Karnatak Law Society's

Vishwanathrao Deshpande Institute of Technology, Haliyal.

Department of computer science & Engg

Sample Question Paper

Sub : Introduction to operating systems

Subcode : 18CS654

Module : 1

1a. Distinguish between the following

a) multiprogramming and multitasking

b) Multiprocessor systems & clustered system 4M

b What are system programs? Briefly discuss its types. 6M

c. Demonstrate the concept of virtual machine with example. 8M

OR

2a. Explain the role of operating system from different view points. Explain the dual mode of operation of operating system. 8M

2b. Analyse modular kernel approach with layered approach with a neat diagram. 6Mh

2c. List and explain the services provided by operating system for the user and efficient operation of system. 6M

Module -2

3a Explain threading models. Discuss benefits of multithreaded program 8M

b. Explain IPC in Posix 6M

3c. Explain Scheduler activations with example. 6M

OR

4a Describe implementation of IPC using shared memory and message passing 8M

4b Demonstrate the operations of process creation and process termination in UNIX. 6M.

4c. Explain short term, long term and medium term Schedulers. 6M

Module-3

5a What are semaphores? Explain how mutual exclusion is implemented with semaphores. 6M

b. Is cpu scheduling necessary? Discuss five different criteria used in CPU scheduling. 6M

c. What are monitors? Explain dining philosophers problem using Monitors. 8M

OR

6a. For the process listed below, draw gantt charts using preemptive & non preemptive priority scheduling algorithm. A larger priority number has higher priority. Calculate average waiting time & average turn around time.

Process	Arrival Time	Burst Time	Priority
P_1	0	10	5
P_2	1	6	4
P_3	3	2	2
P_4	5	4	0

6M

6b. Define Test and set() and Swap() instruction. Explain how mutual exclusion is implemented using these instructions. – 8M

6c Explain Readers-writers problem with semaphore in detail. – 6M.

Module 4

7a. Explain how prevention from deadlock takes place. 6M

b. Analyse the problems in simple paging technique & show how TLB is used to solve the problem. Discuss how effective access time can be computed with suitable example. 8M.

7c. Define paging. Explain paging hardware with a neat block diagram. 6M.

OR

8a System consists of five jobs (J1, J2, J3, J4, J5) & three resources (R1, R2, R3). Resource type R1 has 10 instances, R2 has 5 instances & R3 has 7 instances. The following snapshot of the system has been taken.

Jobs	Allocation			maximum			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
J1	0	1	0	7	5	3	3	3	2
J2	2	0	0	3	2	2			
J3	3	0	2	9	0	2			
J4	2	1	1	2	2	2			
J5	0	0	2	4	3	3			

Find need matrix & calculate the safe sequence by using bankers algorithm. Mention the above system is safe or not safe. 8M

8b Given the memory partition of 200k, 700k, 500k, 300k, 100k, 400k. Apply first fit & best fit & worst fit to place 315k, 427k, 250k, 550k. which algorithm makes most efficient use of memory. 4M

8c. What is segmentation? Explain basic method of segmentation with example. 8M

Module 5

9a Explain demand paging in details. 7M

9b. Explain various directory structures briefly. 6M

9c. Explain the steps to handling page faults with a neat diagram. 7M

OR

10a Consider the following page reference string 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1. How many page faults would occur for LRU, optimal & FIFO page replacement algorithm. Assume 3 frames. Which of the above algorithm is more efficient. 8M

10b Explain memory mapping on files with a neat diagram 6M.

10c. What is thrashing? How it can be controlled. 6M.

Karnatak Law Society's

Vishwanathrao Deshpande Institute of Technology, Haliyal.

Department of Computer Science & Engg

Question Paper Solution

Sub: Introduction to Operating System [18CS654]

Sample Question paper.

Staff Name: Prof. Yasmeen Shaikh.

Module 1

1a Distinguish between the following

a) multiprogramming and multitasking.

b) multiprocessor systems and clustered systems.

→a) Multiprogramming

i) Multiprogramming system provide an environment in which various system resources are utilized effectively.

ii) do not provide user interaction with computer system

iii) reduces the CPU idle time as long as possible

Multitasking

Multitasking or time sharing is a logical extension of multiprogramming, in which CPU executes multiple jobs by switching among them

ii) Switching is so frequently that user can interact with each program while it is running.

iv) improves CPU utilization by increasing responsiveness

b) Multiprocessor Systems

- These systems have two or more processors in close communication, sharing the bus, & sometimes the clock, memory & peripheral devices
- In a multiprocessor system, the failure of one operating system takes everything down
- multiprocessor systems are faster because work communication & coordination is handled by cache memory.

Clustered systems

- These systems are composed of two or more individual systems coupled together & are connected over network.
- cluster systems have multiple operating system instances, so that the failure of one or does'nt take out all the machines.
- clustered computers are slower, they have to communicate/coordinate via network.

16. What are system programs? Briefly discuss its types. SM

- System program provides a convenient environment for program development and execution. Eg

System programs can be divided into these categories

i) File management - These programs create, delete copy, rename, print, dump, list and generally manipulate files and directories.

ii) Status information - Some programs simply ask the system for the date, time, amount of available memory or disk space, number of users, or similar status information.

iii) File modification - Several text editors may be available to create and modify the content of files stored on disk or other storage devices. There may be special commands to search contents of files or perform transformations of the text.

iv) Programming-language support - Compilers, assemblers, debuggers and interpreters for common programming languages such as C, C++, Java, VB, Perl are often provided to the user with the operating system

v) Program loading & execution - Once a program is assembled or compiled, it must be loaded into memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors & overlay loaders.

vi) Communications - These programs provide the mechanism for creating virtual connections among processes, users & computer systems. They allow users to send messages to one another's screen, to browse web pages, to send electronic-mail messages, to log in remotely, or to transfer files from one machine to another.

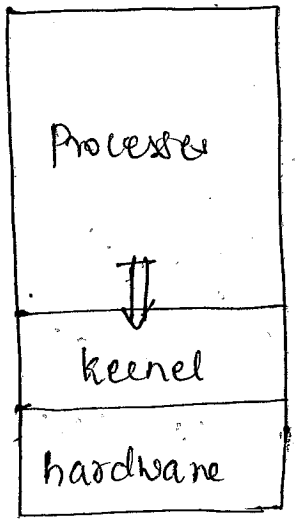
Ic demostrate the concept of virtual machine with example. 8M.

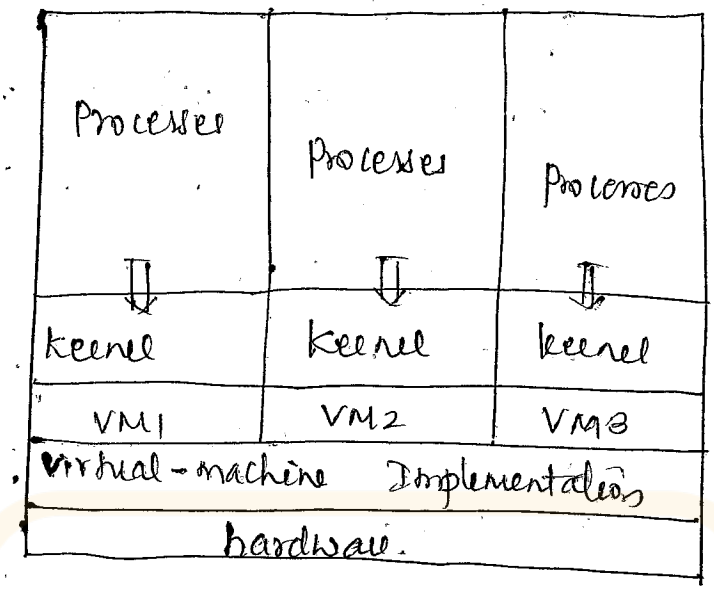
→ The fundamental idea behind a virtual machine is to abstract the hardware of a single computer into several different execution environments, thereby creating illusion that each separate execution environment is running its own private computer. By using CPU scheduling and virtual-memory techniques, an OS can create illusion that a process has its own processor with its own (virtual) memory.

Normally a process has additional features, such as system calls & a file system, that are not provided by the bare hardware.

- The virtual machine approach does not provide any such additional functionality but rather provides an interface that is identical to the underlying bare hardware.
- Each process is provided with a (virtual) copy of the underlying computer, as shown in figure below.
- There are several reasons for creating a virtual machine, all of which are fundamentally related to being able to share same hardware yet run

Several different execution environments:





A major difficulty with the virtual machine approach involves disk systems.

- Suppose that the physical machine has three disk drives but wants to support seven virtual machines. Clearly, it cannot allocate a disk drive to each virtual machine, because the VM software itself will need substantial disk space to provide virtual memory & spooling.

Benefits

The virtual machine concept has several advantages.

i) In this environment, there is complete protection of the various system resources.

ii) Each virtual machine is completely isolated from all other virtual machines, so there are no protection problems.

example: VMware is a popular commercial application that abstracts Intel 80x86 hardware into isolated virtual machines.
VMware runs as an application on a host OS such as Windows or Linux & allows this host system to concurrently run several different guest operating systems as independent virtual machines.

OR

2a Explain the role of operating system from different view points - Explain the dual mode of operating of operating system 8M.

→ The role of operating system from different view points

i) user view -

- user view of the computer varies according to the interface being used.
- Most computer user sits in front of a PC, consisting of a monitor, keyboard, mouse & system unit. Such a system is designed for one user to monopolize its resources.
- The goal is to maximize the work that the user is performing.
- In this case OS is designed for ease of use, with some attention paid to performance & resource utilization.

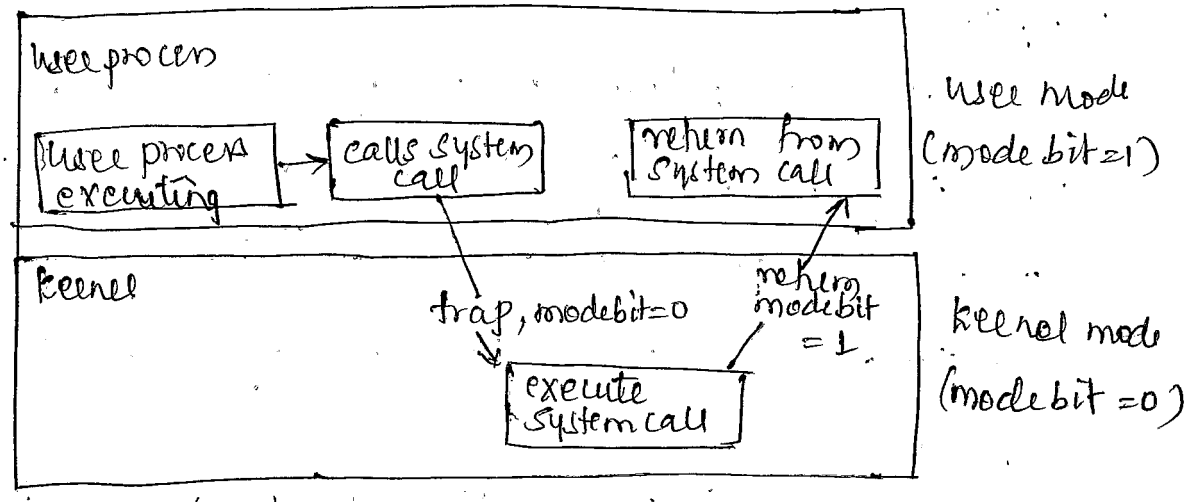
ii) System view -

- from the computer point of view the OS is the program mostly involved with the hardware.
- In this context, OS can be viewed as a resource allocator. OS acts as the manager of these resources.
- A slightly different view of an OS emphasizes the need to control the various I/O devices & user programs.
- An OS is a control program which manages the execution of user programs to prevent errors & improper use of the computer.

Dual mode of operating System

In order to ensure the proper execution of the OS, we need to distinguish between the execution of the OS code & user defined code.

- We need two separate modes of operation : user mode and kernel mode.
- A bit, called the mode bit, is added to the hardware of the computer to indicate the current mode : kernel (0) or user (1).
- With the mode bit, we are able to distinguish between a task that is executed on behalf of the OS & one that executed on behalf of the user.



When a user application requests a service from the operating system, it must transition from user to kernel mode to fulfill the request.

- At system boot time, the h/w starts in kernel mode.
- The OS is then loaded & starts user applications in user mode.
- Whenever a trap or interrupt occurs, the h/w switches from user mode to kernel mode.
- When the OS gains control of the computer, it is in the kernel mode.

2b Analyse modular kernel approach with layered approach with a neat diagram.

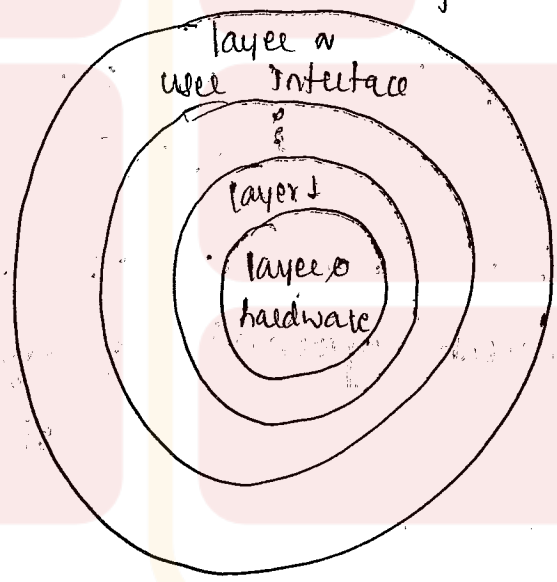
- With proper hardware support, OS can be broken into pieces that are smaller & more appropriate than those allowed by the original MS-DOS or UNIX systems.
- The OS can then retain much greater control over the computer & over the applications that make

use of that computer.

Implementers have more freedom in changing the inner workings of the system & in creating modular operating systems.

- A system can be made modular in many ways. One method is the layered approach, in which the OS is broken up into a number of layers.
- The bottom layer (layer 0) is the hardware, the highest (layer N) is the user interface.

This layering structure is depicted in figure below.



The main advantage of the layered approach is simplicity of construction and debugging.

The layers are selected so that each user function & services of only lower-level layers.

- This approach simplifies debugging and system verification.
- The first layer can be debugged without any

correct for the rest of the system.

Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged & so on.

- If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged.

Thus the design & implementation of the system is simplified.

2c. List and explain services provided by the operating system for the user & efficient operation of system.

⇒ Services provided by the operating system for the user :

i) User interface
ii) program execution
iii) I/O operations
iv) File-system manipulation
v) Communications
vi) Error detection

i) User interface - All OS have user interface for interaction with computer system. One is a command-line interface which uses text commands. Another is

graphical user interface (GUI) - which provides menus & icons to interact.

ii) Program execution - The system must be able to load a program into memory and run that program.

iii) I/O operations - A running program may require I/O which may involve file or an I/O device

iv) File system manipulation - The file system is of particular interest. Program needs to read & write files & directories. They also need to create & delete them by name, search for a given file & list file information

v) Communications - communication among processes may be implemented via shared memory or through message passing, in which packets of information are moved between processes by the OS.

vi) Error detection

Errors may occur in the CPU & memory hardware, in I/O devices. For each type of error, the OS should be take the appropriate action to ensure correct & consistent computing.

Services provided by the OS for efficient operation of the system are

i) Resource allocation ii) Accounting

iii) Protection & security.

i) Resource allocation – when there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. For this purpose many types of resources are managed by OS

ii) Accounting – To keep track of which users use how much & what kind of computer resources, OS maintain accounting informations. These usage statistics may be a valuable tool for researchers who wish to configure the system to improve computing services.

iii) Protection & security –

OS must provide protection & security of information in shared environment. Protection involves ensuring that all access to system resources is controlled. Security from outsiders is equally important.

Module 2

3a. Explain threading models. Discuss benefits of multithreaded program. 8M

⇒ Multithreading models are

i) Many - to - one model

ii) One - to - one model

iii) Many - to - many model.

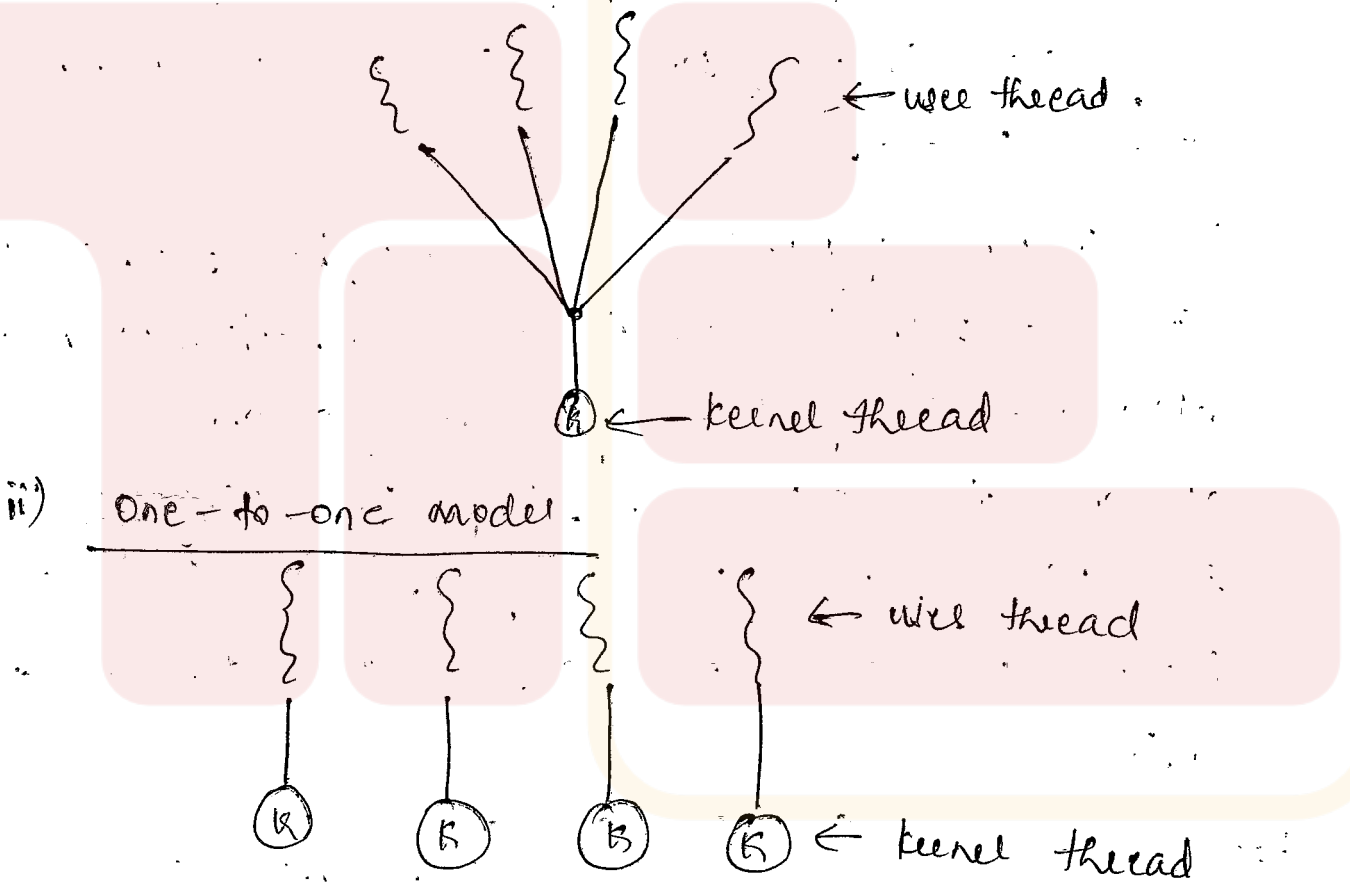
i) Many - to - one model.

Many - to - one model maps many user-level threads to one kernel thread. Thread management

is done by thread library in user space, so it is efficient, but entire process will block if a thread makes a blocking system call. Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

ex Green threads

Figure below shows many to one model.



ii) One-to-one model.

The one-to-one model maps each user thread to a kernel thread. as shown in figure above It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call; It also allows multiple threads to run in

parallel on multiprocessor.

Only drawback of this model that creating a user thread requires creating the corresponding kernel thread.

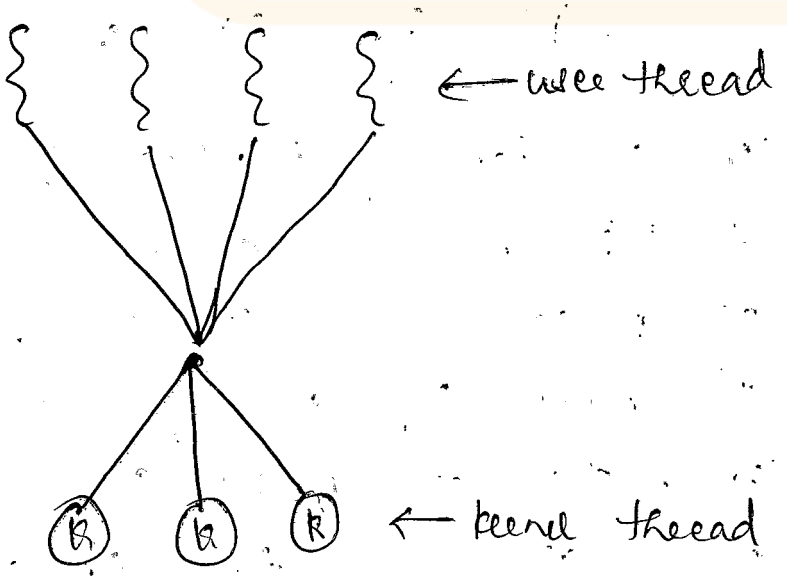
ex - ~~Freelinux~~, Windows 95, 98, NT, 2000 & OS/2

implement one-to-one model.

iii) Many to many model

The many to many model multiplexes many user-level threads to ~~to~~ smaller or equal number of kernel threads.

The number of kernel threads may be specific to either a particular application or a particular machine. However, many to many model allows the developer to create as many user threads as he wishes, true concurrency is not gained because the kernel can schedule one thread at a time.



Benefits of multithreaded programming can be broken down into four major categories.

i) Responsiveness - Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

ii) Resource sharing - By default, threads share the memory & the resources of the process to which they belong. The benefit of sharing code & data allow an application to have several different threads of activity within the same address space.

iii) Economy - Allocating memory & resources for process creation is costly. Because threads share resources of the process to which they belong, it is economical to create & context switch threads.

iv) Utilization of multiprocessor architecture - The benefits of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. Multithreading on a multi-CPU machine increases concurrency.

3b Explain IPC in Posix. 6M

→ A process must first create a shared memory segment using shmget() system call. The following example illustrates the use of shmget():

```
Segment_id = shmget(IPC_PRIVATE, size, S_IRUSR |
                                       S_IWUSR);
```

This first parameter specifies the key of the shared_memory segment. If this is set to IPC_PRIVATE, a new shared-memory segment is created. The second parameter specifies the size of the shared memory.

The third parameter specifies the mode, which indicates how shared_memory segment is to be used. i.e. for reading, writing or both.

By setting the mode to S_IRUSR | S_IWUSR, we are indicating that the owner may read or write to the shared memory segment.

A successful call to shmget() returns an integer identifier for the shared memory segment.

Processes that wish to access a shared-memory segment must attach it to their address space using the shmat() system call.

The call to shmat() expects 3 parameters. The first parameter is the integer identifier of the shared-memory segment being attached, & the second parameter is a pointer location in memory indicating where the shared memory will be attached.

If we pass a value of NULL, the OS selects the location on the user's behalf.

The third parameter identifies a flag that allows the shared memory region to be attached in a read-only or read-write mode, by passing a parameter of 0, we both both reads & writes to the shared region.

We attach a region of shared memory using shmat() as follows:

```
shared-memory = (char *) shmat (id, NULL, 0);
```

If successful, shmat() returns a pointer to the beginning location in memory where the shared-memory region has been attached.

Once the region of shared memory is attached to a process's address space, the process can access the shared memory as a routine memory access using the pointer returned from shmat().

shmat() returns a pointer to a character string. Thus, we could write to the shared-memory region as follows:

```
sprintf(shared-memory, "writing to shared memory");
```

When a process no longer requires access to the shared-memory segment, it detaches the segment from its address space.

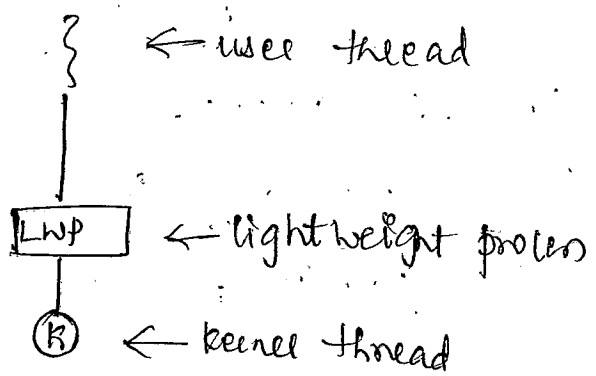
To detach a region of shared memory, the process can pass the pointer of the shared-memory region to shmdt() system call, as follows:

```
shmdt(shared-memory);
```

A shared-memory segment can be removed from the system with the shmctl() system call, which is passed the identifier of the shared segment along with the flag IPC-RMID.

3c. Explain scheduler activation with example. 6M

— Many systems implementing either the many to many or two level model place an intermediate data structure between the user & kernel threads.



This data structure typically known as light weight process or LWP as shown in figure above. An application may require any number of LWPs to run efficiently.

One scheme for communication between the user-thread library & the kernel is known as scheduler activation. It works as follows:

- The kernel provides an application with a set of virtual processors (LWPs) & the application can schedule user threads onto an available virtual processor.
- The kernel must inform an application about certain events. This procedure is known as an upcall.
- upcalls are handled by the thread library with an upcall handler, & upcall handlers must run on a virtual processor.
- One event that triggers an upcall occurs when an application thread is about to block.
- For this scenario, the kernel makes an upcall to the application informing it that a thread is about to block & identifying the specific thread.
- The kernel then allocates a new virtual processor to the application.

The application runs an upcall handler on this new virtual processor, which saves the state of the blocking thread & relinquishes the virtual processor, on which the blocking thread is running.
- The upcall handler then schedules another thread that is eligible to run on the new virtual processor.
- When the event that the blocking thread was waiting for occurs, the kernel makes another upcall to the thread library informing it that the previously blocked thread is now eligible to run.
- The upcall handler for this event also requires a virtual processor & the kernel may allocate a new virtual processor or preempt one of the user threads & run the upcall handler on its virtual processor.
- After marking the unblocked thread as eligible to run, the application schedules an eligible thread to run on an available virtual processor.

OR

4a Describe Implementation of IPC using shared memory & message passing. [8M]

→ 1) Interprocess communication (IPC) using shared memory requires communicating processes to establish a region of shared memory.

Typically, a shared memory region resides in the address space of the process creating the shared-memory segment.

other processes that wish to communicate using this shared memory segment must attach it to their address space.

We can illustrate shared memory using producer-consumer problem

To allow a producer & consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer & emptied by the consumer.

- This buffer will reside in a region of memory that is shared by the producer & consumer processes.
- A producer can produce one item while the consumer is consuming another item.

The producer & consumer must be synchronized so that the consumer does not try to consume an item that has not yet been produced.

- Two types of buffers can be used.

a) unbounded buffer places no practical limit on the size of the buffer.

The consumer may have to wait for new items, but the producer can always produce new items.

b) bounded buffer... assumes a fixed buffer size.

The consumer must wait if the buffer is empty, & the producer must wait if the buffer is full.

The following variables reside in a region of memory shared by the producer & consumer processes.

```
#define BUFFER_SIZE 10
typedef struct
{
...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

The shared buffer is implemented as a circular array with 2 logical pointers in & out. The buffer is empty when in == out, the buffer is full when ((in+1) % BUFFER_SIZE) == out.

The code for producer process is

```
item nextproduced;
while (true)
{
    while (((in+1) % BUFFER_SIZE) == out)
        ;
    buffer[in] = nextproduced;
    in = (in+1) % BUFFER_SIZE;
};
```

The code for consumer process is

```
item next_consumed;
while (true)
{
    while ( in == out)
        ;
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
```

ii) Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
A message-passing facility provides at least two operations: send(message) & receive(message).
Messages sent by a process can be of either fixed or variable size
If processes P & Q want to communicate, they must send messages to & receive messages from each other; a communication link must exist between them.
Here are several methods for logically implementing a link & the send() / receive() operations:

i) Direct or Indirect communication
ii) Synchronous or asynchronous communication
iii) Automatic or explicit buffering.

a) Under direct communication, each process must explicitly specify the recipient or sender name

In this scheme, the send() & receive() primitives are defined as

send(P, message) & receive(Q, message).

A communication link has the following properties

i) a link is established automatically between every pair of processes that want to communicate

ii) A link is associated with exactly two processes

iii) there exist exactly one link between each pair of processes.

b) In indirect communication, the messages are sent to & received from mailboxes or ports.

A mailbox can be viewed abstractly as an object into which messages can be placed by processes & from which messages can be removed. Each mail box has a unique identification.

Two processes can communicate only if the processes have a shared mailbox, the send() & receive() primitives are defined as follows

send(A, message) & receive(A, message)

A communication link has following properties

i) a link is established between a pair of processes only if both members of a pair have a shared mailbox

ii) a link may be associated with more than

two processes

iii) Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox

c) Synchronization

Communication between processes takes place through calls to send() & receive primitives.

message passing may be either blocking or nonblocking - also known as synchronization & asynchronous.

Blocking send - sending process is blocked until the message is received by the receiving process or by the mailbox

Nonblocking send - sender sends the message and continue

Blocking receive - receiver is blocked until a message is available

Non blocking receive - receiver receives a valid message or NULL

d) Buffering - Queue of messages are attached to link & are implemented in 3 ways

i) Zero capacity - store maximum of message. Sender must wait for receiver

ii) Bounded capacity - finite length of n messages. Sender waits if that is full.

iii) Unbounded capacity - infinite length, Sender never waits

4. Demonstrate the operations of process creation & process termination in UNIX. 6M

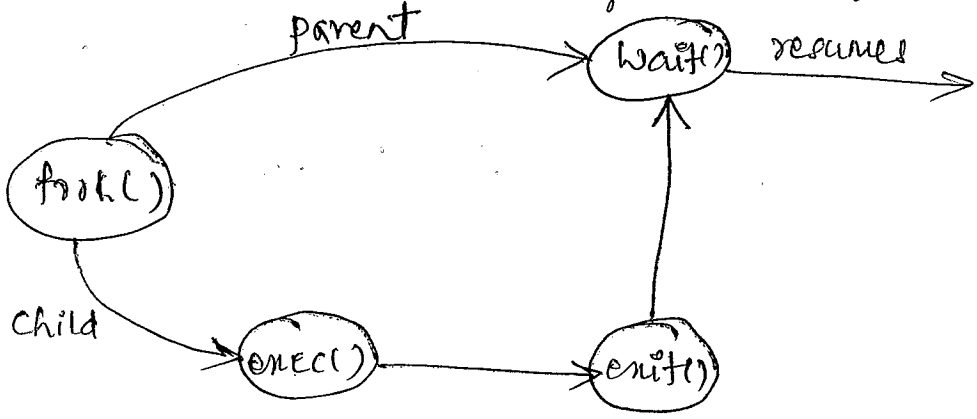
→ Process creation

- A process may create new process via a create process system call during the course of execution.
- The creating process is called parent process, and the new processes are called the children of that process.
- Each of these new processes may in turn create other processes, forming a tree of processes.
- Generally, process identified and managed via a process identifier (pid).
- When a process create a new process, two possibilities exist in terms of execution
  a) the parent continues to execute with its children
  b) the parent waits until some or all of its children have terminated
- There are two possibilities in terms of the address space of the new process
  a) The child process is a duplicate of the parent process (it has same program &

data as the parent)

b) the child process has a new program loaded into it.

- A new process created by the fork() system call
- The new process consists of a copy of the address space of the original process.
- This mechanism allows the parent process to communicate easily with its child process.
- The return code for the fork() is zero for the child process & nonzero for parent process
- The exec() system call is used for a fork() system call by one of the two processes to replace the process's memory space with a new program.
- The parent can create more children or if it has nothing else to do while the child children exist, it can issue wait() system call to move itself to the ready queue until the termination of the child



Process Termination

- A process terminates when it finishes executing its final statement & asks the OS to delete it by using the exit() system call.
- At this point, the process may return a status value to its parent process via wait() system call.
- All the resources of the process are deallocated by the OS.
- A process can cause termination of another process via an appropriate system call ex TerminateProcess() in Win32.
- A parent may terminate the execution of one of its children for a variety of reasons
  - i) The child has exceeded its usage of some of the resources that it has been allocated
  - ii) The task assigned to child is no longer required
  - iii) The parent is exiting and the OS does not allow a child to continue if its parent terminates

4c Explain short term, long term and medium term schedulers. 6M.

→ Short term scheduler

The short term scheduler selects among the processes that are ready to execute & allocates CPU to one of them.

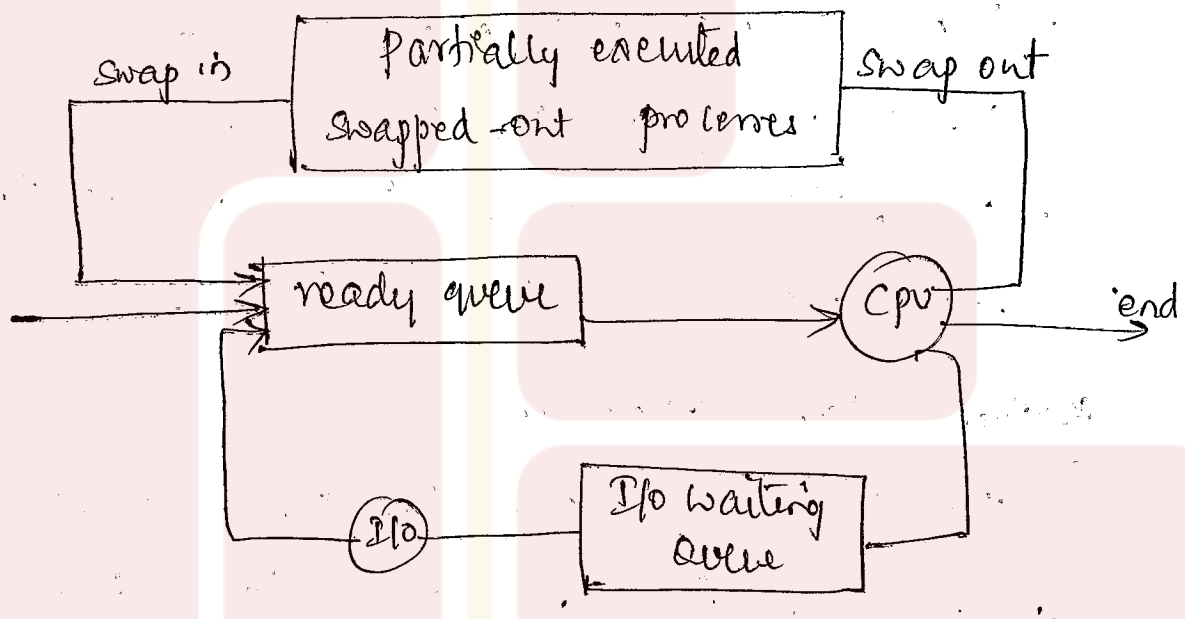
- The short-term scheduler must select a new process for the CPU frequently.
- A process may execute for only a few milliseconds before waiting for an I/O request.
- Often, the short term scheduler executes at least once every 100 milliseconds

Long - term scheduler

- The long term scheduler or job scheduler selects process from job pool and load them into Memory for execution.
- The long-term scheduler executes much less frequently,
- Long term scheduler controls the degree of multiprogramming
- Because of the longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution

Medium term scheduler

- The key idea behind medium term scheduler is that it can be advantageous to remove processes from memory and thus reduce the degree of multiprogramming.
- Later the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called swapping.



Module - 3

5a What are semaphores? Explain how mutual exclusion is implemented with semaphores. 6M

→ Semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait() & signal().

The definition of wait() is as follows

```
wait (S)
{
  while (S<=0)
    ;
  S--;
}
```

The definition of signal is as follows

```
signal(S)
{
  S++;
}
```

One some systems, binary semaphores are known as muten lock, as they are locks they provide mutual exclusion.

We can use binary semaphore to deal with the critical section problem for multiple proceses.

The n proceses share a semaphore, muten initialized to 1.

Each proces P_i can accen the critical section as follows

```
do
{
  waiting (muten);
    // critical section
  signal (muten)
    // remainder section
} while (TRUE);
```

The procen Pi which wishes to execute its critical section executes wait (mutex) & enters critical section.

On completion of execution procen Pi executes signal (mutex); which signals waiting procen to enter critical section next.

5b. Is CPU scheduling necessary? ~~discuss~~ Give different criteria used in CPU scheduling.

→ CPU scheduling is necessary to decide which procen should be scheduled on CPU for execution. Every time a procen is ready to execute CPU scheduler select the procen for execution & dispatcher gives control of the CPU to the procen selected by the CPU scheduler.

The scheduling criteria are as follows:

i) CPU utilization - we want to keep the CPU as busy as possible. CPU utilization can range from 0 to 100%. In a real system, it should range from 40% to 90%.

ii) Throughput - If the CPU is busy executing procenes then work is being done. One measure of work is number of procenes that are completed per time unit called throughput

iii) Turn around time :- From the point of view of a particular process, the important criterion is 'how long it takes to execute that process'. The interval from the time of submission of a process to the time of completion is the turnaround time.

iv) Waiting time : The CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O ; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of periods spent waiting in the ready queue.

v) Response time : In an interactive system, turnaround time may not be the best criterion. Often a process can produce some output fairly early & can continue computing new results while previous results are being of a request until first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response. The turn around time is generally limited by the speed of the output device.

Sc. what are monitors? Explain dining philosophers problem using monitors. 8M

→ Monitor is a type of abstract data type, encapsulates private data with public methods to operate on the data.

A monitor type represents a set of programmer defined operations that provide mutual exclusion within the monitor.

The monitor type also contains the declaration of variable whose values define the state of an instance of that type, along with the bodies of procedures or functions that operate on these variables.

The syntax of monitor is as follows

```
monitor monitor_name
{
    // shared variable declarations
    procedure P1()
    {
    }
    procedure P2()
    {
    }
    :
    procedure Pn()
    {
    }
    initialization_code()
    {
    ...
    }
}
```

- Monitor solution to dining philosophers problem imposes the restriction that a philosopher may pick up her chopsticks only if both of them are available.
- To code this solution, we need to distinguish among 3 states in which we may find a philosopher.
- For this purpose, we introduce the following data structures.

  enum {thinking, hungry, eating} state[5];
- Philosopher i can set the variable state[i] = eating only if her two neighbors are not eating:

  (state[(i+4)%5] != eating) && (state[(i+1)%5] != eating)
- We also need to declare

  condition self[5];

  where philosopher i can delay herself when she is hungry but is unable to obtain the chopsticks she needs.
- The distribution of the chopsticks is controlled by monitor dp, whose definition is given below.

```
monitor dp
{
 enum {THINKING, HUNGRY, EATING} state[5];
 condition self[5];
 void pickup (int i)
 {
  state[i] = THINKING;
  test(i);
```

```
    if (state[i] != EATING)
        self[i].wait();
}

void putdown(int i)
{
    state[i] = THINKING;
    test((i+4)%5);
    test((i+1)%5);
}

void test(int i)
{
    if (state[(i+4)%5] != EATING) &&
        (state[i] == HUNGRY) &&
        (state[(i+1)%5] != EATING)
    {
        state[i] = EATING;
        self[i].signal();
    }
}

initialization_code()
{
    for (int i=0; i<5; i++)
        state[i] = THINKING;
}
```

Each philosopher, before starting to eat, must invoke the operation pickup(). This may result in the suspension of the philosopher process.

- After the successful completion of the operation, the philosopher may eat.
- Following this, the philosopher invokes the putdown() operation.
- The philosopher i must invoke the operations pickup() & putdown() in the following sequence

```
dp.pickup(i);
...
eat();
...
dp.putdown(i);
```

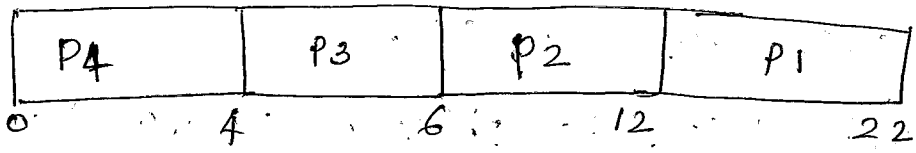
It is easy to show that this solution ensures that no two neighbours are eating simultaneously & that no deadlocks will occur.

OR

6a. For the process listed below, draw gantt chart using preemptive & non preemptive priority scheduling algorithm. A larger priority number has higher priority. Calculate average waiting time and average turn around time. 6M

Process	Arrival time	Burst time	Priority
P1	0	10	5
P2	1	6	4
P3	3	2	2
P4	5	4	0

→ non preemptive priority scheduling



Waiting time of P1 = 12

Waiting time of P2 = 6

waiting time of P3 = 4

Waiting time of P4 = 0

Avg waiting time = $\frac{(12+6+4+0)}{4}$

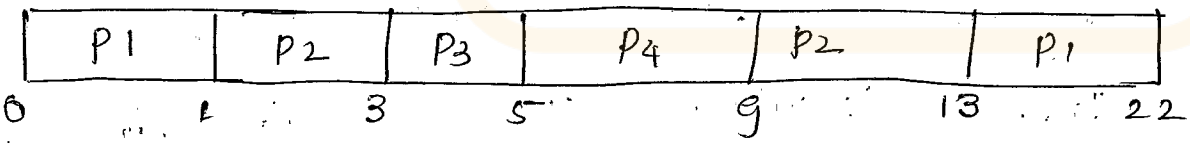
= 5.5 millisecond

Average turn around time = $\frac{(22-0)+(12-6)+(6-4)+(4-0)}{4}$

= $\frac{10+6+2+4}{4}$

= 5.5 millisecond

Preemptive priority scheduling



Average waiting time =

Waiting time of P1 = 13 − 1 = 12

Waiting time of P2 = 9 − 3 = 6

Waiting time of P3 = 3 − 3 = 0

Waiting time of P4 = 5 − 5 = 0

Avg waiting time = (12+06+0+0)/4

= 4.5 millisecond

Avg turnaround time = (10+6+2+4)/4

= 5.5 milliseconds

6b. Define Test and set() & Swap() instruction. Explain how mutual exclusion is implemented using these instructions

8M

→ TestANTSet() instruction can be defined as

```
boolean Testandset (boolean *target)
{
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```

The important characteristic is that this instruction is executed atomically.

Thus if two Testandset() instruction are executed simultaneously, they will be executed sequentially in some arbitrary order.

Mutual exclusion can be implemented by declaring a boolean variable lock, initialized to FALSE

```
do
{
while (TestandSetLOCK (&lock))
    ;
critical section
```

```
    lock = FALSE;
    remainder section
} while (TRUE);
```

The swap instruction can be defined as,

```
void swap(boolean *a, boolean *b)
{
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```

swap() instruction is executed atomically & mutual exclusion can be provided by declaring variable lock, initialized to false.

```
do
{
    key = TRUE;
    while (key == TRUE)
        swap(&lock, &key);

    critical section

    lock = FALSE;

    remainder section
} while (TRUE);
```

6C Explain Readers – writers problem with semaphore in detail. 6M.

→ A database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update the database. These processes can be referred to as readers & writers respectively.

- If two readers access the shared data simultaneously no adverse effects will result. If a writer & some other thread access the database simultaneously, chaos may happen.
- To ensure that these difficulties do not arise, we require that writers have exclusive access to the shared database. This synchronization problem is referred to as the readers – writers problem.
- To provide solutions using semaphores, following data structures are used.

```
Semaphore mutex, wrt;
int readcount;
```

The semaphores mutex & wrt are initialized to 1; readcount is initialized to 0.

The semaphore wrt is common to both reader & writer process.

... The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated.

- The readcount variable keeps track of how many processes are currently reading the object.
- The semaphore wrt functions as a mutual exclusion semaphore for the writers. It is also used by the first or last reader that enters or exits the critical section.
- It is not used by readers who enter or exit while other readers are in their critical section.

The code for writer process is:

```
do
{
    wait(wrt);
    ...
    //writing is performed
    ...
    signal(wrt);
} while(TRUE);
```

The code for reader process is

```
do
{
    wait(mutex);
    readcount++;
    if (readcount == 1)
        wait(wrt);
```

```
signal (mutex);
...
// reading is performed
...
wait (mutex);
readcount --;
if (readcount == 0)
    signal (wrt);
signal (mutex);
} while (TRUE);
```

MODULE-4

7a. Explain how prevention from deadlock takes place 6M.

→ For deadlock to occur four necessary conditions must hold. We can prevent the occurrence of a deadlock, by ensuring that atleast one of these conditions cannot hold.

1) Mutual exclusion

The mutual exclusion condition must hold for nonsharable resources. ex printer cannot be simultaneously shared by several processes.

- Sharable resources, do not require mutual exclusion & thus cannot be involved in deadlock.
- A process never needs to wait for a sharable resource.

ii) Hold & wait

To ensure that the hold & wait condition never occurs in the system, we must guarantee that, whenever a process request a resource, it does not hold any other resources.

Protocol1: It requires each process to request & be allocated all its resources before it begins execution.

Protocol 2: allows a process to request resources only when it has none.

iii) No preemption

To ensure that this condition does not hold, we can use the following protocol.

Protocol 1: If a process is holding some resources & request another resource that cannot be immediately allocated to it, then all resources current being held are preempted

Protocol 2: If a process request some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process & allocate them to the requesting process.

iv) Circular wait : One way to ensure that this condition never holds is to impose a total ordering of all resource types to require that each process requests resources in an increasing order of enumeration. Each resource is assigned a unique integer number

Protocol 1 : A process can initially request resources R_j. Then, a process can request instances of resource type R_i if and only if $F(R_j) > F(R_i)$

Protocol 2 :

We can require that, whenever a process requests an instance of resource type R_i, it has released any resources R_j such that $F(R_i) \geq F(R_j)$.

7b Analyse the problems in simple paging technique & show how TLB is used to solve the problem. Discuss how effective access time can be computed with suitable example. [8M]

→ Most computers allow large page table, for these machines use of fast registers to implement page table is not feasible, rather page table is kept in main memory a page table base register (PTBR) points to the page table changing page table - requires changing only this register. but with this approach, time required to access a user memory location is higher.

If we want to access location i, we must first index into page table using value in PTBR offset by the page number for i, this requires memory access; this gives frame number. This frame number is combined with page offset to produce actual address. Then we can access the desired place in memory. So two memory accesses are required, so memory access is slowed by factor of 2

The solution to this problem is to use a special, small fast lookup hardware called Translation look-aside buffer TLB. Each entry in TLB consists of two parts a key & a value.

When the associative memory is presented with an item, the item is compared with all keys simultaneously. If item is found, corresponding value field is returned. With this search is fast and hardware is expensive.

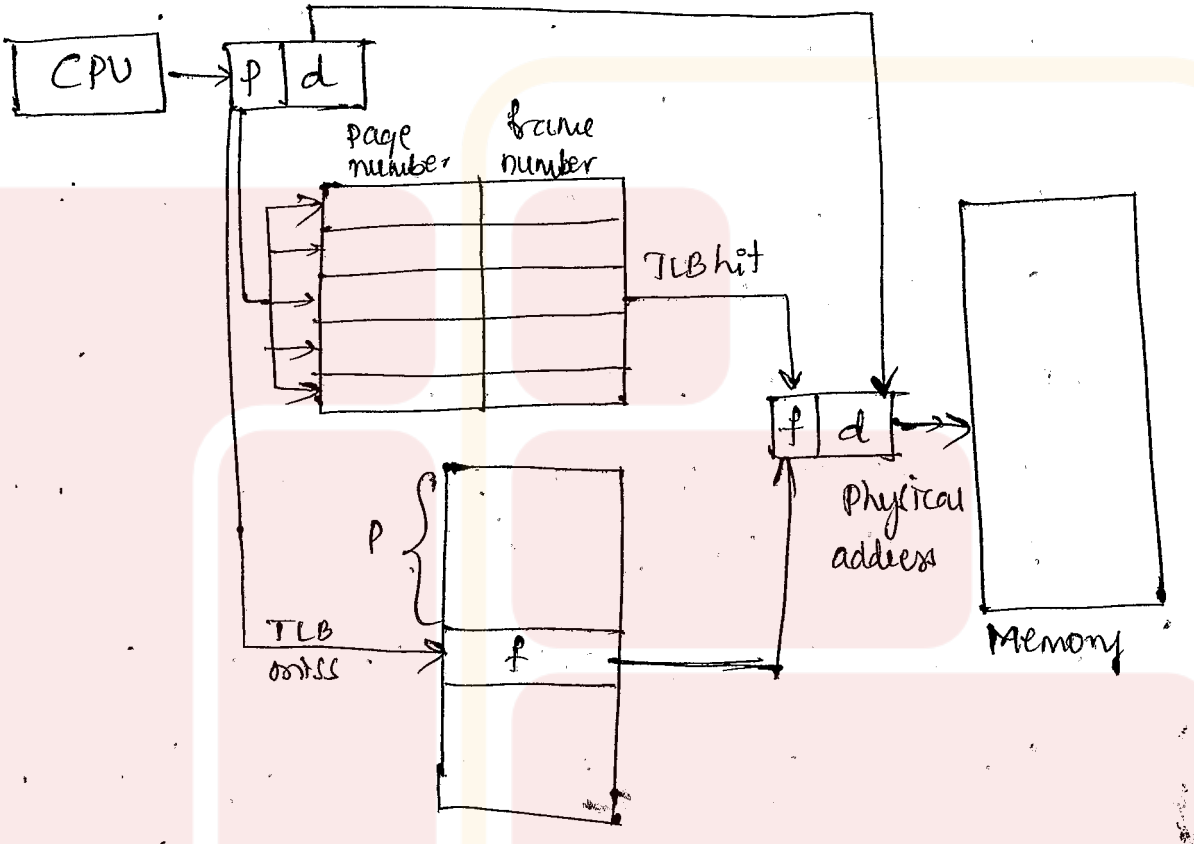
The TLB contains only a few of the page table entries.

When a logical address is generated by the CPU, its page number is presented to TLB.

If the page number is found (TLB hit), its frame

number is immediately available to used to access memory.

If page number is not in TLB (TLB miss) a memory reference to the pagetable must be made when frame no is obtained, we can use it to access memory.



An 80% hit ratio, means that we find the desired page in the TLB 80% of the time

If it takes 20ns to search TLB

100 ns to access memory.

∴ Memory mapped access = 100 + 20 = 120ns

When page is not in TLB

Memory access = 100 + 100 + 20 = 220ns

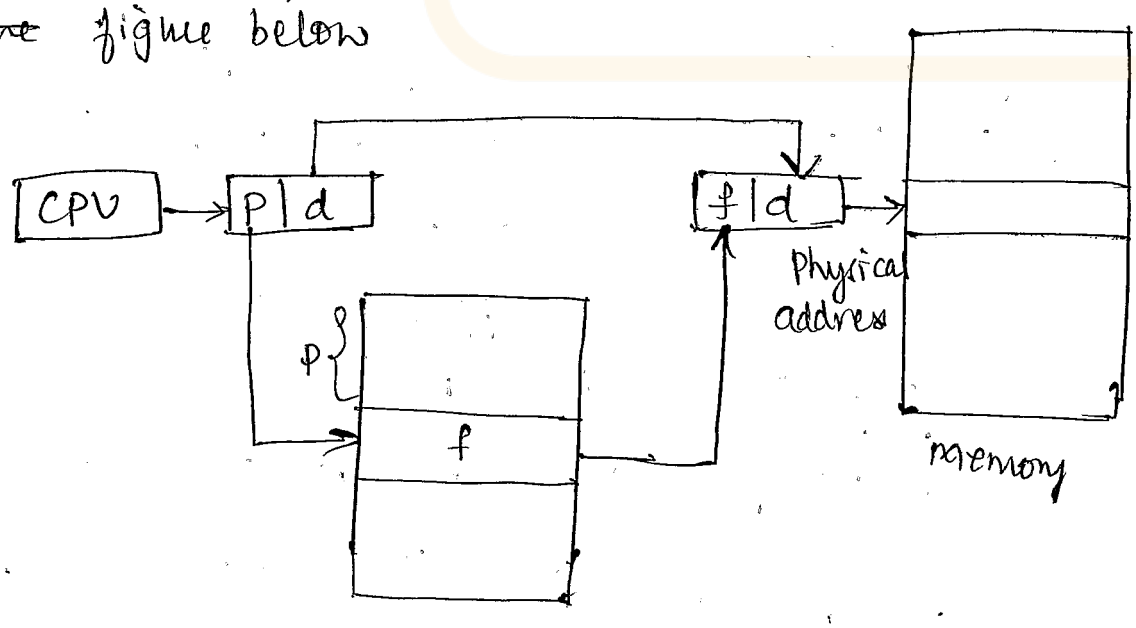
effective memory access time = 0.80 × 120 + 0.20 × 220 = 140 ns.

7C. Define paging. Explain paging hardware with a neat block diagram. 6M.

→ Paging is a memory management scheme that permits physical address space to be non contiguous.

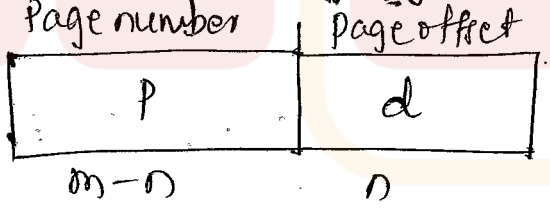
Basic method

- Physical memory is broken into fixed-sized blocks called frames.
- logical memory is broken into blocks of same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- Backing store is divided into fixed sized blocks that are of same size as memory frames.
- Hardware support for paging is illustrated in ~~above~~ figure below



Every address generated by CPU is divided into two parts page number (p) & page offset (d)
page number is used as index into a page table.
page table contains base address (f) of each page in physical memory. This base address is combined with page offset to define physical memory address that is sent to memory unit.

- page size is defined by hardware. Size of a page is typically a power of 2, varying between 512 bytes & 16 MB per page depending on computer architecture.
- If size of logical address space is 2^m & page size is 2^n addressing unit, then higher order $m-n$ bits of logical address designate page number & n-low order bits designate page offset.



8a) System consists of five jobs (J1, J2, J3, J4, J5) & three resources R1, R2, R3. Resource type R1 has 10 instances, R2 has 5 instances & R3 has 7 instances. The following snapshot of the system has been taken.

Jobs	Allocation			Maximum			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
J1	0	1	0	7	5	3	3	3	2
J2	2	0	0	3	2	2			
J3	3	0	2	9	0	2			
J4	2	1	1	2	2	2			
J5	0	0	2	4	3	3			

Find need matrix and calculate the safe sequence by using Bankers algorithm. mention the above system is safe or not. Ans

→ Contents of need matrix are

Process	Need		
	R1	R2	R3
J1	7	4	3
J2	1	2	2
J3	6	0	0
J4	0	1	1
J5	4	3	1

Step1 : for Job J1.

Need <= available , 7,4,3 <= 3,3,2 condition is false

Step2 : for job J2

Need <= available, 1 2 2 <= 3 3 2 condition True

available = available + allocation

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

Step3 : for job J3

Need <= available, condition is false.

Step4 : for process P4,

need <= available , condition is true.

available = available + allocation

7,4,3 = 5,3,2 + 2,1,1.

Step5 : for job J5

need <= available, condition is true

available = available + allocation

7,4,5 = 7,4,3 + 0,0,2.

Step 6 : for job J1

need <= available, condition is true

available = available + allocation

7,5,5 = 7,4,5 + 0,1,0.

Step 7 : for job J3

need <= available, condition is true

available = available + allocation

10,5,7 = 7,5,5 + 3,0,2

sequence J2, J4, J5, J1 & J3 are safe.

8b. Given the memory partitions of 200k, 700k, 500k, 300k, 100k, 400k. Apply first fit, best fit & worst fit to place 315k, 427k, 250k, 550k. Which algorithm makes more efficient use of memory. [4m]

→ First fit

315k is put in 700k

427k is put in 500k

200k is put in 300k

550k must wait

Best Fit

315k is put in 400k
427k is put in 500k
250k is put in 300k
550k is put in 700k.

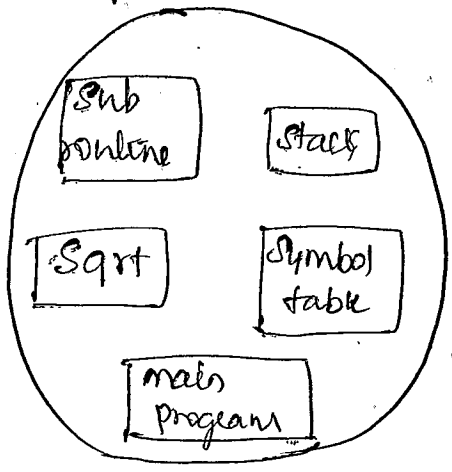
Worst Fit

315k is put in 700k
427k is put in 500k
250k is put in 400k
550k must wait

Best fit algorithm provides most efficient use of memory.

8c. What is segmentation? Explain basic method of segmentation with example. 8M

→ Segmentation is a memory management scheme that supports this user view of memory. Logical address space is collection of segments. Each segment has a name & length.



Addresses specifies both segment name & offset within the segment. Therefore user specifies each address by two quantities - a segment name & an offset.

For simplicity segments are numbered & are referred by segment number rather than by a segment name

logical address consists of two tuples:

<segment-number, offset>

Hardware - Basic method

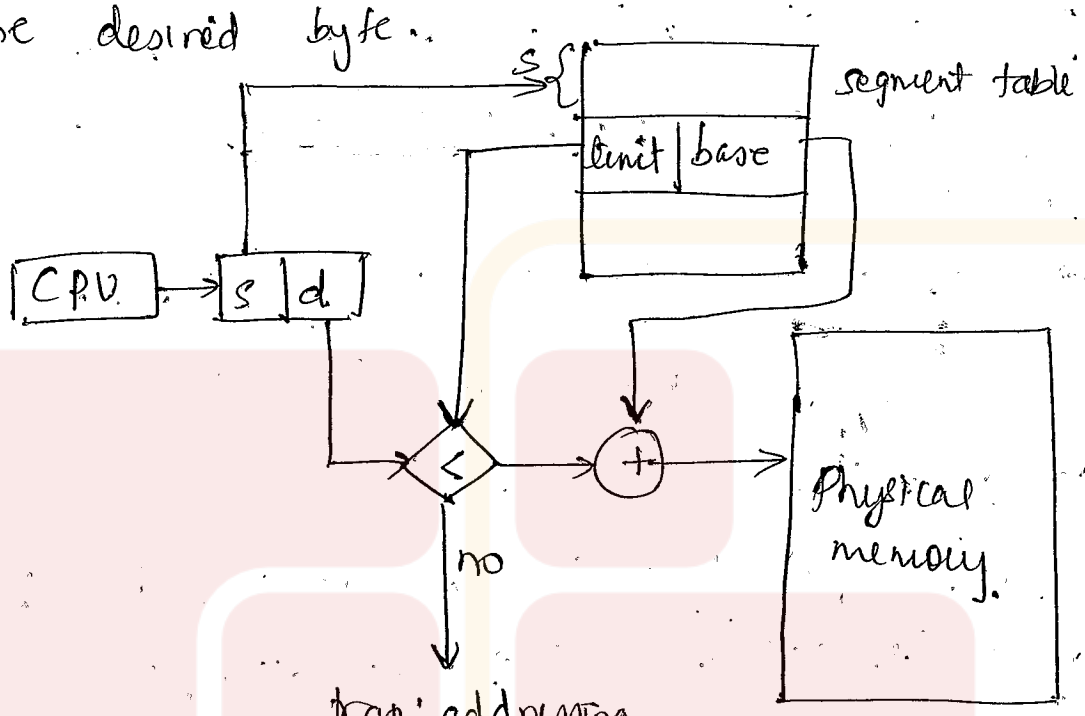
User refer to objects in program by 2D address, actual physical memory is one-dimensional sequence of bytes

- To map 2D user defined addresses into 1D physical address - segment table is used.
- Each entry of segment table has segment base & segment limit.

- segment base contains starting physical addresses where segment resides in memory, whereas the segment limit specifies length of the segment.
- logical address consists of 2 parts - segment number S & offset d. segment number is used as an index into the segment table.

offset d of logical addres must be between 0 & segment limit. If it is not, trap to O.S.

- If offset is legal it is added to the segment base to produce address in physical memory of the desired byte.



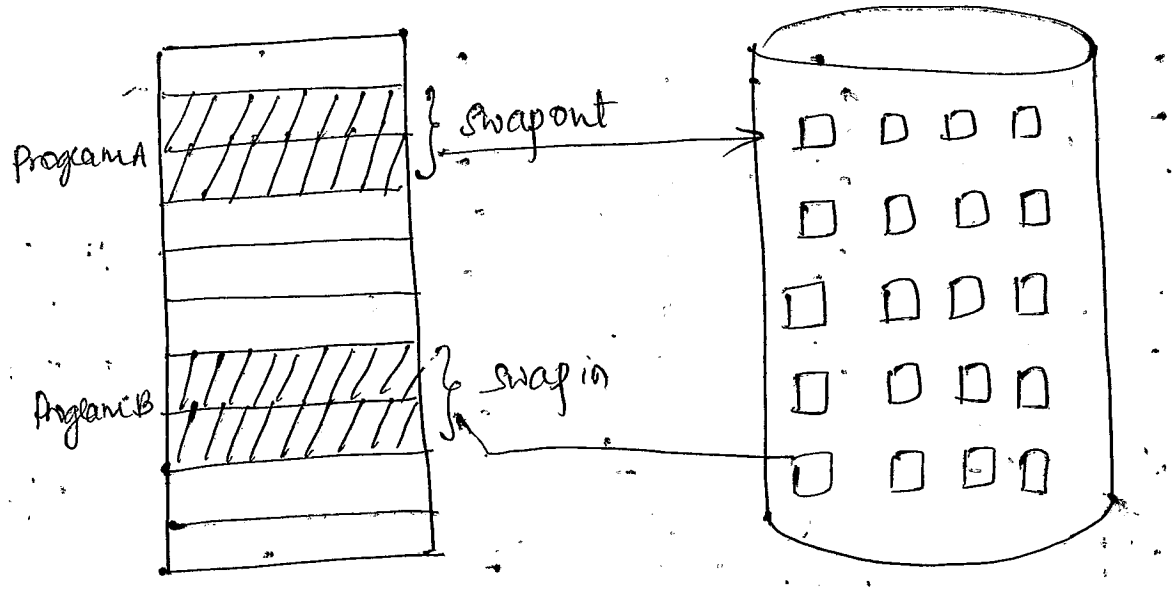
Module - 5

9a Explain demand paging in details. 7M.

⇒ Demand paging is paging system with swapper. Process reside in secondary memory. To execute a process, it must be swapped in to memory. Instead of swapping entire process into memory lazy swapper is used.

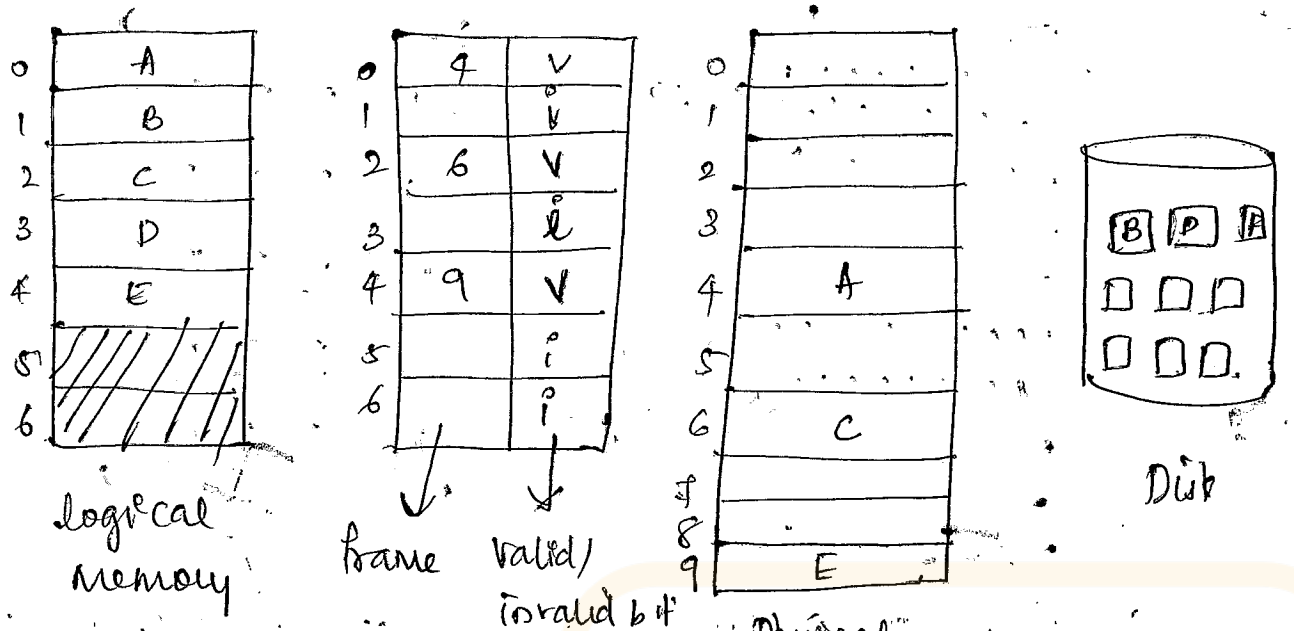
lazy swapper never swaps page in to memory unless page is needed.

Swapper manipulates entire processes, whereas pager is concerned with individual pages of a process



- Pager is used instead of swapper in demand paging when a process is to be swapped in, pager guesses which pages will be used before process is swapped out.
- Instead of swapping in whole process, the pager brings only those necessary pages into memory. Thus decreases swap time & amt of physical memory needed.
- Valid & invalid bits are used to distinguish between pages in memory & pages on the disk.
  - i) When bit is set to valid - indicates page is both legal & is in memory.
  - ii) When bit is set to invalid - indicates page is either valid or invalid but currently on disk.

  Page table entry for page that is brought into memory is set as usual, but page table entry for page that is not currently in memory is marked invalid or contains address of page on disk.



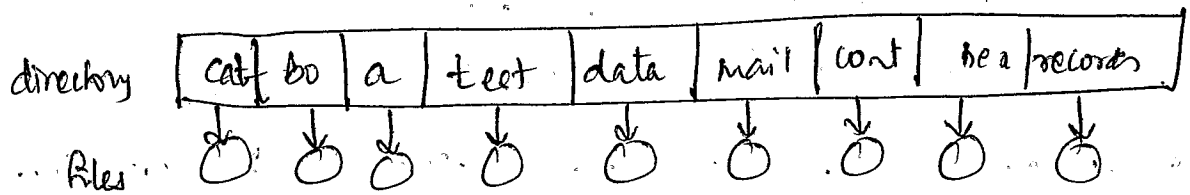
9b Explain various directory structures briefly. [6m]

⇒ Various directory structures are

i) Single-level directory
ii) Two-level directory
iii) Tree-structured directory
iv) Acyclic-Graph directories
v) General graph directories

i) Single-level directory

It is the simplest structure. All files are contained in same directory, which is easy to support & understand

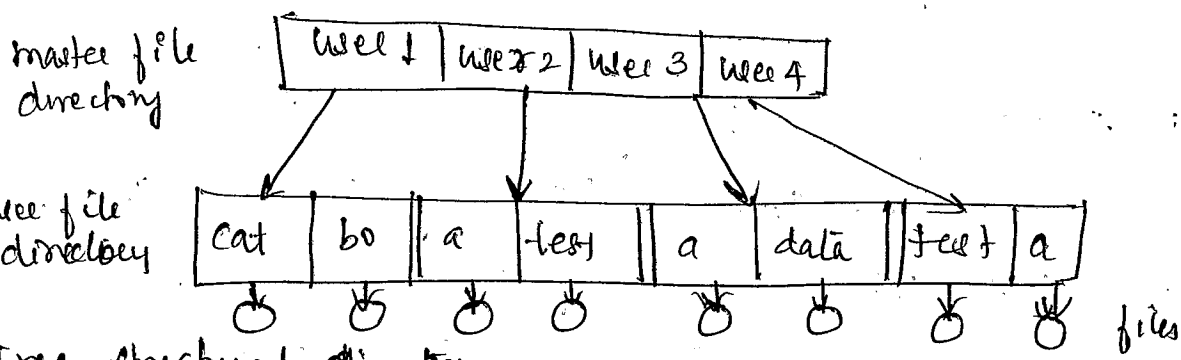


ii) Two-level directory

- Here, each user has his own user file directory (UFD)

The UFDs have similar structure but each lists only

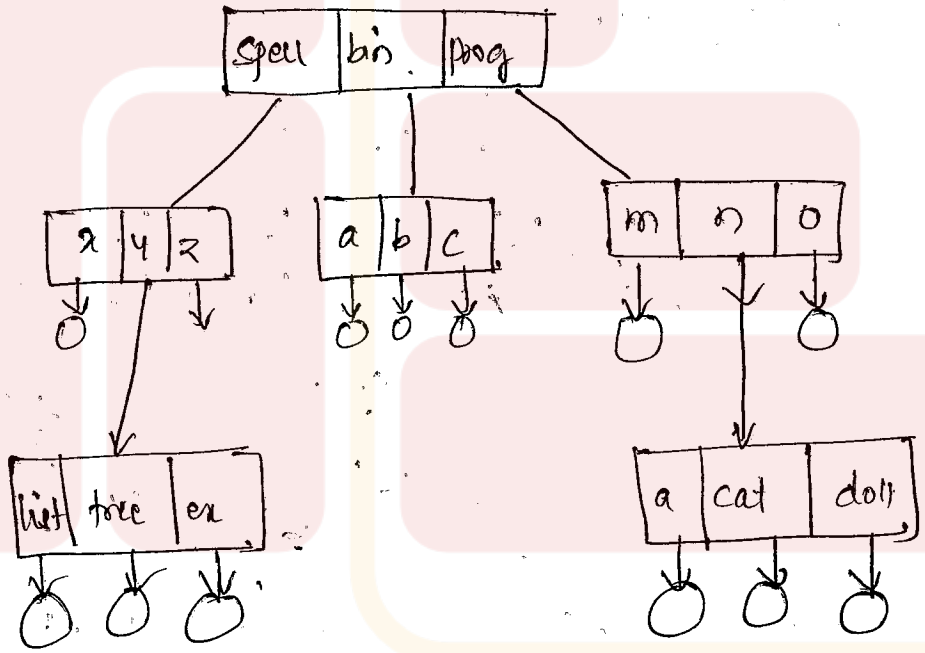
the files of a single user



iii) Tree structured directory

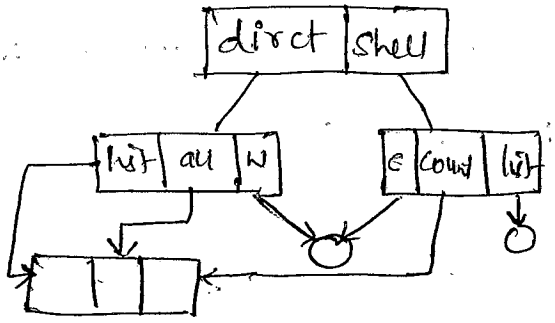
The tree has a root directory, & every file in the system has a unique path name

- directory contain a set of files or subdirectories.
- System calls are used to create & delete directories



iv) Acyclic graph directories

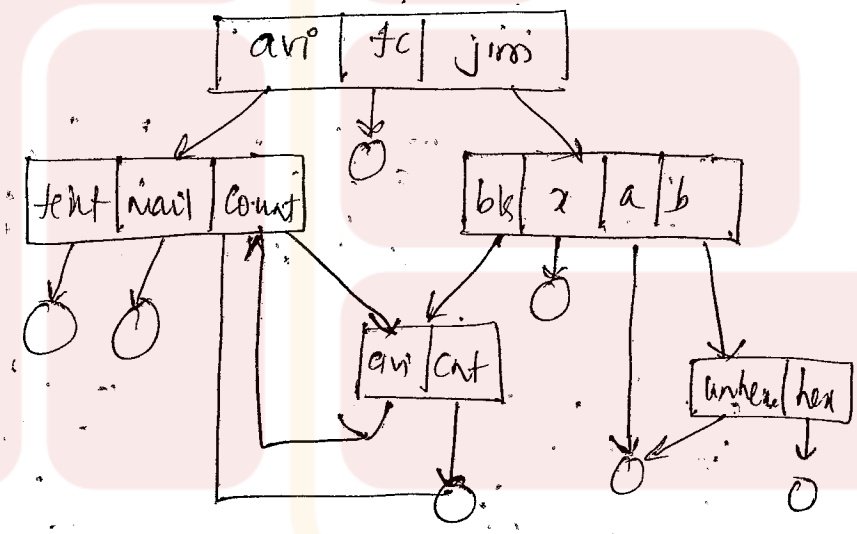
Acyclic graph is a graph with no cycle, allows directories to share subdirectories & files.



Acyclic graph structure is more flexible than a simple tree structure, but is also more complex.

v) General graph directory

- A serious problem with acyclic graph is ensuring that there are no cycles.
- If we start with a two-level directory & allow users to create subdirectories, a tree-structured directory results.
- It should be fairly easy to see that simply adding new files & subdirectories to an existing tree structured directory preserves the tree structured nature



Q: Explain the steps in handling page faults with a neat diagram. 7M

→ The steps in handling page faults are

1) We check an internal table for this process to determine whether the reference was a valid or an invalid memory access

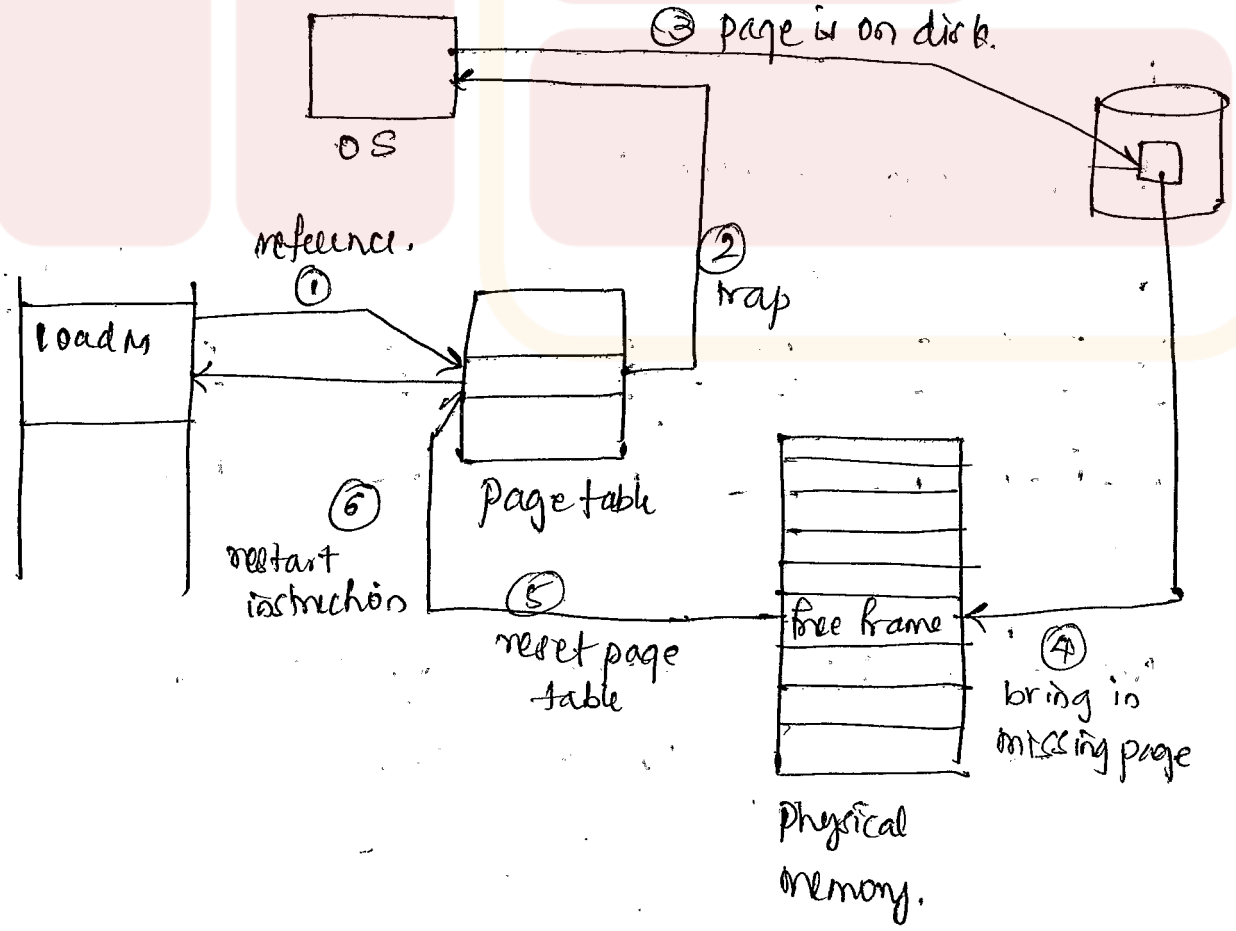
2) If reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3) We find a free frame

4) We schedule a disk operation to read the desired page into the newly allocated frame.

5) When the disk read is complete, we modify the internal table kept with the process & the page table to indicate that the page is now in memory.

6) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in

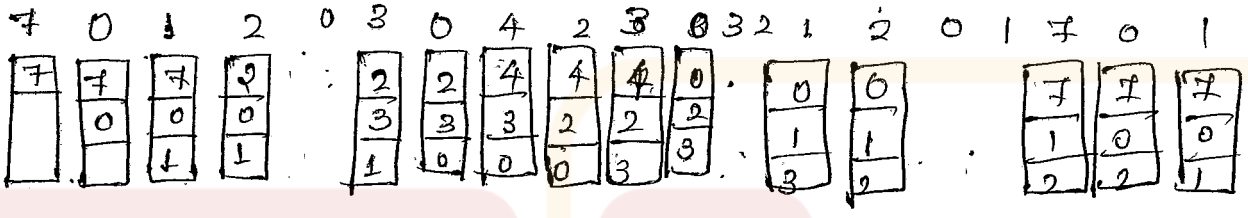


10a) Consider the following page reference string:

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

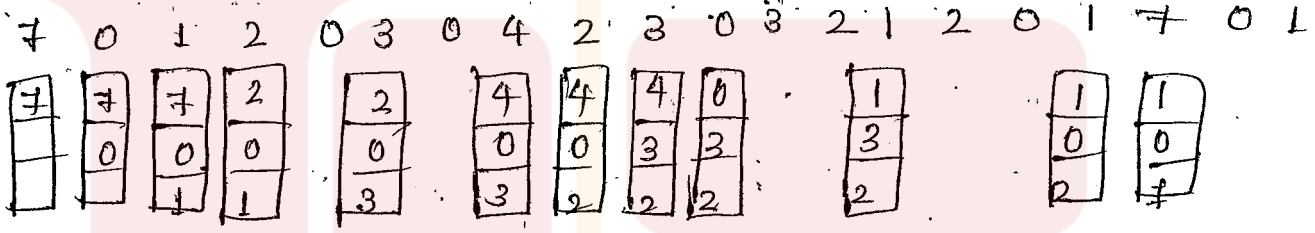
How many faults would occur for LRU, optimal & FIFO page replacement algorithm. Assume 3 frames. Which of the above algorithm is efficient

→ FIFO page replacement



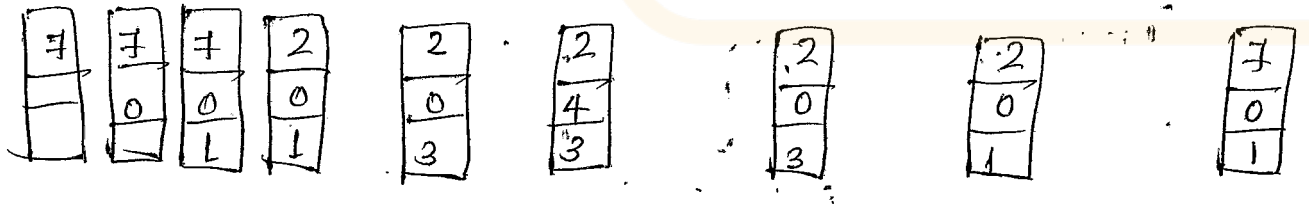
no. of faults = 15

LRU page replacement



no. of faults = 12

Optimal page replacement



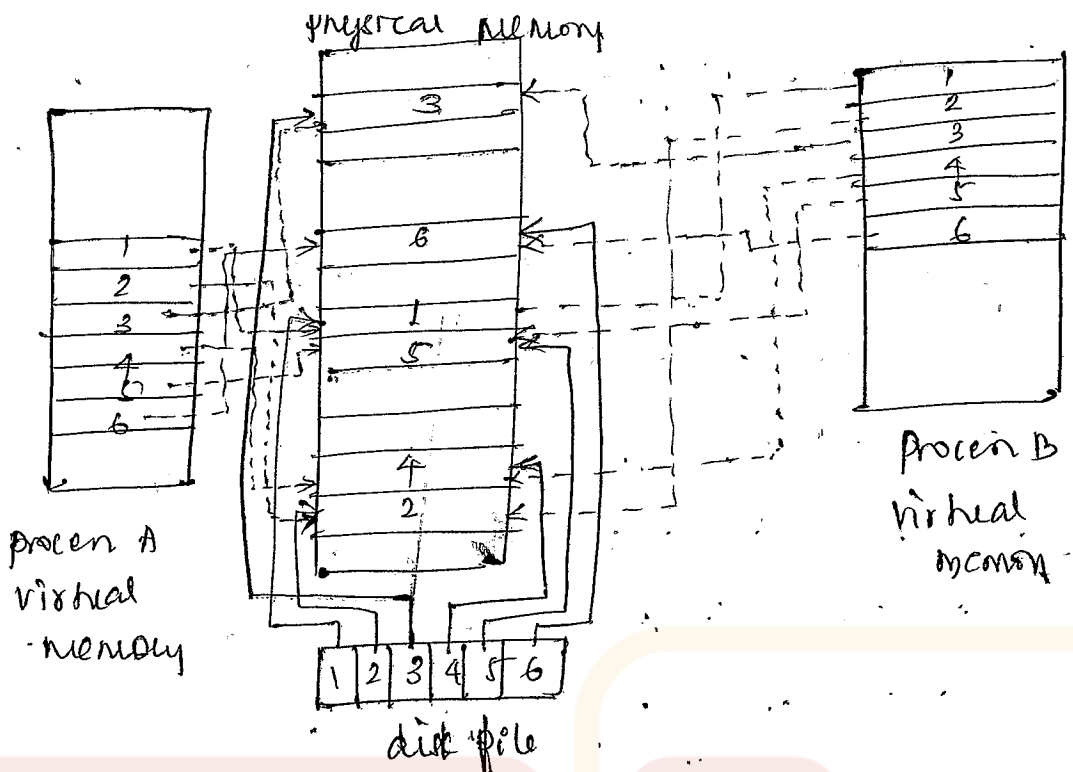
No. of faults = 9

Optimal page replacement algorithm is more efficient because no. of faults are less.

10b) Explain memory mapping of files with a neat diagram. [6M]

→ Memory mapping a file is accomplished by mapping a disk block to a page in memory.

- Initial access to the file proceeds through ordinary paging, resulting in a page fault.
- However, a page-sized portion of the file is read from the file system into a physical page.
- Subsequent reads & writes to the file are handled as routine memory accesses, thereby simplifying file access & usage by allowing the system to manipulate file through memory rather than incurring the overhead of using the read() & write() system calls
- Some systems may choose to update the physical file when the OS periodically checks whether the page in memory has been modified.
- When the file is closed, all the memory-mapped data are written back to disk & removed from the virtual memory of the process.
- Some OS provide memory mapping only through a specific system call & use the std system calls to perform all other file I/O.



10) What is thrashing ? How it can be controlled [6 M.]

→ i) If number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, then suspend that process execution.

ii) We should then page out its remaining pages, freeing all its allocated frames. This provision introduces swap-in & swap-out level of intermediate cpu scheduling.

2) If any process does not enough frames, it will quickly page fault.

At this point it must replace some pages.

Since these pages are in active use, it must replace a page that will be needed again

Consequently it faults again & again.

This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

- To prevent thrashing, working set strategy is used.

- This technique starts by looking at how many frames a process is actually using
- This approach defines locality model of process execution.
- locality model states that, as a process executes it moves from locality to locality.
- locality is set of pages that are actively used together.
- Program is composed of several different localities which may overlap.

Example: If a process under execution calls a subroutine, it defines a new locality where memory references are made to the instructions of subroutine, its local variable & subset of global variables.

- When subroutine is called, process leaves the locality, process can return to this locality later.
- localities are defined by program structure & its data structure.

Karnataka Law Society's

Vishwanathrao Deshpande Institute of Technology,

Haliyal.

Department of Computer Science & Engg

Sample Question Paper

Sub: Introduction to operating system

Subcode : 18CS654

Module 1

1a Define operating system. What are multiprocessor systems. Explain its types and their three main advantages. [8M]

1b. What are microkernels? point out their advantages. [6M]

1c. Explain VMWare architecture with a neat digram. [6M]

OR

2a List and explain the services provided by OS for the user & efficient operation of system [8M]

2b. What are system calls. Explain their types [6M]

2c. Explain the dual mode of operation [6M]

Module 2

3a Define proces. Explain different states of a procen with a neat diagram. [6M]

3b. Explain Scheduling queues with a neat diagram. Illustrate different queues with queuing diagram. [8M]

3c Compare & contrast shortterm, long term and medium term schedulers. [6M]

OR

4a Describe the implementation of interprocess communication using shared memory and message passing. [8M]

4b. Discuss the threading issues in multithreaded program. [6M]

4c Write short notes on windows XP thread [6M]

Module-3

5a Calculate the average waiting time and average turn around time by drawing Gantt chart using FCFS, SRTF, Round Robin (Quantum = 2 ms) and Priority scheduling algorithm. [10M]

Process	Arrival Time	Burst Time
P1	0	9
P2	1	4
P3	2	9
P4	3	5

5b Define Monitor? Explain solution to dining philosophers problem using monitor. [10M]

OR

6a Define semaphore? Explain its usage and implementation. [8M]

6b Explain peterson solution for critical section problem. [6M]

6c. Explain synchronization in window XP. [6M]

Module 4

7a. Define deadlock. Explain 4 necessary conditions that lead to deadlock. [6M]

7b. Determine whether the following system is in safe state using banker's algorithm

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

[8M]

If a request for P1 arrives for (1,0,2), can the request be granted immediately?

7c. What is resource allocation graph (RAG)? Explain how RAG is useful in determining deadlock, illustrate with example? [6M]

OR

8a Define paging. Explain paging hardware with a neat diagram. [6M]

8b Write short notes on

i) External & Internal fragmentation [6M]

ii) Dynamic loading & linking

8c. What is TLB? Explain TLB in detail with simple paging system with a neat diagram. [8M]

Module 5

9a. Consider the following page reference string

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

How many page faults would occur in the following algorithm [8M]

i) LRU ii) FIFO iii) optimal

9b Explain steps in handling page faults with a neat diagram. [6M]

9c. What is thrashing? How it can be controlled? [6M]

OR

10a Explain demand paging in detail. [7M]

10b. Explain briefly various operations performed on files. [7M]

10c Explain various access methods on files. [6M]

Karnataka Law Society's

Vishwanathrao Deshpande Institute of Technology, Haliyal

Department of Computer Science & Engg

Sub: Introduction to operating system

Subcode : 18CS654

Module 1

1a Define operating System. What are multiprocessor Systems? Explain its types and their three main advantages. [8M]

Solution:

An operating system is a program that manages the computer hardware. It also provides a basis for application program & acts as an intermediary between the computer user and the computer hardware.

Multiprocessor system have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory and peripheral devices.

Types

i) Symmetric multiprocessing - here each processor performs all tasks within the OS. SMP means that all processors are peers, no master-slave relationship exists between processors.

ii) Asymmetric multiprocessing - each processor is assigned a specific task. A master processor controls the

system, the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship.

Advantages

i) Increased throughput: By increasing the number of processes, we expect to get more work done in less time

ii) Economy of scale: Multiprocessor system can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage and power supplies.

iii) Increased reliability: If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down.

1b. What are microkernels? Point out their advantages. [6M]

- In the mid 1980's, researchers at Carnegie Mellon university developed an operating system called Mach that modularized the kernel using the microkernel approach.

This method structures the operating system by removing the nonessential components from the kernel and implementing them as system and user-level programs.

The result is a smaller kernel.

The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.

- communication is provided by message passing

- If the client program wishes to access a file, it must interact with the file service.

The client program & service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.

One benefit of the microkernel approach is ease of extending the OS.

All new services are added to user space & consequently do not require modification of the kernel.

When the kernel does have to be modified, the changes tend to be fewer, because the microkernel is a smaller kernel.

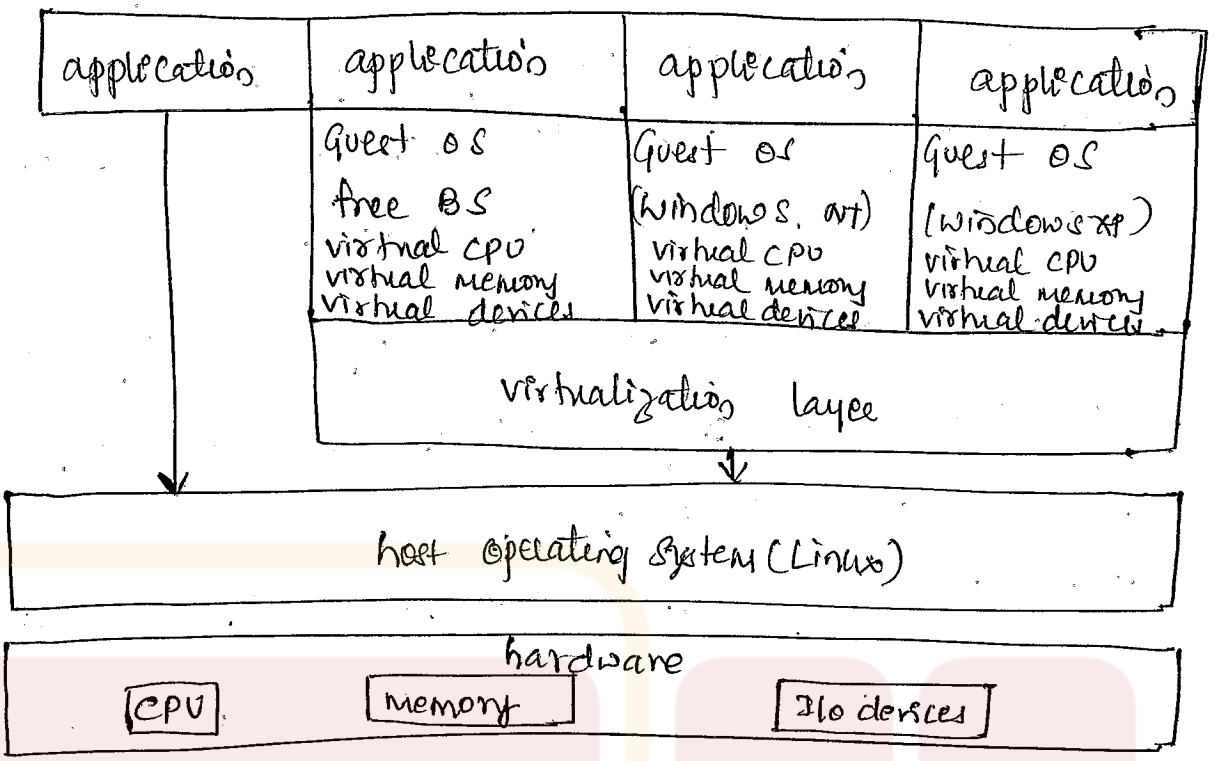
The resulting OS is easier to port from one h/w to another.

- The microkernel also provides more security and reliability, since most services are running as user - rather than kernel - process.

- If a service fails, the rest of the operating system remains untouched.

ic Explain vmware architecture with a neat diagram. [6M]

→



- vmware is a popular commercial application that abstract Intel 80x86 hardware into isolated virtual machines.
- vmware runs as an application on a host OS such as windows or Linux and allows this host system to concurrently run several different guest os as independent virtual machines.
- consider the following scenario: A developer has designed an application and would like to test it on linux, FreeBSD, windows NT, & windows xp.

  One option is for her to obtain four different computers, each running a copy of one of these operating systems.

  Another alternative is install each OS one by one & test the application. Such testing could be accomplished concurrently on the same physical computer using

vmware.

In the above figure, Linux is running as the host operating system. FreeBSD, Windows NT, and Windows XP are running as guest OS.

The virtualization layer is the heart of vmware, as it abstracts the physical hardware into isolated vms running as guest OS's.

Each vm has its own virtual CPU, memory, disk drives, network interfaces & so forth.

2a List and explain the services provided by OS for the user & efficient operation of system. [8M]

→ Set of operating system services helpful for user are

i) User Interface: Almost all OS have a user interface (UI). This interface can take several forms. One is a command-line interface (CLI) and another is graphical user interface (GUI).

ii) Program execution: The system must be able to load a program into memory & to run that program. The program must be able to end execution, either normally or abnormally.

iii) I/O operation: A running program may require I/O, which may involve file or an I/O device.

iv) File system manipulation: the file system is of particular interest. Programs need to read & write

files and directories.

v) Communications: There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer system tied together by a computer network.

vi) Error detection: The OS needs to be constantly aware of possible errors. Errors may occur in the CPU & memory hardware, in I/O devices & in the user program.

Set of services for efficient operation of system are

i) Resource allocation: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by OS.

ii) Accounting: We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting

iii) Protection & security: Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important

Q6 What are system calls. Explain their types. [6M]

→ System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C and C++,

Types of system calls.

System calls can be grouped roughly into five major categories:

i) Process control: A running program needs to be able to halt its execution either normally or abnormally. If a system call is made to terminate the currently running program abnormally or if the program runs into a problem and causes an error trap, a dump of memory is sometimes taken and an error message generated.

ii) File management: We need to be able to create & delete files. Either system call requires the name of the file and perhaps some of the files attributes. Once file is created, we need to open it and to use it. We may also read, write or reposition.

iii) Device management: A process may need several resources to execute - main memory, disk drives, access to files & so on. If the resources are available, they can be granted, & control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

iv) Information maintenance : Many system calls exist for the purpose of transferring information between user programs & the OS. For example most systems have a system call to return the current time & date.

v) Communication : There are two common models of interprocess communication : the message passing model & the shared-memory model. In the message-passing model, the communicating processes exchange messages between the processes either directly or indirectly through a common mailbox. In shared memory model, processes use shared memory create and shared memory attach system calls to create and gain access to regions of memory owned by other processes.

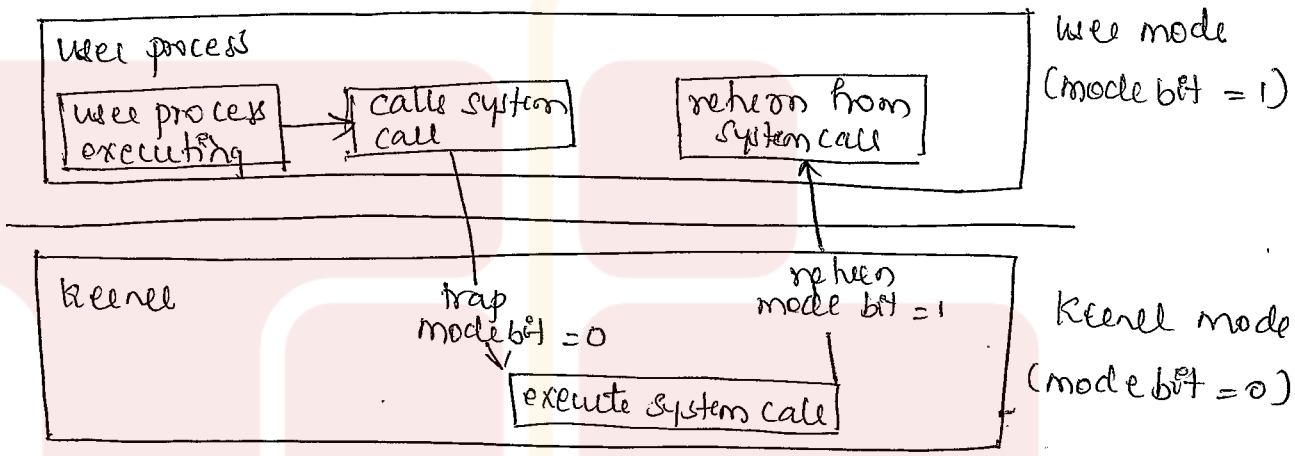
2c. Explain the dual mode of operation [6m]

→ In order to ensure the proper execution of the OS, we must be able to distinguish between the execution of OS code and user defined code. We need two separate modes of operation : user mode and kernel mode.

A bit called, the mode bit, is added to the hardware of the computer to indicate the current mode : kernel (0) or user (1).

With the mode bit, we are able to distinguish between a task that is executed on behalf of

the operating system and one that is executed on behalf of the user. When the computer system is executing on behalf of a user application, the system is in user mode. When a user application requests a service from the operating system, it must transition from user to kernel mode to fulfill the request. This is shown in figure below.



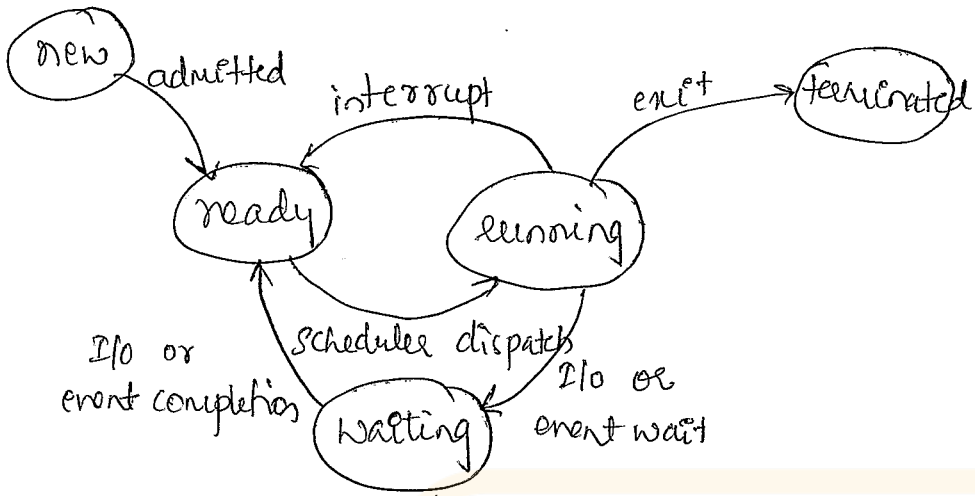
MODULE - II

3a. Define Process. Explain different states of a process with a neat deagram. [6M]

→ A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor registers.

As a process executes, it changes states. The state of a process is defined in part by the current activity of that process.

Each procen may be in one of the following states:



new: The procen is being created

Running: Instructions are being executed.

Waiting: The procen is waiting for some event to occur (such as an I/O completion or reception of a signal)

Ready: The procen is waiting to be assigned to a processor.

Terminated: The procen has finished execution.

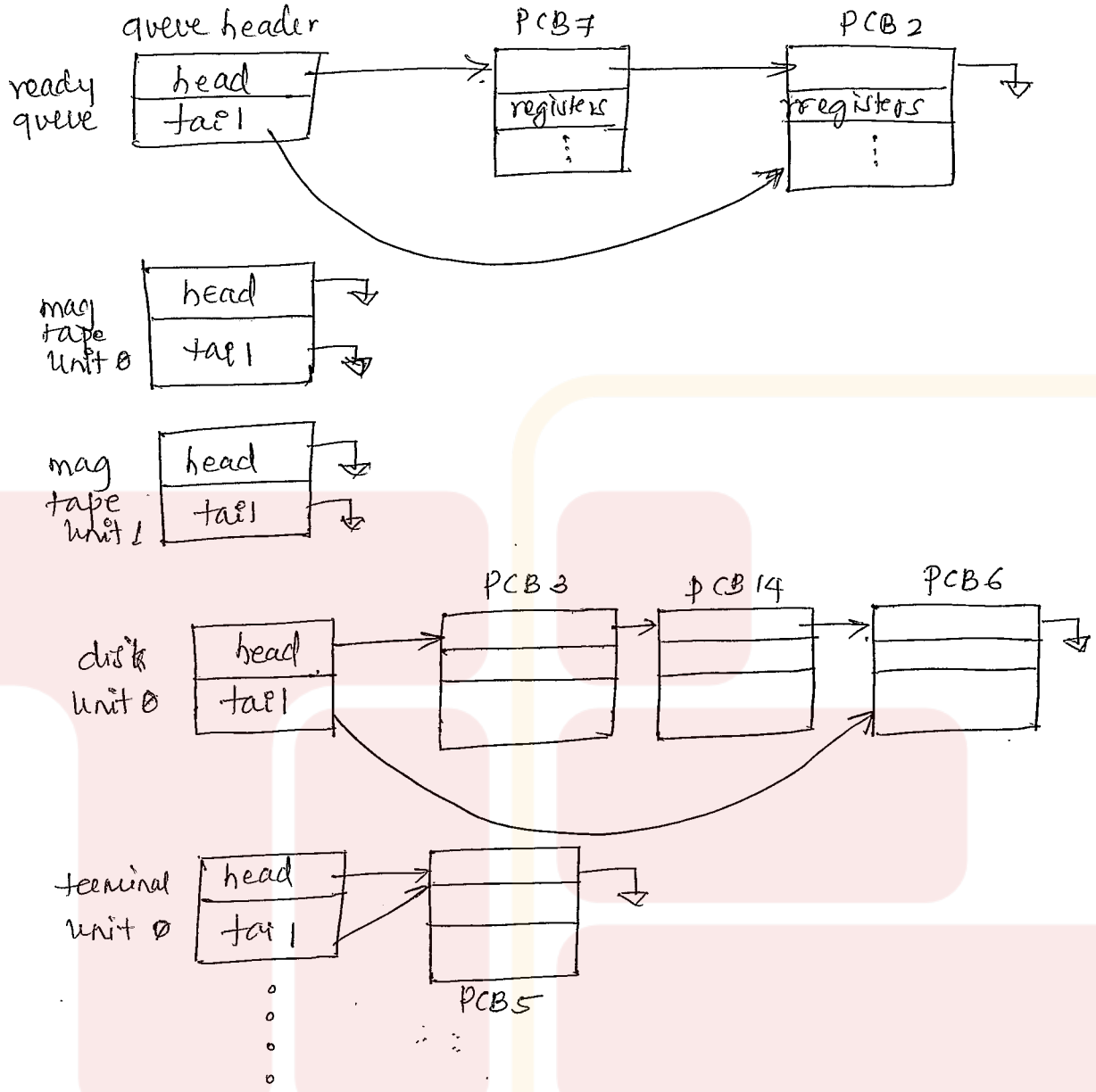
3b) Explain Scheduling queues with a neat diagram. Illustrate different queues with queuing diagram. [8M]

→ The objective of multiprogramming is to have procen running at all times, to maximize CPU utilization.

The objective of time sharing is to switch the CPU among procenes so frequently that users can interact with each program while it is running.

To meet this objective, procen scheduler selects

an available process for program execution on the CPU.

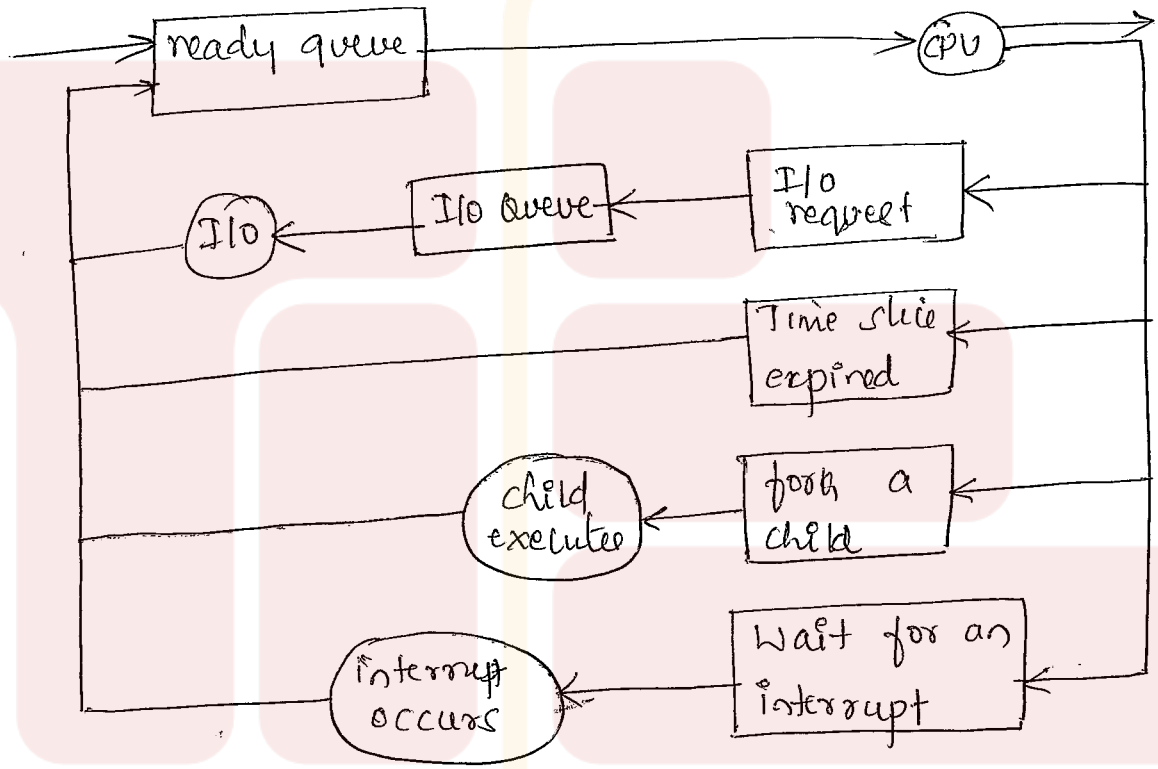


As process enters the system, they are put in a job queue, which consists of all processes in the system.

The processes that are residing in main memory & are ready & waiting to execute are kept on a list called the ready queue.

A ready queue is generally stored as a linked list. A ready-queue header contains pointers to the first and final PCB's in the list.

The system also includes other queues. When a process is allocated the CPU, it executes for a while & eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. The list of processes waiting for a particular I/O device is called a device queue. Each device has its own device queue.



A new process is initially put in the ready queue. It waits there until it is selected for execution or is dispatched. Once the process is allocated the CPU & is executing, one of several events could occur:

- The process could issue an I/O request & then be placed in an I/O queue.
- The process could create new subprocess & wait for the subprocess's termination

- The process could be removed forcibly from the CPU, as a result of an interrupt, & be put back in the ready queue.

3c Compare & contrast short term, long term and medium term schedulers. [6M]

→ Long-term scheduler

selects processes from job pools and loads them into memory for execution.

- It executes much less frequently.

Short-term scheduler

- selects processes among processes that are ready to execute and allocates the CPU to one of them.

- It selects a new process frequently

Medium term scheduler

medium term scheduler uses intermediate level of scheduling.

- key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove process from memory and thus reduce the degree of multiprogramming. Later, the process can be reintroduced in to the memory & its execution can be continued from where it left off.

4a. Describe the implementation of interprocess communication using shared memory and message passing. [8M]

→ Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.

- Typically a shared memory region resides in the address space of the process creating the shared memory segment.

→ Other processes that wish to communicate using this shared memory segment must attach it to their address space.

We can illustrate shared memory using producer-consumer problem

To allow producer & consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer & emptied by the consumer.

- This buffer will reside in a region of memory that is shared by the producer & consumer processes.
- A producer can produce one item while the consumer is consuming another item.
- The producer & consumer must be synchronized so that consumer does not try to consume an item that has not yet been produced.

Two types of buffers can be used

a) unbounded buffer places no practical limit on the size of the buffer.

The consumer may have to wait for new items, but the producer can always produce new items.

b) <u>bounded buffer</u> assumes a fixed buffer size.

The consumer must wait if the buffer is empty & the producer must wait if the buffer is full.

The following variables reside in a region of memory shared by the producer & consumer processes.

```
#define BUFFER_SIZE 10
typedef struct
{
...
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

The shared buffer is implemented as a circular array with 2 logical pointers in & out.

The buffer is empty when in == out & the buffer is full when ((in+1)% BUFFER_SIZE) == out.

code for producer process is

```
item next produced;
while (true)
{
while (((in+1) % BUFFER_SIZE) == out);
buffer[in] = next produced;
in = (in+1) % BUFFER_SIZE;
}
```

The code for consumer process is

```
item next consumed;
```

```
while (true);
{
    while (in == out)
        ;
    nextConsumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
}
```

ii) Message passing provides a mechanism to allow processes to communicate & to synchronize their actions without sharing the same address space

A message-passing facility provides atleast two operations send (message) & receive (message).

Messages sent by a process can be either fixed or variable size.

If process P & Q want to communicate, they must send messages to & receive messages from each other, a communication link must exist between them.

Here are several methods for logically implementing a link & the send() / receive() operations.

i) Direct or indirect communication
ii) Synchronous or asynchronous communication
iii) Automatic or explicit buffering.

a) Under direct communication, each process must explicitly specify the recepient or sender name

In this scheme, the send() & receive() primitives are defined as

send (P, message)

receive (Q, message).

A communication link has the following property.

i) a link is established automatically between every pair of processes that want to communicate

ii) A link is associated with exactly two processes

iii) There exist exactly one link between each pair of processes.

b) In indirect communication, the messages are sent to and received from mailboxes or ports. A mailbox can be viewed abstractly as an object into which messages can be removed. Each mailbox has a unique identification.

TWO processes can communicate only if the process have a shared mailbox, the send() & receive() primitives are defined as follows

send (A, message)

receive (A, message)

A communication link has following properties

i) a link is established between a pair of processes only if both members of a pair have a shared mailbox.

ii) a link may be associated with more than 2 processes.

iii) Between each pair of communicating processes there may be a number of different links, with each link corresponding to one mailbox.

c) Synchronization

Communication between process takes place through calls to send() & receive() primitives. Message passing may be either blocking or nonblocking, also known as Synchronous or asynchronous.

Blocking send - Sending process is blocked until the message is received by the receiving process or by the mailbox.

Non blocking send - Sender sends the message & continue.

Non blocking receive - receiver receives a valid message or NULL

Blocking receive - receiver is blocked until a message is available.

d) Buffering : Queue of messages are attached to link & are implemented in three ways

i) Zero capacity - store maximum zero messages.
- Sender must wait for receiver.

ii) Bounded capacity - finite length of n messages.
- sender wait if link is full.

iii) Unbounded capacity - infinite length
- sender never wait

4b Discuss the threading issues in multithreaded program. [6M]

→ The threading issues in multithreaded program are : i) fork() & exec() system calls

ii) Cancellation

iii) Signal Handling

iv) Thread pools

v) Thread - specific data

vi) Scheduler activations

i) fork() & exec() System calls is used to create a separate, duplicate process. Some UNIX systems have chosen to have 2 versions of fork(), one that duplicates all threads & another that duplicates only the thread that invoked the fork() system call.

If a thread invokes the exec() system call, the program specified in the parameter to exec() will replace the entire process - including all threads.

ii) Cancellation : Thread cancellation is the task of terminating a thread before it has completed.

ex. If multiple threads are concurrently searching through a database & one thread returns the result, the remaining threads might be cancelled.

iii) Signal handling : A signal is used in UNIX systems to notify a process that a particular event has occurred. A signal may be received either synchronously or asynchronously, depending on the source of & the reason for the event being signaled.

iv) Thread pools :

The idea behind a thread pool is to create a number of threads at process startup & place them into a pool, where they sit & wait for work. When a server receives a request, it awakens a thread from this pool - if one is available & passes the request to service. Once the thread completes its service, it returns to pool & awaits more work. If pool contains no available thread, the server waits until one becomes free.

v) Thread-specific data : Threads belonging to a process share the data of the process. This sharing of data provides one of the benefits of multithreaded programming. In some circumstances, each thread might need its own copy of certain data. Such data is called thread-specific data.

vi) Scheduler activations : Many systems implementing either the many-to-many or two-level model place an intermediate data structure between the user & kernel level threads. One scheme for communication between between the user-thread library & the kernel is known as scheduler activations.

4c. Write short notes on Windows XP. thread [6M]

→ Windows XP implements the Win32 API. The Win32 API is the primary API for the family of Microsoft OS.

A Windows XP application runs as a separate process & each process may contain one or more threads.

The Win32 API for creating threads uses the one to one mapping, where each user-level thread maps to an associated kernel thread.

- Windows XP also provides support for a fiber library, which provides the functionality of the many to many model.
- By using the thread library, any thread belonging to a process can access the address space of the process.

The general components of a thread include:

- A thread ID uniquely identifying the thread.
- A register set representing the status of the processor.
- A user stack, employed when the thread is running in user mode & a kernel stack, employed when the thread is running in kernel mode.
- A private storage area used by various run-time libraries & dynamic link libraries (DLLs).

The register set, stacks & private storage area are known as the content of the thread.

The primary data structures of a thread include:

- ETHREAD - executive thread block
- KTHREAD - kernel thread block
- TEB - thread environment block.

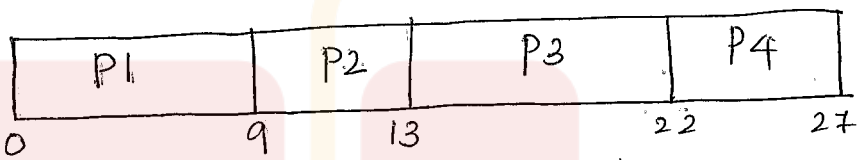
The key component of the ETHREAD include a pointer to the process to which the thread belongs & the address of the routine in which thread starts control.

- The ETHREAD also contains a pointer to the corresponding KTHREAD.
- The KTHREAD includes scheduling & synchronization information for the thread.
- The KTHREAD includes the kernel stack & a pointer to the TEB.
- TEB is a user-space data structure that is accessed when the thread is running in user mode.
- Among other fields, the TEB contains the thread identifier, a user-mode stack & an array for thread specific data.

5a. Calculate average waiting time and average turn around time by drawing Gantt chart using FCFS, SRTF, RR (quantum = 4ms) [10M]

process	Arrival Time	Burst time
P1	0	9
P2	1	4
P3	2	9
P4	3	5

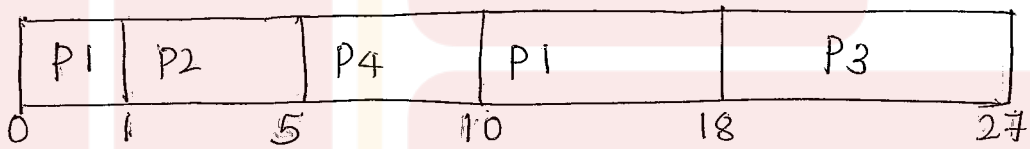
→ FCFS



Average waiting time = (0+9+13+22)/4 = 11 ms.

Average turn around time = [(9-0)+(13-1)+(22-2)+(27-3)]/4
= (9+12+20+24)/4 = 16.25 ms

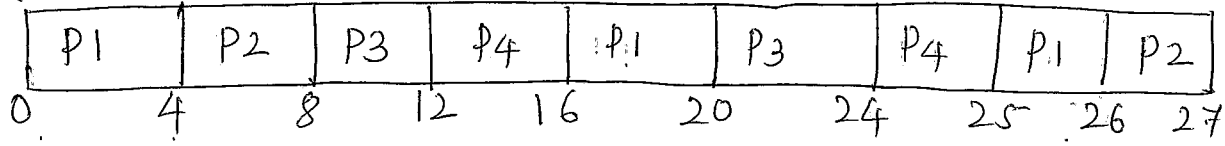
SRTF



Average waiting time = [(10-1)+(11-1)+(18-2)+(5-3)]/4
= [9+0+16+2)/4 = 6.75 ms

Average turn around time = [(18-0)+(5-1)+(27-2)+(10-3)]/4
= [18+4+25+7)/4 = 13.5 ms.

RR



Average waiting time = [(25-4-4)+(26-4-1)+(20-4-2)+(24-4-3)]/4
= (17+21+14+17)/4 = 17.25 ms

Average turn around time = [(26-0)+(27-1)+(24-2)+(25-3)]/4
= (26+26+22+22)/4 = 23.75 ms

5b define monitor? Explain solution to dining philosopher problem. [10M]

→ Monitor is a type of abstract data type, encapsulates private data with public methods to operate on the data.

A monitor type represents a set of programmer defined operations that provide mutual exclusion within the monitor.

The monitor type also contains the declaration of variables whose values define the state of an instance of that type, along with the bodies of procedures or function that operate on these variables.

```
Monitor monitor_name
{
    // shared variable declaration
    procedure P1()
    {
    }
    procedure P2()
    {
    }
    initialization_code()
    {
    }
}
```

Monitor solution to dining philosopher problem

To code this solution, we need to distinguish among 3 states in which we may find a philosopher

For this purpose, we introduce the following data structures

enum {thinking, hungry, eating} state[5];

- philosopher i can set the variable state[i] = eating only if her two neighbour are not eating.

(state[(i+4)%5] != eating) && (state[(i+1)%5] != eating)

We also need to declare

condition self[5];

where philosopher i can delay herself when she is hungry but is unable to obtain the chopsticks she needs.

- The distribution of the chopsticks is controlled by monitor dp, whose definition is given below.

```
monitor dp
{
    enum{THINKING, HUNGRY, EATING} state[5];
    condition self[5];
    void pickup(int i)
    {
        state[i] = THINKING;
        test(i);
        if (state[i] != EATING)
            self[i].wait();
    }
    void putdown(int i)
    {
        state[i] = THINKING;
        test((i+4)%5);
        test((i+1)%5);
    }
    void test(int i)
    {
```

```
    {
        if ((state[(i+4)%5] != EATING) &&
            (state[i] == HUNGRY) &&
            (state[(i+1)%5] != EATING))
        {
            state[i] = EATING;
            self[i].signal();
        }
    }
    initialization_code()
    {
        for(int i=0; i<5; i++)
            state[i] = THINKING;
    }
```

The philosopher i must invoke the operation pickup() & putdown in the following sequence

```
dp.pickup(i);
...
eat;
...
dp.putdown(i);
```

6a Define Semaphore. Explain its usage and implementation. [8M]

→ Semaphore S is an integer variable that apart from initialization, is accessed only through standard atomic operation: wait() & signal()

The definition of wait() is as follows

```
wait(S)
{
    while(S<=0)
        ;
    S--;
}
```

The definition of signal is as follows

```
signal(s)
{
  s++;
}
```

We can also use semaphore to solve various synchronization problem.

Consider two concurrently running process : P1 with a statement S1 & P2 with a statement S2. Suppose we require S2 be executed only after S1 has completed.

```
S1;
signal(synch);        // process P1

wait(synch);          // process P2
S2
```

wait() semaphore operation can be defined as

```
wait(semaphore *S)
{
  S->value--;
  if(S->value <0)
  {
    add this process to S->list;
    block();
  }
}
```

signal() semaphore operation can be defined as

```
Signal(semaphore *S)
{
  S->value++;
  if(S->value <=0)
  {
    remove a process P from S->list;
    wakeup(P);
  }
}
```

6b. Explain peterson's solution for critical section Problem. [6M]

→ Peterson solution is restricted to two processes that alternate execution between their critical sections and remainder sections.

The processes are numbered P0 & P1.

Peterson solution requires two data items to be shared between the two processes

```
int turn;
boolean flag[2];
```

The variable turn indicates whose turn it is to enter its critical section.

i.e. if turn == i, then process Pi is allowed to execute in its critical section.

The flag array is used to indicate if a process is ready to enter its critical section.

To enter the critical section, process Pi first sets flag[i] to be true & then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section, it can do so.

```
do{
    flag[i] = TRUE;
    turn = j;
    while (flag[j] && turn == j);
        Critical section
```

flag[i] = FALSE;

remainder section

} while (TRUE);

To prove mutual exclusion, we note that Pi enters its critical section only if either flag[j] == false & turn == i.

To prove property 2 & 3, Pi can be prevented from entering the CS only if it is stuck in the while loop with the condition flag[j] == true & turn == j

If Pj is not ready to enter the critical section, then flag[j] = false & Pi can enter its CS.

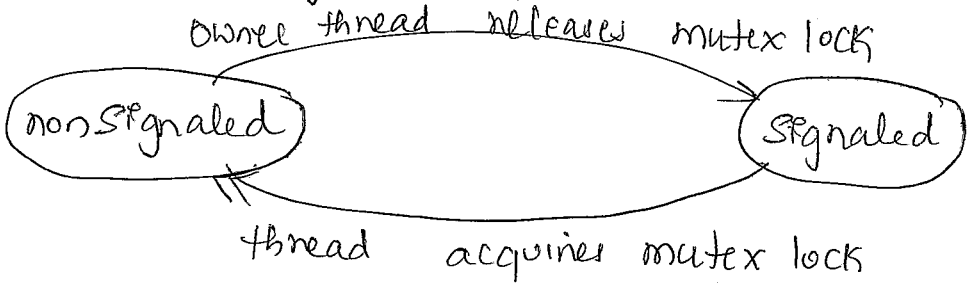
If Pj has set flag[j] to true & is also executing in its while statement, then either turn == i or turn == j. If turn == i, then Pi will enter CS. If turn == j, then Pj will enter CS.

6c Explain synchronization in Windows XP. [6M]

⇒ The Windows XP OS is a multithreaded kernel that provides support for real time applications & multiple processors.

- When the windows XP kernel accesses a global resource on a uniprocessor system, it temporarily masks interrupts for all interrupt handlers that may also access the global resource.
- On a multiprocessor system, windows xp protects access to global resources using spinlocks.

- For thread synchronization outside the kernel, Windows XP provides dispatcher objects
- Using a dispatcher object, threads synchronize according to several different mechanisms, including mutexes, semaphores, events & timers.
- The system protects shared data by requiring a thread to gain ownership of a mutex to access the data & to release ownership when it is finished.
- Events are similar to condition variables, that is, that is they may notify a waiting thread when a desired condition occurs.
- Timers are used to notify one or more than one thread that a specified amount of time has expired.
- Dispatcher objects may be either in a signaled state or nonsignaled state.
- A signaled state indicates that an object is available & a thread will not block when acquiring the object.
- A nonsignaled state indicates that an object is not available & a thread will block when attempting to acquire a object



Module 4

1a. Define deadlock. Explain 4 necessary conditions that lead to deadlocks? [6M]

→ In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources & if the resources are not available at that time, the process enters a waiting state. Sometimes a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called deadlock.

Necessary Conditions

i) Mutual Exclusion: At least one resource must be held in a non shareable mode, that is only one process at a time can use the resource. If another process requests the resource, the requesting process may be delayed until the resource has been released.

ii) Hold & wait: A process must be holding atleast one resource & waiting to acquire additional resources that are currently being held by other processes.

iii) No preemption: Resources cannot be preempted, that is a resource can be released only voluntarily by the process holding it, after

that process has completed its task.

iv) Circular wait : A set {P0, P1, ... Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ... Pn-1 is waiting for a resource held by Pn, Pn is waiting for a resource held by P0.

7b. Determine whether the following system is in safe state using bankers algorithm.

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

If a request for P1 arrives for (1,0,2), can the request be granted immediately?

→ need matrix can be computed as

Process	Need A B C
P0	7 4 3
P1	1 2 2
P2	6 0 0
P3	0 1 1
P4	4 3 1

Step1 : for process P0

Need <= available 7, 4, 3 <= 3, 3, 2, False

Step2 : for process P1

Need <= available => 1, 2, 2 <= 3 3 2 , True.

available = available + allocation

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

Step3 :- for job P2

Need <= available ; 6, 0, 0 <= 5 3 2 false

Step4 :- for procen P3

Need <= available , 0, 1, 1 <= 5 3 2 True,

available = available + allocation

7, 4, 3 = (5, 3, 2) + (2, 1, 1)

Step5 : for procen P4.

Need <= available , 4 3 1 <= 7 4 3 True

available = available + allocation

7, 5, 5 = 7, 4, 3 + 0, 0, 2

Step6 : for procen P6

Need <= available , 7, 4, 3 <= 7 5 5 True

available = available + allocation

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

Step7 : for procen P2

Need <= available 6, 0, 0 <= 7 5 5 True

Available = available + allocation

10, 5, 7 = 7, 5, 5 + 3, 0, 2

procen sequence P1, P3, P4, P0, P2 are safe.

If process P1 requests one additional instance of resource type A & 2 instance of resource type C so request := (1, 0, 2)

We check request$_i$ ≤ Available i.e. $(1,0,2) \leq (3,3,2)$ which is true.

∴ The request can be granted.

7c. What is resource allocation graph (RAG)? Explain how RAG is useful in determining deadlocks, illustrate with example? [6M]

→ Deadlock can be described in terms of directed graph called system resource allocation graph. This graph consists of set of vertices V, & set of edges E.

- set of vertices V is partitioned into two different types of nodes, $P = \{P_1, P_2, \ldots P_n\}$ = set of processes in system

Set of all resource types in system $R = \{R_1, R_2, \ldots R_n\}$

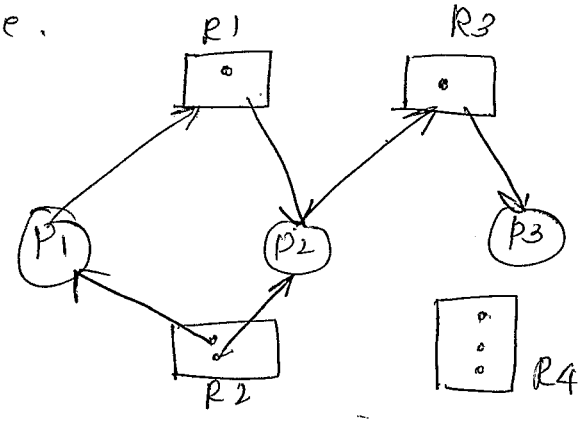
- A directed edge from P_i to resource R_j is denoted by $P_i \rightarrow R_j$ ⇒ P_i requested R_j & is currently waiting for R_j.

A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$ ⇒ instance of resource R_j is allocated to P_i.

- A directed edge $P_i \rightarrow R_j$ is called Request edge.

A directed edge $R_j \rightarrow P_i$ is called assignment edge.

- Process is represented as circle & resource as a square.



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

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

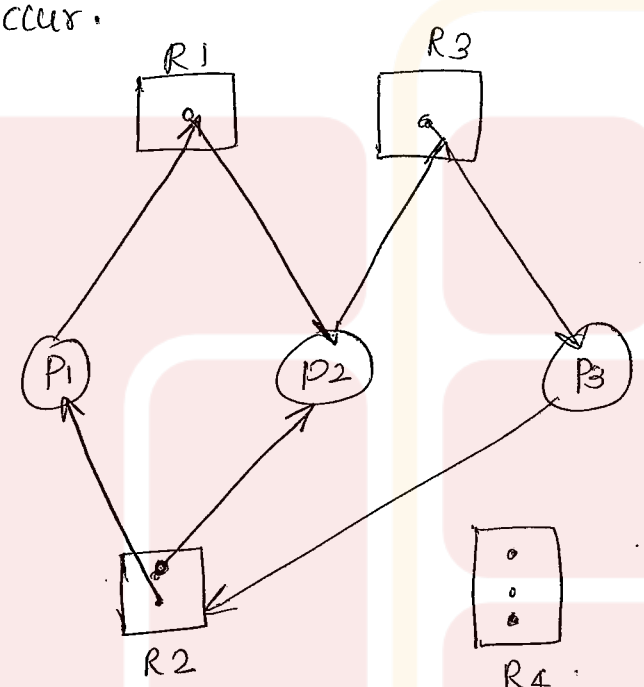
$E = \{P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow R_2, R_2 \rightarrow P_2, R_2 \rightarrow P_1, R_3 \rightarrow P_3\}$

P1 is holding an instance of R2 & is waiting for an instance of resource type R1

P2 is holding an instance of R1 & R2 & is waiting for an instance of resource type R3

P3 is holding an instance of R3

- If graph contains no cycle, then no process is deadlocked. If graph contains cycle then a deadlock may occur.



In the above graph, 2 cycles exist, they are

$P1 \rightarrow R1 \rightarrow P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow R1$

$P2 \rightarrow R3 \rightarrow P3 \rightarrow R2 \rightarrow P2$

∴ deadlock occurs i.e P1, P2 & P3 are deadlocked.

8a Define Paging. Explain paging hardware with a neat diagram [6M]

→ Paging is a memory management scheme that permits physical address space to be noncontiguous.

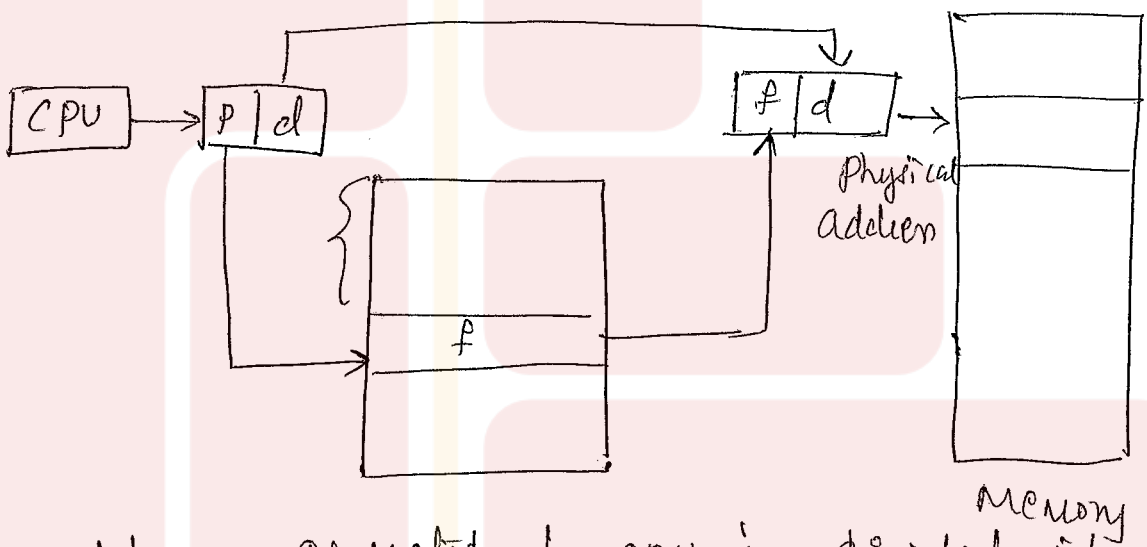
- Physical memory is broken into fixed sized blocks called frames.

dogical memory is broken into blocks of same size called pages.

When a process is to be executed, its pages are loaded into any available memory frames from the backing store.

- Backing store is divided into fixed sized blocks that are of same size as memory frames.

Hardware support for paging is illustrated in figure below



Every address generated by CPU is divided into two parts page number (P) & page offset (d).

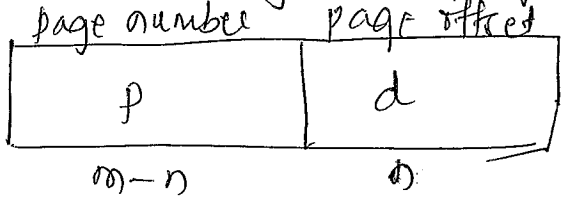
Page number is used as index into a page table.

Page table contains base address (f) of each page in physical memory.

This base address is combined with page offset to define physical memory address that is sent to memory unit.

- Page size is defined by hardware.

If size of logical address space is 2^m & page size is 2^n addressing unit, then higher order $m-n$ bits of logical address designate page number & n-low order bits designate page offset



8b. Write short note on [6M]

i) External & Internal Fragmentation

ii) Dynamic loading & Linking

→ i) External & Internal Fragmentation

- As processes are loaded and removed from memory, the free memory space is broken into small pieces
- External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into large number of small holes
- In worst case, we have block of free & wasted memory between every two processes. If all this memory were in one big free block, we would be able to run several more processes.
- Consider a multiple partition allocation scheme with a hole of 18,464 bytes. Suppose process request 18,462 bytes. If request is allocated, then we are left with 2 bytes.

for this purpose, Break physical memory into fixed-sized blocks & allocate memory on unit of block sizes

- With this memory allocated to process may be slightly larger than requested memory.
- The difference between these two numbers is internal fragmentation - memory that is internal to a partition but is not being used.

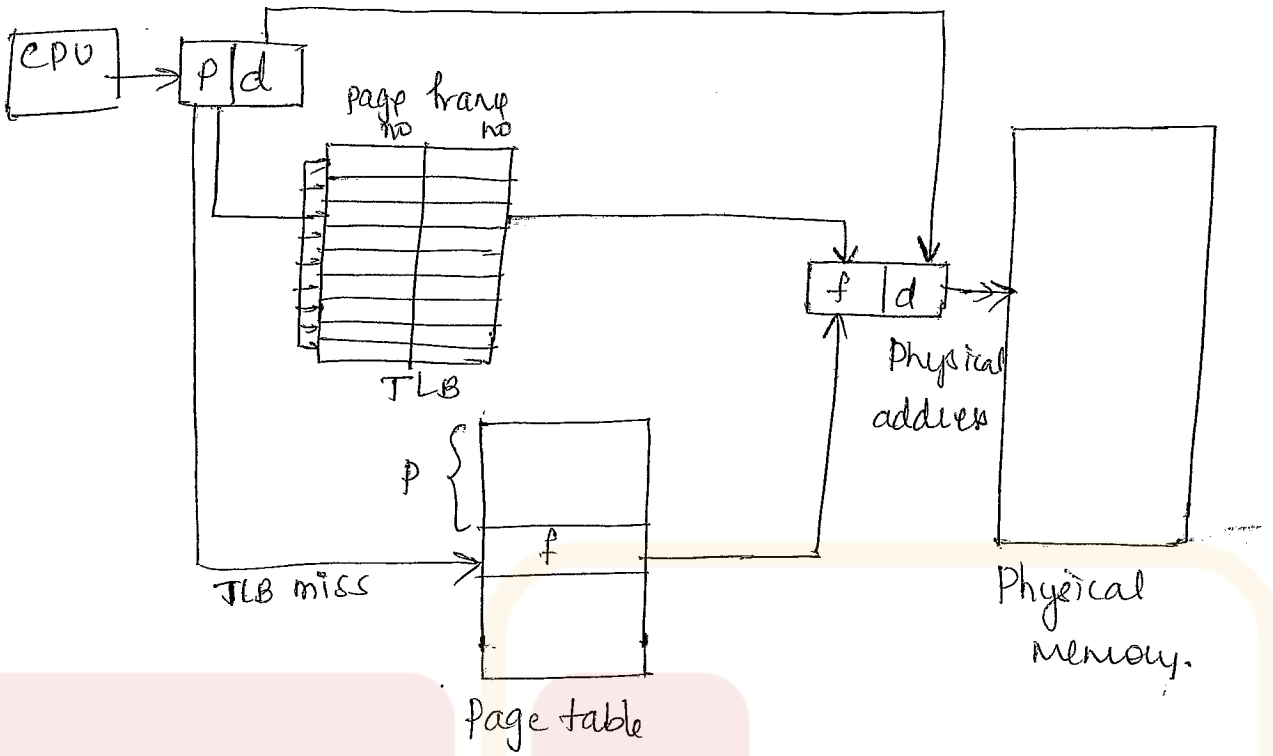
ii) Dynamic Loading & Dynamic Linking

- Size of process is limited to size of physical memory
- To obtain better memory-space-utilization, dynamic loading is used.
- With dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in relocatable load format
- The main program is loaded into memory & is executed.
- When routine needs to call another routine, the calling routine first checks to see whether other routine has been loaded.
- Dynamic linking is similar to dynamic loading, rather than postponing loading until execution, linking is postponed.

With dynamic linking, a stub is included in the image for each library routine reference.
- When stub is executed, it checks to see whether needed routine is already in memory, if not otherwise, stub replaces itself with address of routine & executes the routine.
- Next time when code segment is reached, the library routine is executed directly incurring cost for dynamic linking.

8C What is TLB? Explain TLB in detail with simple paging system with a neat diagram. [8M]

→ TLB is a special, small fast-look up hardware called translation-look-aside buffer (TLB)
- Each entry in TLB consists of two parts a key & value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously
- if item is found, corresponding value field is returned.
- Search is fast, hardware is expensive.
- The TLB contains only a few page table entries.
- When a logical address is generated by the CPU, its page no is presented to the TLB.
- If the page no is found, its frame number is immediately available & is used to access memory



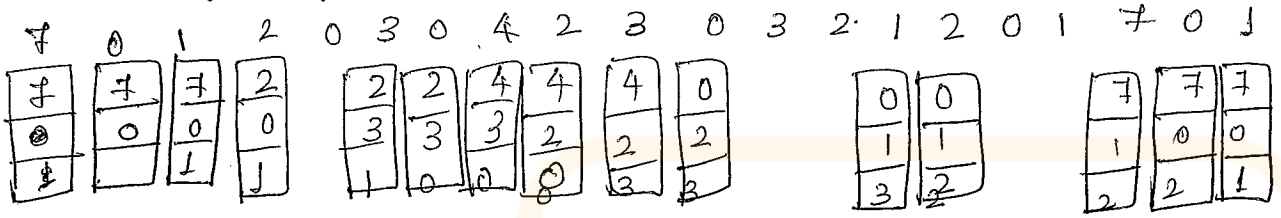
If page is not in TLB (TLB miss) a memory reference to the page table must be made when frame number is obtained, we can use it to access memory.

- This page number & frame number will be added to TLB, so that they will be found quickly on the next reference.
- If TLB is already full of entries, the OS must select one for replacement.
- Some TLB allows certain entries to be wired down, meaning that they cannot be removed from the TLB

Module 5

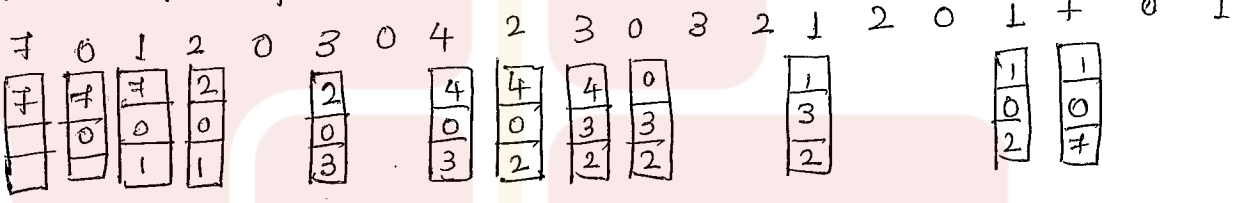
9a Consider the following page reference string 7 0 1 2 0 3 4 2 3 0 3 2 1 2 0 1 7 0 1. How many page faults would occur in the following algorithm
i) LRU ii) FIFO iii) Optimal [8M]

→ FIFO page replacement



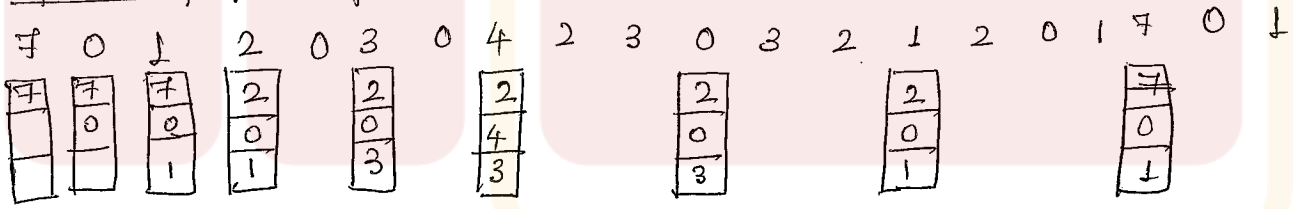
no. of faults = 15

LRU page replacement



no. of faults = 12

Optimal page replacement



No. of faults = 9

9b. Explain steps in handling page faults with a neat diagram.

→ The steps in handling page faults are

i) We check an internal table for this process to determine whether the reference was a valid or an invalid memory access.

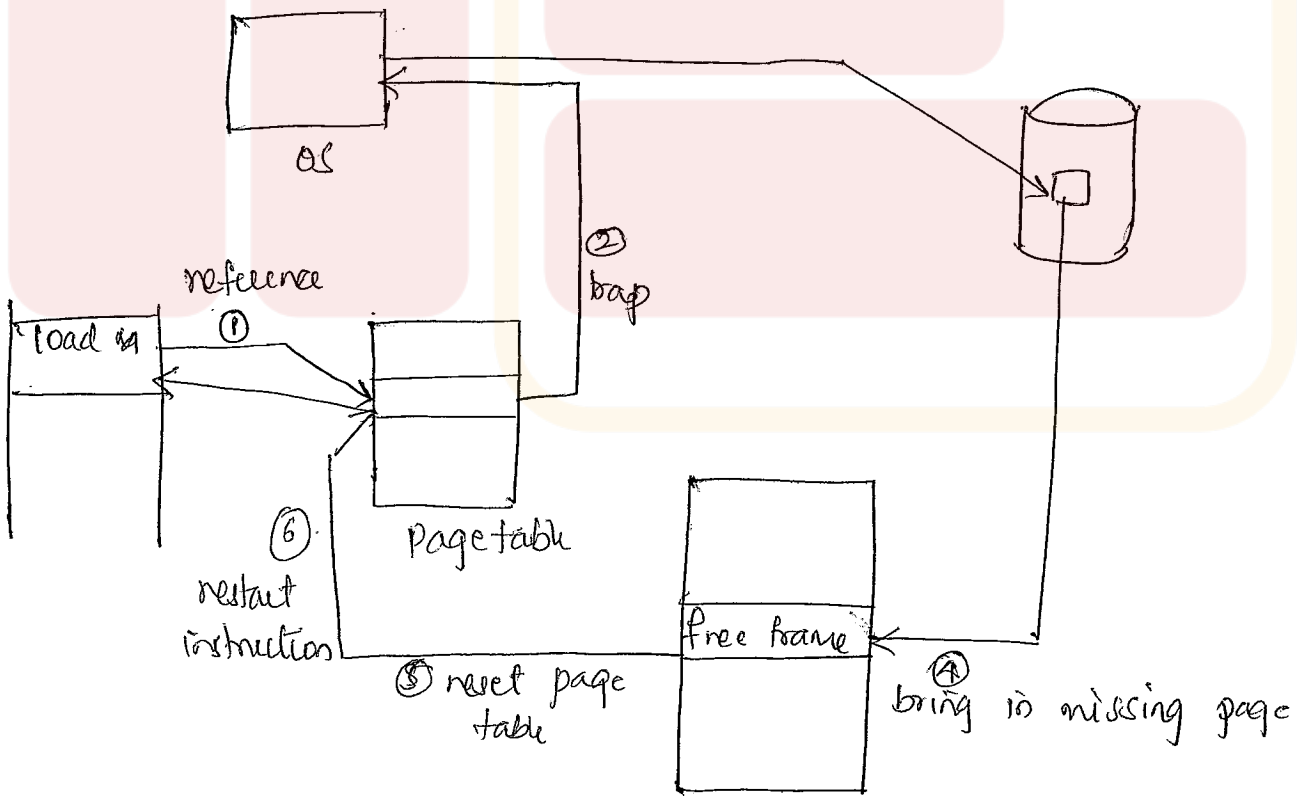
2) If reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page we now page it in.

3) We find a free frame.

4) We schedule a disk operation to read the desired page into the newly allocated frame

5) When the disk read is complete, we modify the internal table kept with the process & the page table to indicate that the page is now in memory.

6) We restart the the instruction that was interrupted by the trap. The process can now access the page as though it has always been in



qc What is thrashing? How it can be controlled? [6M]

→ If number of frames allocated to a low priority process falls below the minimum number required by the computer architecture, then suspend that process execution.

- We should then page out its remaining pages, freeing all its allocated frames. This requires swap-in & swap out level of scheduling.
- If any process does not have enough frames, it will quickly page fault.
- At this point, it must replace some pages.
- Since these pages are in active use, it must replace a page that will be needed again. Consequently it faults again & again.
- This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

<u>To prevent thrashing, working set strategy is used</u>

- This technique starts by looking at how many frames a process is actually using.
- This approach defines locality model of process execution
- locality model states that, as a process executes it moves from locality to locality.

locality is set of pages that are actively used together.

- program is composed of several different localities which may overlap.

example — If a process under execution calls a subroutine, it defines a new locality where memory references are made to the instructions of subroutine, its local variable & subset of global variable.

- When subroutine is exited, process leaves the locality; process can return to this locality later.
- localities are defined by program structure & its data structure.

10a. Explain demand paging in detail.

→ demand paging is paging system with swapper. process reside in secondary memory. To execute process, it must be swapped in into memory. Instead of swapping entire process into memory lazy swapper is used.

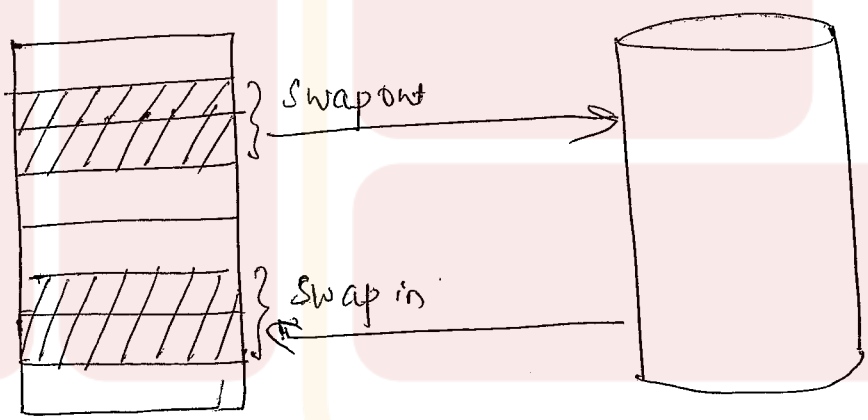
- Lazy swapper never swaps page into memory until page is needed.
- swapper manipulates entire process, whereas pager is concerned with individual pages of a process.
- pager is used instead of swapper in demand paging.
- When a process is to be swapped in, pager guesses

which pages will be used before procen is swapped out.

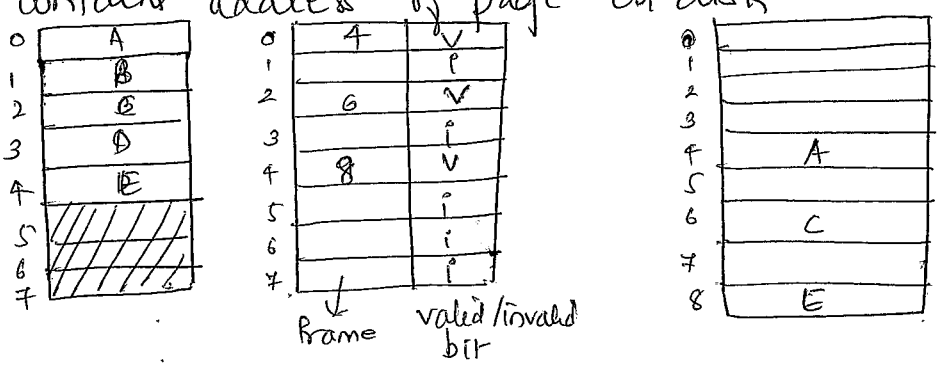
- Instead of swapping in whole procen, the pager brings only those necessary pages into memory. Thus decreases swap time & amount of physical memory needed.
- valid & invalid bits are used to distinguish between pages in memory & pages on the disk.

i) When bit is set to valid - indicates : page is both legal & is in memory.

ii) When bit is set to invalid indicates page is either valid or invalid but currently on disk



page table entry for page that is brought into memory is set as usual, but page table entry for page that is not currently in memory is marked invalid or contains address of page on disk



10b) Explain briefly various operations performed on files. [7M]

→ various operations performed on files are

i) creating a file - Two steps are necessary 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.

ii) writing a file - To write a file, we make a system call specifying both the name of the file & the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location. A write pointer must be updated whenever a write occurs.

iii) Reading a file - To read from a file, we use a system call that specifies the name of the file & where the next block of the file should be put. Again, the directory is searched for the associated entry, & the system needs to keep a read pointer to the location in the ~~total~~ file where the next read is to take place

iv) Repositioning within a file - The directory is searched for the appropriate entry & the current-file-position pointer is repositioned to a given value.

v) 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 & leave the directory entry.

vi) Truncating a file - The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file & then recreate it, this function allows all attributes to remain unchanged - except for file length - but lets the file be reset to length zero & its file space released.

10c) Explain various access methods on files [6M].

→ files store information. When it is used, this information must be accessed and read into computer memory.

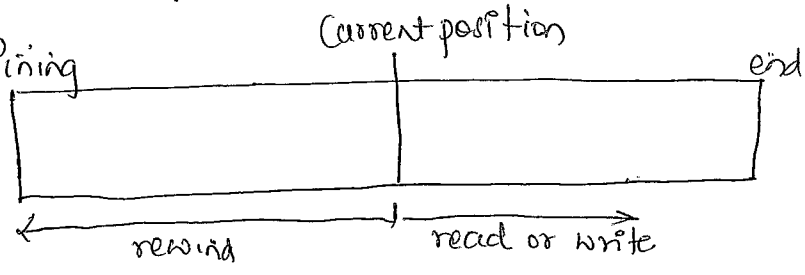
i) Sequential access

ii) Direct access

iii) Other access methods.

i) Sequential access

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



ii) Direct access

- The direct-access method is based on a disk model of a file, since disks allow random access to any file blocks.
- For direct access, the file is viewed as a numbered sequence of blocks or records
- Thus, we may read block 14, the read block 53 & then write block 7.

iii) Other access methods

- Other access methods can be built on top of a direct-access method,
- These methods generally involve the construction of an index for the file.
- The index, like an index in the back of a book, contains pointers to the various blocks.
- To find a record in the file, we first search the index & then use the pointer to access the file directly & to find the desired record.

Prepared by
Prof. Yasmeen. Shaikh

HOD

DEAN ACADEMICS