INDEX

NAME: T. Prashanth Reddy STD.: B Tech SEC.: IT ROLL NO.: 12B170077 SUB.: Operating Systems

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		Textbook:		
		A. Silberschatz, P.B Galvin G. Gagne. "Operating Systems concepts"	SJT 310 A24	
		Reference: W. Stallings, O.S.		
		Quiz - 1 - 29th Jan, wed. > syl:- before 26th		
		Cat-2:- after cat 1 to Inverted page table		
		https://sites.google.com/site/osf1f2.		
		Quiz - III - 28th ~~March~~ April, Monday, 9-10 pm.		
		evaluation pattern. Continuity		
		Thurs - 20th March - Quiz-2 syllb: upto friday class ↳ Recovery from deadlock, process termination		
		MIT → CSAIL → comp sci artificial intelligence lab → Brain comp. Interface		

6/1/14

cmd to know abt O.S
- systeminfo.

Operating Systems

O.S. - system s/w controls set of h/w.

O.S

→ O.S is an pgm that controls the execution of app. pgms and acts as an interface. h/w user & computer hardware.

→ It is an pgm that manages the comp h/w

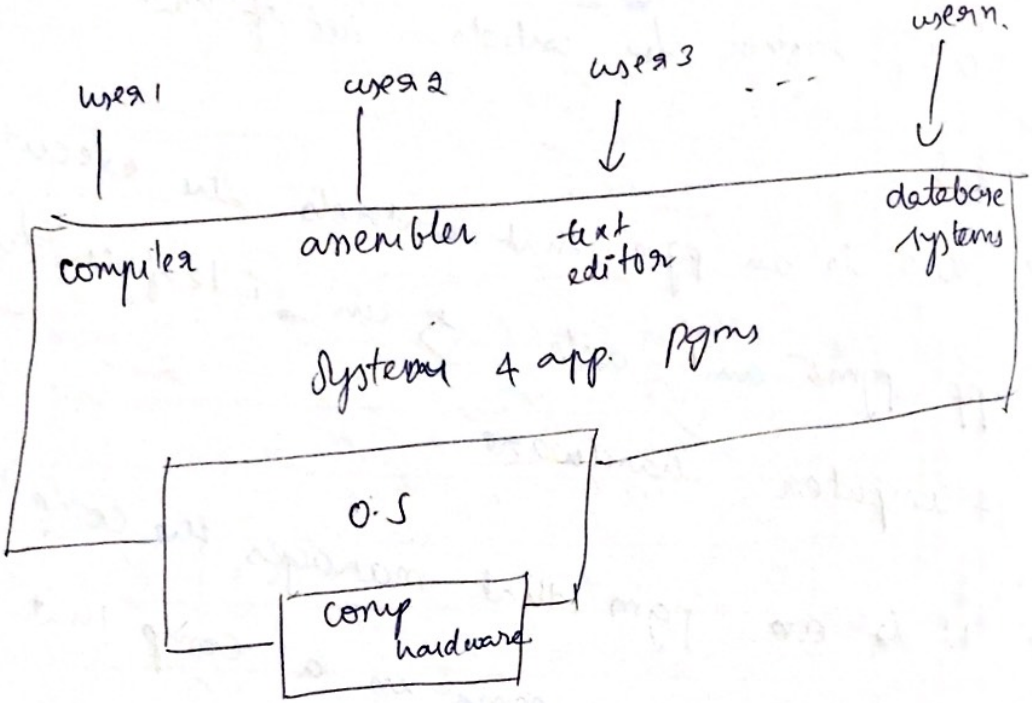
→ set of sys. s/w pgms in a comp. that regulates

Comp. Sys. Components

Compiler - H/L to L/L (low level) } Checks entire pgm

Translator - H/L to M/L } checks line by line

abstract view of sys components.



Pgm for adding 2 numbers in pneumonics

```
LOAD  A , 100
LOAD  B , 100
ADD   B
STORE A.
```

User View: - O.S

1) In PC PDA - personal digital assistance
2) In minicomputer & mainframe
3) In workstation
4) In handheld system
5) In embedded computer.

System - View - O.S

1. Resource Allocator:
   Manages and allocates resources
2. Control Program:
   Controls the execution of user pgms & operations of I/O devices
3. Kernel:
   - The one pgm running at all times.

History

1. Mainframe Systems:
   ENIAC - first comp.
   = Electronic numerical Integrated and Computer.

Batch Systems

similar jobs will be combined

1. Simple batch Systems :-
   multiprogrammed Batch Systems

Process States

1) Running
2. Ready
3. Waiting
4. Blocked

OS features needed for multiprogramming

1. I/O routines supplied by the sys
2. Memory management
3. CPU Scheduling
4. allocation of devices.

Time-Sharing Systems:

→ also called "Interactive System" or "Multitasking System"

→ short period of time during that a user gets attention of the CPU is known as "time slice" or "quantum".

OS Features needed:

1. Memory management
2. Virtual memory
3. File System
4. Disk Management
5. CPU scheduling
6. ~~Job~~ synchronisation & communication mechanism

7. Deadlock handling mechanism.

2) Desktop systems:

Personal Computers (PC's)

3) Parallel systems :-

→ multiprocessor sys. with more than one CPU in close communication

Tightly coupled system:-

processors share memory & a clock.
Communication usually takes place through the shared memory

1) SMP (Symmetric multip P)
2) ASMP (Asymmetric ...)

Advantages:

1. Increased throughput.
2. Economy of scale
3. Increased reliability
4. System fault tolerant

Symm MP	ASMP
1. all 'p' can execute jobs	

9/1/14

Operating Systems

→ Why OS?

→ What OS?

→ Functions of OS?

→ concurrency

→ V.M. (Virtual memory)

→ An. operating systems is a program that acts as an intermidiary / interface between the user of a computer and the computer hardware

→ It is a s/w that provides an elaborate illusion to applications.

Services provided by OS

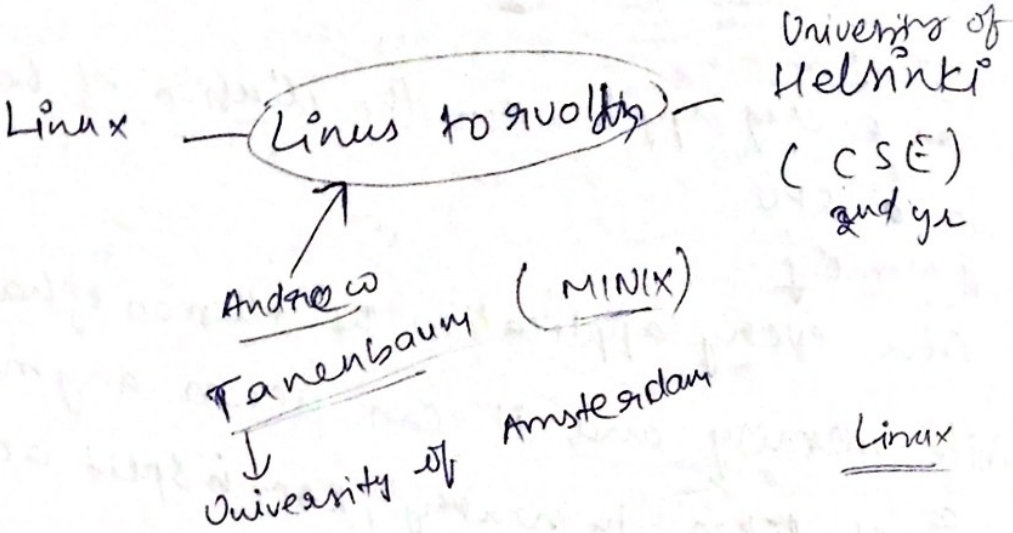
1. facilitates program creation execution
2.
3. access to i/o & files
4. System access

Important functions of O.S

1. Concurrency:
   Gives every application the illusion of having its own CPU.
2. Virtual Memory:-
   Give every applications the illusion of having infinite memory and, it can access any memory address it likes. In reality, RAM is split across multiple applications ~~Important functions of OS~~
3. multiprocessor support
   OS responsible for allocating of CPU to the processor
   OS [C.P.U] [C.P.U] [C.P.U]
4. File System - windows - NTFS (network F.S)
   ↳ mechanism to arrange files in some order
   Linux / Unix - hierarchy F.S
   * Real disks have sector based access model
   → User app. see flat files arranged in a hierarchical namespace

   Unix
   $ pwd

5. Security & protection:
) prevent multiple apps from interfacing with each other and with with normal system ope.



"GNU" – Richard Stallmann

Important feature of O.S :-

1. Abstraction / Abstrad :

Hides details of different hardware configuration

Ex: int, stack, list etc

2. Arbitration :-

manages access to shared hardware resources

OS Types :

.dll → dynamic linked library

1) no. o.s. – only libraries

2) Simple batch systems
(mid 1950s – mid 1960s)

– permanent resident O.S in primary memory

→ loaded a single job from card reader, ran it loaded next job

Cons:

1. lack of interaction b/w user & job
2. C.P.U is often idle

16/1/13

3. Multi-Programming System:

idea: multiple jobs reside in main memory

Pros:

CPU, memory, I/O utilised effectively

4. Multi Tasking / Time sharing:-

→ multiple users simultaneously access the system through terminals.

(Linked I/O)

→ processor time is shared among multiple users.

5. Distributed system / Loosely Coupled:

→ resources ~~the~~ not shared only jobs gets distributed

→ distribute computation among several processors

Real time - precision - accuracy

Q. which is the very first O.S introduced the concept of distributed O.S

(or which windows O.S introduced the concept of distributed O.S. → windows 3.1x

Spooling:

Simultaneous peripheral operation Online (SPOOLING)

Q. Diff b/w Interpreter & compiler.

PROCESS MANAGEMENT

#include <stdio.h>

#define TEST 0;

A macro is a shorthand for longer constraints

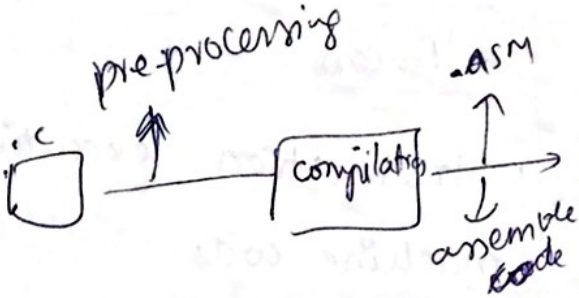
Preprocessing is done before the compilation

Pre-Processor:-

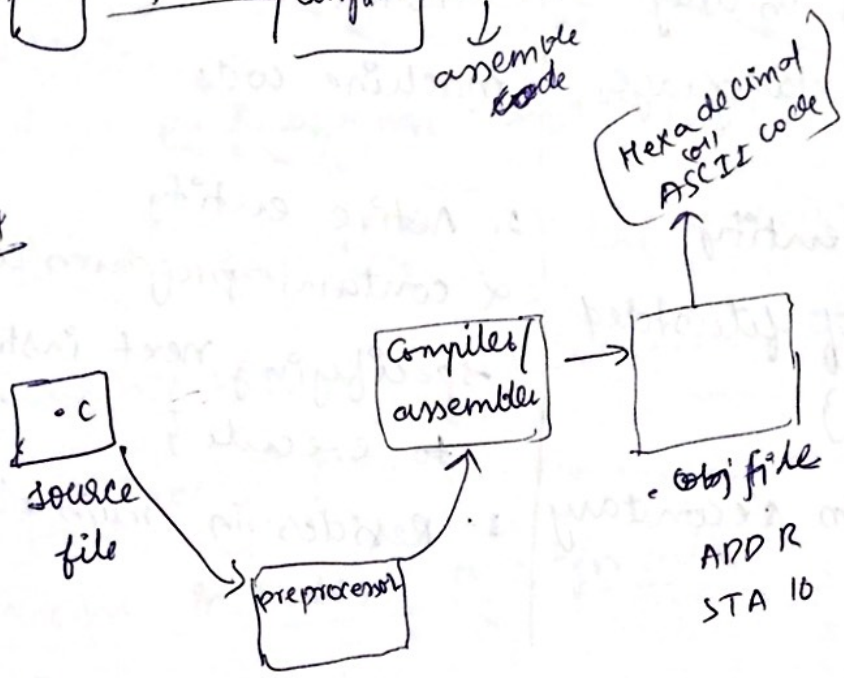
3 operations

1) replaces header files 2) removes comment lines

4) Replaces macro values



18/1/14



File → command → utility pgm
↳ shows the details of the file

$ file os.o

os.o : ELF 32-bit LSB relocatable, Intel 80386,
↓ machine architecture
version 1 (SYSV), not stripped
↳ both executable & object file format

ELF - executable and linking format

COFF - common object file format
↳ win → object file format for windows

PE - Portable executable
↓ executable file format for windows

a.out → executable file

Program	Process
1. Instructions in any programming language	1. Instruction execution machine code
2. Passive entity { content of file stored on disk }	2. Active entity & contains program counter specifying next instruction to execute }
3 Resides in secondary storage	3. Resides in main memory

UNIT-2

PROCESS MANAGEMENT

Process:-

1) A program under (or) in execution

2) The collection of data structures that fully describes how far the execution of the program has processed

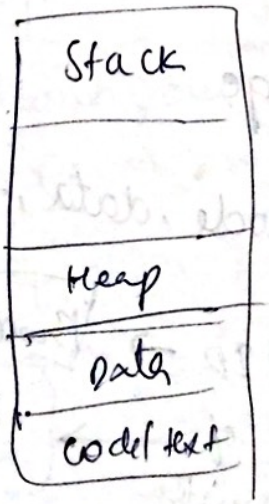
process includes 4 segments

1) CODE/TEXT :- Holds program.

2. DATA :- Holds program variables

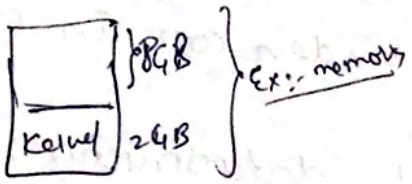
3 Heap :→ dynamic memory allocation.
: Holds intermediate computation data generated during run time.

4. STACK : Holds
1) Local Variables
2. Temporaries & Procedure (functions) calls
3. Return addresses.



* Task Manager Analysis

Kernel memory - protected memory



Kernel controls whole allotment of memory, etc

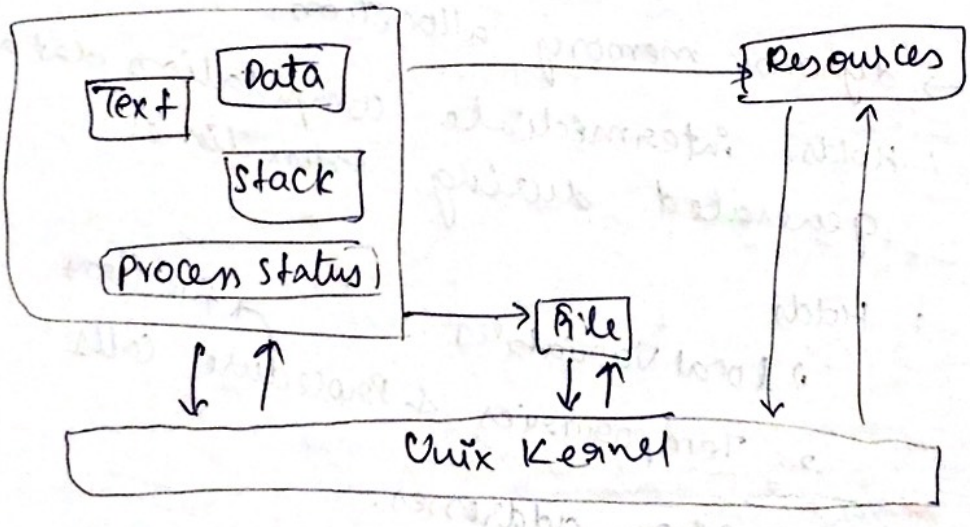
maybe O.S resides in Kernel

* Kernel cannot be communicated directly

Unix - file descriptor

window - handles.

Unix - Process



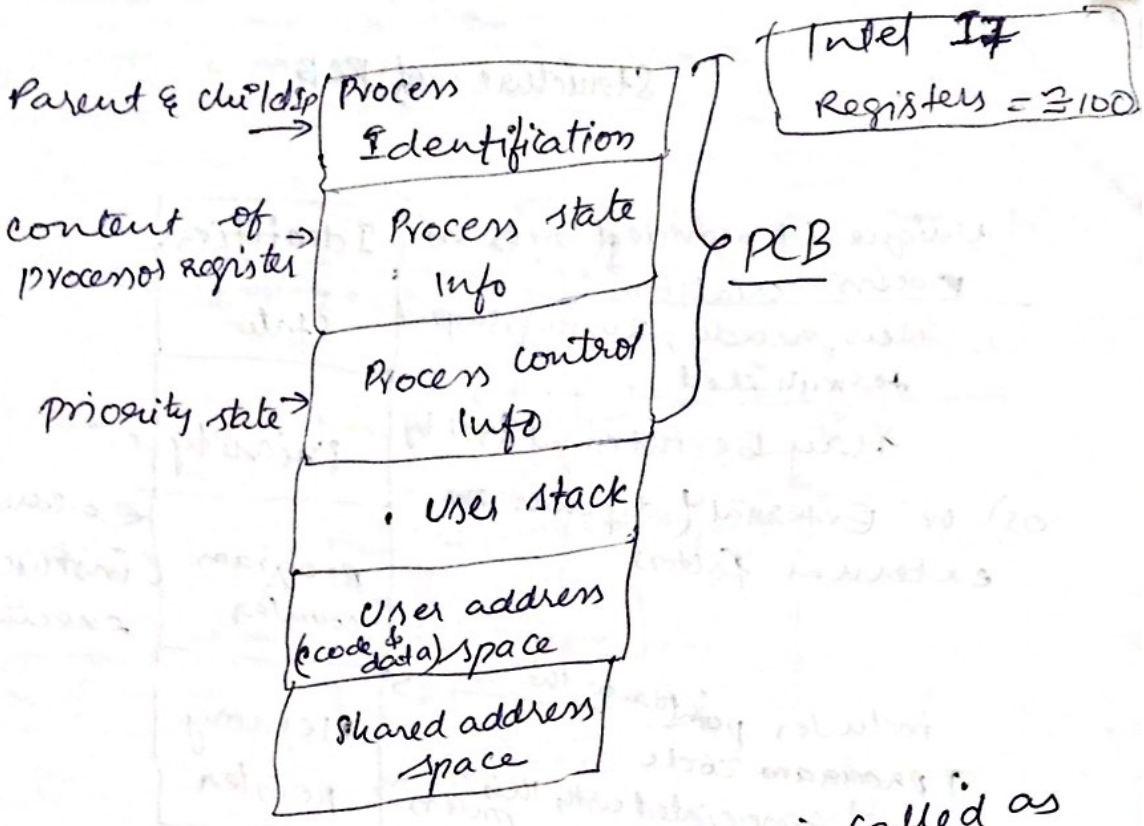
20/1/14

Process Image

* collection of code, data, stack & attributes

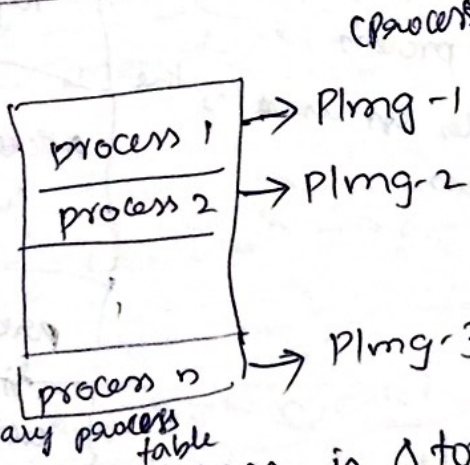
Parent & child ID → Process identification

content of processor register → process state information



* The collection of attributes is called as process control block (PCB) (or) Task control block (TCB)

primary process table



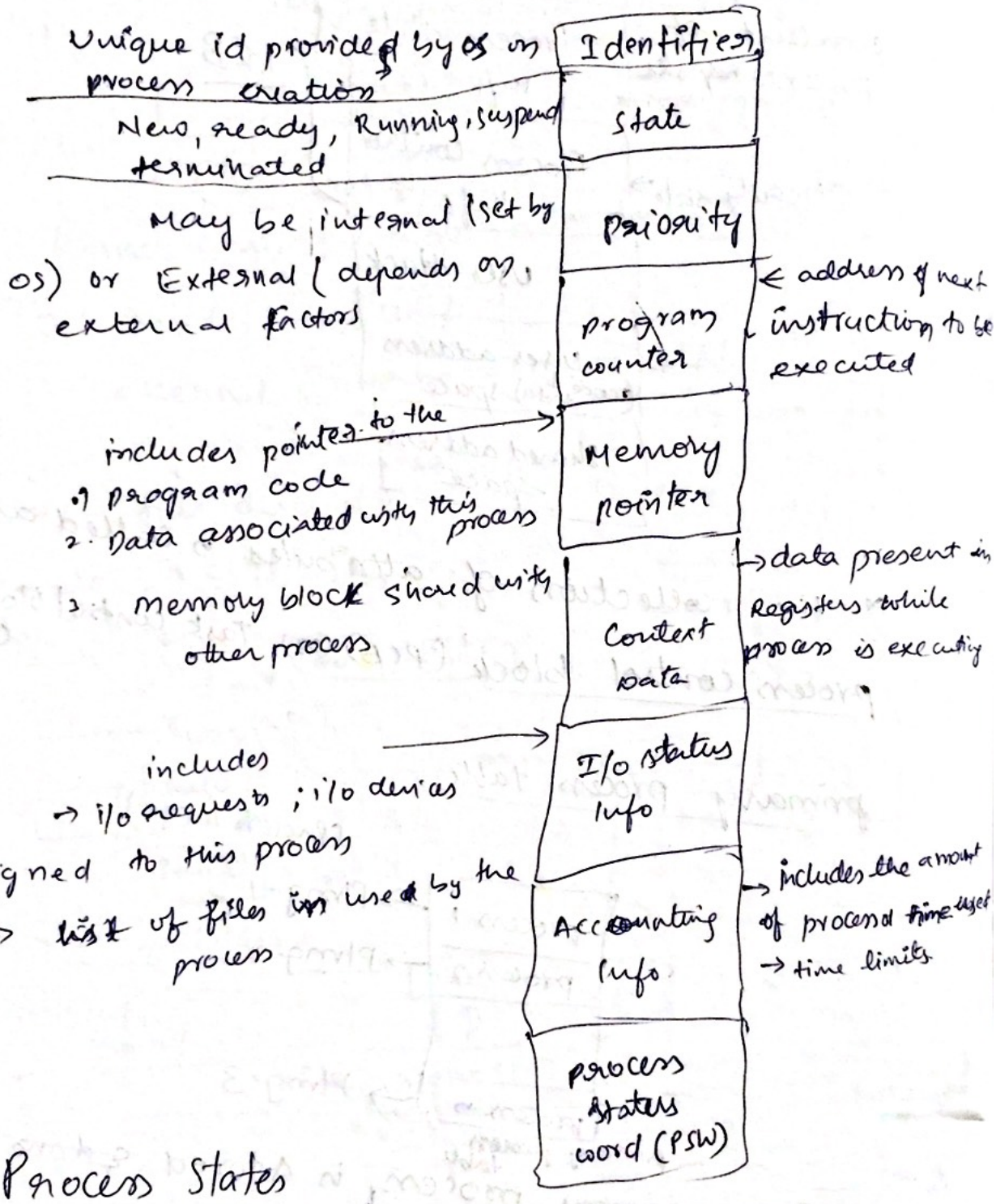
This primary process table is stored on the disk as contigous block. this block may be image

→ Fixed size (pages) (or) → Variable size (segments)

* The address of each process Image is stored on primary process table

* To maintain the process the Image must be brought into main memory

Structure of PCB



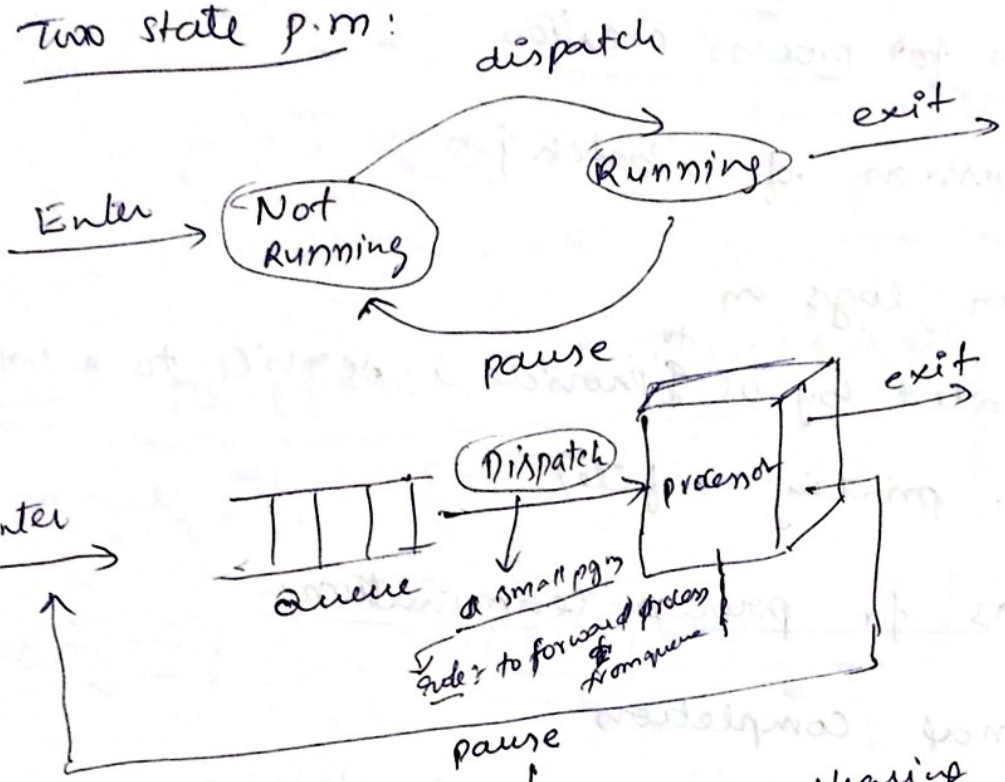
Process States

→ Two state process model

→ Three " "

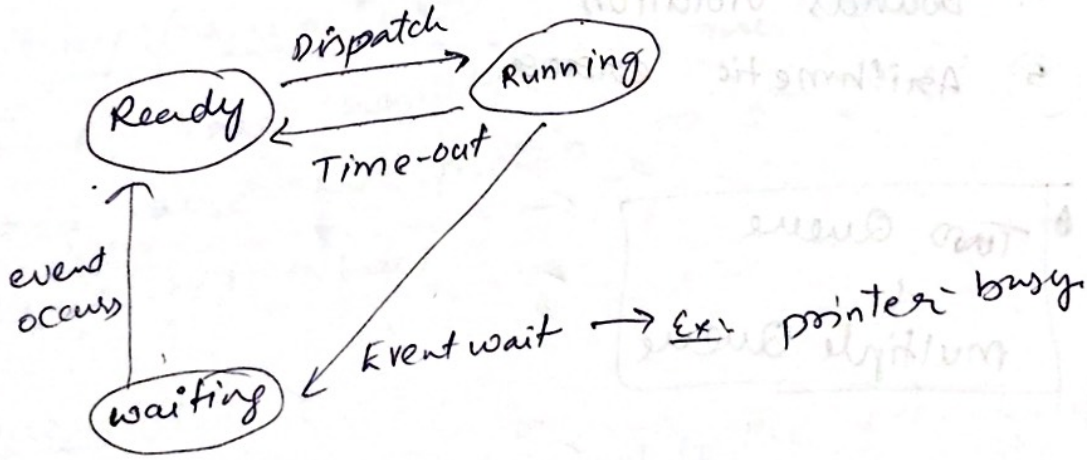
→ Five " "

Two state p.m:



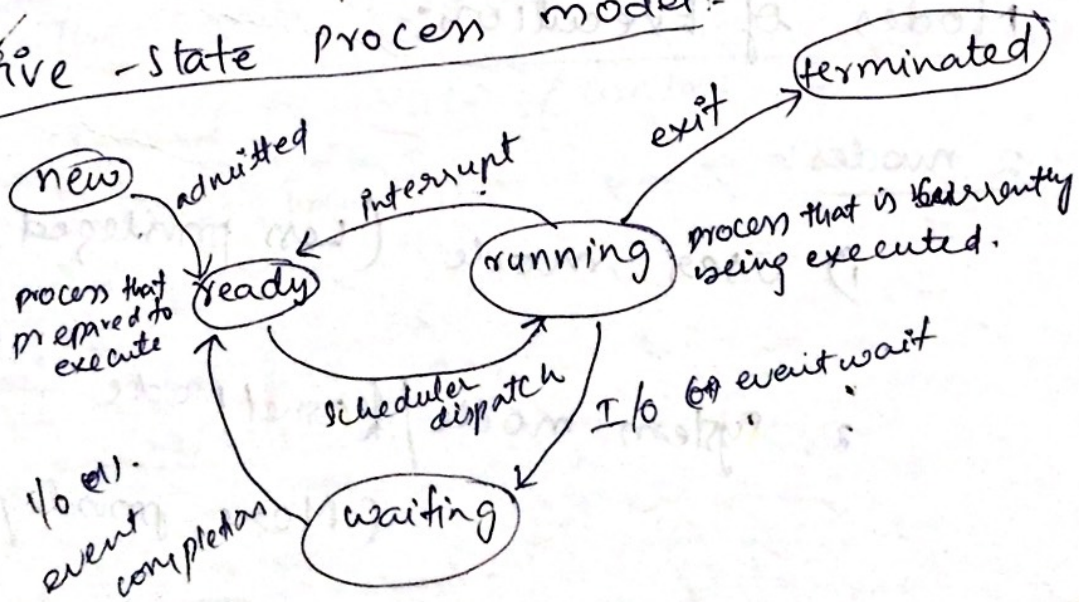
↳ ex: time sharing
printer - busy

→ Three state p.m:



22/1/14

→ Five - state process model:-



Reason for process creation

1) submission of a batch job
2. User logs on
3. Created by OS to provide a service to a user (eg: printing a file).

Reasons for process Termination :-

1. Normal completion
2. Time limit Exceeded
3. memory unavailable
4. Bounds violation
5. Arithmetic error.

Two Queue & multiple Queue

Modes of Executions :-

2 modes :-

1) User mode (Less privileged mode)
2. System mode / Kernel mode (More privileged mode)

Why?:

To protect os programs from interface by user pgms

How? PSW (process status word) indicates the mode of execution

command : cat/proc/stat

Process Switching:

When to switch a process:

1. Trap (type of signal) :-

An error resulted from the last instruction (it may cause the process to be moved to terminated state).

2. Interrupt:-

The cause is external to the execution of the current instruct (control is transfered to interrupt handler)

priority:

19/20 - low pr

23/01/14

Context switching: → The act of swapping a process state on or off the CPU is a context switch

→ when CPU switches to another process, the system must save the ~~state~~ state of the old process and load the saved data for the new process.

→ Context of a process represented in the PCB

- context switch time is overhead, the system does the useful work while switching

Steps in context switch:-

→ Save context of processor including program counter and other registers.

→ Update the PCB of the running process with its new state & other associate info

→ move PCB for appropriate queue - ready, blocked

→ select another process for execution

→ Update PCB of the selected process

- Restore CPU context from that of the selected process.

Revision:-

1) pgm
2) process
3) pgm vs process
4) Process Image
5) PCB
6) process creation
7) process termination
8) process control modes
9) process switching
10) context switching

27/1/18

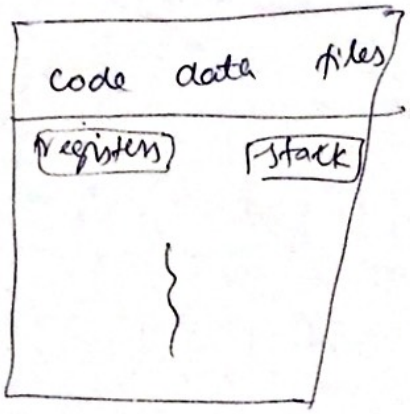
Threads:

A process is divided into number of light weight processes. Each lightweighted process is called as Thread.

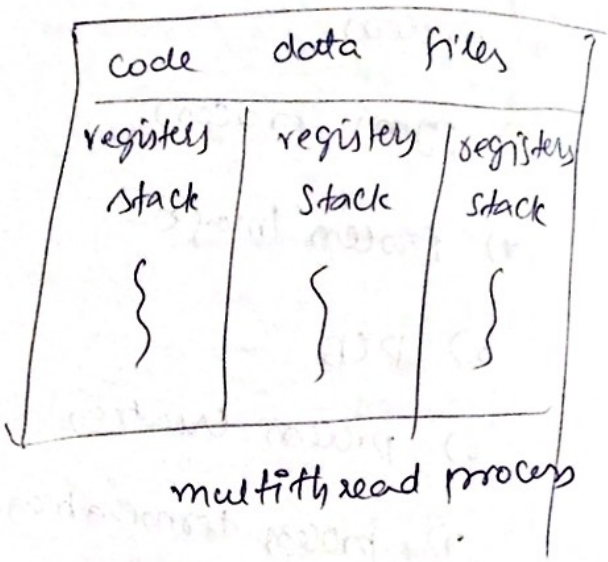
Each thread has its own

1. Program counter
2. Registers (holds current working variables)
3. stack (contains execution history)
4. Thread state.

Q. Diff b/w child process & a thread.



single threaded process.



multithread process

Ex. Internet Explorer, Microsoft word.

Properties:-

1. Threads can share
   1. address space (pgm)
      ↳ place where process is stored.
      (location)
   2. Opened files and other resources
   3. C.P.U
      (but only one thread is active at a time)

→ Threading technology in Intel i7 - Hyper Threading

→ 'Threads' can 'create' 'child Threads'

II'ly

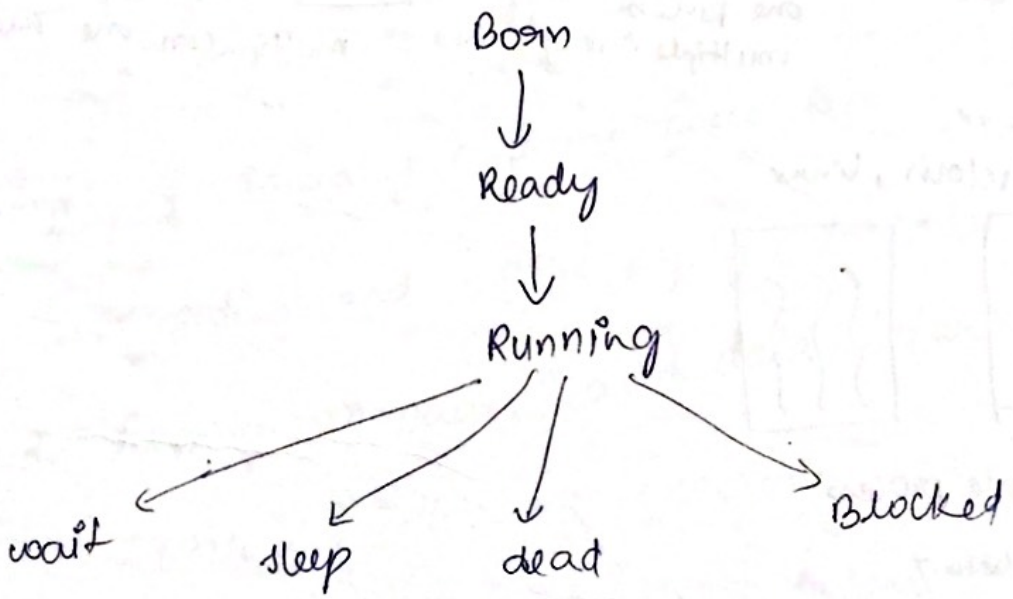
'process' can create 'child processes'

* Threads are not independent of one another

Benifits:-

* Less time to create and terminate a thread than a process (because we do not need another address space).
* Less time to switch b/w two threads than b/w ~~two~~ processes
* Inter-thread communication and synchronisation is very fast.

Thread Life Cycle



Thread Types

Based on Implementation

↓

1. User level thread
2. ⇒ Kernel level thread

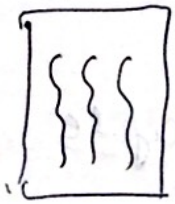
Based on functionality

1. one process one thread
2. one process multiple thread
3. multi process one thread
4. multi process multi thread.

MS DOS ↑ Disk operating system

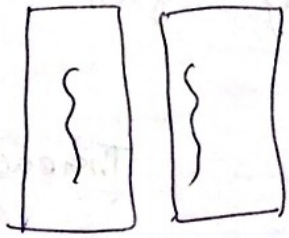
one process one thread

JRE ↑ Java Runtime environment



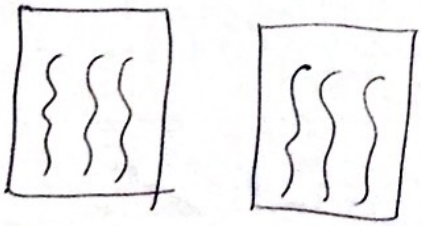
one process multiple thread

unix



multiprocess one thread

windows ↑ 2000, xp / solaris, linux



multiple process multi thread

29/1/14

User level thread:

1. Threads are loaded entirely in User space, Kernel is not aware about them
2. Each process maintain its own private thread table which contains details about PC, STACK, Register state
3. Thread management done by user-level threads library

Three primary thread libraries

1. POSIX Pthreads
2. Win32 threads
3. Java threads

} each one is collection of libraries

Thread library contains code for

1. creating and destroying threads
2. Passing messages and data between threads
3. Scheduling thread execution
4. Saving & restoring thread contents

30/1/18

system call - command
communicate shell
Kernel

Kernel level threads:

1. the kernel does total work of thread management
2. No thread table in each process
3. Kernel has thread table that keeps track of
   - all threads in system

Process vs threads

1. Can't share the same memory area (address space)	1. can share memory & file
2. Take more time to create a process	2. takes less time to creat
3. more time to complete the execution & termination	3. less time to terminate
4. Takes more time to switch b/w process	4. Takes less time to switch b/w two threads
5. Communication b/w two process are difficult to implement.	5. Easy to implement
6 Sys Call req for communication	6. Not req
7. Processes are loosely coupled	7 tightly coupled
8. Require more resource to execute	8. less resource to execute
9. Not suitable for parallel activities	9. Suitable for ||l activities

Why scheduling?

1. efficient utilisation of processor
2. minimise the wastage of resource

CPU Scheduling

1. Long term Scheduler (Job Scheduler): → selects job from pool of jobs and loads into main memory (ready q)
2. Short term Scheduler (CPU Scheduler): → job from ready q into processor
3. Medium Term Scheduler → keeps req. process in memory & removes the process that are not needed

method of selecting a process from ready q depends on CPU scheduling algorithm

Dispatcher:

- It is a module which connects the CPU to the process selected by the short term scheduler

Main function:

- Switching the CPU from one process to another

Dispatch latency:

→ The time takes by the dispatcher to stop one process and start another running

Scheduling criteria:

1. Throughput:
   - Number of jobs completed by the CPU within a time period

2. Turn around time:

The time interval b/w the submission of the process & time of completion.

3. Waiting Time:

- Sum of periods spent waiting by a process in the ~~waiting~~ Ready queue

4. Response Time:

The time duration b/w the submission job & first response

5. C.P.U utilisation:

percentage of time the processor is busy

3/2/14:

Policies followed by scheduling algorithms:

1. Pre-emptive:-

Once the CPU is assigned to a process, the CPU can release the processes even in the middle of the execution

2. Non-Preemptive:-

Once the CPU assigned to a process, the processor do not release, until the completion of that process.

Types of
Scheduling Algorithms:-

1. First come First Serve (FCFS)
2. Shortest job first (SJF)
3. Shortest Remaining time First (SRTF)
4. priority scheduling
5. Round Robin Scheduling
6. Highest Response ratio next (HRRN)
7. Multi level feedback queue

Q which scheduling algo. is used in win 7/8 & Linux?
Q. In Linux - Ubuntu - Cfq (Completely fair queuing) - 2003

→ First Come First Serve :- (FCFS):-

1. Simplest
2. process requests the CPU first is given the CPU
3. Implementation using of FIFO queue
4. Once a process is given the CPU, it keeps till the completion

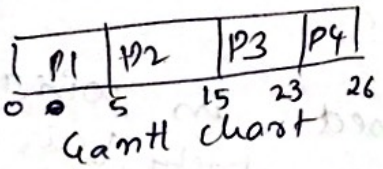
Non preemptive

Burst time - C.P.U. Time - total time required for completion of a process

Gantt chart - ~~used~~ a chart used to represent in which order the process will execute

Ex1:

Arrival time	process	CPU time
0	P1	5
0	P2	10
0	P3	8
0	P4	3



Gantt chart

Average waiting time:-

waiting time : Starting time - Arrival time

P1 = 0-0 = 0

P2 = 5-0 = 5

P3 = 15-0 = 15

P4 = 23-0 = 23

Ave. wait time = $(0+5+15+23)/4$ = 10.75 ms

Turn around time:-

T.A.T = (Finished time - Arrival time)

P1 = 5 - 0 = 5

P2 = 15 - 0 = 15

P3 = 23 - 0 = 23

P4 = 26 - 0 = 26

A.T.A.T = $\frac{(5+15+23+26)}{4}$

= 69/4 = 17.2 ms

Average Response Time:-

A.R.T = First response - Arrival time

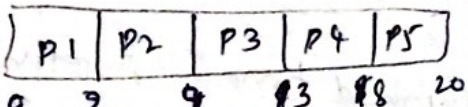
A.R.T = $\frac{(0-0)+(5-0)+(23-0)+(15-0)}{4}$ = 43/4 = 10.75 ms

Cons:-

1. High turn around & waiting time
2. Low rate of CPU utilisation
3. short job have to wait for long time

Ex 2:

Arrival time	process	CPU time (or) Burst time
0	P1	3
2	P2	6
4	P3	4
6	P4	5
8	P5	2



Gantt chart

avg. wait. time:

P1 = 0-0 = 0
P2 = 3-2 = 1
P3 = 9-4 = 5
P4 = 13-6 = 7
P5 = 18-8 = 10

$\frac{23}{5} = 4.6$

Turn-around time

P1 = 3-0 = 3
P2 = 9-2 = 7
P3 = 13-4 = 9
P4 = 18-6 = 12
P5 = 20-8 = 12

$\frac{43}{5} = 8.6$

Response time

0
1
5
7
10

$\frac{23}{5} =$

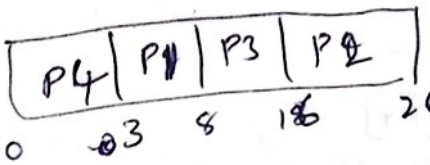
Shortest job first scheduling (SJF):-

- Non-Pre-emptive

1.

Ex1:

Arrival time	Process	CPU time(or) Burst time
0	P1	5
0	P2	10
0	P3	8
0	P4	3
0		



Gantt chart

Waiting time:

$$\text{wait. time} = \frac{((8-0)+(3-0)+(16-0)+(8-0))}{4} = \frac{27}{4}$$

$$\text{Resp. time} = ((0-0)+(3-0)+(16-0)+(8-0))/4 = 27/4$$

$$\text{T.O.T} = [(8-0)+(26-0)+(16-0)+(3-0)]/4 = 53/4$$

A.T.A.T = 13.25 ms

5/2/14

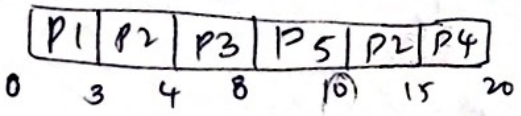
Shortest Remaining Time First (SRTF):-

1. Policy - preemptive scheduling
2. Scheduler compare the remaining time of executing process & new process

Ex:-

Arrival time	process	CPU time or burst time
0	P1	3
2	P2	6
4	P3	4
6	P4	5
8	P5	2

Gantt chart:



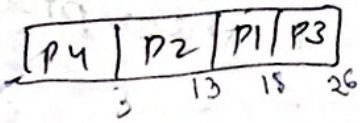
Priority Scheduling:-

1. Priority is associated with each process
2. Scheduler always picks up the highest priority process for execution from ready queue
3. priority scheduling can be either
   1. pre-emptive
   2. Non-pre-emptive

Ex:

priority	process	CPU time (burst time)
3	P1	5
2	P2	10
4	P3	8
1	P4	3

Gantt chart:



Round Robin Scheduling

1. Designed specially for time sharing system
2. CPU time is divided into time slices

(or) time quantum, each process is allocated a small time slice

3. time quantum is generally 10 to 100 ms

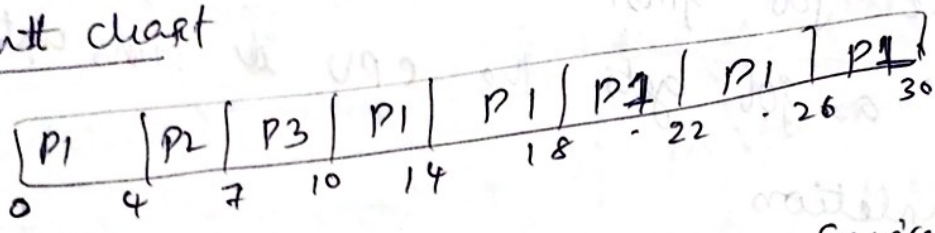
4. pre-emptive

Ex1:

process	CPU time (or) burst time
P1	24
P2	3
P3	3

Time quantum = 4ms

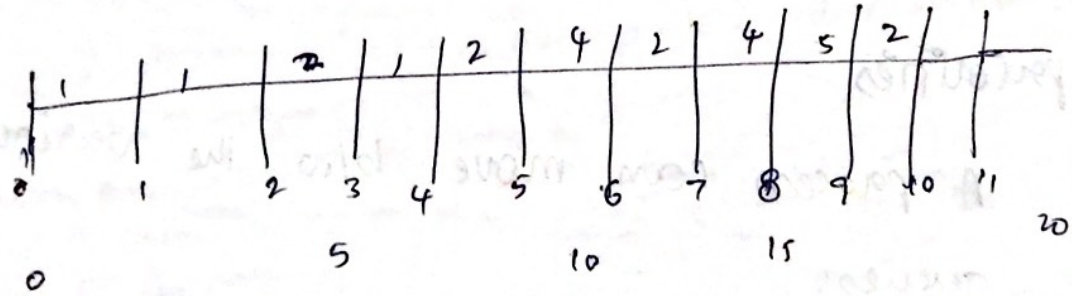
Gantt chart



Ex2:

process	Arrival time	Service time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Time quantum = 1ms



Highest Response Ratio Next (HRRN) :-

1. Process having the highest response ratio execute job first
2. once a job gets the CPU it runs upto completion

Response ratio = $(w + s)/s$

w = time spent waiting for the processor

s = total service time required by the process

Multilevel feedback Queue:-

1. pre-emptive scheduling with dynamic priorities
2. A process can move b/w the various queues

3 Multilevel-feedback queue scheduler defined by the following parameters

* Ready queue is divided into 'n' queues each queue has its own scheduling algorithm

Algorithm Comparison:-

1. which one is best?

The answer depends on?

1) on the system workload (extremely variable)
2. hardware support for the dispatcher
3. relative weighting of performance criteria (response time, CPU utilisation, throughput)

HRRN Example:

P	A.T	S.T
P1	0	3
P2	2	6
P3	4	4
P4	6	5

At time, t=0, process P1 arrives & is scheduled for 3 cpu cycles. At t=3 only process P2 has arrived, hence no need to calculate RR & schedule P2. The process P2 needs 6 CPU cycles i.e, P2 finishes at t=9

Now at t=9, both P3 & P4 have arrived so you have calculate the RR of both P3 & P4

wait. time of P3 = 9-4 = 5

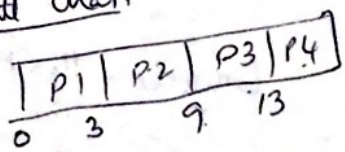
" " P4 = 9-6 = 3

RR(P3) = (5+4)/4 = 9/4 = 2.25

RR(P4) = (3+5)/5 = 8/5 = 1.6

Thus P3 is scheduled next & P4 next

Gantt Chart



12/2/14

Process Synchronisation

Co-operating Processes:

1. processes within a system may be independent (or) co-operating.

2. Independent process cannot affect (or) be effected by execution of other process but in co-ope. it is converse.

1) Inter-leaving: one process is permitted (multiprogramming, one processor)

2) Inter-leaving & Overlapping) → more than one proces can be permitted (multiprogramming, two processors)

Race Condition:-

- Several processes access & manipulate the same data concurrently &
the outcome of the execution depends o the particular order in which the access takes pl

Solution: Ensure that only one process at a time can manipulating the shared variable

2. Require some form of Synchronisation

Critical - Section Problem

- Each process has a code segment called critical section(CS). In which the shared data is accessed.

- Before entering C.S. at first request permission

→ The section of code implementing request is Entry section (E.S).

→ The C.S. might be followed by a leave / Remainder section (RS)

Exit Section (LS)

13/2/14

General structure of a Typical process

```
do {
    entry section
        critical section
    exit section
        Remainder section
} while (True).
```

Solutions to C.S Problems:

- must satisfy 3 requirements for a correct solution.

1. Mutual Exclusion
2. Progress
3. Bounded waiting.

M.E:- If process , Pi is executing in its critical section then no other processes can be executing in their critical sections.

Progress:-

If no process is executing in its C.S & there exist some processes that wish to enter their C.S then

1. processes that are not executing in the remainder section can participate in the decision of which will enter its critical section next.
2. Selection of the process that will enter the C.S next cannot be postponed indefinitely (dead lock)
3. The decision ~~should~~ about who enter C.S should be taken in ~~if~~ finite time.

Bounded waiting:- A bound must exist on the number of times that other processes are allowed to enter their C.S. after a process has made a request to enter its C.S & before that request is granted.

Initial attempts to solve problem:

- Shared Variables:

int turn;

initially turn = 0 (or) 1

Structure of process P_i:-

turn = i (P_i can enter its critical section)

repeat

while (turn ≠ i) do nothing

critical section;

turn = j;

Remainder section;

Until false;

Algorithm 2 :-

Shared Variables:-

boolean flag[2]

initially flag[0] = flag[1] = false;

flag[i] = true ⇒ P_i ready to enter its C.S

Structure of process P_i:-

```
repeat
    flag[i] = true;
while (flag[j]) do nothing
    critical section :
    flag[i] = false
    remainder section;
until false;
```

17/2/14

Algorithm3 (Peterson Algorithm):-

- combined shared variables of algorithms 1 & 2

```
do {
    flag[i] = true;
    turn = j;
    while ( flag [j] and turn = j) do nothing;
    critical section ;
    flag [ i] = false ;
    remainder section ;
} while (1)
```

19/2/14

Bakery Algorithm

→ used mainly to provide a soln for problem of synchronisation when 'n' process are competing for a resource

21/2/14

Semaphore :- (To provide 3 main things) → mutual exclusion, progress, bound waiting

It is a variable which is used to protect shared variable. It may be name.

variable : non negative value (initial)

mutex = 1 (initial)
(semaphore)

wait(), signal() through only semaphore variable can be accessed (all pro

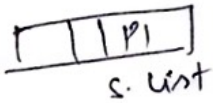
wait():-
decrement semaphore variable (s--);

signal():
Increment semaphore variable (s++);

```
do {
  wait (mutex);
   C.S
   signal (mutex);
   remainder section;
 }
wait (mutex)
   mutex . value --;
if (mutex . value < 0)
{  add this process to S.L; block ; }
```

semaphore list

signal (mutex)



mutex. value ++;

if (mutex. value <=0)

remove a process from

→ Interrupt is not permitted while executing wait() & signal().

3/3/14

Classical problems of Synchronisation:

1. Bounded Buffer Problem / Producer - Consumer Problem
2. Dining - Philosphers Problem
3. Readers - Writers Problem

Bounded Buffer Problem:-

Shared data:-

semaphore full, empty, mutex;
(full: stores how many buffers are full; mutex: used to protect cons & prod.)

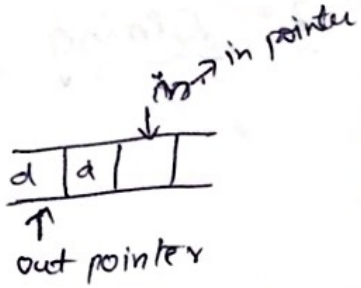
Initially:-

full = 0, empty = n, mutex = 1;

1. Buffer size is 'n'
2. Mutex provides
3. Consumer waits on full
4. producer waits on empty

→

```
do
{
    ...
    produce an item in next p
    ...
    wait (empty);
    wait (mutex);
    ...
    add next p to buffer
    ...
    signal (mutex);
    signal (full);
} while (1);
```



≡ queue

if (mutex == 0)
{ ⇒ someone in c.s. }
else {
no one in c.s }

wait (mutex) ⇒ mutex = 0

→ Consumer Process

```
do {
    wait (full)
    wait (mutex);
    ... remove buffer to nextp;
    signal (full);
    signal (mutex);
} while (1).
```

2. Dining - Philosphers Problem:-

Shared data:-

```
semaphore chopstick [5];
(initially all values are 1)
```

Philosopher i:-

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

        eat

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

} while (1);
```

→ The soln is not "dead lock" free

⇒ All philosphers pick up left chopsticks !!

Solutions:

1. allow at most 4 philosphers

2.

3.

3. Readers_writers Problem:-

Shared data:-

```
var mutex, wrt : semaphore (=1);
    readcount  : Integer (=0);
```

Writers Process:

```
wait(wrt);
---
writing is perfolmed;
---
signal(wrt);
```

Reader Process:

```
wait(mutex);
    readcount := readcount + 1;
    if readcount = 1 then wait(wrt);
    signal(mutex);
    ....
    Reading is performed
    ....
wait(mutex);
    readcount :=
```

→ checks whether the reader(s) are present

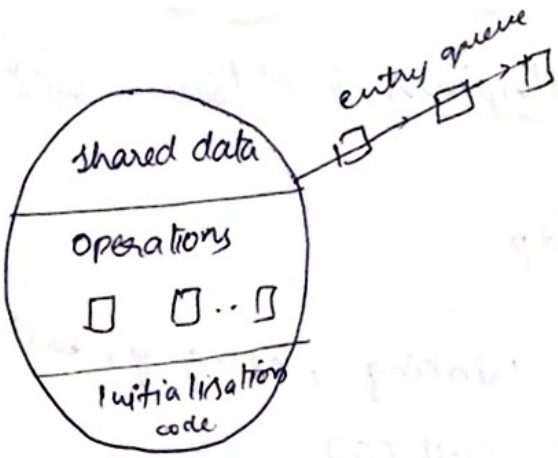
5/3/18

Monitors

It is a kind of synchronisation tool where common data is protected among 'n' concurrent users

→ class - no synch tool but in monitor it is present

→ A monitor defines a (lock) and zero (or) more (condition variables) for managing concurrent access to shared data.



Schematic view of monitor

lock - monitors can controls entry into the monitor

condⁿ variables: - inside monitor

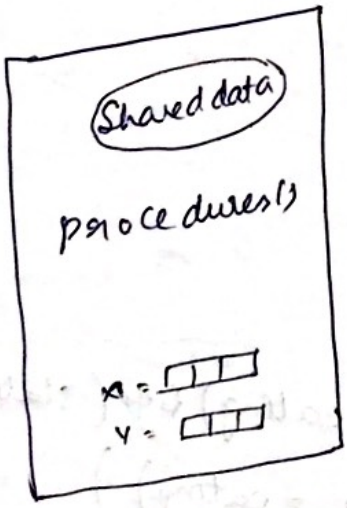
↳ operations

wait (Lock lock);

signal();

Broadcast();

6/3/14



wait → sleep & unlock lock

signal → wake up

Dining Philosphers Problem with Monitor

```
monitor dp
{
    enum { thinking, hungry, eating } state[5];
    condition self[5];

    void pickup(int i);
    void putdown(int i);
    void test(int i);

void init()
{   for(int i=0; i<5; i++)
        state[i] = thinking;
    }
void pickup(int i) {
    state[i] = hungry
    test[i];
    if (state[i] != eating)
        self[i].wait();
}
void test(int i) {
if ((state[i+4 mod 5] != eating) && (state[i] = hungry
    && (state[i+1 mod 5] != eating))

{
}
```

void putdown (int i)

{

}

Producer - Consumer Problem with Monitor:-

monitor producer-consumer {

Deadlocks

Indefinite waiting for some thing ⇒ deadlock

Deadlock characterisation

Four condn's:

1. mutual exclusion
2. Hold & wait
3. No-Pre-emption
4. Circular wait

Methods for handling deadlocks

1) Deadlock prevention
2) Deadlock Avoidance

→ ensure atleast one of the necessary conditions cannot hold

10/3/14

2. Hold & wait:

1st protocol:

All the requested resources

3. No - Pre-emption :-

4. Circular Wait:-

→ cannot hold

protocol:

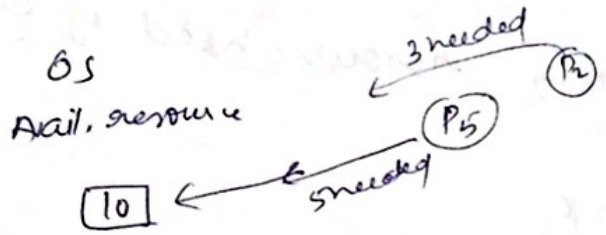
→ Assign unique integer to resource type

→ In increasing order only process requests will be done

12/3/14

Deadlock Avoidance:-

→ . keep always system in safe



checks before allocation of resource whether it will be safe or not to allocate resource

Banker's Algorithm:

purpose: OS will check the safety after allocation of resource in diff combinations.

data Structures:

Let n = no of process; m = no. of resource types

Available: Array Available = [4]

Number of available resources. If available [j] = k ; there are 'k' instances of resource type R_j available

Max:- maximum demand of each process.

If Max [i, j] = k, then process P_i may request at most 'k' instances of resource type R_j

Allocation:- no. of resource currently allocated

Allocation [i, j] = K

Need:-

remaining resource need of each proc

Need [i, j] = K.

$$\text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]$$

Ex:

1. 5 processes P0 through P4
2. 3 resource types A (10 inst), B (5 inst) & C (7

Snapshot at time (T0):-

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

Sequence: $< P_1, P_3, P_4, P_2, P_0 >$

Seq	Avail	Need	New = Avail - Need	Max	Max + new
P_1	3 3 2	1 2 2	2 1 0	3 2 2	5 3 2
P_3	5 3 2				
P_4					
P_2					
P_0					

Safety Algorithm :-

Finding out whether or not a system is in safe state

1. Work = Available
   Finish [i] = False for i = 1, 2, 3 ... n
2. Find an 'i' such that both,
   a) Finish [i] = false
   b) Need i ≤ work
   if not such 'i' exists ; go to step '4'.
3. work = work + allocation$_i$
   Finish [i] = true
   go to step 2.b
4. If finish [i] == true for all i, then the sys is in a safe state.

13/3/14

Resource Request: Algorithm :-

→

Deadlock Detection:

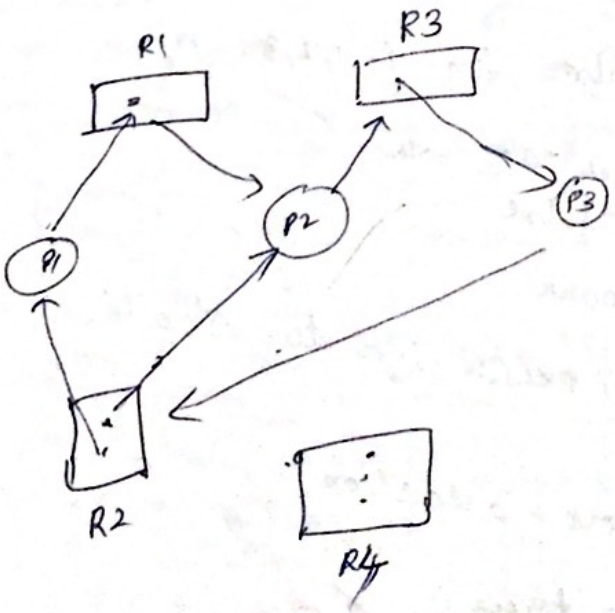
Allow system to enter deadlock state.

1. Detection algo
2. Recovery scheme

1. Wait for: (Single Instance):
2. Resource Allocation Graph (multiple instance)

Request edge : directed edge : $P_i \rightarrow R_j$

assignment edge : directed edge $R_j \leftarrow P_i$



Single Instance

Detection: ≅ Safety Algo

b. Datastructures

1. Available

2.

3. Request

~~19/3/19~~

19/3/14

Memory Management

memory → physical → collection of bytes (or) addresses

→ permanent add ≡ Name - always remains same

Physical add - ≡ Reg. no - changes

logical add (or) virtual add ⇒ generated by CPU

→ Virtual add space ∈ collection of virtual add

→ If CPU need to process anything virt. add is necessary

→ MMU (memory management unit) is used to convert phy add virtual add. It is within the CPU for better speed

20/8/19

Binding → conversion of inst format to machine code format
of instructions and data to memory

Add. binding of instructions and data to memory
add can happen at three diff stages

1. Compile Time :-
   absolute code - location fixed for pgm
2. Load Time :-
3. Run Time :

1. Dynamic loading :-
   1. Routine is not loaded until it is called
   2. Better memory space utilisation
2. Dynamic linking :-
3. Overlays :
   for execution, entire program and data must be in physical memory

24/3/14

Memory Management Unit (MMU)

- H/w device that maps virtual address to physical address

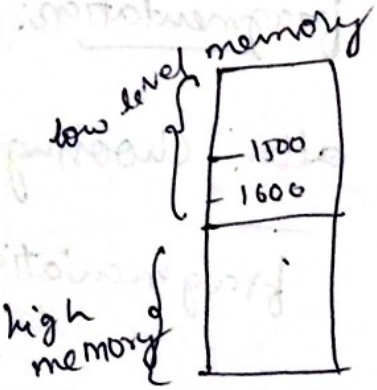
Relocation Reg: contains value of smallest physical add (of user pgm)

Limit Reg: contains range of logical add. Each logical add must be less than the limit reg

Various Partition Techniques:-

Single Partition:-

- Resident O.S, usually held in low mem
- User processes, held in high mem



→ Multiple Partition:-

dynamic storage allocation problem :

- How to satisfy a request of size, n from a list of free holes

First fit: Allocate the first hole that is big enough

Best fit: Allocate the smallest hole, that is big enough : must search entire list ; unless ordered by size

Worst fit:
Allocate the largest hole ; must also search the entire list.

fragmentation :- wastage of memory space while ~~after~~ choosing d.s. allocation prob is called fragmentation.

Internal fragmentation :-
1. allocated mem may be slightly larger than requested mem.
2. this size diff is memory internal to a partition, but not being used.

External fragmentation:

total mem. space exists to satisfy a request, but it is not contiguous

Defragmentation = compaction

shuffle memory contents to place all free memory together in one large block.

Disk defragmenter

Higgs-Boson

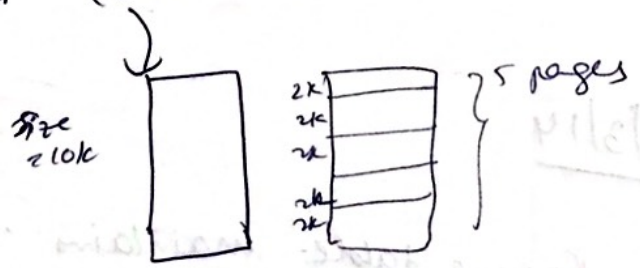
26/3/14

Paging → H/w

paging: → one type of memory management technique → efficient

→ Division of logical add space into a set of

CPU (300-400) for process 1



Address Translation Scheme:-

- add generated by CPU is divided into

→ page number (p) - used as an index into a page table which contains base add of each page in physical mem

Page Offset (d):- combined with base add to define the physical mem add that is sent to the memory unit.

→ The page size is defined by the hardware

→ page table (→ data structure) contains info about frames.

→ Mapping logical address to physical address

physical ad = (Frame no. * page size) + offset value

→ log add (page 0, offset 0) maps to phy add 20. (5 * 4 + 0) = 20

27/3/14

Frame table maintains details of frames which are currently free in main memory

Implementation of Page Table:-

each process - separate page table
↳ divided into pages

pagetable - main memory

- Page table base register (PTBR)

- Translation look aside buffers (TLBs)

- 32 TLB

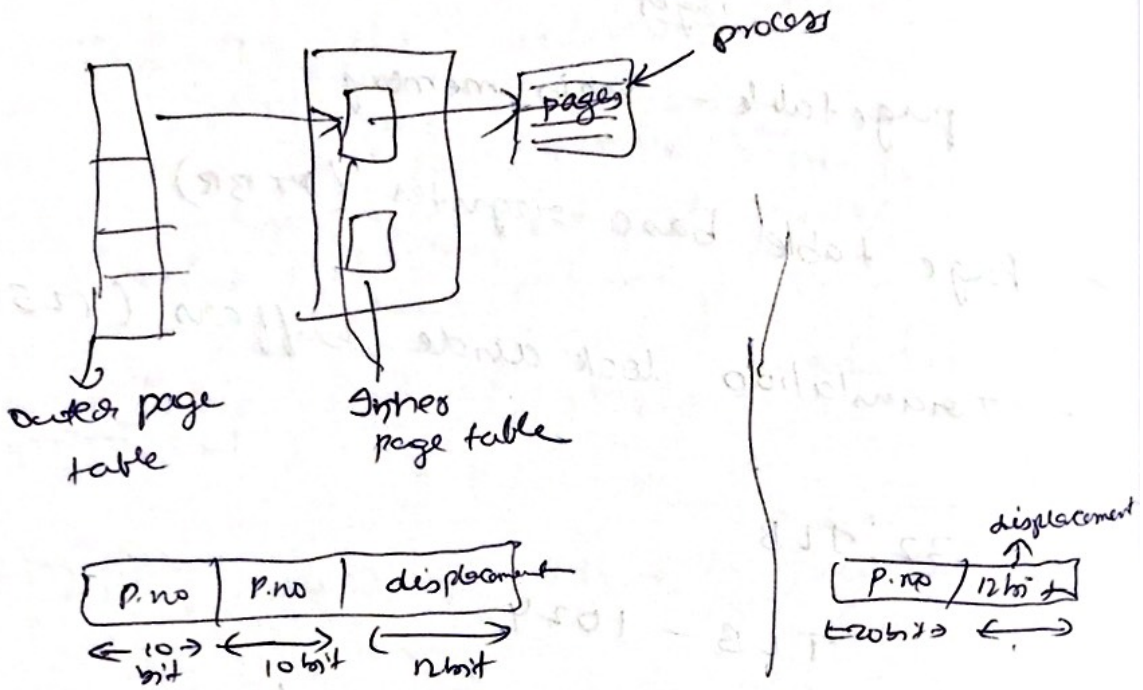
TLB - 1024

32 TLB : 32 x 1024

Memory Protection :-

2/4/14

Two level paging



Inverted Page Table:-

- a frame with variable i'e

3/4/14

Virtual memory → why ?
→ what ?

↳ technique used if size of logical address space is "large" but physical space is "less"

3 policies

- Fetch → when to get pages from disk to main memory
- Placement
- Replacement

→ i) pre-paging → limits no. of users
ii) Demand paging

Page fault:
If searching page is not available in main memory then page fault occurs

16/4/14

Virtual memory

3 policies

1. Fetch
2. placement (first, best, worst)
3. Replacement policy

— Replacement policy
  1. FIFO
  2. LRU
  3. Optimal Replacement

— Thrashing :-

A process is ~~stay~~ busy in swapping in & out of ~~the~~ pages

UNIT-4

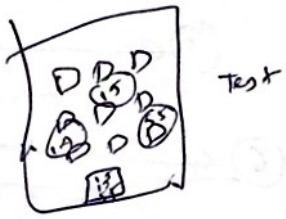
FILE

File - collection of attributes

→ where ? → secondary storage.

how memory allotment done ?

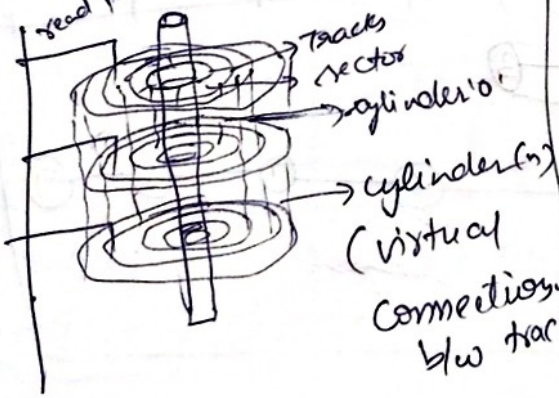
- contiguous allocation
- linked allocation
- indexed allocation →



17/4/14

Disk Scheduling

read / write head (or) plot



Processor Scheduling

FCFS

SJF

RR

SRTF

Disk Scheduling

1. FCFS - Request satisfies in sequences

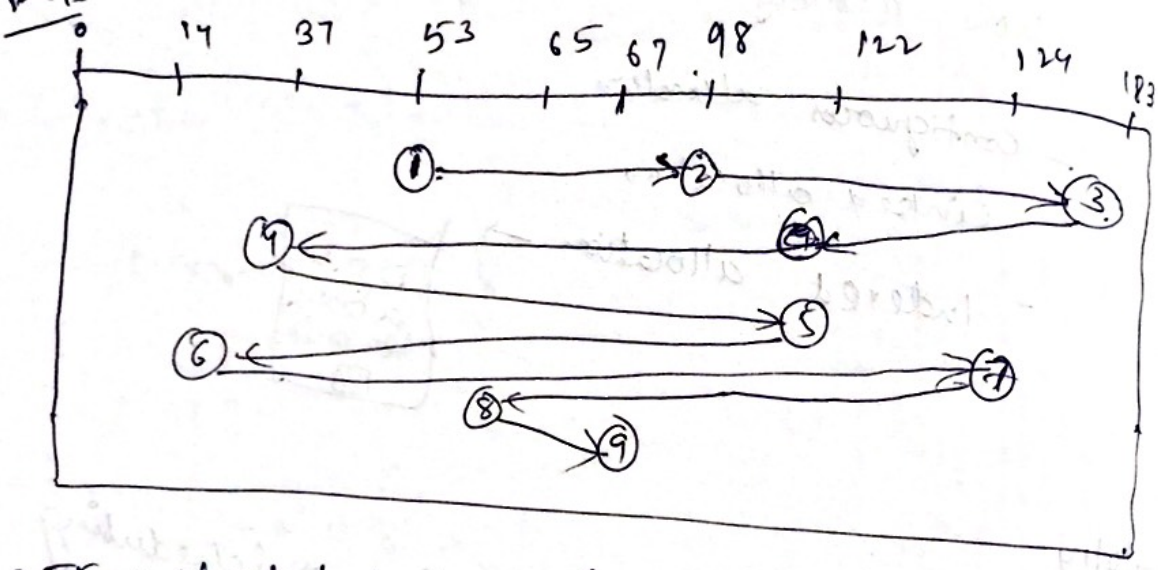
{ 56, 5, 95, 2 } → D, A, I, C

Request queue:-

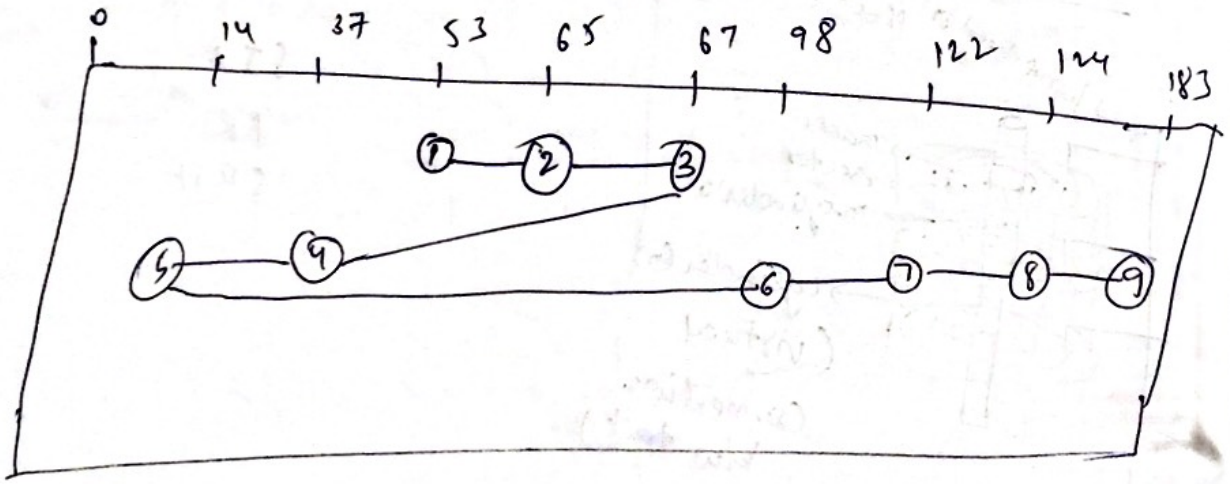
98, 183, 37, 122, 14, 124, 65, 67

Head position: 53

FCFS

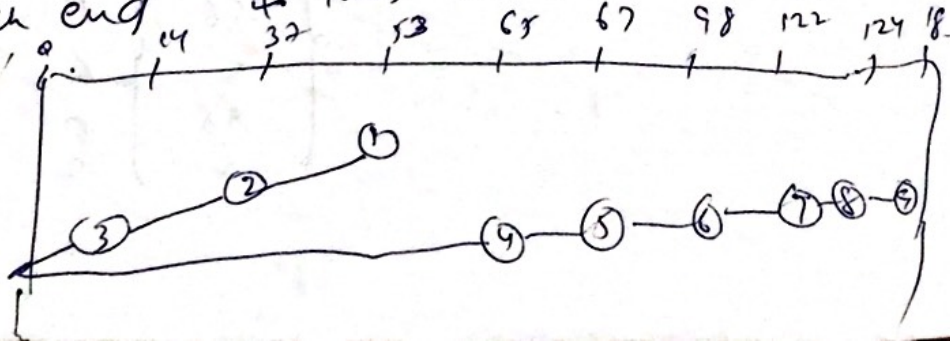


SSTF: Shortest seek time first

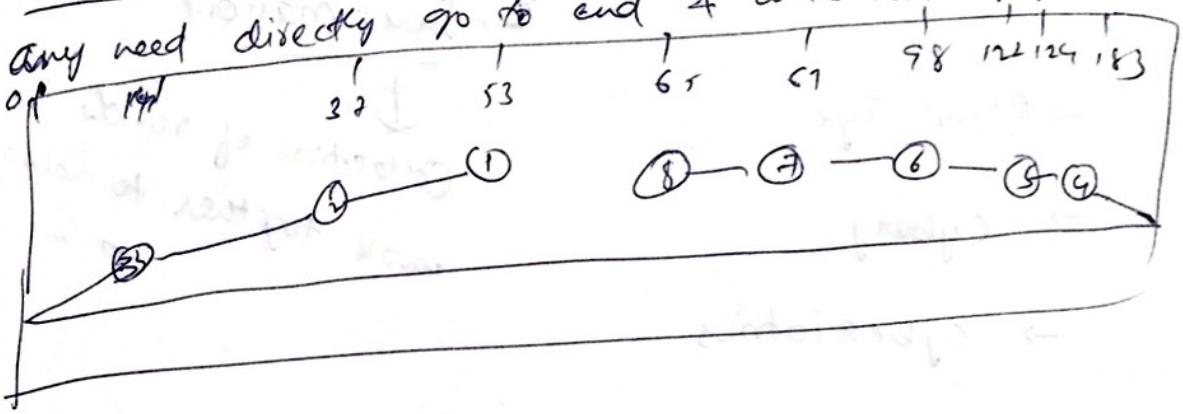


Scan algorithm

take decision to move either left (or right) & reach end & then start in the opposite direction



C-Scan:- while returning back do not satisfy any need directly go to end & while returning start satisfying need.



In Segmentation

Computation of Physical address

Segment	Base	limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Segment Table

logical address

① 0, 430
② 1, 10
③ 2, 500
④ 3, 400
⑤ 4, 112

- 219 + 430 = 649
- 2300 + 10 = 2310
- segmentation fault
- 1327 + 500 = 1827
- Segmentation error

21/4/14

→ Swarmanoid

Collection of robots work together to achieve task

→ Bionic Eye

→ Cyborg

→ cybernetics

→ Rondey Brooks

"Electronics for you" → book for robotics