Unit - 5

- **Memory Management** : It handles & manages primary memory & moves processes back & forth between main memory & disk during execution.
  - → It keeps track of each & every memory location, regardless of either it is allocated to some process or it is free.
  - → It checks how much memory is to be allocated to processes.
  - → It decides which process will get memory at what time.
  - → It tracks whenever some memory gets freed or unallocated & correspondingly it updates the status.

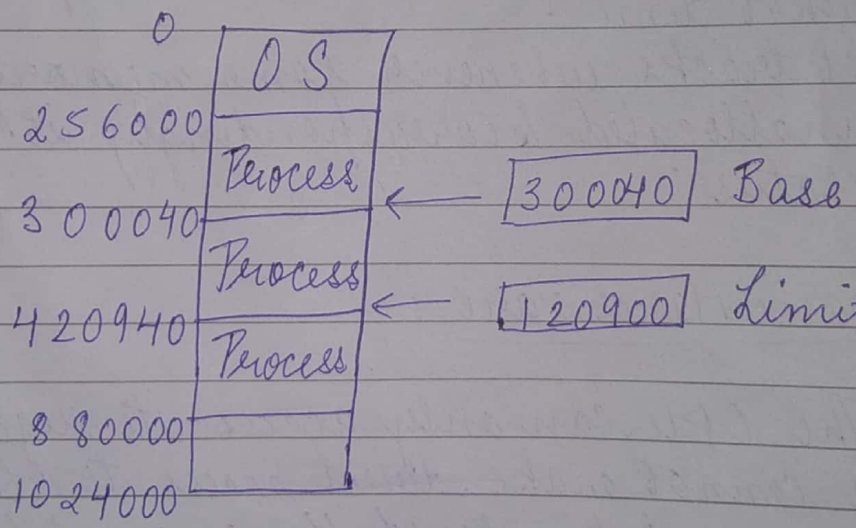
i) Basic Hardware :

- The CPU can only access its registers & memory. It cannot make direct access to the hard drive, so any data stored there, must be transferred into main memory chips before the CPU can work with it.

- Memory access to registers are very fast, generally one clock tick & memory access to main memory are comparatively slow & may take a no. of clocks ticks to complete.

- We need to make sure that each process has a separate memory space. To do this, we need the ability to determine the range of legal addresses that the process may access & to ensure that the process can access only these legal addresses.

We can provide this protection using two registers. The Base Register holds the smallest legal physical memory address. The limit register specifies the size of the range.



ii) Address Binding: User programs refer to memory addresses with symbolic names. These symbolic names must be mapped to physical memory addresses, which occurs in several stages:

a. Compile Time: If it is known at compile time where the process will reside in

memory then the absolute code can be generated.

b. Load time: If the location at which a program will be loaded is not known at compile time, then the compiler generate relocatable code, which references addresses relative to the start of program.

c. Execution time: If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time.

iii) Logical V/S Physical Address Space:

- The address generated by CPU is called as logical address whereas the address loaded into the memory address register of memory is referred to as physical address.
- Addresses bound at compile time or load time have identical logical & physical addresses.
- Addresses created at execution time have different logical & physical addresses.
  → In this case, the logical address is also known as a virtual address.

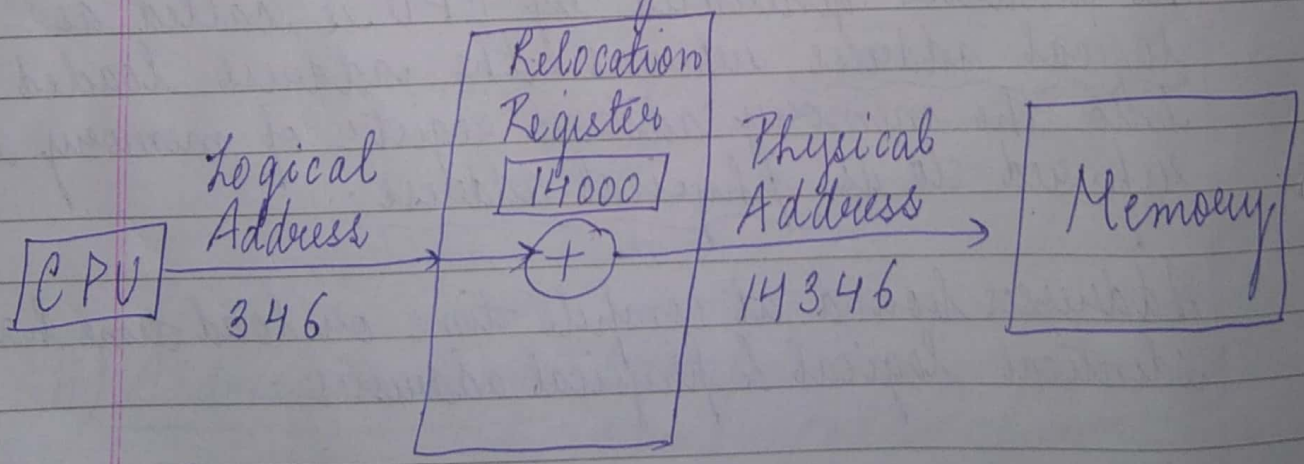
→ The set of all logical addresses generated by a

Teacher's Signature

program is a logical address space & the set of all physical addresses corresponding to these logical addresses is a physical address space.

- The run time mapping from virtual to physical addresses is done by Memory Management Unit (MMU).

→ In this, the base register is now called a relocation register & the value in the relocation register is added to every address generated by a user process at that time the address is sent to memory.



MMU

Dynamic Relocation using a Relocation Register

iv) Dynamic Loading:

Rather than loading an entire program at once into memory, dynamic loading loads up each routine as it is called. The advantage is that unused routines is never loaded, reducing total memory usage & generating faster program startup times.

The downside is the added complexity & overhead of checking to see if a routine is loaded every time it is called & then loading it up if it is not already loaded.

v) Dynamic Linking & Shared Libraries:

Some OS support only static linking, in which system language libraries are treated like any other object module & are combined wasting both the disk space & main memory usage because each program on a system must include a copy of its libraries.

With dynamic linking, a stub is included for each library-routine reference. A stub is a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not present.

This method saves disk space, because the library routines do not need to be fully included, only the stubs are included in the executable module.

Benefit of dynamically linked libraries (DLL) also known as shared libraries involves easy upgrades & updates. When a program used a routine from a standard library & the routine changes, then the program must be relinked or rebuilt in order to incorporate the changes.

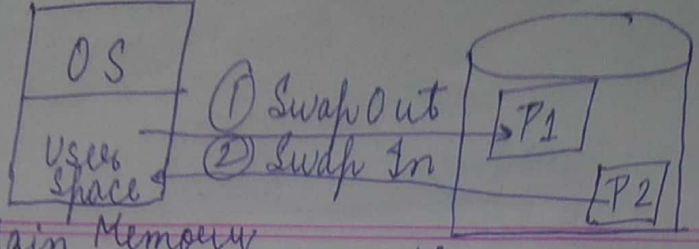
However, if DLL's are used, then as long as the stub doesn't change, the program can be updated merely by loading new versions of DLLs onto the system. Version information is maintained in both the program & the DLLs, so that a program can specify a particular version of DLL if necessary.

- Swapping: A process must be in memory to be executed. A process, however can be swapped temporarily out of memory to a backing store & then brought back into memory for execution.

continued

A variant of this swapping policy is used for priority based scheduling algorithms. If a higher priority process arrives &

Teacher's Signature



wants services, the memory manager can swap out the lower-priority process & then load & executes the higher priority process. When the higher priority process finishes, the lower priority process can be swapped back in & continued. This variant of swapping is sometimes called roll out, roll in.

- Contiguous Memory Allocation: It allocates consecutive blocks of memory to a process. In this, the memory is divided into several fixed sized partitions. Each partition may contain exactly one process.

In the multiple-partition method, when a partition is free, a process is selected from the input queue & is loaded into the free partition. When the process terminates, the partition becomes available for another process.

In the variable-partition method, the OS keeps a table indicating which parts of memory are available & which are occupied. Initially all memory is available for user processes & is considered one large block of available memory, a hole. So, memory contains a set of holes of various sizes.

As processes enter the system, they are put into an input queue. The OS takes into account the memory requirements of each process & the amount of available memory space in determining which processes are allocated memory. When a process is allocated space, it is loaded into memory & it can then compete for CPU time. When a process terminates, it releases its memory, which the OS may then fill with another process from input queue.

For Dynamic storage allocation problem, which concerns how to satisfy a request of size 'n' from a list of free holes. There are many solutions to this problem.
The First-fit, Best-fit & Worst-fit strategies are the ones most commonly used to select a free hole from the set of available holes.

i) First-Fit : Allocates the first hole that is big enough.

ii) Best-Fit : Allocates the smallest hole that is big enough.

iii) Worst-Fit : Allocates the largest hole.

Teacher's Signature

- **Fragmentation**: It occurs in dynamic memory allocation system when many or most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

i) **External Fragmentation**: It occurs when there is a sufficient amount of space in the memory to satisfy the memory request of a process. But the process's memory request cannot be satisfied as the memory available is in a non contiguous manner.

One solution to this problem is compaction. The goal is to shuffle the memory contents so as to place all free memory together in one large block.

ii) **Internal Fragmentation**: It occurs whenever a process request for the memory, the fixed sized block is allocated to the process. In case, the memory assigned to the process is somewhat larger than the memory requested, then the difference between assigned & request memory is the Internal Fragmentation.

- **Segmentation**: It is a memory management scheme that supports the user view of memory. A logical address space

Leads to External Fragmentation

→ Variable size

is a collection of segments. Each segment has a name & a length. The addresses specify both the segment name & the offset within the segment.

The user therefore specifies each address by 2 quantities: a segment name & an offset.

Segments are numbered & are referred to by a segment number, rather than by a segment name. Thus a logical address consists of a 2 tuple: 'd' as the logical address must be in b/w 0 & segment limit

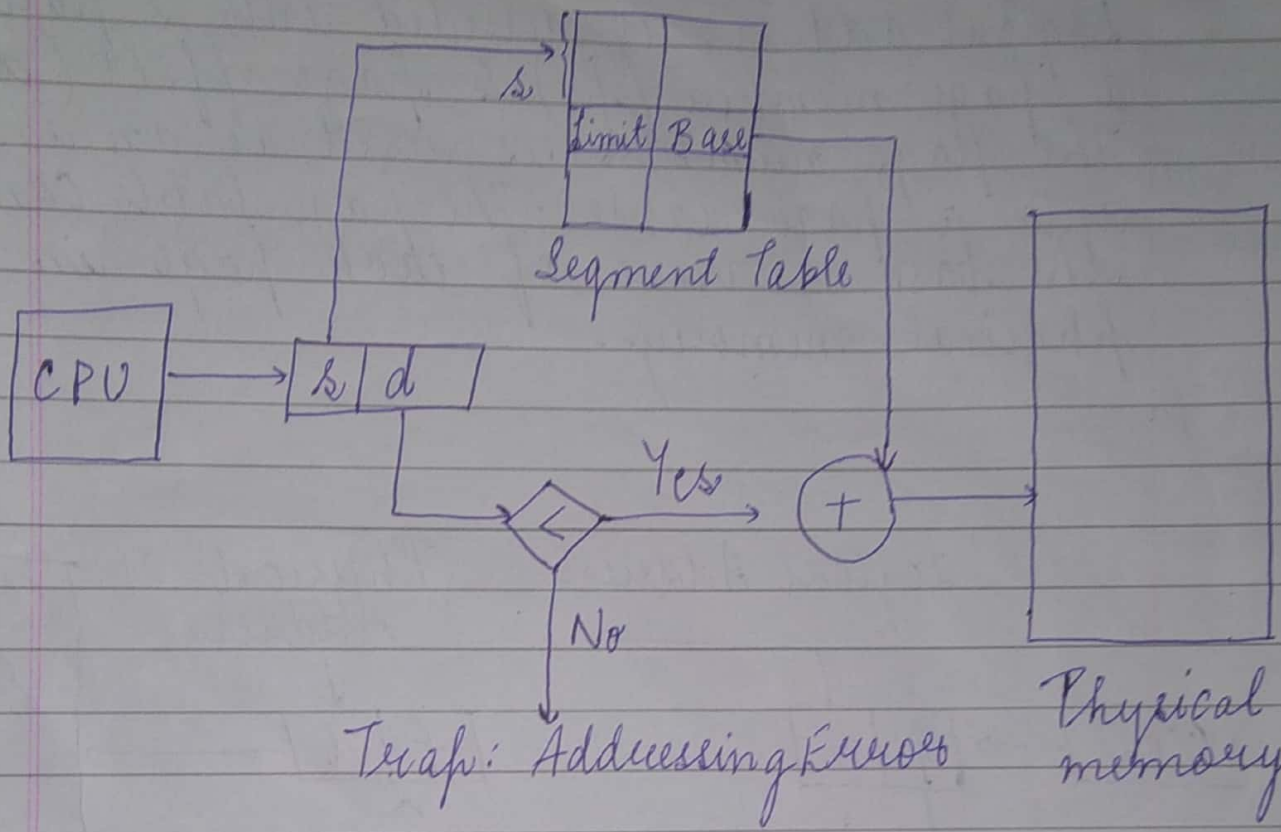
< segment-no., offset >

's' it is used as an index to the segment table

Thus, we define an implementation to map 2-D user defined addresses into 1D physical addresses & this mapping is effected by a segment table.

Each entry in the segment table has a segment base & a segment limit. Each entry in the segment table has a segment base & a segment limit. The segment base contains the starting physical address where the segment resides in memory & the segment limit specifies the length of the segment.

Teacher's Signature



Segmentation Hardware

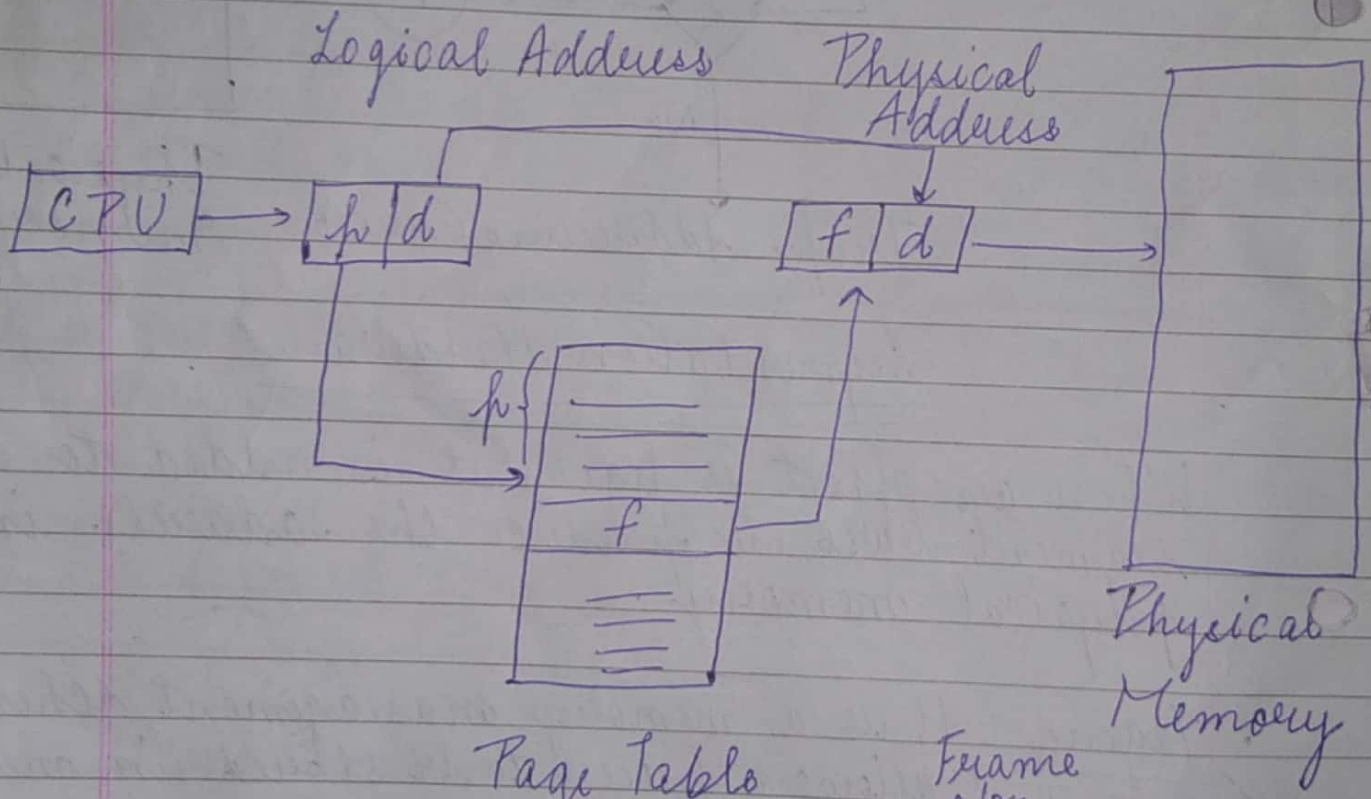
When an offset is legal, it is added to the segment base to produce the address in physical memory.

- **Paging**: It is a memory management scheme & allows a process to be stored in memory in a non-contiguous manner.

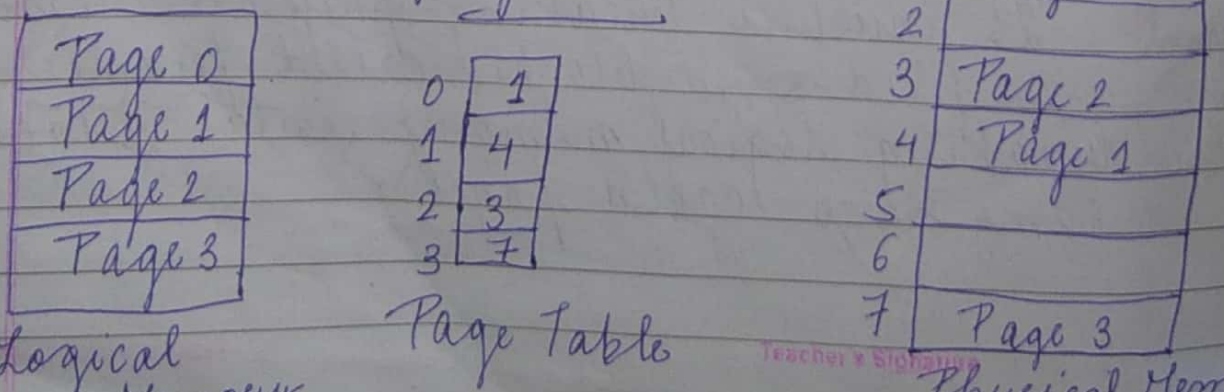
avoids external fragmentation

It involves breaking physical memory into fixed-sized blocks called frames & breaking logical memory into blocks of same size called pages.

Logical address is divided into 2 parts: a page number (p) & page offset (d). The page number is used as an index into a page table. The page table contains the base address of each page in physical memory.



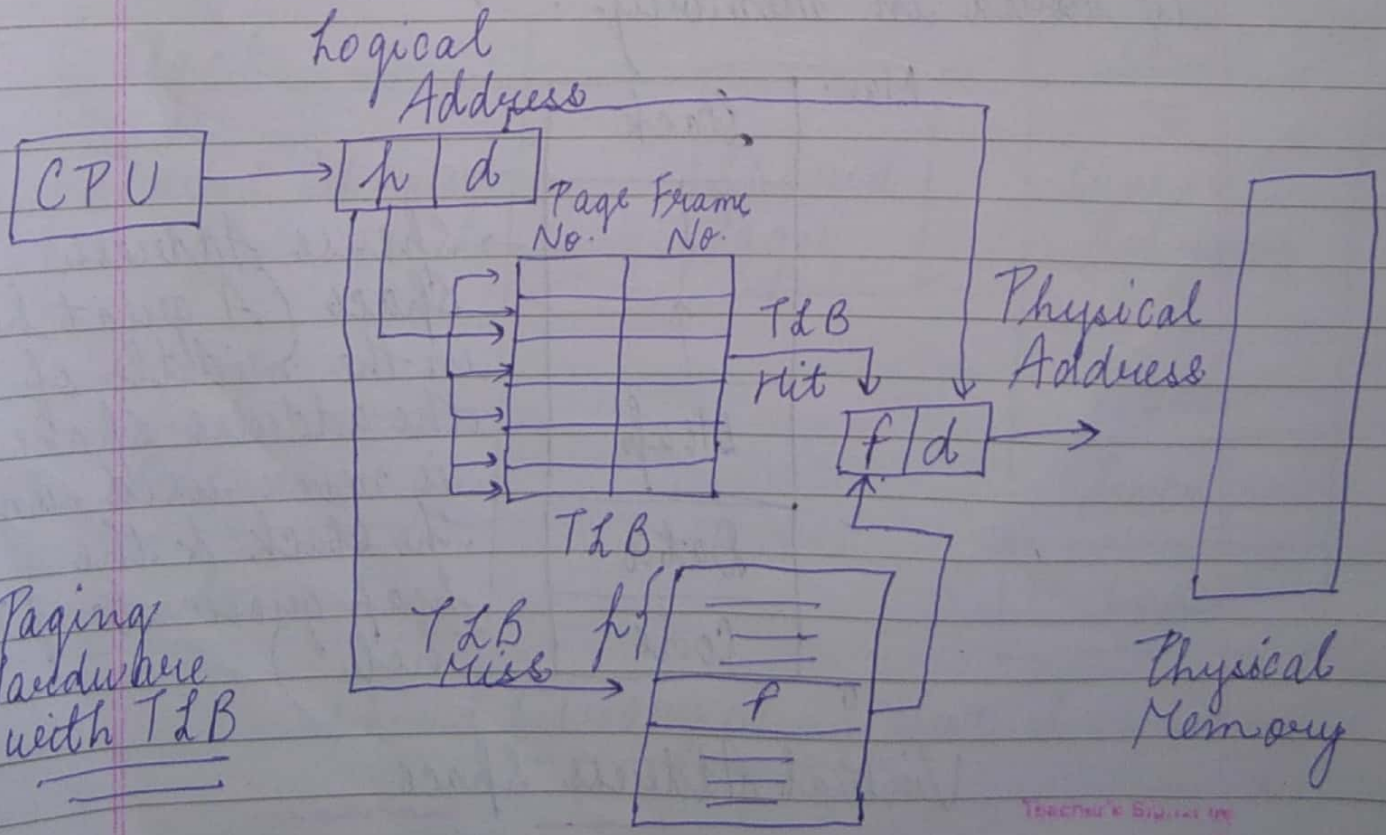
Paging Hardware



The standard solution is to use a special, small, fast-lookup hardware cache, called a translation look aside buffer (TLB). It is associative, high speed memory. Each entry in TLB consists of 2 parts: a key (tag) & a value.

The TLB contains only a few of the page table entries. When a logical address is generated by CPU, its page number is presented to the TLB. If the page no. is found, its frame no. is immediately available & is used to access memory.

If the page no. is not in TLB known as TLB Miss.

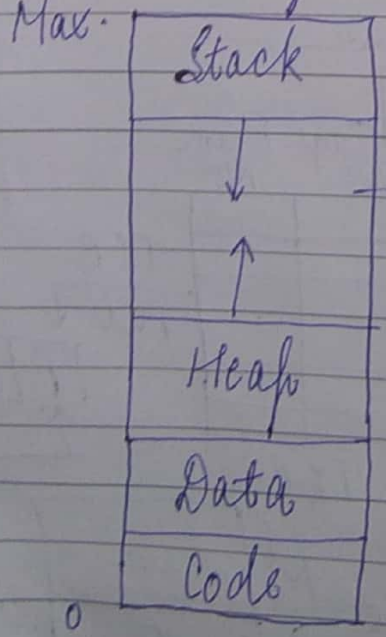


The percentage of times that a particular page no. is found in the TLB is called the hit ratio.

- Virtual Memory: It involves the separation of logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

It makes the task of programming much easier, because the programmer no longer need to worry about the amount of physical memory available.

The virtual address space of a process refers to the logical view of how a process is stored in memory.



Virtual Address Space

Teacher's Signature

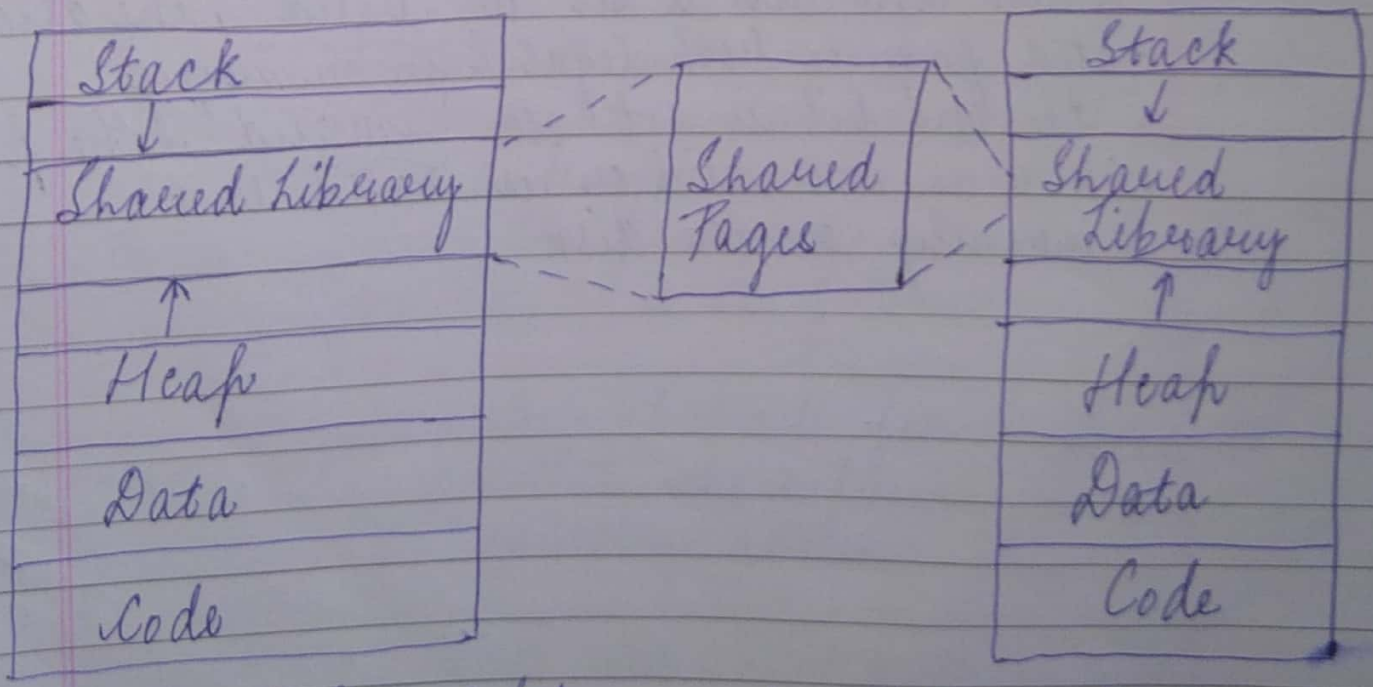
1 Memory

The ability to load only the portions of processes that are actually needed has several benefits:

i) Programs could be written for a much larger address space than it physically exists on the computer.

ii) Because process is only using a fraction of their total address space, there is more memory left for other programs, improving CPU utilization & system throughput.

It allow the sharing of files & memory to be shared by two or more processes through page sharing.



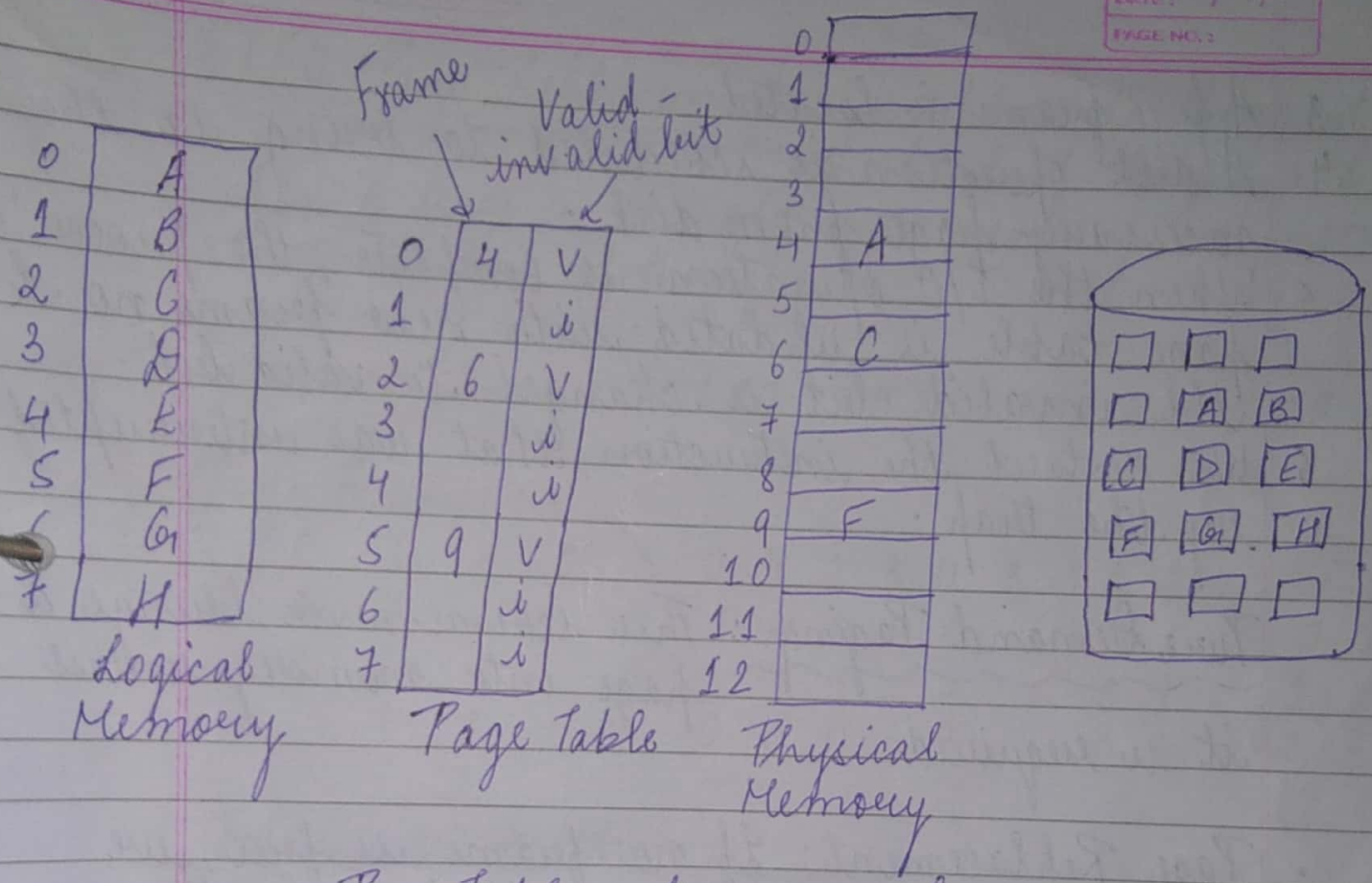
Shared library using Virtual Memory

- Demand Paging : It is similar to a paging system with swapping. Where processes reside in secondary memory. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however we use a lazy swapper. A lazy swapper never swaps a page into memory unless that page will be needed.

A swapper manipulates entire process, whereas a pager is concerned with the individual pages of a process. So, we thus use pagers rather than swappers, in connection with demand paging.

When the bit is set to 'valid', the associated page is both legal & in memory.
If the bit is set to 'invalid', the page either is not valid or is valid but is currently on the disk.

Teacher's Signature



Page Table when some pages are not in memory

If the process tries to access a page that was not brought into memory then access to a page marked invalid causes a page fault. The procedure for handling this page fault is as follows:

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

2. If the reference was invalid then the process is terminated. Otherwise the page must be paged in.

3. A free frame is located.
4. A disk operation is scheduled to bring in the necessary page from disk.
5. When the I/O operation is complete, the process's page table is updated with new frame no. & the invalid bit is changed to valid bit.
6. We restart the instruction that was interrupted by the trap.

Pure Demand Paging: This scheme never brings a page into memory unless it is required.

Page Replacement: If no frame is free, we find one that is not currently being used & free it.

1. Find location of desired page on disk.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to disk, change the page & frame table accordingly.
3. Read the desired page into the newly freed frame; change the page & frame tables.
4. Restart the user process.

Teacher's Signature

a. **FIFO Page Replacement**: We replace the page at the head of the queue & when a page is brought into memory, we insert it at the tail of the queue.

Eg- Reference String : 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page Frames : 3

→

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

Total = 15 Faults

b. **Optimal Page Replacement**: It has the lowest page fault rate of all the algorithms. In this, we replace the page that is not used for the longest period of time.

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

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

Total = 9 Faults

Teacher's Signature

c. LRU Page Replacement: We replace the page that has not been used for longest period of time. This approach is called LRU.

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

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

Unit-4

Interprocess Communication (IPC):

Cooperating processes requires an IPC mechanism that will allow them to exchange data & information. There are 2 fundamental models of IPC:

1. **Shared Memory**: IPC using shared memory requires communicating processes to establish a region of shared memory. 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.

Producer - Consumer problem: A producer produces information that is consumed by a consumer process. For eg - a Web server produces HTML files & images, which are consumed by the client requesting the resource.

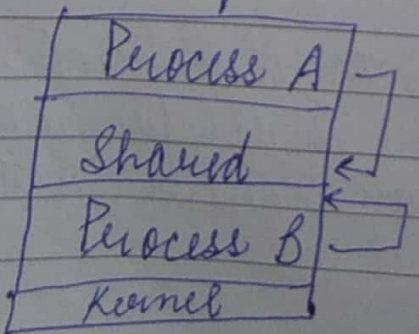
One solution to this problem is to use shared memory. To allow producer & consumer processes to run concurrently, we must

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 by the producer & consumer process. 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.

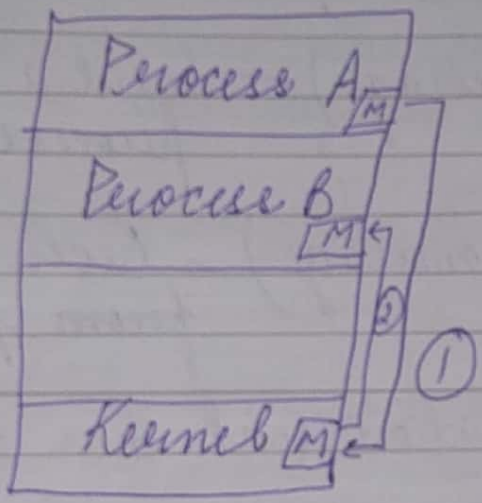
2 type of buffers are used :

i) Unbounded buffer places no practical limit on the size of buffer. The consumer may have to wait for new items, but the producer can always produce new items.

ii) Bounded Buffer assumes a fixed size of buffer. In this case, the consumer must wait if the buffer is empty & the producer must wait if the buffer is full.



2. Message Passing: In this, communication takes place by means of messages exchanged between the cooperating processes.



They do not share the same address space. A message passing facility provides at least two operations: send (message) & receive (message)

If processes P & Q want to communicate, they must send messages to & receive messages from each other; a communication link must exist between them. Several methods for implementing a link & the send ( )/receive ( ) operations:

1. Direct or Indirect Communication
2. Synchronous or asynchronous communication
3. Automatic or Explicit Buffering

1. Under direct communication, each process

that wants to communicate must explicitly name the recipient or sender of the communication. In this, send() & receive() primitives are defined as:

- send (P, message) – Send a message to process P.
- receive (Q, message) – Receive a message from process Q.

A communication link has the following properties:

- A link is established between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
- A link is associated with exactly two processes.
- Between each pair of processes, there exists exactly one link.

Symmetry in addressing i.e. both the sender & receiver processes must name the other to communicate.

Teacher's Signature

DATE: / /
PAGE NO.:

Asymmetry in addressing i.e. only the sender names the recipient & the recipient is not required to name the sender. In this scheme, the send() & receive() primitives are defined as follows:

- send (P, message) - Send a message to process P.
- receive (id, message) - Receive a message from any process; the variable id is set to the name of process with which communication has taken place.

Under indirect communication, the messages are sent to & received from mailboxes or ports. A mailbox can be viewed as an object into which messages can be placed by a process & from which messages can be removed. Each mailbox has a unique identification. Two processes can communicate only if the processes have a shared mailbox. The send() & receive() primitives are defined as:

- send (A, message) — Send a message to mailbox A.
- receive (A, message) — Receive a message from mailbox A.

In this scheme, a communication link has the following properties:

- A link is established between a pair of processes only if both members of the pair have a shared mailbox.
- A link may be associated with more than two processes.
- Between each pair of communicating processes, there may be a no. of different links, with each link corresponding to one mailbox.

A mailbox may be owned either by a process or by OS. If the mailbox is owned by a process then we distinguish between the owner (which can only receive messages through this mailbox) & the user (which can only send messages to the mailbox). When a process that owns a mailbox terminates, the mailbox disappears.

If a mailbox is owned by the OS then it has an existence of its own. It is independent & is not attached to any particular process. The OS allow a process to do the following:

- Create a new mailbox.

- Send & receive messages through mailbox.
- Delete a mailbox.

In this, the ownership & receiving privilege may be passed to other process through appropriate system calls.

2. Message passing may be either blocking or non blocking also known as Synchronous & asynchronous.

- Blocking send - The sending process is blocked until the message is received by the receiving process or by the mailbox.
- Nonblocking send - The sending process sends the message & resume operation.
- Blocking receive - The receiver blocks until the message is available.
- Nonblocking receive - The receiver retrieves either a valid message or a null.

3. Whether communication is direct or indirect, messages exchanged by communicating process -es reside in temporary queue. Basically, such queues can be implemented in 3 ways:

Teacher's Signature

- Zero capacity : In this, the link cannot have any messages waiting in it. The senders must block until the recipient receives the message.

- Bounded Capacity : The queue has finite length n ; thus at most n messages can reside in it.
  If the queue is not full when a new message is sent, the message is placed in the queue & the sender can continue execution without waiting. The link's capacity is finite. The sender must block until space is available in the queue, if the link is full.

- Unbounded Capacity : The queue's length is infinite, thus any no. of messages can wait in it. The senders never blocks.

Process Synchronization : It means sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.
Process synchronization was introduced to handle problems that arose while multiple process executions.

Race Condition: A situation where several processes access & manipulate the same data concurrently & the outcome of execution depends on the particular order in which the access takes place is called Race Cond.

Critical Section problem: It is a code segment that accesses shared variables & has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.

Each process must request permission to enter its critical section. The section of code implementing this request is the entry section. The critical section may be followed by an exit section. The remaining code is the remainder section.

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

General structure of typical process

A solution to the critical section problem must satisfy the following requirements:

1. Mutual Exclusion: If process P_i is executing in its critical section then no other processes can be executing in their critical sections.

2. Progress: If no process is executing in its critical section & some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical sections next.

3. Bounded Waiting: There exists a bound or limit, on the no. of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section & before that request is granted.

Two general approaches are used to handle critical sections in OS:

i) Preemptive Kernels:
ii) Non-preemptive Kernels

Teacher's Signature

Preemptive Kernels	Non-preemptive Kernels
i) It allow a process to be preempted while it is running in kernel mode.	It does not allow a process running in kernel mode to be preempted.
ii)	Only one process is active in the kernel at a time.
iii) More suitable for real-time programming as it will allow a real time process to preempt a process currently running in the kernel.	

- Peterson's Solution: It is a classic software based solution to the Critical section problem. It is restricted to two processes & the processes are numbered P_0 & P_1.

It requires two processes to share two data items:

```
int turn;          → turn is to enter its critical section
boolean flag[2];   → array is used to indicate if a process is ready to enter CS
```

If turn == i, then process P_i is allowed to execute in its critical section.
If flag[i] is true, then P_i is ready to enter its critical section.

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

    Critical section

    flag[i] = FALSE;

    Remainder section
} while (TRUE);
```

It preserves all the 3 conditions:

i) Mutual exclusion is assured as only one process can access the critical section at any time.

ii) Progress is also assured as a process outside the critical section does not block other processes from entering critical section.

iii) Bounded waiting is preserved as every process gets a fair chance.

Teacher's Signature

- Syncheronization Hardware: It requires a Lock i.e., a process must acquire a lock before entering a critical section; it releases the lock when it exits the critical section.

```
do {
    [acquire lock]
        Critical Section
    [release lock]
        Remainder Section
} while (TRUE);
```

The critical section problem can be solved easily in a single processor environment if we disallow interrupts to occur while a shared resource is being modified. This solution is not feasible in a multiprocessor environment.

Disabling interrupts on multiprocessor can be time consuming so we use special instruction TestAndSet( ) & Swap( ) instruction to solve the critical section problem.

- Classic Problems of Synchronization:

i) Bounded Buffer Problem: We assume that pool consists of 'n' buffers, each capable of holding one item.

a. Mutex Semaphore provides mutual exclusion for access to the buffer pool & is initialized to the value 1.

b. Empty Semaphore count the no. of empty buffers & is initialized to the value n.

c. Full Semaphore count the no. of full buffers & is initialized to the value 0.

```
do {
    // produce an item in nextp
    wait(empty);
    wait(mutex);
    // add nextp to buffer
    signal(mutex);
    signal(full);
} while (TRUE);
```

Structure of Producer Process

```
do {
    wait (full);
    wait (mutex);

    // remove an item from buffer to next c

    signal (mutex);
    signal (empty);

    // consume the item in next c
} while (TRUE);
```

Structure of Consumer Process

ii) Readers - Writers Problem: Suppose that a database is to be shared among several concurrent processes. Some of the processes may want to read the database referred as the readers & some others want to update the database referred as writers.

If two readers access the shared data simultaneously, no adverse effects will result. However, if reader & writer process access database simultaneously, chaos or adverse effect will result.

To ensure that these difficulties do not arise, we require that the writers have exclusive access to shared database while

Teacher's Signature

writing to the database.

semaphore mutex, wrt; ← initialized to 1
int readcount; ← initialized to 0

a) 'wrt' is common to both readers & writers processes. It functions as a mutual-exclusion semaphore for writers.

b) 'mutex' is used to ensure mutual exclusion when the variable readcount is updated.

c) 'readcount' variable keeps track of how many processes are currently reading the object.

If a writer is in the critical section & 'n' readers are waiting, then one reader is queued on 'wrt' & 'n-1' readers are queued on 'mutex'.

When a writer executes a signal (wrt), we assume the execution of either the waiting readers or single waiting writer. The selection is made by the scheduler.

Teacher's Signature

```
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);
```

Structure of Reader Process

```
do {
    wait (wrt);

// Writing is performed

    signal (wrt);
} while (TRUE);
```

Structure of Writer Process

iii) Dining - Philosophers Problem: Consider 5 philosophers sharing circular table surrounded by 5 chairs, each belonging to one philosopher. In the center of the table is a bowl of rice & table is laid with 5 single chopsticks. A philosopher may pick up one chopstick at a time & cannot pick up a chopstick that is already in the hand of neighbour.

Represent each chopstick with a semaphore. A philosopher tries to grab a chopstick by executing wait operation on semaphore & releases the chopstick by executing the signal operation on semaphore.

```
semaphore chopstick[5];
```

where all the elements of chopstick are initialized to 1.

Structure of Philosopher

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);

    // Eat
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);

    // Think
} while(TRUE);
```

Teacher's Signature

- **Monitors:** All processes share a semaphore variable 'mutex' which is initialized to 1. Each process must execute wait(mutex) before entering the critical section & signal(mutex) afterwards. If this sequence is not observed, 2 processes may be in their critical sections simultaneously.

i) Suppose that a process interchanges the order in which wait() & signal() operations on the semaphore mutex are executed.

```
signal(mutex);
critical section
wait(mutex);
```

In this, several processes may be executing in their critical section simultaneously, violating the mutual exclusion requirement.

ii) Suppose that a process replaces a signal(mutex) with wait(mutex).

```
wait(mutex);
Critical Section
wait(mutex);
```

Deadlock will occur.

iii) Suppose that a process omits the wait(mutex)

Teacher's Signature

or signal (mutex) or both. In this case, either mutual exclusion is violated or a deadlock will occur.

These examples illustrate that various types of errors can be generated easily when programmers use semaphores incorrectly to solve critical section problem.

To deal with such errors, a high level language construct — monitor type has been developed.

A Monitor type is an ADT which presents a set of programmer-defined operations that provide mutual exclusion.

This contains the declaration of variables along with bodies of functions that operate on these variables.

Syntax of a Monitor

```
monitor monitor name
{
    // Shared Variable Declaration

    Procedure P1 ( ) {
    }
    .
    .
    Procedure Pn ( ) {
    }
    initialization Code ( ) {
    }
}
```

Teacher's Signature

The monitor construct ensures that only one process at a time is active within the monitor.

Define one or more variables of type condition:

```
condition x, y;
```

The only operations that can be invoked on condition variable are wait( ) & signal( ).

```
x.wait( );
x.signal( );
```

- Semaphores: The hardware based solutions to the critical section problem presented are complicated for application programmers to use. To overcome this difficulty, we use a synchronization tool called semaphore.

A Semaphore 'S' is an integer variable that is accessed only through two standard atomic operations: wait( ) & signal( ). Wait & Signal operations are also called P & V operations.

once started, execution cannot be interrupted

Definition of wait( )

Entry to the critical section is controlled by wait operation

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

Definition of Signal ( )

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

Exit from critical section is taken care by signal operation.

When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

	Counting Semaphore	Binary Semaphore
i)	Value range over an unrestricted domain.	Value can range only between 0 & 1.
ii)		Known as Mutex Locks as they are the locks that provide mutual exclusion.

Disadvantage of semaphore is that it requires busy waiting. While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the entry code. This continual looping is clearly a problem in real multiprogramming system.

Teacher's Signature

This type of semaphore is also called a spinlock because the process "spins" while waiting for the lock.

To overcome the need for busy waiting, we can modify the definition of wait() & signal() semaphore operations. When a process executes the wait() operation & finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore & the state of the process is switched to the waiting state. Then control is transferred to CPU scheduler, which selects another process to execute.

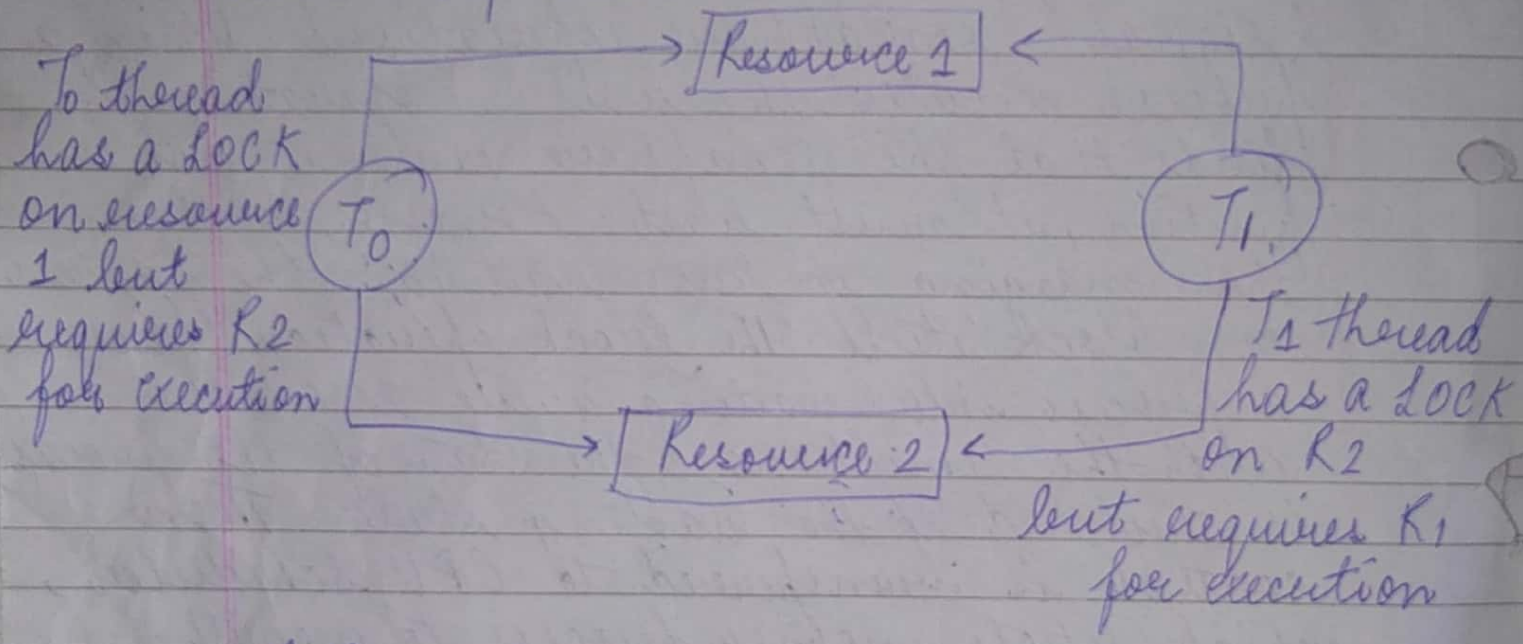
A process that is blocked, waiting on semaphore S, should be restarted when some other process executes a signal() operation. The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state. The process is then placed in the ready queue.

The block() operation suspends the process that invokes it. The wakeup(P) operation

Teacher's Signature

resumes the execution of blocked process P.

- Deadlock : Deadlocks are a set of blocked processes each holding a resource & waiting to acquire a resource held by another process.



A process may utilize a resource in only the following sequence:

1. Request : The process requests the resource. If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.
2. Use : The process operate on the resource.
3. Release : The process releases the resource.

Teacher's Signature

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. Mutual Exclusion: Only one process at a time can use the resource & if another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold & Wait: A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes.

3. No preemption: Resources cannot be preempted.

4. Circular Wait: A set of waiting processes exist such that P_0 is waiting for a resource held by P_1 & P_1 by P_2 & so on.

- Resource Allocation Graph: Deadlock can be described in terms of directed graph called a system resource allocation graph. This graph consist of a set of vertices 'V' & a set of edges 'E'.

The 'V' is partitioned into 2 different types of nodes:

$P = \{P_1, P_2, \ldots P_n\}$, the set consisting of all the active processes in system

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

$P_i \rightarrow R_j$ means that process P_i has requested an instance of resource type R_j & is currently waiting for that resource.

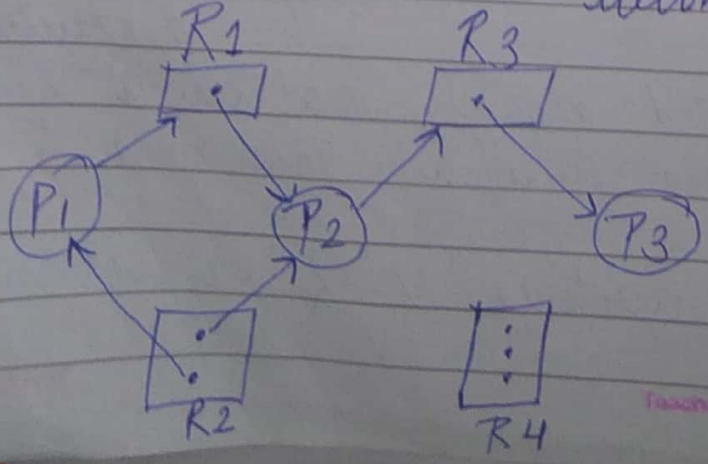
Request Edge

$R_j \rightarrow P_i$ means that instance of resource is allocated to process P_i.

Assignment Edge

Representation

Process P_i → Circle
Resource R_j → Rectangle
Each instance of the resource → dot within rectangle



- **Handling Deadlock**: We can deal with the deadlock problem in one of the three ways:

→ We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.

→ We can allow the system to enter a deadlocked state, detect it & recover.

→ We can ignore the problem altogether & pretend that deadlocks never occur in the system.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or deadlock avoidance scheme.

- **Deadlock Prevention**: It provides a set of methods for ensuring that atleast one of the necessary conditions cannot hold.

i) **Mutual Exclusion**: We cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non sharable.

Teacher's Signature

ii) Hold & Wait: There are two possibilities for elimination of 'hold & wait' cond.:

The first alternative is that it requires each process to request & be allocated all its resources before it begins execution.
The second alternative is that it allows a process to request resources only when it has none. A process may request some resources & use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.
Both these protocols have two main disadv-antages. First is resource utilization which may be low, since resources may be allocated but unused for a long period. Second, starvation is possible. A process that needs several resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

iii) No preemption: To ensure that this condition does not hold, we can use the following protocol. If a process is holding some resources & request another resource that cannot be immediately allocated to it, then all resources the process is currently

Teacher's Signature

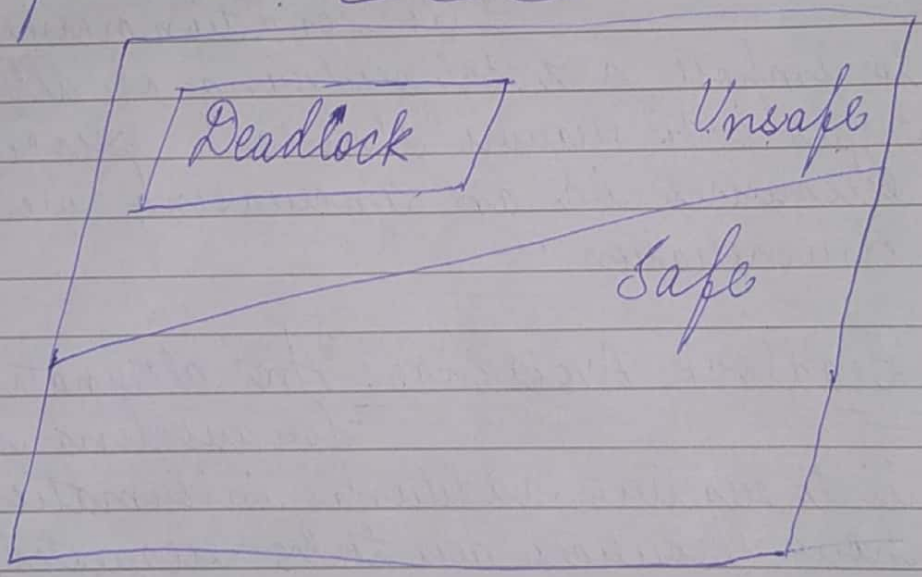
holding are preempted. These resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

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.

- Deadlock Avoidance: An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested. With the knowledge of complete sequence of requests & releases for each process, the system can decide for each request whether or not the process should wait in order to avoid a possible future deadlock.

i) Safe State : A state is safe if the system can allocate resources to each process in some order & still avoid a deadlock.

A sequence of processes $\langle P_1, P_2, \ldots P_n \rangle$ is a safe sequence, if for each P_i, the resource requests that P_i can be satisfied by the currently available resources plus the resources held by all P_j with $j < i$.
If no such sequence exists, then the system state is said to be unsafe. A safe state is not a deadlocked state. An unsafe state may lead to a deadlock.



Safe, Unsafe & Deadlocked State Spaces

Eg- 12 Tape Drives 3 Free

	Max. Needs	Current Needs	
P_0	10	5	→ At time t_0
P_1	4	2	
P_2	9	2	
		9	

Teacher's Signature

At time t_0, the system is in a safe state. The sequence $\langle P_1, P_0, P_2 \rangle$ satisfies the safety condition.

We have 3 free tape drives. Process P_1 can immediately be allocated all its tape drives & then return them (5 available tape drives) then P_0 can get all its tape drives & return them (10 available tape drives) & finally P_2 get all its tape drives & return them (12 tape drives available).

Whenever a process requests a resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait. The resource is granted only if the allocation leaves the system in a safe state.

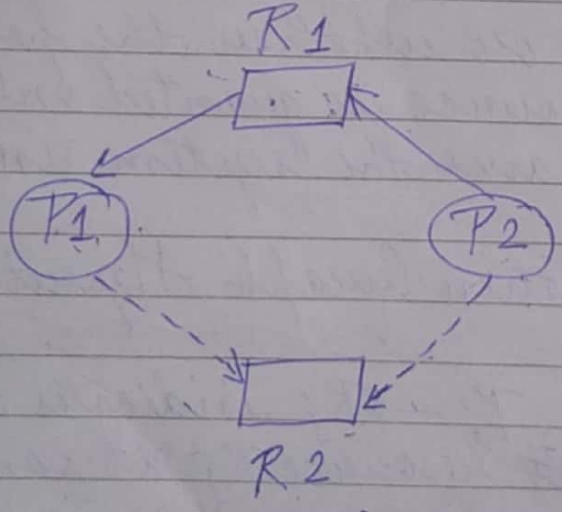
ii) Resource Allocation Graph Algorithm:

A Claim Edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future. This edge resembles a request edge in direction but is represented in the graph by a dashed line.

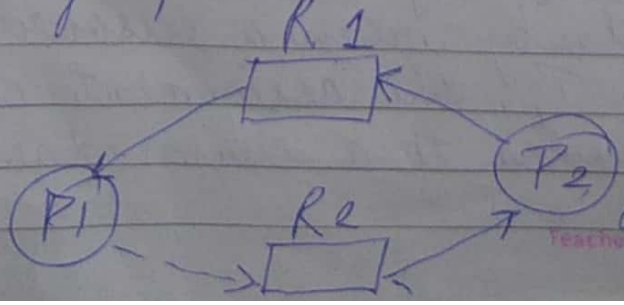
When process P_i requests resource R_j, the claim edge $P_i \rightarrow R_j$ is converted to request edge. Similarly when a resource R_j is released by P_i, the assignment edge $R_j \rightarrow P_i$ is reconverted to a claim edge $P_i \rightarrow R_j$.

Suppose that process P_i requests resource R_j. The request can be granted only if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation of cycle in the resource allocation graph. We check for safety by using a cycle-detection algorithm.

If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state.



Suppose that T_2 requests R_2. Although R_2 is currently free, we cannot allocate it to P_2, since this action will create a cycle in the graph.



If T_1 requests R_2 & T_2 requests R_1, then a deadlock will occur.

Teacher's Signature

iii) Banker's Algorithm : In this, when a new process enters the system, it must declare the maximum no. of instances of each resource type that it may need. This no. may not exceed the total no. of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise the process must wait until some other process releases enough resources.

Several data structures must be maintained to implement the Banker's Algorithm. We need the following data structures, where 'n' is the no. of processes in the system & 'm' is the no. of resource types.

- Available : A vector of length 'm' indicates no. of available resources of each type.
- Max : An n x m matrix defines the max. demand of each process.
- Allocation : An n x m matrix defines the no. of resources of each type currently allocated to each process.

Teacher's Signature

- Need: An n×m matrix indicates the remaining resource need of each process.

Safety Algorithm: This algorithm is used for finding out whether or not a system is in a safe state. This algorithm can be described as follows:

a. Let Work & Finish be vectors of length 'm' & 'n' respectively. Initialize
[Initially, no process has finished] Work = Available
Finish[i] = false for i = 0, 1, - - - n-1

b. Find an index 'i' such that both
Finish[i] == False
Need i ≤ Work
[It means, we need to find an unfinished process whose need can be satisfied by the available resources]
If no such i exists, go to step 4.

c. Work = Work + Allocation [When an unfinished process is found, then the resources are allocated & the process is marked finished.]
Finish[i] = true
Go to step 2.

d. If Finish[i] == true for all i, then the system is in a Safe state.
[If all processes are finished, then the system is in safe state].

Teacher's Signature

Resource - Request Algorithm : In this algorithm, we determine whether requests can be safely granted.

Let $Request_i$ be the request vector for process P_i. If $Request_i[j] == K$, then process P_i wants K instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

a. If $Request_i \leq Need_i$ go to Step 2. Otherwise, raise an error condition, since the process has exceeded its max. claim.

b. If $Request_i \leq Available$, go to Step 3. Otherwise, P_i must wait, since the resources are not available.

c. Assume that resources have been allocated.

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

If the resulting resource-allocation state is safe, the transaction is completed & process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$

Eg- 5 Processes P_0, P_1, P_2, P_3, P_4
3 Resource Types A, B, C
Resource type A has 10 instances
" " B " 5 "
" " C " 7 "

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

Q1- What is the content of Need Matrix?
Ans1- Need = Max - Allocation

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

Q2- Is the system in safe state? If Yes, then what is the safe sequence?

Ans2- Applying the safety algo-

1. Work = Available = 3 3 2
   Finish = False False False False False
   0 1 2 3 4

2. For $i = 0$
   $Need_0 = 7\ 4\ 3$
   Finish[0] is false & $Need_0 > Work$
   7 4 3 3 3 2
   So P_0 must wait

2. For $i = 1$
   $Need_1 = 1\ 2\ 2$
   Finish[1] is false & $Need_1 < Work$
   1 2 2 3 3 2

   So, P_1 must be kept in safe sequence
3. Work = 332 + 200 = 5 3 2
   Finish[1] = True
   Finish = False True False False False
2. For $i = 2$
   $Need_2 = 6\ 0\ 0$
   Finish[2] is false & $Need_2 > Work$
   6 0 0 5 3 2

   So, P_2 must wait

2. For $i = 3$

   $Need_3 = 0\ 1\ 1$
   Finish[3] is false & $Need_3 < Work$
   0 1 1 5 3 2
   So, P_2 must be kept in safe sequence

Teacher's Signature

3. Work = Work + Allocation
= 5 3 2 + 2 1 1
= 7 4 3

	0	1	2	3	4
Finish =	False	Tenue	False	Tenue	False

2. For $i = 4$
$Need_4 = 4\ 3\ 1$

Finish[4] is False & $Need_4 < Work$
4 3 1 < 7 4 3

So, P_4 must be kept in safe sequence

3. Work = Work + Allocation
= 7 4 3 + 0 0 2
= 7 4 5

	0	1	2	3	4
Finish =	False	Tenue	False	Tenue	Tenue

2. For $i = 0$

$Need_0 = 7\ 4\ 3$

Finish[0] is false & $Need_0 < Work$
7 4 3 < 7 4 5

So, P_0 must be kept in safe sequence

Teacher's Signature

3. Work = Work + Allocation
= 7 4 5 + 0 1 0
= 7 5 5

	0	1	2	3	4
Finish =	True	True	False	True	True

2. For $i = 2$

$Need_2 = 6\ 0\ 0$

Finish [2] is false & $Need_2 < Work$
$6\ 0\ 0 \quad 7\ 5\ 5$

So, P_2 must be kept in a safe sequence

3. Work = Work + Allocation
= 7 5 5 + 3 0 2
= 10 5 7

	0	1	2	3	4
Finish =	True	True	True	True	True

4. Finish [i] == true for all $0 \leq i < 4$

Hence, the system is in safe state

The safe sequence is P_1, P_3, P_4, P_0, P_2.

Q3. What will happen if process P_1 requests one additional instance of resource type A & 2 instances of resource type C?

Ans3. $Request_1 = 1\ 0\ 2$ (A B C)

We use resource request algorithm,

i) $Request_1 < Need_1$

$1\ 0\ 2 < 1\ 2\ 2$

ii) $Request_1 < Available$

$1\ 0\ 2 < 3\ 3\ 2$

iii)

Process	Allocation (A B C)	Need (A B C)	Available (A B C)
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	Available
P_3	2 1 1	0 1 1	$-Request_1$
P_4	0 0 2	4 3 1	

Allocation + $Request_1$

$Need_1 - Request_1$

We must determine whether this new system state is safe. To do so, we again execute Safety Algorithm.

1. Work = 2 3 0
   Finish = False False False False False
   0 1 2 3 4

2. For $i = 0$

   $Need_0$ = 7 4 3
   Finish[0] is false & $Need_0$ <= Work
   7 4 3 2 3 0

   So, P_0 must wait.

2. For $i = 1$

   $Need_1$ = 0 2 0
   Finish[1] is false & $Need_1$ < Work
   0 2 0 2 3 0

   So, P_1 must be kept in safe sequence

3. Work = Work + Allocation
   = 2 3 0 + 3 0 2
   = 5 3 2
   0 1 2 3 4
   Finish = False True False False False

2. For $i = 2$

$Need_2 = 6\ 0\ 0$
$Finish[2]$ is false & $Need_2 > Work$
6 0 0   5 3 2

So, P_2 must wait.

2. For $i = 3$

$Need_3 = 2\ 1\ 1$
$Finish[3]$ is false & $Need_3 < Work$
0 1 1   5 3 2

So, P_3 must be kept in safe sequence

3. Work = Work + Allocation
= 5 3 2 + 2 1 1
= 7 4 3

	0	1	2	3	4
Finish =	False	True	False	True	False.

2. For $i = 4$

$Need_4 = 4\ 3\ 1$
$Finish[4]$ is false & $Need_4 < Work$
4 3 1   7 4 3

So, P_4 must be kept in safe sequence

3. Work = Work + Allocation
= 7 4 3 + 0 0 2
= 7 4 5

	0	1	2	3	4
Finish =	False	True	False	True	True

2. For $i = 0$

$Need_0 = 7\ 4\ 3$
Finish[0] is false & $Need_0 < Work$
$7\ 4\ 3 \quad 7\ 4\ 5$

So, P_0 must be kept in safe sequence

3. Work = Work + Allocation
= 7 4 5 + 0 1 0
= 7 5 5

	0	1	2	3	4
Finish =	True	True	False	True	True

2. For $i = 2$

$Need_2 = 6\ 0\ 0$
Finish[2] is false & $Need_2 < Work$
$6\ 0\ 0 \quad 7\ 5\ 5$

So, P_2 must be kept in safe sequence.

3. Work = Work + Allocation
= 7 5 5 + 3 0 2
= 10 5 7

	0	1	2	3	4
Finish =	True	True	True	True	True

4. Finish [i] = true for $0 \leq i \leq 4$
Hence, the system is in safe state

Safe Sequence is P_1, P_3, P_4, P_0, P_2

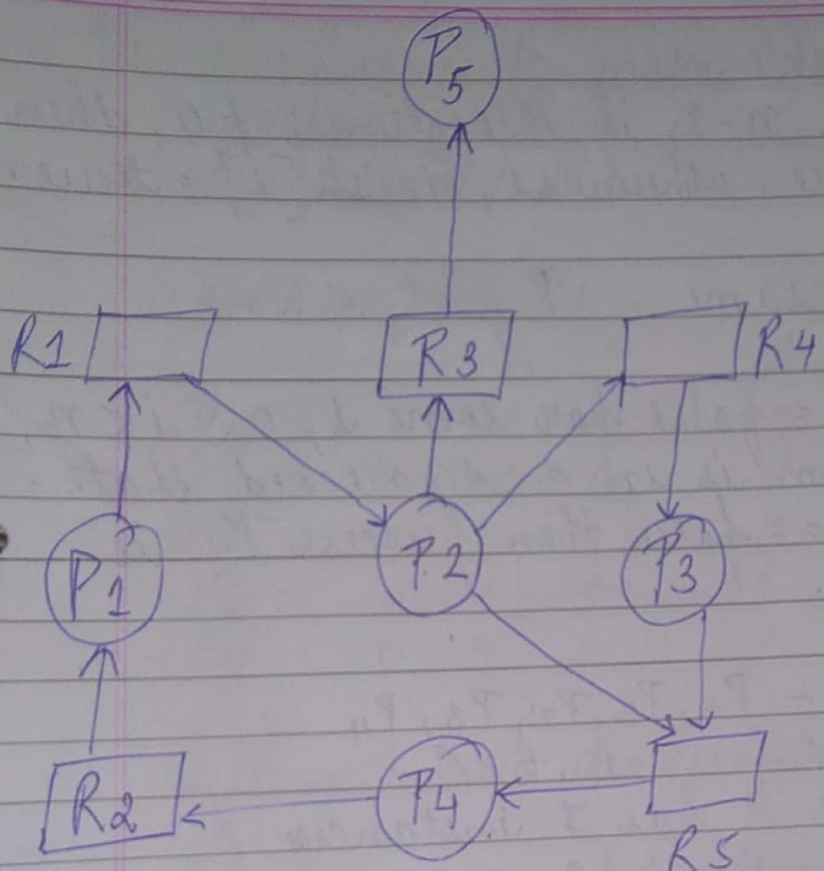
Hence the new system state is safe, so we can immediately grant the request for process P_1.

- Deadlock Detection :

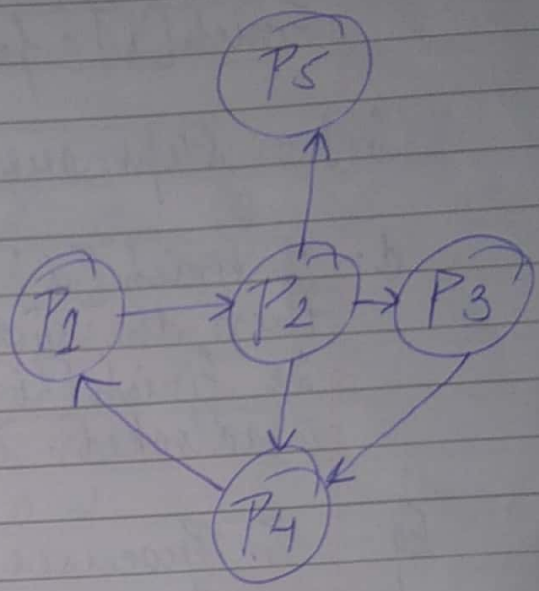
i) Single Instance of each resource type :

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of resource allocation graph called a wait for graph. We obtain this graph from the resource allocation graph by removing the resource nodes & collapsing the appropriate edges.

Teacher's Signature



Resource Allocation Graph



Corresponding Wait-for Graph

A deadlock exists in the system if & only if wait-for graph contains a cycle.

ii) Several Instances of Resource type:

This algorithm employs several data structures that are similar to those used in the Banker's algorithm.

a) Available
b) Allocation
c) Request

Teacher's Signature

a. Work = Available
For $i = 0, 1, \ldots, n-1$, if Allocation$_i \neq 0$, then
Finish[i] = false, otherwise, Finish[i] = true.

b. & c. Steps are same. Request$_i \leq$ Work

d. If Finish[i] == false for some i, $0 \leq i < n$, then the system is in a deadlocked state.
If Finish[i] == false, then process P_i is deadlocked.

Eg- 5 Processes - P_0, P_1, P_2, P_3, P_4
3 Resource types - A, B, C
Resource type A has 7 instances
" " B " 2 "
" " C " 6 "

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

Q1. Determine if the system is deadlocked!
$\langle P_0, P_2, P_3, P_1, P_4 \rangle$
System is not in deadlocked state.

Teacher's Signature

Q2. Now process P_2 makes a request for additional instance of type C. Is the system now deadlocked?

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

System is now deadlocked.

- Recovery from Deadlock:

i) Process Termination: To eliminate deadlocks by aborting a process, we use one of the two methods:

a. Abort all deadlocked processes.
b. Abort one process at a time until the deadlock cycle is eliminated.

ii) Resource Preemption: To eliminate deadlock, we successively preempt resources from processes & give these resources to other processes until the deadlock cycle is broken.

a. Selecting a victim : Which resources & which processes are to be preempted? We determine the order of preemption to minimize cost.

b. Rollback : If we preempt a resource from a process, what should be done with that process? Clearly, it cannot continue with its normal execution; it is missing some needed resource. We must rollback the process to some safe state & restart it from that state.

c. Starvation : How do we ensure that starvation will not occur? That is, how can we guarantee that resources will not always be preempted from the same process?

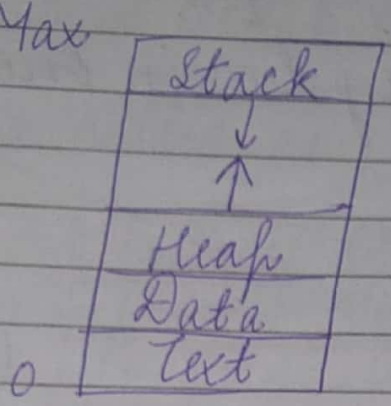
It may happen that same process is always picked as a victim. As a result, this process never completes its designated task, a starvation situation occurs in the system. So, we must ensure that a process can be picked as a victim only a finite number of times.

Teacher's Signature

Unit-2

Date: / / Page no:

Process: A process is a program in execution. Also known as the text section.

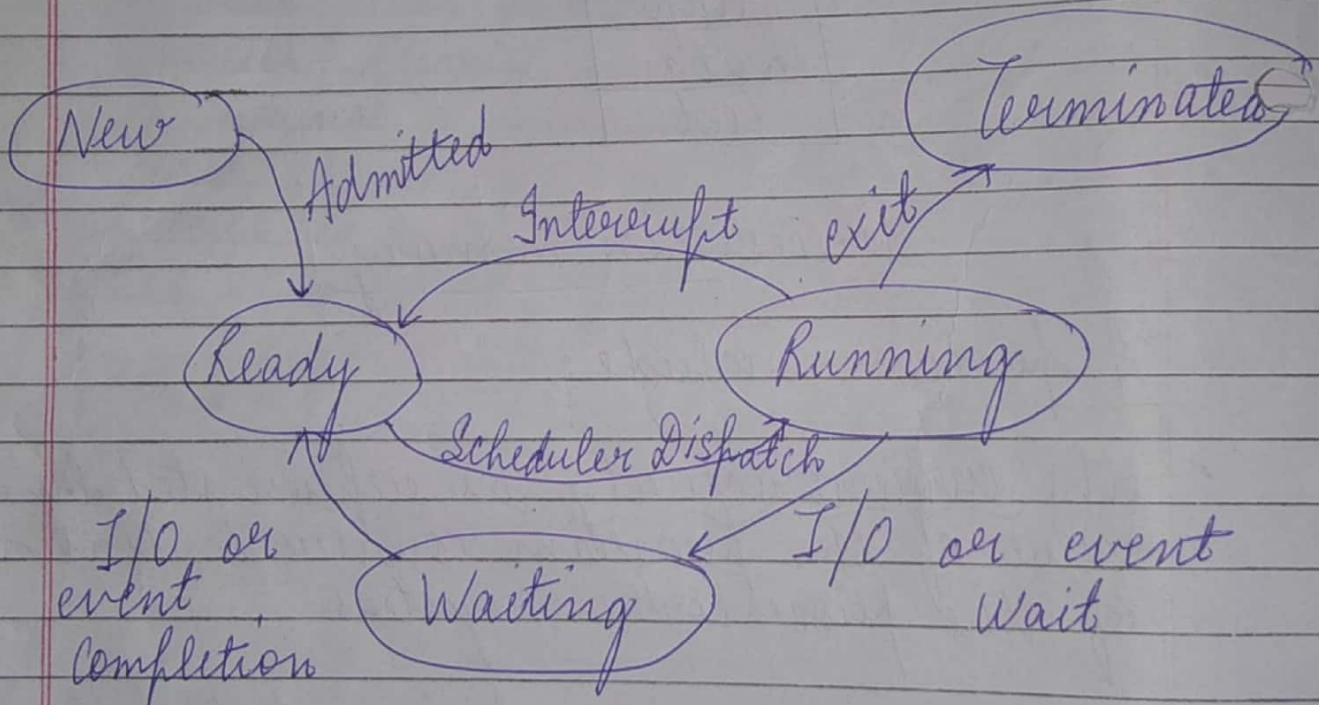


Process in memory

A process include:

- the current activity as represented by the value of the program counter & content of the processor's registers.
- the process stack which contains the temporary data.
- the data section which contains global variables.
- the heap which is the memory i.e. dynamically allocated during process run time.

Process State: As a process executes, it changes state. The state of a process is defined by the current activity of that process.
Each process may be in one of the following states:

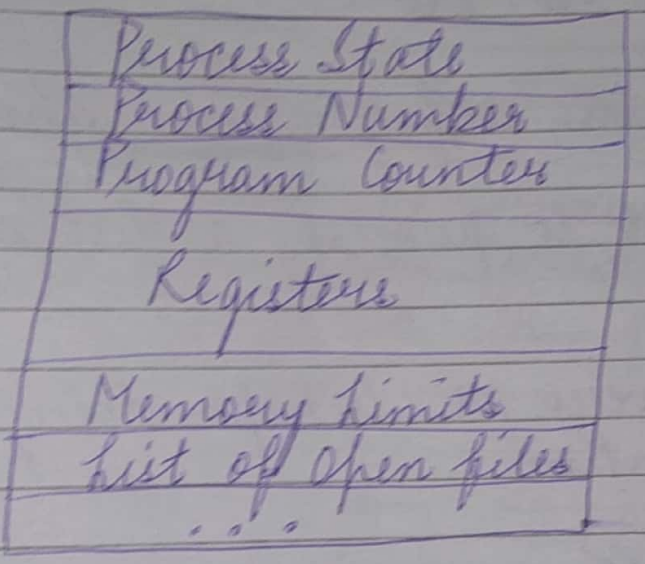


Process State

- New: the process is being created
- Running: Instructions are being executed.
- Waiting: the process is waiting for some event to occur.
- Ready: the process is waiting to be assigned to a processor.

- Terminated : the process has finished execution.

Process Control Block : Each process is represented in OS by PCB also called as Task Control Block



PCB

- Process State
- Program Counter : indicates the address of next instruction to be executed for the process.
- CPU Registers : the registers may vary in number & type. When an interrupt occurs, this state information along with the PC information is saved, to allow the process to be continued correctly afterward.

- CPU Scheduling information :
- Memory Management information :
- Accounting information : this info. include the amount of CPU used, time limits, process no. etc.
- I/O status information : this include the list of I/O devices allocated to process, a list of open files etc.

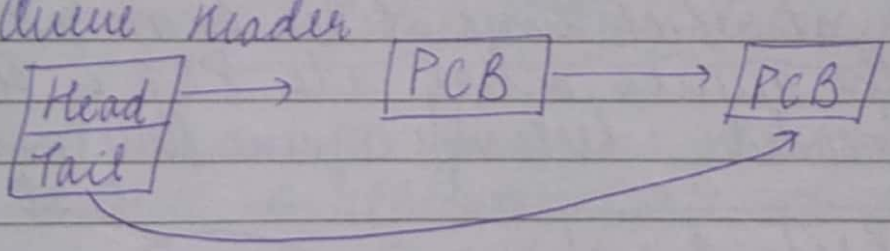
PCB serves as a repository for any information.

Thanks :)

Process Scheduling : The act of determining or selecting an available process from a set of several available processes for program execution on CPU.

- Scheduling Queues :

1. As processes enter the system, they are put into a job queue.
2. The processes that are residing in main memory & are ready to execute are placed in the ready queue. This queue is stored as a linked list.



3. The list of processes waiting for a particular I/O device is called a device queue.

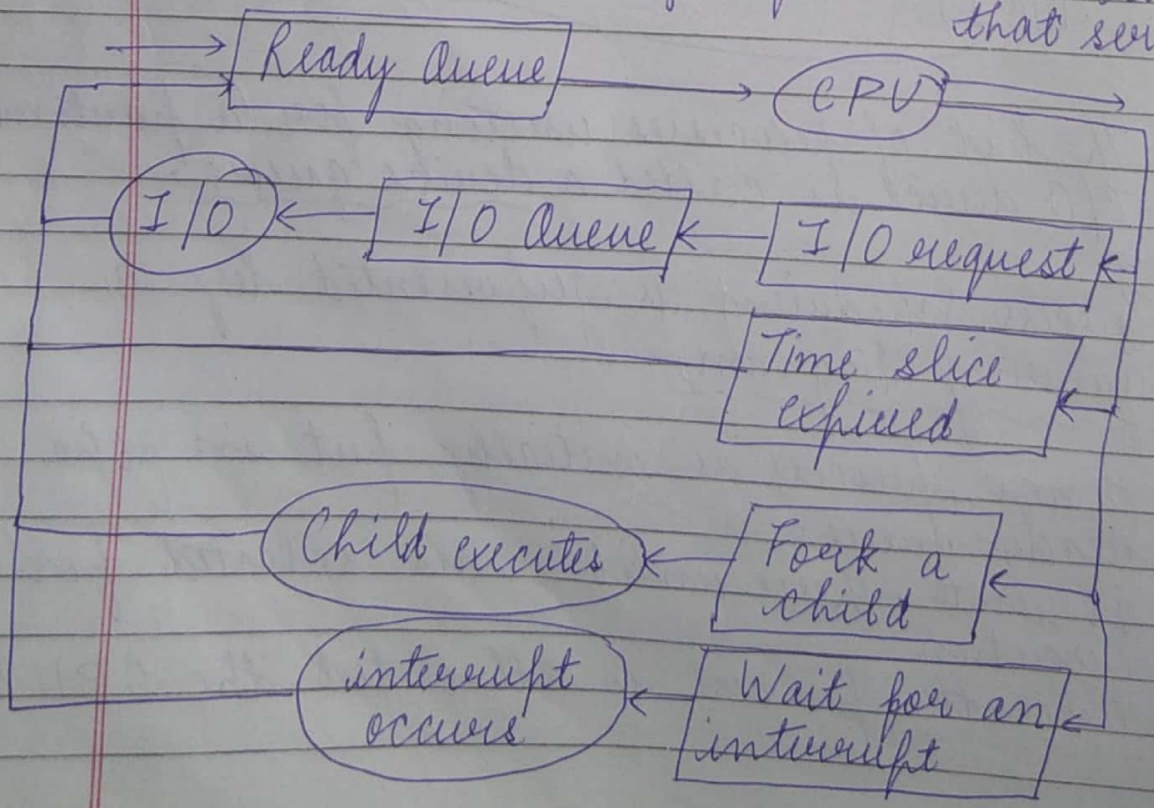
Process Scheduling is represented by a Queueing diagram.

1. A new process is initially put in the ready queue.
2. It waits there until it is selected for execution.
3. 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 a new subprocess & wait for the subprocess's termination.
- The process could be removed forcibly from CPU as a result of an interrupt, & be put back in the ready queue.

A process continues this cycle until it terminates, at which time it is removed from all the queues & has its PCB & resources deallocated. Rectangle → queue Circle → resources that serve queue



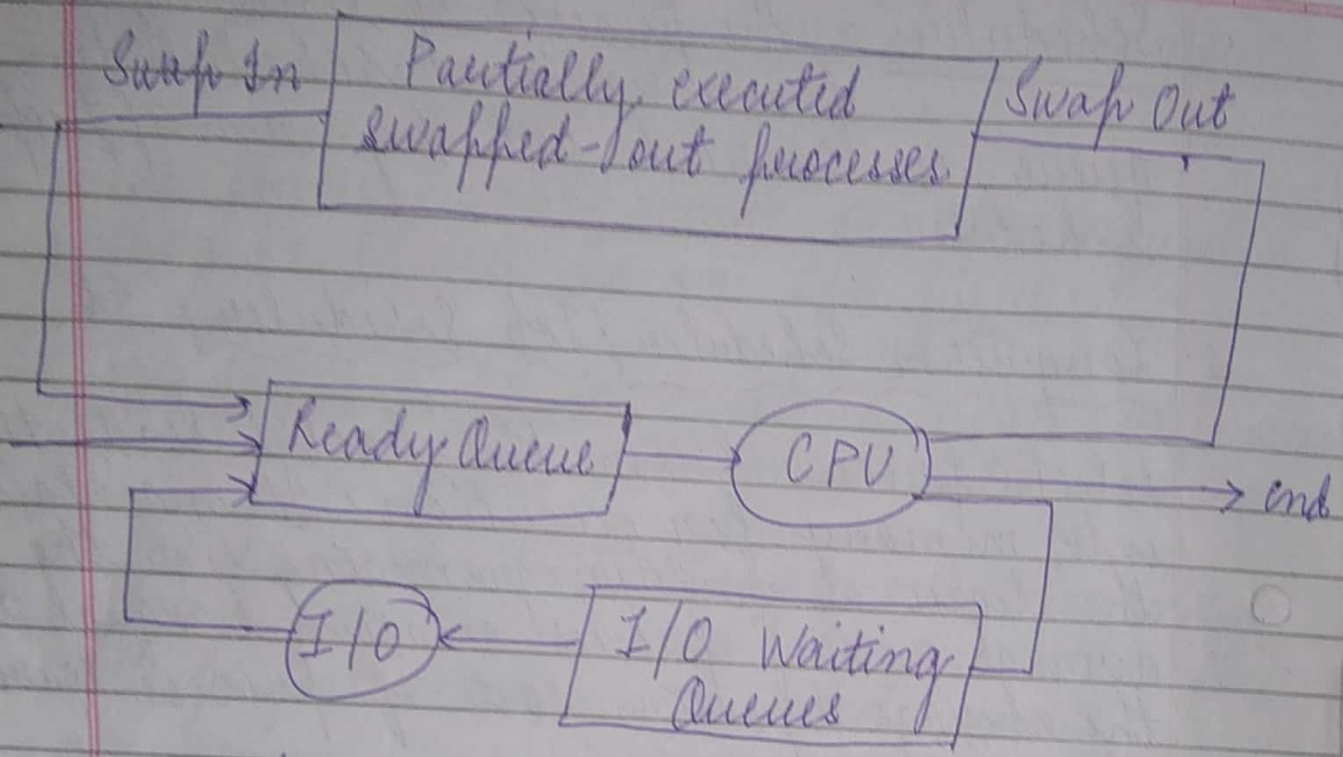
Date: / / Page no:

Schedulers: The process of selecting process from various scheduling queue is carried out by an appropriate scheduler.

1. Long term Scheduler / Job Scheduler: It selects process from the pool & loads them into memory for execution. It maintain the degree of multiprogramming i.e. the average rate of process creation is equal to the average departure rate of process leaving the system.

2. Short term Scheduler / CPU Scheduler: It selects from the processes that are ready to execute & allocates CPU to one of them. Aim is to enhance CPU performance & increase process execution rate.

3. Medium Term Scheduler: This scheduler removes the process from memory, reducing the degree of multiprogramming. Later, the process can be reintroduced into memory & its execution can be continued where it is left off. This scheme is called swapping.



Addition of medium-term scheduling to the Queueing diagram

- Context Switch: Switching the CPU into another process requires performing a state save of the current process & a state restore of a different process. This is known as a Context Switch.

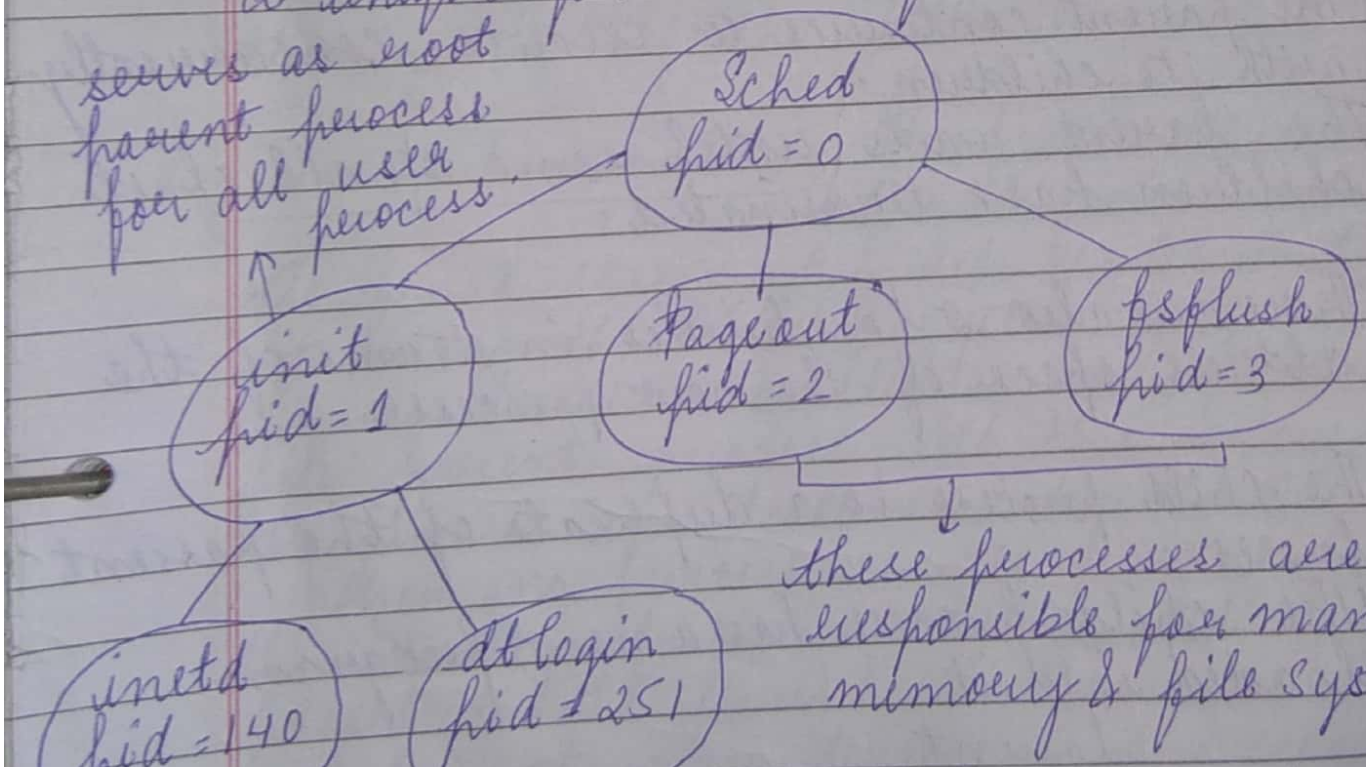
The context of a process is represented in the PCB of a process; it includes the value of the CPU Registers, the process state & memory management information. When a context switch occurs, the kernel saves the context of the old process in its PCB & loads the saved context of new process scheduled to run.

Date: / / Page no:

Operations on process :

1. **Process Creation:** A process may create several new processes, via a create-process system call, during the course of execution.

The creating process is called a parent process & the created new processes are called the children of that process. The processes are identified according to a unique process identifier or PID.



A Tree of process on a Solaris system

(Unix)

Date: / / Page no:

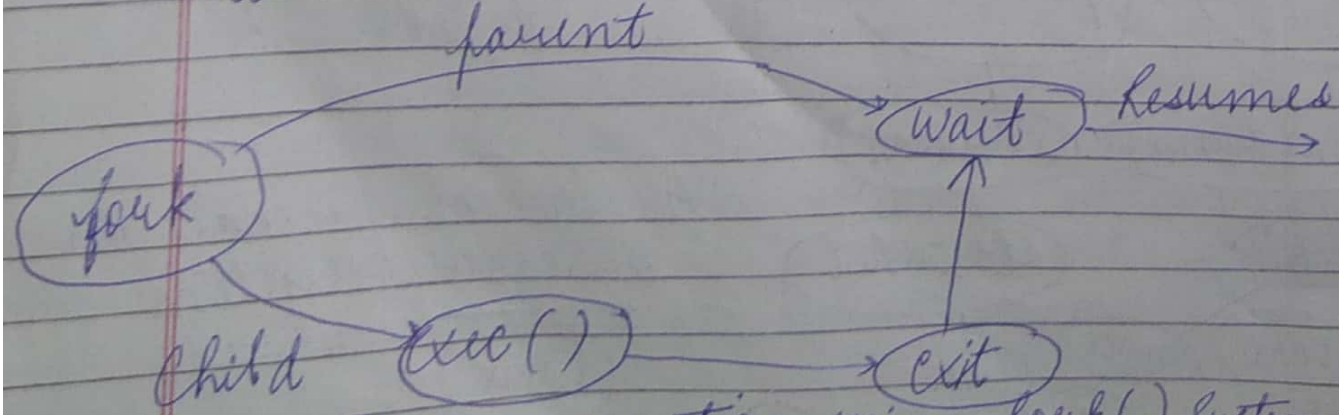
inetd is responsible for networking services such as telnet & ftp.
dtlogin is the process representing a user login screen.

The parent may have to partition its resources among its children or it may be able to share some resources among several of its children.
When a process creates a new process, two possibilities exist in terms of execution:

i) The parent continues to execute concurrently with its children.
ii) The parent waits until some or all of its children have terminated.

There are also 2 possibilities in terms of the address space of the new process:

i) The child process is a duplicate of the parent process.
ii) The child process has a new program loaded into it.



Process creation using fork() System call

→ A new process is created by the fork() system call. The new process consists of copy of address space of original process.

→ This mechanism allows the parent process to communicate easily with its child process.

→ The exec() system call loads a binary file into memory & starts its execution.

→ The parent issue a wait() system call to move itself off the ready queue until the termination of the child.

2. Process Termination: The process terminates when it finishes executing its final statement & ask the OS to delete it using the exit() system call. The process returns a status value (integer) to its parent process. All the resources of the process are deallocated by the OS.

When one process creates a new process, the identity of the newly created process is passed to the parent.

A parent may terminate the execution of one of its children for a variety of reasons:

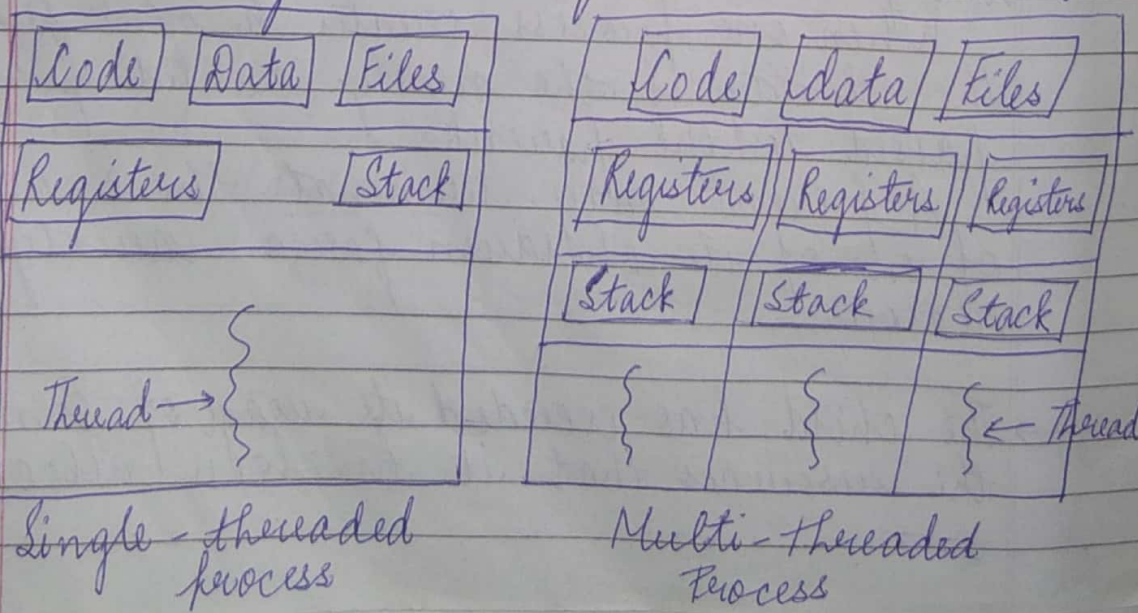
→ The child has exceeded its usage of some of the resources that it has been allocated.

→ The task assigned to the child is no longer required.
→ The parent is exiting & the OS does not allow a child to continue if its parent terminates.

If a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon referred to as Cascading Termination.

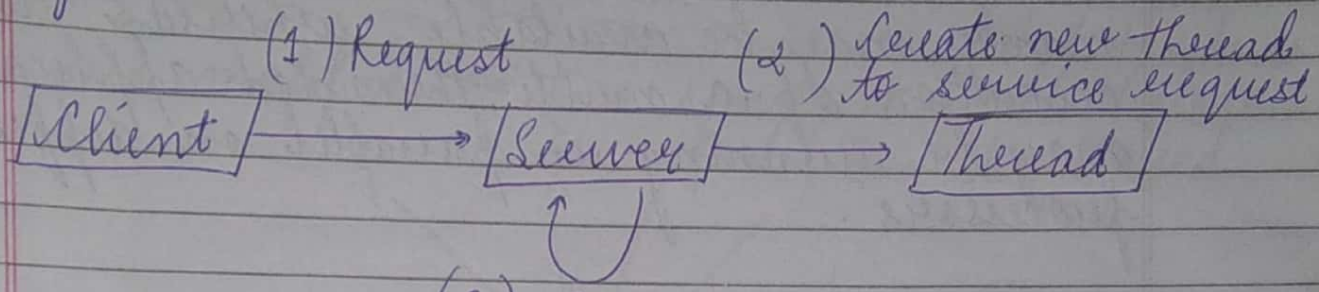
Thread:

A thread is a basic unit of CPU utilization, which comprises of a thread ID, a PC, a set of registers & a stack. It is also known as lightweight process.
As each thread has its own independent resource for process execution, multiple processes can be executed parallely by increasing number of threads.



Date: / / Page no:

Eg - Web Server



Multithreaded server architecture

Benefits :

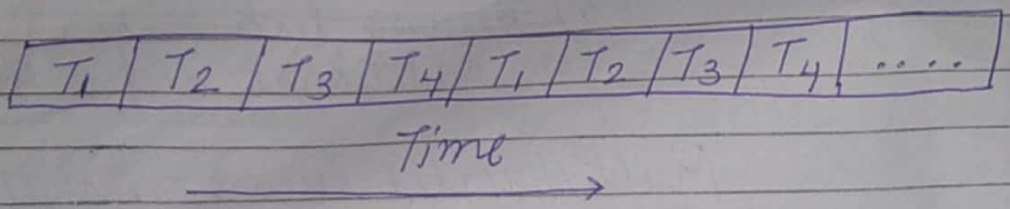
1. Responsiveness : Multithreading allows a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

2. Resource Sharing : Threads share common code, data & other resources which allow an application to have several different threads of activity within the same address space.

3. Economy : Creating & managing threads is much faster than performing the same task for processes.

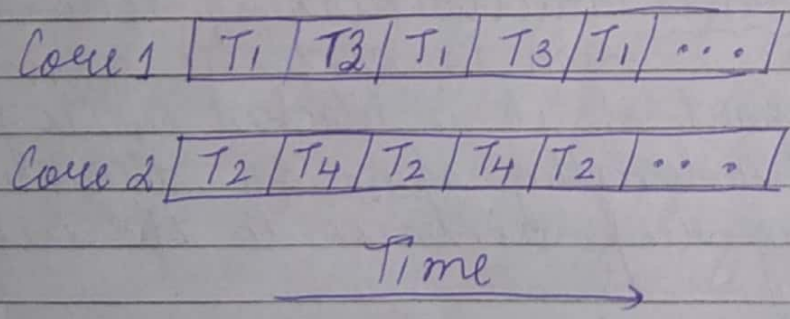
4. Scalability : A single threaded process can

only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be running in parallel on different processors.



(a)

Concurrent execution on a single-core system



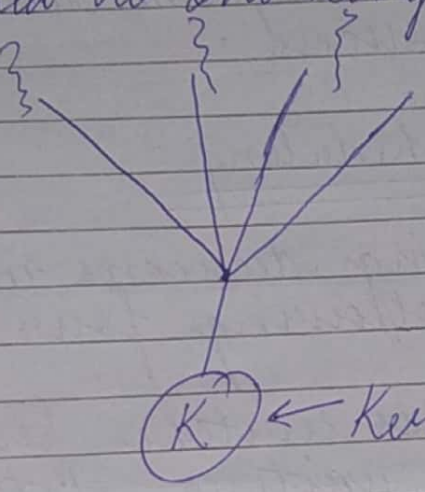
(b)

Parallel execution on a multicore system

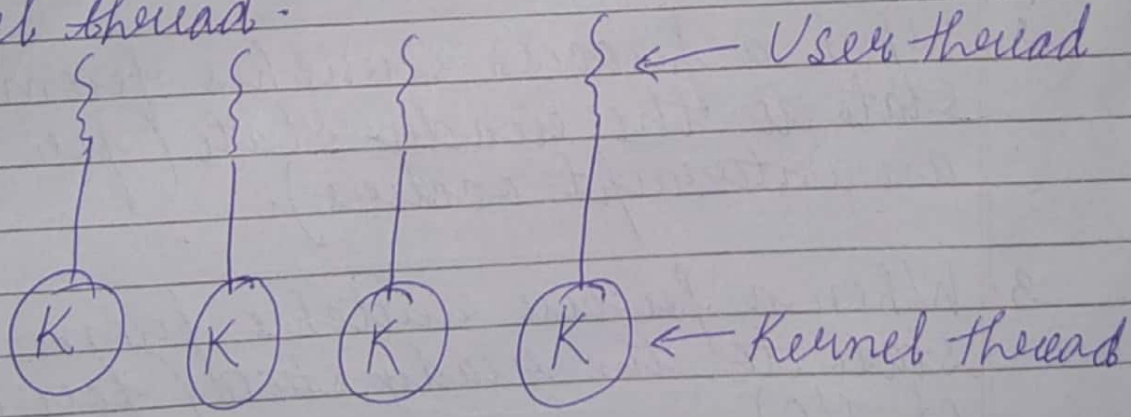
Multithreading Models: 2 types of threads:

1. User threads: They are supported above the kernel, without the kernel support.
2. Kernel threads: They are supported & managed directly by the O.S.

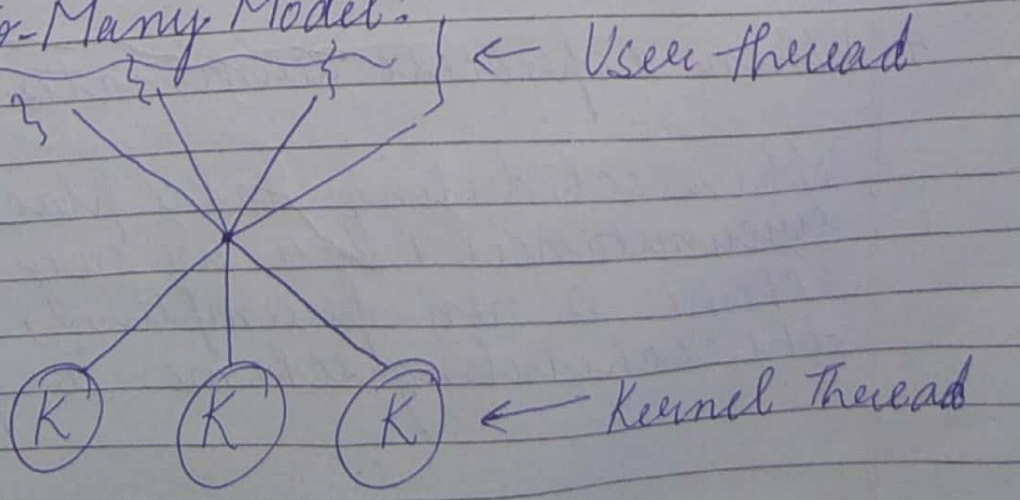
i) Many - to - One Model : In this, many user-level threads are mapped to one single kernel thread.



ii) One-to-One Model: In this, each user thread maps to a kernel thread.



iii) Many-to-Many Model:



It multiplexes many user-level threads to a smaller or equal number of kernel threads. Blocking kernel system calls do not block the entire process.

Preemptive Scheduling:

CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for I/O request or invocation of wait for termination of one of child processes).
2. When a process switches from the running state to the ready state (for eg - when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for eg - completion of I/O).
4. When a process terminates.

When scheduling take place only under circumstances 1 & 4, we say the scheduling scheme is non-preemptive, otherwise the scheduling scheme is preemptive.

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

Preemptive Scheduling : In this, the tasks are assigned with priorities. The running task is interrupted by the task that has a higher priority & resumed later when the priority task has finished its execution.

Dispatcher : The dispatcher is the module that gives control of CPU to the process selected by the short-term scheduler. This function involves :

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart the program.

The time it takes for the dispatcher to stop one process & start another running is known as the dispatch latency.

The dispatcher should be as fast as possible, since it is invoked during every process switch.

Scheduling Criteria:

1. CPU Utilization: To keep the CPU as busy as possible.
2. Throughput: It is the number of processes completed per unit time.
3. Turnaround time: The interval from time of submission of process to time of completion is the turnaround time.
4. Waiting time: It is the sum of periods spent waiting in the ready queue.
5. Response time: It is the time from the submission of request until the first response is produced.

Scheduling Algorithms:

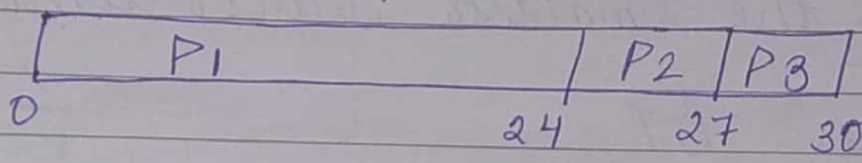
1. First Come First Serve Scheduling (FCFS):
With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation is easily managed with a FIFO queue.

Eg-

Process	Burst Time	AT	C.T.	WT	TAT
P1	24	0	24	0	24
P2	3	0	27	24	27
P3	3	0	30	27	30

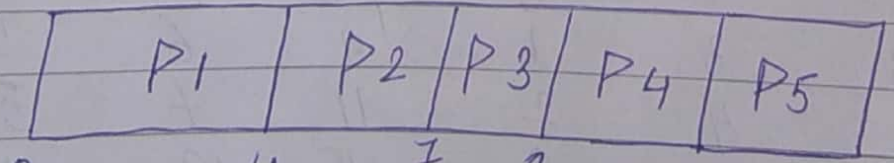
$TAT = WT + BT$ or $CT - AT$

Grantt Chart is a bar chart that illustrates a particular schedule.



Eg-

Process	AT	BT	CT	TAT	WT
P1	0	4	4	4	0
P2	1	3	7	6	3
P3	2	1	8	6	5
P4	3	2	10	7	5
P5	4	5	15	11	6



$$ATAT = (4+6+6+7+11)/5 = 6$$

$$AWT = (0+3+5+5+6)/5 = 3$$

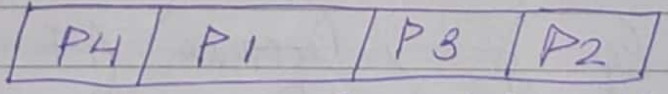
Convoy Effect: All the other process wait for the one big process to get off the CPU. When the process has high burst time then all the process are going to starve or wait.

2. Shortest Job First Scheduling (SJF):

The CPU is assigned to the process that has the smallest burst time.

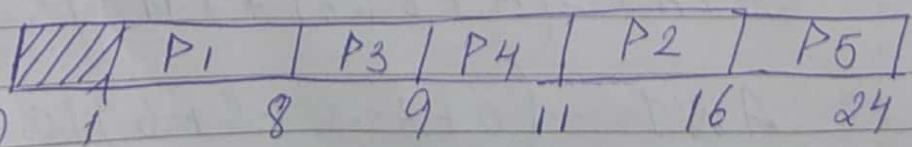
Eg-

Process	BT
P1	6
P2	8
P3	7
P4	3



Eg-

Process	AT	BT	CT	TAT	WT
P1	1	7	8	7	0
P2	2	5	16	14	9
P3	3	1	9	6	5
P4	4	2	11	7	5
P5	5	8	24	19	11



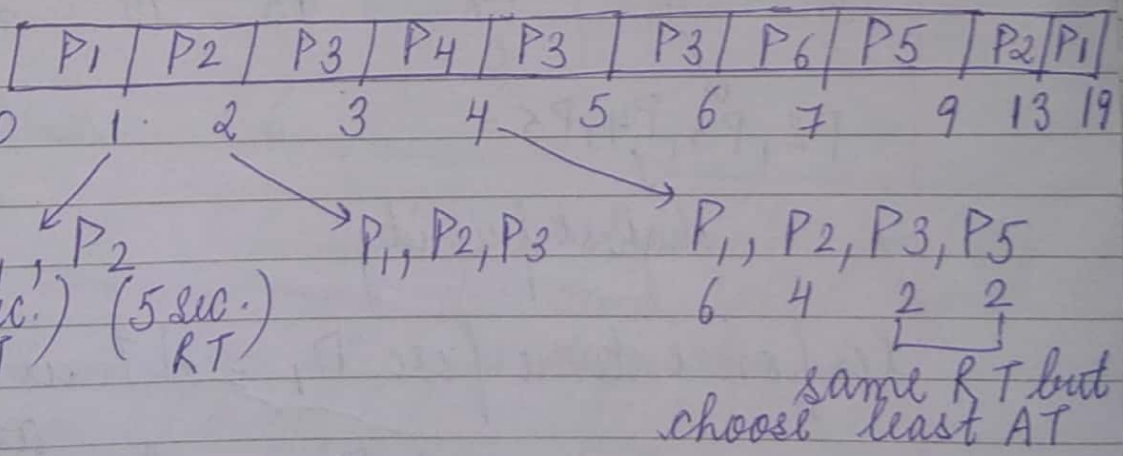
Teacher's Signature

3. Shortest Remaining Time First Scheduling (SRTF) :

Preemptive SJF Scheduling is called SRTF.

Eg -

Process	AT	BT	CT	TAT	WT
P1	0	7	19	19	12
P2	1	5	13	12	7
P3	2	3	6	4	1
P4	3	1	4	1	0
P5	4	2	9	5	3
P6	5	1	7	2	1



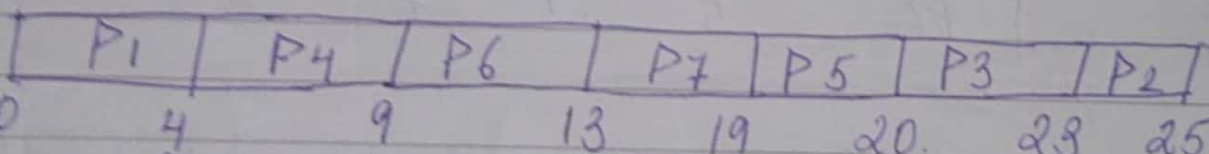
4. Priority Scheduling : A priority is associated with each process & the CPU is allocated to the process with the highest priority. Equal priority process are scheduled in FCFS order.

Smaller the CPU burst, larger the priority & vice-versa.

Non-preemptive priority scheduling - (Response time)

Eg:

Process	Priority	AT	BT	CT	TAT	WT	RT
P1	10 (2)	0	4	4	4	0	0
P2	6	1	2	25	24	22	22
P3	4	2	3	23	21	18	18
P4	10	3	5	9	6	1	1
P5	8	4	1	20	16	15	15
P6	12 (H)	5	4	13	8	4	4
P7	9	6	6	19	13	7	7



P2, P3, P4, P5

Highest priority

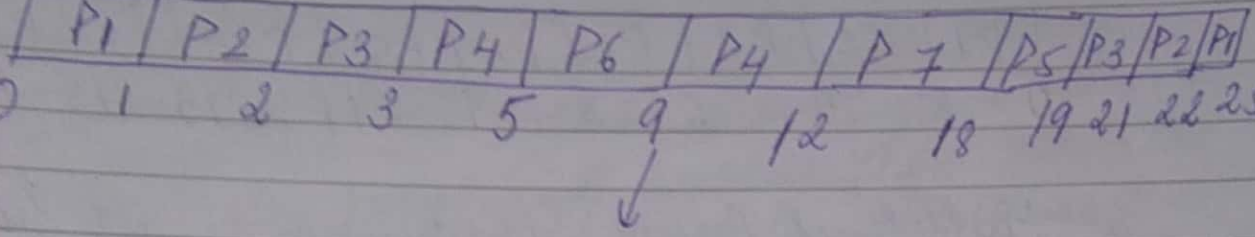
Response time for P_1 = AT is 0 & arrived at 0 = 0

" " " P2 = Arrived at 23 & AT is 1 = 23 - 1 = 22

Preemptive priority scheduling:

Same example as previous Scheduling

Eg:-



All processes are available

	(CT-AT)	(TAT-BT)	
CT	TAT	WT	RT
25	25	21	0
22	21	19	0
21	19	16	0
12	9	4	0
19	15	14	14 (18 − 4)
9	4	0	0
18	12	6	6

scheduled time → 18, AT → 4

5. Round-Robin Scheduling: A small unit of time, called time quantum or time slice is defined.

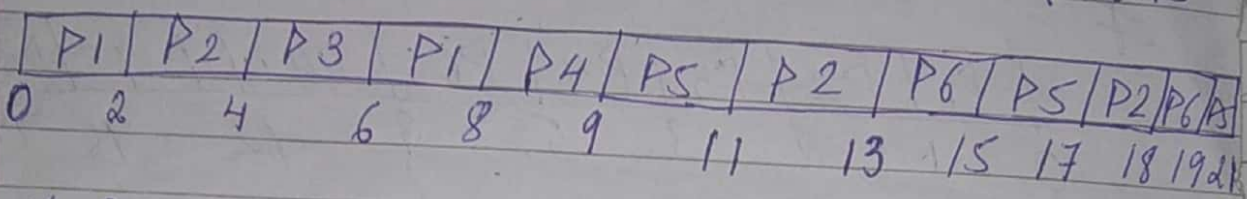
Criteria (TQ+AT) (CT-AT) (TAT-BT)

Eg- TQ

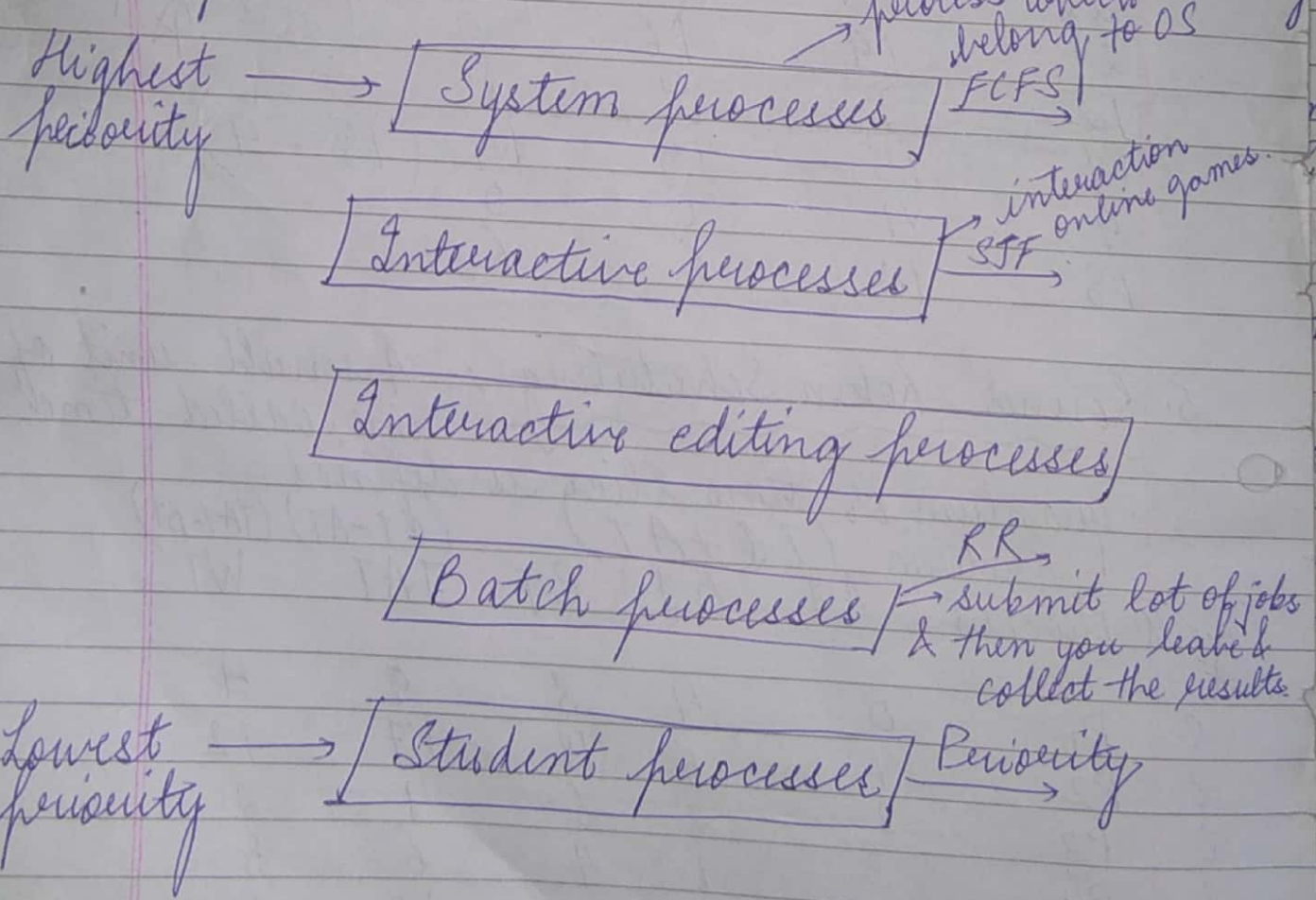
Process	AT	BT	CT	TAT	WT
P1	0	4	8	8	4
P2	1	5	18	17	12
P3	2	2	6	4	2
P4	3	1	9	6	5
P5	4	6	21	17	11
P6	6	3	19	13	10

Teacher's Signature

Queue P1 P2 P3 P1 P4 P5 P2 P6 P5 P2 P6 P5



6. Multilevel Queue Scheduling : It partitions the ready queue into several separate queues. The processes are permanently assigned to one queue. Each queue has its own scheduling algorithm:



Teacher's Signature

7. Multilevel Feedback Queue Scheduling :

It allow a process to move between queues. If a process uses too much CPU time, it will be moved to a lower priority queue. A process that waits too long in a lower priority queue is moved to a higher priority queue & this prevents starvation.

A multilevel feedback queue scheduler is defined by the following parameters :

- The no. of queues
- The scheduling algorithm for each queue
- The method used to determine when to upgrade a process to higher priority queue.
- The method used to determine when to demote a process to a lower priority queue.
- The method used to determine which queue a process will enter when that process needs a service.

Cooperating & Independent process :

A process is independent if it cannot affect or be affected by the other processes executing in the system. An independent

process does not share data with any other process.

A process is cooperating if it can affect or be affected by the other processes executing in the system. A cooperating process share data with other process.

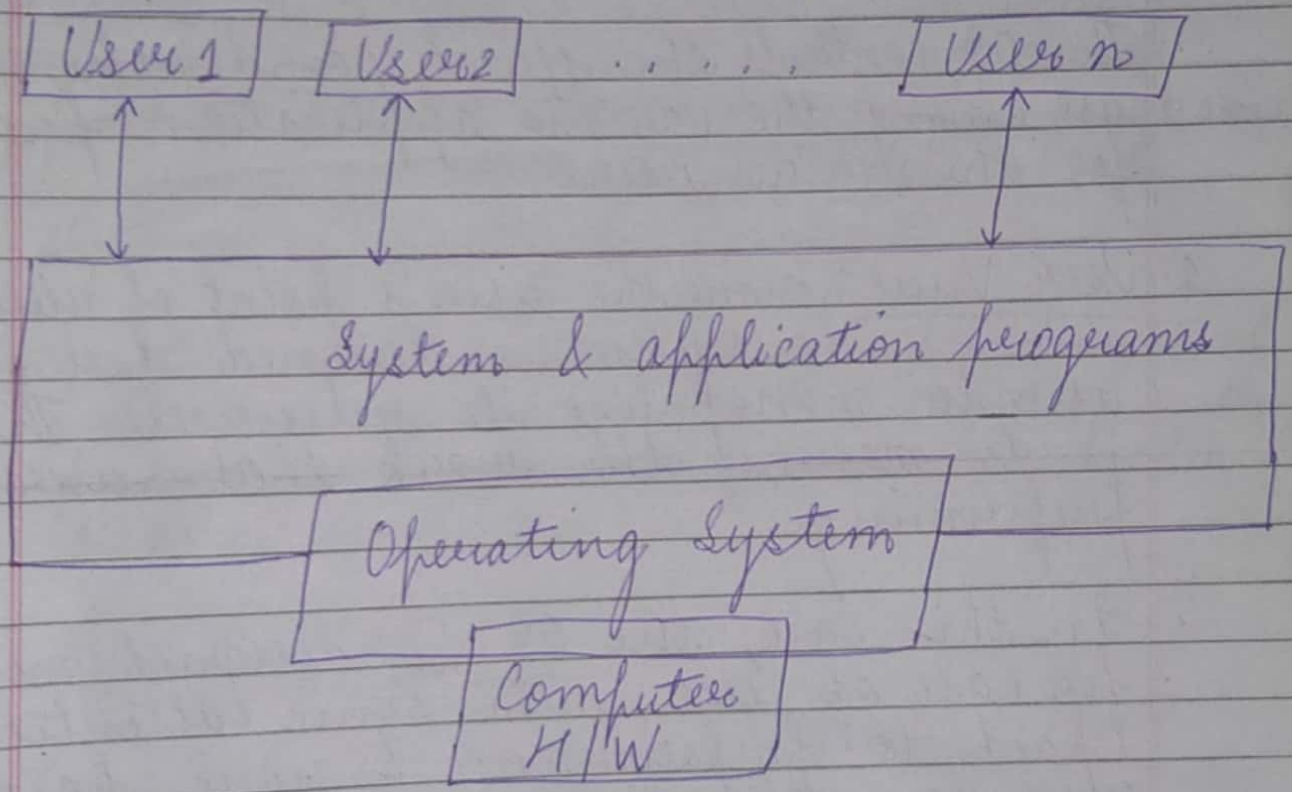
Teacher's Signature

Unit - 1

Date: / / Page no:

Operating System:

- It is a program that manages the computer hardware.
- It acts as an intermediary between the computer user & the computer hardware.



Abstract view of component of computer system

A computer system can be divided roughly into 4 components: the hardware, the O.S, the application programs & the user.

- The H/W — the CPU, memory & the I/O

devices provides the basic computing resources for the system.

- The application programs define the ways in which these resources are used to solve user's computing problems.
- The OS controls the H/w & coordinates its use among the various application programs for the various users.

1. User View: From the user's point of view, a system is disigned for one user to monopolize its resources. The goal is to maximize the work that user is performing.
   - In this case, the OS is designed mostly for ease of use, with some attention paid to performance & none paid to resource utilization.
   - Performance is important to user but such system are optimized for the single user experience rather than the requirements of multiple users.
2. System View: From the computer's point of view, an OS can be viewed as a resource allocator.

Date: / / Page no:

- A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space & so on. The OS acts as manager of these resources.
- Facing numerous & possibly conflicting requests for resources, the OS decide how to allocate them to specific programs & users so that it can operate the system efficiently & fairly.
- An OS is a control program. A control program manages the execution of user programs to prevent errors & improper use of computer.

Evolution of OS:

1. Simple Batch System:

- In this type of system, there is no direct interaction b/w user & computer.
- The user has to submit written on cards or tape to Computer Operator. Then computer places a batch of several jobs on input device.
- Jobs are batched together according to requirement & then the monitor manages the execution of each program in batch.

Date: / / Page no:

2. Multiprogramming System:

- In this, the OS picks up & begins to execute one of the jobs from memory & when this job is completed, it switches to another job.
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of CPU scheduling.
- The prime focus is to maximize the CPU usage.
- Time sharing systems are an extension of multiprogramming systems. The prime focus is on minimizing the time.

3. Multiprocessor System:

- Tasks are executed concurrently & parallelly in different processors. Advantages are enhanced performance, increased throughput (by increasing the no. of processors, we expect to get more work done in less time)

4. Desktop System: These system opt for maximizing user convenience & responsiveness. These include PCs running Windows & Macintosh.

Date: / / Page no:

5. Distributed OS: It manages a group of independent computers & makes them appear to be a single computer. Advantages are fast processing.

2 types:

a. Client - Server System: Centralized systems act as server systems to satisfy requests generated by client systems.

b. Peer to peer system: It refer to the distributed computer architecture that is designed for sharing resources, by direct exchange, rather than requiring a central coordinating server.

6. Clustered Systems: They gather together multiple CPUs to accomplish task or work. The clustered computers share storage & are closely linked via LAN networking. Clustered system use SANs (storage area networks) which allow easy attachment of multiple hosts to multiple storage units.

7. Real Time OS: it is an OS that guarantees a certain capability within a specified time constraint.

Classified into 3 categories:

i) Hard Real time: Degree of tolerance for missing deadline is negligible. A missed deadline can result in catastrophic failure of the system.

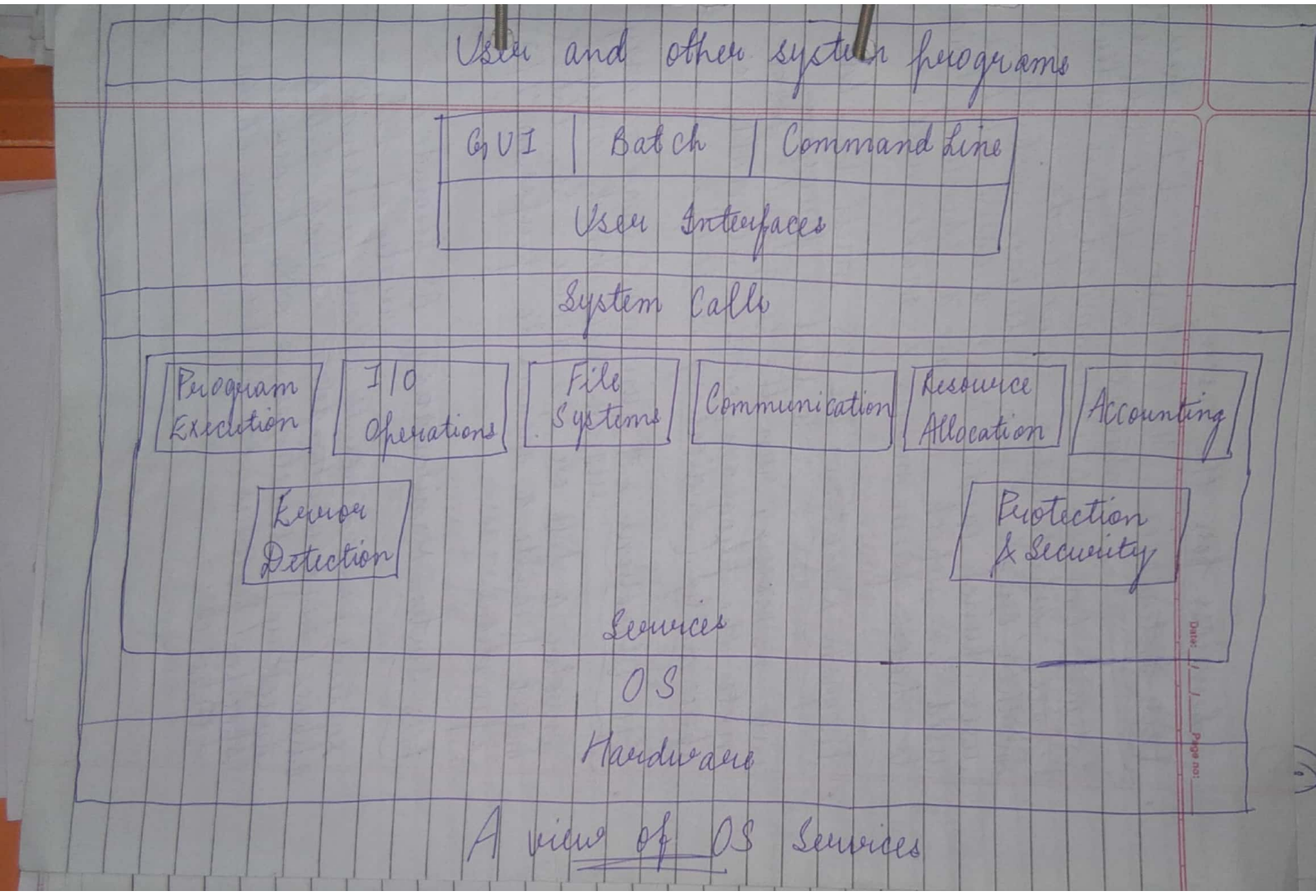
ii) Firm Real time: Missing a deadline might result in an unacceptable quality reduction but may not lead to failure of the system.

iii) Soft Real time: Deadlines may be missed occassionally.

8. Handheld Systems: These include PDAs (Personal Digital Assistants) such as cellular telephones with connectivity to internet. Eg - mobile phone, pagers etc.

Operating System Services:

OS services are provided for the convenience of the programmer, to make the programming task easier.



A view of OS Services

Services for the users:

- **User Interfaces**: All OS have a UI. These interface can take several forms. One is CLI (Command line interface) which uses commands. Another one is Batch interface which uses punch cards. Now, a days GUI is used in which we have a pointing device to choose from menus & make selection.

- **Program Execution**: The OS must be able to load a program into memory, run the program & terminate the program, either normally or abnormally (indicating error).

- **I/O Operations**: A running program may require I/O, which may involve a file or I/O device. So, the OS must provide a means to perform I/O operations.

- **File System Manipulation**: OS provide a variety of file systems, sometimes to allow personal choice (read, write, delete, search file & directory), sometimes to ~~allo~~ provide specific features or performance characteristics.

- Communications: Communications may be implemented via shared memory or through message passing in which packets of information are moved b/w processes by OS (process exchange info. with another process).

- Error Detection: The OS needs to be aware of possible errors such as errors in CPU & memory H/W, in I/O device, in user program etc. For each type of error, the OS should take appropriate action to ensure correct & consistent computing.

Services for ensuring the efficient operating of system:

- Resource Allocation: When there are multiple users or multiple processes running at the same time, resources must be allocated to each of them. In determining how best to use the CPU, OS have CPU scheduling routines that take into account all the factors.

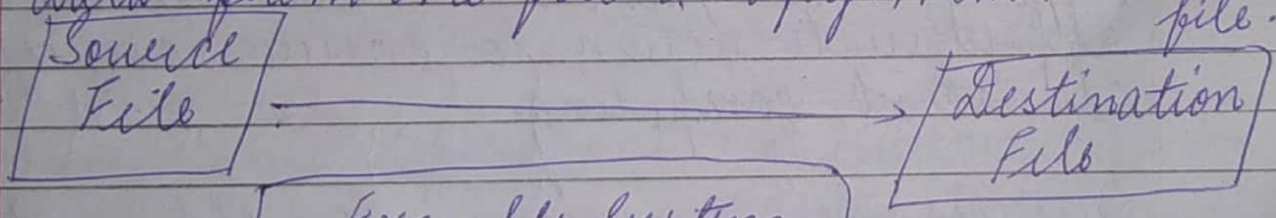
- Accounting: Keep track of system activity & resource usage for optimizing future performance.

Date: / / Page no:

- **Protection & Security**: Protection involves ensuring access to system resources is controlled. Security from outsiders is important. Security include authentication.

System Calls: It provide an interface to the services made available by an OS. Generally written in C or C++. Each OS has its own name for each system call. Eg - Reading data from one file & copy them to another file.

Eg-

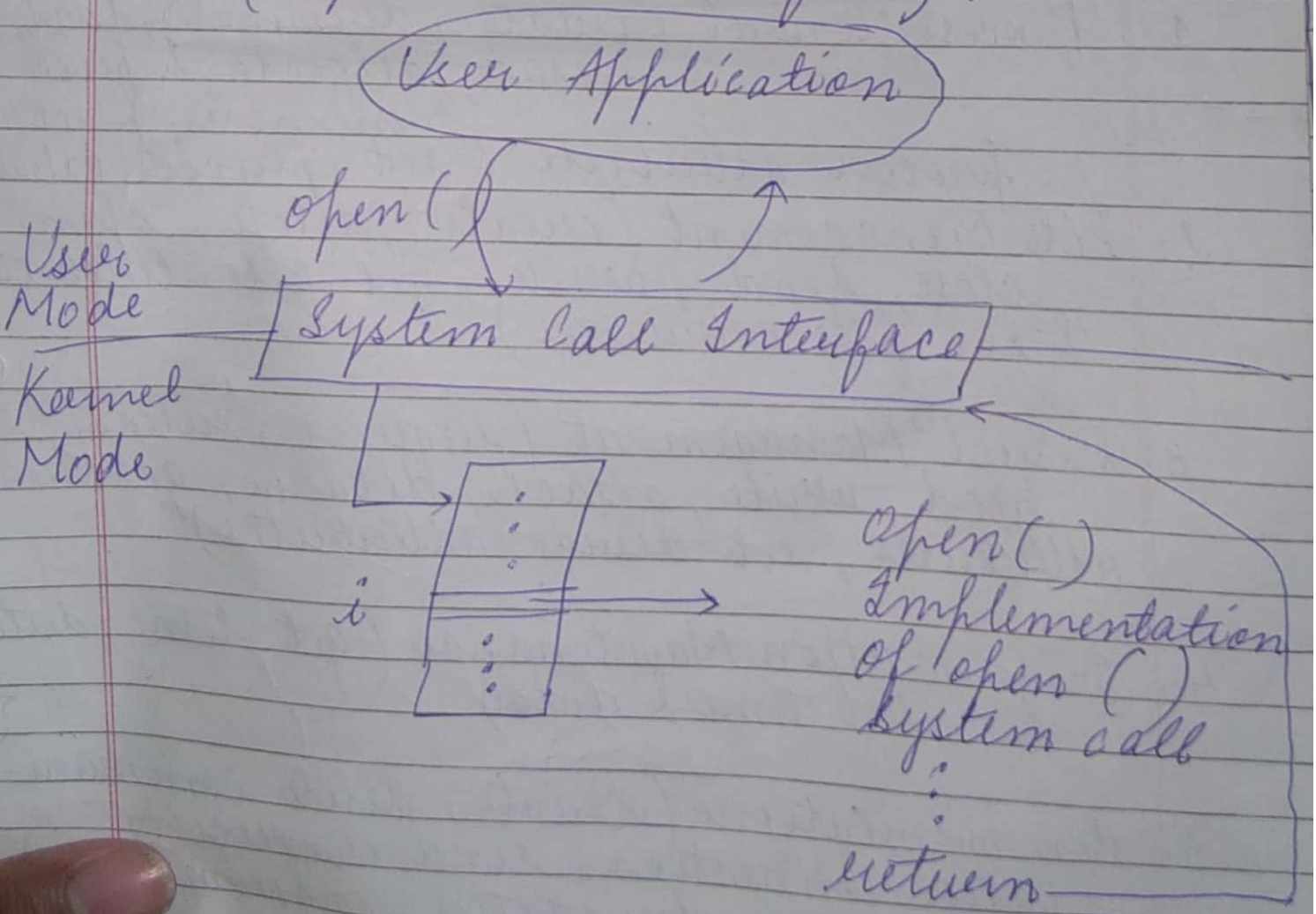


Example System Call Sequence

Acquire input file name
Write prompt to screen
Accept input
Acquire output file name
Write prompt to screen
Accept input
Open the input file
if file doesn't exist, abort
Create output file
if file exists, abort
Loop
Read from input file
Write to output file

Until Read fails
Close output file
Write completion message to screen
Terminate normally

Most programmers use API which specify set of functions that are passed to each function & return values the programmer expect. The API then makes the appropriate system calls through SCI (System Call Interface).



A number is associated with each system call, & the SCI maintains a table indexed according to these numbers. The SCI then invokes the intended system call & returns the status of the system call.

The caller need to know nothing about how the system call is implemented or what it does during execution.

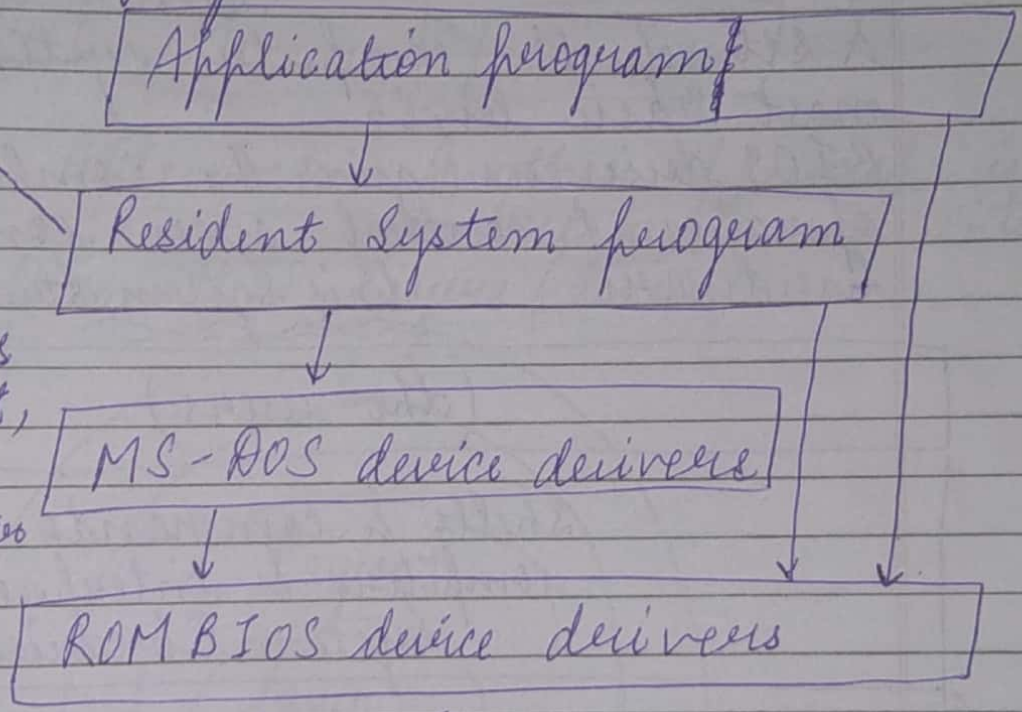
Types of System Calls:

1. Process control (create, terminate, load, execute, allocate & free memory, get process attributes & set process attributes)
2. File Management (create, delete, open, close, read, write, get file attributes & set file attributes)
3. Device Management (request, release, read, write, attach, detach, get device attributes, set device attributes)
4. Information Maintenance (get time, date & set time & date)
5. Communications (Create, delete communication connection, send or receive message, transfer status information)

Date: / / Page no:

OS Structure:

1. **Simple Structure**: MS-DOS. In this, the interface & levels of functionality are not well separated. It has no distinction between user & kernel mode, allowing all programs direct access to the underlying hardware.



MS-DOS Structure

(is a computer program that uses a system call to return control to OS as the program quit, but stays in ~~resident~~ computer memory so that it can be reactivated again)

Microsoft Disk OS

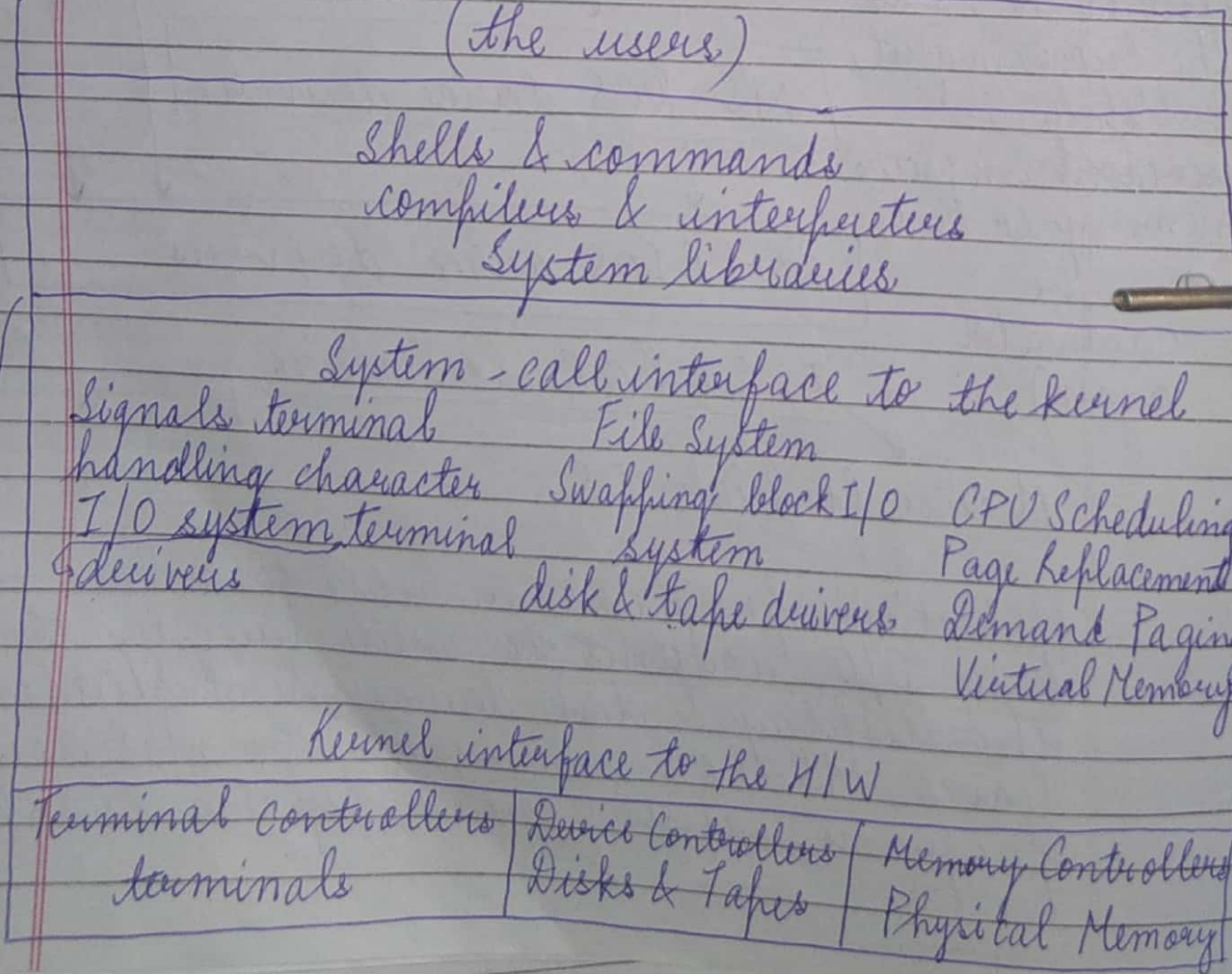
Applications programs are able to access the basic I/O routines to write directly to the display & disk drives. Such freedom leaves MS-DOS vulnerable to malicious programs, causing entire system crash

when users programs fail.

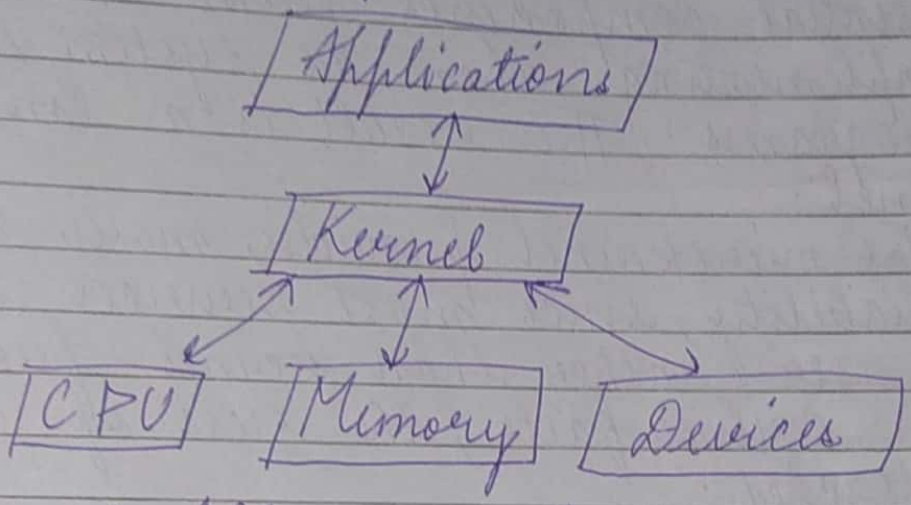
MS DOS Device Drivers : are sets of procedures that are used to communicate with H/W on the computer.

BIOS Device Drivers : Basic I/O system allows us to access & set up the computer system at the most basic level.
BIOS drivers give the computer basic operational control over computer's hardware.

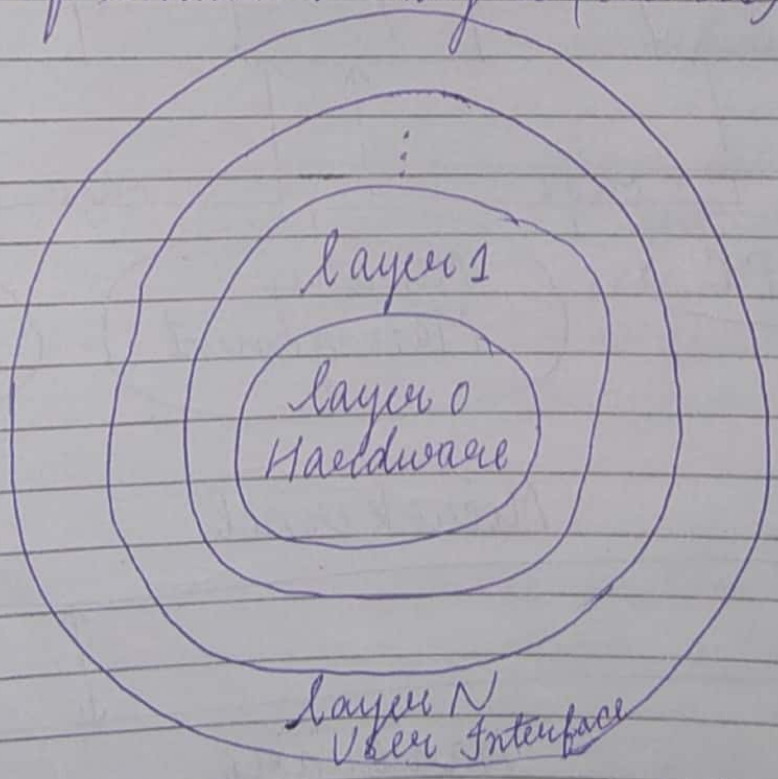
Unix System Structure



A Kernel connects the application softwares to the H/W of computer.



2. Layered Approach: Another approach is to break the OS into a no. of smaller layers (levels).

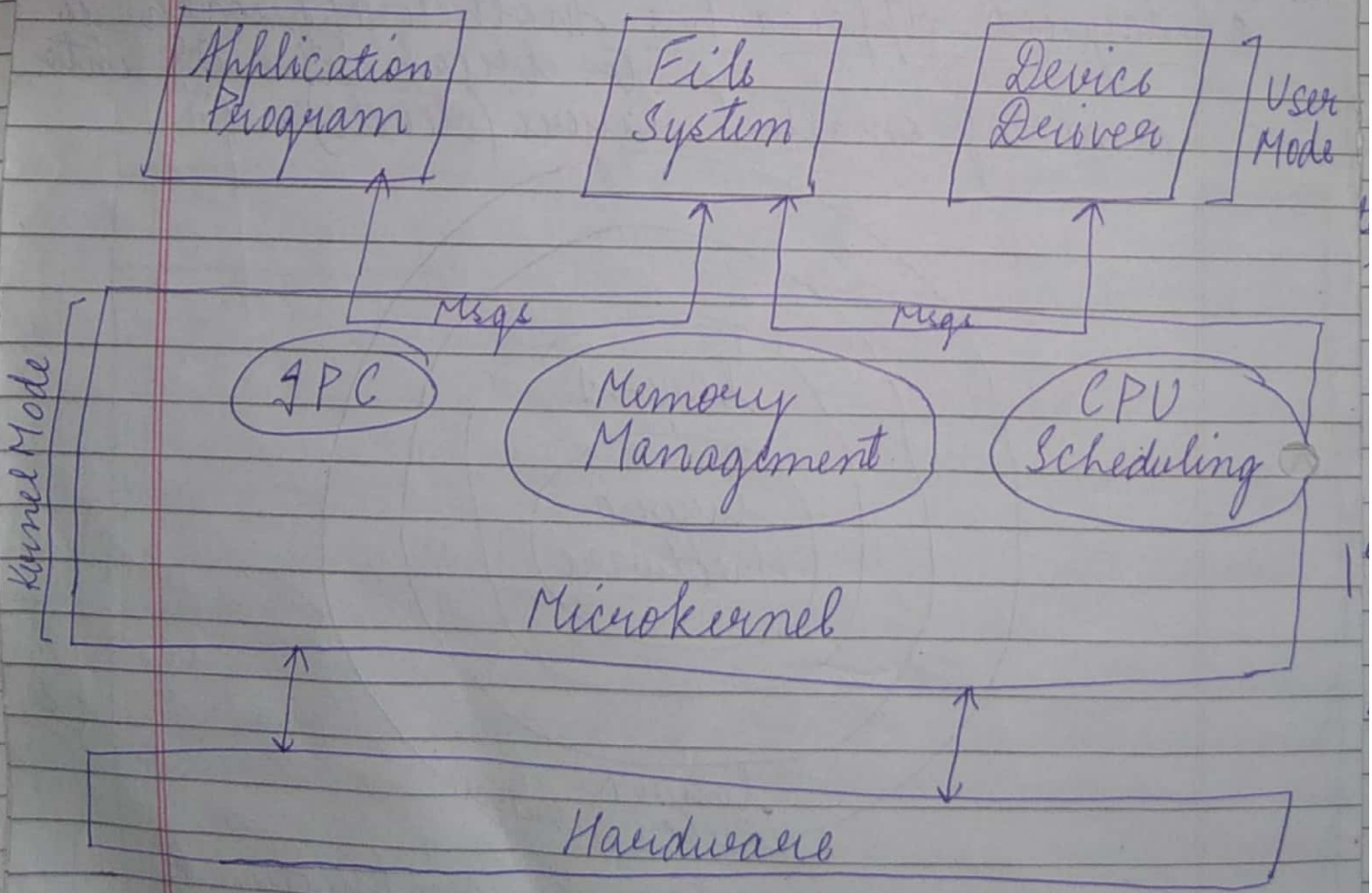


The main advantage is simplicity of construction & debugging.

method Date: / / Page no:

3. Microkernels : This structures the OS by removing all non-essential components from the kernel & implementing them as system & user-level programs. The result is a smaller kernel.

The microkernel provides more security & reliability, since most services are running as user rather than kernel-processes. If a service fails, the rest of OS remains untouched.



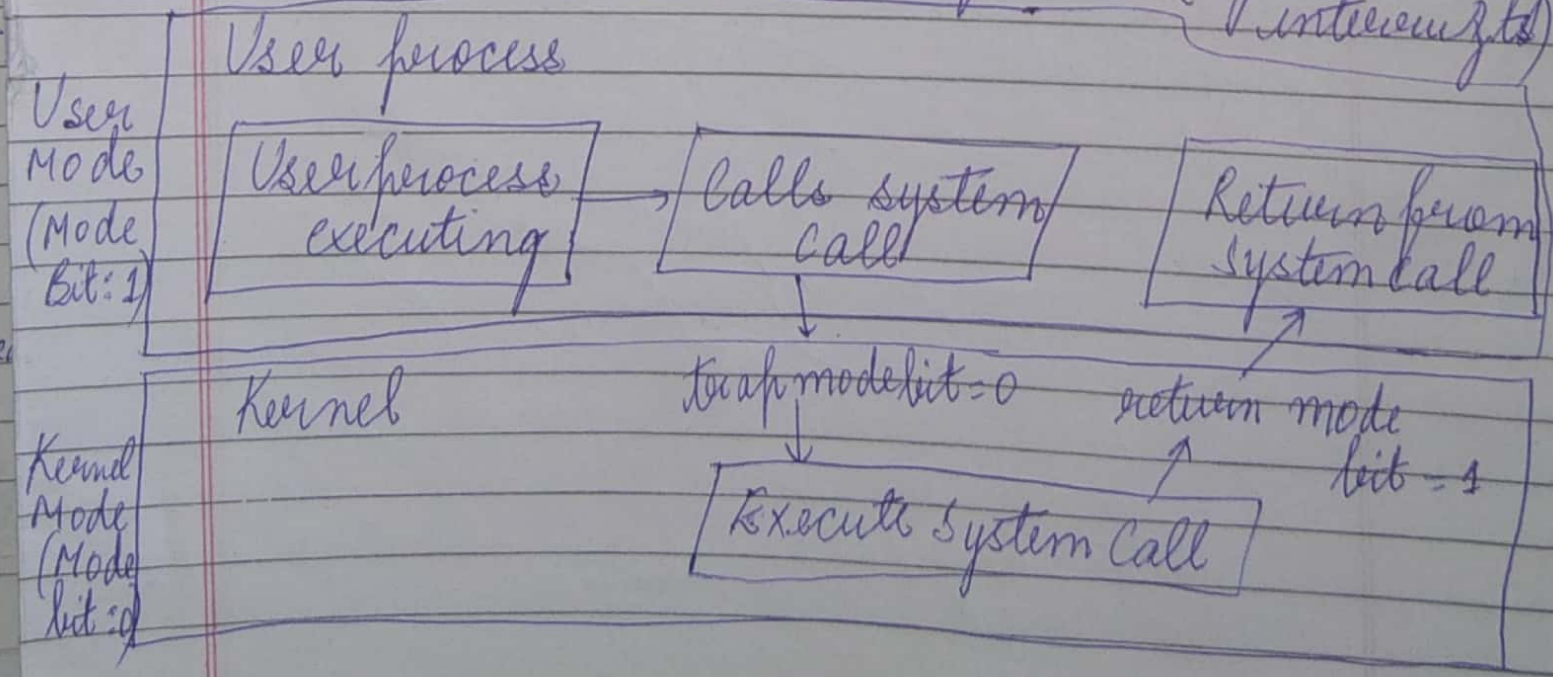
Dual Mode Operation:

In order to ensure the proper execution of the OS, we need two separate modes of operation: user mode & kernel mode (system mode, supervisor mode, privileged mode).

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

When the system is executing on behalf of a user application (on system OS) it is in user mode. (eg - creation of word doc.)

When a user application requests a service from the OS (via a system call), it transits from user mode to kernel mode to fulfill the request. (eg - handling interrupts)



Assignment-4

Q1. Explain the fundamental models of IPC?

Ans1. There are 2 fundamental models of IPC:

i) Shared Memory: IPC using shared memory requires communicating processes to establish a region of shared memory. 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.

ii) Message Passing: In this, communication takes place by means of messages exchanged between the cooperating processes. They do not share the same address space. This facility provide atleast two operations: send (message) & receive (message).

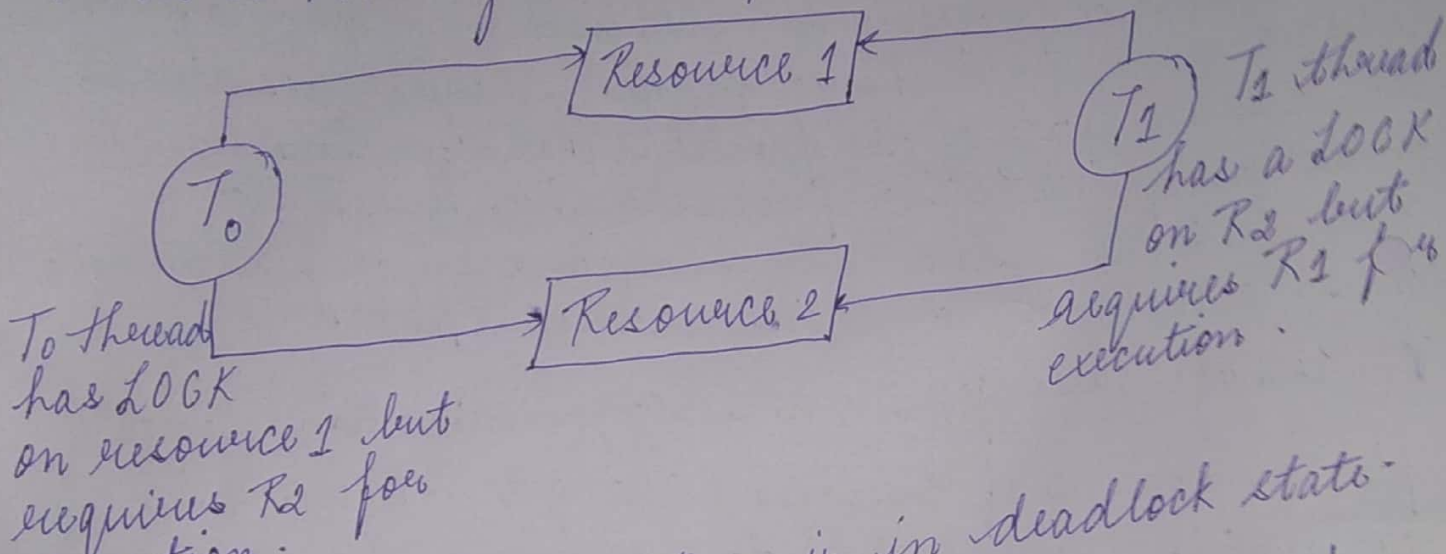
Q2. Define the following term:

1. Critical Section: It is a code segment that accesses shared variables & has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section.

2. Process Synchronization: It means sharing system resources by processes in such a way that concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.

It was introduced to handle problems that arose while multiple process executions.

3. Deadlock: Deadlock are a set of blocked processes each holding a resource & waiting to acquire a resource held by another process.



Currently, this system is in deadlock state.

Q3. Explain Semaphores & types of Semaphores?

Ans 3. A Semaphore 'S' is an integer variable that is accessed only through two standard atomic operations: wait() & signal(). They are also called P & V operations.

Definition of wait()

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

Definition of Signal()

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

Types of Semaphore:

i) Counting Semaphore: Value range over an unrestricted domain.

ii) Binary Semaphore: Value range only between 0 & 1. Also known as Mutex Locks.

Q4. Consider 5 processes & 3 resource types. Resource type A has 10 instances, B has 5 instances & C has 7 instances.

i) Need Matrix = Max. − Allocation

A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

ii) $Request_1$ = 1 0 2

$Request_1 < Need_1$ 1 0 2 < 1 2 2

$Request_1 < Available$ 1 0 2 < 3 3 2

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

1. Work = 2 3 0
   Finish = F F F F F
            0 1 2 3 4

2. For i = 0 $Need_0$ = 7 4 3 & Finish [0] is false.
   $Need_0$ > Work. So, P_0 must wait.

2. For i = 1 $Need_1$ = 0 2 0 & Finish [1] is false.
   $Need_1$ < Work. So, P_1 is kept in safe sequence.

3. Work = Work + Allocation = 2 3 0 + 3 0 2 = 5 3 2
   Finish = F T F F F

2. For i = 2 $Need_2$ = 6 0 0 & Finish [2] is false.
   $Need_2$ > Work. So, P_2 must wait.

2. For i = 3 $Need_3$ = 2 1 1 & Finish [3] is false.
   $Need_3$ < Work. So, P_3 is kept in safe sequence.

3. Work = Work + Allocation = 5 3 2 + 2 1 1 = 7 4 3
   Finish = F T F T F

2. For i = 4 $Need_4$ = 4 3 1 & Finish [4] is false &
   $Need_4$ < Work. So, P_4 is kept in safe sequence.

3. Work = 7 4 3 + 0 0 2 = 7 4 5
   Finish = F T F T T

2. For i = 0 $Need_0$ = 7 4 3 & Finish [0] is false &
   $Need_0$ < Work. So, P_0 is kept in safe sequence.

3. Work = 7 4 5 + 0 1 0 = 7 5 5
   Finish = T T F T T

For i = 2 $Need_2$ < Work. So, P_2 is kept in safe sequence.

3. Work = 7 5 5 + 3 0 2 = 1 0 5 7
   Finish = T T T T T. Safe Seq is P_1, P_3, P_4, P_0, P_2

Assignment - 5

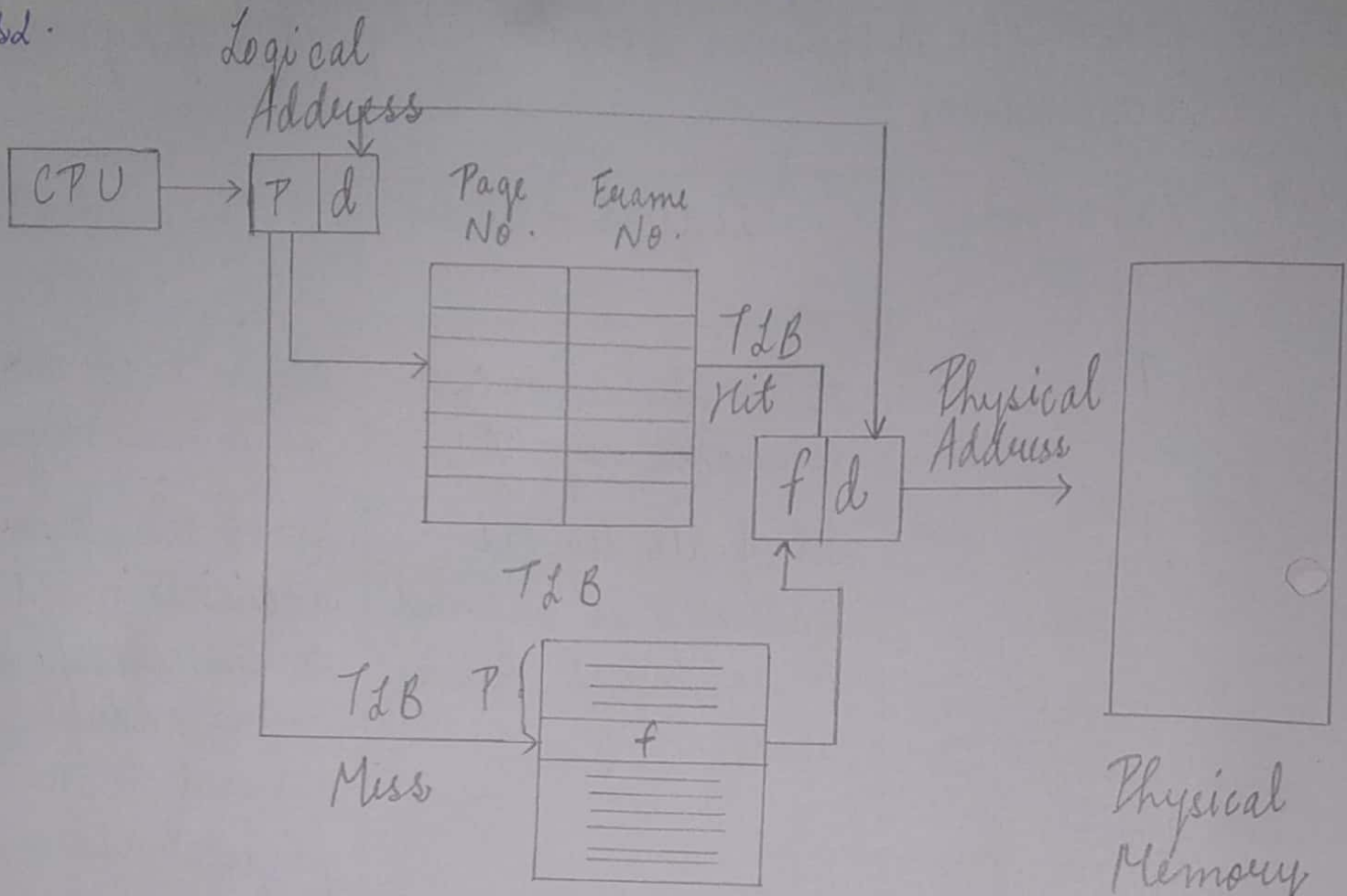
Q1. Explain the difference between logical & physical address space?

Ans 1.

Basis for Comparison	Logical Address	Physical Addr.
Basic	It is the address generated by CPU.	It is a location in a memory unit.
Address Space	Set of all logical addresses is referred to as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical address is referred to as physical address space.
Visibility	The User can view the logical address of a program.	The user can never view physical address of a program.
Access	The user uses the logical address to access the physical address.	The user cannot directly access the physical address.

Q2. Describe the concept of Paging with TLB?

Ans2.



A special, small, fast look up hardware cache, called a Translation Look Aside Buffer (TLB). It is associative, high speed memory. Each entry in TLB consists of 2 parts: a key (tag) & a value.

The TLB contains only a few of the page table entries. When a logical address is generated by CPU, its page number is presented to the TLB. If the page no. is found, its frame no. is immediately available & is used to access memory.

If the page no. is not in TLB known as TLB Miss.

Q3. Write short note on :

1. Fragmentation : It occurs in dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as the inability to use the available memory. It is of two types : External & Internal Fragmentation.

2. Segmentation : It is a memory management scheme that supports the user view of memory. A logical address is a collection of segments of variable size. Each segment has a name & a length. The address specify both the segment name & the offset within the segment.

3. Virtual Memory : It involves the separation of logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. It allow the sharing of files & memory to be shared by two or more processes through page sharing.

4. Demand Paging : It is similar to a paging system with swapping where processes reside in secondary memory. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, we use a lazy swapper which swaps a page into memory unless that page will be needed.

5. Swapping : A process must be in memory to be executed. A process however can be swapped temporarily out of memory to a backing store & then brought back into memory for continued execution.

Q4. Given page reference string : 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Compare the no. of page faults for LRU, FIFO & Optimal page replacement algorithm using 4 page frames.

Ans4. FIFO

1	1	1	1	5	5	5	5	3
	2	2	2	2	6	6	6	6
		3	3	3	3	2	2	2
			4	4	4	4	1	1

3	3	3	1	1
7	7	7	7	3
2	6	6	6	6
1	1	2	2	2

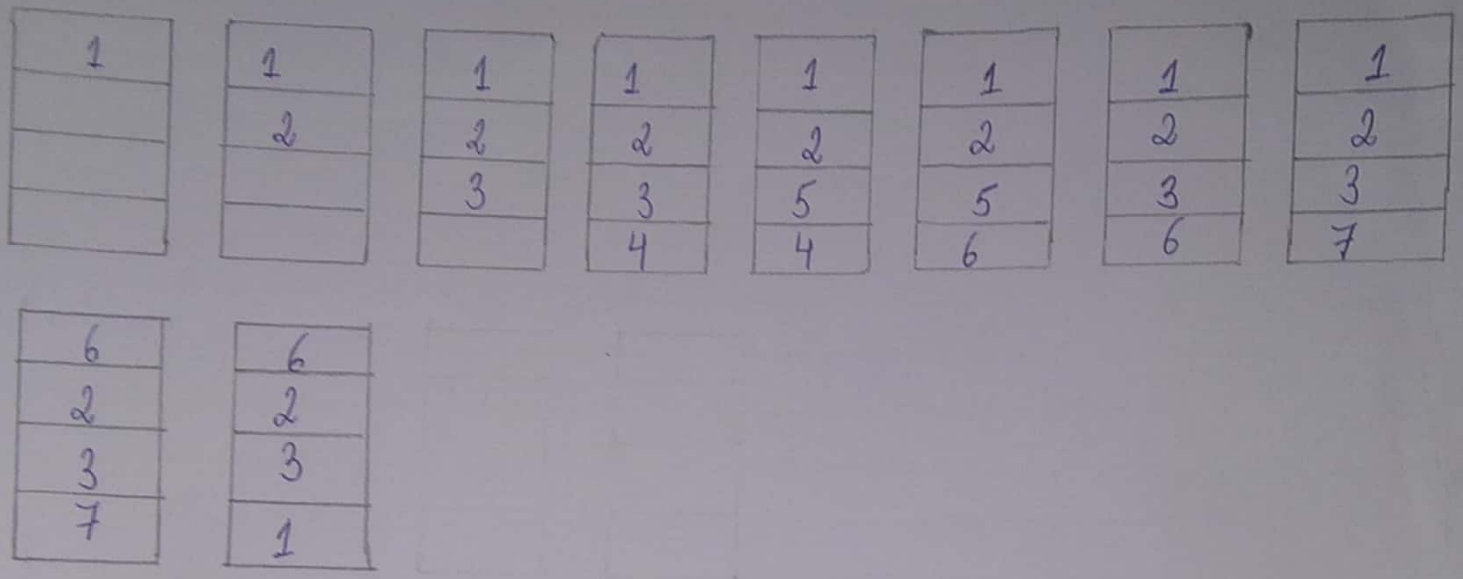
Page Fault = 14

Optimal

1	1	1	1	1	1	7	1
	2	2	2	2	2	2	2
		3	3	3	3	3	3
			4	5	6	6	6

Page Fault = 8

LRU



Page Fault = 10

Assignment - 1

Q1. Explain the following:

1. ANSI C Standard: It refers to the successive standards for the C programming language published by the ANSI & ISO. Historically, the names referred specifically to the original & best-supported version of the standard. Many programmers use ANSI C to refer to the standard while some software developers use the term ISO C & others are standards-body neutral & use Standard C.

2. ANSI/ISO C++ Standard: C++ is a general purpose programming language. It has imperative, object-oriented & generic programming features, while also providing facilities for low-level memory manipulation. It is standardized by the ISO with the latest standard version ratified & published by ISO in Dec. 2017 as ISO/IEC 14882:2017 (Informally known as C++17).

3. POSIX Standard: The Portable Operating System Interface (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between OS. It defines the API along with command line shells & utility interfaces, for software compatibility

with variants of UNIX & other OS.

Q2. Explain the difference between ANSI C & ANSI C++ Std?

Ans2 i) C is a procedural programming language whereas C++ is an Object oriented programming language.

ii) C does not support exception handling.

iii) C does not support templates.

iv) C does not support namespaces.

v) C does not support references.

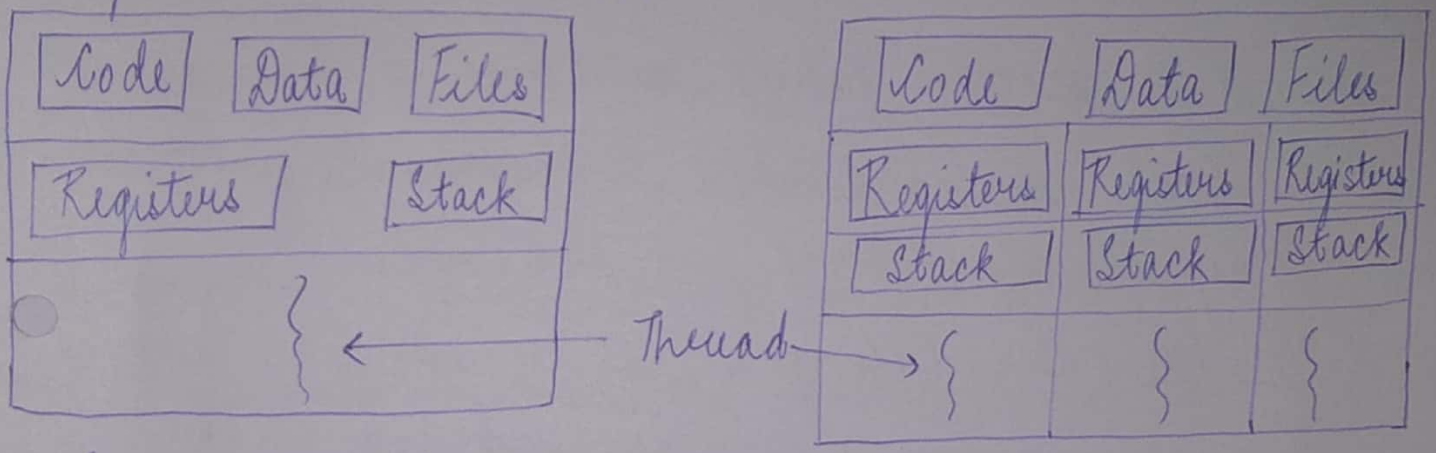
vi) C does not support default parameter values.

vii) C does not support overloaded function names.

Assignment-2

Q1. Write short note on:

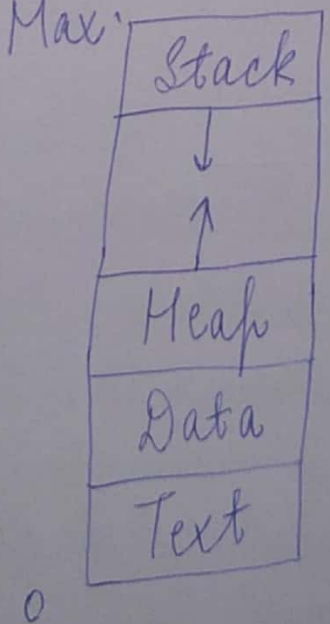
1. Threads: A thread is a basic unit of CPU utilization. It comprises a thread ID, a program counter, a register set & a stack.



Single Threaded Process

Multithreaded Process

2. Process: A process is a program in execution. Also known as the text section.



Process in Memory

A process include:

- the current activity as represented by the value of the program counter & content of the processor's registers.
- the process stack which contains temporary data.
- the data section which contains global variables.
- the heap which is the dynamically allocated memory during process run time.

3. Preemptive Scheduling : The scheduling which takes place when a process switches from running state to ready state or from waiting state to ready state. Processes can be interrupted in between. If a high priority process frequently arrives in the ready queue, low priority process may starve.

4. Non-preemptive Scheduling : The scheduling which takes place when a process terminates or switches from running to waiting state. Processes cannot be interrupted till it terminates. If a process with long burst time is running CPU, then another process with less CPU burst time may starve. It does not have overheads. It is rigid & not cost associative.

Assignment-3

Q2. Explain the different scheduling algorithm?

Ans2. i) FCFS : With this scheme, the process that requests the CPU first, is allocated the CPU first. The implementation is easily managed with a FIFO queue.

ii) SJF : In this scheme, the CPU is assigned to the process that has the smallest burst time.

iii) SRTF : Preemptive SJF scheduling is called SRTF.

iv) Priority : A priority is associated with each process & the CPU is allocated to the process with the highest priority. Equal priority process are scheduled in FCFS order. Smaller the CPU Burst time, higher the priority & vice-versa.

v) Round Robin Scheduling : A small unit of time, called the time quantum or slice is defined.

vi) Multilevel Queue Scheduling : It partitions the ready queue into several separate queues. The processes are permanently assigned to one queue & each queue has its own scheduling algorithm.

vii) Multilevel Feedback Queue Scheduling : It allow a process to move between queues. If a process uses too much CPU time, then it is moved to a lower priority queue.

Q3. Describe the differences among short-term, medium-term & long-term scheduling?

Ans3.

Long-Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1. It is a Job scheduler.	It is a CPU Scheduler	It is a process swapping scheduler
2. Speed is lesser	Speed is fastest.	Speed is in between both scheduler.
3. It controls the degree of multi-programming.	It provides lesser control over degree of multiprogramming.	It reduces the degree of multi-programming.
4. It selects processes from pool & loads them into memory for execution.	It selects those processes which are ready to execute.	It reintroduce the process into memory & execution can be continued.

Q4. Explain Process Control Block?

Ans4. Each process is represented in OS by PCB also called as Task Control Block.

Process State
Process No.
Program Counter
Registers
Memory Limits
List of Open Files
...

Assignment-3

Q1. Explain the memory layout of a C program?

Ans1. A C program is composed of the following pieces:

- Text Segment: consist of the machine instructions that the CPU executes.
- Initialized data segment: contains variables that are specifically initialized in the program.
- Uninitialized data segment: data in this segment is initialized by the kernel to arithmetic 0 or null pointer before the program starts executing.
- Stack: stores automatic variables
- Heap: Heap is where dynamic memory allocation usually takes place.

Q2. Write short note on:

1. setjmp function: We use this function to goto a label that is in another function. This function is useful for handling error conditions that occur in a deeply nested function call.
2. longjmp function: We use this function to goto a label that is in another function. This function is useful for handling error conditions that occur in a deeply nested function call.

3. getrlimit function : This function is defined in the XSI option in the single UNIX specification.

int getrlimit (int resource, struct rlimit *rlptr);

It gets limit on the consumption of a variety of resource.

4. setrlimit function : This function is defined in the XSI option in the single UNIX specification.

int setrlimit (int resource, const struct rlimit *rlptr);

It sets limit on the consumption of a variety of resource.

Q3. Describe the purpose of environment variable?

Ans3. An environmental variable is a dynamic named value that can affect the way running processes will behave on a computer. They are part of the environment in which a process runs.

Q4. Explain the functions related to the process termination?

Ans4. Normal termination occurs in 5 ways :

1. Return from main
2. Calling exit
3. Calling _exit or _Exit
4. Return of last thread from its start routine
5. Calling pthread_exit from the last thread

Abnormal termination occurs in 3 ways :

1. Calling abort
2. Receipt of a signal
3. Response of the last thread to a cancellation request.