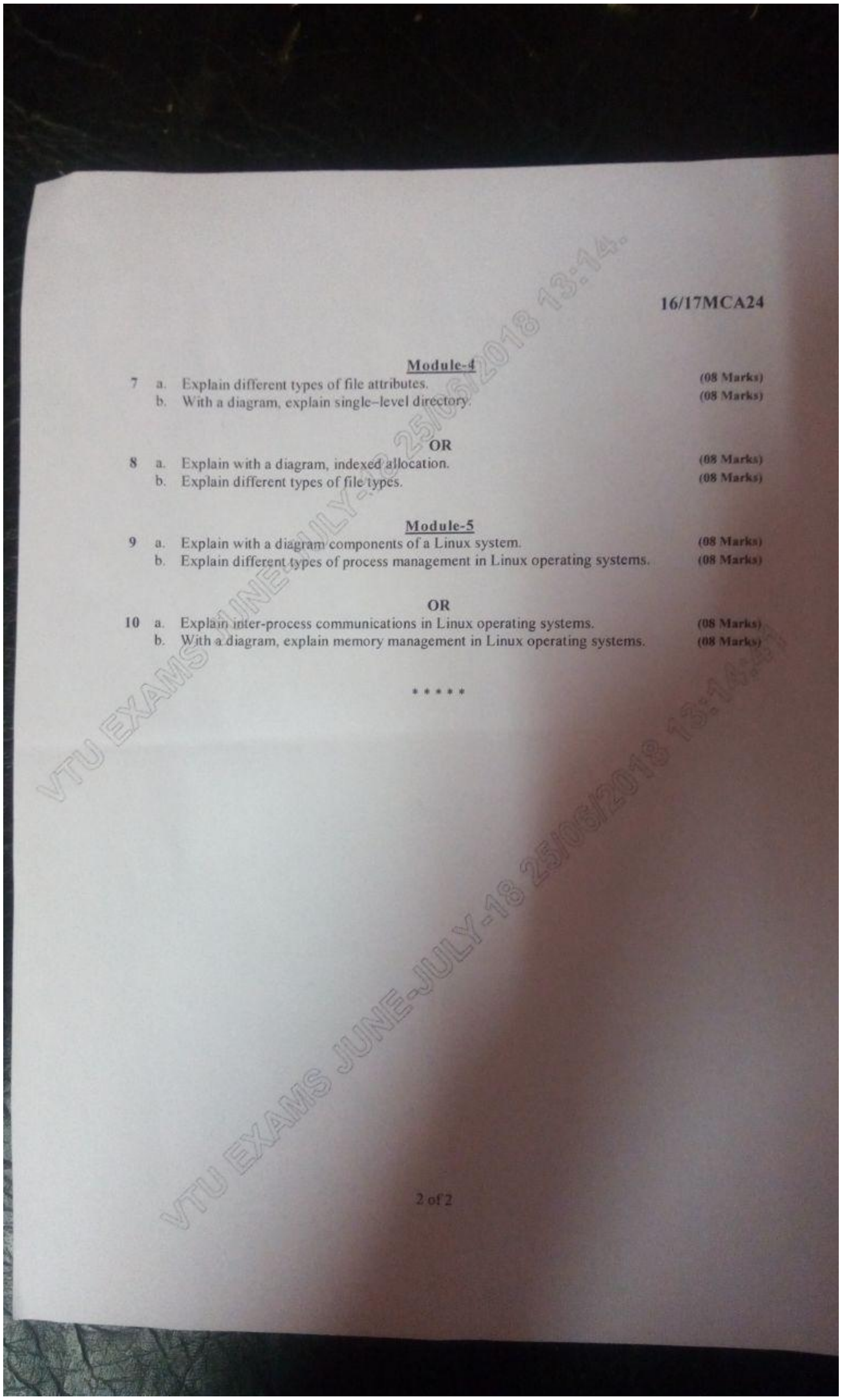
OR

6 a. Explain Translation LookASide Buffer (TLB) with diagram. **(08 Marks)**

b. Solve page replacement algorithms using FIFO(3 frames) 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1. **(08 Marks)**

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

1 of 2



Module-4

7 a. Explain different types of file attributes. (08 Marks)
b. With a diagram, explain single–level directory. (08 Marks)

OR

8 a. Explain with a diagram, indexed allocation. (08 Marks)
b. Explain different types of file types. (08 Marks)

Module-5

9 a. Explain with a diagram components of a Linux system. (08 Marks)
b. Explain different types of process management in Linux operating systems. (08 Marks)

OR

10 a. Explain inter-process communications in Linux operating systems. (08 Marks)
b. With a diagram, explain memory management in Linux operating systems. (08 Marks)

* * * * *

1.a) What is operating system? Explain with a neat diagram the components of computer system

- An operating system or OS is a system software program that enables the computer hardware to communicate and operate with the computer software. Without a computer operating system, a computer and software programs would be useless.
- When computers were first introduced, the user interacted with them using a command line interface, which required commands. Today, almost every computer is using a Graphical User Interface (GUI) operating system that is much easier to use and operate.
- Examples of computer operating systems - Microsoft Windows 10, Linux

BASIC ELEMENTS

A computer consists of processor, memory and I/O components with one or more modules of each type. These components are interconnected to ease job of computer. Thus, there are four structural elements:

- **Processor:** It controls the operation of the computer and performs its data processing functions. When there is only one processor, it is called as Central Processing Unit (CPU).
- **Main Memory:** It stores data and programs. But, the contents of memory are lost when the computer shuts down. Whereas, the contents of disk memory are retained.
- **I/O Modules:** Move data between the computer and its external environment. The external environment consists of a variety of devices including secondary memory devices, communications equipment and terminals.
- **System Bus:** This provides communication among processors, main memory and i/o modules.

The top-level components of computer are shown in Figure 1.1. One of the functionality of a processor is to exchange data with memory. For this purpose, the following registers are used:

- **Memory Address Register (MAR):** specifies the address in memory for the next read or write.
- **Memory Buffer Register (MBR):** contains data to be written into memory or it receives the data read from memory.
- **I/O Address Register (I/OAR):** indicates particular I/O device
- **I/O Buffer Register:** used to exchange data between an I/O module and the processor.

A memory module contains a set of locations – which are sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as an instruction or data. An I/O module transfers data from external devices to processor and memory and vice versa. It contains temporary buffers for holding data till they are sent.

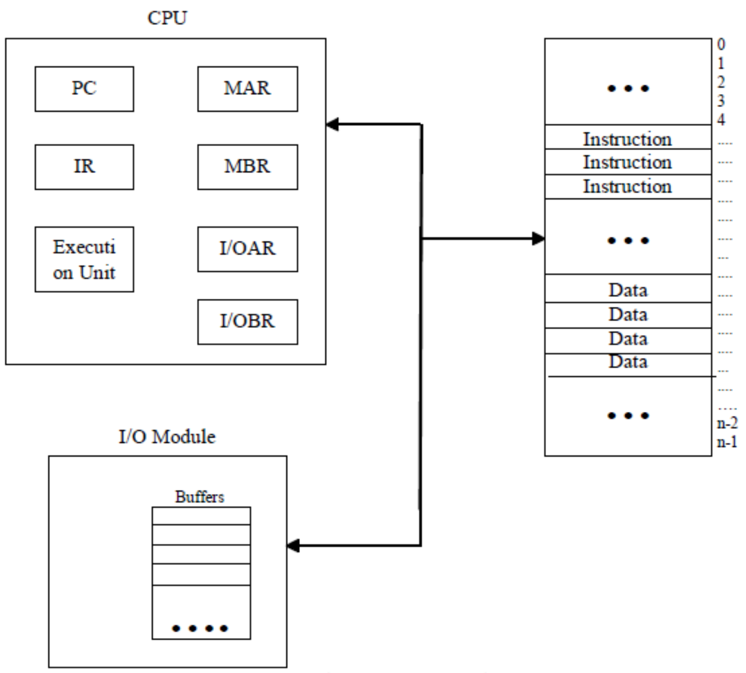
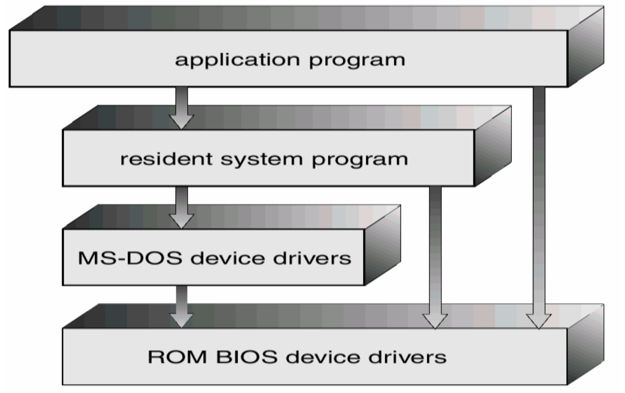
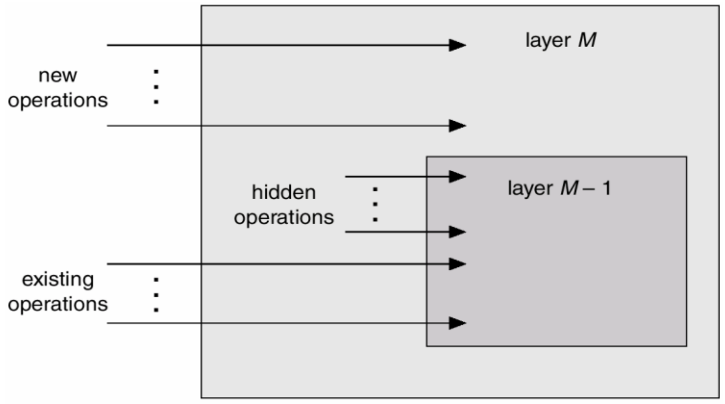


Figure 1.1 Computer Components: Top Level View

b) Explain layered approach with diagram





MS DOS layer structure with a neat diagram. A layered approach of OS

In the layered approach, the OS is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.

The main advantage of layered approach is modularity. The layers are selected such that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification.

The major difficulty with the layered approach involves the careful definition of the layers, because a layer can use only those layers below it.

2.a) Explain the different types of system calls

System calls can be grouped roughly into five major categories: process control, file management, device management, information maintenance, and communications. These are discussed in the following sections. Table 2.1 summarizes the types of system calls provided by OS.
Explain the different types of system calls

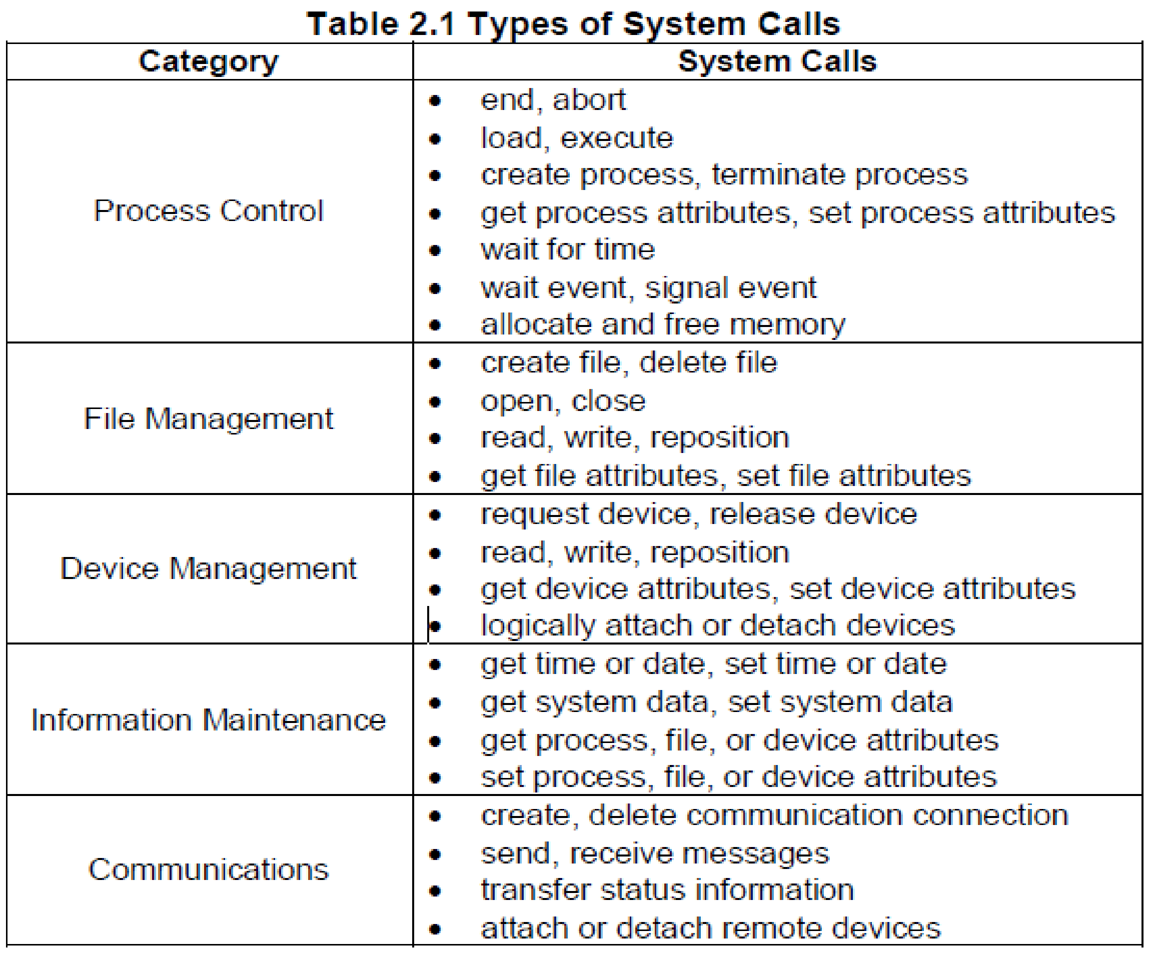


Table 2.1 Types of System Calls

Category	System Calls
Process Control	• end, abort • load, execute • create process, terminate process • get process attributes, set process attributes • wait for time • wait event, signal event • allocate and free memory
File Management	• create file, delete file • open, close • read, write, reposition • get file attributes, set file attributes
Device Management	• request device, release device • read, write, reposition • get device attributes, set device attributes • logically attach or detach devices
Information Maintenance	• get time or date, set time or date • get system data, set system data • get process, file, or device attributes • set process, file, or device attributes
Communications	• create, delete communication connection • send, receive messages • transfer status information • attach or detach remote devices

2.b) Explain different types of service provided by the OS

OS provides an environment for the execution of programs. Also, it provides certain services to the programs and its users. Though these services differ from one OS to the other, following are some general services provided by any OS.

□ Program execution: The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

□ I/O operations: A running program may require I/O. This I/O may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the OS must provide a means to do I/O.

□ File-system manipulation: The OS must facilitate the programs to read and write the files. And also, programs must be allowed to create and delete files by name.

 Communications: Processes may need to exchange information with each other. These processes may be running on same computer or on different computers. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of information are moved between processes by the OS.
 Error detection: The OS constantly needs to be aware of possible errors. Errors may occur in any of CPU, memory hardware, I/O devices, user program etc. For each type of error, the OS should take the appropriate action to ensure correct and consistent computing. OS also has another set of functionalities to help the proper functioning of itself.
 Resource allocation: When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. OS has to manage many resources like CPU cycles, main memory, and file storage etc. OS uses CPU scheduling routines for effective usage of CPU. These routines manage speed of the CPU, the jobs that must be executed, the number of registers available, and such other factors.
 Accounting: We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.
 Protection:Information on a multi-user computer system must be secured. When multiple processes are executing at a time, one process should not interfere with the others. Protection involves ensuring that all access to system resources is controlled. System must be protected from outsiders as well. This may be achieved by authenticating the users by means of a password. It also involves defending external I/O devices, modems, network adapters etc. from invalid access

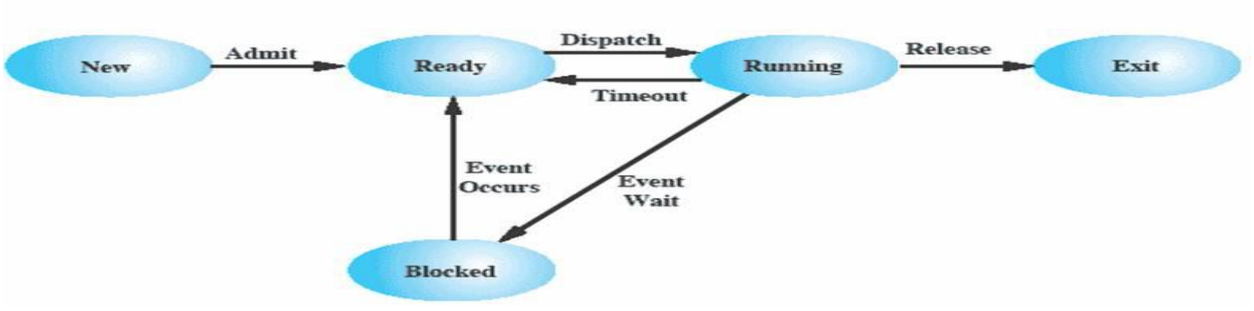
3.a) Explain process states with a diagram

PROCESS

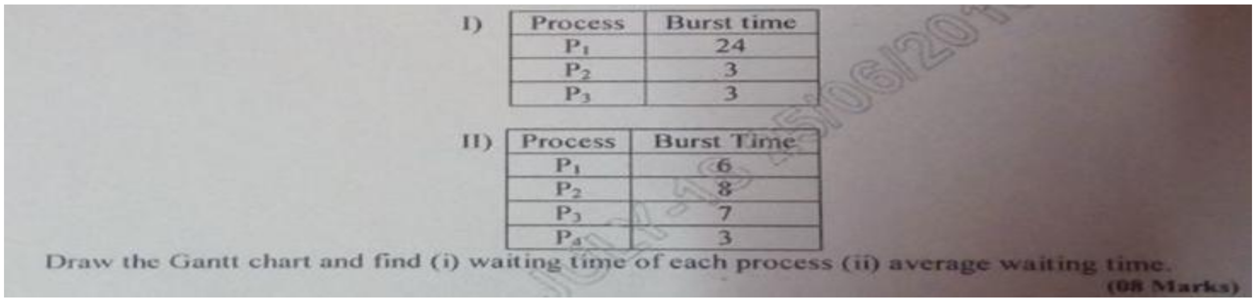
A process can be defined in several ways:
 A program in execution
 An instance of a program running on a computer
 The entity that can be assigned to and executed on a processor
 A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources.
Two essential elements of a process are:
 program code: which may be shared with other processes that are executing the same program
 Set of data: associated with that code

The various States of the Process are as Followings:-
1) **New State:** When a user request for a Service from the System , then the System will first initialize the process or the System will call it an initial Process . So Every new Operation which is Requested to the System is known as the New Born Process.

2) **Running State:** When the Process is Running under the CPU, or When the Program is Executed by the CPU , then this is called as the Running process and when a process is Running then this will also provides us Some Outputs on the Screen.
3) **Waiting:** When a Process is Waiting for Some Input and Output Operations then this is called as the Waiting State. And in this process is not under the Execution instead the Process is Stored out of Memory and when the user will provide the input then this will Again be on ready State.
4) **Ready State:** When the Process is Ready to Execute but he is waiting for the CPU to Execute then this is called as the Ready State. After the Completion of the Input and outputs the Process will be on Ready State means the Process will Wait for the Processor to Execute.
5) Terminated State: After the Completion of the Process , the Process will be Automatically terminated by the CPU . So this is also called as the Terminated State of the Process. After executing the whole Process the Processor will also de-allocate the Memory which is allocated to the Process. So this is called as the Terminated Process.



b) Consider the following set of process that arrive at time zero 1) FCFS 2)SJF



I)

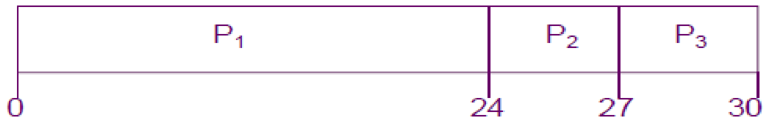
Process	Burst time
P_1	24
P_2	3
P_3	3

II)

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

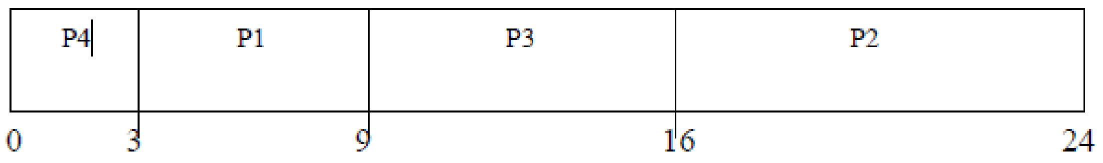
Draw the Gantt chart and find (i) waiting time of each process (ii) average waiting time.
(08 Marks)

Solution: Suppose, the processes arrive in the order P_1, P_2 , P_3, then the Gantt Chart for the schedule is –



We can observe that,
Waiting time for P_1 = 0
Waiting time for P_2 = 24
Waiting time for P_3 = 27
Thus, **Average waiting time = (0 + 24 + 27)/3 = 17 milliseconds**

ii)



Waiting time for P1 = 3
Waiting time for P2 = 16
Waiting time for P3 = 9
Waiting time for P4 = 0
Average waiting time = (3+16+9+0)/4 = 7 milliseconds

4.a) Explain monitors with diagram

A monitor is a software module consisting of one or more procedures, an initialization sequence, and local data.The chief characteristics of a monitor are the following:
1. The local data variables are accessible only by the monitor's procedures and not by any external procedure.
2. A process enters the monitor by invoking one of its procedures.
3. Only one process may be executing in the monitor at a time; any other processes that have invoked the monitor are blocked, waiting for the monitor to become available.
The first two characteristics are reminiscent of those for objects in object-oriented software. Indeed, an object-oriented OS or programming language can readily implement a monitor as an object with special characteristics.
By enforcing the discipline of one process at a time, the monitor is able to provide a mutual exclusion facility. The data variables in the monitor can be accessed by only one process at a time. Thus, a shared data structure can be protected by placing it in a monitor. If the data in a monitor represent some resource, then the monitor provides a mutual exclusion facility for accessing the resource. To be useful for concurrent processing, the monitor must include synchronization tools.

For example, suppose a process invokes the monitor and, while in the monitor, must be blocked until some condition is satisfied. A facility is needed by which the process is not only blocked but releases the monitor so that some other
process may enter it. Later, when the condition is satisfied and the monitor is again available, the process needs to be resumed and allowed to reenter the monitor at the point of its suspension.

A monitor supports synchronization by the use of **condition variables** that are contained within the monitor and accessible only within the monitor. Condition variables are a special data type in monitors, which are operated on by two functions:
• cwait(c): Suspend execution of the calling process on condition *c*. The monitor is now available for use by another process.

• csignal(c): Resume execution of some process blocked after a cwait on the same condition. If there are several such processes, choose one of them; if there is no such process, do nothing.

Note that monitor *wait* and *signal* operations are different from those for the semaphore. If a process in a monitor signals and no task is waiting on the condition variable, the signal is lost.

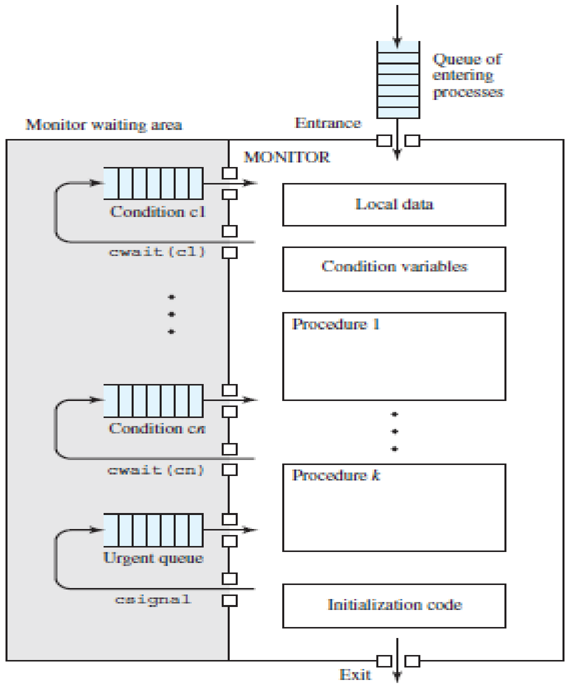


Figure 5.15 **Structure of a Monitor**

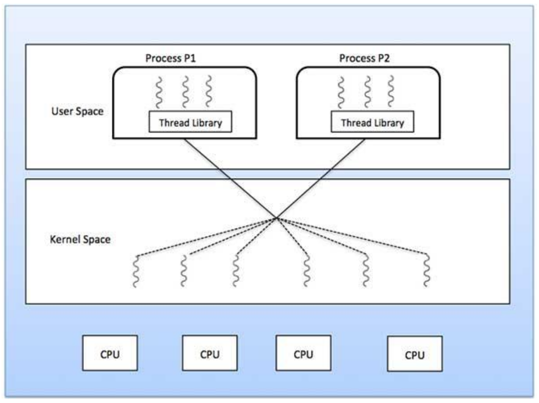
b) With diagram, explain different types of multithreading models

Multithreading models are three types

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

Many to Many Model

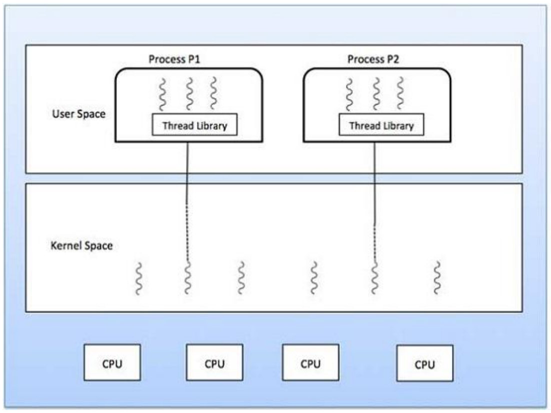
The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads. The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model, developers can create as many user threads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



Many to One Model

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

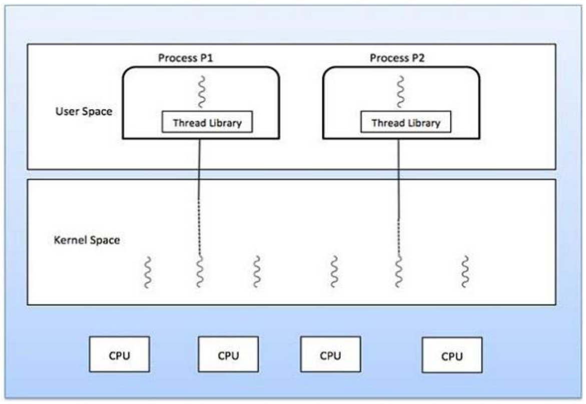
If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the Kernel threads use the many-to-one relationship modes.



One to One Model

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

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



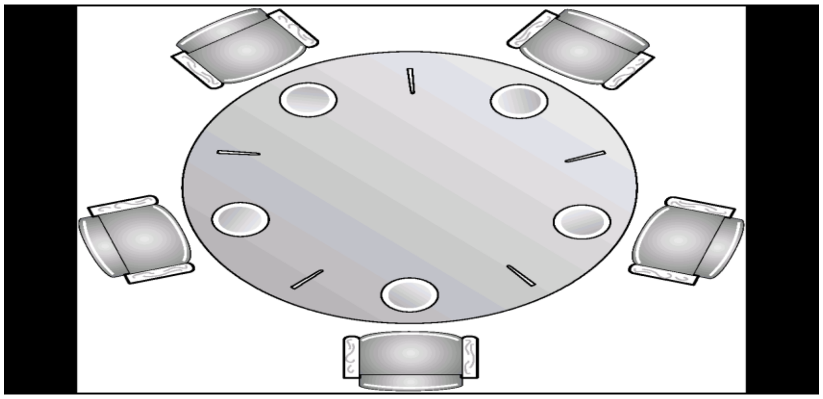
5.a) Explain with diagram dining philosophers problem

Five philosophers spend their lives thinking and eating.

 Philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher.

 In center of the table is a bowl of rice (or spaghetti), and the table is laid with five single chopsticks.

 From time to time, philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors).



A philosopher may pick up only one chopstick at a time.

 She cannot pick up a chopstick that is already in hand of a neighbor.

 When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks.

 When she finishes eating, she puts down both of her chopsticks and start thinking again.

The problem is to ensure that no philosopher will be allowed to starve because he cannot ever pick up both forks.

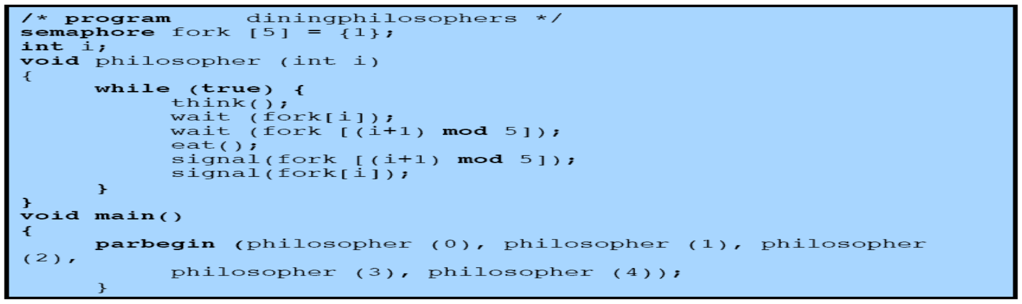
The dinning philosopher problem is considered a classic problem because it is an example of a large class of concurrency-control problems.

 Shared data

 semaphore chopstick[5];

 Initially all values are 1

 A philosopher tries to grab the chopstick by executing wait operation and releases the chopstick by executing signal operation on the appropriate semaphores.



```
/* program      diningphilosophers */
semaphore fork [5] = {1};
int i;
void philosopher (int i)
{
     while (true) {
          think();
          wait (fork[i]);
          wait (fork [(i+1) mod 5]);
          eat();
          signal(fork [(i+1) mod 5]);
          signal(fork[i]);
     }
}
void main()
{
     parbegin (philosopher (0), philosopher (1), philosopher
(2),
          philosopher (3), philosopher (4));
     }
```

Figure 6.12 A First Solution to the Dining Philosophers Problem

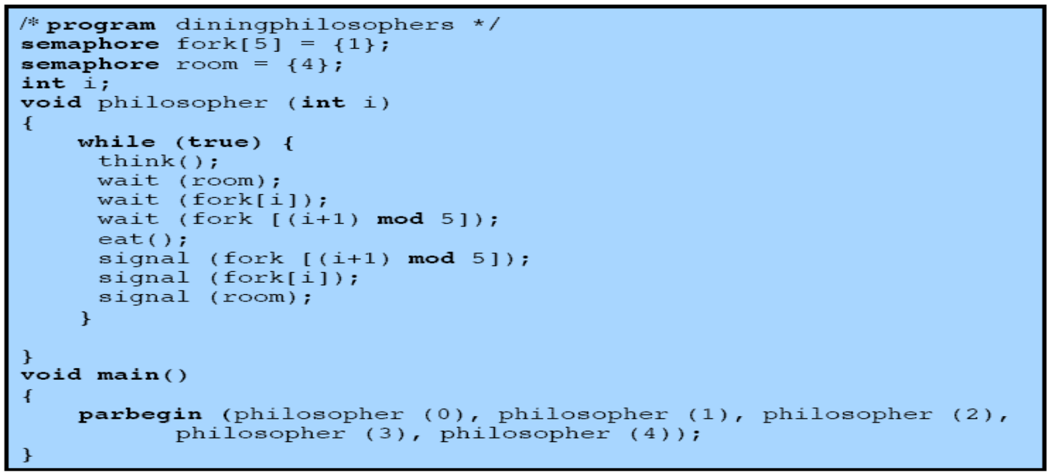
This solution guarantees that no two neighbors are eating simultaneously but it has a possibility of creating a deadlock and starvation.

 Allow at most four philosophers to be sitting simultaneously at the table.

 Allow a philosopher to pick up her chopsticks if both chopsticks are available.

 An odd philosopher picks up her left chopstick first and an even philosopher picks up her right chopstick first.

 Finally no philosopher should starve.



```
/* program diningphilosophers */
semaphore fork[5] = {1};
semaphore room = {4};
int i;
void philosopher (int i)
{
     while (true) {
      think();
      wait (room);
      wait (fork[i]);
      wait (fork [(i+1) mod 5]);
      eat();
      signal (fork [(i+1) mod 5]);
      signal (fork[i]);
      signal (room);
     }

}
void main()
{
     parbegin (philosopher (0), philosopher (1), philosopher (2),
           philosopher (3), philosopher (4));
}
```

Figure 6.13 A Second Solution to the Dining Philosophers Problem

b) Explain with diagram with concept of swapping

A process needs to 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 as shown in Figure 6.3. Such swapping may be necessary in priority based scheduling or round-robin scheduling.

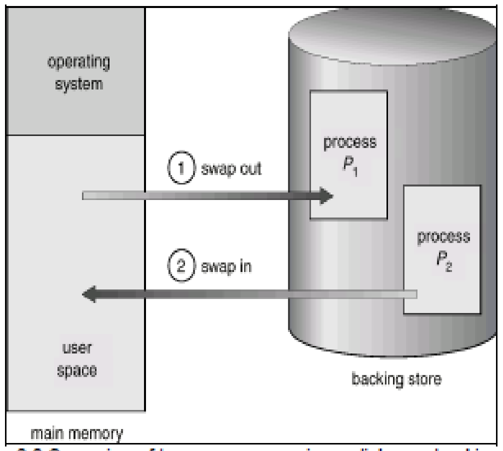


Figure 6.3 Swapping of two processes using a disk as a backing store

Figure 6.3 Swapping of two processes using a disk as a backing store Normally a process that is swapped out will be swapped back into the same memory space that it occupied previously. This restriction is dictated by the method of address binding. If binding is done at assembly or load time, then the process cannot be moved to different locations. If execution-time binding is being used, then a process can be swapped into a different memory space, because the physical addresses are computed during execution time.

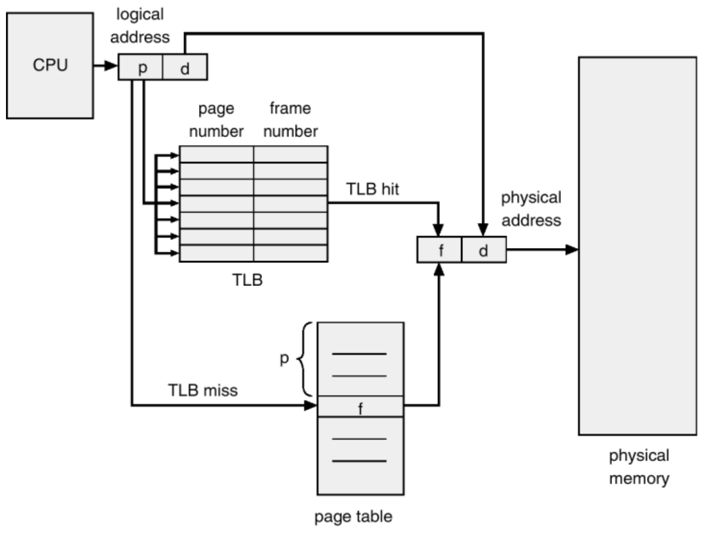
Swapping requires a backing store. The backing store is a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images. The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run. Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.

The dispatcher checks to see whether the next process in the queue is in memory. If not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control to the selected process. The context-switch time in such a swapping system is high. If we want to swap a process, it must be idle.

6.a) Explain Translation LookAside(TLB) Buffer with diagram

Each operating system has its own methods for storing page tables. In the simplest case, the page table is implemented as a set of dedicated registers. Another method is: the page table is kept in main memory, and a ***page-table base register (PTBR)*** points to the page table. In this scheme every data/instruction access requires two memory accesses: One for the page table and one for the data/instruction. This double access problem can be solved by the use of a special fast-lookup hardware cache called ***associative memory*** or ***translation look-aside buffers (TLBs)***. The TLB is
associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. The TLB is used with page tables in the following way:

 The TLB contains only a few of the page-table entries.
 When a logical address is generated by the CPU, its page number is presented to the TLB.
 If the page number is found, its frame number is immediately available and is
 If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. This is shown in Figure 6.8.



6.b) Solve page replacement algorithms using FIFO(3 frames)

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Assuming 3 frames, find the number of page faults when the following algorithms are used: FIFO Note that initially all the frames are empty.

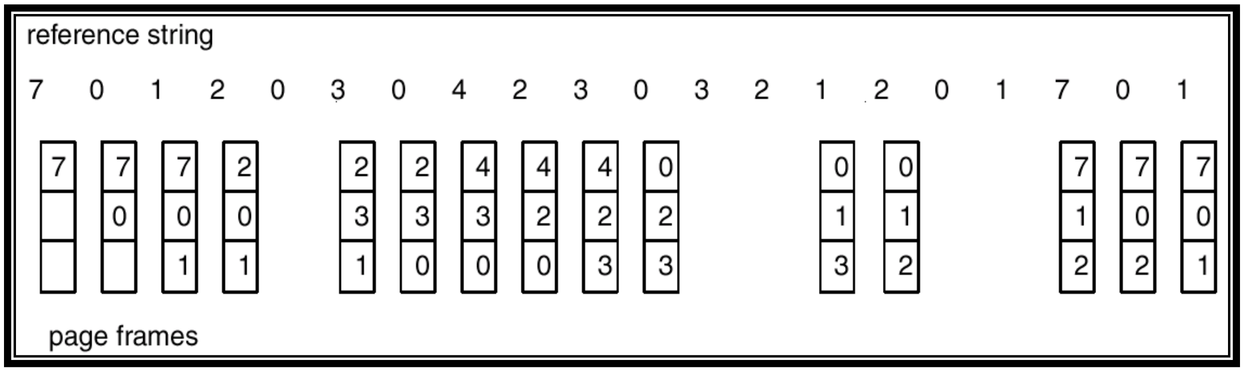
Assuming 3 frames, find the number of page faults when the following algorithms are used: i) LRU ii) FIFO iii) Optimal. Note that initially all the frames are empty.

FIFO Page Replacement

It is the simplest page – replacement algorithm. As the name suggests, the first page which has been brought into memory will be replaced first when there no space for new page to arrive. Initially, we assume that no page is brought into memory. Hence, there will be few (that is equal to number of frames) page faults, initially. Then, whenever there is a request for a page, it is checked inside the frames. If that page is not available, page – replacement should take place.

Example: Consider a reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Let the number of frames be 3.



In the above example, there are 15 page faults.

7.a) Explain different types of file attributes

A file has certain attributes, which vary from one OS to another, but typically consist of the following:

 Name: The symbolic file name is the only information kept in human readable form.

 Identifier: This unique number identifies the file within the file system; it is the nonhuman-readable name for the file.

 Type: This information is needed for those systems that support different types.

 Location: This information is a pointer to a device and to the location of the file on that device.

 Size: The current size of the file (in bytes, words, or blocks), and possibly the maximum allowed size are included in this attribute.

 Protection: Access-control information determines who can do reading, writing, executing, and so on.

 Time, date, and **user identification**: This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

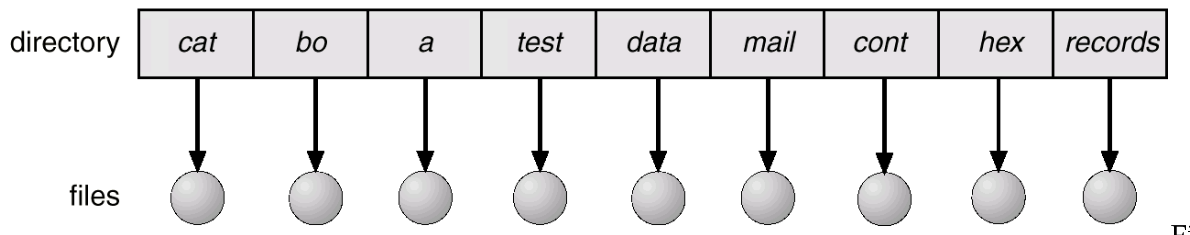
b) With a diagram, explain single level directory

The simplest directory structure is the single-level directory. All files are contained in the same directory, which is easy to support and understand. It is shown in Figure 7.4. There are certain limitations for single-level directory:

 As all files will be in one directory, each file should have a unique name.

 If there are more users on the same system, each one of them should remember their file names – which is difficult.

 It is not possible to group the related files together.



Figure

Single – level directory structure

8.a) Explain with a diagram indexed allocation

The direct-access nature of disks allows us flexibility in the implementation of files. In almost every case, many files will be stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. Three major methods of allocating disk space are: contiguous, linked, and indexed.

This method allows direct access of files and hence solves the problem faced in linked allocation. Each file has its own index block, which is an array of disk-block addresses. The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block as shown in Figure 7.13.

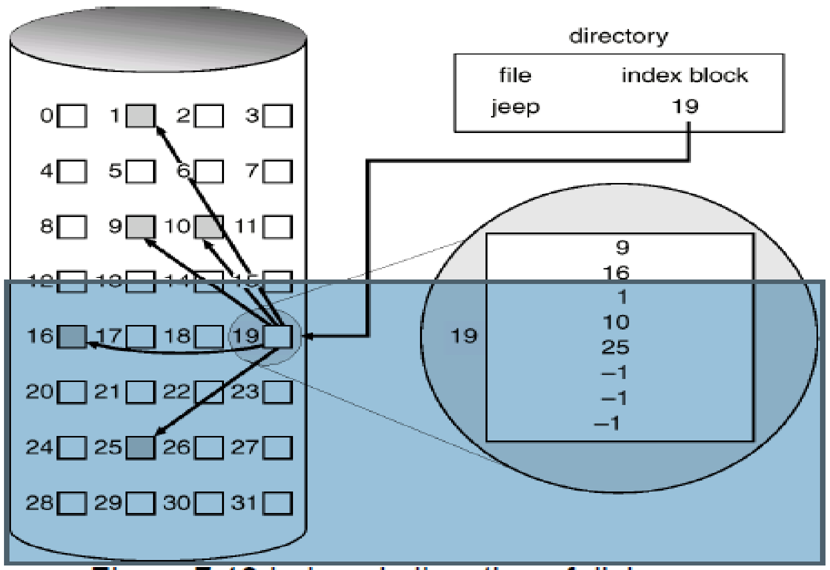
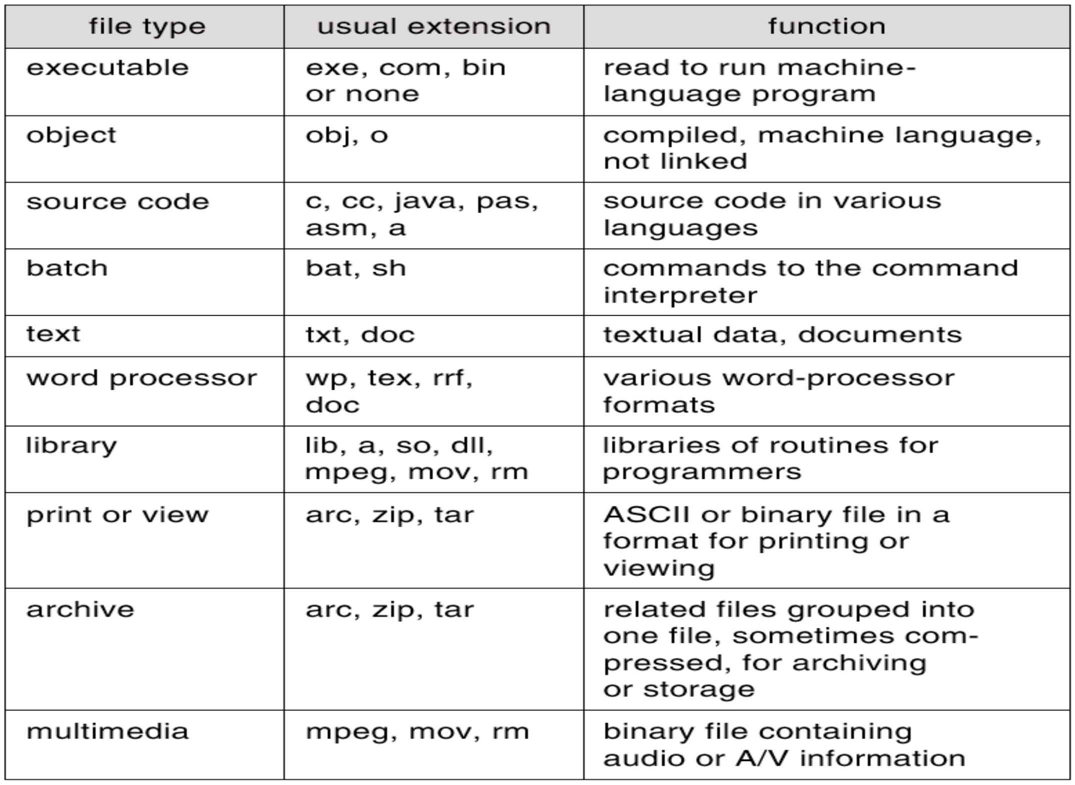


Figure 7.13 Indexed allocation of disk space

8.b) Explain different types of file types

An OS can operate the file in a required manner only if it recognizes the type of that file. A file name is split into two parts – name and extension. The file extension normally indicates the type of a file. Table 7.1 indicates various file types.



file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Table 7.1 Common File Types

9.a) Explain with a diagram components of a Linux OS

The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementations:

1. Kernel. The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.

2. System libraries. The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality that does not needthe full privileges of kernel code.

3. System utilities. The system utilities are programs that perform individual, specialized management tasks. Some system utilities may be invoked just once to initialize and configure some aspect of the system; others—

known as *daemons* in UNIX terminology—may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files.

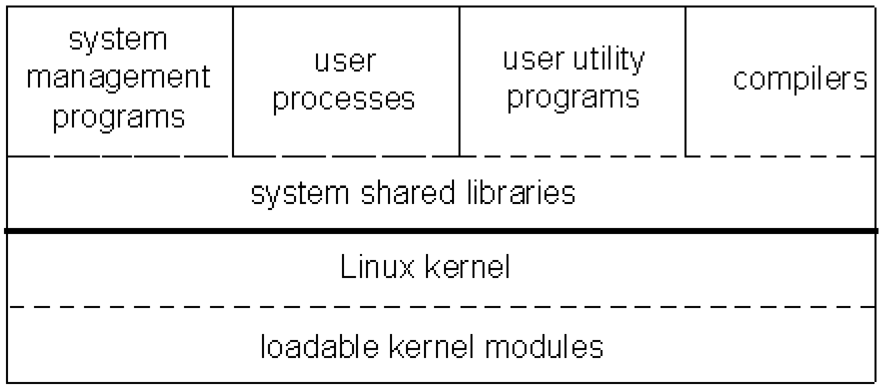


Figure 21.1 illustrates the various components that make up a full Linux system. The most important distinction here is between the kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer. Linux refers to this privileged mode as **kernel mode.**

9.b) Explain different types of process management in linux OS

A process is the basic context within which all user-requested activity is serviced within the OS. To be compatible with other UNIX systems, Linux must use a process model similar to those of other versions of UNIX.

8.11.1 The Fork/Exec Process Model

UNIX process management separates the creation of processes and the running of a new program into two distinct operations. The ***fork*** system call creates a new process. A new program is run after a call to ***exec***. Under UNIX, a process encompasses all the information that the operating system must maintain t track the context of a single execution of a single program.

Under Linux, process properties fall into three groups:

- **Process Identity:** The process identity consists of mainly following items:
- **rocess ID (PID)**: The unique identifier for the process; used to specify processes to the OS when an application makes a system call to signal, modify, or wait for another process.
- **Credentials:** Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files.
- **Personality:** Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls. Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX.
- **Process environment:** The process's environment is inherited from its parent, and is composed of two null-terminated vectors:

  o The **argument vector** lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself o The **environment vector** is a list of “NAME=VALUE” pairs that associates named environment variables with arbitrary textual values.

Passing environment variables among processes and inheriting variables by a process’s children are flexible means of passing information to components of the user-mode system software. The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole.

Context: It is the (constantly changing) state of a running program at any point in time. There are various parts:

Scheduling context is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process.

Accounting: The kernel maintains accounting information about the resources currently being consumed by each process, and the total resources consumed by the process in its lifetime so far.

o **File table** is an array of pointers to kernel file structures. When making file I/O system calls, processes refer to files by their index into this table.

o **File System Context:** Whereas the file table lists the existing open files, the file-system context applies to requests to open new files. The current root and default directories to be used for new file searches are stored here.

o **Signal-handler table** defines the routine in the processs’s address space to be called when specific signals arrive.

o **Virtual-memory context** of a process describes the full contents of its private address space.

Processes and Threads

Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent. A distinction is only made when a new thread is created by the **clone** system call:

- **fork** creates a new process with its own entirely new process context
- **clone** creates a new process with its own identity, but that is allowed to share the data structures of its parent

Using **clone** gives an application fine-grained control over exactly what is shared between two threads.

10.a) Explain inter process communication in Linux OS

Like UNIX, Linux informs processes that an event has occurred via signals. There is a limited number of signals, and they cannot carry information: Only the fact that a signal occurred is available to a process. The Linux kernel does not use signals to communicate with processes with are running in kernel mode, rather, communication within the kernel is accomplished via scheduling states and wait-queue structures.

Passing of Data among Processes: The pipe mechanism allows a child process to inherit a communication channel to its parent; data written to one end of the pipe can be read by the other. Shared memory offers an extremely fast way of communicating; any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space. To obtain synchronization, however, shared memory must be used in conjunction with another inter process communication mechanism.

10.b) With a diagram, explain memory management in Linux OS

- Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory
- It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes

Linux's physical memory-management system deals with allocating and freeing pages, groups of pages, and small blocks of memory. It has additional mechanisms for handling virtual memory, memory mapped into the address space of running processes.

Management of Physical Memory:
The page allocator allocates and frees all physical pages; it can allocate ranges of physically-contiguous pages on request. The allocator uses a buddy-heap algorithm to keep track of available physical pages:

o Each allocatable memory region is paired with an adjacent partner.
o Whenever two allocated partner regions are both freed up they are combined to form a larger region.
o If a small memory request cannot be satisfied by allocating an existing small free region, then a larger free region will be subdivided into two partners to satisfy the request.

Memory allocations in the Linux kernel occur either statically (drivers reserve a contiguous area of memory during system boot time) or dynamically (via the page allocator).

Management of Virtual Memory: The VM system maintains the address space visible to each process: It creates pages of virtual memory on demand, and manages the loading of those pages from disk or their swapping back out to disk as required. The VM manager maintains two separate views of a process's address space:
A logical view describing instructions concerning the layout of the address space. The address space consists of a set of nonoverlapping regions, each representing a continuous, page-aligned subset of the address space.
o A physical view of each address space which is stored in the hardware page tables for the process.
Virtual memory regions are characterized by:
o The backing store, which describes from where the pages for a region come; regions are usually backed by a file or by nothing (demand-zero memory)
o The region's reaction to writes (page sharing or copy-on-write). The kernel creates a new virtual address space
1. When a process runs a new program with the exec system call
2. Upon creation of a new process by the fork system call

Executing and Loading User Programs:
Linux maintains a table of functions for loading programs; it gives each function the opportunity to try loading the given file when an exec system call is made. The registration of multiple loader routines allows Linux to support both the ELF and a.out binary formats. Initially, binary-file pages are mapped into virtual memory; only when a program tries to access a given page will a page fault result in that page being loaded into physical memory. An ELF-format binary file consists of a header followed by several page-aligned sections; the ELF loader works by reading the header and mapping the sections of the file into separate regions of virtual memory.