O.S.

Syllabus :

* Introduction
  1) Defn of O.S.
  2) Functn of O.S.
  3) Objective of O.S.
  4) structure of O.S.
  5) Evolution of O.S.
  6) Types of O.S.
  7) Shell
  8) Modes of execution
  9) System calls of Drivers
  10) Kernel
  11) O.S. structures

* Process
  Defn of process
  Process state
  Process control Block
  Threads
  process vs threads

PAGE NO.
DATE

Process Management :

1) Process creation
   Process termination
   Process suspension

* Process Models
  2 state models
  5 state models
  5 " " with 1 suspend state
  5 " " " 2 " "

* Process scheduling
  1 Types of schedule
  schedule criteria
  scheduling models
  Uniprocessor scheduling Algo [7]

* Process synchronization
  Race condition
  Critical section Problem
  solution to CSP
  Requirment of CSP
  Dekkers Algo
  Peterson's Algo
  Samophore
  Monitors

Message Processing
Procedure consumer Problem
Reader writer problem

* Deadlock:
Defⁿ & condition of Deadlock
Deadlock Precautions
" Avoidance
" Detection
" Recovery
Bankers Algo
Dining philosopher problem

* Memory Management
Def & requirements
fixed partitioning
Dynamic partitioning
Buddy system
Pagging
Segmentation
Virtual memory

* I/O Management
  I/O buffering

* File Management
  RAID
  File orientation & Access

* Disk Management
  Disk Mgt Policies

* S/N

* UNIX

* WINDOWS

5/2/08

Defn of O.S.

O.S. is a program which control the execution of app. programs and acts as an interface b/w application and h/w.

Functions of O.S.

The foll. are the funct^n of O.S.

1.) It controls the execution of application prgms.
2.) It acts as an interface b/w applicatio & h/w

Objectives of O.S.

The foll. are the objectives of O.S.

1.) Convenient :
O.S. should allow the usage of the system in a convenient manner
2.) Efficient :
O.S. should make the utilization of system resources in an efficient manner
3.) Ability to evolve :
O.S. should allow the intro of new features & w/o affecting the existing ones.

Services of O.S.

1) Program Development :
OS should provide tools for program development Eg : Editors

2) Program execution :
OS provides the service of loading the program, the running of the prog. and the termination of the prog which can be normal or abnormal

3) I/O funct^n :
OS provides the service of performing the I/O operations like reading from keyboard, displaying the O/P on the monitor etc.

4) File system manipulation :
OS provide the service of working with the files which include its creation, reading, writing, sharing, renaming and the deletion

5) Resource utilization :
OS provides the service of utilizing the resources in an efficient manner. Resources include processor memory etc.

6) Error Detection & Recovery :
OS provides the service of detecting the errors and also recovering from errors. Errors include floppy disk not responding, virtual memory too low.

7) Security :
OS provides the service of securing the users data from being affected intentionally or unintentionally by other users.
Eg: of security mechanism include password management.

8) Accounting :
OS provides some tools which can help to gather the statistical info regarding the system utilization so that the performance can be improved.

Evolution of OS :

- Serial Processing
- Simple Batch system
- Multiprogrammed system
- Time shared system

Program :

A program is a passive entity as it is merely collection of instr

Process :

A process is an active entity. It is a program in execution along with the current activity which is indicated by the contents of the registers, program counter accumulator etc

Multiprogramming :

It implies a single processor being distributed among several programs.

Multiprocessing :

It implies several processors being distributed among several programs

Multitasking :

It is a logical extension of multiprogramming where in each progm. gets the processor time which is determined by the time slice or the time quantum.

Types of os.

* Batch system
* Multiprogrammed
* Time shared
* Desktop
* Multiprocessor → Assymetric / Symetric
* Distributed
* Network
* Real time system → Hard R.T / Soft R.T.

PAGE NO.
DATE

Hard Real Time :

In this system the computation to be performed is time bounded and if not completed with the time limit then it would damage the system 1>

Soft Real time :

In this the computation to be performed is time bounded & if not completed within the time limit then it may lead to performance degradation. 2>

Shell :

1> Shell is a command interpreter. It is not the part of OS but makes heavy usage of the OS features.

2> It acts as an interface b/w the user & the OS a)

3> Whenever a user logs in a shell is started for that user b)

4> Shell is available in 2 catagories c)

a) CLI b) GUI d)

command line graphic user interface

Modes of execution :

The system supports 2 modes of execution

1.) User Mode :

This is the mode in which the application prog. gets executed. It is considered to be a less privilaged mode because no system related activitie can be performed in this mode.

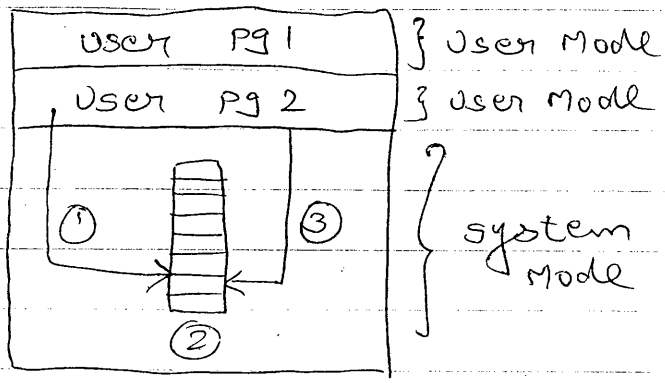
2.) Kernel / system / Priviledged / control or supervisor Mode

OS runs in this mode. It is considered to be a priviledged mode because some system related activities like

a) I/O functions
b) Memory Management Functions
c) The reading & the changing of the control register
d) some portion of the memory can be accessed only in this mode

System calls :

It is a mechanism by which the appl^{n} can demand some service from the OS



1.) A system call is issued by the application & hence the system moves from user mode to system mode

2.) The OS routine performs the intended function.

3.) The control is transferred back from the system mode to the user mode

H → Hogeschool
E → Eindhoven

Kernel

It is the heart of the OS. There are 2 Kernel designing strategies

A) Monolithic Approach

1) In this approach there is no structure
2) Any procedure is allowed to call any other procedure provided the latter performs the funct^n requested by the former.
3) The structure becomes unmanagable if the size of os increases.
4) It is suitable for small OS eg: DOS

B] Layered Approach

1) This approach was proposed by dijkstras for his THE OS

5	THE operator
4	User Prgms
3	I/O Management
2	operator - Process Mgt
1	Memory & Drum Mgt
Layer 0	Processor Allocation & Multiprocessing

Layer 0 :

This layer perforns the basics of multiprogramming i.e to decide which progns should be alloated to the processor.

Layer 1 :

This layer perforns the memory management functions i.e to decide which progns should be kept in main memory & which once should be kept in secondary memory.

Layer 2 :

This layer deals with the operator process communication

Layer 3 :

This layer perforns the I/O related functions.

Layer 4 :

This is the layer where the user progns are located. They need not worry about I/O managemt, memory

management & processor management.

Layer 5 :

The system operator is located at this layer

Eg : UNIX , WINDOWS

PAGE NO.
DATE 12/2/08

Process

* Def^{n} :

Process is an active entity. It is a program in execution including the contents of program counter, accumulator and the registers which indicate the current activity being performed by the process.

* PROCESS CONTROL BLOCK (PCB) :

Whenever a process is created by the O.S. then the info regarding the process is maintained in a file which is called as process control block or task control block (TCB).

Process control Block

Pointer	Process state
Program counter	
Process Number	
Registers	
Memory Limits	
List of open files	
⋮	

PCB includes the following info:

1.) Process state:
It indicates the activity being performed by the process eg running, not running

2.) Program counter:
It indicates the address of the next instructions which is to be executed

3.) CPU registers:
It includes the contents of accumulator, GPRegisters, stacks etc

4.) CPU scheduling info:
It includes the info which is required by CPU for the scheduling of the process which includes the type of the process the priority of the process etc.

5.) Memory Management Info:
It indicates the info. regarding memory managemet which include the page table, segment table etc

6) I/O status Info :

It indicates the info. regarding the I/O which includes the I/O devices been accessed by the process, the I/O files been accessed by the process.

7) Accounting :

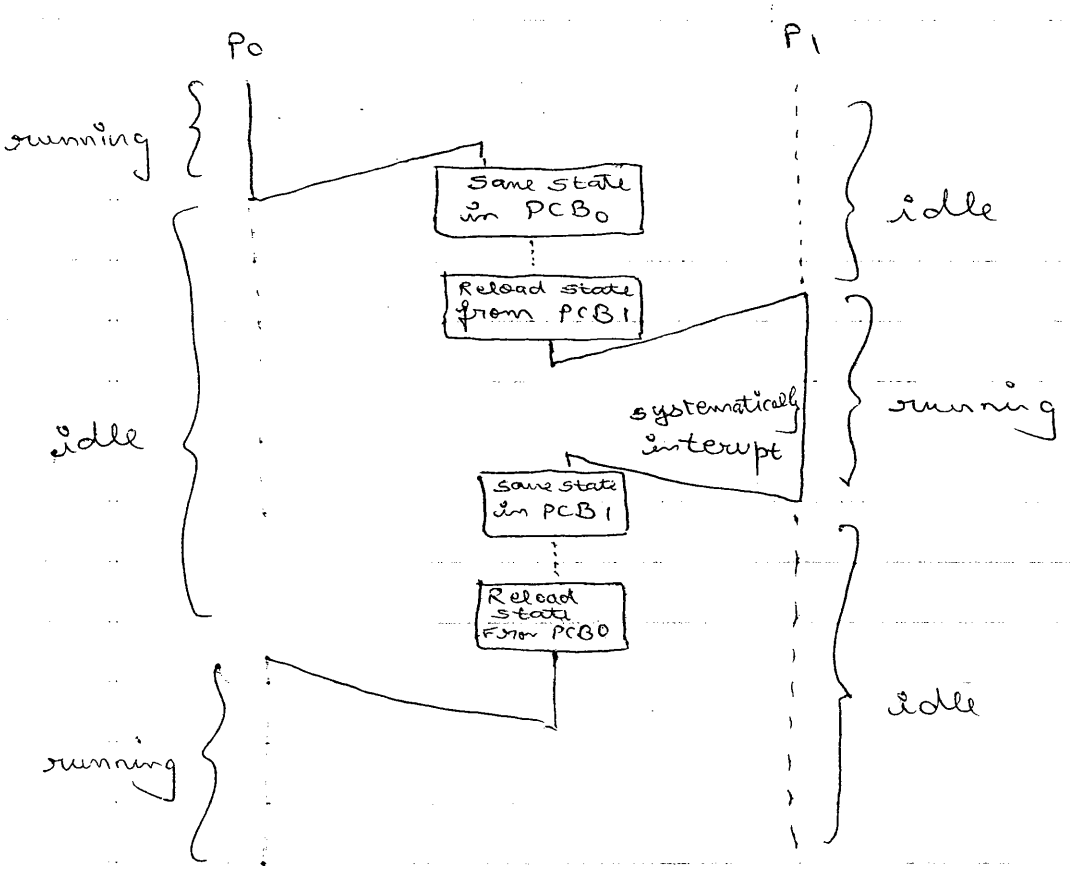
It indicates the accounting related info which includes the process number, time for which the process was asigned to the processor etc.

* CONTEXT OF A PROCESS

contesct of a process is represented by the contents of its PCB.

* contesct switching

It implies saving the contesct of old process and loading the contesct of new process which is scheduled for the execution



May 2006 / IT / 10M

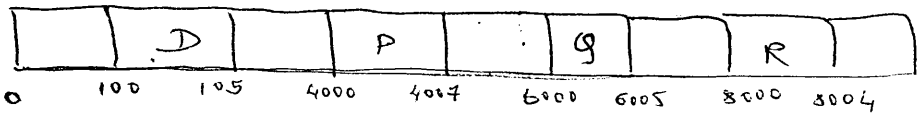
* Following are the 3 process which perform specific task. Sequence for process are each P, Q & R. Give sequence and address in which inst" within the process will execute and why process are executed in the sequence.

Process P		Process Q		Process R	
4000	Addition	6000	Data xfer within seg	8000	Multiply
4001	Subtraction	6001	Take complement	8001	Find > number
4002	Multiply	6002	Take printout	8002	Store results on FD
4003	Division	6003	Addition	8003	Addition
4004	Take i/p from Kbd	6004	Subtraction	8004	Division
4005	SORT	6005	Multiply		
4006	change sign				
4007	calculate Mem				

Soln :

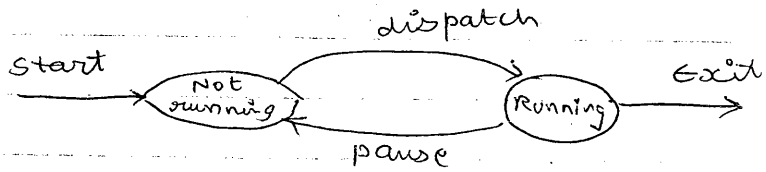
Soln makes the assumption that the process would execute at the most 6 instn at a stretch or it some I/O

1.) 100
2.) 101
3.) 102
4.) 103
5.) 104
6.) 105
7.) 4000
8.) 4001
9.) 4002
10.) 4003
11.) 4004
12.) 100
13.) 101
14.) 102
15.) 103
16.) 104
17.) 105
18.) 6000
19.) 6001
20.) 6002
21.) 100
22.) 101
23.) 102
24.) 103
25.) 104
26.) 105
27.) 8000
28.) 8001
29.) 8002
30.) 100
31.) 101
32.) 102
33.) 103
34.) 104
35.) 105
36.) 4005
37.) 4006
38.) 4007
39.) 100
40.) 101
41.) 102
42.) 103
43.) 104
44.) 105
45.) 6003
46.) 8003
47.) 6005
48.) 100
49.) 101
50.) 102
51.) 103
52.) 104
53.) 105
54.) 8003



Process Model :

* 2 State Process Model :-



State xsition diagram

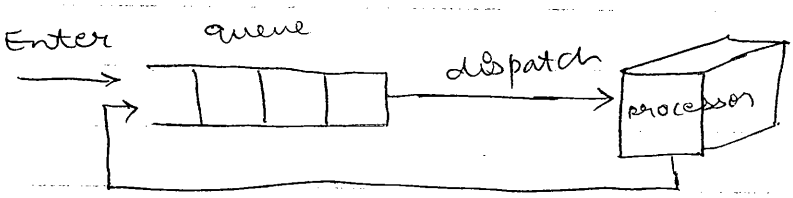
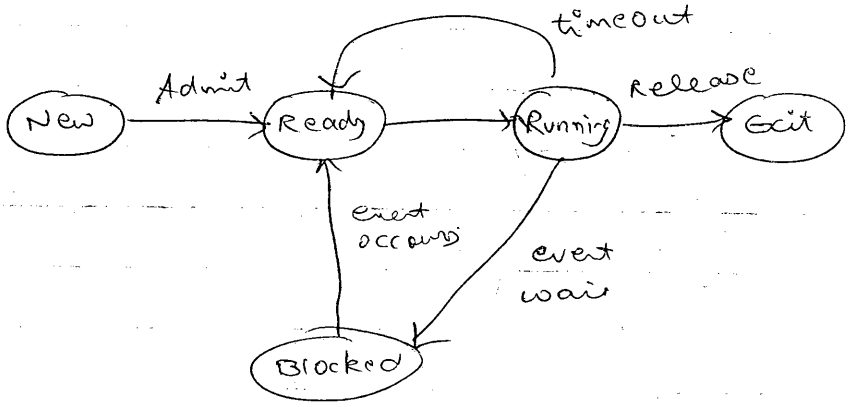
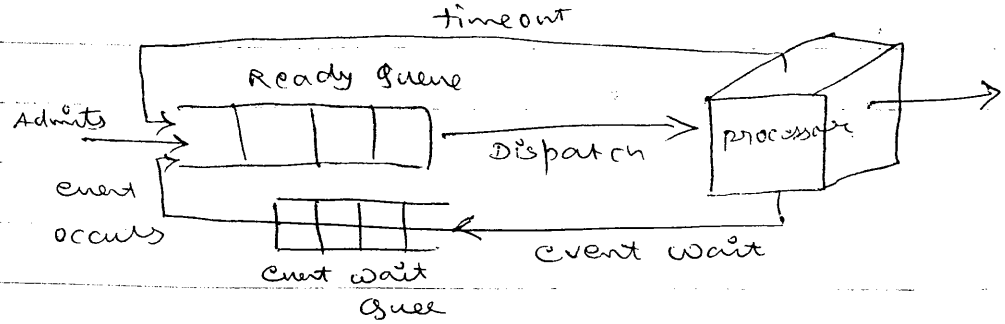


Fig: queuing Diagram

* Five state Process Model

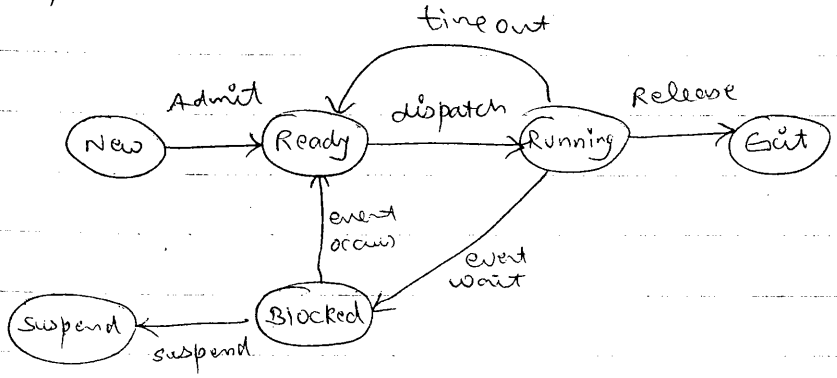


S.T. Diagram

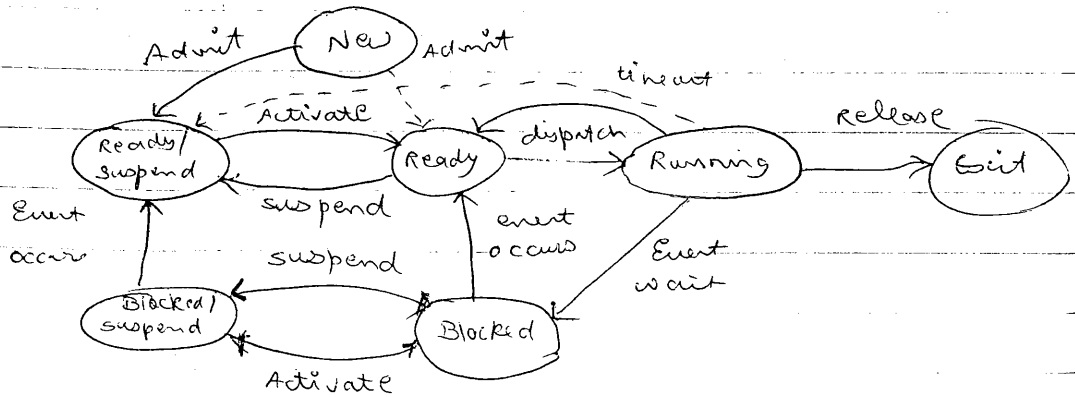


Queing Diagram

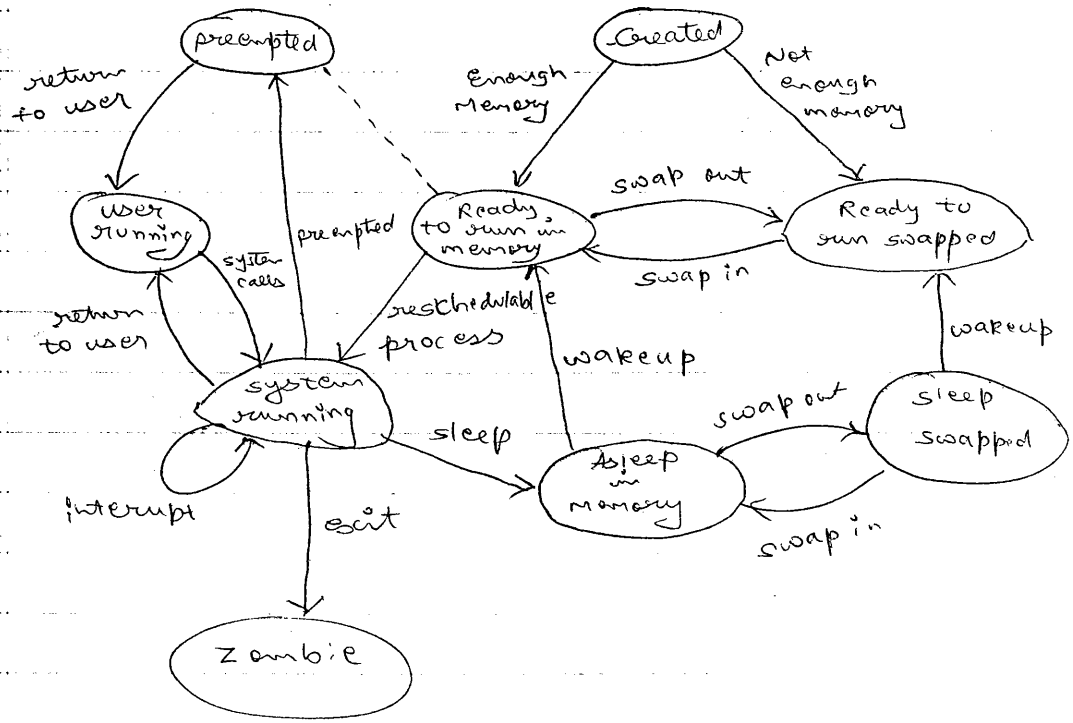
* Five state Process Model with one suspend state



* Five state process Model with 2 suspd state: Ans



UNIX SVR4 (System V Release 4)
State Transition Diagram



Q] Why Zombie state required?

Ans The foll. are the reasons for which the zombie state is required.

1.) To calculate the statistical info about the process.

2.) To return the data to the parent process

Process Management

* Process creation

- created by the parent process
- New interactive login
- New batch job
- spawned by the OS to provide some service

Process Switching:

- Interrupt (timeout)
- Trap (error handling)
- System call (I/O)

- save the context of the process
- Update PCB state (Running → Blocked / ready)
- Move to the appr. queue
- select another process 4 execution

19/2/08

Update PCB state (Ready → Running)
Retire the context of the process

Process Termination :

why:
- Normal completion
- I/O error
- Time limit Exceeded
- Timer Overrun
- Memory Unavailable
- Parent request
- Parent termination
- Bounds violation

HOW:
- Deallocate resources
- Return data to the parent

Threads :

Thread is a light weight process and is considered to be a basic unit of CPU utilization. With each thread the foll. are associated

* thread ip
* program counter
* register
* stack

PAGE NO.
DATE

Thread during its lifetime can be in 1 of the foll. states

→ spawned
→ Ready
→ Running
→ Blocked
→ unblocked
→ Finish

Multithreading :

It refers to the ability of the OS to support more than 1 threads in a single process

Benefits of multithreading :

→ Responsiveness
→ Resource sharing
→ Economy
→ utilization of MPA

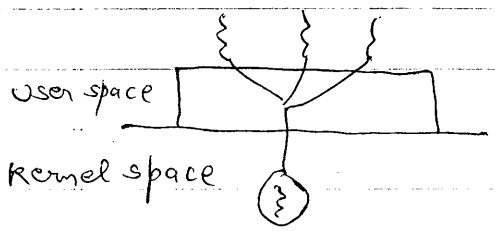
1.) Pr
to
2.) Pr
to
pr
of
3.) Pr
ve
4.) Pr
ge
co
5.) Pr
th
be
6.) Pr
ti
7.) O
co

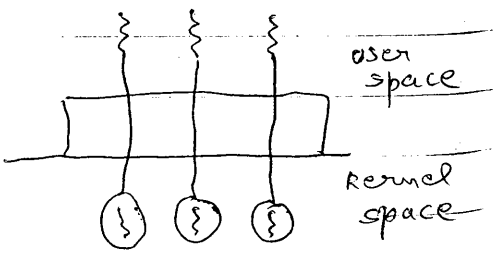
Process	Threads
1.) Process is considered to be heavy weight.	1.) Thread is considered to be a light weight.
2.) Process is considered to be a unit of resource allocation & unit of protection	2.) Threads is considered to be unit of CPU utilization.
3.) Process creation is very costly	3.) Thread creation is very much economical.
4.) Programs using process get executed comparatively slow.	4.) Programs using thread get executed comparatively faster.
5.) Process cannot access the memory area belonging to another process.	5.) Thread can access the memory area belonging to another thread within the same process.
6.) Process switching is time consuming.	6.) Thread switching is faster.
7.) One process can contain multiple threads	8.) One thread can belong to exactly one process.

Thread Implementation

ULT (User Level Thread)	KLT (Kernel level Thread)
1.) Thread creation, switching & management is done by the user. using the user libraries	1.) Thread creation, switching & management is done by the kernel.
2.) Kernel is not aware about the existance of multiple threads.	2.) Kernel is aware of existance of multiple threads.
3.) ULT requires less mode switching	3.) KLT requires frequent mode switching.
4.) If 1 thread gets blocked then all the threads belonging to that process would be blocked.	4.) If 1 thread gets blocked then kernel can schedule another thread belonging to that process for execution.
5.) ULT cannot take the advantage of multiprocessor architecture.	5.) KLT can take the advantage of MPA.

Defn :





Process scheduling :

Process scheduling implies a selection of a process for the execution in a way to meet the objective of the system.

Defn : Schedular :

It is a program which is responsible for the process scheduling.

Types of schedular :

→ Long Term schedular (LTS) :

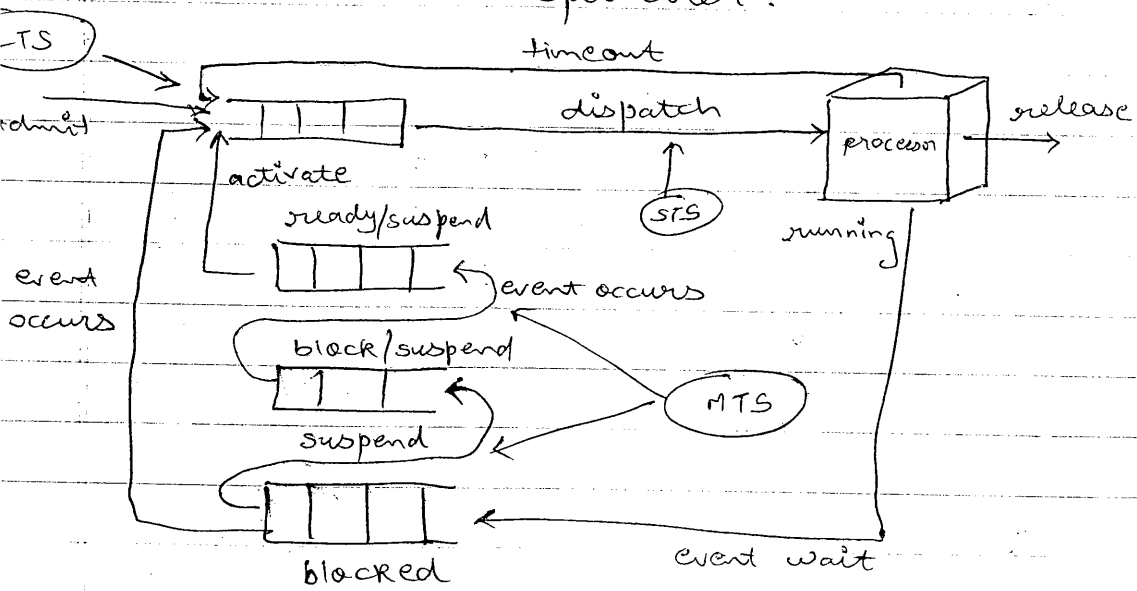
This schedular is responsible for the selection of a process which is to be added into the system and hence LTS controls the degree of multiprogramming (DOM) imply the no. of process that a system can handle w/o any performance degradation

→ Medium Term scheduler (MTS):

This scheduler is responsible for managing the degree of multiprogramming by deciding which prog should be kept in main memory & which is in 2ndary memory. Because of the swapin & swapout operation MTS is also called swapper.

→ short term scheduler (STS)

This scheduler is responsible for the selection of the process from the ready queue to be assigned to the processor. Since this performs the job of dispatching STS is also called dispatcher.



Time :

1> Arrival Time :

It indicates the time at which the process arrived into the system. If not mentioned it is assumed to be zero

2> Burst Time :

It is the time required by the process from the processor for its completion.

CPU scheduling Parameters :

These are the parameters which are used for the measurement of the algo performance

1> Response Time (↓) :

It is the time required to generate the 1st response for the process since its arrival into the system.

2.) Waiting Time :
It is the time for which the process was kept away from processer.

3.) Turn Around Time :
It is the total time required by the process from its submission till its completion.

$TAT = BT + WT - AT$

4.) CPU utilization :
It indicates the value of CPU utilization (ideally 100 %)

5.) Throughput :
It indicates the no. of process completed per unit time.

6.) Fairness :
It implies all process in the system should be given a fair chance for execution.

7.) Priority :
It implies that a process having higher priority should be 1st given a

chance for execution.

Note: Fairness & Priority are contradicting scheduling parameters.

Algorithm Modes:

It implies the mode in which a particular algo would work. There are 2 algo modes

1) Non preemptive Mode

In this mode the process would be taken away from processer if any 1 of the foll. events happen

a) Its completion
b) It requires some I/O
c) It executes a system call.

Eg i) FCFS (First come First serve)
ii) SJF (shortest job first)
iii) Priority

2) Preemptive Mode:

In this mode the process would be taken away from processer if any 1 of the foll. events happen

1) Timeout occurs
2) On the account of the arrival of a new process with a higher priority

eg 1>

eg 1> Round Robin
2> SRT (shortest Remaining Time)
3> Priority

First come First serve

1> This algo works in non preemptive mode
2> The process which arrives in system first would be assigned to the processer 1st.
3> FCFS can be implemented with the help of FIFO queue wherein the new process would always be inserted at the tail of the queue and the process at the head of the queue would always be selected for the assignment to the processor

eg 2>

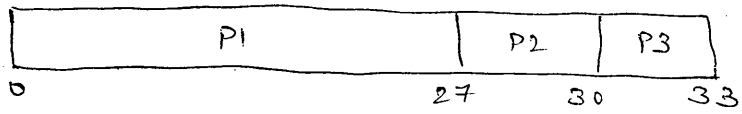
4> convoi effect : It is a result of 1 large process blocking a few smaller process which would degrade the scheduling parameters.

eg 1>

Process	B.T
P1	27
P2	3
P3	3

ART ?
AWT ?
ATAT ?

Soln: Gantt chart



$$ART = \frac{0+27+30}{3} = \frac{57}{3} = 19 ms$$

$$AWT = \frac{0+27+30}{3} = \frac{57}{3} = 19 ms$$

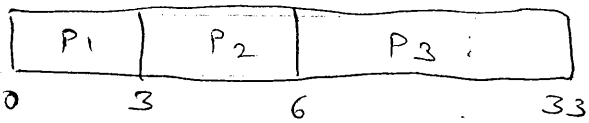
$$ATAT = \frac{(0+27)+(27+3)+(30+3)}{3} = 30 ms$$

eg 2>

Process	B.T
P1	3
P2	3
P3	27

ART ?
AWT ?
ATAT ?

Soln: Gantt chart



$$ART = \frac{0+3+6}{3} = 3 ms \qquad AWT = \frac{0+3+6}{3} = 3 ms$$

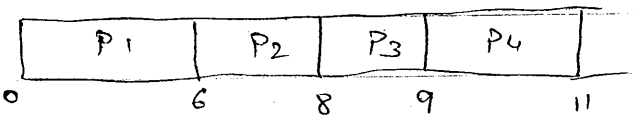
$$ATAT = \frac{(0+3)+(3+3)+(6+27)}{3} = 14 ms$$

PAGE NO.:
DATE:

3.]

Problem	B.T	A.T
P_1	6	0
P_2	2	1
P_3	1	2
P_4	2	3

Soln: Gantt chart



$$ART = \frac{0+6+8+9}{4} = \frac{23}{4}$$

$$AWT = \frac{0+6+8+9}{4} = \frac{23}{4}$$

$$ATAT = \frac{(0+6)+(6+2)+(8+1)+(9+2)}{4}$$

$$= \frac{6+8+9+11}{4}$$

$$= \frac{34}{4} \text{ ms}$$

P_1 P_2 P_3 P_4

$$ART = \frac{(0-0)+(6-1)+(8-2)+(9-3)}{4} = 4.25 \text{ ms}$$

$$AWT = \frac{(0-0)+(6-1)+(8-2)+(9-3)}{4} = 4.25 \text{ ms}$$

$$ATAT = \frac{(0+6)+(5+2)+(6+1)+(6+2)}{4} = 7 \text{ ms}$$

Round Robin Algo

1> It is a preemptive FCFS algo

2> In this algo each process would be assigned to the processer for a time interval which is called time slice or time quantum.

3> Round Robin Algo can be implemented by adding a time slice mechanism to the FIFO queue.

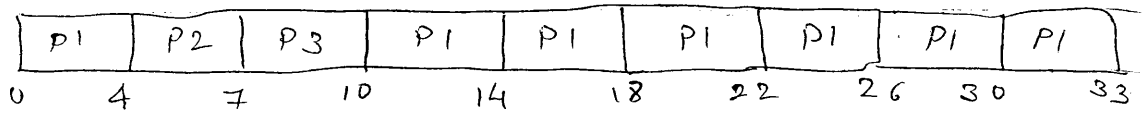
4> If the time slice is too low then more time would be spent in contest switching and if the time slice is too high then other process may suffer from starvation.

5> RR1 implies Round Robin algo with a time slice of 1ms & RR4 implies Round Robin algo with a time slice of 4ms.

Process	B.T
P1	27
P2	3
P3	3

RR4

Soln :



$$ART = \frac{0+4+7}{3} = 3.67 \text{ ms}$$

$$AWT = \frac{(10-4)+(4-0)+(7-0)}{3} = 5.67 \text{ ms}$$

$$ATAT = \frac{(6+27)+(4+3)+(7+3)}{3} = 16.67 \text{ ms}$$

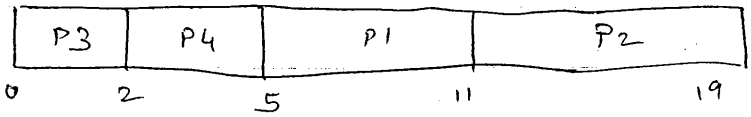
Shortest Job First (SJF)

1) It is a non-preemptive algo.
2) The process which requires the shortest burst time would be given the 1st chance for the execution
3) The difficulty in implementing the SJF algo is that the burst time cannot be exactly determined, it can only be estimated by some empirical formula.

Process	B.T
P1	6
P_2	8
P_3	2
P4	3

ART = ?
AWT = ?
ATAT = ?

Soln: Gantt chart:



$$ART = \frac{5+11+0+2}{4} = 4.5 \text{ ms}$$

$$AWT = \frac{5+11+0+2}{4} = 4.5 \text{ ms}$$

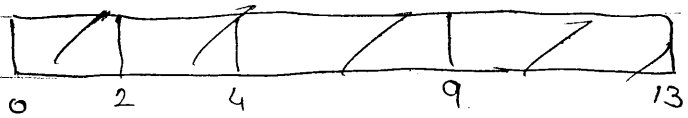
$$ATAT = \frac{(5+6)+(11+8)+(0+2)+(2+3)}{4} = 9.25 \text{ ms}$$

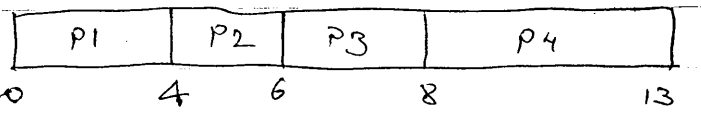
*

Process	B.T.	A.T
P1	4	0
P_2	2	2
P_3	2	3
P4	5	5

ART = ?
AWT = ?
ATAT = ?

Soln :





$$ART = \frac{(0-0)+(4-2)+(6-3)+(8-5)}{4} = 2\ ms$$

$$AWT = \frac{(0-0)+(4-2)+(6-3)+(8-5)}{4} = 2\ ms$$

$$ATAT = \frac{(0+4)+(2+2)+(3+2)+(3+5)}{4} = 5.25\ ms$$

Shortest Remaining Time (SRT)

1.) It is a preemptive SJF algo.
2.) The process having th SRT would be scheduled for the execution first.
3.) The difficulty in implementing SRT algo is that the burst time cannot be exactly calculated only a prediction can be made by using some empirical formula.

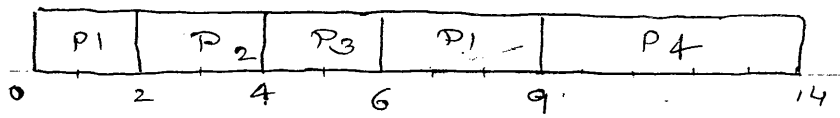
*

Process	B.T	A.T
P1	5	0
P2	2	2
P3	2	3
P4	5	5

AWT = ?
ART = ?
ATAT = ?

Soln:

Process	R.T
P1	~~5 4 3 2 1~~ 0
P2	~~2 1~~ 0
P3	~~2 1~~ 0
P4	~~5 4 3 2 1~~ 0



$$ART = \frac{(0-0)+(2-2)+(4-3)+(9-5)}{4} = 1.25 \text{ ms}$$

$$AWT = \frac{(4-0)+(2-2)+(4-3)+(9-5)}{4} = 2.25 \text{ ms}$$

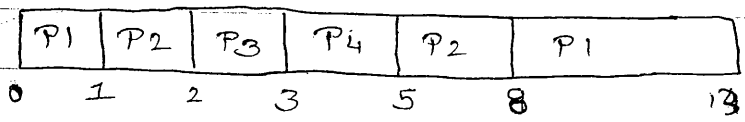
$$ATAT = \frac{(4+5)+(0+2)+(1+2)+(4+5)}{4} = 5.75 \text{ ms}$$

2.)

Process	B.T	A.T
P1	6	0
P2	4	1
P3	1	2
P4	2	3

Soln:

Process	R.T
P1	~~6~~ 5
P2	~~4~~ 3
P3	~~1~~ 0
P4	~~2~~ ~~1~~ 0



$$ART = \frac{(0-0) + (1-1) + (2-2) + (3-3)}{4} = 0 \text{ ms}$$

$$AWT = \frac{(7-0) + (4-1) + (2-2) + (3-3)}{4} = 2.5 \text{ ms}$$

$$ATAT = \frac{(7+6) + (3+4) + (0+1) + (0+2)}{4} = 5.75 \text{ ms}$$

Priority Algorithm :

1> It can work in both pre-emptive & non preemptive mode.

2> It process having the highest priority would be scheduled for execution first.

3> Priorities can be determined by the internal factors like

a) Time limit
b) Memory requirement
c) Number of open files or

priority can be determined by external factors like

a) Type of process
b) Amounts of funds paid for running the process.

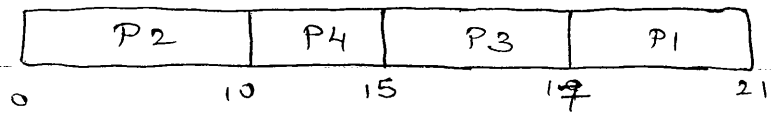
Starvation & Aging

Priority algorithm suffer from a problem which is called starvation wherein a process is kept away from the process for indefinitely for long interval of time.

solution to the problem of starvation is aging in which we logically increment the priority of the process until it reaches a threshold value. after which it would be scheduled for execution.

1.)

Problem	B.T.	Priority
P1	4	4
P2	10	1
P3	2	3
P4	5	2



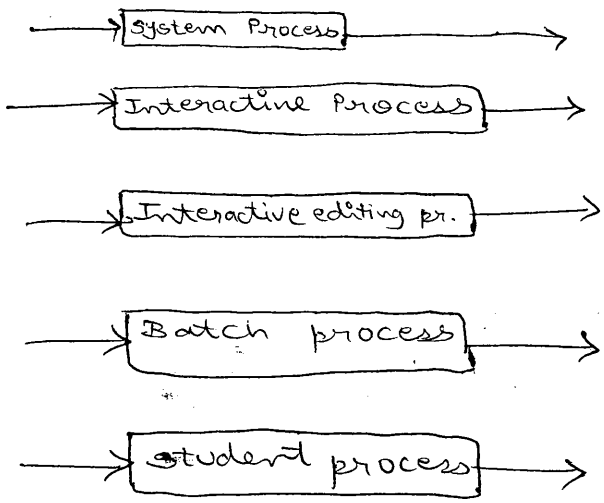
$$ART = \frac{17 + 0 + 15 + 10}{4} = 10.5 \text{ ms}$$

$$AWT = ART$$

$$ATAT = \frac{(17+4) + (0+10) + (15+2) + (10+5)}{4} = 15.75 \text{ ms}$$

Multilevel Priority Queue (MLPQ)

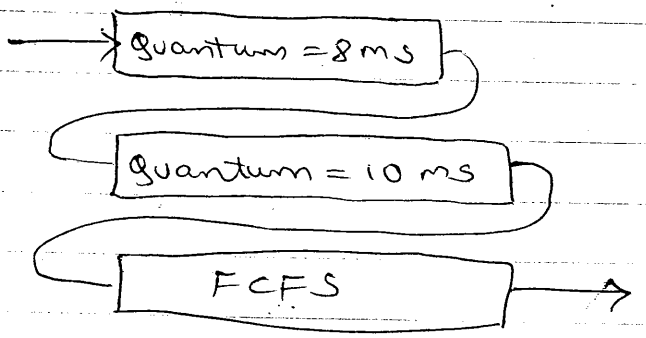
1) In this the ready queue is partitioned into several queues
2) Each queue would be having an absolute priority over the other queues.
3) Each queue can have its own scheduling algo.
4) A process would be admitted in any 1 of the queue depending on the type of the process.
5) In this the process is not allowed to change the queue
6) MLPQ will suffer from a problem which is called starvation.



PAGE NO.
DATE :

MLFQ

1. In this the ready queue is partitioned into several queues.
2. Depending on the feedback of the process the process would be upgraded in the queue.
3. The several queues can be scheduled using some algo.
4. MLFQ does not suffer from starvation.



Interprocess communication

→ Types of process

1) Independent process

Process is independent if it cannot effect or cant be affected by another process executing in the system.

such process do not share the data with another process in the system.

2) cooperative Process :

Process is co-operative if it can affect or can be affected by another process which is executing in the system.

such process share the data with another process executing in the syster

Race condition :

Whenever several process access and manipulate shared data concurrently and the outcome is dependent on the order in which they are executed is called race cond^n

balance = 500

P0 :

```
amt 1 = balance
amt 1 = amt 1 + 100
balance = amt 1
```

P1 :

```
amt 2 = balance
amt 2 = amt 2 - 100
balance = amt 2
```

T0 : P0 : amt 1 = 500	T0 = P1 : amt 2 = 500
T1 : P1 : amt 2 = 500	T1 = P0 : amt 1 = 500
T2 : P0 : amt 1 = 500 + 100 = 600	T2 = P1 : amt 2 = 500 - 100 = 400
T3 : P1 : amt 2 = 500 + 400 = 400	T3 = P0 : amt 1 = 500 + 100 = 600
P4 : P0 : balance = 600	T4 = P1 : balance = 400
T5 : P1 : balance = 400	T5 = P0 : balance = 600

Critical section Problem (CSP)

It is a problem of designing a protocol which would allow the process to co-operate.

The proposed solution to CSP would be of the foll. format

```
do
{
    entry section
    critical section
    exit section
    remainder section
} while (1);
```

1> entry section :

It is the portion of the code in which the process is asking for a permission to enter the critical section.

2> critical section :

It is the portion of the code which is accessing and manipulating the shared data.

3> Exit section :

The process is leaving the critical section & hence it must set the condition which would allow some other

process to enter the critical section.

4) Remainder section :

It is the portion of the code which is not accessing the shared data.

Requirements :

1) Mutual Exclusion :

If a process is inside the critical section then another process should not be allowed to enter the critical section.

2) Progress :

If there is no process inside the critical section & if a process wants to enter the critical section then it should not be stopped by a process which is executing in the remainder section.

3) I

interrupt

3) Bounded waiting.

↳ There exists a limit on the number of times the other process should be allowed to enter the critical section after a process has made a request for the same.
This decision should not be postpone indefinitely

Solution using Lock variable

```
lock = 0 ;
```

P0 :

```
do
{
    while (lock == 1) ;
    lock = 1 ;

    /* C.S */

    lock = 0 ;

    /* R.S */
} while (1);
```

P1 :

```
do
{
    while (lock == 1) ;
    lock = 1 ;

    /* C.S */

    lock = 0 ;

    /* R.S */
} while (1);
```

interrupt →

This solution is rejected because it does not satisfy the criteria of mutual exclusion

Solution using strict Alternation :

turn = 0 ;

P0 :

```
do
{
    while (turn==1);
    /* C.S */
    turn=1 ;
    /* R.S */
} while (1) ;
```

P1 :

```
do
{
    while (turn==0);
    /* C.S */
    turn = 0;
    /* R.S */
}while(1);
```

This soln is rejected because it does not satisfy the progress condition as

Solution using Status Flag

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

P0 :

```
do
{
    flag [0] = true ;
    while ( flag [1] );

    /* C.S */

    flag [0] = false ;

    /* R.S */
} while (1);
```

P1 :

```
do
{
    flag [1] = true ;
    while ( flag [0] );

    /* C.S */

    flag [1] = false ;

    /* R.S */
} while (1);
```

This sol^n is rejected because it voilates the Bounded waiting criteria.

Peterson's solution :

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

P0 :

```
do
{
    flag[0] = true;
    turn = 1;
    while (turn==1 && flag[1]);

    /* C.S */

    flag [0] = false;

    /* R.S */
} while (1);
```

P1 :

```
do
{
    flag [1] = true;
    turn = 0;
    while (turn==0 && flag[0]);

    /* C.S */

    flag [1] = false;

    /* R.S */
} while (1);
```

Dekker's solution :

```
turn = 0;
flag[0] = flag[1] = false
```

P0 :

```
do
{
    flag[0] = true
    while (flag[1])
    {
        if (turn ≠ 0)
        {
            flag[0] = false;
            while (turn ≠ 0);
            flag[0] = true;
        }
    }

    /* C.S */

    turn = 1;
    flag[0] = false;

    /* RS */

} while (1);
```

P1 :

```
do
{
    flag[1] = true;
    while (flag[0])
    {
        if (turn ≠ 1)
        {
            flag[1] = false;
            while (turn ≠ 1);
            flag[1] = true;
        }
    }

    /* C-S */

    turn = 0;
    flag[P] = false;

    /* RS */

} while (1);
```

PAGE NO.
DATE

Solution using interrupt disabling

3)

```
void P (int T)
{
  do
    {
      /* interrupt code disable */
      /* critical section */
      /* interrupt enable */
      (* R.S *)
    } while (1);
void main ()
  {
    parbegin ( P(0) , P(1) ..... P(n));
  }
```

Parbegin is a system which will schedule the concurrent execution of the specified process

The above 3 soln has the foll. 3 drawbacks.

1) The soln will lead to the degradation of CPU performance.
2) The soln is not applicable for multiprocessor environment.

3) If the soln is not used properly then the system might lose the power of generating the interupts.

Solution using special Instn

Some processor designers have proposed some special instn which can perform 2 operations i.e test & set in a single instn cycle

```
boolean testandset (int i)
 {
  if (i==0)
  {
    i=1;
    return true;
  }
   else
    {
      return false;
    }
   }
```

```
void P(int i)
{
  do
  {
  } while (! testandset (bolt));

    /* C.S */

    bolt = 0 ;

    /* R.S */
  } while (1);

int bolt = 0 ;
void main ()
{
  par begin ( P(0), P(1), .... P(n) );
}
```

The soln has 2 drawbacks

1) The soln employes the problem of busy waiting i.e the process is not doing any fruitfull work in its allocated CPU cycles.

2) Th
pro

Le

1) be
co
d

2) Se
1)

3) Ov
Ca

1) C
ne

2) U
Vc
Va
pr
b

3) E
V
Vc
Se
pr
p

2.) The soln may also suffer from the problem of starvation.

Semaphore

1.) Semaphore is a programming language concept which was proposed by dijkstras for inter process comm"
2.) Semaphore consists of 2 elements
   1.) Count 2.) Queue
3.) On the semaphore the foll. operations can be performed
   1.) count is always initialised to a non-negative value.
   2.) Wait operation : It decrements the value of count by 1 and if the value becomes negative (< 0) then the process that exeecuted wait would get blocked
   3.) Signal operation : It increments the value of count by 1 and if the value of count is not +ive (≤ 0) then semaphore will unblock one of the process from its queue & add that process to the ready queue.

4.) General semaphore (counting semaphore):
It is a semaphore which can take any integer value. These semaphore are used for process synchronization.

5.) Binary semaphore (Mutex):
It is a semaphore where the value of count can be either 0 or 1. These semaphore are used for mutual exclusion purpose.

6.) Strong semaphore:
If the queue maintained by the semaphore is implemented as FIFO queue then it is called strong semaphore. strong semaphore never sugger starvation

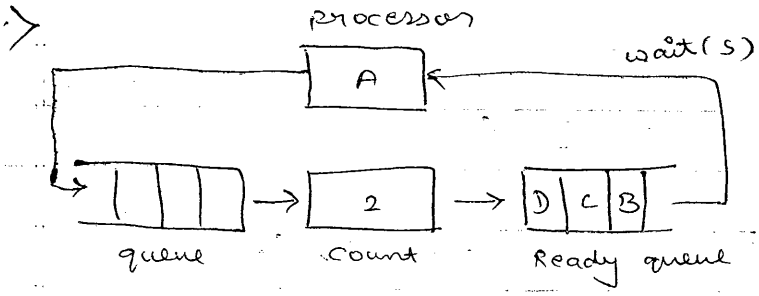
7.) Weak semaphore:
If the queue maintained by the semaphore is implemented as a priority queue then it is called a weak semaphore. Weak semaphore may sugger from starvation.

Pseudo Implementation of Semaphore

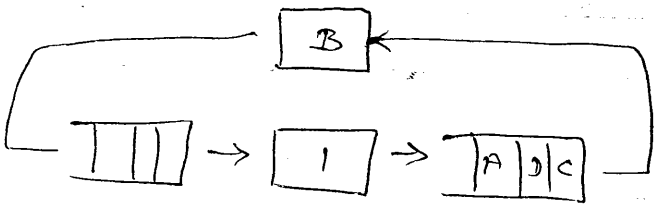
```
struct semaphore
{
    int count;
    if queuetype queue;
}
wait (semaphore s)
{
    s.count--;
    if (s.count < 0)
    {
        Add process to s.queue;
        block the process;
    }
}
```

```
signal (semaphore s)
{
    s.count++;
    if (s.count <= 0)
    {
        Remove a process from s.queue & add it to ready queue, unblock the process;
    }
}
```

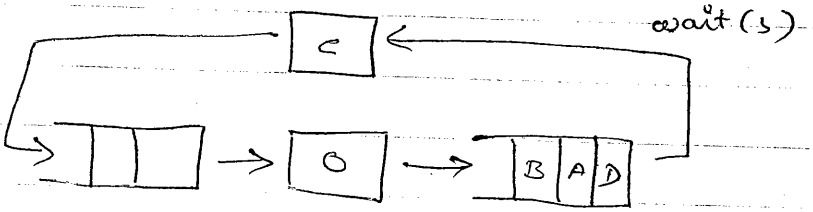
1.)



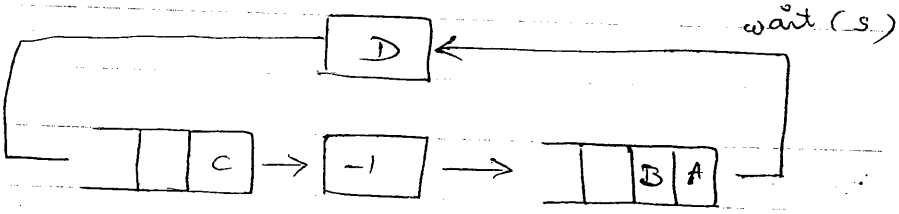
2.)



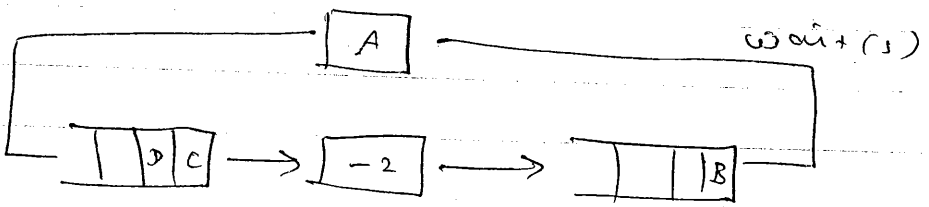
3.)



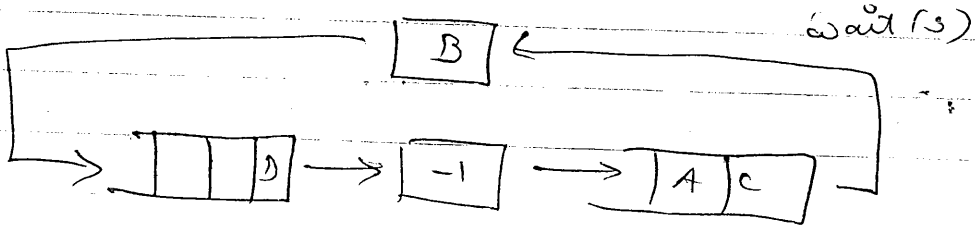
4.)



5.)



6.)



```
void P(int i)
{
  do
  {
    wait (s);

    /* C.S */

    signal (s);

    /* R.S */
  } while (1);
}
```

```
semaphore S = 1;
void main()
{
  parbegin (P(0), P(1)...
            P(n))
}
```

Producer - consumer Problem (unbounded Buffer Problem)

```
semaphore   S = 1    /* mutual exclusion */
semaphore   S = 0    /* proces synchr. */
```

PAGE NO.
DATE

```
void producer()          void consumer()
{                        {
 produce();               wait(n);
 wait(s);                 wait(s);
  append();               take();
 signal(s)                signal(s);
 signal(n);               consume();
}                        }

void main()
{
   Parbegin (producer(), consumer())
}
```

Producer consumer Problem (Bounded Buffer Prob)

```
semaphore   s=1;    /* mutual exclusion */
"           n=0;    // process synchr.
"           e=N;    // process  "
```

```
void producer()          void consumer()          void main()
{                        {                        {
 produce();               wait(n);                 parbegin
 wait(e);                 wait(s);                 (producer(), consumer())
 wait(s);                 take();                  }
 append();                signal(s);
 signal(s);               signal(e);
 signal(n);               consume();
}                        }
```

Reader - Writer Problem

```
Semaphore usem = 1;
semaphore x = 1;
int readcount = 0;
```

```
void writer()
{
  wait (usem);
  WRITE UNIT ();
  signal (usem);
}
```

```
void reader()
{
  wait (x);
  readcount++;
  if (readcount==1) wait(us
  signal (x);
  Readunit ();
  wait (x);
  readunit --;
if (readcount==0)
        signal (usem);
signal (x);
```

Drawbacks of semaphore:

Because the wait & signal operation can appear across the program it becomes difficult to check for the validity of the program

Monitors:

Monitor is a programming language construct which contains local data, condition variables, local procedures, initialization code & some queues.

The following are the characteristics of a monitor

1.) Local Data can be accessed by the procedure within the monitor. No other procedure would be allowed to access the local data
2.) Process can enter the monitor by calling any 1 of its procedures
3.) When a process is inside the monitor no other process would be allowed to enter the monitor.

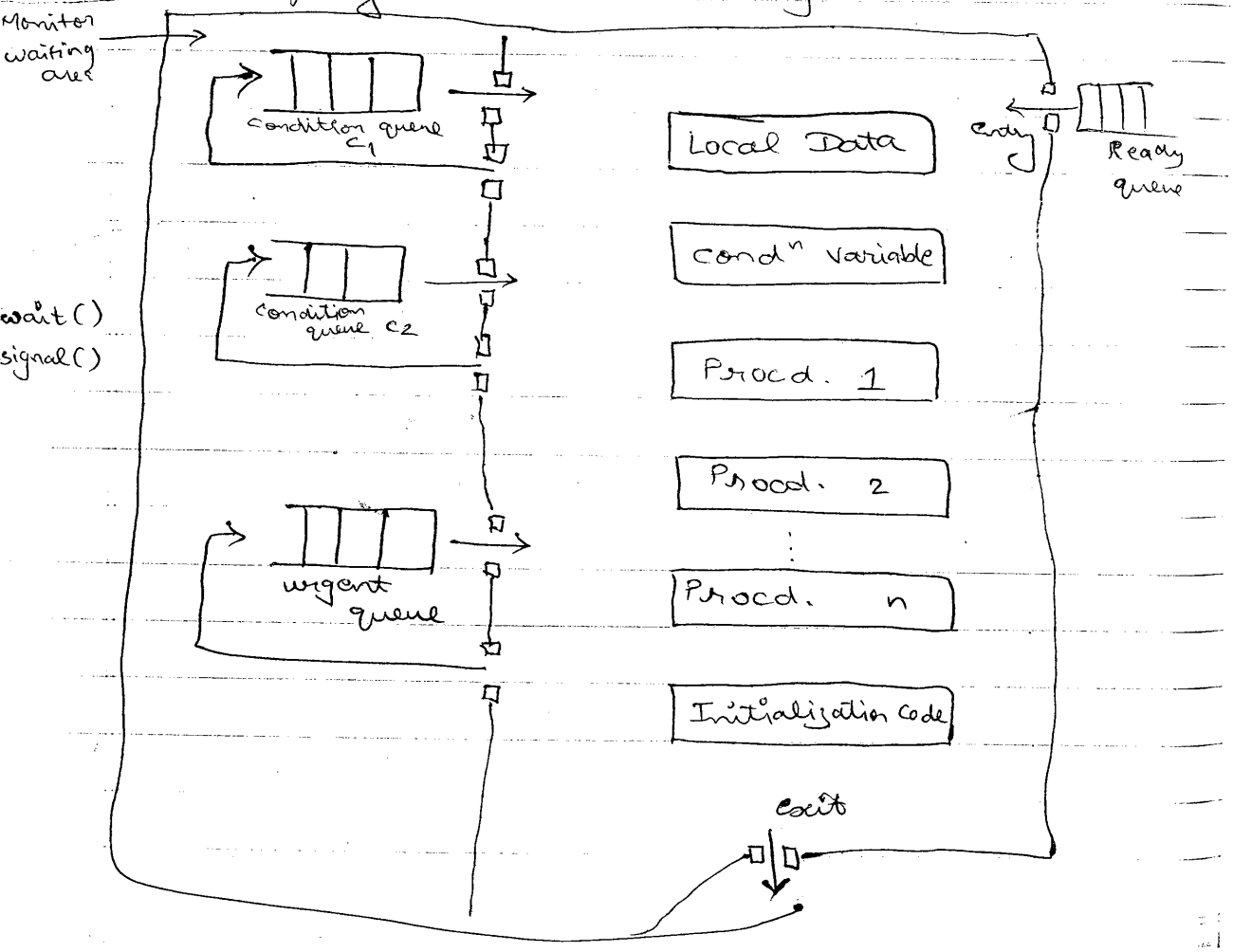
Hence monitor itself provides mutual exclusion.

Operation on condition variable

1.) cwait operation : This operation would suspend the process which executed cwait & add that process to its condition queue. Now the monitor would be available for some other process

2.) csignal operation : This operation would suspend the process which executed csignal & add it to the urgent queue. Then it would bring one of the process from the condition queue of that condition variable into the monitor. If the condition queue is empty then the signal would be lost

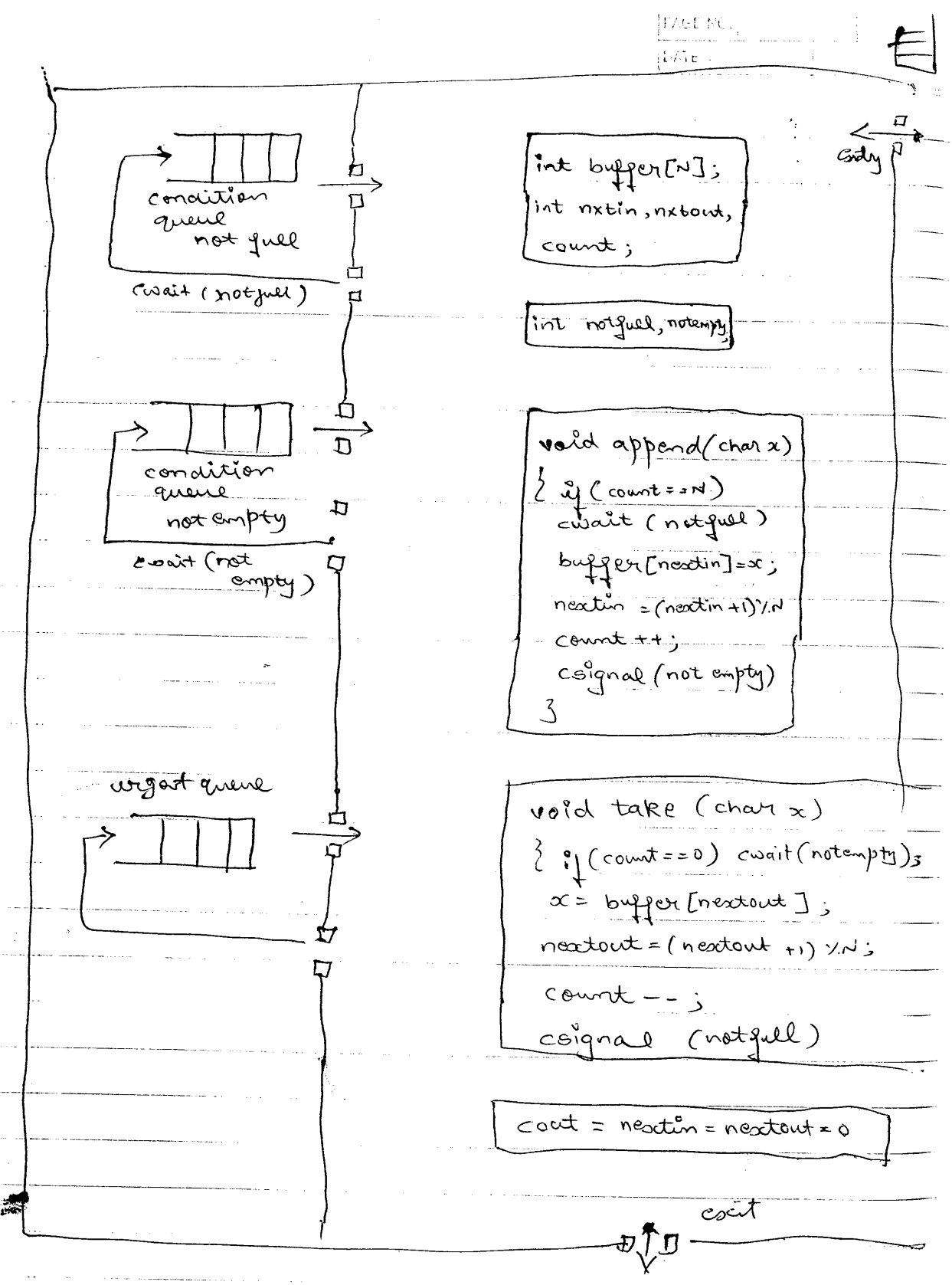
Semaphore	Monitors
1) The programmer has to provide the code for process synchronization & mutual exclusion.	1) Programmer has to provide code for process synchronization
2) Semaphore requires less contesct switching.	2) Monitors requires more contesct switching.
3) wait & signal oper^n can appear anywhere in the program.	3) cwait & csignal operation can appear only within the monitor



as to
process
m

quires
switching.
gnal
n appear
ne monitor


```
void produce ()
{
  produce (x);
  append (x);
}
void consumer ()
{
  take (x);
  consume (x);
}
void main()
{
  parbegin ( producer () , consumer () )
}
```


condition queue not full
cwait (notfull)
condition queue not empty
cwait (not empty)
urgent queue
int buffer[N];
int nxtin, nxtout,
count;
int notfull, notempty;
void append(char x)
{ if (count == N)
cwait (notfull)
buffer[nextin] = x;
nextin = (nextin + 1) % N
count ++;
csignal (not empty)
}
void take (char x)
{ if (count == 0) cwait(notempty);
x = buffer[nextout];
nextout = (nextout + 1) % N;
count --;
csignal (notfull)
}
cout = nextin = nextout = 0
exit

Message Passing :

In msg passing processes communicate by sending msg to each other

⇒ Communication :

→ Direct comm[illegible]

(a) send (P, msg) :

It indicates to send a msg "msg" to process P.

(b) Receive (P, msg) :

It indicates to receive a message "msg" to process P.

Advantage :

→ It is simple to implement

Disadvantage :

→ If the name of the process changes then lot of recoding is to be done.

* Indirect comm"

(a) send (A, msg) :

It indicates to send a message "msg" to a mailbox 'A'

(b) Receive (A, msg) :

It indicates to receive a message "msg" to mailbox 'A'

Advantage :

No recoding required in case of the process name getting changed.

Disadvantage :

It requires an overhead of maintaining a mailbox.

Synchronization :

→ Blocking send
→ Unblocking send
→ Blocking receive
→ Unblocking receive

Solution using Msg Passing :

```
void P(int i)
{
  do
  {
    receive(mutex, null);

    /* C.S */

    send(mutex, null);

    /* R.S */
  } while(1);
}

void main()
{
  create-mailbox(mutex);
  send(mutex, null)
  parbegin( P(0), P(1), ..., P(n) );
  destroy-mailbox(mutex);
}
```

Deadlock

Resource Allocation Graph (RAG) :

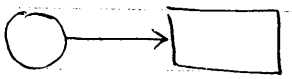
It is a graph which is used for illustrating a state of a system.

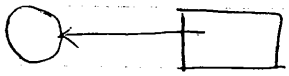
Vertices :

○ → process

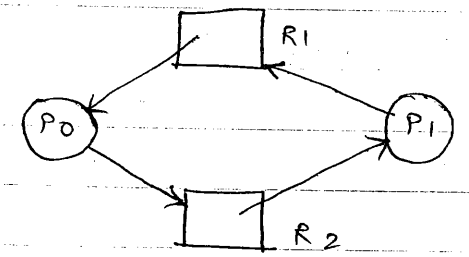
□ → Resource

Edge :

 → request edge

 - Assignment edge

eg :



⊙ → process with single instance

⋮ → process with multiple instance

Defⁿ deadlock :

It is defined as indefinite blocking of set of processes which are competing for the resources or communicating for with each other.

Conditions for deadlock :

+ The following are the 4 conditions that should hold simultaneously for a system to be in a deadlock

OR

The following are the conditions ~~that — should~~ among which atleast one should not hold for the system not to be in a deadlock.

1) Mutual Exclusion :

Process must utilize one resource in a mutually exclusive manner.

2) Hold & wait :

Process must hold some resource & must be waiting

PAGE NO.
DATE

to acquire additional resources

3) No - preemption :

Resource cannot be pre-empted from a process. Process itself would release the resource once its processing is done.

4) circular wait :

There must exist a circular chain of processes wherein each process is waiting to acquire additional resources which are currently holded by another process in the chain.

Deadlock Prevention :

Deadlock ,, implies preventing 1 of the conditions which is necessary & ~~condition~~ sufficient for the system to be in a deadlock.

1> Mutual Exclusion :

Soln : Make all resources sharable.

Drawback :
This soln is applicable for read only file & not applicable for resources like printers which are intrinsically non-sharable.

2> Hold & wait :

Soln 1 :
Aquire all the resources before the process starts its processing

Soln 2 :
If a process wants an additional resource then it must release the resource currently holded by the process and reacquire it along with the additional resource

Drawback 1 :
Resource utilization will be very poor.

Drawback 2 :
starvation may occur.

3.] No - preemption :

Soln :
Preempt the resource from a process which inturn is waiting to acquire additional resource for completing its processing.

Drawback :
The soln can work for those resources whose state can be restored (eg: registers).

4.) circular wait :
In this we no. the resources i.e $f(R) \rightarrow N$
and the process can request an additional resource provided the foll. conditions are satisfied.

$$f(R_i) < f(R_i + 1)$$

Proof by contradiction :

P_0 P_1 P_2 ... P_n

R_0 R_1 R_2 P_3 R_n R_0

$f(R_0) < f(R_1) < f(R_2) \ldots\ldots f(R_n) < f(R_0)$

Drawback :

It is difficult to determine a global numbering system which can satisfy the requirement of all the processes.

Deadlock avoidance :

Deadlock avoidance imply the allocation of the resource should be done to a process provided the allocation does not have a potential of making the system enter a deadlock situation.

Note : From the defn of RAG the foll. can be concluded

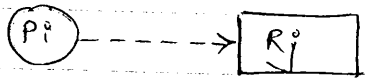
1.) If the resources are having single instance then a cycle in RAG indicates the system is in

deadlock state.

2> If the resources are having multiple instance then a cycle in RAG indicates there may or may not be a deadlock

* Deadlock avoidance algorithm for resources having single instance

<u>step 1</u> : claim edge



<u>step 2</u> : when Pi requests Rj
CE to be converted into RE

<u>step 3</u> :

<u>check</u> :

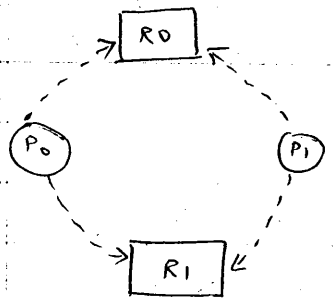
RE should be converted into A.E. provided :-

(i) Rj is available

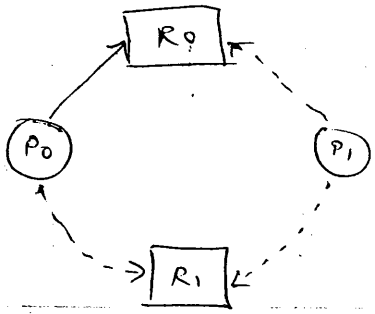
(ii) A.E wont result in any cycle in the RAG

<u>step 4</u> : convert AE to CE

i)

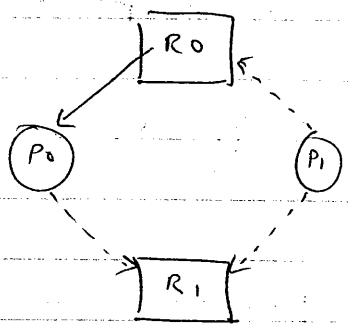


ii)



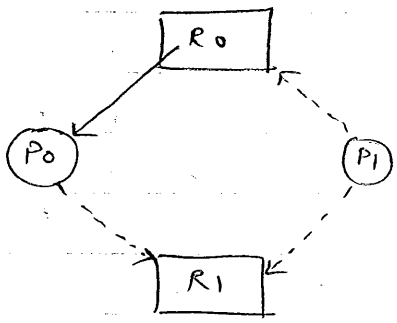
P0 request R0

iii)



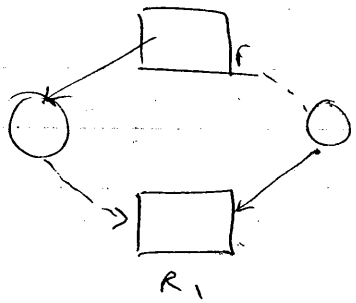
check

iv)



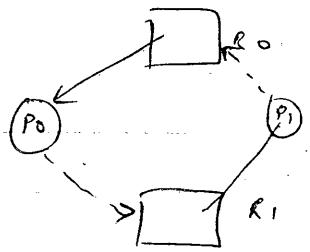
P0 granted R0

v)



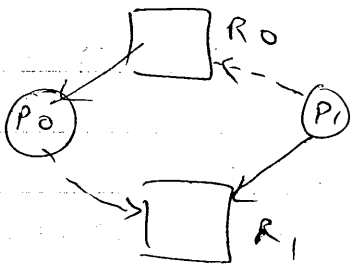
P1 request P1

(vi)



check X

(vii)



P1 not granted R1

* Deadlock avoidance for resources having multiple instances

Dijkstras proposed the bankers algo for avoiding the deadlock when the resources are having multiple instances.

Bankers algo requires the following data structures

n be the no. of processes
m be the no. of resources

available [m] :
It is a vector of length 'm'

available [j] = k :
There are k instances of resource Rj available.

max [m * n] : It is a vector of length m x n

max [i, j] = k : It implies that process Pi could require at the max k instances of resource Rj.

3.] ALLOCATED [n * m] : It is a vector of length n x m

Allocated [i, j] = k : It implies that process Pi would require to allocate k inst. of resource Rj.

4.] Need [n x m] : It is a vector of length n x m

Need [i x j] = k : Process Pi would need k instances of Rj

Need [i, j] = Max [i, j] - Allocated [i, j]

Safety Algo :

Step 1 : Work = available
FINISH [i] = false for i = 1 to n

Step 2 : Find an 'i' such that
FINISH [i] = false
Needi ≤ work go to step 3
else go to step 4

PAGE NO.
DATE

Step 3 : FINISH [i] = true
WORK = WORK + Allocated ; go to step 2

Step 4 : if FINISH [i] = true for i = 1 to n
then the system is in a safe mode
else the system is in unsafe mode.

Resource Request Algorithm :

Step 1 : Process P_i requests Request;

Step 2 : if Request i ≤ Need; then go to step 3
else stop

Step 3 : if Request i ≤ Available then go to
step 4 else stop.

Step 4 : Make a pretention that request
has been granted.

Available = Available − Request ;
Allocated i = Allocated i + Request i
Need i = Need i − Request ;

→ Run Safety Algo.

if the system is in a safe state then grant request to Pi, else undo calculation of step 4 and deny request i to Pi

*] Allocated Max Available

	A	B	C
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2

	A	B	C
P0	7	5	3
P1	3	2	2
P2	9	0	2
P3	2	2	2
P4	4	3	3

A	B	C
3	3	2

Q1) $P_S < P_1\ P_3\ P_4\ P_2\ P_0 >$ a safe sequence?

Q2) is there any other safe sequence?

Q3) can request = (1, 0, 2) be allocated?

Soln:

Step 1

Need = Max - Available

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

	T	T	T	T	T
FINISH	F	F	F	F	F
	P0	P1	P2	P3	P4

Work

A	B	C	...				
3	3	2	+	2	0	0	(P1)
5	3	2	+	2	1	1	(P3)
7	4	3	+	0	0	2	(P_4)
7	4	5	+	3	0	2	(P_2)
10	4	7	+	0	1	0	(P0)
10	5	7					

After running the safety algo the system is in a safe state & hence the given sequence is safe

Soln 2

	T	T	T	T	T
FINISH	F	F	F	F	F
	P0	P1	P2	P3	P4

WORK						
A	B	C				
3	3	2	+	2	0	0 (P1)
5	3	2	+	2	1	1 (P3)
7	4	3	+	0	1	0 (P0)
7	5	3	+	3	0	2 (P2)
10	5	5	+	0	0	2 (P4)
10	5	7				

After running the safety algo. because the system is in a safe state $< P1, P3, P0, P2, P4 >$ is another safe sequence

Soln 3 :

P_1 request Request 1 = (1, 0, 2)

* Request 1 ≤ Need$_1$
  (1, 0, 2) ≤ (1, 2, 2)
* Request 1 ≤ Available
  (1, 0, 2) ≤ (3, 3, 2)

Need

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

Allocated

	A	B	C
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

Available

A	B	C
2	3	0

WORK	2	3	0

FINISH	P0	P1	P2	P3	P4
	T	T	T	T	T
	F	F	F	F	F

WORK

2	3	0	+	3	0	2	P1
5	3	2	+	2	1	1	P3
7	4	3	+	0	1	0	P0
7	5	3	+	3	0	2	P2
~~10~~ 7	5	5	+	3	0	2	P4
10	5	7					

After running the safety algo because the system is in safe state request 1 can be granted.

(P) Ce

Allo

A 1

P0	0
P1	1
P2	1
P3	0
P4	0

P0
P1
P2
P3
P4

Fig

(P) Consider the following snapshot of system.

	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	P0	0	0	1 2	1	5	2	0
P1	1	0	0	0	P1	1	7	5 0				
P2	1	3	5	4	P2	2	3	5 6				
P3	0	6	3	2	P3	0	6	5 2				
P4	0	0	1	4	P4	0	6	5 6				

Need = Max − All

Need

	A	B	C	D
P0	0	0	0	0
P1	0	7	5	0
P2	1	0	0	2
P3	0	0	2	0
P4	0	6	4	2

FINISH

F	F	F	F	F
P0	P1	P2	P3	P4

WORK

1	5	2	0	+	0	0	1	2	P0
1	5	3	2	+	1	3	5	4	P2
2	8	8	6	+	1	0	0	0	P1
3	8	8	6	+	0	6	3	2	P3
3	14	11	8	+	0	0	1	4	P4
3	14	12	12						

Deadlock Detection

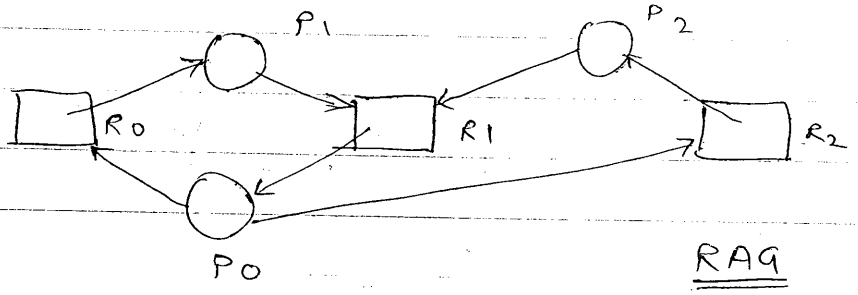
If the system does not implement any strategy for deadlock prevention or avoidance then there are chances that system may enter into a deadlock state.

In deadlock detection strategy we try to identify where the deadlock has occured so that a recovery can be made from the deadlock.

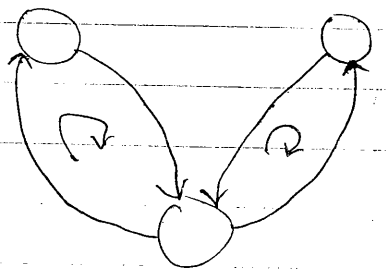
* Deadlock detection for resources with single instance

In this case from the RAG we can construct a wait for graph (WFG) by removing the resource vertice & colapsing the appropriate edges.

After const. WFG we must run a cycle detection algo. If there exist a cycle then it implies that a system is in deadlock state & the process involved in the cycle are deadlock.



RAG



WFG

* Deadlock detection with multiple instance for resources

Deadlock Detection Algo :

Step 1 : Work = Available
If allocated $i \neq 0$ then FINISH [i] = false
else FINISH [i] = true

Step 2 : Find an 'i' such that
FINISH [i] = false
Request i ≤ work go to step3
else go to step 4

Step 3 : FINISH [i] = true
work = work + allocated i go to step 2

Step 4 : If finish [i] = false, for any P=1 to n
then the system is deadlocked
else the system is not "

instance.

i] = false
] = true

to step 2

] P=1 to n
ocked
"

P.]

	Allocated			Request			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

1.> is system deadlocked?

2.> if request 2 = (0, 0, 1) then would the system be deadlocked?

Work

A	B	C					
0	0	0	+	0	1	0	(P0)
0	1	0	+	3	0	3	(P2)
3	1	3	+	2	0	0	(P1)
5	1	3	+	2	1	1	(P3)
7	2	4	+	0	0	2	(P4)
7	2	6					

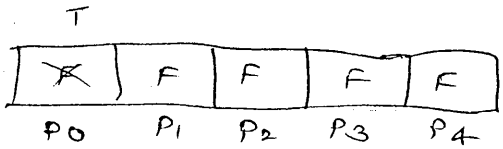
	T	T	T	T	T
FINISH	T	T	T	T	T
	P0	P1	P2	P3	P4

Step 2 Make P_2 as 0 0 1.

Work

0 0 0 + 0 1 0 (P0)

0 1 0 +



After running the deadlock detection algo we can see that the system is in a deadlock state & P_1, P_2, P_3, P_4 are involved in the deadlock.

Deadlock Recovery

The foll. are the techniques which can be used for recovering from a deadlock

1) Terminate all the processes.
2) successively terminate the processes and after each termination run to deadlock detection algo.

2. If deadlock still persist then repeat the step.

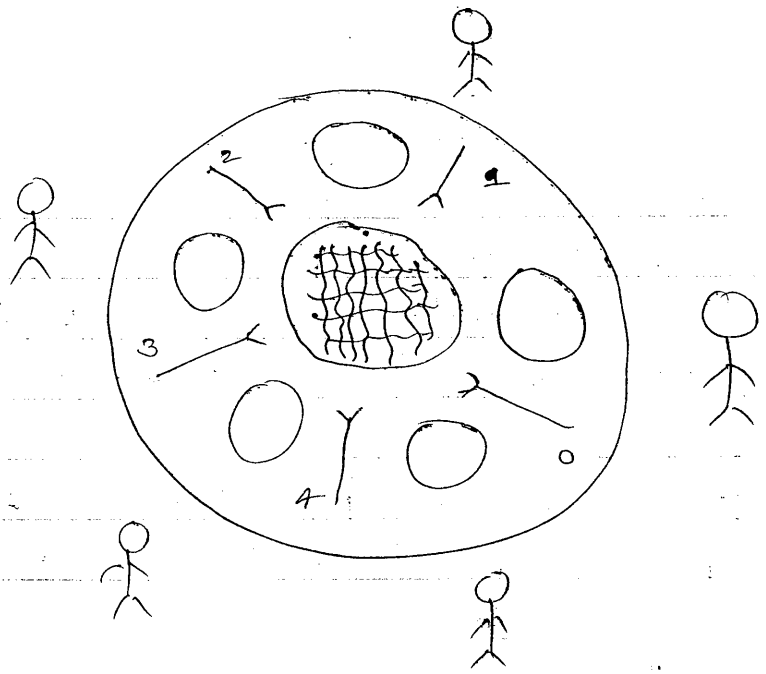
3) Successively pre-empt the resources from the processes which are involved in the deadlock.

After each preemption run the deadlock detection algo. & if the deadlock still persist then repeat the step.

4) Rollback to a previously defined checkpoint & restart the processes

This soln requires the rollback & restart mechanism to incorporated into the system.

Dining Philosopher Problem




```
semaphore fork[5] = { 1, 1, 1, 1, 1 }
```

Ph 1 :

```
void philosopher (int i)
{
    think();
    wait ( fork[i] );
    wait ( fork[ (i+1) % 5 ] );
    eat();
    signal ( fork [ (i+1)%5 ] );
    signal ( fork [i] );
}
void main()
{
    parbegin ( philosopher(0) ..... );
}
```

Soln 2 :

~~Semaphore~~

~~Semaphore fork [5] = {1, 1, 1, 1, 1}~~

The above soln has a potential to go into a deadlock situation.

Hence we appoint a room attendant so that we can target the circular chain condition.

Soln 2 :

```
Semaphore room = 4 ;
semaphore fork [5] = {1, 1, 1, 1, 1}

void philosopher ( int i )
{
 think ( ) ;
 wait ( room ) ;
 wait ( fork [i] ) ;
 wait ( fork [(i+1) % 5 ] ) ;
 eat ( ) ;
 signal ( fork (i+1) % 5) ] ;
 signal ( fork [i] ) ;
} signal (room);
    void main()
{ parbegin ( philosopher (0), ....(4)) }
```

Memory Management

It is the technique which is used for the management of the memory & the component of OS. which performs it is called as memory manager.

Requirements of Memory Management

* Relocation
* Security
* Protection
* Logical organization
* Physical organization

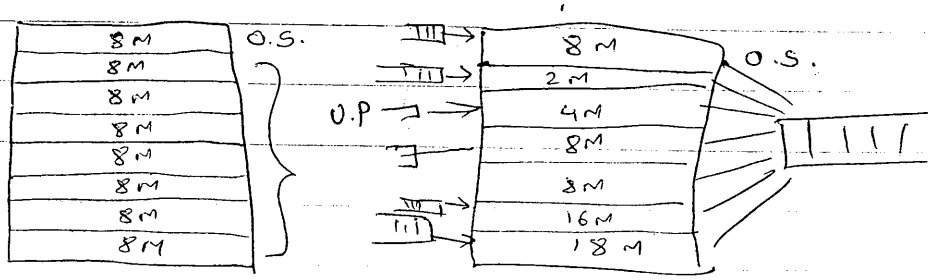
Memory Management Techniques

* Fixed Partitioning
* Dynamic "
* Buddy system
* Paging
* Segmentation.

Fixed Partitioning:

1) In FPartitioning memory is divided into fixed no. of partition
2) Size of partition may be equal or unequal. In a partition one process can be kept at a time.
3) If the size of partition are equal then the placement algo is very simple if the size of partition are unequal then placement algo complexity increases.
4) FP suffers from internal fragmentation.
5) FP limits the no. of process that can be active in a system.
6) Programmer should be aware about the size of partition.

eg) FP is used in IBM-360 where it is called as MFT (multiprogramming with fixed no. of tasks)



Equal size partition

unequal size partition

Dynamic Partitioning

1) In this memory is not divided into no. of partitions
2) In D.P for placement fitting algo are used

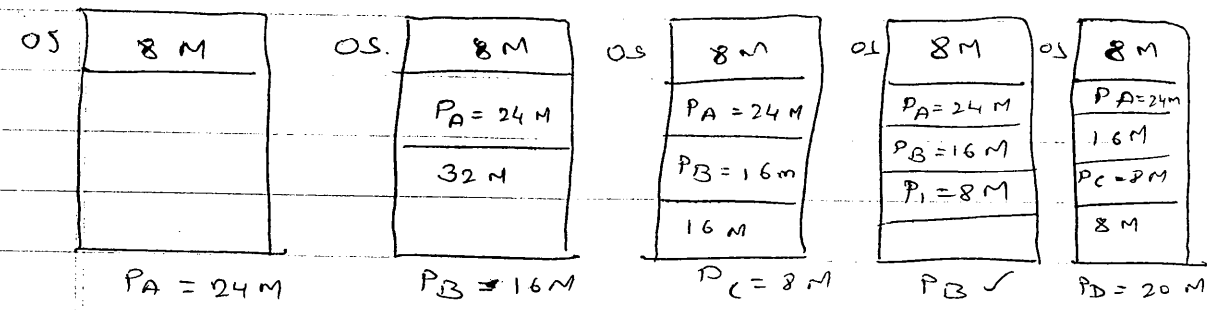
3)

Fitting Algo

(i) **First Fit** : In this the search begins for 1st user loct^n
(ii) **Next fit** : In this the search begins from the last allocation.
(iii) **Best fit** : It searches the entire user location & makes the best allocation.
(iv) **Worst fit** : It searches the entire user location & makes the worst allocation.

3) DP suffers from internal fragmentation.
4) DP does not limit the no. of active process in a system.
5) Programmer need not be aware the size of the partition.
eg: DP is used in IBM 360 where it is called as MVT i.e multiprogr.

with variable no. of tasks.



(P) May 2006 / 10M

The available space list of 1K word memory has the foll. entries at time t.

Relogin Address	Size (word)
0	200
250	150
450	100
700	185
999	25

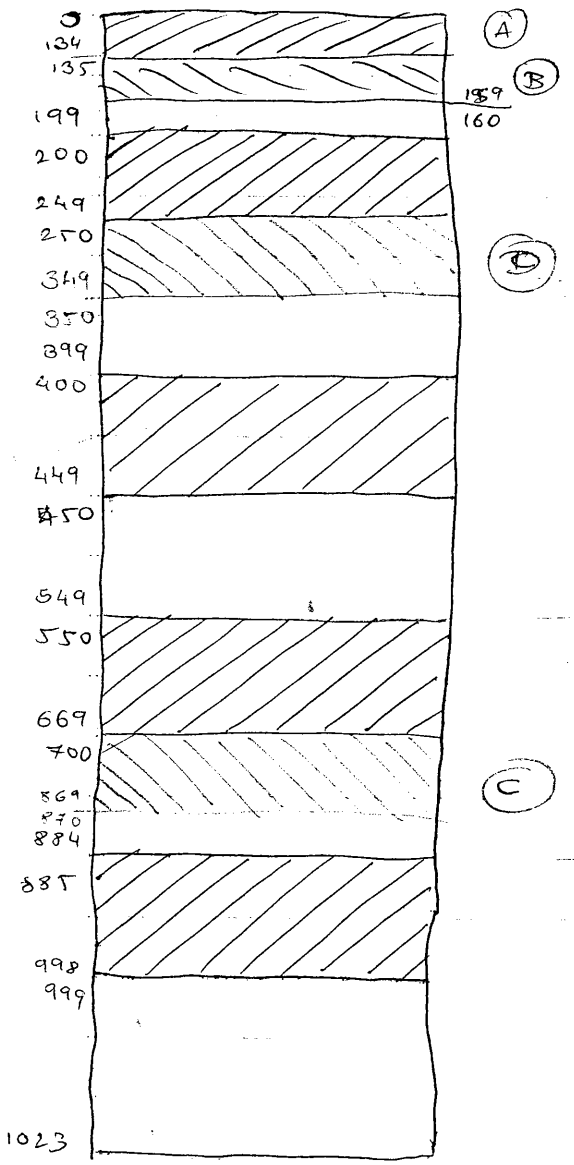
The foll sequence of allocation requests is then received.

Time	t+1	t+2	t+3	t+4
Size of block	135	25	170	100

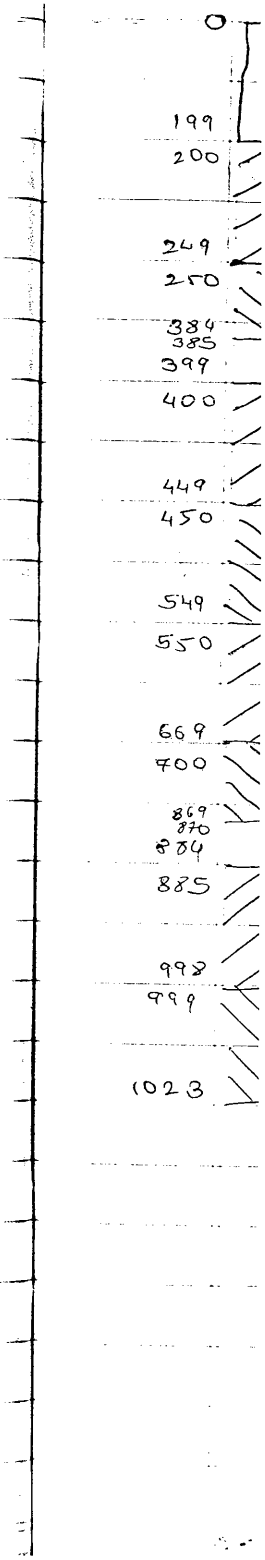
Determine the available space list after all request have been serviced using

1.) First Fit
2.) Best Fit

Soln: First Fit

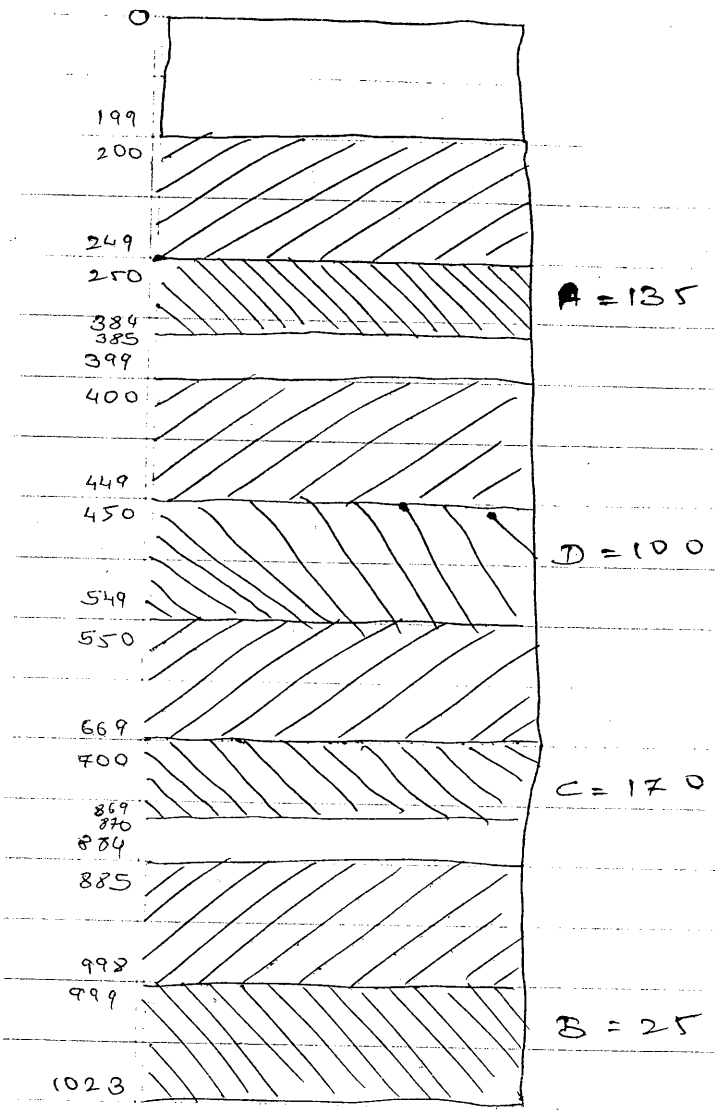


RA	SIZE
160	40
350	50
450	100
870	15
999	25



ist

Best Fit

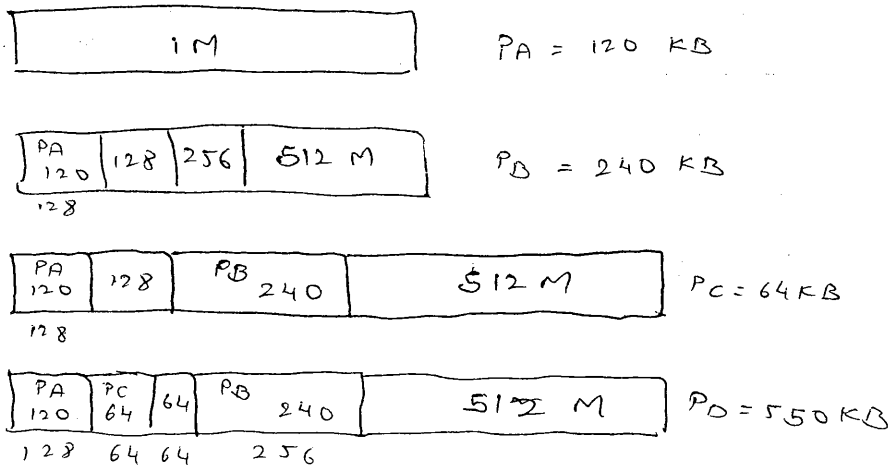


RA	SIZE
0	200
385	15
870	15

Buddy System :

1) In buddy system memory is not divided into no. of partitions
2) For placement buddy algo is used which continuosly divides the memory into 2 until the allocation can be done
3) Buddy system suffers from both internal & external fragmentation
4) Buddy system does not limit no. of active process in the system
5) Programmer need not be aware about the size of the partitions

Eg: Buddy algo is used for managing the part of the memory where OS resides.



Paging :

1> Paging is a memory management technique which allows the memory allocation to be non contiguous

2> Logically the memory is divided into pages & physically the memory is divided into a no. of frames

3> The size of a page is always equal to the size of the frame

4> When a process begins its execution its pages needs to be loaded in the available frames.

5> Each process needs to maintain a page table which provides the mapping b/w the page no. & frame no.

6> Paging suffers from internal fragmentat

7> It does not limit the no. of active process in the system

8> Programmer need not be aware about the size of the frames / pages.

Process A

Page 0
Page 1
Page 2
Page 3

Free frame list

13
10
12
16
15

Frame 10
11
12
13
14
15
16
17

Page Table

Page #	Frame #

efore Allocation

Process A

Page 0
Page 1
Page 2
Page 3

Free Frame List

15

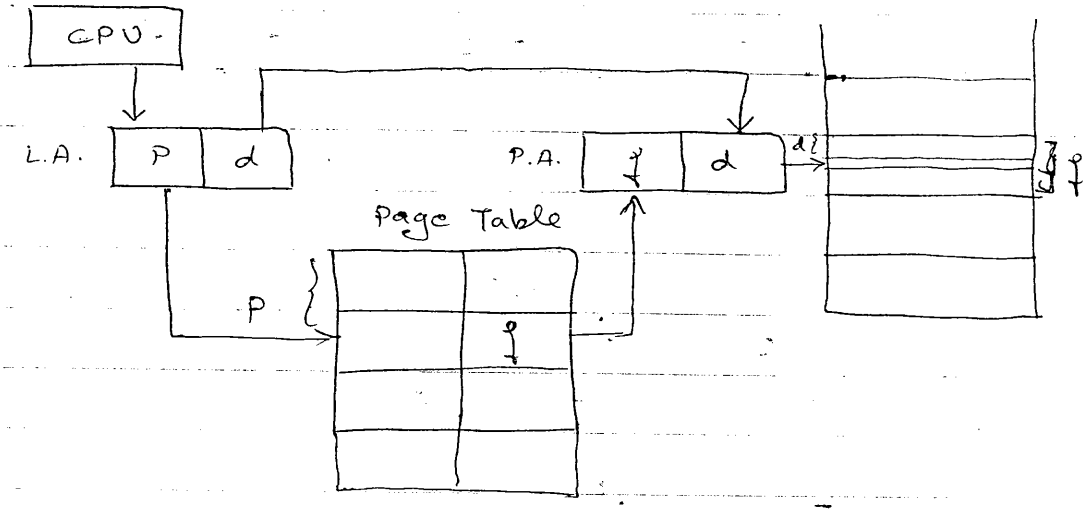
Page Table

Page #	Frame #
0	13
1	10
2	12
3	16

Frame page 1	10
	11
Page 2	12
Page 0	13
	14
	15
Page 3	16
	17

After Allocation

Logical address to physical address Translation



* Where should we keep the page table

1> Registers :

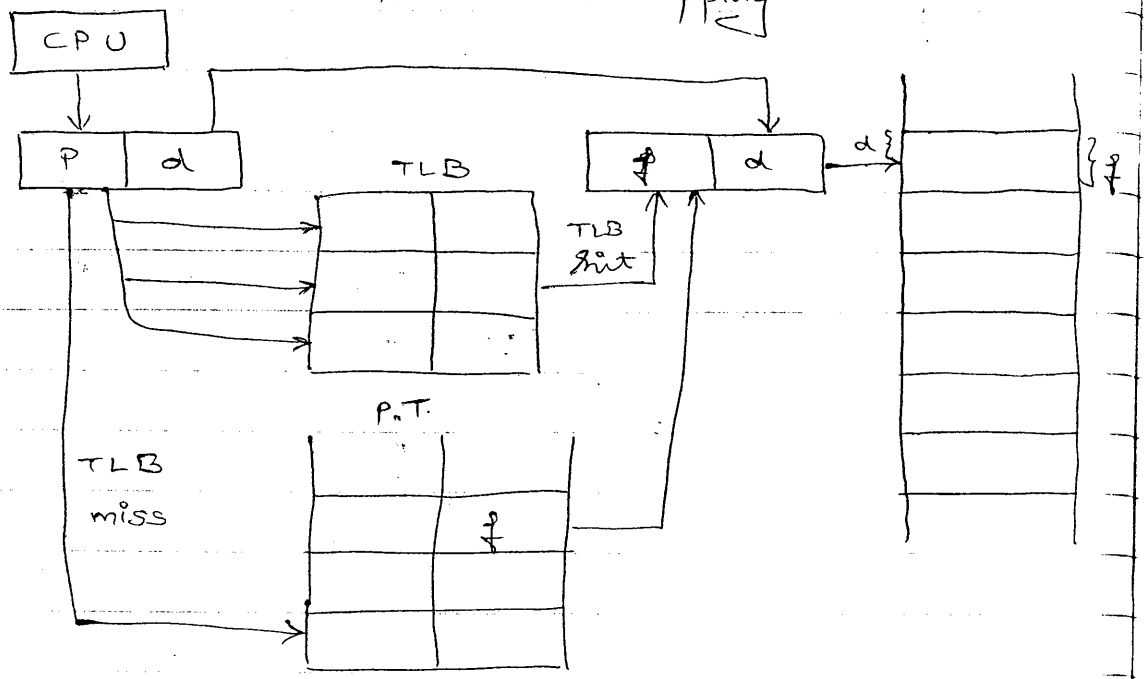
It is applicable if the size of pagetable is small

2> Memory :

In this case the size of page table can vary. Drawback of keeping the pagetable in memory is the speed of access becomes slow by a factor of 2.

To increase the speed we can use a translation look aside buffer (TLB).

Associative Mapping



Q How to handle large page table ?

2 Level Paging :

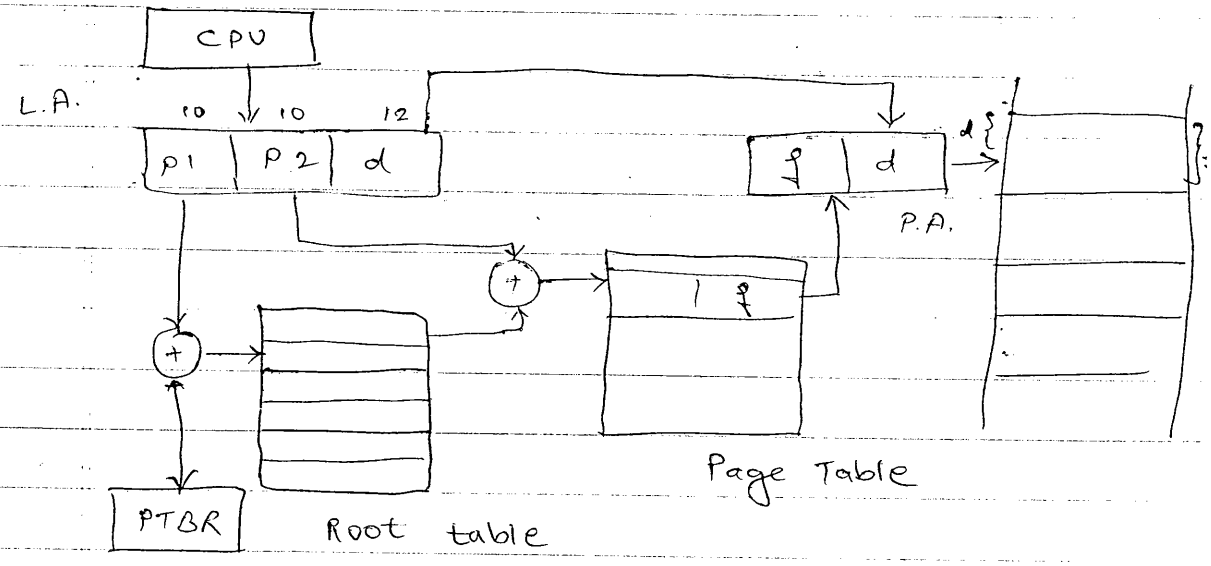
In this we page the page table i.e page table is ÷ into no. of pages.

Now each process need to maintain PTBR & ~~root~~ table. (page table base register).

Steps for performing Logic add. to physical address translation

1) Using PTBR & P1 we get the access in root table
2) Entry in root table + P2 will indicate the page required from the page table.
3) The entry in the page table will give the frame no. which when combined with the displacement would result into the PA.

2 level paging

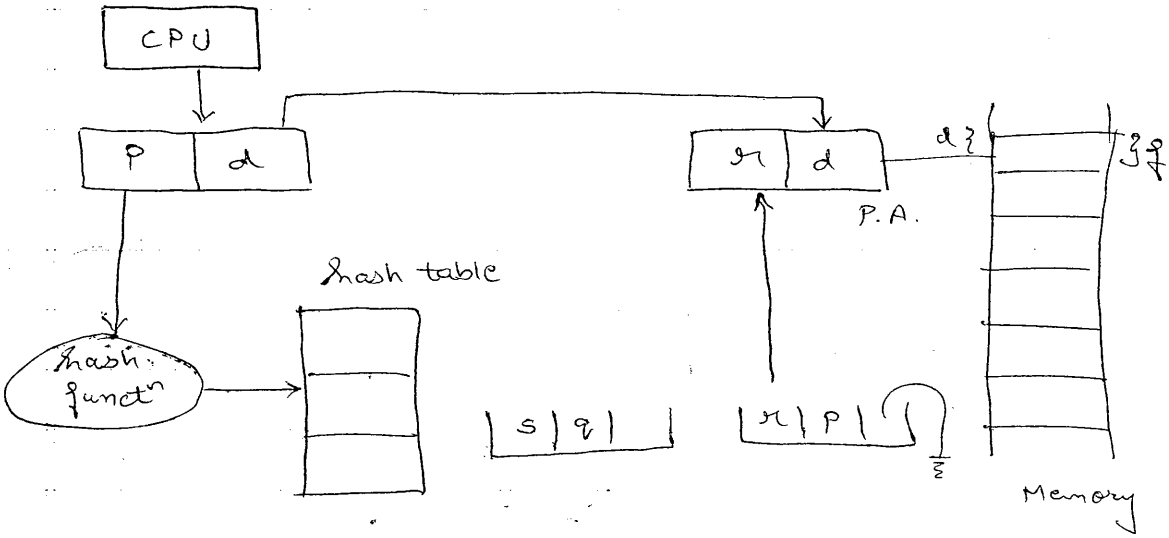


Hashed Page Table :

In this the page table is maintained using the concepts of hashing.

Each entry (node) contains 3 elements

1.) Frame No.
2.) Page No.
3.) Pointer to the nesct node

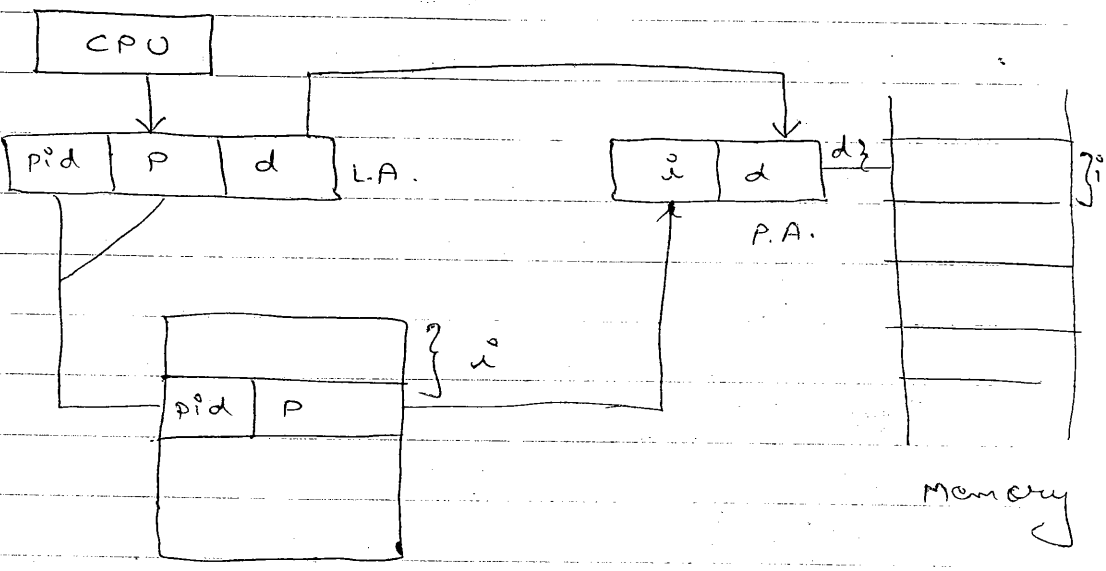


Inverted Page Table :

In this instead of maintaining a page table for each process a single page table is maintained for all the process.

The no. of entries in the page table = no. of frames.

The drawback of this approach is the search time is very high because the search is made page wise whereas the page table is maintained frame wise.



To intensify the search TLB can be used along with the concepts of Hashing.

Segmentation

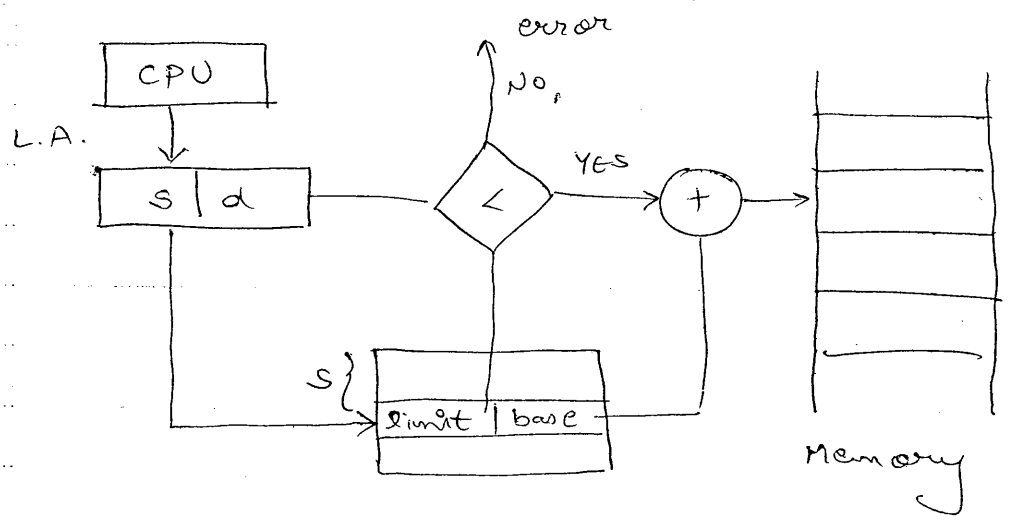
1.) Segmentation is a memory management technique which allows the allocation to be noncontiguous & supports the users view of memory.

2.) Each process needs to maintain a segment table.

3.) Segmentation suffers from external fragmentation.

4.) Segmentation does not limit the no. of active process in the system

5.) Programmer need not be aware about the size of the segments.



Paging	Segmentation
1.> Paging does not support users view of memory.	1.> Segm. supports users view of memory.
2.> Paging is a non-contiguous version of partitioning.	2.> Segmentation is a non-contiguous version of partitioning.
3.> Each process needs to maintain pagetable.	3.> Each process needs to maintain segment table.
4.> The size of page is fixed	4.> The size of the segment can vary.
5.> Paging suffers from internal fragmentation.	5.> segmentation suffers from external fragmentation.
6.) In paging the algo used as FIFO, LRU etc.	6.> In segmentation algo used as first fit, best fit etc
7.> Address Translation	7.) Address Translation

Internal fragmentation	External fragmentation
1> It is a memory area not usable within the frame	1.> It is the memory area not usable outside the segment.
2.> Paging suffers from internal fragmentation	2.> Segmentation suffers from external frag.
3.> The solution to internal fragmentation is to reduce the size of the page.	3.> The soln to external fragmentation is compaction
4.> The drawback of the above soln is (i) size of page ↓ (ii) No. of pages ↑ (iii) No. of page table entries (PTE) ↑ (iv) size of page table increases.	4.> The drawback of above soln is it is very much time consuming.