OS

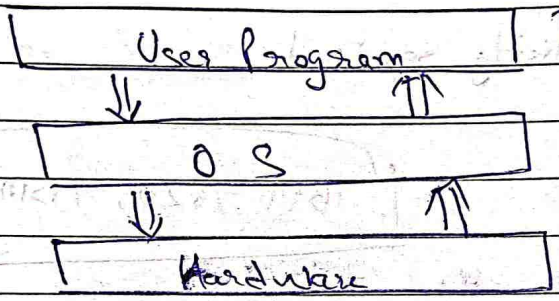
Date
Page No.

Unit - I

Operating System :- OS acts as an interface between User and Hardware.

Objectives of Operative System :- It is a collection of software that
- manages hardware resources
- Provides various services to the users.

Examples of operating system :- Microsoft, Linux, Android, etc.



Functions of Operating System :-

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system Performance
- Job accounting
- Error detecting aids
- Coordination blw other software and users.

Generations of Operating Systems :-

There are totally five generations till date.

Date
Page No.

① First Generation — IB-- 701, IBM-650

→ 1946- 1959
→ Vacuum tube based. as basic component for memory and CPU.
→ Batch processing operating system used
→ Punch cards, paper tape, magnetic tape for input and output
→ Unreliable, Costly
→ Lot of heat
→ Slow input, output
→ Huge size
→ Non portable
→ Huge amount of electricity required

② Second Generation — IBM 1620, IBM 7094

→ 1959- 1965
→ Transistor based.
→ Cheaper, consumed less power, more compactin size.
→ More reliable and faster than 1st generation
→ Magnetic cores used as primary memory and magnetic tape and disks as secondary storage devices
→ assembly and high- level programming language like FORTRAN, COBOL were used.
→ Batch processing and multiprogramming O.S

③ Third Generation — IBM 360 series

→ IC based
→ A single IC has many transistors, resistors and capacitors along with the associated circuitry.
→ Remote processing, time-sharing, multi ~~processing~~ programming O.S.

→ Smaller in size, reliable and efficient.

Fourth generation

→ 1971 - 1980
→ VLSI microprocessor based (Very large scale Integrated).
VLSI have about 5000 transistor and other circuit.
→ More powerful, compact, reliable, affordable.
→ Time sharing, real time networks, distributed operating system.

Types of Operating System

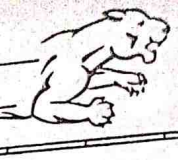
1) Serial processing / Bare Machines
2) Batch processing / Simple batch system
3) Multiprogramming Batch System
4) Time sharing System.
5) Multiprocessing
6) Desktop
7) Distributed.
8) Network
9) Real time
10) Clustered.

1) Serial processing :-

→ Performs all the instruction in a sequence or serial manner.
→ Executes in a FIFO manner.
→ In this system all the jobs are firstly prepared and stored on the punch card and after that card be entered in the system and then all instructions will be executed one by one sequentially.

Scheduling, Resources, Setup time

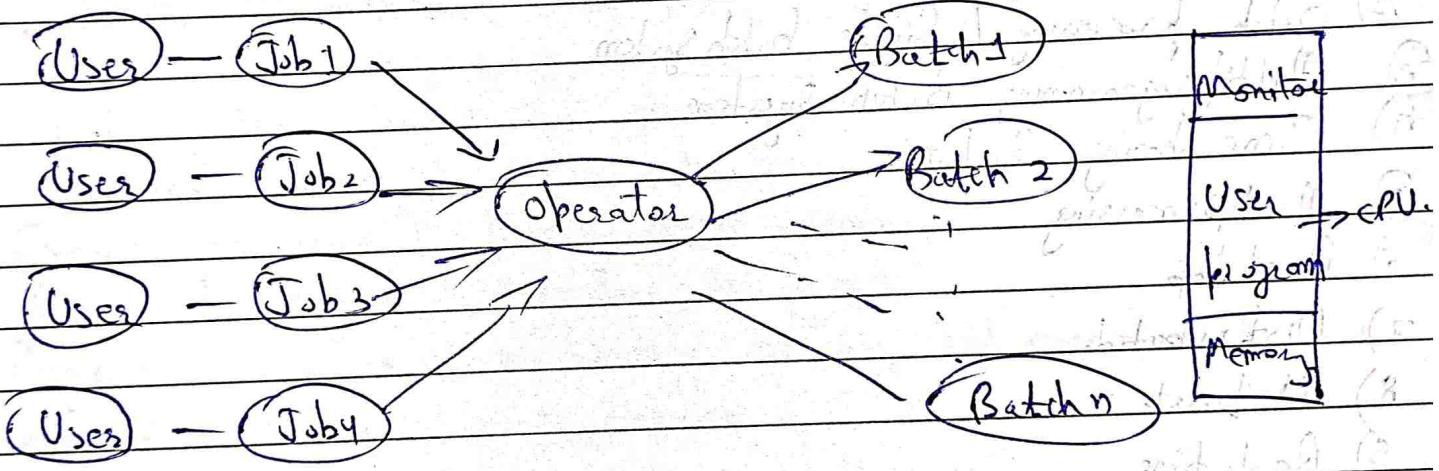
Date
Page No.



→ Also called Base machines as they do not have OS.
→ programmer directly interact with hardware
→ User can't be able to enter the data for execution or user cannot interact with system.

Batch OS:— eg Payroll system, Bank statement

→ Improve the utilisation of computer resources
→ Early computer were very expensive and therefore it was important to minimize processor (CPU) utilization
→ Batch is defined as a group of jobs with similar needs
- In Batch system - the user has no direct access to system
- Cannot interact with this task to fix problem



- Uses submits the job on cards or tape to a computer system, who batches the jobs together sequentially and places the entire batch on an input device. for use by ~~monitor~~ program that control the execution of jobs.
- Monitor first selects first job from the batch 1 and executes it, after completion of 1st job it selects another job from batch to execute

Date
Page No.

adv :- ① It takes work of the operator to computer
② Increased performance

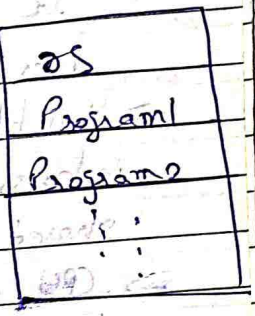
disad :- ① Difficult to debug program
② A job could enter in infinite loop.
③ Lack of protection scheme
④ Lack of protection
⑤ CPU time idle.

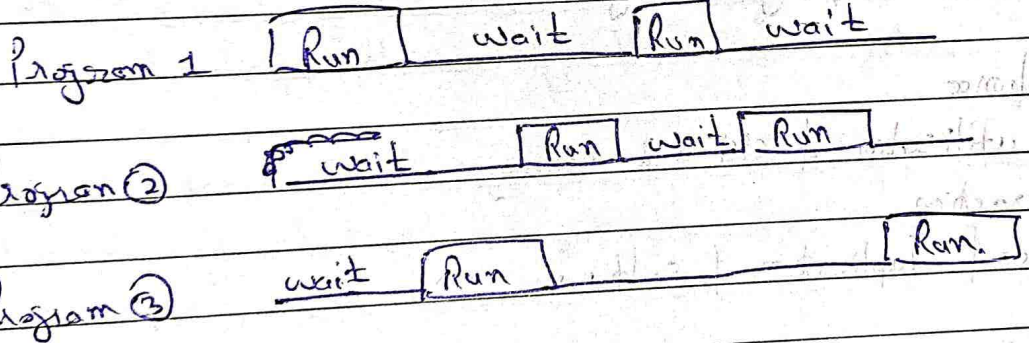
Multiprogramming Batch System :-

→ Sharing the processor (CPU), when two or more programs reside in memory at the same time is known as Multiprogramming.

→ It increases CPU utilization by organising programs into memory so that the CPU always has one to execute.

→ In this OS picks up and begins to execute one of jobs from memory. Once job needs I/O operation OS switches to another job using CPU scheduling algo.

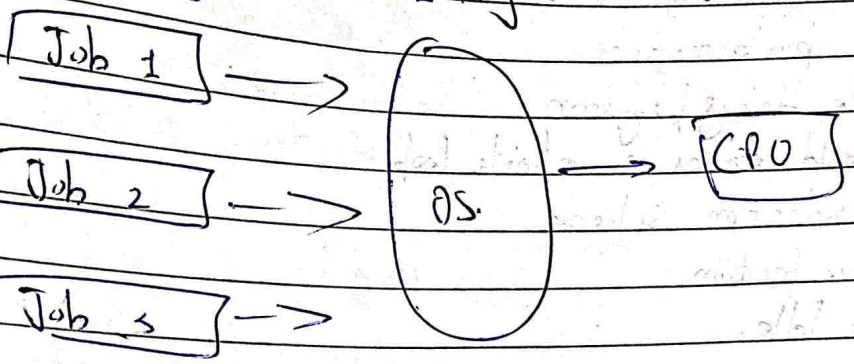




adv :- High and efficient CPU utilization
:- " " " Memory "

disad :→ CPU scheduling is required.
→ Memory management is required

(3) Timesharing or Multitasking :- eg :- multics, unix. etc.



→ It is a logical extension of Multiprogramming that provides user interaction
→ Switching of CPU b/w 2 users is so fast that is gives the impression to user that he is only using the system but actually shared among different users
→ CPU time is shared among multiple users hence it is called as time sharing system.
→ It uses CPU scheduling and multiprogramming to provide each user with a small portion of CPU.
→ Multiple users simultaneously access the system through terminals.
→ Complex than Multiprogramming.

adv :- Quick response
→ Efficient utilisation of CPU and memory
→ User interaction
→ Less chance of duplication of software

disad :- Reliability problem
→ Problem of data communication
→ Security and integrity of user program and data
→ Complex.

Date
Page No.

Multiprocessor System :-

⇒ Also known as parallel or tightly coupled systems.
→ A multiprocessor system contains of several processors that share a common physical memory, computer bus, system clocks, I/O devices.
⇒ Here multiple CPU's are connected in close communication through interconnection network.
→ It control and manages the hardware and software resources such that user can view the entire system as a powerful uniprocessor as he is not aware of multiple processor and interconnection networks.
⇒ In this all processors operate under single OS.

Ad → Execution of several tasks by different processors concurrently, increases the system's throughput
→ Because of resource sharing, multiprocessor system are cheaper than multiple single processor systems
→ Speeding up the execution of single tasks by subdividing it into subtasks. and then execute these subtasks in parallel in different processor
⇒ Increases in the system reliability as the failure of one processor will not stop the working the whole system.

Disad :- Process synchronization, task scheduling, Memory management, Security and protection, complex due to shared memory among multiple processors.

Desk top System :- A computer system dedicated to a single use.
→ Also called as Personal Computers (PCs)
→ Instead of maximising CPU and peripheral utilization, the systems are designed for maximizing user convenience and responsiveness.
⇒ These are microcomputers that are smaller and less expensive than mainframe computer

Date
Page No.

Real time operating System :-

objective :- is to provide quick response time to meet a scheduling deadline
- User convenience and resourse utilization

→ A real time OS is one that must react to inputs and responds to them quickly.
→ Real time systems are used when there are time requirements are very strict like missile systems, air traffic control system, robots etc.
→ In RTOS, OS typically read from and react to sensor.
→ The OS must guarantee response to events with fixed periods of time to ensure correct performance.
→ Real time systems can be used as a control device in a dedicated application.
→ There are 2 types of RTOS.
(1) Soft real time System
(2) Hard real time System.

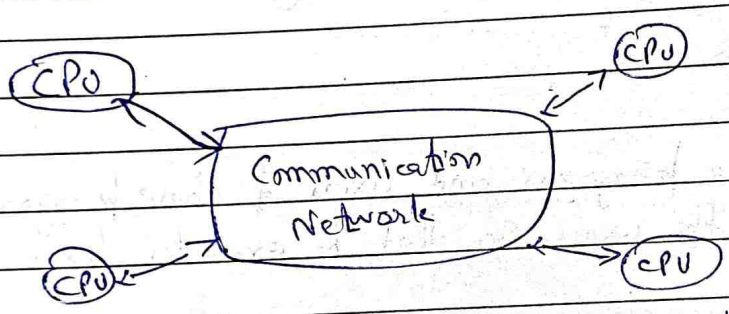
Soft real time OS	Hard real time OS
→ Less restrictive → Used where time constraint is less stricter → Performance degrades → Critical task gets priority, but no assurity of completing in defined time → Limited Utility → Uses in VR, Multimedia, advanced scientific projects	→ Guarantee task completion on time → If deadline is missed, system fail to work or does not work properly → Used in application where time constraint are very stricted and even the shortest possible delay is not acceptable → Used for saving life like automatic parachutes or air bags in case of accidents → No secondary storage, data stored in ROM

Date
Page No. 14

Adv :- Maximum utilization of device and system, more output
:- Focus on running operation as compared to operation in queue.
:- Used in embedded system
:- Error free
:- Best management of memory

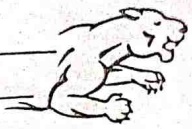
Disad :- Very few task run at same time
:- Use heavy system resources, they are expensive
:- Complex algo.

Distributed OS :- Also called as loosely coupled system.



→ Distributed OS are the OS for a network of autonomous computers connected by a communication network through a message passing mechanism.
→ Independent system possess their memory unit and CPU
→ When a program is executed on a distributed system, user is not aware of where the program is executed or the location of the resources accessed.
→ User can access the files or software which are actually not present on his system but on some other system connected within this network.
→ Eg :- Alpha, Kernel, Mach
→ The OS distributes computation logics among several physical processors.

Date
Page No.



Adv - ① Resource sharing facility
② Failure of one system will not affect the another system, as all system are independent from each other.
③ High computation speed.
④ Scalable - as systems can be easily added.
⑤ Load on host reduces.

Disad - ① Failure of main network will stop the entire communications.
② They are very expensive and complex
③ Absence of shared memory and global clock and unpredicatable communication delays make design & very difficult

⟹ OS Services

→ OS provides services to programs and users of those programs.
→ Os provides services to the users so that he executes programs in a convenient manner.
→ Os provide an environment to programs to complete its execution efficiently.
→ Services offered by all OS :-

① User Interface
② Program execution
③ File System Manipulation
④ I/o operations
⑤ Communication
⑥ Resource allocation
⑦ Error Detection
⑧ Accounting
⑨ Security and protection

Date
Page No.

User Interface :-

3 types

Common line Interface (CLI)
- uses text commands and methods for entering them.
- We interact with computer using typing commands

Batch Interface (BI)
- uses files → commands and direction for controlling cmds
- → files executed

Graphical User Interface (GUI)
- mostly used pointing devices (mouse)
- → Menu driven interface
- → Make selection
- → keyboard to enter text
- → Uses window icons, menus and other graphical objects to enter cmd

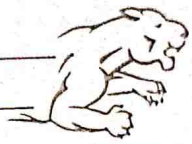
Program Execution :-

- → A program is loaded in the memory before it can be executed.
- → OS provides the facility to load programs in memory easily and then execute it
- → Program must be able to end its execution either normally or abnormally (errors).

File System Manipulation :-

- → Program can also take input from file (read a file) or store output to file (write on file).
- → OS gives the permission to the program for performing operation on file.
- → In case of multiple users System OS may provide protection mechanism to control access to file (read or R/W).

Date
Page No.



I/O operations :-

→ A running program may require Input/Output from a file or I/O device.
→ For protection and efficiency, users cannot directly interact with I/O devices.
→ OS provides a user interface that hides the details so the programmer can access devices using simple read and write operation.

Communication :-

→ Processes may communicate with each other for data transfer.
→ Processes involved in communication may reside on same computers or on different computers (via network)
→ OS supports for communication b/w processes using shared memory or by message passing

Resource allocation :-

→ When multiple processes running at the same time in a computer, then resources must be allocated to each of them which is done by OS.
→ Resources can be CPU, main memory, file and I/O devices.
→ For allocating CPU, CPU scheduling algorithm are used for efficient utilization of CPU.

Errors detection :- Errors may occur in CPU, memory, I/O devices and in user program.
→ For correct and consistent computing, OS takes adequate action for each type of errors.

Date
Page No.

Accounting :- OS keeps records of which users uses how much and what kinds of computer resources

→ This accounting data may be used for statistics, for the billing or to improve System's efficiency.

Protection and Security:- In protection, all the access to the resources of computer is controlled.

→ For security of computer, users needs to authenticate him to the system using login ID and password before using the computer.

→ Protection is also required to protect one process to interface the other process or OS.

⇒ **Process** :- A process can be defined as:-

→ A program in execution

→ An instance of a running program

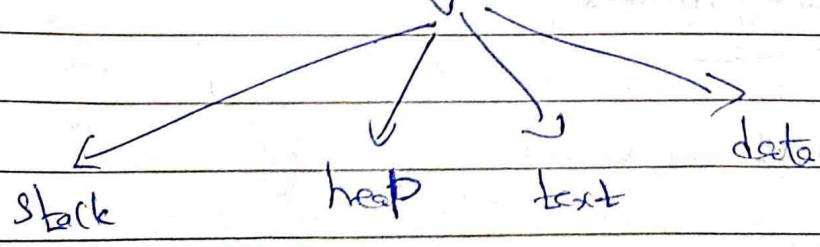
→ The entity that can be assigned to, and execute on a processor.

Comp progr. in text file.

↓ executes

Process stored in Main memory

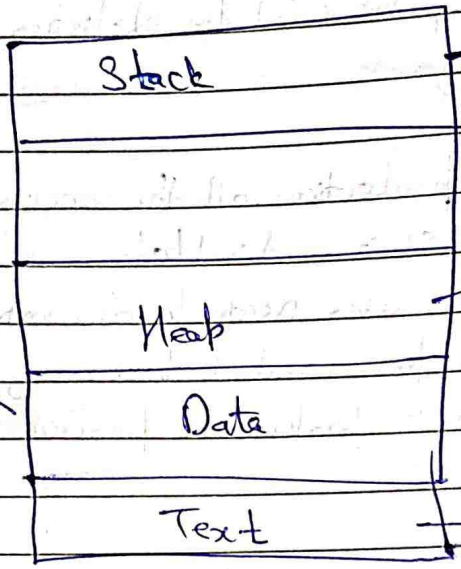
↓ which perform all tasks mentioned in prog.



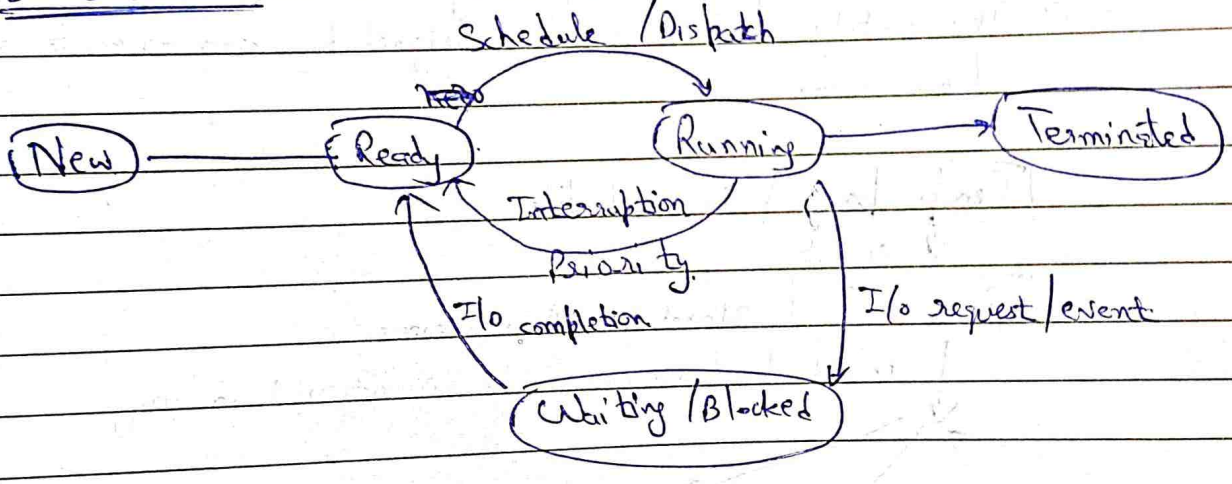
Process is an active entity
Program is an passive entity

Date
Page No.

Layout of process in Main memory :-



Process states :-



New :- Process has been created

Ready :- Process is ready to be executed

Running :- The process is running

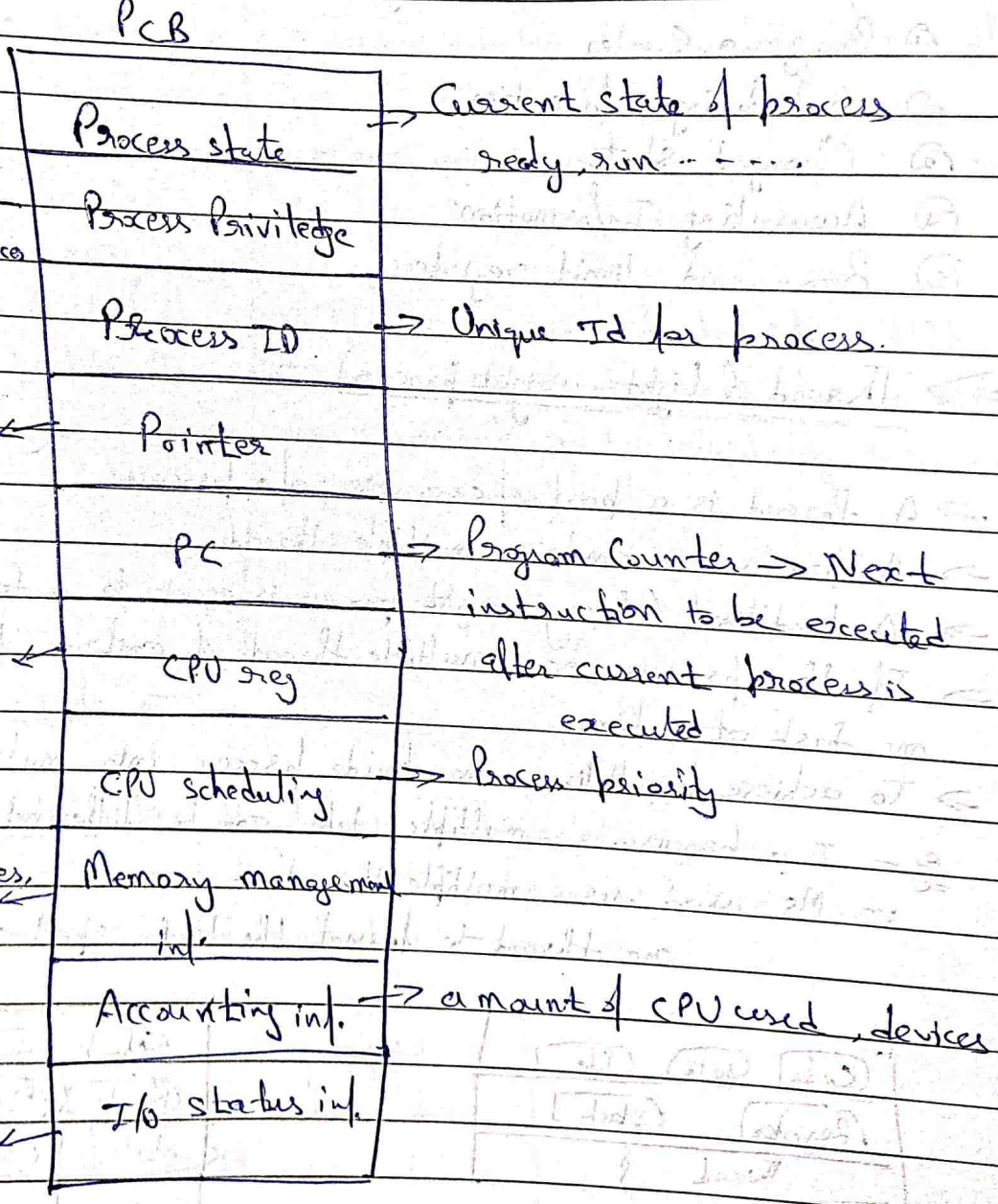
Terminated :- The process has been terminated or completed

Date
Page No.

Waiting :- The process is waiting to occur due to I/o operation or waiting for an event to occur.

⇒ Process Control Block (PCB)

→ It is a data structure maintained by OS for every process.
→ Identified by an integer. i.e. Process ID (PID).
→ PCB keeps all the information needed to keep track of a process.



Date
Page No.

=> Context Switching :- Switching of CPU to another process means saving the state of old process and loading saved state for new process.

-> In context switching, the process is stored in PCB to resave for the new process, so that old process can be resumed from the same part it was left

=> To make context switching time to be less, registers which are the fastest access memory are used.

=> Information to be stored for context switching in PCB is :-

1. Program Counter
2. Scheduling Information
3. Changed state
4. Accounting Information
5. Base and limit registers.

=> Thread (Light weight process) :-

-> A thread is a part of execution of a process.

-> A process can contain multiple threads.

-> A traditional / heavy weight process is equal to a task with one thread.

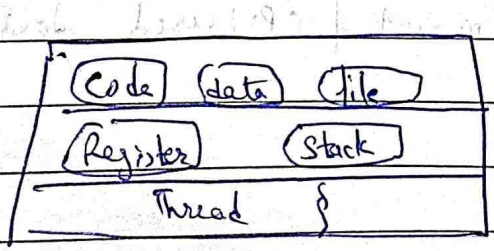
-> If the process has multiple threads of control it can do more than one task at a time.

-> To achieve parallelism we divide process into multiple thread.

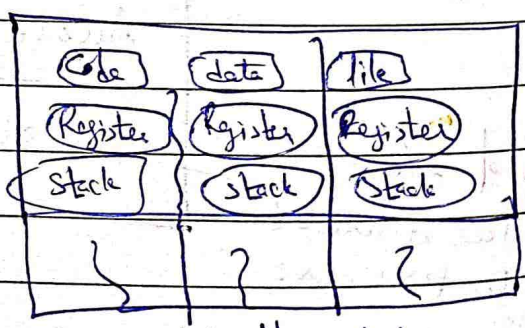
eg :- In a browser, multiple tabs can be different threads

:- Ms word uses multiple threads :-

one thread to format the text, spell check, type text etc.



Single threaded



Multi threaded.

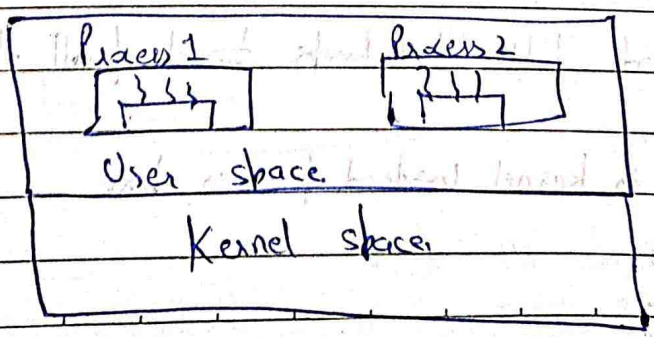
Date
Page No.

Process	Thread
→ Heavy weighted Process	→ Light weighted process.
→ Run in seperated memory space	→ Run in a shared memory space
→ Independent of each other	→ Depended on each other
→ Share physical memory, disks, printers etc.	→ Share with other threads their code section, data section and OS resource like (open file and signals)
→ Requires more resources to execute	→ Requires less resources to execute
→ Implementing communication blw two processes is a bit more difficult	→ Communication blw two threads are easy to implement bcs threads shares common address space.
→ Context switching blw two process is time consuming	→ Less time consuming
→ A process without multiple threads has slow response time	→ If the process is divided into multiple threads, if one thread compute its execution, then its output can be immediately returned (faster response)
→ Slow execution	→ fast execution.

⇒ Types of Threads :-

① User Level Threads :- User threads are implemented by users.
→ Kernel knows nothing about users level threads.

→ User level threads are loaded entirely in user space.



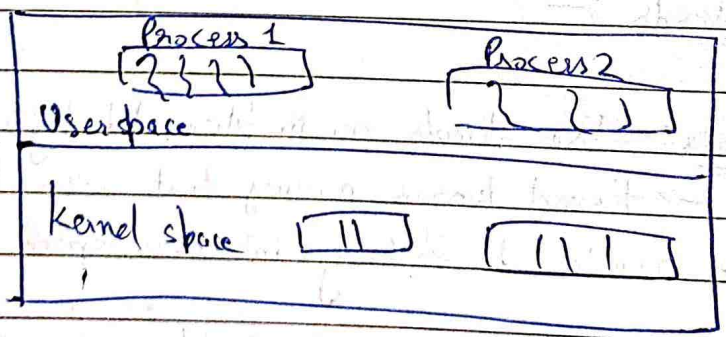
Date
Page No.

-> Each process needs its own private threads table which consists of program counter, stack pointer, registers, ~~state~~ state etc.
-> Thread table managed by runtime system.
-> Scheduling are done in user space without the need of kernel intervention.

ad :- fast to create and manage
-> Context switching time is less, bcz no hardware support is required.
-> User level thread runs on any OS.
~~that~~ -> Each process can have its own scheduling algorithm
disad :- If one user level thread perform block system call then entire process will be blocked that means all of the threads within the process are blocked
:- Multiprocessing can't be implemented only a single thread within process can execute at a time.

(2) Kernel level threads :-

-> Implemented by OS
-> Thread management is done by the kernel
-> No thread table in each OS.



-> Kernel has a thread table that keeps track of all the threads in system.
-> Information stored in kernel instead of user space.

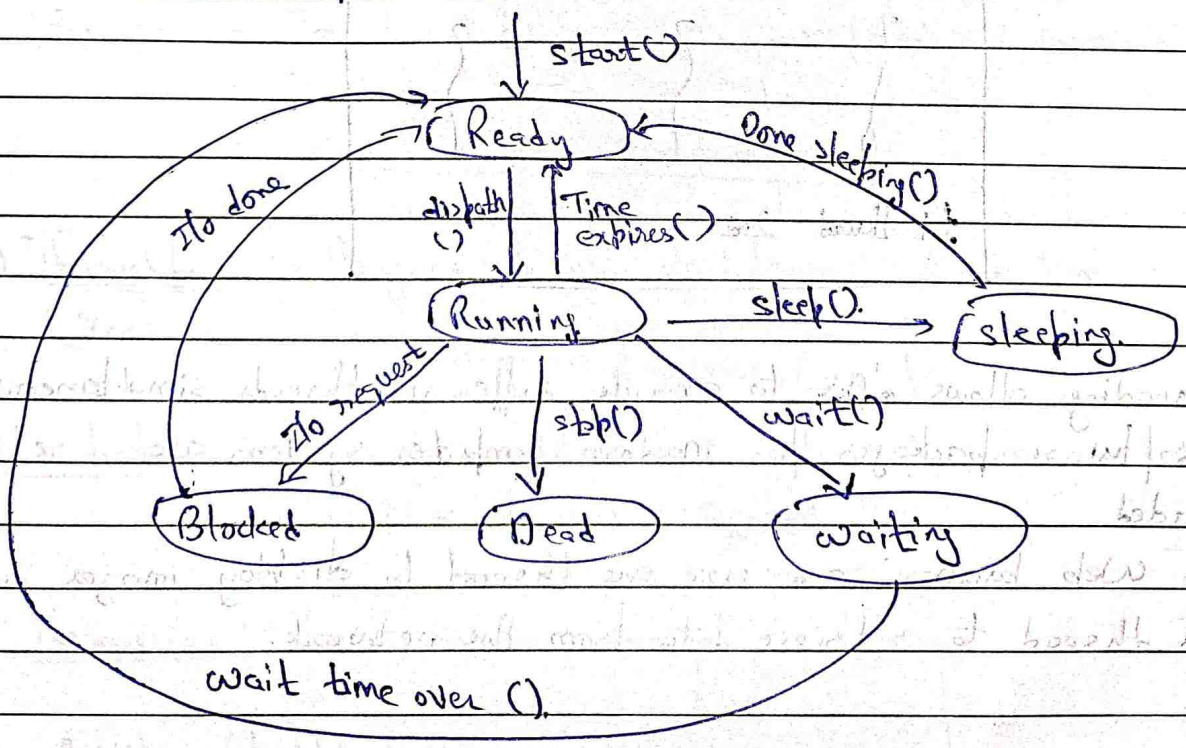
Date
Page No.

→ When a thread wants to create a new thread or destroy any existing thread, it makes a kernel call, which takes the action.
→ The Kernel performs thread creation, scheduling and management in kernel space.

Adv:- ① Kernel can simultaneously schedule a multiple threads from the same process on multiple processors. It supports multiprocessing where the user level threads cannot support this
→ If one thread in a process is blocked, the kernel can schedule another thread of the same process.

disadv:- Implementation is complicated
:- Context switch time is more, it requires hardware support
:- Kernel threads are generally slow to create and manage them the user thread.

⇒ States of threads :-



Date
Page No.

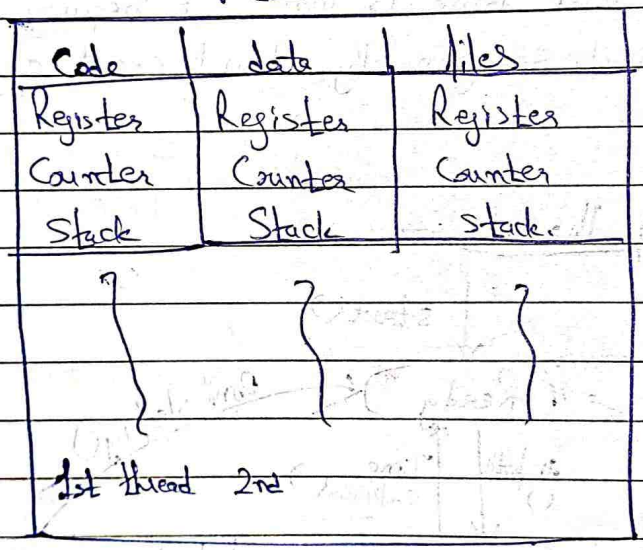
IPC → Inter process communication

⇒ Benifits of Threads :-

① Threads share common data and do not need to use IPC.
② There is a higher throughput when multiple threads cooperate in a single job.
③ Threads are cheap to create as they only need a stack and storage for registers.
④ Uses very little resources of OS
⑤ Context switching becomes fast

⇒ Multithreading :-

Process 1.



→ Multithreading allows OS to execute different threads simultaneously
→ Many software packages for modern computer systems are multithreaded
→ eg:- a web browser can use one thread to display images and a second thread to retrieve data from the network.

Date
Page No.

Multi-threading models :-

① Many to One :- Many user-level threads maps to single kernel thread

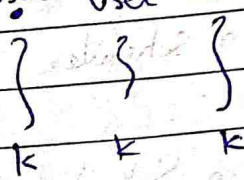
-> Only one thread can access the kernel at a time.

eg :- Solaris is green thread, GNU portable thread.

② One-to One :- User to kernel

eg :- Windows NT / xp / 2000

:- Linux, Solaris 9 and later



③ Many-to-Many :- Allows many user level threads to be mapped to many kernel level threads.

⟹ Objectives of Scheduling :-

① Fairness :- Treat all processes as same and no process should suffer.

Indefinite postponement

② Throughput :- largest possible no of process / Unit time (ms)
(Max)

Max no of process / ms

③ Predictable :- Process should run in about the same amount of time irrespective of load on System.

④ Resources :- keep resources busy

⑤ Avoid indefinite postponement

⑥ Priorities :- Should favor high priority process

Date
Page No.

⑦ Overhead :- Position of system resources is invested as overhead can greatly improve performace of system.

Scheduler :- Manages the processes.

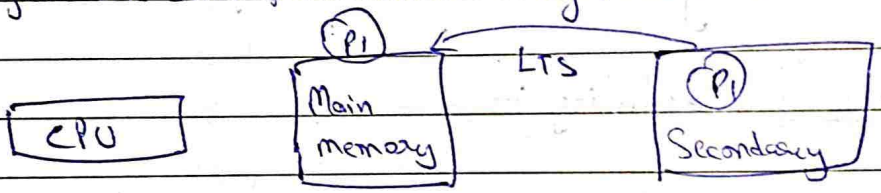
Types of Schedulers

LTS
Long Term Scheduler

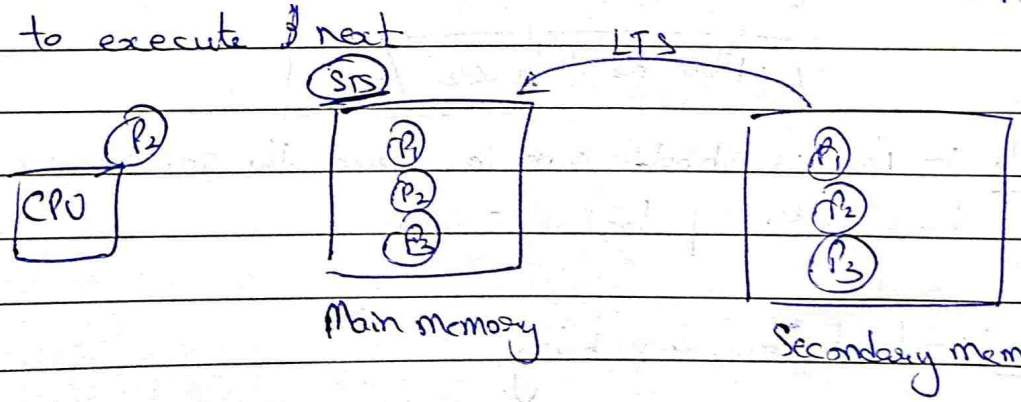
MTS
Middle term scheduler

STS
Short Term Schedulers

① Long Term Scheduler (LTS) :- It selects processes from the queue and loads them into the memory. when the process changes the state from new to ready, then there is use of LTS.

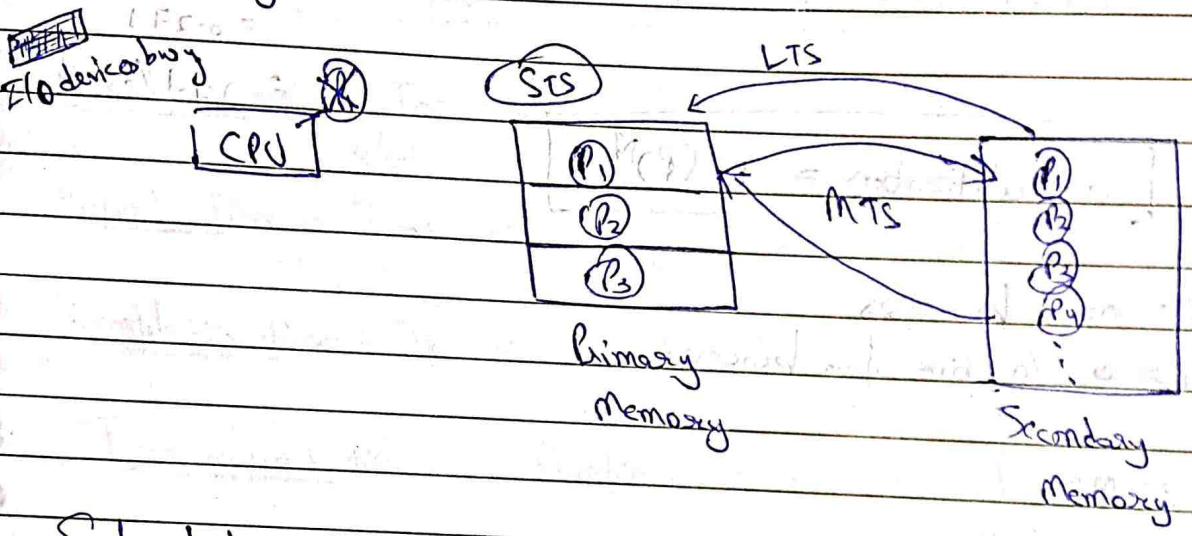


② Short Term Scheduler (STS) :- It is also called as CPU scheduler or dispatchers. STS selects a process among the processes that are ready to execute and allocates CPU to one of them. That means STS/CPU scheduler make the decision of which process to execute next



Date
Page No.

③ Medium Term Scheduler (MTS) :- It is a part of swapping. Some running processes requires I/O operation. In this condition, process suspends from main memory and placed on the Secondary memory, and then these process after a while reloaded in memory continued where they left earlier. Swap in and Swap out is done by MTS.



Scheduling :- The Scheduling is an activity of the scheduler that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

→ Process scheduling is an essential part of a multiprogramming OS. Such OS allows more than one processes to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

⇒ CPU utilisation :-

→ CPU utilization is percentage of time a CPU is occupied running processes

→ CPU utilization is always less than 100% because of I/O operation and context swithing

→ Ideally we want to reach 100% CPU utilization

Date
Page No.

Eg ① RAM = 4MB
Process size = 4MB.
hence No of processes = 1
I/O Time = 90%
CPU utilization = 100% - 90% = 10%
in (P) = 1 - 0.9 = 1

② RAM = 16 MB
Process size = 4MB
no of Process = 3
I/O Time = 90%.
~~Probability of~~ CPU utilization = $1-(0.9)^3$
$= 1 - 0.729$
$= 0.271$
$= 27.1\%$

$$\text{CPU utilization} = 1-(P)^N$$

where N = no. of processes.
P = I/O time of a process.

③ RAM = 36 MB
Process size = 4 MB
No of process = 9
I/O time = 90%.
Probability of CPU idle time = $0.9 \times 0.9 \ldots\ldots$ 10 times $= 0.3874$
CPU utilization = $1 - 0.3874$
$= 0.6126 = 61.26\%$

Note :- No of process in RAM ↑ CPU utilization ↑
(Degree of Multiprogramming)

:- If no. of processes in RAM $\to \infty$
then $\lim_{n\to\infty}$ CPU utilisation $= \lim_{n\to\infty} 1-(P)^{\infty}$
$= 1$ (i.e. 100%)
It means there is always a process available to occupy CPU.

Date
Page No.

⇒ Throughput :- Throughput is the amount of work completed in a unit of time. In other words throughput is the processes executed to number of jobs completed in a unit of time. The scheduling algorithm must look to maximize the number of jobs processed per unit time.

| No. of process. / Unit of time (in ms) |

Arrival time :- The time at which process enter the Ready Queue or state.

Burst time :- Time required by a process to get execute on CPU.

Completion time :- The time at which process completes or terminates.

Turn around time :- Completion time – Arrival time.

Waiting time :- Turn around time – Burst time

Response time :- [(The time at which a process gets CPU first time) – (Arrival time)]

⇒ First Come First Serve (FCFS)

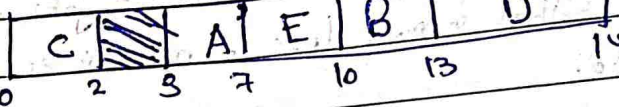
→ Simplest Scheduling algorithm, it assign CPU to the process which arrives fast.
→ Easy to understand and can easily be implemented using Queue data structure.
→ Always non-preemptive in nature

Date
Page No.

Q

Process	AT	BT
A	3	4 ✓
B	5	3 ✓
C	0	2 ✓
D	5	1 ✓
E	4	3 ✓

Gantt chart



TAT = CT − AT	WT = TA − BT
7 − 3 = 4	4 − 4 = 0
13 − 5 = 8	8 − 3 = 5
2 − 0 = 2	2 − 2 = 0
14 − 5 = 9	9 − 1 = 8
10 − 4 = 6	6 − 3 = 3

Av. Turnaround time = (4 + 8 + 2 + 9 + 6)/5 = 29/5

Av. Waiting time = (0 + 5 + 0 + 8 + 3)/5 = 16/5

Convoy effect – Smaller process have to wait for long time for bigger process to release CPU

adv – Simple, easy to use, easy to understand, easy to implement, must be used for background where execution is not urgent

disadv – Suffer from convoy effect, normally higher average waiting time, no consideration of priority or Burst time, should not be used for interactive system.

⇒ Shortest Job first (SJF) (Non-Preemptive) / Shortest Remaining Time first (SRTF) Preemptive

⇒ Out of all available process, CPU is assigned to the process having smallest burst time requirement (no priority, no seniority).

→ If there is a tie, FCFS is used to break the tie

→ Can be used both with non-preemptive and preemptive approach

Date
Page No.

→ Preemptive version (SRTF) is also called as optimal and guarantee minimal average.

Q

Process	A.T	B.T	TAT	WT
P_1	3	1	7-3=4	4-1=3
P_2	1	4	16-1=15	15-4=11
P_3	4	2	9-4=5	5-2=3
P_4	0	6	6-0=6	6-6=0
P_5	2	3	12-2=10	10-3=7

Non-preemptive

Gantt chart

P_4	P_1	P_3	P_5	P_2

0 6 7 9 12 16

Q Preemptive

Process	A.T	B.T	TAT	WT
P_1	3	1 ✓	1	0
P_2	1	~~1~~ 2 ✓	5	1
P_3	4	2 ✓	4	2
P_4	0	~~5~~ 6 ✓	16	10
P_5	2	3 ✓	9	6

Gantt chart

P_4	P_2	P_1	P_2	P_3	P_5	P_4

0 1 3 4 6 8 11 16

adv :- SRTF (preemptive) guarantees minimal average waiting time
:- Provide a standard for other algo in terms of average waiting time
:- Better average response time compared to FCFS

disadv :- Algo cannot be implemented as there is no way to know the burst time of a process.

Date
Page No.

→ Process with larger CPU burst time requirement will go into starvation
→ No idea of priority, process with large burst time have poor response time.

⟹ Round Robin (RR) Scheduling :-

→ This algo is designed for time sharing system where it is not necessary to complete one process and then start another, but to be responsive and divide time of the CPU among the process in ready state.
→ Here ready queue will be treated as circular queue
→ We fix a time quantum up to which a process can hold the CPU. in one go, with in which either a process terminates or process must release the CPU and reenter in the circular queue and wait for the next chance.
→ RR is always pre-emptive in nature.

Ex)

Process	AT	BT	TAT	WT
P_0	0	8 3 1 ✓	13	8
P_1	1	3 1 ✓	11	8
P_2	2	1 ✓	3	2
P_3	3	2 ✓	6	4
P_4	4	8 1	10	7

$T_q = 2$

P_0	P_1	P_2	P_0	P_3	P_4	P_1	P_0	P_4
0	2	4	5	7	9	11	12	13	14

$P_0\ P_1\ P_2\ P_0\ P_3\ P_4\ P_1\ P_0\ P_4$

Date
Page No.

Operating System

Unit - II

⇒ Critical Section :-

Critical Section is the part of a program which tries to access shared resources. That resources may be any resources in a computer like a memory location, Data structure, CPU or any IO device. The Critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

The Solution of Critical section problem requires following conditions :-

① Mutual Exclusion :- Our solution must provide mutual exclusion. It means that if one process is executing inside critical section then the other process must not enter in the critical section

② Progress :- Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

③ Bounded Waiting :- We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical Section.

Date
Page No.

Let P_1, P_2 and P_3 are using shared resources

```
do {
    Entry Section  => Controls the entry into Critical section and gets lock on required resources.
    Code seg.      <= Critical Section
    Exit Section   => removes lock on resources and others know that its Critical Section is over
    Remainder Section
} while (true);
```

=> **Race Condition** :- A race condition is a situation that may occur inside a critical section. This happens when the result of multiple thread execution in critical section differs according to the order in which the threads execute.

Race conditions in critical sections can be avoided if the critical section is treated as an atomic instruction. Also, proper thread synchronization using locks or atomic variables can prevent race conditions.

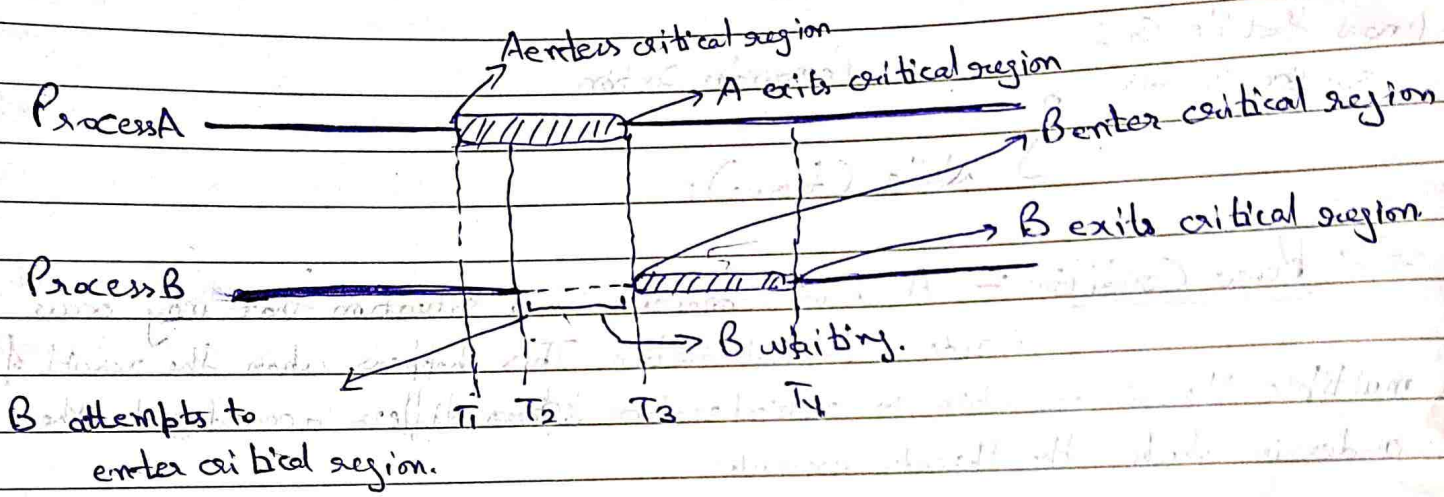
=> **Mutual Exclusion** :- A way of making sure that if one process is using a shared modifiable data, the other processes will be excluded from doing the same thing.

Requirements for Mutual Exclusion :-

- Must be enforced
- A process that halts must do so without interfering with other processes.

Date
Page No.

- No deadlock or starvation
- A process must not be denied access to a critical section when there is no other process using it
- No assumptions are made about relative process speeds or number of processes.
- A process remains inside its critical section for a finite time only.



⇒ The Producer / Consumer Problem :-

The producer consumer problem is a synchronization problem. There is a fixed size buffer and the producer produces items and enters them into the buffer. The consumer removes the items from the buffer and consumes them.

A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice-versa. So the buffer should ~~not~~ only be accessed by the producer or consumer at a time.

Date
Page No.

The producer consumer problem can be resolved using semaphores. The codes for the producer and consumer process are given as follows:-

Semaphores :- ① S=1 for mutual exclusion, only one person is accessing Critical Section.

② E=n → Size of the buffer
↓
Empty

③ F=0
↳ full

Two atomic microoperations :- wait and Signal.
wait → decrements the count
Signal → increments the count

Producer :-
```
do{
wait(E); //wait until empty(E) > 0 and then decrement
wait(s); //Lock
//Add item to Buffer
Signal(s); // Released
Signal(F); // Increment full
}
while(true);
```

Consumer :-
```
do{
wait(F); //wait until full > 0 and then decrement
wait(s); // Acquire lock
// Consume item
Signal(s); //Released
Signal(E); // Increment empty;
}while(true);
```

Date
Page No.

⇒ Semaphores :- Semaphores are integer variable that are used to solve the critical section problem by using two atomic operations., wait and signal that are used for process synchronization.

- Wait :- The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

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

Signal :- The Signal operation increments the value of its argument S.

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

Types of Semaphores :- There are two main types of semaphores :-

① Counting Semaphores :- These are integer values semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count automatically incremented and vice-versa.

② Binary Semaphores :- The Binary Semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Date
Page No.

Ad :- ① Semaphores allow only one processes into the critical section. They follow the mutual exclusion principle strictly and are much more efficient than some other methods of synchronization.

② There is no resource wastage because of busy waiting in Semaphores as processor time is not wasted unnecessarily to check if a condition is fulfilled to allow a process to access the critical section.

③ Semaphores are implemented in machine independent code of microkernel. So they are machine independent.

disad :- These are complicated so the wait and signal operations must be implemented in the correct order to prevent deadlocks.

← It may lead to a priority inversion where low priority processes may access the critical section first and high priority processes later.

⟹ Monitors :- Monitors are a synchronization construct that were created to overcome the problems caused by Semaphores such as timing errors.

Monitors are abstract data types and contain shared data variable and procedures. The shared data variables cannot be directly accessed by a process and procedures are required to allow a single process to access the shared data variables at a time.

Only one process can be active in a monitor at a time. Other processes that need to access the shared variables in a monitor have to line up in a queue and are only provided access when the previous process release the shared variables.

Date
Page No.

Syntax :-

```
Monitor Demo          // Demo is the name of monitor
{
    Variables;
    condition variables;
    procedure p1 {.....}
    procedure p2 {.....}
    Initialize {.....}
}
```

Ad :- Makes parallel processing easier
:- less error prone than using techniques such as semaphore.

Eg :-

```
Monitor inca;
EXPORT IncrementX, ReadX;
int x;                    /* Variables shared */
GUARD PROCEDURE IncrementX()
{
    Int Temp;
    Temp = x;
    Temp = Temp + 1;
    x = Temp;
}
GUARD PROCEDURE ReadX()
{
    Return (x);
}
Initialize                /* initialize variable */
{
    x = 0
}
```

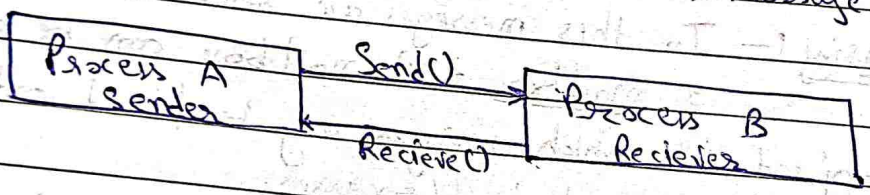
Date
Page No.

⇒ Message Passing :-

It refers to the means of communication between
- different threads within a process
- different processes running on same node
- different process running on different node.

In this a sender or a source process sends a message to a known reciever or destination process.
Message has a predefined structure and message passing uses two system call :- Send and Recieve.

Send (name of destination process, Message);
Recieve (Name of ~~destin~~ source process, Message);



In this call, the sender and ~~se~~ reciever processes address each other by names!
Mode of communication b/w two processes can take place through two methods :-

- Direct Addressing :- In this type, the two processes need to name each other to communicate.

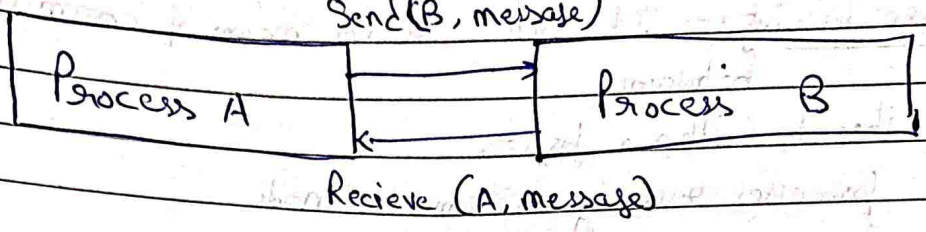
This becomes easy if they have the same Parent. eg :-

If process A sends a message to process B, then

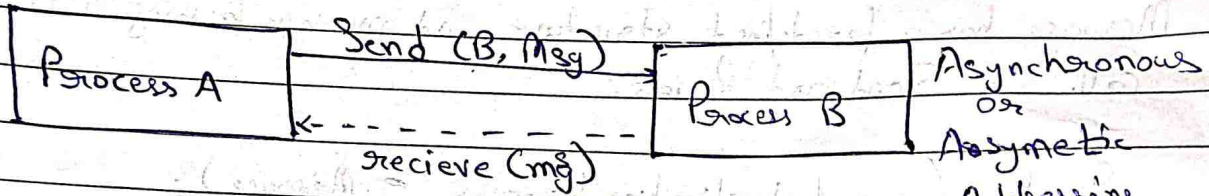
Send (B, message);
Recieve (A, message);

By message passing a link is established b/w A and B.
Here the reciever knows the identity of sender message destination.
This type in direct communication, is known as Symmetric addressing.

Date
Page No.

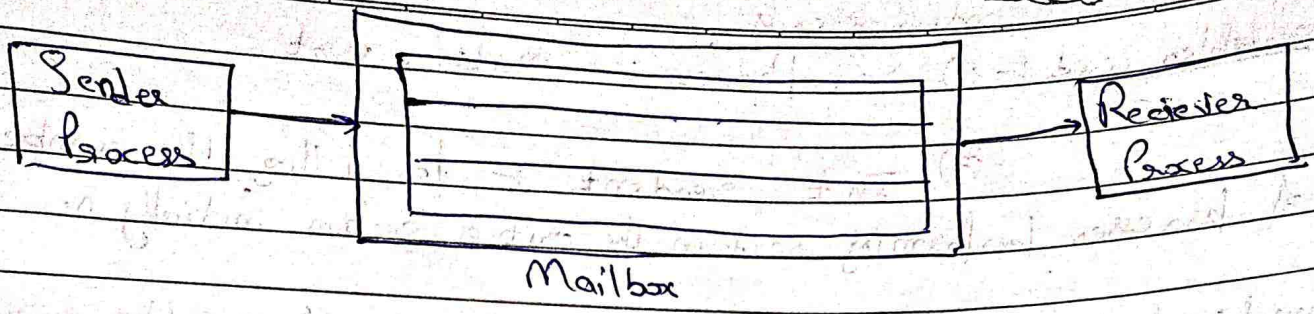


Another type of addressing known as assymetric addressing where reciever does not know the ID of the sending process in advance.



Indirect addressing :- In this messeges are sent and recieved from a mailbox. A mail box can be abstractly viewed as an object into which messages may be placed and from which messages may be removed by processes.
The sender and reciever processes should share a mail box to communicate.

→ One to One link :- One Sender wants to communicate with one reciever. Then single link is established.
→ Many to one link :- Multiple Senders wants to communicate with single reciever. eg :- in Client-Server system, there are many clients processes and one server process.
The mail box here known as PORT.
→ One to Many link :- One sender wants to communicate with multiple recievers, that is, to broadcast a message.
→ Many to Many link :- Multiple Sender wants to communicate with multiple resources.



⇒ Classical IPC problems :-

The operating system literature is full of interprocess communication problems that have been widely discussed using a variety of synchronization methods.

⇒ Readers - Writers Problems :-

Consider a situation where we have a file shared between many people.

- If one of the writer tries editing the file, no other is reading or writing at the same time, otherwise changes will not be visible to him / her.
- However if some person is reading the file, then others may read it at the same item.
- One set of data is shared among a number of processes.
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it.
- If at least one reader is reading, no other process can write.
- Readers may not write and only read.

Mutex is used for mutual exclusion. when readcnt is updated. i.e. when any reader enters or exits

Date
Page No.

Variables used :- ① Semaphores :- mutex, wrt

② Int readcnt :- for telling the number of processes performing read in the critical section initially 0.

Function for Semaphores :- wait () :- decrements the semaphore value

:- Signal () :- Increments the semaphore value.

Writer process :-

```
do {
    wait (wrt); // writer requests for critical section.
    // performs the write.
    Signal (wrt); // leaves the critical section
} while (true);
```

Readers Process :-

```
do {
    wait (mutex); // Reader wants to enter the critical section
    readcnt++; // The number of readers has increased by 1.
    // there is atleast one reader in the critical section
    // this ensure no writer can enter if there is atleast one reader
    // thus readers have a preference.
    if (readcnt == 1)
        wait (wrt);
    // other readers can also enter while reader is inside critical section
    Signal (mutex);
    wait (mutex);
    readcnt--;
```

// a reader wants to leave

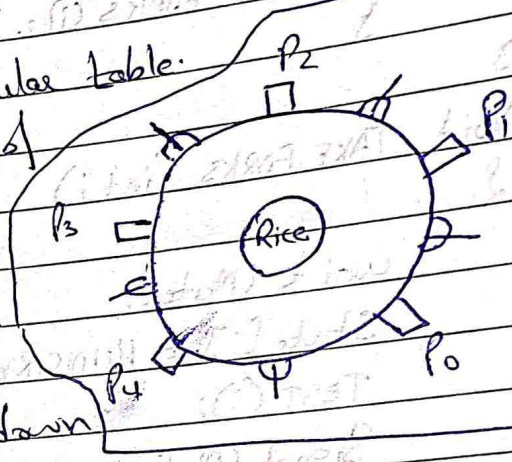
// current reader performing reading // a reader wants to

Date
Page No.

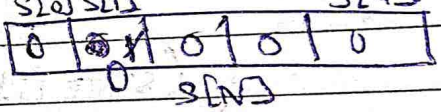
// no reader is left in the critical section
if (readcnt == 0)
    Signal (wrt); // writers can enter
Signal (mutex); // reader leaves.
} while (true);

⇒ Dinning – Philosophers Problem :-

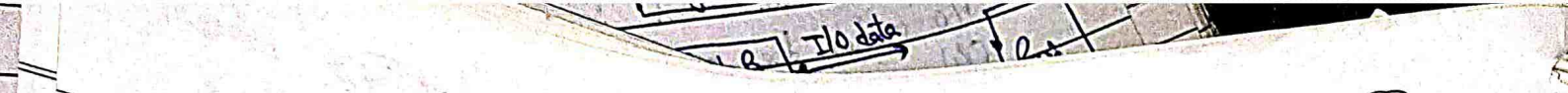
:- Five philosphers are sitting around a circular table.
:- Dining table has 5 chopsticks., 2 bowl of rice in the middle.
:- Philosopher either Eat or think.
:- When philosopher wants to eat, he use 2 chopsticks.
:- When he wants to think, he keeps down chopsticks at their place.



Problem :- Develop an algorithm where no philosphers starves i.e. Every phy. should eventually get a chance to eat




```
#define N 5                    // Number of Philosphers
#define Thinking 0
#define HUNGRY 1               // states of philosphers
#define EATING 2
#define LEFT (i+N-1) % N.      // Left of philosphers
#define RIGHT (i+1) % N.       // Right of Philosphers
B Semaphore Mutex = 1;
B Semaphore S[N] = {0};        // S[N] -> Array of B Semaphore
int STATE[N] = 0;              // Array to keep Track of Every One's state.
// STATE[N] = 0 means every philosopher is thinking
```



Date
Page No.

```
Void PHILOSPHER (int i)                    // i is the index of philosopher
{
    while (TRUE)
    {
        Thinking ();                       // philospher is thinking
        TAKE FORKS (i);                    // Head towards Take forks function
        EAT ();                            // philospher eats
        PUT FORKS (i);
    }
}
Void TAKE FORKS (int i)
{
    wait (Mutex);
    State [i] = HUNGRY                     // Philospher is hungry.
    TEST (i);                              // Head towards Test function.
    Signal (Mutex);
    wait (S[i]);                           // the value becomes 1 to 0 for philospher
}
Void Test (int i)
{
    If (STATE [i] == HUNGRY && STATE[LEFT] != EATING &&
        STATE [RIGHT] != EATING) // Checks if the philospher
    {                                      // is hungry and both left and right are not eating
        STATE[i] = EATING                  // philospher eats
        Signal (S[i]);                     // the value becomes 0 to 1 for philospher
    }
}
```

Date
Page No.

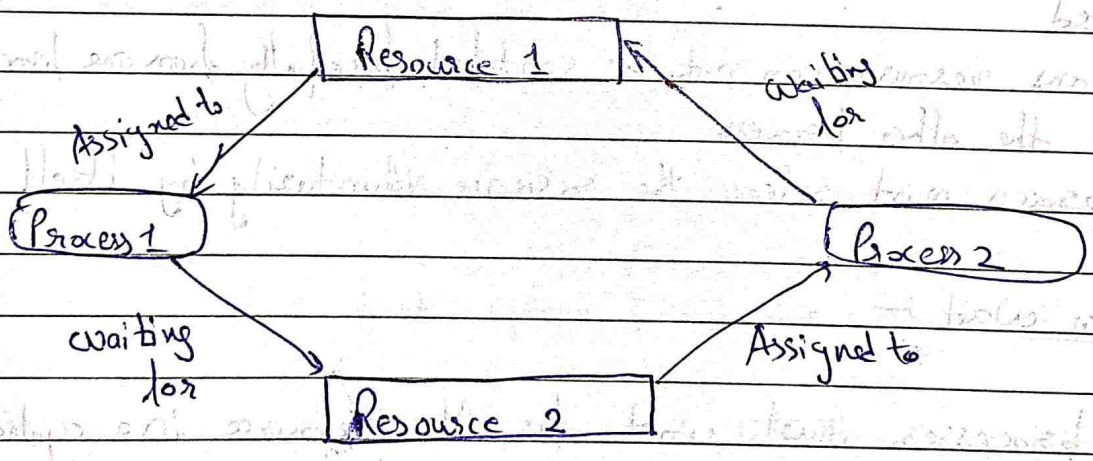
```
Void PUTFORKS(int i)
{
    wait(Mutex);
    STATE[i] = THINKING      // Now philospher thinking
    TEST (LEFT) ;            // Test Right philosopher
    TEST (RIGHT);            // test Right philospher
    Signal (Mutex);
}
```

⟹ Deadlocks

A process in operating systems uses different resources and uses resources in following way :-

1) Requests a resource
2) Use the resource
3) Releases a resource

Deadlock is a situation where a set of processes are blaked because each process is holding a resource and waiting for another resource acquired by some other process.



⇒ Neccessary and Sufficient Conditions for Deadlock :-

There are 4 necessary conditions for occurence of deadlock :-

① Mutual Exclusion :-

- There must exist at least one resource in the system which can be used by only one process at a time.
- If there exists no such resource, then deadlock will never occur.
- Printer is an example of a resource that can be used by only one process at a time.

② Hold and Wait :-

- There must exist a process, which holds some resources and waits for another resource held by some other process

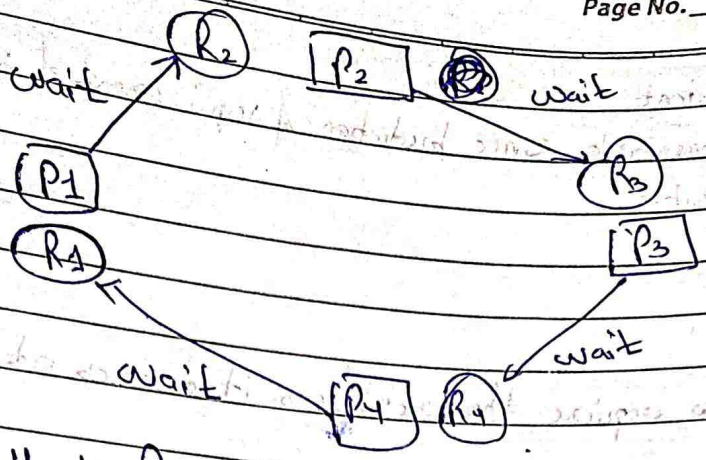
③ No Preemption :-

- Once the resource has been allocated to the process, it cannot be preempted
- It means resource can not be snatched forcefully from one process and given to the other process
- The process must release the resource voluntarily by itself

④ Circular wait :-

All the processes must wait for the resource in a cyclic manner where the last process waits for the resource held by the first process.

Date
Page No.



⇒ Deadlock Prevention :-

- This strategy involves designing a system that violates one of the four necessary conditions required for the occurence of deadlock.
- This ensures that the system remains free from the deadlock.

(1) Mutual Exclusion :-

- To violate this condition, all the systems resources must be such that they can be used in a shareable mode.
- In a system, there are always some resources which are mutually exclusive by nature.
- So, this condition cannot be violated.

(2) Hold and Wait :-

(a) Approach - 1

- A process has to first request for all the resources it requires for execution
- Once it has acquired all the resources, only then it can start its execution
- This approach ensures that the process does not hold some resources and wait for other resources.

drawback :- Less efficient
:- Non implementable since prediction of requirement of resources is not possible.

(b) Approach - 2 :-

- A process is allowed to acquire the resources it desires at the current moment.
- After acquiring the resources, it starts its execution.
- Now before making any new request, it has to compulsarily release all the resources that it holds currently.
- This approach is efficient and implementable.

(c) Approach - 3 :-

- A timer is set after the process acquires any resources.
- After the time expires, a process has to compulsarily release the resource.

(3) No Preemption :-

- This condition can be violated by forceful preemption.
- Consider a process is holding some resources and request other resources that can not be immediately allocated to it.
- Then, by forcefully preempting the currently held resources, the condition can be violated.
- The process is preempted only if :-
  -> It is a high priority process or a system process
  -> The victim process is in the waiting state.

Date
Page No.

(4) Circular Wait :- This condition can be violated by not allowing the processes to wait for resources in a cyclic manner. By following:-

- A natural number is assigned to every resource.
- Each process is allowed to request for the resources either in only increasing or only decreasing order of the resource number.
- In case increasing order is followed, if a process requires a lesser number resource, then it must release all the resources having larger number and vice-versa.
- This approach is the most practical approach and implementable.
- However, this approach may cause starvation but will never lead to deadlock.

⇒ Deadlock Avoidance : Bankers's Algorithm :-

Deadlock avoidance can be done with Bankers's Algorithm

Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs for Bankers's Algorithm :-

⇒ n = no. of processes

⇒ m = no of resource types

:- Available$_m$ = Vector of length m = if Available[j] = k, there are k instances of resource R_j available

- Max$_{[n \times m]}$:- 'n x m' matrix, if Max[i,j] = k, Process P_i may request at most k instances of resource R_j.
- Allocation$_{[n \times m]}$:- 'n x m' matrix, Allocation[i,j] = k, then P_i is currently allocated k instances of resource R_j.

Date
Page No.

:- $Need_{[n \times m]}$ = 'n x m' matrix, if $Need[i,j] = k$, Process P_i may need k more instances of resources R_j to complete.

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Request Resource Algorithm :-

Let Praess $P_i \rightarrow Request_i$;

(i) If $Request_i \leq Need_i$, go to step 2, else error

(ii) If $Request_i \leq Available$, go to step 3, else wait

(iii) $Available = Available - Request_i$;
$Allocation_i = Allocation_i + Request_i$;
$Need_i = Need_i - Request_i$;

(iv) Check if the new state is safe or Not by using Safety Algorithm.

Safety Algorithm :-

(i) Work = Available
~~Finish~~ $Finish_i = False$

(ii) Find an 'i' such that
$Finish[i] = False$ and
$Need_i \leq Work$
if no such i, go to step 4
else go to step 3

(iii) $Finish[i] = True$;
$Work = Work + Allocation_i$;
Go to step 2.

(iv) If Finish[i] = true for all i, the system is safe.

Example :- 5 Processes (P_0, P_1, P_2, P_3, P_4)

3 resource types :- (A B C)
Instances :- 10 5 7

Snapshot at time T_0 :-

	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			

Need Matrix

	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

Now following Safety algorithm

(i) Work = Available
Work = [3 3 2]
Finish$_i$ = false

For i = 0 P_0 can't be allocated at the moment

For i = 1 Need$_i$ < Work.
∴ Finish[1] = True
Work = Work + Allocation
= [3 3 2] + [2 0 0]
= [5 3 2]

For i = 2 P_2 can't be allocated at the moment

For i = 3 Need ≤ Work
∴ Finish[3] = True.
Work = [5 3 2] + [2 1 1]
= [7 4 3]

Date
Page No.

For i=4, Finish[4] = True
Work = [743] + [002]
= [745].

Now again For i=0, Finish[0] = True
Work = [745] + [010]
= [755]

For i=2, Finish[2] = True
Work = [755] + [302]
= [10 5 7].

Hence the system is in Safe state
and Safe sequence is [$P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_0 \rightarrow P_2$]

Example :- If Process P_1 requests (1, 0, 2) (A B C) check whether this will be safe or not.

5 Processes (P_0 ..... P_4)
Resources — A [10]
B [5]
C [7]

Snapshot at time T_0 :-

	Allocation	Max	Available
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	

Following the Resource Request algo :-

$P_1 \rightarrow R(1, 0, 2)$

(i) Need $(P_1) = (322) - (200) = (122)$ Request[i] ≤ Need[i]

(ii) Also Request[1] ≤ Available[1]

(iii) Available = (332) − (102)

= (230)

Allocation $[P_1]$ = (200) + (0102)

= (302)

Need $[P_1]$ = (122) − (102) = (020)

Now following Safety Algorithm

Need Matrix

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

Available = [230]

(i) Work = 230, Finish = false

For i = 0, No

For i = 1, Finish[1] = True

Work = 230 + 302 = 532

For i = 2, No

For i = 3, Finish[3] = True

Work = 532 + 211 = 743

For i = 4, Finish[4] = True

Work = 743 + 002 = 745

For i = 2, Finish[2], True

Work = 745 + 302 = 1047

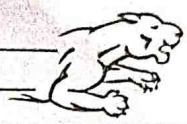
for i = 0, Finish[0], True

Work = 1047 + 010 = 1057

Hence the system is safe and resources requested by P_1 can be allocated to it

Safe sequence :— $\{P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_0\}$

Date
Page No.



⇒ Deadlock Detection and Recovery :-

Deadlock Detection :-

1. If resources have single Instance :- In this case for deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.

2. If there are multiple instances of resources :- Detection of the cycle is ~~necessary~~ necessary but not sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.

Deadlock Recovery :-

A traditional OS such as Windows doesn't deal with deadlock recovery as it is time and space consuming process. Real-time OS use deadlock recovery.

① Killing the process :- Killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.

② Resource Preemption :- Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Operating System

Date.....

Unit-3

Memory Management

Memory Management is the functionality of an OS which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

Memory Management keeps tracks of each and 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 and correspondingly it updates the status.

⟹ Logical and Physical Address :-

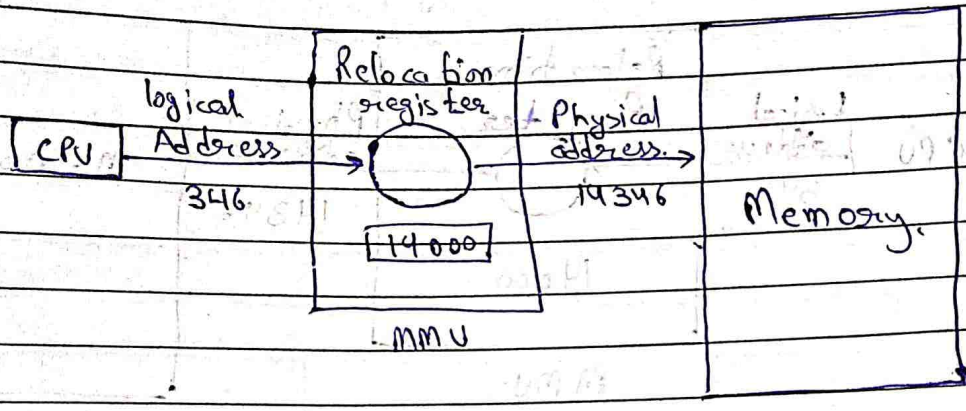
Logical Address is generated by CPU while a program is running. This is a virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU.

The term logical Address Space is used for the set of all logical addresses generated by a programs perspective.

Date.....

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address.

The logical address must be mapped to the physical address by MMU before they are used. The term Physical address space is used for all physical addresses corresponding to the logical addresses in a Logical address space. The hardware device called MMU is used for mapping logical address to its corresponding physical address.



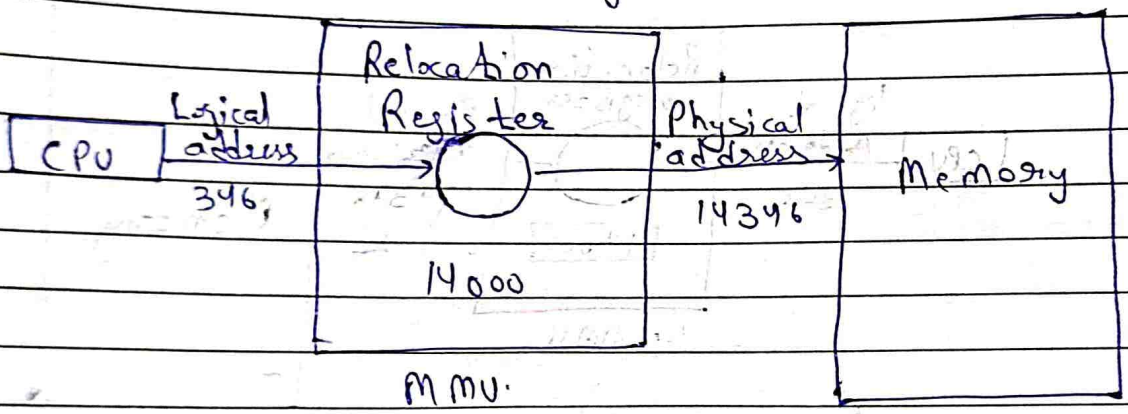
	Logical Address	Physical Address
• Basic.	Generated by CPU	location in a memory unit
• Address space	It is set of all logical addresses generated by CPU in reference to a program.	It is set of all physical addresses mapped to the corresponding logical addresses
• Visibility	User can view the logical addres of a program	User can never view physical address of program.
• Generation	Generated by the CPU.	Computed by MMU.
• Access	The user can use the logical address to access the physical address	The user can not directly access it.

Spiral

Date.....

The run time mapping from virtual to physical address is done by the memory management unit which is a hardware device. MMU uses following mechanism to convert virtual address to physical address :-

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. eg:- if the base register value is 14000, then an attempt by the user to use address location 346 will be dynamically reallocated to location 14346.



⟹ Memory Allocation :- It is an action of assigning the physical or the virtual memory address space to a process.

⟹ Contiguous Memory Allocation :-

It is a memory allocation technique. It allows to store the process only in a contiguous fashion. Thus, entire process has to be stored as a single entity at one place inside the memory.

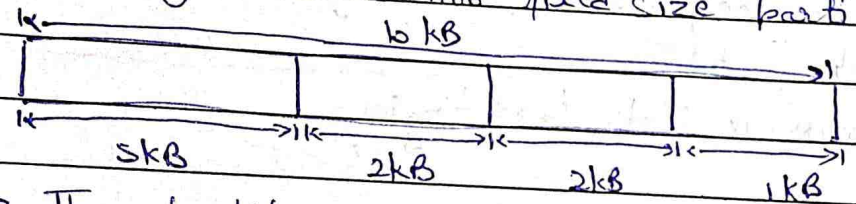
Date.....

Techniques :- There are two popular techniques used for contiguous memory allocation.

- Static Partitioning
- Dynamic Partitioning

① **Static Partitioning :-** It is a fixed size partitioning scheme.
→ In this technique, main memory is predivided into fixed size partitions.
→ The size of each partition is fixed and not be changed.
→ Each partition is allowed to store only one process.

eg :- Under fixed size ~~partition~~ partitioning scheme, a memory of size 16 KB may be divided into fixed size partitions as :-



→ These partitions are allocated to the processes as they arrive.
→ The partitions allocated to the arrived process depends on the algorithms

Algorithms for Partition Allocation :-

① **First fit Algorithm :-** This algorithm starts scanning the partitions serially from the starting.
→ When an empty partition that is big enough to store the process is found, it is allocated to the process.
→ Obviously, the partition size has to be greater than or at least equal to the process size.

Date.....

② Best fit algorithm :- This algorithm first scans all the empty partitions.
:- It then allocates the smallest size partition to the process.

③ Worst fit algorithm :- This algorithm first scans all the empty partitions.
⇒ It then allocates the largest size partition to the process.

⇒ Internal fragmentation :-

→ It occurs when the space is left inside the partition after allocating the partition to a process.
→ This space is called as internally fragmented space.
→ This space can not be allocated to any other process.
→ This is because only static partitioning allows to store only one process in each partition.
→ It only occurs in static partitioning.

⇒ External fragmentation :-

→ It occurs when the total amount of empty space required to store the process is available in the main memory.
→ But because the space is not contiguous, so the process can not be stored.

Ad :- The advantages of static partitioning are :-
→ Simple and easy to implement
→ Supports multiprogramming
→ One memory access is required, thus reduces access time.

Spiral

Date.....

Disad :- Suffers from both internal and external fragmentation.
:- Utilizes memory inefficiency
:- Degree of multiprogramming is limited equal to number of partitions.
:- There is a limitation on the size of process since processes with size greater than the size of largest partition can't be stored and executed.

(2) Dynamic Partitioning :- It is a variable size partitioning scheme.
:- It performs the allocation dynamically.
:- When a process arrives, a partition of size equal to the size of process is created.
:- Then that partitions is allocated to the process

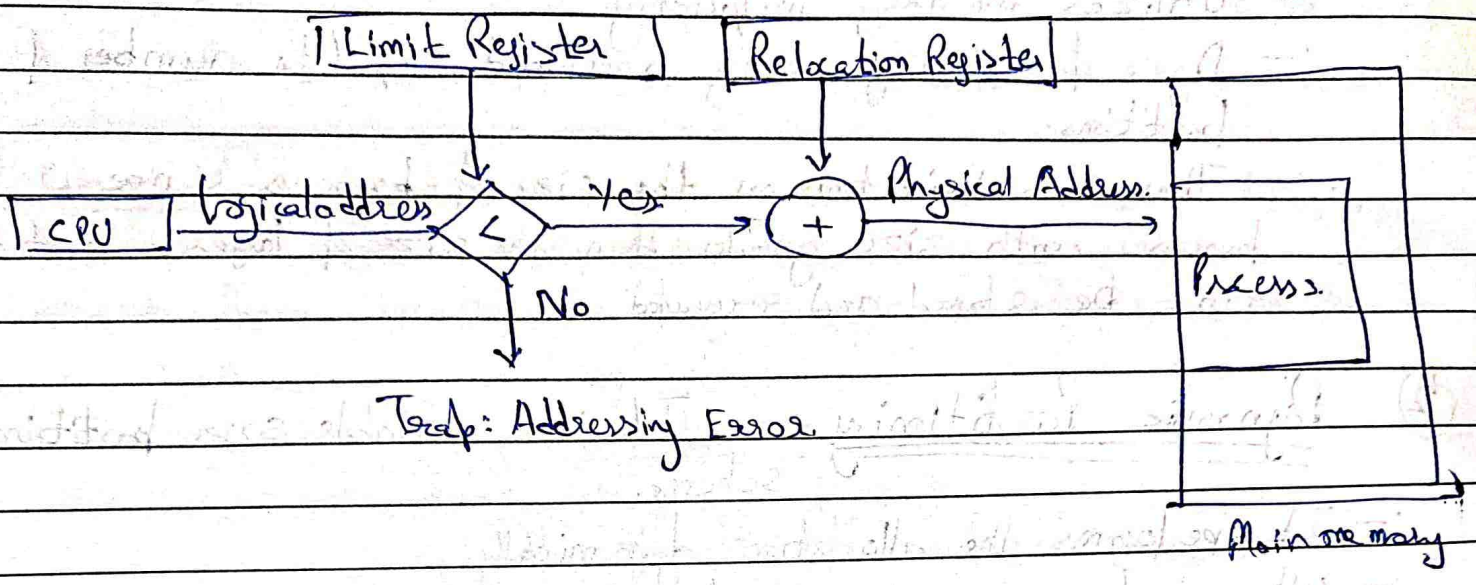
→ Partition Allocation Algorithm :- The processes arrive and leave the main memory. As a result holes of different size are created in the main memory. These holes are allocated to the processes that arrive in future.

Adv :- Does not suffer from internal fragmentation
:- Degree of multiprogramming is dynamic
:- There is no limitation on the size of processes

Disad :- Suffers from external fragmentation
:- Allocation and deallocation of memory is complex.

Date.....

Translating Logical Address into Physical Address



⟹ Compaction :-

:- Compaction is a process in which the free space is collected in a large memory chunk to make some space available for processes.

:- In memory management, swapping creates multiple fragments in the memory because of the processes moving in and out.

:- It refers to combining all the empty spaces together.

:- It helps to solve the problem of fragmentation, but it requires too much of CPU time.

:- It moves all the occupied areas of store to one end and leaves one large free space for incoming jobs, instead of numerous small ones.

:- In compaction, the system also maintains relocation information and it must be performed on each new allocation of job to the memory or completion of job from memory.

:- It can minimize the probability of external fragmentation.

:- This technique is also called defragmentation.

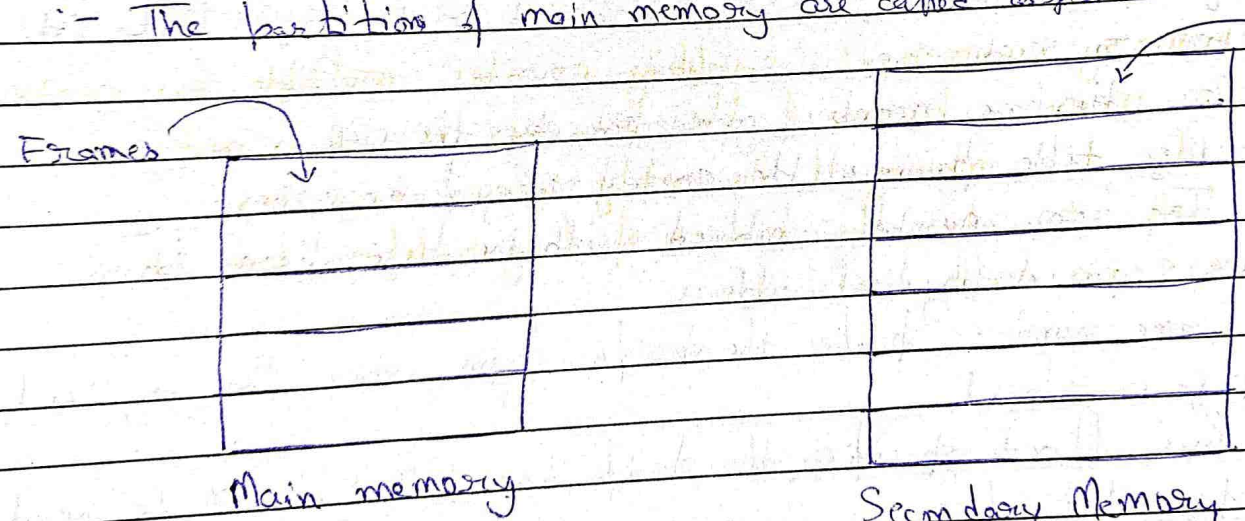
Date.....

disad :- The efficiency is decreased due to the fact that all free spaces will be transfered from several places to single place.
:- Huge amount of time is required.
:- CPU will remain idle for all this time.

⟹ Non-Contiguous Memory Allocation :-

- This technique allows to store parts of a single process in a non-contiguous fashion.
- Thus, different parts of the same process can be stored at different places in the main memory.

Paging :- Paging is a fixed size partitioning scheme.
:- In paging, secondary memory and main memory are divided into equal fixed size partitions.
:- The partitions of secondary memory are called as pages.
:- The partition of main memory are called as frames.



:- Each process is divided into parts where size of each part is same as page size.
:- The pages of process are stored in the frames of main memory depending upon their availability.

Spiral

Date.....

ad:- It allows to store parts of a single process in a non-contiguous fashion.
:- Solves the problem of external fragmentation.

disad:- Suffers from internal fragmentation
:- An overhead of maintaining a page table for each process.
:- Time taken to fetch the instruction increases since now two memory accesses are required.

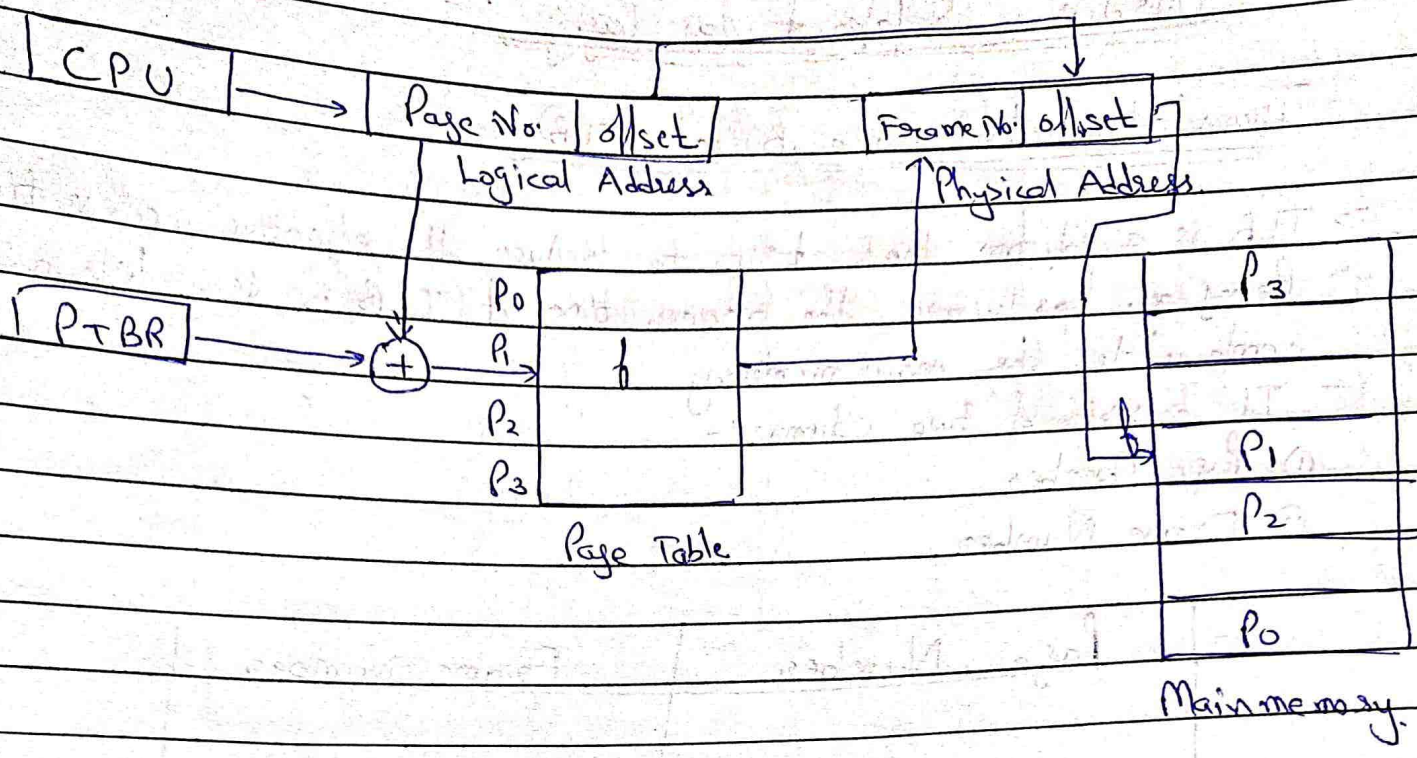
→ Translating Logical Address into Physical Address :-

Step-1 :- CPU generates a logical address containing of two parts :-
1) Page number
2) Page offset
→ Page number specifies the specific page of the process from which CPU wants to read the data.
→ Page offset specifies the specific word on the page that CPU wants to read.

Step-2 :- For the page number generated by the CPU,
:- Page table provides the corresponding frame number

Step-3 :- The frame number combined with the page offset forms the required physical address.
:- Frame number specifies the specific frame where the required page is stored.
:- Page offset specifies the specific word that has to be read from that page.

Date.....



⟹ Page Table :-

:- Page Table is a data structure.
:- It maps the page number referenced by the CPU to the frame number where that page is stored
:- Page table is stored in the main memory.
:- No. of entries in a page table = No. of pages in which the process is divided.
:- Page Table Base Register (PTBR) contains the base address of page table.
:- Each process has its own independent page table

Date.....

⇒ Hardware Support for Paging :-

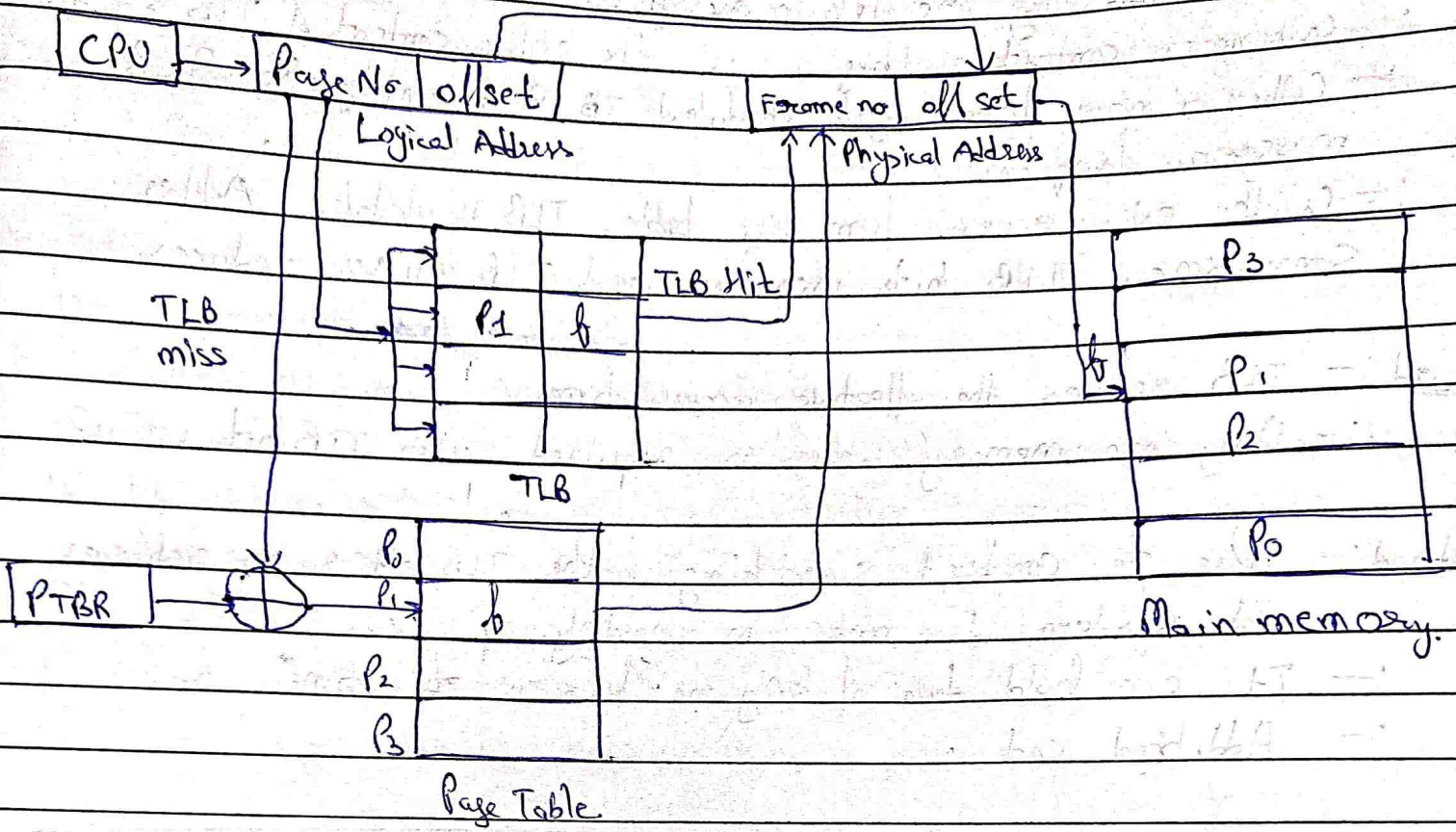
Translation Lookaside Buffer (TLB) :-

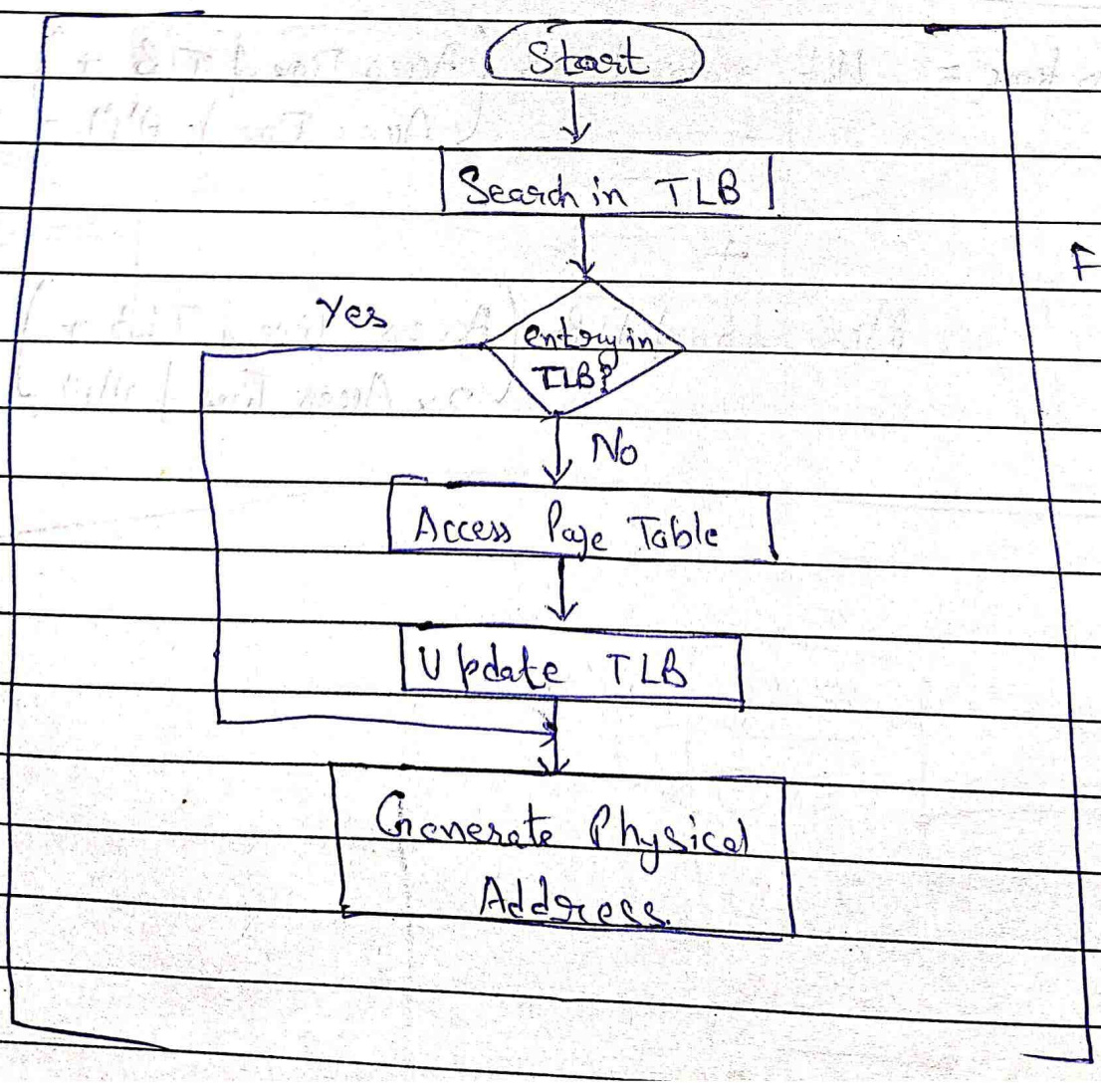
→ TLB is a solution that tries to reduce the effective access time.
→ Being a hardware, the access time of TLB is very less as compared to the main memory.
:- It consists of two columns:-
① Page Number
② Frame Number

Page Number	Frame Number

:- The logical address generated by the CPU is translated into the physical address using following steps:-
① CPU generates a logical address consisting of two parts:-
Page Number
Page offset
② TLB is checked to see if it contains an entry for the page no.
③ After the frame number is obtained, it is combined with the page offset to generate the physical address.
④

Date.....





Flowchart

Date.....

:- There exists only one TLB in one system.
:- Whenever context switching occurs, the entire content of TLB is flushed.
:- When a new process gets scheduled TLB is empty so, TLB misses are frequent.
:- With every access from page table, TLB is updated. After some time, TLB hits increases and TLB misses reduces.

adv :- TLB reduces the effective access time.
:- Only one memory access is required when TLB hit occurs.

disadv :- Due to context switching the TLB hit ratio reduces and system does not run smoothly.
:- It can hold data of only one process at a time.
:- Additional cost.

Effective Access time = Hit ratio of TLB (Access Time of TLB + Access Time of MM)

+

Miss ratio of TLB (Access Time of TLB + 2x Access Time of MM)

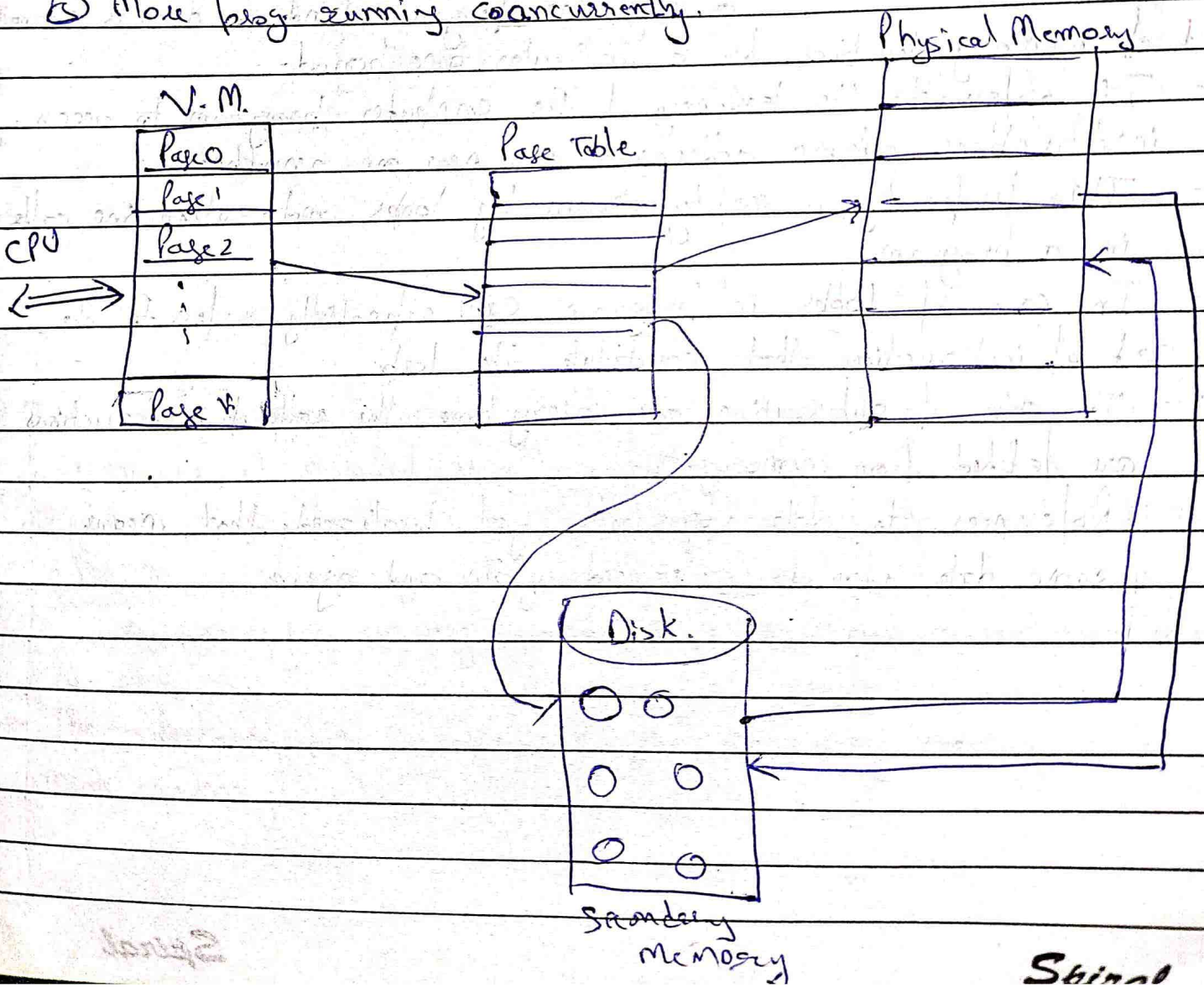
Date.....

Virtual Address :- A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.

:- The main visible advantage of this scheme is that programs can be larger than physical memory.

:- Virtual memory serves ~~two~~ purposes :-

1. It allows us to extend the use of physical memory by using disk.
2. It allows us to have memory protection, because each virtual address is translated to a physical address.
3. Logical address space can therefore be much larger than physical.
4. Allows address space to be shared by several purposes.
5. More prog. running concurrently.



Shirsat

Date.....

⇒ Hardware and Control Structure :-

→ All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time.

→ This means that a process may be swapped in and out of main memory such that it occupies different regions of main memory at different times during the course of execution.

→ A process may be broken up into a number of pieces and these pieces need not be contiguously located in main memory during the execution.

⇒ Locality of Reference :- It refers to a phenomenon in which a computer program tends to access same set of memory locations for a particular time period.

:- It refers to the tendency of the computer program to access instructions whose addresses are near one another.

:- This property is mainly shown by loops and subroutine calls in a program.

:- In case of loops in program, CPU repeatedly refers to the set of instructions that constitute the loop.

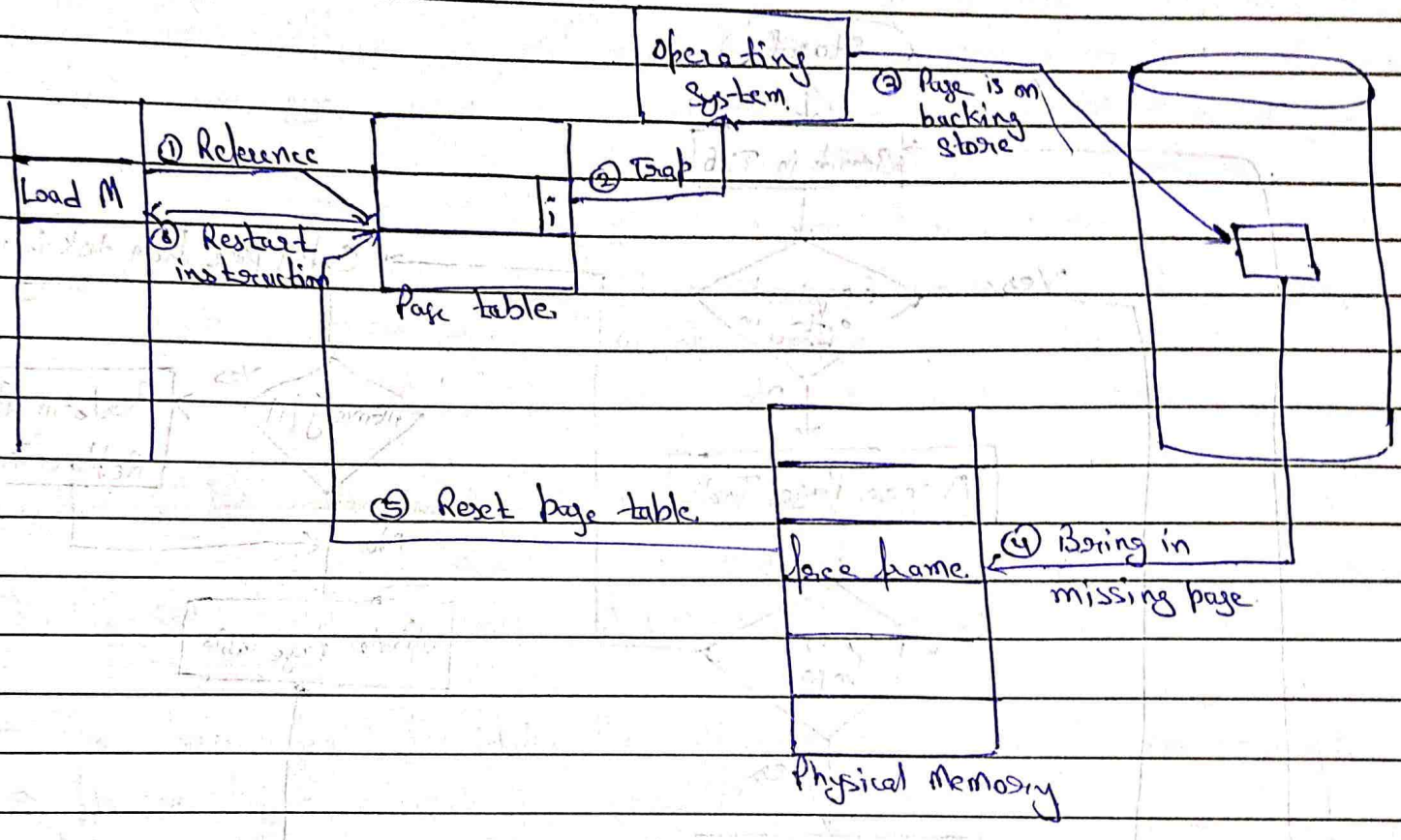
:- In case of subroutine calls, every time the set of instructions are fetched from memory.

:- References to data items also get localized that means same data item is referenced again and again.

Date.....

⇒ Page fault in Virtual Memory :-

It occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of event happens :-

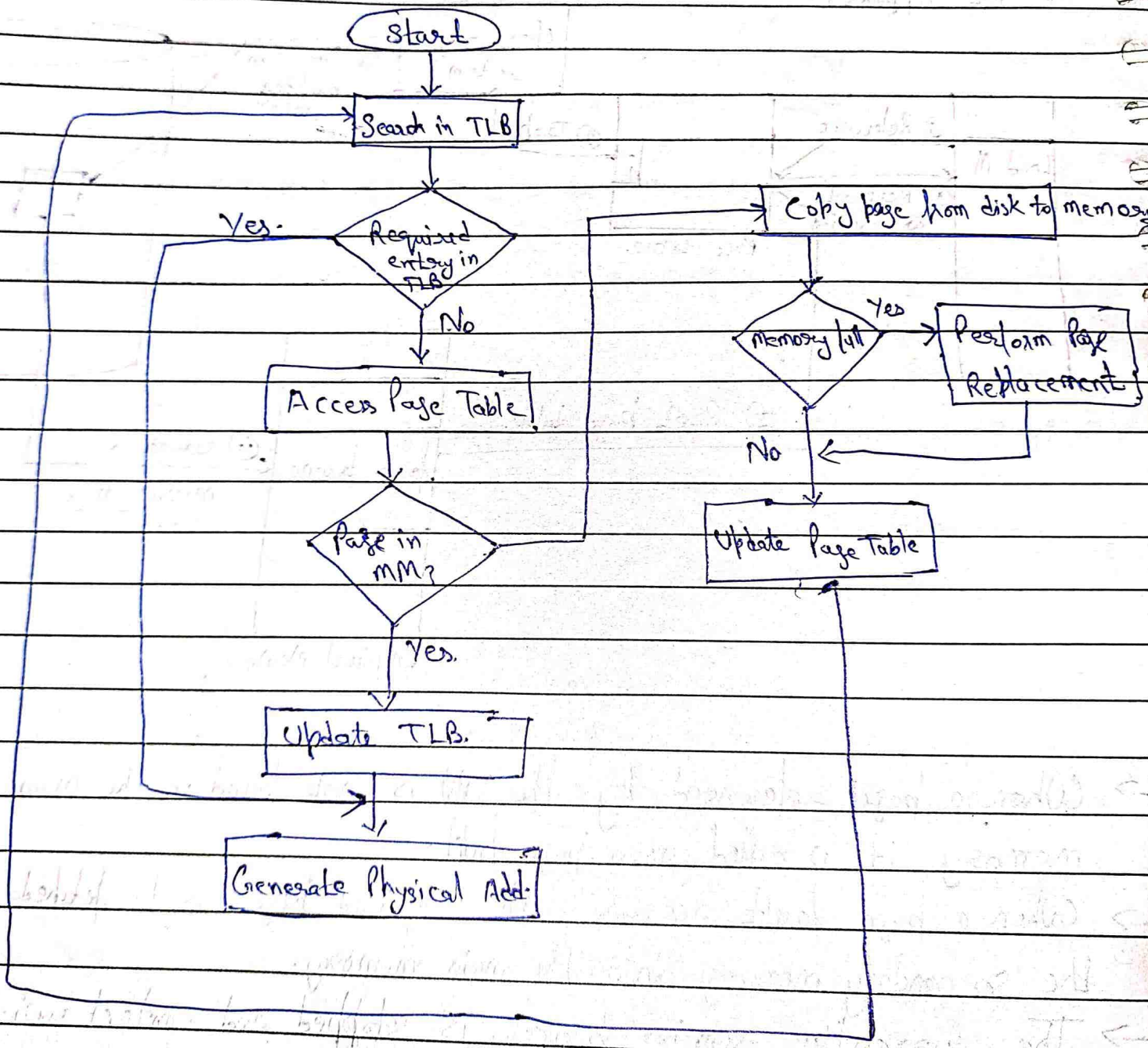


→ When a page referenced by the CPU is not found in the main memory, it is called as a page fault.
→ When a page fault occurs, the required page has to fetched from the secondary memory into the main memory.
→ The currently running process is stopped and context switching occurs.
→ The referenced page is copied from the secondary memory to the main memory.

Shirol

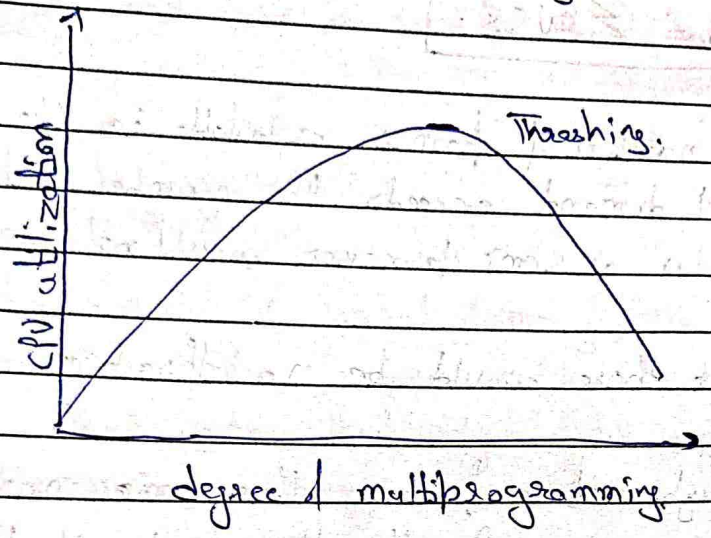
Date.....

→ If the main memory is already full, a page is replaced to create a room for the referenced page.
→ After copying the referenced page successfully in the main memory, the page table is updated.
→ Then the normal flow of execution resumes



Date.....

⇒ Thrashing :- It is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



~~Locality Model : A locality is a set of pages~~

Working Set Model :- This model is based on the Locality Model.

:- The basic principle states that if we allocate enough frames to a process to accomodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

⇒ A/c to this model, based on a parameter A, the working set is defined as the set of pages in the most recent 'A' page references. Hence all the actively used pages would always end up being a part of the working set.

Date.....

→ If D is the total demand for frames and WSS$_i$ is the working set size for a process i,

then

$$D = \Sigma WSS_i$$

Now if 'm' is the number of frames available in the memory,

- if D>m, total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- if D<=m, then there would be no thrashing.

→ If there are enough extra frames, then some more processes can be loaded in the memory. On the other hand, if the summation of working set sizes exceeds the availability of frames, then some of the processes have to be suspended. (swapped out of memory).

→ This technique prevents thrashing along with ensuring the highest degree of multiprogramming possible. Thus, it optimizes CPU utilization.

⇒ Dirty Bit :-

In order to reduce the page fault service time, a special bit called the dirty bit can be associated with each page.

→ The dirty bit is set to '1' by the hardware whenever the page is modified.

Date.....

→ When we select a victim by using a page replacement algorithm, we examine its dirty bit. If it is set, that means the page has been modified since it was swapped in. In this case, we have to write that page into the backing store.

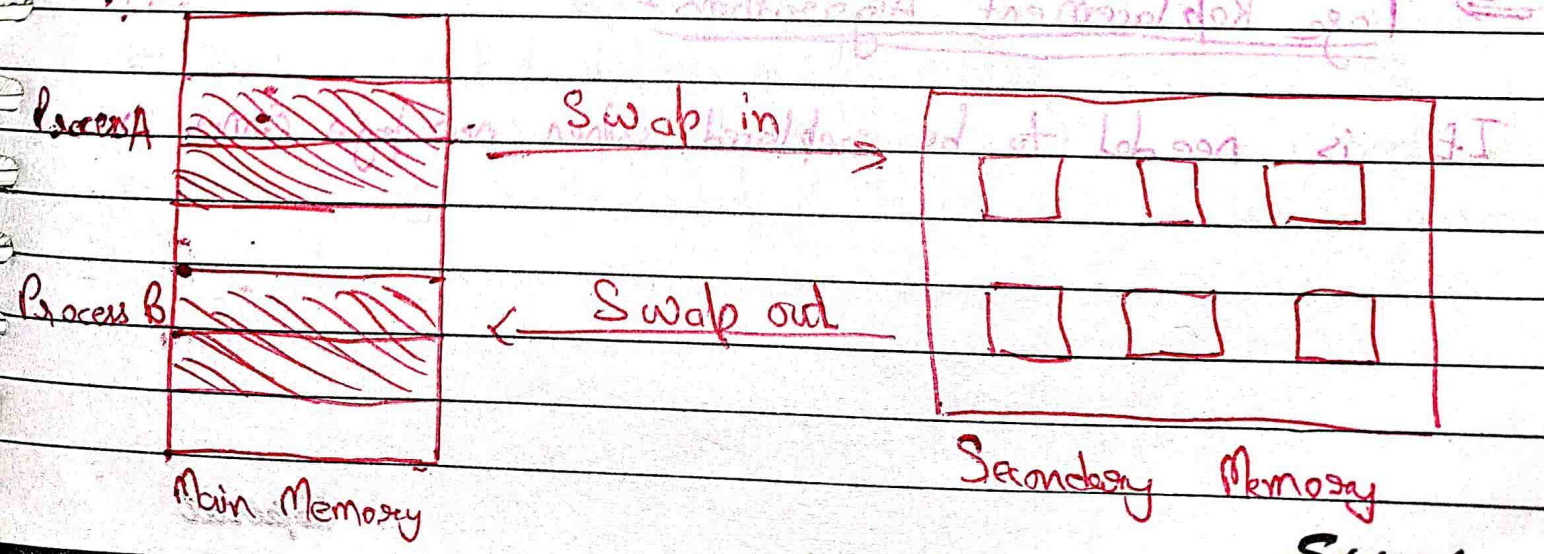
⇒ Demand Paging :-

According to the Concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However deciding, which pages need to be kept in the main memory and which need to be kept in disk, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

To overcome this problem, there is a concept called Demand Paging is introduced.

Demand paging is the process that solves this by only swapping pages on Demand. Also known as lazy swapper.



Shirol

Date.....

→ The components involved in the process are:-
- → CPU
- → Main Memory
- → Interrupt
- → Secondary Memory
- → Logical Address Space
- → Physical Address Space
- → Page Table
- → Operating System.

Steps:- 1) A request may be made to CPU for a page that may not be available in the main memory in active status. Thus, it has to generate an interrupt
2) This is when the OS moves the process to be a blocked state as an interrupt has occured.
3) The OS then searches the given page in the Logical address space
4) Finally with the help of page replacement algorithm, replacements are made in the physical address space. Simultaneously page tables are updated.
5) CPU is informed about update and process gets back into the ready state

⇒ Page Replacement Algorithms:

It is needed to be replaced when new page comes in.....!

Date.....

⇒ **Optimal Page Replacement:-** In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Sequence string

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
				Hit		Hit		Hit	Hit			Hit	Hit		Hit	Hit	Hit

Page Hit = 9
Page fault = 9.

This algorithm is perfect but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

⇒ **First In First Out (FIFO):-**

This is the simplest page replacement algorithm. In this algorithm, the OS keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Date.....

Reference string :- 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
7	7	7	0	0	1	2	3	0	4	2	2	3	0	0
	0	0	1	1	2	3	0	4	2	3	3	0	1	1
		1	2	2	3	0	4	2	3	0	0	1	2	2
*	*	*	*		*	*	*	*	*	*		*	*	
				Hit							Hit			Hit

Page Hit = 3
Page faults = 12

⇒ Least Recently Used (LRU) algorithm :- This algorithm replaces the page which has not been refered for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this we look at the past instead of staring at future.

Reference string :- 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
7	7	7	2	2	2	2	4	4	4	0	0	0	2	2
	0	0	0	0	0	0	0	0	3	3	3	3	3	0
		1	1	1	3	3	3	2	2	2	2	1	1	1
*	*	*	*	*	*	*	*	*	*		*	*	*	
				Hit		Hit					Hit			

Page Hits = 3
Page faults = 12