16 January
Friday

2015

Week 3 3rd Day

Important

8.00

9.00

10.00

11.00

12.00

1.00

2.00

3.00

4.00

5.00

6.00

O.S.

90 lect ✓

500 ques

66

JANUARY 2015

Monday		5	12	19	26	
Tuesday		6	13	20	27	
Wednesday		7	14	21	28	
Thursday	1	8	15	22	29	
Friday	2	9	16	23	30	
Saturday	3	10	17	24	31	
Sunday	4	11	18	25		

Multiprogram multitasking

Operating System And Its functions

It is a system software that works as an interface b/w user and hardware

User
↕
Applications ↕ OS
↕
Hardware
CPU | I/O Devices | RAM

We will have to write programs for every interaction b/w user and hardware in absence of OS

If one person is accesing the hardware, no one else can do it due to absence of regulator.

Examples
Windows

Primary Goal
Convinience for user to hardware interaction (Windows)
Throughput No. of tasks executed per unit time (Linux)

Functionalities
1) Resource management : when multiuser exists ie parallel process.
2) Process management : when many processes run together
3) Storage management : How to store permanent data on hard disk
4) Memory management : RAM managing allocation and delocation of the RAM by processes
5) Security : Windows uses Kerberos security protocol which uses password to provide safe access.

22 January Thursday 2015

Important

2 Week 4th Day 22

It provides security b/w processes also to avoid interfere

When we directly/indirectly access the OS using command prompt or terminal, it is system call.

Types of OS :-

1. Batch : A batch of similar kind of jobs is made and given to the computer for execution. Earlier people used punch cards, paper tapes, map tapes to load the jobs offline and take them to system for execution.

Operator → Batches (B_1, B_2, B_3)

2 3 4

↓

CPU

↳ I/O

CPU sits idle till one job does I/O processes.

Non preemptive.

Fortran and IBSYS 709X were made in 196 by IBM for directly punching jobs into system.

2. Multiprogrammed
3. Multi tasking
4. Real time OS
5. Distributed
6. Clustered
7. Embedded

JANUARY 2015

Monday		5	12	19	26
Tuesday		6	13	20	27
Wednesday		7	14	21	28
Thursday	1	8	15	22	29
Friday	2	9	16	23	30
Saturday	3	10	17	24	31
Sunday	4	11	18	25	

Multiprogrammed OS :- Idleness is also less.
We take more and more processes inside RAM.

8.00 Non premptive :- CPU executes one process completely then it goes to the next. Unless process wants to go for I/O itself,
9.00 the CPU will complete it first and then go ahead.

10.00 Multi tasking OS / Time Sharing.
It is premptive : ie we have decided we will be running
11.00 a process for fixed time. If it gets executed fully in that much time then well and good, else we will
12.00 schedule it later in future.
Adv :- Response time increases.
1.00 Idleness also decreases
We use this on our systems.
2.00

Real Time OS :- There can be no delays. Time matters
3.00 a lot. Hard and Soft Real Time OS.
missile launch. Gaming
4.00

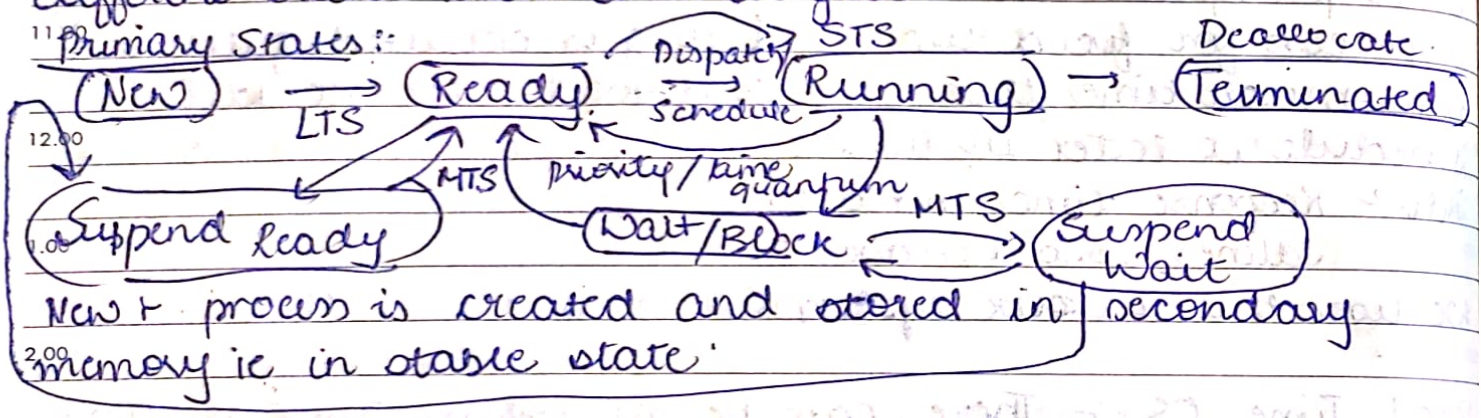
Distributed :
5.00 The machines are lging in loosely coupled environment.
Every machine has its own environment and resources
6.00 and are geographically separated. If one system fails, other system can do its work.

Clustered :-
Connected through local network and act as one server
Computation and processing power increases
Scalability can be increased

Embedded
Work on a fixed functionality. It cannot change.
ie Washing machine, oven

Process states :-
If we start a process on our system, what are the different states that it undergoes.

Primary States :



New : process is created and stored in secondary memory ie in stable state.

Ready : after poking process, it comes in active state in RAM. [Ready Queue]

Long Term Scheduler : Tries to bring max. processes in Ready state ie multiprograming.

Running :- Process gets CPU to execute process. Uniprocessor system has 1 CPU. Multiprocessor or grid computing has multiple CPUs. so they can use parallel processing.

Terminate : when process enters ready state, it gets an address when process leaves all given resources are taken back.

JANUARY

Monday		5	12	19	26
Tuesday		6	13	20	27
Wednesday		7	14	21	28
Thursday	1	8	15	22	29
Friday	2	9	16	23	30
Saturday	3	10	17	24	31
Sunday	4	11	18	25	

Important

Multitasking : At one time multiple processes are running.

8.00 If a high priority process comes to CPU, it executes that first and returns old process to ready state.

9.00 Time Quantum : CPU runs a process for fixed time, if that time expires, CPU gives it back to ready state.

10.00

Short Term Scheduler

11.00 It picks one process from ready state and gives it to running state. If CPU runs the process fully and

12.00 terminates it then non premptive.

1.00 Wait / Block state : Sometimes process has I/O request. This cannot be fulfilled by CPU and needs secondary

2.00 memory which is slow, so we put it in wait state. This state is in RAM only.

3.00

Additional State :

Suspend Wait : Sometimes all processes has I/O request

4.00 so the wait queue gets filled. Thus we swap out few processes and send them to suspend wait in

5.00 secondary memory itself

6.00 Medium Term Scheduler.

If the RAM is filling up so we send few processes to secondary memory.

Backing Store :

If process cant enter wait state, it will go to ready state for resuming.

FEBRUARY 2015

Monday	2	9	16	23
Tuesday	3	10	17	24
Wednesday	4	11	18	25
Thursday	5	12	19	26
Friday	6	13	20	27
Saturday	7	14	21	28
Sunday	1	8	15	22

FEB MAR APR MAY JUN

PS command in linux shows process ID, process details
Task manager does the same in windows

Question :-
Which command is used to assign only read permission to all three categories of file note :-
chmod ← change mode

		r w x	r w -	r w -	
		u	g	o	
u	user	r	read	4	r 4 2
g	group	w	write	2	w 2
o	others	x	execute	1	x 1

Ans :- chmod ugo=r note

Also, u+r ⇒ user is given read permit
u-r ⇒ user is denied of read permit.

Question :-
Which command is used to assign only read permission
'chmod ugo+rw note' command can be represented in actual notation as :-
Ans. chmod 666 note 6 :: 4+2 ie r+w
u g o

Question :-
Suppose you have a file 'f1' whose contents are
1234567890abcdefghij
here lseek is used two times sequentially.
lseek (n, 10, SEEK_CUR);

JANUARY
Monday 5 12 19 26
Tuesday 6 13 20 27
Wednesday 7 14 21 28
Thursday 1 8 15 22 29
Friday 2 9 16 23 30
Saturday 3 10 17 24 31
Sunday 4 11 18 25

lseek(n, 5, SEEK_SET); n is file descriptor.
After applying lseek 2 times what will be current position of R/W head?

Main system calls: lseek(), read(), write(), fork()

If we want to move read/write head and randomly access data, we use lseek(). By default, it is at 1st position.

Ans: (1) lseek on 10th location from current.
(2) seek_set sets r/w head to 5th position from start.
∴ 5th location ie 5
Seek_end sets position from end.

System Call :-
A programatic way of going from user to kernel mode or access functionalities of kernel.

1 File Related : Open(), read(), write(), close(), create file etc
2 Device Related: read, write, reposition, ioctl (input/output), fctrl (function) (file control)
3 Information : get pid, attributes, get system time and data
4 Process Control: load, execute, abort, fork, wait, signal, allocate etc
5 Communication : pipe(), create/delete connections, shmget().

1. When we need to access a file, program becomes process and is sent to RAM where kernel also exists. Process doesn't have privilege of file so it invokes system call to kernel for accessing file.

2. If any hardware is to be accessed, the privilege is needed using system call.

FEBRUARY 2015

Monday	2	9	16	23	
Tuesday	3	10	17	24	
Wednesday	4	11	18	25	
Thursday	5	12	19	26	
Friday	6	13	20	27	
Saturday	7	14	21	28	
Sunday	1	8	15	22	

28 January 2015
Wednesday
Week 5th Day 2

Important

3. Info. about processes or devices. Get pid & process id
ppid : parent id. It helps to get metadata ie the attributes (size etc).

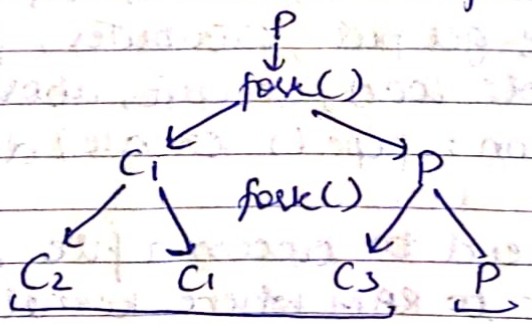
4. map. data is stored on secondary memory but when we have to load them for executing in main memory/ RAM, process control calls are used.

5. Interprocess comm. ie comm. b/w various processes.

Fork() :-
It is used to create child process. It is clone of parent but has different id. [port is shared].
But in thread, we create an extra part whereas all other parts remain same.

Fork generates values
- 0 child made
- +1 +ve parent
- -1 child not formed.

```
main(){
fork();
fork();
printf("hello")
}
```



∴ 4 times hello printed — 3 children — 1 parent

Child process generated : $2^n - 1$
printed hello : 2^n
n = no. of fork done

JANUARY 2015

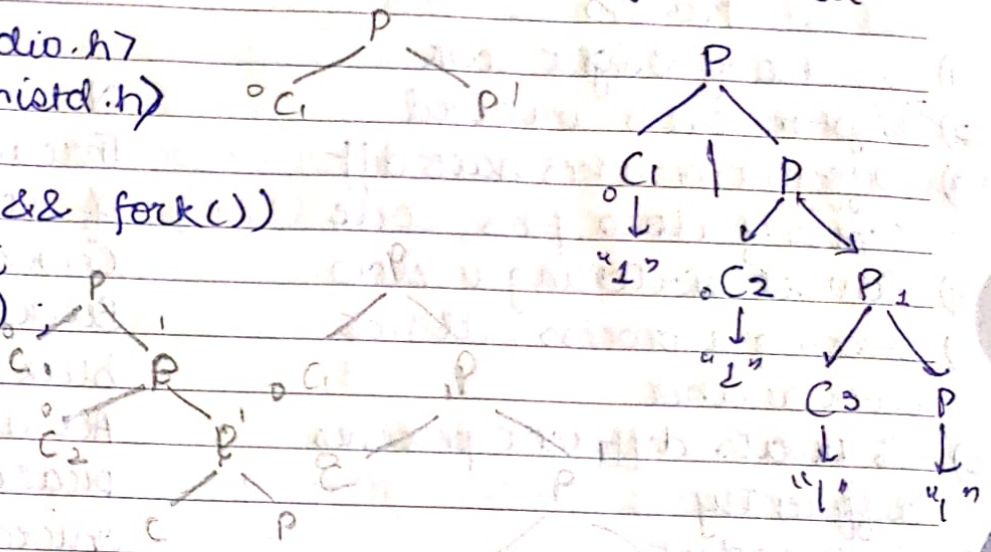
Monday		5	12	19	26
Tuesday		6	13	20	27
Wednesday		7	14	21	28
Thursday	1	8	15	22	29
Friday	2	9	16	23	30
Saturday	3	10	17	24	31
Sunday	4	11	18	25	

Question :-

```
# include <stdio.h>
# include <unistd.h>
int main() {
    if (fork() && fork())
        fork();
    printf("1");
    return 0;
}
```

∴ Ans (1111)

* if first value is 0, just break.

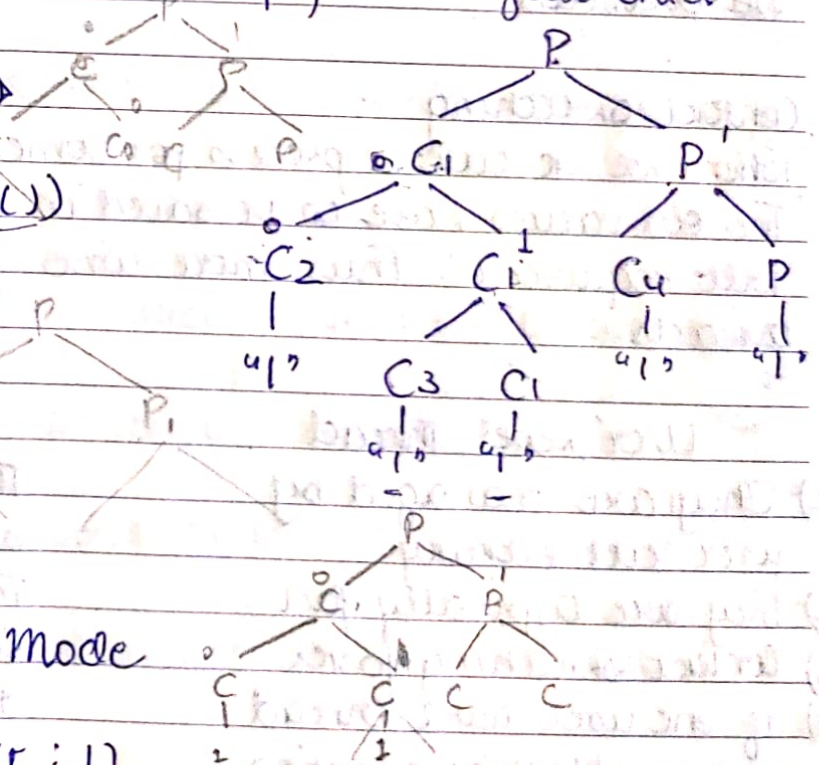


Question :-

```
# include <stdio.h>
# include <unistd.h>
int main() {
    if (fork() || fork())
        fork();
    printf("1");
    return 0;
}
```

Ans (1111)

* if 1st value of fork is 1 just enter



User Mode V/s Kernel Mode

User Mode (Mode Bit : 1)

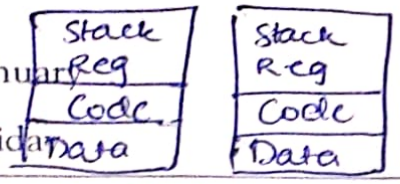
(API) User process executing → Get System call → Trap

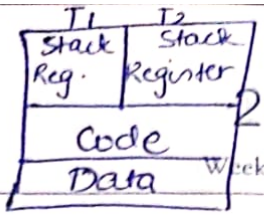
Kernel mode (Mode Bit : 0) → Execute System Call → Return from system call

Hard Disk

30 January Friday 2015

Important



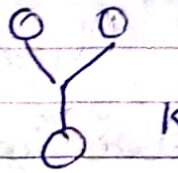


Eg fork()

Process	Threads
1) Heavy weight task	Light weight task
2) System calls involved	No system call is involved
3) Different processes have diff copies of data, files, code	Threads share same copy of code and data.
4) Context switching is slow.	Context switching is faster.
5) Blocking a process will not block another	Blocking a thread will block entire process.
6) OS treats different processes differently	All user level threads are treated as single task for OS
7) Independent	Interdependent
8) ~~##~~ Overhead.	No overhead.

Context switching :-
When we execute a process for some time and then switch it. The old values have to be saved in process control block. This will require OS thus more time for processes lesser for the threads.

User level Thread	Kernel Level Thread
1) They are managed by user level library.	They are managed by OS system calls.
2) They are typically fast.	They are slower.
3) Context switching faster	Context switching slower.
4) If one user level thread performs blocking operation then entire process blocked.	If one kernel level thread blocked, no effect on others.



many to one mapping.

Important

Scheduling Algorithm

Premptive	Non premptive
SRTF (Shortest Remaining time first)	FCFS (First come first serve)
LRTF (Longest Remaining time first)	SJF (shortest job first
Round Robin	LJF (Longest Job first)
Priority Based	HRRN (Highest response ratio next)
	Multilevel Queue
	Priority Based

It is a way of selecting a process from ready queue and putting it on CPU.

CPU Scheduling :-

Arrival time :- The time at which process enters ready queue.

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

Completion time :- Time at which process completes its execution

Turn around time :- Completion time - arrival Time

Waiting Time :- Turn around time - Burst Time

Response time :- (The time at which process gets CPU first time) - (Arrival Time)

01 February 2015
Sunday

Important

FCFS Scheduling

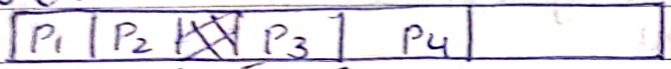
Criteria : Arrival Time
Mode : Non preemptive

In non preemptive cases, response time and waiting time will be same.

Process No	Arrival Time	Burst Time	Complete Time	TAT	WT	RT
P1	0	2	2	2	0	0
P2	1	2	4	3	1	1
P3	5	3	8	3	0	0
P4	6	4	10	6	2	2

Given

Gantt chart :-



* CT - AT = TAT
* TAT - BT = WT

Shortest Job First

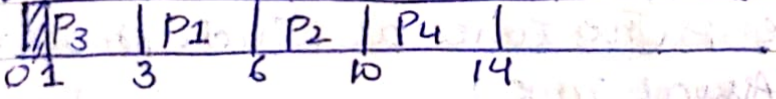
Criteria : Burst time
Mode : Non preemptive

Process No	Arrival Time	Burst Time	Complete Time	TAT	WT	RT
P1	1	3	6	5	2	2
P2	2	4	10	8	4	4
P3	1	2	3	2	0	0
P4	4	4	14	10	6	6

Given

* RT = CPU Time - AT

Gantt Chart :-



Avg TAT = 25/4 = 6.25
Avg WT = 12/4 = 3

When same burst time, check arrival time

2015

Week 6th Day 33rd

February 02 Monday

Important

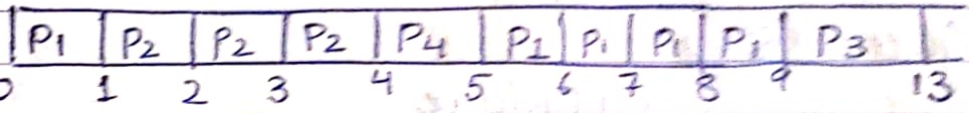
Shortest Remaining Time First :

Criterion : Burst Time

Mode : Preemptive

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	~~5 4 3 2 1~~	9	9	4	0
P_2	1	~~3 2 1~~	4	3	0	0
P_3	2	~~4 3 2 1~~ 0	13	11	7	7
P_4	4	~~1~~	5	1	0	0

Gantt chart



Avg TAT = 24/4 = 6

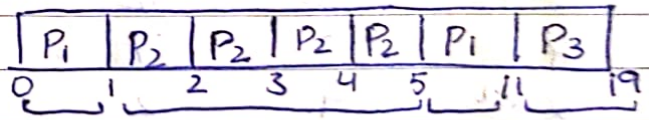
Avg WT = 11/4 = 2.75

Avg RT = 7/4 = 1.75

Question :-

Process	Arrival Time	Burst Time
P_1	0	7 6
P_2	1	~~4 3 2 1~~ 0
P_3	2	8

The Gantt chart for Preemptive SJF (SRJF) Scheduling.



Running Queue to ready queue and vice versa is called context switching.

03 February 2015
Tuesday
Important

14 Week 6th Day 34

~~Round Robin~~ Criteria Time Quantum mode Preemptive

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	~~5~~ 3 1	12	12	7	0
P_2	1	~~4~~ 2 0	11	10	6	1
P_3	2	~~2~~ 0	6	4	2	2
P_4	4	~~1~~ 0	9	5	4	4

Given

Given TQ = 2

Ready Queue

~~P_1~~	~~P_2~~	~~P_3~~	~~P_1~~	~~P_4~~	~~P_2~~	P_1

Running Queue

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0 2 4 6 8 9 11 12

Context Switch

6 times context switch

~~Priority~~ Scheduling Criteria Priority Mode Preemptive

Priority	Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
10	P_1	0	~~5~~ 4	12	12	7	0
20	P_2	1	~~4~~ 3	8	7	3	1
30	P_3	2	~~2~~ 0	4	2	0	2
40	P_4	4	~~1~~ 0	5	1	0	4

Here, higher no higher priority

Gantt Chart

P_1	P_2	P_3	~~P_3~~	P_4	P_2	P_1

0 1 2 3 4 5 8 12

FEBRUARY 2015

Monday	2	9	16	23
Tuesday	3	10	17	24
Wednesday	4	11	18	25
Thursday	5	12	19	26
Friday	6	13	20	27
Saturday	7	14	21	28
Sunday	1	8	15	22

2015

February 04 15
Wednesday

Week 6th Day 35th

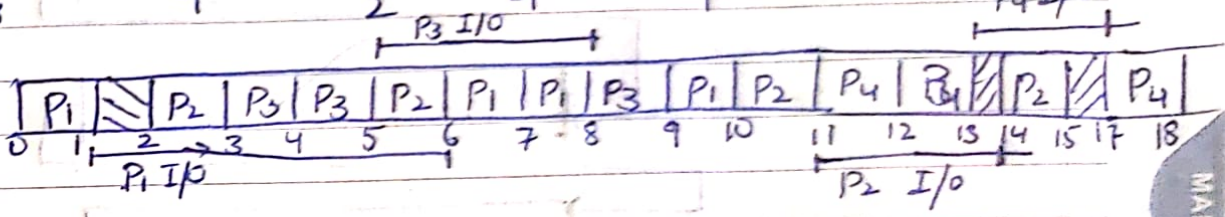
Important

Criteria Priority
mode Premptie

Burst Time (CPU and I/O) :-

Lowest no. highest priority

Process	AT	Priority	CPU	I/O	CPU	Completion Time
P_1	0	2	1	5	3	10
P_2	2	3	3	3	1	15
P_3	3	1	2	3	1	9
P_4	3	4	2	4	1	18

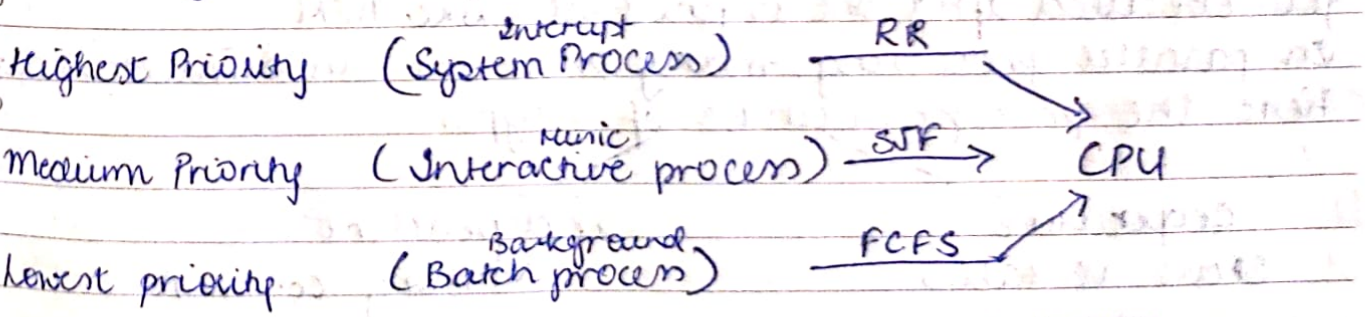


Ratio of CPU idleness = 4/18

CPU usage = 14/18

Multilevel Queue Scheduling

It says that when there are different types of processes, there should be different queues for all of them not just a single ready queue and every process can have its own scheduling algorithm also.



Adv :- Categorisation of processes.

Disadv :- Starvation. High priority will only get CPU maximum

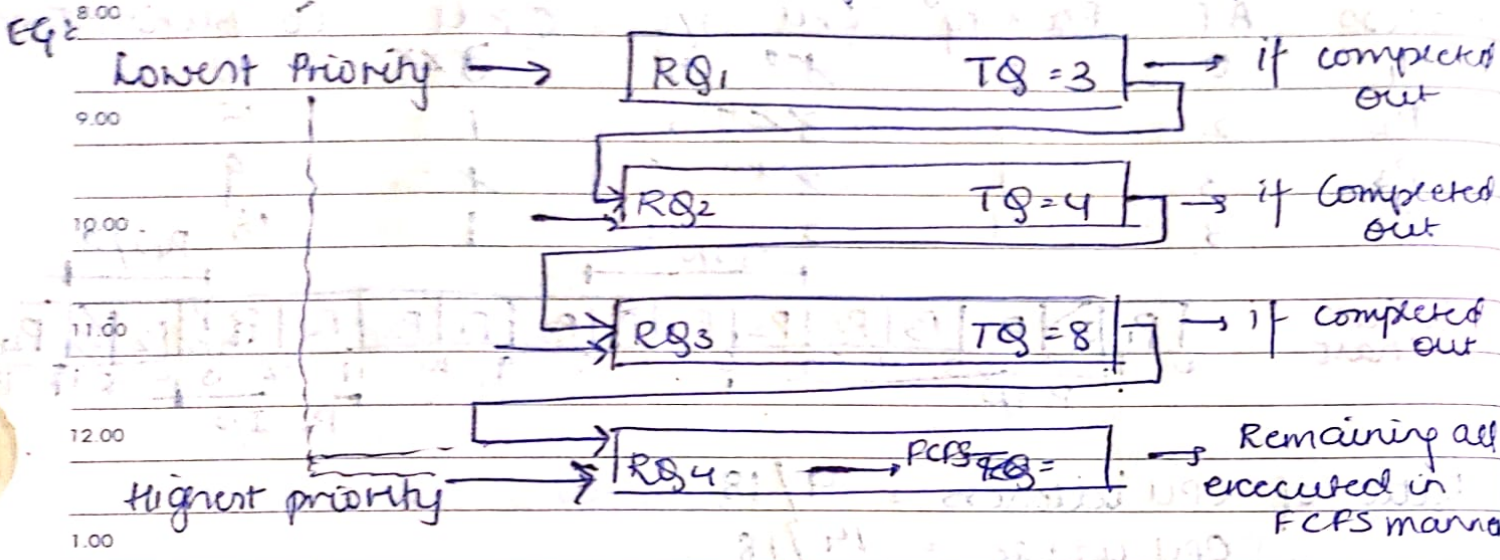
To solve this we can use multilevel feedback queue

05 February 2015
Thursday 16 Week 6th Day

Important

Multilevel feedback Queue

It helps to remove starvation from multilevel queue



Feedback is given by low priority processes only.

Process Synchronisation

When multiple processes run on a system, there can be 2 modes: Series and parallel.

No problem occurs in series because there one process gets executed after one is finished. Like ATM.

In parallel processing, many processes run simultaneously

Here the processes can be of 2 types

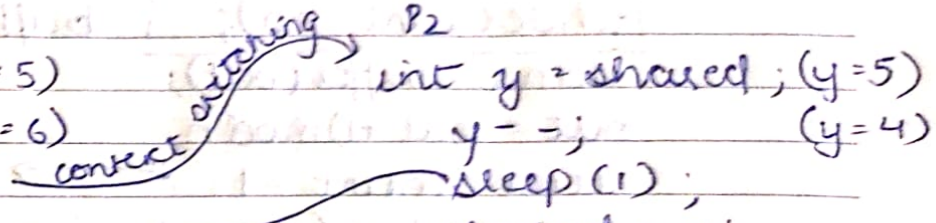
- cooperative: Share variables (memory, Code, Resources — CPU, Printer, Scanner, etc.)
- independent: Nothing common.

Important

Cooperative process : When execution of one affects other. If these process are not synchronized well, they can create problem like deadlock or inconsistent state.

Eg of problem created : int shared = 5

	P1	P2
1)	int x = shared; (x=5)	int y = shared; (y=5)
2)	x++; (x=6)	y--; (y=4)
3)	sleep(1);	sleep(1);
4)	shared = x; (shared=6)	shared = y; (shared=4)
	# Terminated	# terminated.



now finally, shared = 4

but shared should be 5 as x++ and x-- cancel each other.

Such sitation is called race condition because if we execute P1 fist ans is 4 and if we execute P2 first, ans is 6. So 4 and 6 are in a race. This happens because processes are not synchronised in the parallel cooperative process.

Producer Consumer Problem :-

It is a standard problem of multiprocessor synchronisation. Here, we have one producer and one consumer. When producer code runs, it produces an item and puts it in buffer. The consumer executes the code and bring out item from buffer and does whatever process it wants to.

Important

```
void consumer (void)
{ int itemc;
  while(true)
  {
    while(count==0);        - buffer empty
    itemc = buffer(out);
    out = (out+1) mod n;
    count = count - 1;
    process_item(itemc);
  }
}
```

I1 load Rc, m[count]
I2 decrement Rc
I3 store m[count], Rc

```
void producer (void)
{ int itemp;
  while (true)
  {
    produce_item(itemp);
    while (count==n);       buffer full
    Buffer[in] = itemp;
    in = (in+1) mod n;
    count = count + 1;
  }
}
```

I1 load Rp, m[count];
I2 increment Rp;
I3 store m[count], Rp;

Here, we used (in+1) mod n, mod n is used so that once all places are filled it can start from 0th position again.

2015

Week 6th Day 39th

Important

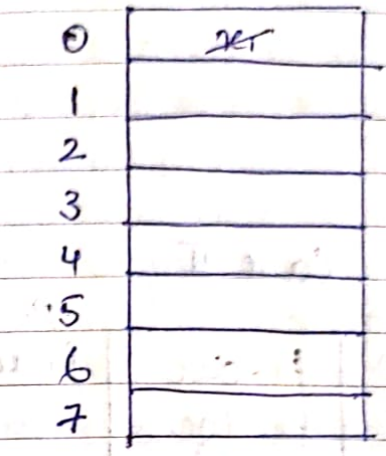
February 08 19

Sunday

Here count is shared.
Buffer is shared.

$n = 8$

Buffer [0 n-1]

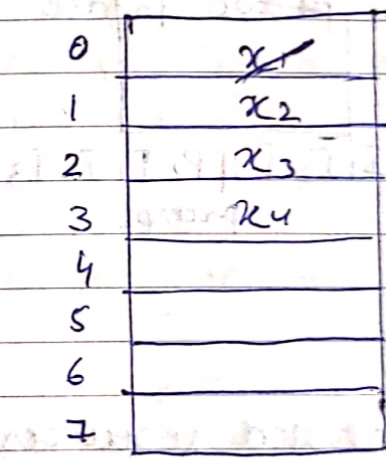


Out = 01

In = 01

Count = 01

Case I :- Synchronized



Out = 01

In = 5 4 (shows next empty slot)

Count = 3 2

Case II :-
Not synch.
Producer executes I_1, I_2 then consumer I_1, I_2, Producer I_3
Consumer I_3

Count gives wrong value. This is the race condition.

Printer - Spooler problem :

We have a network with one printer but many users.
We maintain spooler which is a program that comes into use when multiple users try to print document. They all go in spooler directory. Then spooler takes out those files and gives them to printer one by one

09 February 2015
Monday
Important

I 1. Load Ri, m[in]
I 2. Store SD[Ri], "File name"
I 3. Increment Ri
I 4. Store m[in], Ri

} to send to spooler

Case I

Spooler Directory

0	f1.doc	→ Printer
1		In [0] 1
2		Pi f1.doc
3		Ri [0] 1
4		

Case II

	P1	P2
0	f1.doc	f4.doc
1	f2.doc	f5.doc
2	f3.doc	In [3] 4
3	~~f4.doc~~ f5.doc	R1 [3] 4 R2 [3] 4
4		~ Loss of Data

P1 I1 I2 I3 | P2 I1 I2 I3 I4 | P1 I4
prempt prempt

Critical Section :-
It is part of the program where shared resources are accessed by various processes (cooperative processes).

P1

```
#include ---
main()
---
---
---
```

A, B
Non critical Section

Entry Section

Count++
Critical

Exit Section

P2

```
# include
main()
---
---
---
```

X, Y
Non critical

Entry Section

Count--
Critical section

Exit Section

FEBRUARY 2015
Monday 2 9 16 23
Tuesday 3 10 17 24
Wednesday 4 11 18 25
Thursday 5 12 19 26
Friday 6 13 20 27
Saturday 7 14 21 28
Sunday 1 8 15 22

Important

Since P_1 and P_2 both have count variable. When P_1 is using count and P_2 also executes the critical section then race condition can occur. To avoid that, we have to synchronise.

For this, we have an entry section. So to use critical section process has to execute the entry section first. We have to write such an entry section that only one process passes through it and not all the others.

Later on they have to execute exit section code after critical section also. This stops race condition.

Conditions for Synchronisation:

1) Mutual Exclusion } primary
2) Progress
3) Bounded Wait } secondary
4) No assumption related to H/W speed.

Mutual Exclusion :-

If P_1 is already in critical section using it then P_2 cannot enter the critical section to use it - ie if P_1 successfully enters critical section after running entry section then critical section will be locked then P_2 cannot enter.

Progress :-

If P_1 is interested to enter critical section and P_2 is not by P_2 is blocking P_1's entry then there is no progress. Progress This means in P_2 the entry code of P_2 is hindering the entry of P_1 also.

11 February 2015
Wednesday
Important
22 Week 7th Day 42

Bounded Wait
It means that one progress is in critical section and the other one is waiting outside. Now P_1 comes outside but then enters again and this keeps happening infinitely. So P_2 cannot enter and is stared. This is unbounded wait. Thiss bound wait says all processes should get chance to use critical section.

No assumption related to HW speed
Lets say a sol^n is given for synch problem and it works on 32-bit system not on 64-bit. Thus this is not write right. Or if 1 G-Hz processor will run or less speed will not work. This sol^n is not right. The sol^n should be portable to all OS.

Lock Variable :-
Critical section problem sol^n using "lock"

```
do{
    acquire lock
    CS
    release lock
}
```

1. Executes in user mode
2. Multiprocess sol^n
3. No mutual exclusion Guanteed.

Case I Lock != 0 infinite loop

1. While (Lock == 1);	Entry Code
2. Lock = 1	
3 Critical Section	
4. Lock = 0	Exit Code

Case II — same

1 While (Lock != 0)	
2 Lock = 1	
3 Critical Section	
4 Lock = 0	

P1 preempts

P1 preempts and resumes from step 2.

10.00 Lock = 0 → vacant
1 → full

11.00 Lock = 0 1 0 1

P1	P2
12.00 1 2 3 4	P2

P2 will enter when P1 releases lock and makes it 0.

Lock 0 1 1

P1	P2
1 2	1 2 3

P1 preempts

P2 P1

1.00 P2 and P1 in critical section together

2.00

Test & Set Instruction :

3.00 Test and set atomises / combines the two instructions of entry code so no premption can occur within them.

4.00 Critical section solution using 'Test and set' :-

```
while (test_and_set (& lock));
    CS
lock = false;
```

5.00 P2 will be true and stuck in this infinite loop.

6.00

```
boolean test_and_set ( boolean * target)
{
    boolean r = * target;
    * target = true;
    return r;
}
```

lock	target	r
false	1000	false
true		

P1 true

P2 True

P2 True

13 February 2015
Friday Week 7th Day 44
Important

Turn Variable (Strict Alternation)

1. It is 2 process soln.
2. Runs in user mode
3. It guarantees Mutual exclusion
4. It doesn't allows progress of both processes. ie P_1 can only enter after P_0
5. It allows bounded wait.
6. It is independent of hardware

Proces P_0:
Non Critical
Entry → while (turn != 0);
Critical section
Exit turn = 1;

Process P_1:
Non critical
while (turn != 1);
Critical section
turn = 0;

let int turn = 0
So in this case P_0 runs first and P_1 cannot enter. Then P_2 exits and now P_1 can enter.

Semaphore

It is a tool used to prevent race condition. Race condition can lead to loss of data, deadlock, etc.
Semaphore is an integer variable which is used ~~to prevent~~ in mutual exclusive manner by various concurrent and cooperative process in order to achieve synchronization.

Semaphore
- Counting (>=0)
- Binary (0,1)

In semaphore, we use entry exit operations

P_1 P_2 P_3

Entry code → P(), Down, Wait

CS

Exit code → V(), Up, Signal, Post, Release

COUNTING SEMAPHORE.

Down (Entry)

```
Down (Semaphore S)
{
  S.value = S.value - 1;
  if (S.value < 0)
  {
    put process (PCB) in
    suspend list &
    sleep();
  }
  else
    return ;
}
```

→ process enters in critical section

Up (Exit)

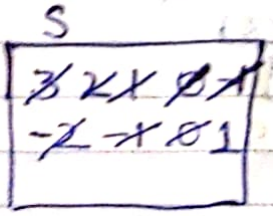
```
Up (Semaphore S)
{
  S.value = S.value + 1;
  if (S.value <= 0)
  {
    Select process
    from suspend list
    & Wake up();
  }
}
```

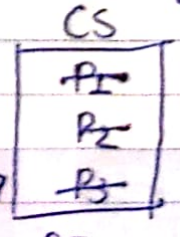
S 10 : 10 processes can enter critical section.

S 0 : No process in suspend list / No more can enter CS

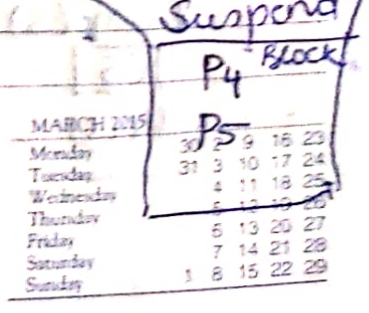
S -4 : 4 processes in suspend list

Let S = 8





After S=1, P4 can enter ready queue



15 February Sunday 2015

26 Week 7th Day

Important

Question :

Find successful operations on S [17] after 5P, 3V, 1P.

17 - 5 + 3 - 1 = 14. ⇒ final value of semaphore.

BINARY SEMAPHORE :-

```
Down(Semaphore S)
{
  if (S.value == 1)
  {
    S.value = 0;
  }
  else
  {
    Block this process
    and place in suspend
    list, sleep();
  }
}
```

```
Up(Semaphore S)
{
  if(Suspend list empty)
  {
    S.value = 1;
  }
  else
  {
    select process from
    suspend list and
    wake up();
  }
}
```

Here it is important to start with S=1 otherwise a deadlock situation is achieved and no process executes.

P1

Down(S)

CS

Up(S)

P2

Down(S)

CS

Up(S)

S 1 0

CS P1

Now when P2 tries to enter CS it cannot as S = 0 and suspended.

FEBRUARY 2015

Monday		2	9	16	23
Tuesday		3	10	17	24
Wednesday		4	11	18	25
Thursday		5	12	19	26
day		6	13	20	27
rday		7	14	21	28
day	1	8	15	22	

Important

Question:- Binary Semaphore

1. Each process P_i {i = 1 to 9} execute following code:

```
repeat
    P(mutex)
    [CS]
    V(mutex)
forever
```

mutex = 1

Process P_{10} executes the following code:

```
repeat
    V(mutex)
    [CS]
    V(mutex)
forever
```

Mutex 1 0 1 0 1 0 1

CS. P_1 P_{10} P_2 P_3 P_{10} P_4 P_{10} P_5 P_{10} P_6 ....

∴ 10 maximum processes can be present in CS at any point of time. $P_1 \to P_{10} \to P_2 \to P_5 \to P_3 \to P_4 \to P_{10} \to P_5 \to P_6$

2.

```
repeat
    P(mutex)
    [CS]
    V(mutex)
forever
```

P_1 P_{10}

```
repeat
    V(mutex)
    [CS]
    P(mutex)
forever
```

Mutex 1 0 1 0

P_1 P_{10} P_2

∴ 3 max processes.

MARCH 2015

Monday	30	2	9	16	23	
Tuesday	31	3	10	17	24	
Wednesday		4	11	18	25	
Thursday		5	12	19	26	
Friday		6	13	20	27	
Saturday		7	14	21	28	
Sunday	1	8	15	22	29	

Important

Solution Of Producer Consumer Problem :
Counting semaphore : — Full = 0 = no. of filled slots
— Empty = N = no. of empty slots

8.00

9.00 Binary Semaphore, S = 1.
Case I : no preemption, full running

10.00 Producer
Produce_item (itemp);
11.00 1) down (Empty);
2) down (S);
12.00 Buffer [in] = itemp
1.00 in = (in + 1) mod n
3) up (S);
2.00 4) up (Full);

0	~~a~~
1	b
2	c
3	d
4	
5	
6	
7	

Consumer
Consumer (itemc)
1) down (full);
2) down (S);
itemc = Buffer [out]
out = (out + 1) mod n
3) up (S);
4) up (Empty);

3.00

in 3 4 out 0 1

4.00 Empty = ~~5~~ 4 5
Full = ~~3~~ 4 3
5.00 S = ~~1~~ 0 ~~1~~ 0 1

6 Case II ; when one process preempts in between (Context Switch)

in 3 4 out 0 1

Empty = 5 ~~4~~ 5
Full = ~~3~~ 3 3
S = ~~1~~ 0 ~~1~~ 0 1

0	~~a~~
1	b
2	c
3	d
4	
5	
6	
7	

Preempts after P[1] and Consumer ... Producer resumes

FEBRUARY 2015

Monday	2	9	16	23
Tuesday	3	10	17	24
Wednesday	4	11	18	25
Thursday	5	12	19	26
Friday	6	13	20	27
Saturday	7	14	21	28
Sunday	1	8	15	22

Important

Reader - Writer Problem

Multiple types of users can use the database at the time ie reader and writer.

The problem is when reader is accessing one data and writer comes for the same data and vice versa. Also, if writer exists and another writer wants to access same data.

R-W problem
W-R problem
W-W problem
} on the same data.
R-R no problem

This can lead to data loss and data inconsistency.

So we need to synchronise these readers and writers. We can use locks, semaphores, monitors etc. We will use semaphore.

rc = reading people count

```
int rc = 0
Semaphore mutex = 1
Semaphore db = 1
void Reader (void)
{ while
  {
    down (mutex);
    rc = rc + 1
    if (rc == 1) then down(db);
    up(mutex);
    DB
    down (mutex)
    rc = rc - 1
    if (rc == 0), then up(db);
    up(mutex)
    Process Data
  }
}
```

Entry Code

Exit

```
void writer(void)
{ while (true)
  {
    down (db);
    DB
    up (db);
  }
}
```

Entry

Exit

19 February Thursday 2015

CASE 1

Reader R_1 comes first

RC [1] 1 Mutex [1] 0 DB [1] 0

Now writer tries to come:-

Writer will get blocked because db is already down

CASE II

Writer W_1 comes first

RC [0] mutex [1] DB [1] 0

Now reader tries to come in before exit of W_1.

RC [1] 1 mutex [1] 0

Reader will be blocked because db is already down.

Case III

Writer W_1 comes first

DB [1] 0

W_2 tries to come

W_2 will be blocked because db already down.

CASE IV

R_1 comes

RC [1] 1 Mutex [1] 0 1 DB [1] 0

R2 tries to come before exit of R_1

R [1] 2 Mutex [1] 0 1

Now,

R_1 wants to leave DB after running code.

RC [1] 1 mutex [1] 0 1 DB [0]

Since DB doesn't go up writer cannot enter.

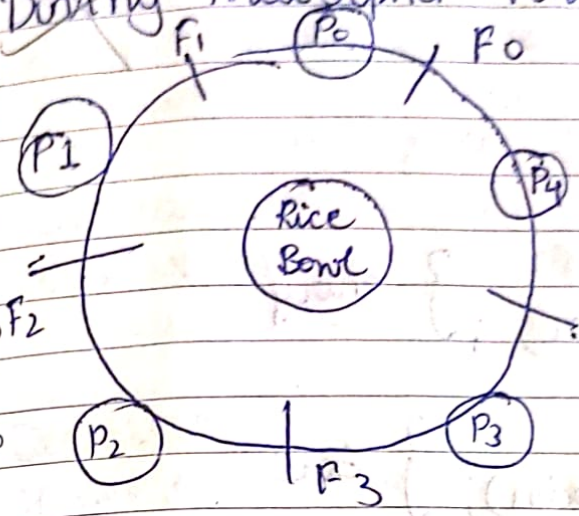
Mutex value is mainly for checking and allowing entry of Readers.

FEBRUARY 2015

Monday	2	9	16	23
Tuesday	3	10	17	24
	4	11	18	25
	5	12	19	26
	6	13	20	27
	7	14	21	28
	1	8	15	22

Important

Dining Philosopher Problem :



N : no of forks.

```
void philosopher (void)
{
  while (true)
  {
    thinking ()
    take_fork (i)          ← left fork
    take_fork ((i+1)%N)    ← right fork
    Eat ();
    put_fork (i);
    put fork ((i+1) % N)
  }
}
```

Case I : Philosophers eat in a serial order

P0 → f0, f1 → P1 → f1, f2.

Case II : When 2 or more philosophers come together, race condition

P0 → f0 | P1 → f1

So to solve this, we will use semaphore ie here we will use semaphores' array equal to number of forks.

S[i] 5 semaphores

S_0, S_1, S_2, S_3, S_4, &
1 1 1 1 1

Now code with binary semaphore :

MARCH 2015

Monday	30	2	9	16	23
Tuesday	31	3	10	17	24
Wednesday		4	11	18	25
Thursday		5	12	19	26
Friday		6	13	20	27
Saturday		7	14	21	28
Sunday	1	8	15	22	29

MAR APR MAY JUN

```
void philosopher(void)
{
  while (true)
  {
    Thinking ()
    Wait ( take_fork(S(i)));
    Wait ( take_fork(S(i+1)%n) ;   } Entry
    (Eat ())
    Signal ( put fork S(i));
    Signal ( put fork S(i+1) mod n);   } Exit
  }
}
```

P_0	S_0	S_1
P_1	S_1	S_2
P_2	S_2	S_3
P_3	S_3	S_4
P_4	S_4	S_0

	S_0	S_1	S_2	S_3	S_4
	1	1	1	1	1
P_0	~~1~~ 0	~~1~~ 0	1	1	1
P_1	already down				
P_2			~~1~~ 0	~~1~~ 0	1

P_0 and P_2 are eating

We had discussed mutual exclusion ie if one process is already in CS then other can't enter but here the independent processes can enter. ie P_0 needs S_0 and S_1 and P_2 needs S_2 and S_3 ∴ semaphores are independent

Lets says all philosophers pick just left fork and then get prempted. Now when these philosophers try to pick right fork then they cannot because of circular wait. This is deadlock situation because all the philosophers can't eat. This cannot be unlocked.

Important

1. This can be treated if we remove one of the philosophers
2. We can reverse the code for last philosopher. ie $P_4 \rightarrow S_0\ S_4$

8.00 This way S_4 will be 1 only. P_3 can pick up S_4 in right hand later it will convert S_3 and S_4 to 1 after eating thus P_2

9.00 will eat with S_2, S_3. then P_1 and P_4 will eat at last
But we can change order of any one philosopher not just last.

10.00

11.00 **Deadlock :-**
If two or more processes are waiting on happening of

12.00 some event, which never happens, then we say these processes are in deadlock

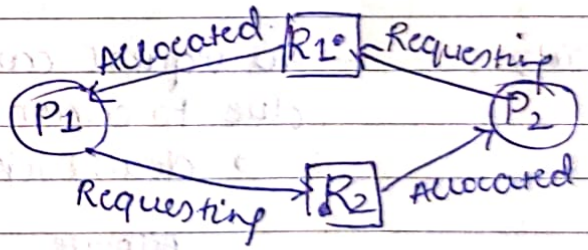
1.00

Conditions for Deadlock :-

2.00 1) Mutual Exclusion :- A resource should not be able to be used by many process together.
2) No premption :- One process prempts when other comes along (not happen)

3.00 3) Hold and wait : Resource will not be released by process
4) Circular wait :- Loop of waiting [dining philosopher left fork only).

4.00



5.00

6.00

MARCH 2015

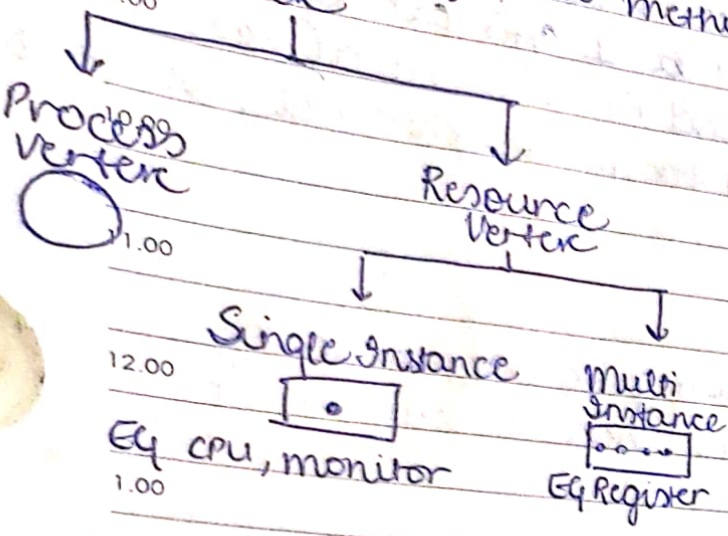
Monday	30	2	9	16	23
Tuesday	31	3	10	17	24
Wednesday		4	11	18	25
Thursday		5	12	19	26
Friday		6	13	20	27
Saturday		7	14	21	28
Sunday	1	8	15	22	29

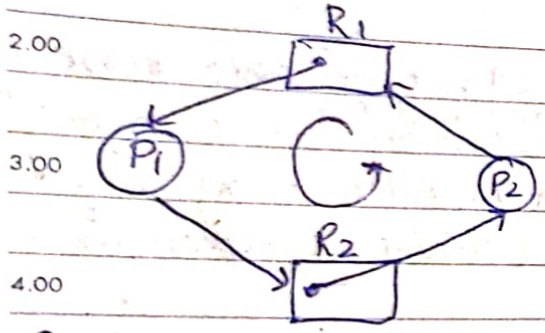
MAR APR MAY JUN

Resource Allocation Graph:

Whether system has deadlock or not can be quickly found using this method:

- Vertex
  - Process Vertex
  - Resource Vertex
    - Single Instance
    - Multi Instance
- Edge
  - Assign Edge
  - Request Edge





Circular wait, Hold & Wait, No preemp:, ME.

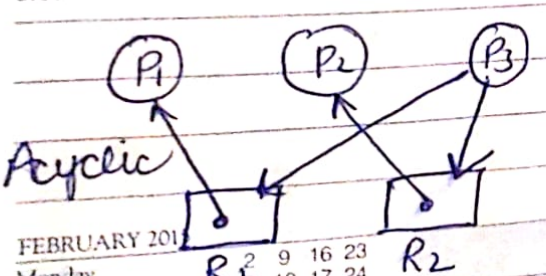
∴ Deadlock

	R1 R2 Allocate	R1 R2 Request
P1	1 0	0 1
P2	0 1	1 0

Availability (0,0) R1 R2

∴ No request can be fulfilled due to current avail.

∴ ⇒ deadlock.



Acyclic

	Allocate R1	Allocate R2	Request R1	Request R2
P1	1	0	0	0
P2	0	1	0	0
P3	0	0	1	1

Availability (0,0) R1 R2

P1, P2 fulfilled. ∴ P1 terminated

∴ new Avail (1,0) Now P2 run & terminate

∴ new Avail (1,1) ∴ P3 fulfilled

∴ no deadlock

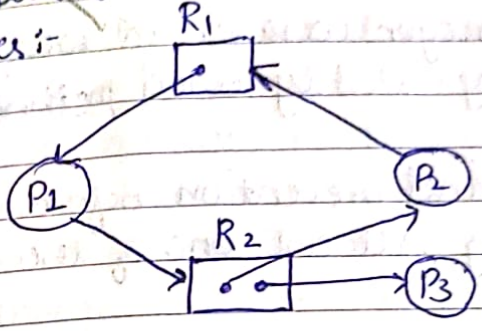
Waiting

Finite (Starvation) (∞) Infinite (Deadlock)

If RAG has circular wait (cycle) then it is always deadlock in case of single instance and vice versa.

Multi Instance Resource Allocation Graph :-

Ques :-



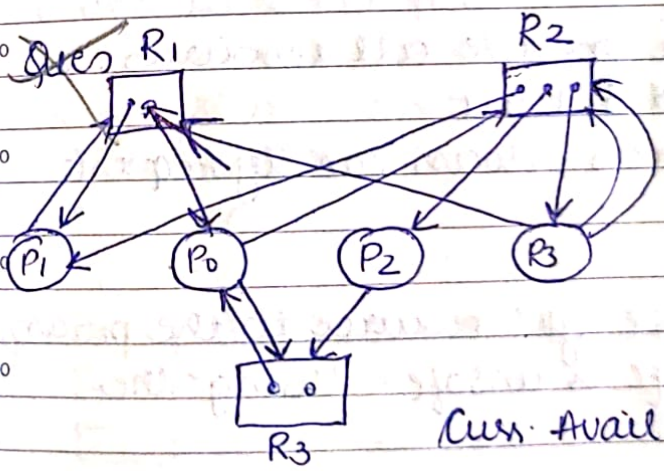
	Allocate R1	R2	Request R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

	R1	R2
Availability	0	0
P3 executes ⇒	0	1
Now P1 executes ⇒	1	1

⇒ P2 also executes as it requests (1,0)

* Single Instance and Circular Wait ⇒ Deadlock (Always)
Multi Instance and circular wait ⇒ Deadlock (not Always).

Ques



	Allocate R1	R2	R3	Request R1	R2	R3
P0	1	0	1	0	1	1
P1	1	1	0	1	0	0
P2	0	1	0	0	0	1
P3	0	1	0	1	2	0

Curr. Avail. :-

R1	R2	R3	
0	0	1	
+ 0	1	0	
0	1	1	
+ 1	0	1	
1	1	2	
+ 1	1	0	
2	2	2	
+ 0	1	0	
2	3	2	⇒ final after all process

$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

∴ no deadlock

Q→ Check for deadlock.

Various methods to handle deadlocks:

1) Deadlock ignorance (Ostrich method): Window / Linux mostly use this because they think it occurs rarely and writing huge code can affect performance speed of system.
Advantage: OS keeps doing its major tasks and we do not increase its functionality. Widely used method.

2) Deadlock prevention: We try to find prevention before any deadlock occurs by removing atleast one of the four conditions of deadlock.
a) To make mutual exclusion false we make resources shareable by multiple processes. But it is not possible for all resources like printer, tape drives
b) We can try to make processes preempt by using time quantum
c) Before starting of process we provide all resources to the process to avoid hold & wait
d) Give numbering to all resources. Process can only request in increasing order.

3) Deadlock Avoidance :- When we give resource to the process it will check first if its safe or unsafe. Using the Banker's Algorithm.

4) Deadlock Detection & Recovery: First we detect using resource allocation graph and then we try to recover it. This is complex for the system. We try to recover by killing the processes in deadlock or by the preemption of low priority processes.

Important

Banker's Algorithm : Deadlock Avoidance Algo.
It is also used for deadlock detection.

Ques :-

Total A = 10 ; B = 5 ; C = 7

This process is deadlock avoidance because we tell the system beforehand how many resources are needed by various processes.

Safe means no deadlock can occur. We have safe seq. which tells in what way should we run the processes to avoid deadlock.

Unsafe means deadlock will occur.

If we cannot fulfil need of all processes then that means there is a deadlock.

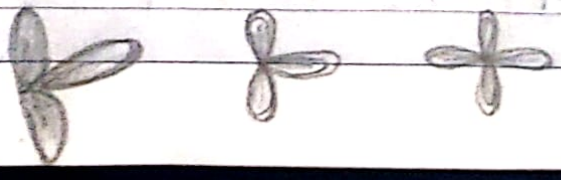
In real life this algo is not possible because processes do not have static needs, they have dynamic needs which change during runtime only.

Process	Allocation			Max Need			Available			Rem. Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	3	3	2	7	4	3
P_2	2	0	0	3	2	2	5	3	2	1	2	2
P_3	3	0	2	9	0	2	7	4	3	6	0	0
P_4	2	1	1	4	2	2	7	4	5	2	1	1
P_5	0	0	2	5	3	3	7	5	5	5	3	1

Given 2 5

Safe Sequence : $P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

5 3 2



Ques:- Solve using Banker's Algo.

Process	Allocation			Max			Current Available			Max Need Alloc. Rem Need		
	E	F	G	E	F	G	E	F	G			
P_0	1	0	1	4	3	1	3	3	0	3	3	0
P_1	1	1	2	2	1	4	4	3	1	1	0	2
P_2	1	0	3	1	3	3	5	3	2	0	3	0
P_3	2	0	0	5	4	1	6	4	6	3	4	1
	5	1	6				+ 2	0	0			
+	3	3	0				8	4	6			
Total	8	4	6									

Safe Seq :- $P_0 \rightarrow P_2 \rightarrow P_3 \rightarrow P_1$

Safe state and no deadlock.

Ques:-
A system is having 3 processes each require 2 unit of resources 'R'. The minimum no. of units of 'R' such that no deadlock will occur :-

For resources = 4, deadlock won't occur.

P_1	P_2	P_3
1	1	1

∴ Ans = min req + 1. min req = total req. − 1

What is max. resources allocated by still deadlock?

Ans = 3.

Ques:-

P_1	P_2	P_3
3	4	5
−1	−1	−1
2	3	4

$+1 = 9+1 = 10$

Ques:-
Consider a system with 3 processes that share 4 instances of same resource type. Each process can request a max of 'k' instances. The largest value of k that will always avoid deadlock is ____?

Sol:- 'R' resources
'n' processes

P_1 P_2 P_3 P_4 ---- P_n process
d_1 d_2 d_3 d_4 ---- d_n demand of resources

Max resources but still deadlock:-
$(d_1-1)+(d_2-1)+(d_3-1)+\cdots(d_n-1)$
$R \leq \sum_{i=1}^{n} d_i - n$ } max till which deadlock exists
$R > \sum_{i=1}^{n} d_i - n$ } deadlock free
$R+n > \sum_{i=1}^{n}$

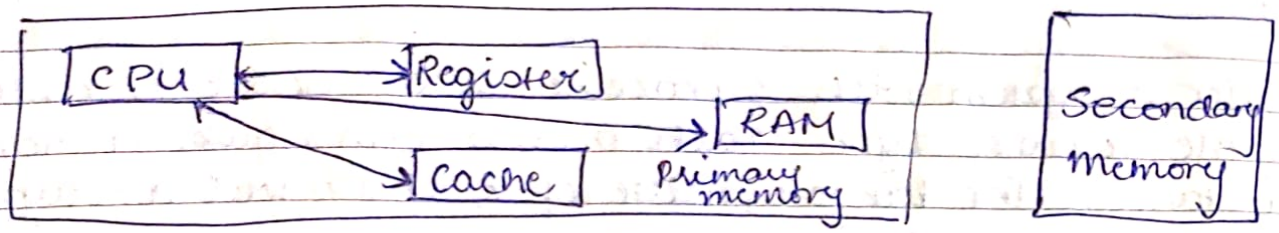
$\therefore 4+3 > 3k$ $\therefore k = 1, 2$ can be answers
2 is max.
$\therefore$ correct ans = 2.

* Total Resources + Total processes > Total demand

Memory management
Method of maintaining majorly primary memory.
Goal:- Efficient utilisation of memory.

CPU

01 March 2015
Sunday
Important
40 Week 9th Day



8.00
9.00

Since secondary memory is slow, it cannot directly interact with CPU. Thus, all programs from there are brought to RAM so that CPU can interact. We can use registers and cache also but there size is too small yet fast

10.00
11.00
12.00

Multiprogramming :-
When many programs are brought from SM to RAM so that degree of multiprog. increases and utilisation of CPU is high.
If process → CPU
I/O Request
If just one program running then

1.00
2.00

when it goes for I/O, then CPU sits idle. Its not good.

3.00

Numerical

4.00

1) RAM (4MB) Process (4MB)
No. of process in RAM = $\frac{4MB}{4MB} = 1$.

5.00

'K' is time for I/O operation. (Lets say 70%.)

6.00 CPU time = $(1-K)$. = 30%.

2) RAM (8MB) Process (4MB)
no. of processes = $\frac{8}{4} = 2$

MARCH 2015
Monday
Tuesday
Wed

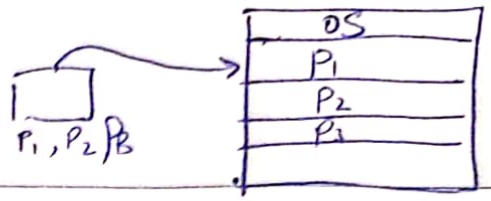
30	2	9	16	23
3	10	17	24	
4	11	18	25	
5	12	19	26	
6	13	20	27	
7	14	21	28	
8	15	22	29	

Time of I/O = K^2 (70%.)
CPU = $(1-K^2)$ = $1-(.7)^2$ = 76%.

2015

Week 10th Day 61st

March 02 Monday 41



Important RAM Process

16MB 4MB No of processes = 4

Time of I/O = K^4

8.00 CPU = $1-K^4$ < 93%.

When size of RAM increases, no. of processes increases then 9.00 CPU utilisation increases.

Management of RAM is the most important.

10.00 When more no. of process enter RAM, OS has to perform lots of allocation and deallocation

11.00 Here, security also comes into play because when more processes enter into RAM, they can interfere with each others' 12.00 performance.

1.00 Memory Management Techniques

2.00 Contagious — Non contagious

3 Fixed Partition [Static] — Variable partition [Dynamic] — Paging — Inverted Paging — Multilevel Paging — Segmentation — Segmented Paging

4.

We keep degree of multiprogramming high so that whenever 5.00 CPU is about to get idle due to processes going for I/O, a new process comes into running state. Thus, more ready 6.00 state elements are good.

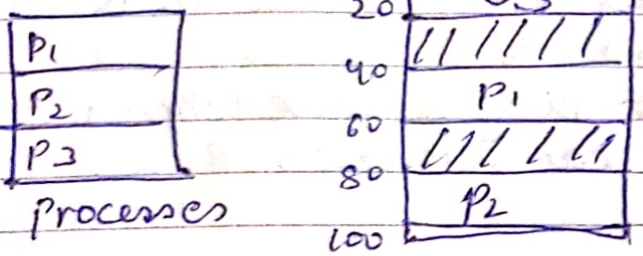
Memory is large array of bytes and every byte has an address.

Contiguous :- We provide continous memory to the processes. Either we can divide the memory into fixed sizes (static) or we can allocate memory during runtime dynamically (variable)

Fixed partitioning was used at the time of main frames when there were huge sewers.

Non contiguous

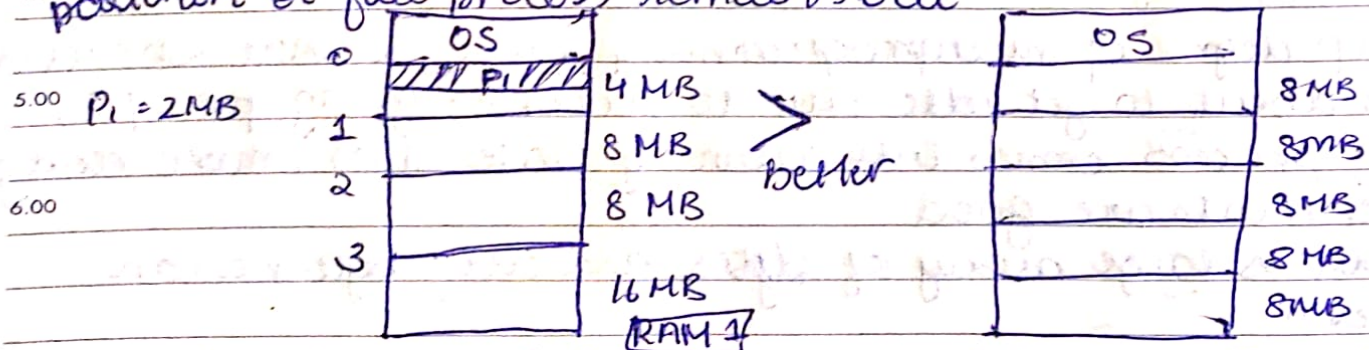
Non continous allocation of memory to the process First we divide the process into parts.



} non contiguous.

Fixed Partitioning (Static Partition)

1. No. of partitions of RAM are fixed.
2. Size of each partition may or maynot same.
3. Contiguous allocation only so spanning is not allowed ie either full process goes in one memory partition or full process remains out.



Here 2MB is occupied but full partition is given to P1. Hence the remaining 2MB is waste. This is called the internal fragmentation.

Important

Disadvantages of Fixed partitioning

1. Internal fragmentation - , external
2. Limit in process size ie a process of 32 MB cannot enter RAM
3. Limitation on degree of multiprogramming because the no. of partitions are fixed in RAM.

External fragmentation :- Although we have availability of memory in diff slots and combination is equivalent / greater than incoming process size, but still we cannot accomodate due to contiguous allocation

* Whenever there is internal frag. in memory, there is always external frag. also.

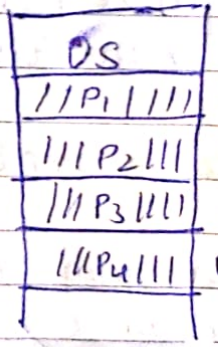
Advantages :

Easy to implement as slots are already decided : allocate and deallocate is easy

Variable Partitioning or Dynamic Partitioning

Partitions are made for the process as they come in RAM and the entire space is empty initially

$P_1 = 2MB$, $P_2 = 4MB$, $P_3 = 8MB$, $P_4 = 4MB$



1) No internal fragmentation
2) No limitation on no of processes
3) No limitation on degree of multiprog.
4) No limitation on process size

When a process leaves ie P_1 and P_4 holes are created. Now if $P_6 = 8$ MB arrives then it cannot be accomodated due to external fragmentation.

Important

Disadvantages

1. External fragmentation in variable partitioning can be removed using compaction
2. Compaction : We take empty spaces to one side and allocated memory to one side. This is like copy / paste But this is undesirable as we will have to stop the running processes and copy / paste takes lots of time.
3. Allocation / Deallocation is not easy as holes will be more and alloc. / dealloc. will be complex as compared to fixed partitioning. We use bitmaps and linked lists for alloc / dealloc of holes and filled memories

Memory Allocation Algorithms { Contiguous Memory Allocation only }

First fit :- Allocates the first hole that is big enough
Adv :- Simple, fast
Dis :- Internal fragmentation

Next fit :- Same as first fit but starts search always from last allocated hole because it keeps a pointer at the last filled hole.
Adv :- Faster than first fit

Best fit :- Allocate smallest hole that is enough for the process
Adv :- Internal frag. very low
Dis :- Very slow as it traverses entire memory
Fragments left are so small that no new process would fit that.

2015 March 06 Friday 65

Week 10th Day 65th

We are not working with fixed partitions so the remaining holes of partitions can be filled with processes

Important

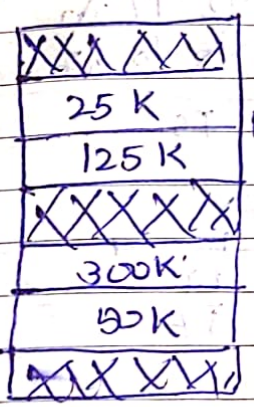
Worst fit → Allocates the largest hole
Dis → internal fragmentation, Overhead

8.00

9.00 Requests from processes are 300K, 25K, 125K, 50K respectively.
The above req. could be satisfied with

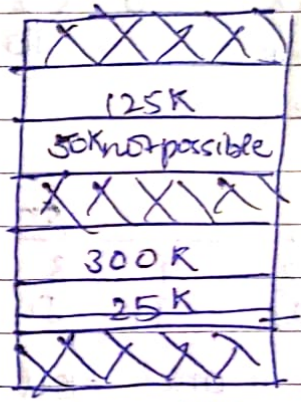
10.00 a) Best but not first c) both
b) first but not best d) none

11.00



12.00 1.00 2.00

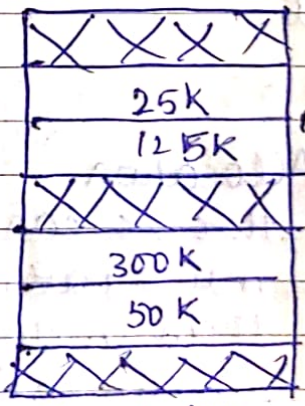
first fit ✓



→ External fragment.

best fit ✗

3.00 4.00



5.00 6.00

Worst fit ✓

APRIL 2015

Monday		6	13	20	27
Tuesday		7	14	21	28
Wednesday	1	8	15	22	29
Thursday	2	9	16	23	30
Friday	3	10	17	24	
Saturday	4	11	18	25	
Sunday	5	12	19	26	

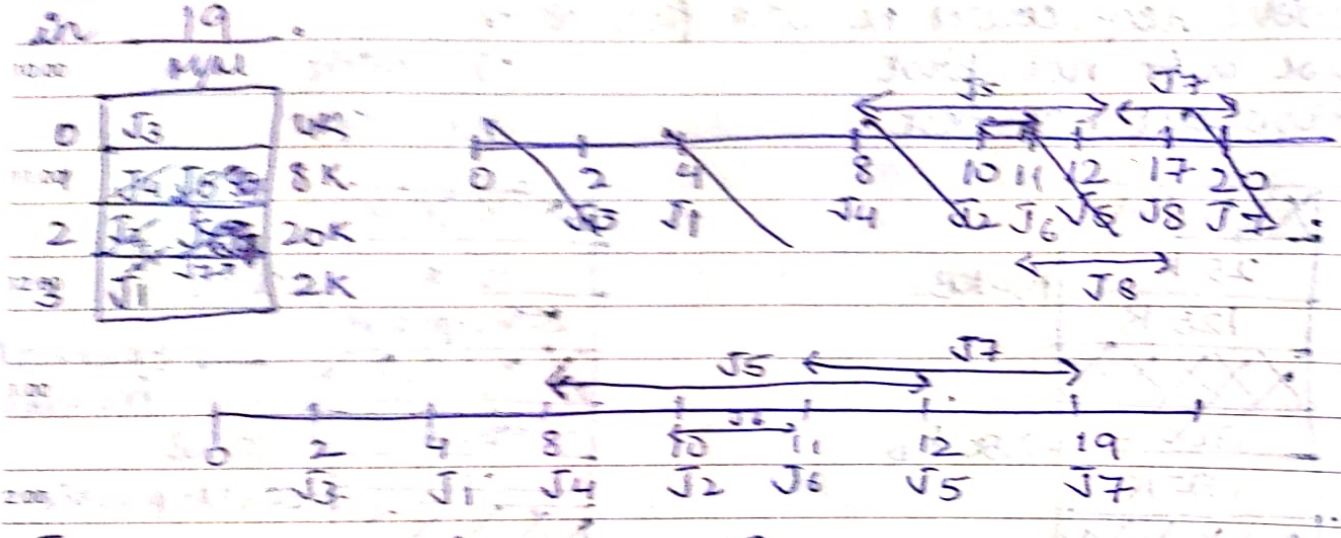
APR MAY JUN

07 March Saturday 2015

Ques:- Solve by Best Fit :-

Request No.	J1	J2	J3	J4	J5	J6	J7	J8
Request Size	2K	14K	3K	6K	6K	10K	7K	20K
Usage Time	4	10	2	8	4	1	8	6

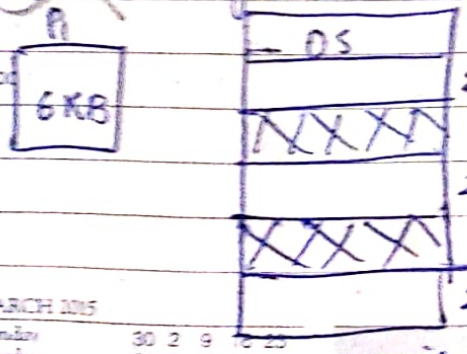
Calculate the time at which 'J7' will be completed



J7 enters at 11 and exits at 19.

So seen as slot is empty we insert process inside it without waiting for best fit.

Non Contiguous Memory Allocation

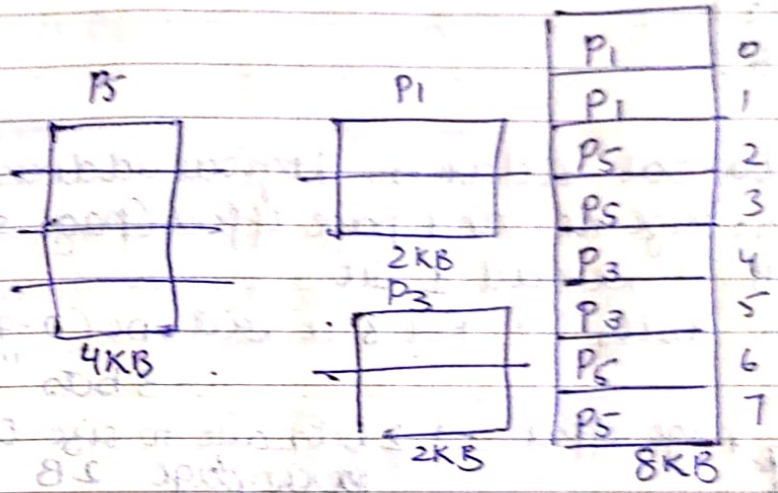


Here we can divide process into 3 parts of 2KB and put them in diff locations. This removes external fragmentation. The division of process is costly as we check for the possible division during dynamic alloc. We have to check no. of holes, size of holes and then divide process accordingly.

So we divide processes early when they are in the secondary memory and these divisions are called pages. The partitions will be done in RAM also and these partitions in RAM are called frames.

* page size = frame size

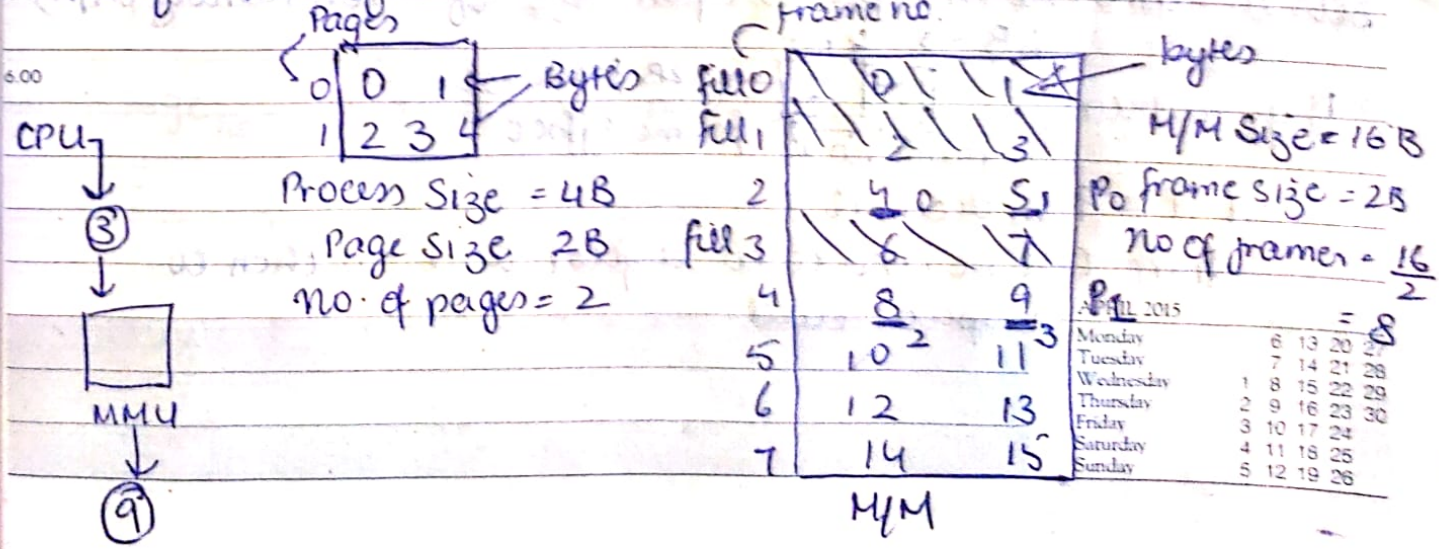
Here we can allocate processes non-continuously. The external fragmentation is totally removed



Frame size = 1KB
∴ Page Size = 1KB

~~Paging~~

We divide process into equal size frames and put them in frames of RAM.



09 March 2015
Monday 48 Week 11th Day
Important

Lets say CPU tries to access P_1 but it doesn't know about paging. Lets say it wants byte no 3 from P_1.

8.00 mapping management unit : The address generated by CPU is converted to absolute address using page table. Page table contains frames no. where page is actually present

9.00

10.00 Page Table P_1

0	f 2
1	f 4

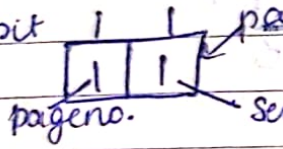
11.00

12 CPU always works on logical address. Logical address is made of 2 components: Page no. and page offset (page size)

1.00 We have a page size of 2 B. ∴ we need 1 bit.

more eg :- (0-15) 16 ⇒ 4 bits. For 8 it will be (0-7) 111 ∴ 3 bits

2.00



3.00 Logical Address

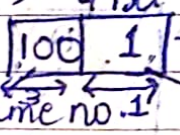
4.00

Physical address is directly related to main memory. The bits to represent PA depends upon size of Main Memory (M/M)

5.00

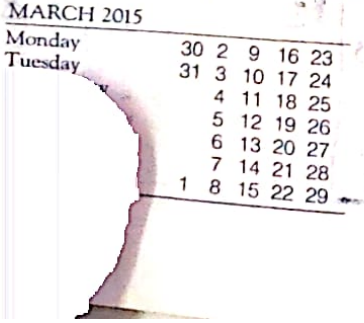
16 B → 4 bit

6.00 Phy. Address



1001 = 9 in decimal

∴ from CPU we convert to logical address then to physical add.



2015

Week 11th Day 69th

March 10 Tuesday

Important

P/T = Page table

Q. CPU wants byte 1 from pages of P1.

CPU → (1) P/T → (5) M/M

Logical address: 0 | 1 → 2-B (page size)

1 → representation of 1 → 0 | f2 ; 1 | f4 ∴ 2=010

4 bit → same

Physical address: 010 | 1

0101 = 5 ∴ 5th location on M/M

CPU → page no. → frame no. → byte

Ques :-

Given :- LAS = 4GB (byte) (Logical address space) [process size].

PAS = 64MB (physical Address Space)

Page size = 4KB = $2^2 \times 2^{10} = 2^{12}$ bits

no. of pages = ?

no. of frames = ?

no. of entries in page table = ?

Size of page table = ?

LA = $2^2 \times 2^{30}$ bytes bits = 2^{32} bits

Page no | page size : 20 | 12 (32)

∴ no. of pages = 2^{20}

physical add = $2^6 \times 2^{20} = 2^{26}$

14 | 12 (26)

∴ no of frames = 2^{14}

$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1K$

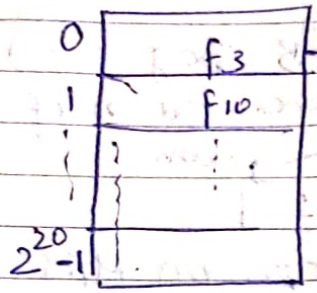
$2^{20} = 1M$
$2^{30} = 1G$
$2^{40} = 1T$

APRIL 2015

Monday					
Tuesday			6	13	20	27
Wednesday			7	14	21	28
Thursday		1	8	15	22	29
Friday		2	9	16	23	30
Saturday		3	10	17	24	
Sunday		4	11	18	25	
		5	12	19	26	

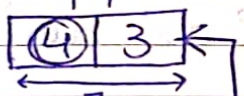
APR MAY JUN

no. of entries in page table = no. of pages in process
$= 2^{20}$ = 1 million



Size of table = entries of table × frame size in bits
$= 2^{20} \times 14$ bits

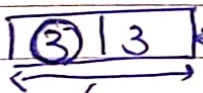
Q. Consider a system which has LA = 7 bits, PA = 6 bits, page size = 8 words / bytes. Then calculate no. of pages & no. of frames.

LA = 

LAS = 2^7

∴ no of pages = 2^4 = 16

page size = 8 = 2^3
∴ 3 bits

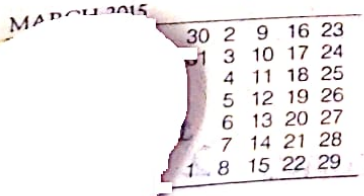
PA = 

PAS = 2^6

∴ no. of frames = 2^3 = 8

no of entries in page table = 2^4 = 16

Size of table = 16 × 3 = 48



Important

Page Table Entry:

page table uses memory management unit to convert logical add. to physical address.

Values in page table entry:

Frame no.	Valid(1)/ Invalid(0)	Protection (RWX)	Reference (0/1)	Caching (Enable/Disable)	Dirty/ Modify (1/0)
← Mandatory field →		Optional Fields			

Valid/Invalid - The page that we are searching for is present in memory or not.

Protection - (Read, Write, Execute) Tells the permission on the data

Reference: Have we brought the page in main memory in the past or not.

LRU method is what check if it occured before or not

Caching - cache means CPU searches for data in cache first of all. When we know a data is being called again and again, we cache it. So now we don't go to main memory or hard disk over and again. It is now in nearest location of CPU can access it in min time. When we need fresh data, we don't want to cache it. Youtube uses cache to show the videos user watches a lot (dynamic)

Dirty :- Let say P1 - Write permit. So CPU can change the value of data but it won't directly get reflected on the secondary memory. So instead of scanning all pages for changed value, we set dirty = (1).

13 March Friday 2015

Page table of each process

Multilevel Paging:

Two level paging

If size of page table is more than the size of frame in main memory, it will not fit there then we have to divide page table into smaller page tables.

Example

PAS = 256MB ∴ PA = $2^8 \times 2^{20} = 2^{28}$ ∴ 28 bits

LAS = 4GB ∴ LA = $2^2 \times 2^{30} = 2^{32}$ ∴ 32 bits

frame size = 4KB = $2^2 \times 2^{10} = 2^{12}$ ∴ 12 bits

Page entry = 2B = 2 bytes = $2 \times 2^3 = 2^4$ = 16 bits = frame size

PA = | 16 | 12 | ∴ 2^{16} frames

(frames after | size; 28)

LA = | 20 | 12 | ∴ 2^{20} pages

(32)

Page Table Entries = 2^{20}

Page table size = $2^{20} \times 2B$

= 2MB

2B
2B
.
.
PT

(inner)

Thus page table size (2MB) > frame size (4K) ∴ it won't fit and we have to divide page table in multiple pages.

No of pages to be divided (page table) = $\frac{2MB}{4K} = \frac{2 \times 2^{20}}{2^2 \times 2^{10}}$

$= 2^9 = 512$ pages.

Here we will make another outer page table

2015 March 14

Week 11th Day 73rd Saturday 58

Important

CPU → page table → frames

512 { 25 | 28 | 2B | } PT

∴ page table size = 512 × 2B = $2^9 \times 2$ = 1KB

Now page table size < frame size ∴ we can fit it there and we don't need to further divide it

Now no of entries in each page using inner PT = $\frac{4K}{2B}$ → frame size / size of data = 2^{11} entries

CPU → LA (32)

20	12
9	11

∴ CPU → LA

p_1	p_2	d
9	11	12

p_1 → outer PT

p_2 → 2^{11} entries

d → page of page table

M/M

Inverted Paging:

Disadvantages of normal paging:

1. Each process has its own page table.
2. PT will be in frames of main memory. This utilizes lot of space to mount PT on frames.

page no	frame no
0	f_0
	X
1	f_1

Page table of P_1

This shows that process 0 of page table lies in frame f_0.

APRIL 2015

Monday	6	13	20	27
Tuesday	7	14	21	28
Wednesday	1	8	15	22	29
Thursday	2	9	16	23	30
Friday	3	10	17	24	
Saturday	4	11	18	25	
Sunday	5	12	19	26	

APR MAY JUN

15 March Sunday 2015

54 Week 11th Day

Important

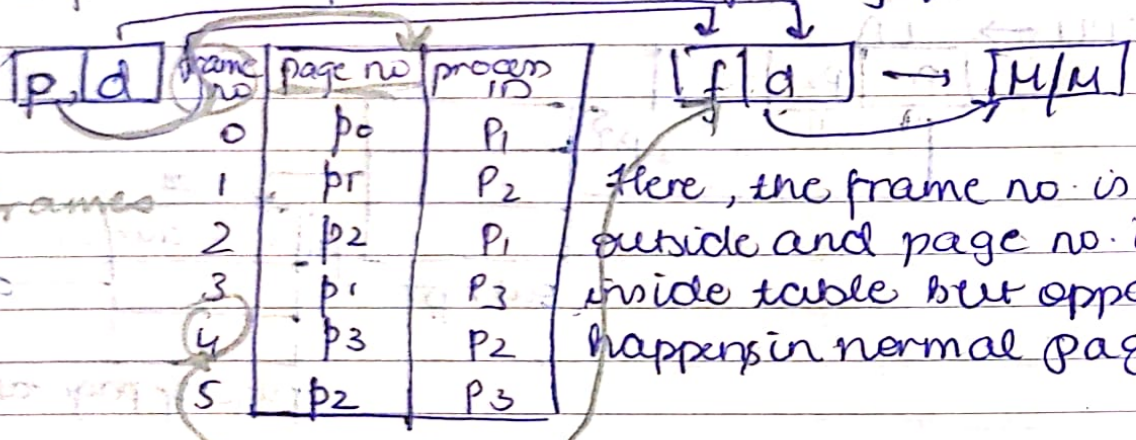
In inverted paging, instead of keeping separate page tables for all processes, we will make only one global page table

We will have as many entries as the number of frames.

In inverted paging :- page no. and process ID are the mandatory fields

Disadvantages :-

The searching takes more time as we have to see the process no. first then process ID and finally frame.



Here, the frame no. is outside and page no. is inside table but opposite happens in normal paging.

Q: Consider a virtual address space of 32 bits and page size of 4KB. System is having RAM of 128KB. Then what will be ratio of page table and inverted page table if size of each entry is of size 4B.

Ans

$4K = 2^{10} \times 2^2 = 2^{12}$
$= 12$ bit

LA = [20 | 12] — 32 —

Page table size = $2^{20} \times 4B$

$128KB = 2^7 \times 2^{10}$
$= 17$ bit

PA = [5 | 12] — 17 —

Important

∴ Inverted page table size = no. of frames × size of each entry
= $2^5 \times 4B$

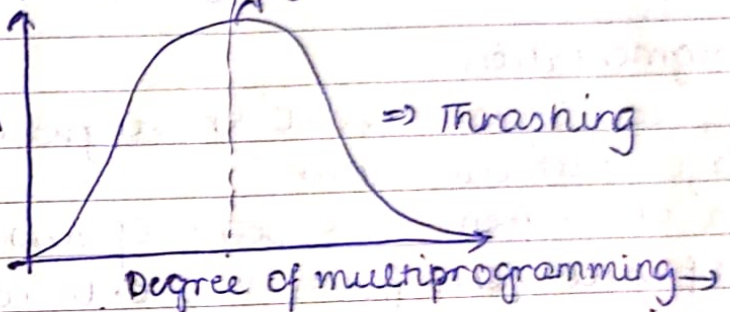
∴ Ratio = $\frac{2^{20} \times 4B}{2^5 \times 4B} = \frac{2^{15} \times 1}{1} = 2^{15} : 1$

8.00

9.00

Thrashing of

10.00 Degree of multiprogramming ↑ No. of processes in RAM.



11.00

12.00

1.00

Degree of multip. can be increased highly if we divide process into pages and bring atleast one page of each process in main memory.

2.00

P1	p1
P2	p2
P3	p3
P4	p4

3.00

If CPU asks for page 1 of any of the processes then its fine but if lets say CPU asks for page 2 of P1 then its not present and this is page fault.

4.00

5.00

Whenever page fault occurs, we use page fault service time. This means we will bring needed page from hard disk to main memory. This service takes a lot of time and OS will be busy to service page fault thus degrading system performance

6.00

Thrashing means there were so many page faults that page hits were lesser due to which system's entire time goes in bringing pages from hard disk to CPU.

APRIL 2015
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
6 13 20 27
7 14 21 28
1 8 15 22 29
2 9 16 23 30
3 10 17 24
4 11 18 25
5 12 19 26

APR MAY JUN

17 March 2015
Tuesday
Week 12th Day 7

Important

Ways to remove thrashing

1 Increase main memory size

2 Long Term Scheduler is asked to lower its speed and not take multiprog. to maximum level.

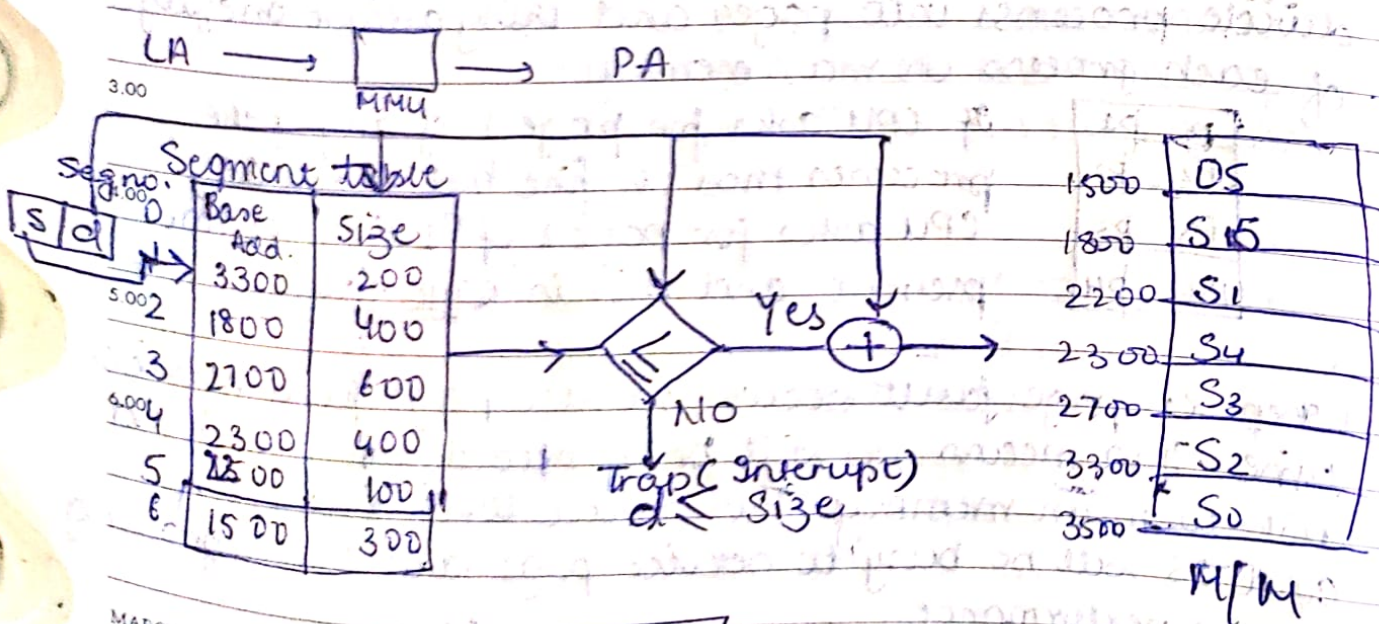
Segmentation :-

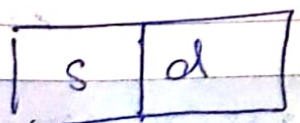
The procedure of dividing processes into parts and then putting them in main memory.

Diff b/w segmentation :-

In paging, we divide without knowing what has the code within. Page fault can occur.

Segmentation works from user point of view. It makes diff. segments of full functions acc. to code. Thus segment sizes are different unlike page which are equal.





seg no. segment offset/size

Important

Overlay:

It is a method by which large size process can be put into memory. Overlay divides process into parts and then each part enters the main memory when needed, then leaves and a new process comes in.

OS has no driver supporting overlay so user has to decide how to divide process for proper execution of functionality of each part. It is usually used in embedded systems where functionalities are fixed.

Our systems are not embedded so we use virtual memory of demand paging process.

Assembler: It is used to convert source code to the object code.

Q:- Consider a two pass assembler pass 1: 80KB, pass 2: 90KB, symbol table: 30KB, common routine: 20KB. At a time only one pass is in use. What is min. partition size required if overlay driver is 10KB size.

Sol:-

	Pass 1	Pass 2
	80	90
	30	30
	20	20
	10	10
	140	150

min partition size = max (140, 150)
= 150

APRIL 2015

Monday		6	13	20	27	
Tuesday		7	14	21	28	
Wednesday	1	8	15	22	29	
Thursday	2	9	16	23	30	
Friday	3	10	17	24		
Saturday	4	11	18	25		
Sunday	5	12	19	26		

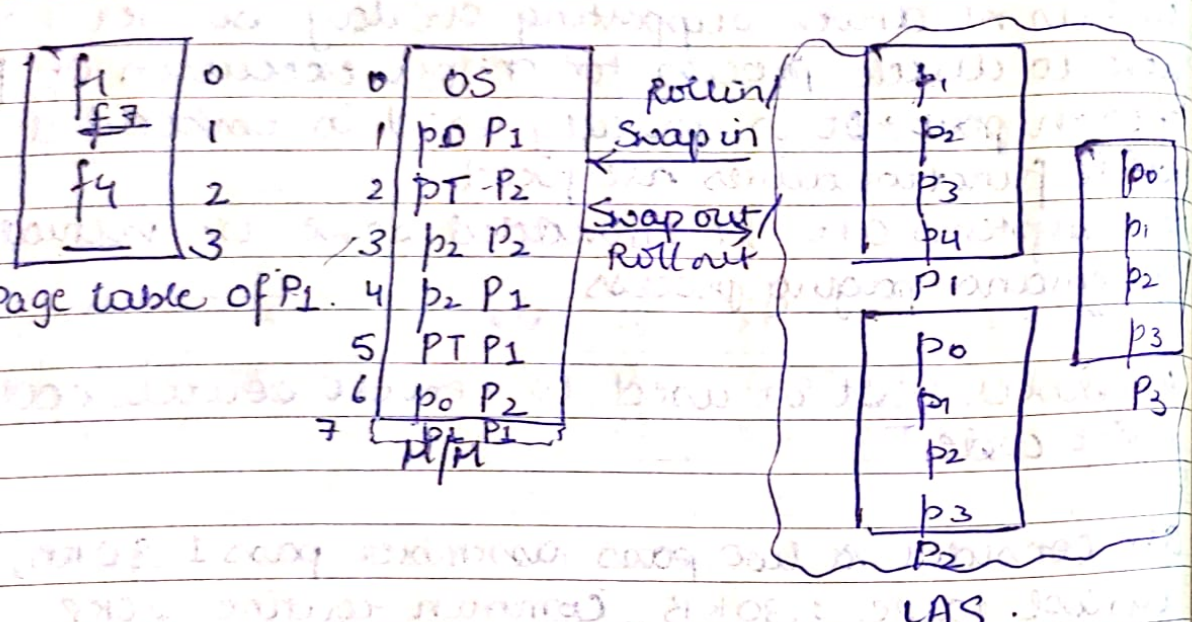
APR MAY JUN

19 March 2015
Thursday 58 Week 12th Day

Important

Virtual memory

It creates an illusion to the programmer that the process whose size is larger than main memory size can also be executed. And for the no. of process too LAS + hard disk.



Locality of reference :

When we bring a page into the memory, we bring in other related pages too.

Page Replacement :

How we bring the pages in and out of the main memory.

Page fault :

When CPU wants to access a page and its absent in M/M, it is page fault. If its not found, trap is generated. Earlier control was with user, now it will go to the OS (Context switching).

30	2	9	16	23		
31	3	10	17	24		
	4	11	18	25		
	5	12	19	26		
	6	13	20	27		
	7	14	21	28		
1	8	15	22	29		

CPU → Page Table (frame) → M/M

Important

OS will first check if the user asking for page demand from CPU is valid or not.

8.00 OS will first go in LAS.
It will pick the page and put it in main memory.

9.00 Then it will update page table
The control is then given back to user.

10.00

~~Effective~~ memory access time (EMAT) : probability (p) (page fault service time (mp)) + (1-p) (main memory access time (mp))

11.00

12.00 Sometime we add MMAT to page fault service time because CPU again searches for that page in main memory but

1.00 MMAT is very less as compared to page fault service time.

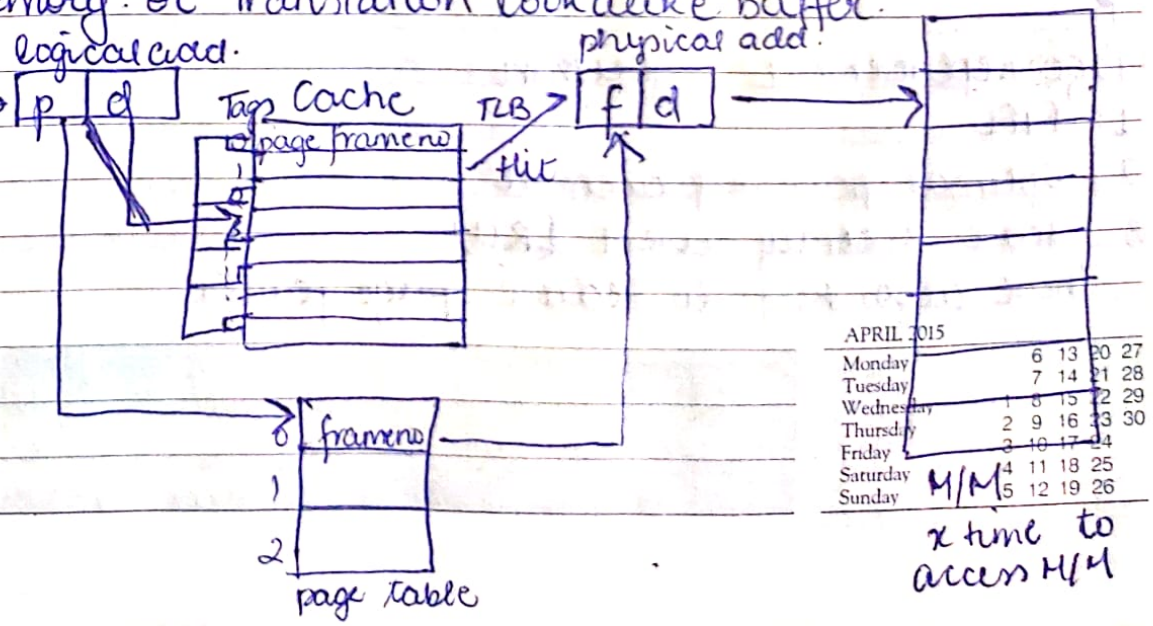
2.00

~~Translation~~ lookaside buffer :- TLB

3.00 We have to access memory twice to get a page. ie
CPU → page table (x) → M/M (x) = 2x [best case]

4.00 But this time will increase in case of multilevel page tables. To reduce this time, we need faster memory ie

5.00 Cache memory. Or Translation lookaside buffer.



We try to check for the needed frame of the page in TLB. We cannot store all frames here because of limited size of Cache.

EMAT = (TLB + x)_hit + (TLB + PT + M/M + x)_Miss

There is advantage of using TLB only when there are more hits ie you can find many frames on TLB. Otherwise extra time gets used to check TLB too.

Question :-

A paging scheme is using TLB. TLB access time 10ns and main memory access time takes 50ns. What is effective memory access time (in ns) if TLB hit ratio is 90% and there is no page fault.

EMAT = (TLB + x)_hit + (TLB + x + x)_miss

$= 90\%(10+50) + 10\%(10+50+50)$

$= 0.9(60) + 0.1(110)$

$= 54 + 11 = 65$ ns

Page Replacement Algorithm

1. FIFO
2. Optimal page replacement
3. least Recently used (LRU)

These algos help to reduce page faults

2015

March 22

Sunday

Week 12th Day 81st

Important

FIFO

Reference String: CPU sends demand of page numbers written in a sequence called reference string.

8.00 We replace the missing pages (page fault) frame by frame

9.00 ie first filled frame is emptied first

Example :-

10.00 Reference String

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

11.00

f3			1	1	1	1	0	0	0	3	3	3	3	2	2
12.00 f2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
1.00	*	*	*	*	H	*	*	*	*	*	*	H	*	*	H

* = page fault H - hit

2.00 Page fault = 12 Page hit = 3

Ratio = $\frac{12}{15} \times 100$ Hit ratio = $\frac{\text{no. of hits}}{\text{no. of reference strings}}$

3.00 = 80% = $3/15 \times 100 = 1/5 \times 100$

= 20%

4.00

Belady's Anomaly :

5.00 When we increase frames, the no. of pages we can keep in main memory increase thus page fault should decrease but that doesn't happen. This is belady's Anomaly.

6.00 In fact, page fault increases. This is problem which occurs in FIFO

APRIL 2015

Monday		6	13	20	27
Tuesday		7	14	21	28
Wednesday	1	8	15	22	29
Thursday	2	9	16	23	30
Friday	3	10	17	24	
Saturday	4	11	18	25	
Sunday	5	12	19	26	

APR MAY JUN

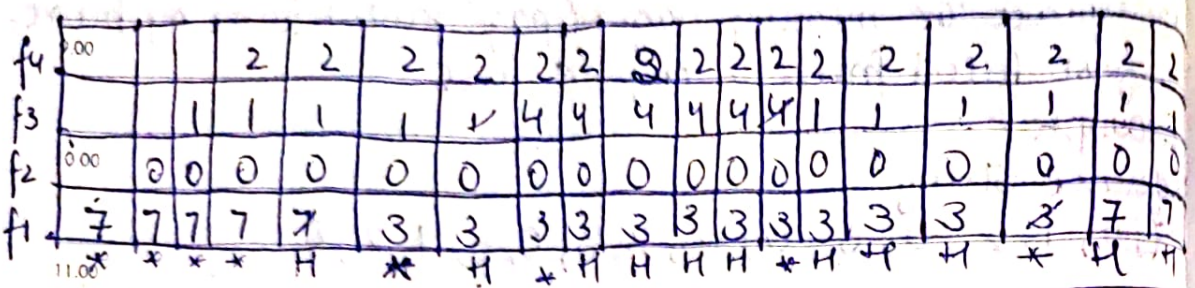
23 March Monday 2015

1Byte = 2^8 bit

Important

Optimal Page Replacement :-

Replace the page which in not used in longest dimension in the future



Ref String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Hits : 12 page faults : 8

Ratio = 12/20 * 100 ratio = 8/20 x 100

Least Recently Used :-

Replace the last recently used page in past

Ref String :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Hits :- 12

Page faults :- 8

Here, we have to search and compare with elements in past. ∴ more time consuming than FIFO. But it performs better in page fault.

MARCH 2015

Monday	30	2	9	16	23
Tuesday	31	3	10	17	24
Wednesday		4	11	18	25
Thursday		5	12	19	26
Friday		6	13	20	27
Saturday		7	14	21	28
Sunday	1	8	15	22	29

Most Recently Used :-

Replace the most recently used page in past

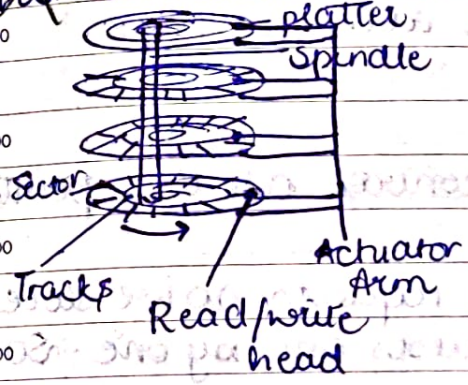
Reference String :-

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

			2	2	2	2	2	2	3	0	3	2	2	2	0	0	0	0	0
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	4	4	4
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
*	*	*	*	H	*	*	*	H	*	*	*	*	H	H	*	H	H	H	H

Hits : 8 page faults : 12

Disk Architecture :-



Platter ⟶ Surface (upper/lower)

Data ⟵ Sectors ⟵ Track

Spindle rotates the platter unidirectionally in clock or anticlockwise direction.

The actuator arm can move the read/write head in and out on the track.

Disk size = Platter x Surfaces x Tracks x Sectors x Disk

Eg = $8 \times 2 \times 256 \times 512 \times 512$ KB

$= 2^3 \times 2 \times 2^8 \times 2^9 \times 2^9 \times 2^{10}$ B

$= 2^{40}$ B = 1TB

No of bits required to represent disk size = 40

APRIL 2015

Monday		6	13	20	27
Tuesday		7	14	21	28
Wednesday	1	8	15	22	29
Thursday	2	9	16	23	30
Friday	3	10	17	24	
Saturday	4	11	18	25	
Sunday	5	12	19	26	

APR MAY JUN

Disk Access Time :-

1. Seek Time :- Time taken by R/W head to read desired track.
2. Rotation time :- Time taken by track for one full rotation
3. Rotational latency :- Time taken to reach to desired sector (half of rotation time)
4. Transfer time = Data to be transfered / Transfer rate

Transfer rate (Data rate) = No. of heads x Capacity of one track x no of rotation in one second

1. Lower the seek time better the performance.
2. Inside track there are many sectors, thus to reach it, rotations are done.
3. We consider half of rotation time for average case.
4. Capacity of track = sectors x data

Controller Time :- Software / driver to control all the operation on disk

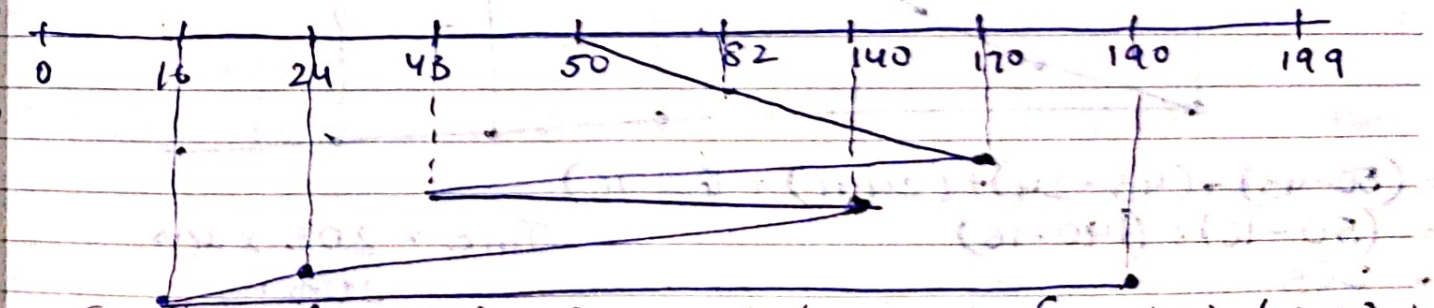
Queue time :- Sometimes disk gets lot of requests so we store them on queue in buffer and fulfill requests one by one. So they wait there ie QT.

Disk Access Time = ST + RT + TT + CT + QT

Disk Scheduling Algorithm :-
Goal :- To minimize the seek time
1. FCFS (First Come first Serve)
2 SSTF (Shortest seek time first)
3 Scan
4 Look
5 CSCAN (Circular Scan)
6 CLOOK (Circular Look)

Request Queue :- Every Disk has a request queue. ~~It a~~ When user creates a request of accessing a data, the data lies on some track. So memory management maps that and inserts the desired track no. in queue.

FCFS Disk Scheduling :-
Q: A disk contains 200 tracks (0-199) request queue contains track no. 82, 170, 43, 140, 24, 16, 190 respectively. Current position of R/W head = 50. Calculate total no. of tracks movements by R/W head.
Sol :-



$= (82-50) + (170-82) + (170-43) + (140-43) + (140-24) + (24-16) + (190-16)$
$= (170-50) + (170-43) + (140-16) + (190-16)$
$= 642$

27 March 2015
Friday W6 Week 13th Day

Important

~~Adv~~ :-

1 no starvation since the sequence in which request enters be service it in that sequence.

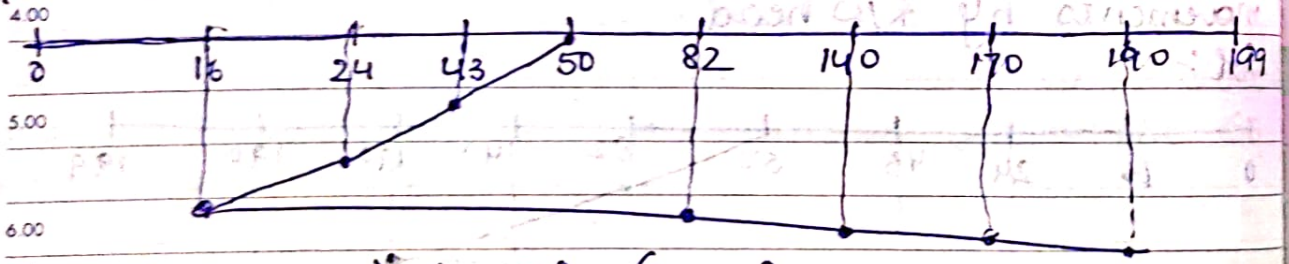
~~Dis~~ :-

1. Performance : Here, we had to go to 190 as well and it was close to 170 but we went to 43.
Thus seek time value is higher in this algo.

~~SSTF~~ Disk scheduling :-

Ques :- A disk contains 200 tracks (0-199). Request queue contains tracks no. 82, 170, 43, 140, 24, 16, 190 respectively. Current position of R/W head = 50.
Calculate total no. of tracks movement by head using shortest seek time first. If R/W head takes 1ns to move from one track to another then total time taken is?

Sol :-



$= (50-43) + (43-24) + (24-16) + (82-16)$

$= (50-16) + (190-16)$

$= 208$

Time = 208 x 1ns
= 208 ns

Advantages :- Better & Optimised performance

Dis-Advantages :- Starvation as the request entering first might have to wait longer.

Overhead time complexity generated as we have to search for the nearest request too.

Important

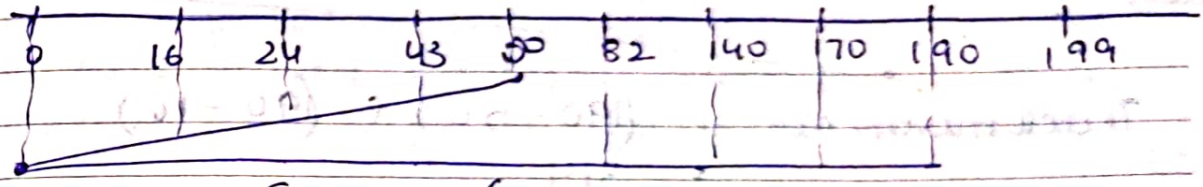
SCAN :-

Ques :- A disk contains 200 tracks (0-199). Request queue contains track nos 82, 170, 43, 140, 24, 16, 190 respectively. Current position of R/W head = 50. Calculate total no. of tracks movements by R/W head using SCAN?

If R/W head takes 1 ns to move from one track to another then total time taken is ___?

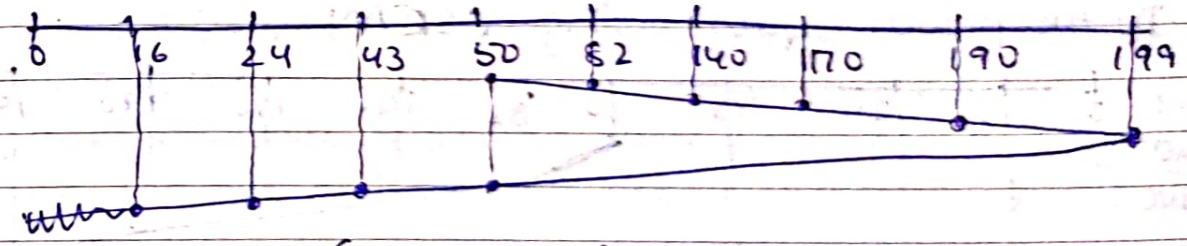
Solution :-

Lower Elevation :-



Track movements = (50-0) + (190+0) Time = 332ns
= 240

Upper Elevation :-



Track movements :- (199-50) + (190-16) Time = 332ns
= 332

Advantages :- If some request comes after 190 till 199, then we can access it since we go till end in one direction.

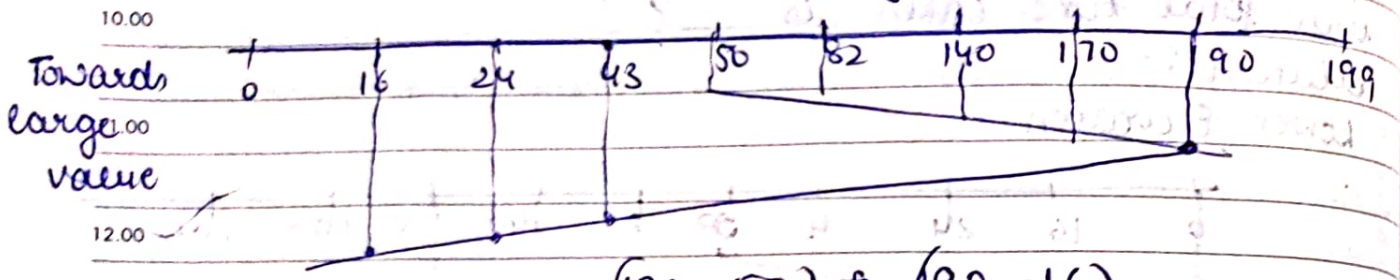
Disad :- If at the end direction changes and then some request comes dynamically like 195, the we can go there as once direction changes we have to go till end on other side and then we can come back to 195.

∴ Starvation can occur.

LOOK :-

Do the previous question using LOOK.

In look, we do not travel extra distance till one of the extreme ends. Rest everything is same as SCAN.

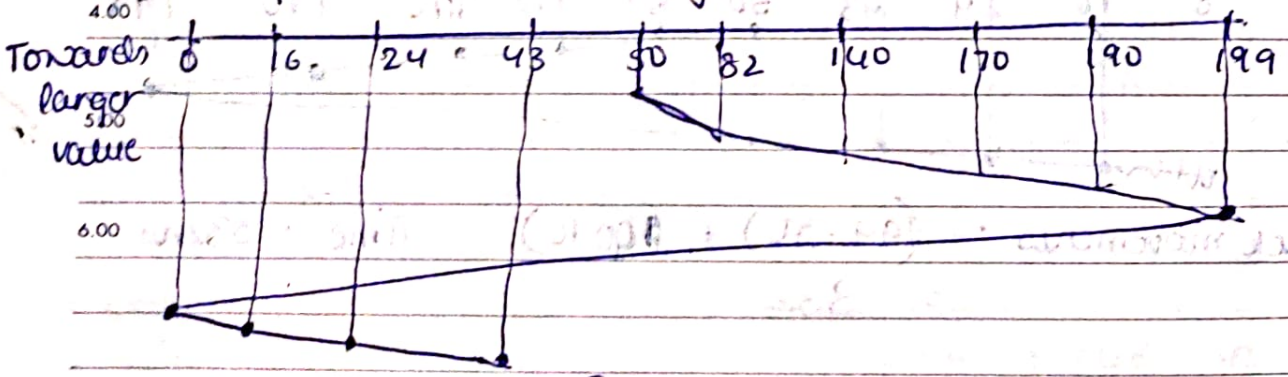


Track movements = (190 - 50) + (190 - 16)
= 314.

∴ Time = 314 ns.

C SCAN Disk scheduling :-

Do the previous ques using CSCAN.

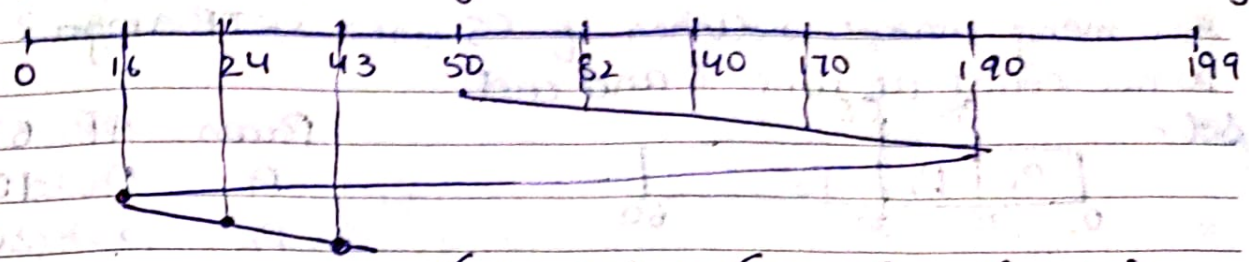


Track movements = (199 - 50) + (199 - 0) + (43 - 0)
= 391.

Important

C LOOK Disk Scheduling :-

Do previous ques. using CLOOK. (Direction is towards large value)

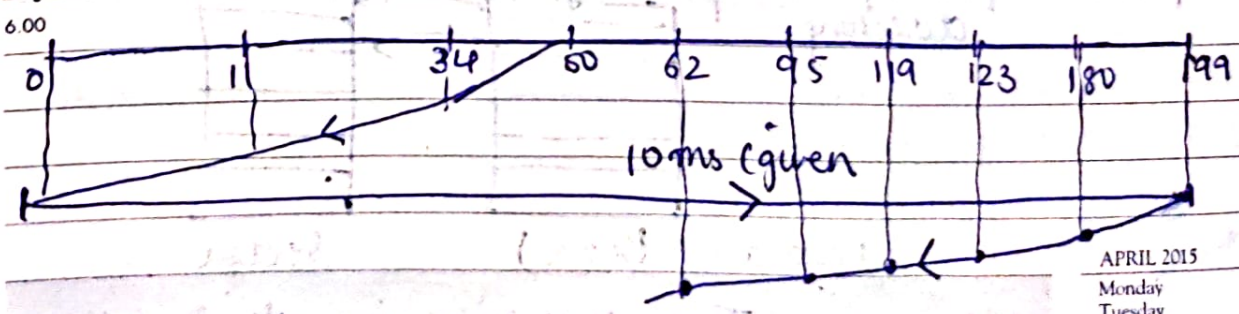


Track movements = (190-50) + (190-16) + (43-16)
= 341.

So we have to go till largest value then come back not fulfilling any request till lowest request and then move forward and fulfil the other requests.

Ques :- Consider the following track requests in disk queue 95, 180, 34, 119, 11, 123, 62, 64. (Scan algo is used R/W head is at location 50. Tracks are numbered from 0 to 199. Head is moving towards smaller track no. on its servicing. Total seek time needed with 2msec time to move from one track to another while servicing these requests is ?
[Assume moving one end to another end will take 10 msec.

Sol :-



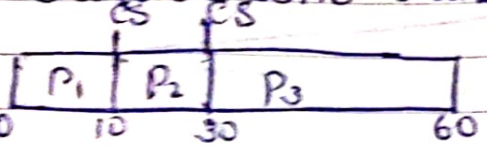
= (50-0) x 2ms + (199-62) x 2ms + 10 ms
= 384 ms

31 March 2015
Tuesday 70 Week 14th Day

Important

Ques: Consider three CPU intensive processes which require 10, 20, 30 time units and arrives at 0, 2, 6 respectively. How many context switches if OS uses SRTF algo? Do not count at time 0 and end.

Sol:



Proan	AT	BT
P1	0	10
P2	2	20
P3	6	30

Context switches (CS) = 2

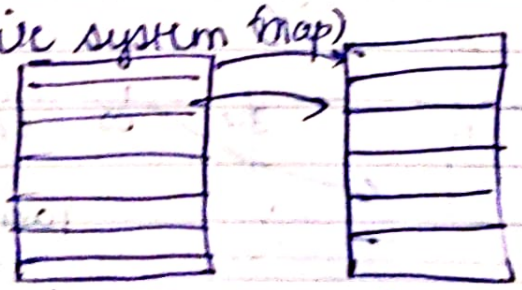
File System in OS:

There is a module in OS which manages all the files of OS. This is file system.

Windows - NTFS (new Technology file system)
Unix - unix file system
Linux - Extended file system
Big data - ZFS file system

File system is a software which decides how the data is stored permanently and later on fetched to the user.

User → file → folder/directory → file system (map)



Data divided (blocks) Sector

file system stores this mapping which helps to store and fetch files.

2015

April
Wednesday 01

Week 14th Day 91st

Important

File : It is collection of data or information.
Windows is majorly GUI based whereas linux is mostly terminal based.

Operation on files

1 Creating	touch	linux
2 Reading	read	
3 Writing	write	
4 Deleting	mkdir	
5 Truncating		
6 Repositioning		

File attributes

1 Name
2 Extension (type)
3 Identifier (given by OS)
4 Location
5 Size
6 Modified date/Created date
7 Protection/Permission
8 Encryption/compression

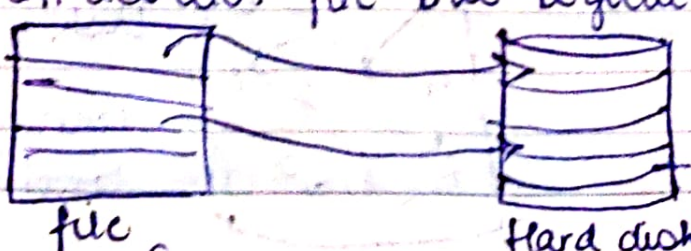
As soon as file is created attributes get generated. It is metadata ie data about data.
Delete : It will erase file and its attributes.
Truncate : It deletes the data inside file but not attributes.
Repositioning : changing the position of read/write head.
Unix - lseek command is used.

File Allocation method :

Contagious Allocation

Non contiguous Allocation
- linked list alloc
- indexed alloc

File system devides file into logical blocks



02 April Thursday 2015

Important

72 Week 14th Day

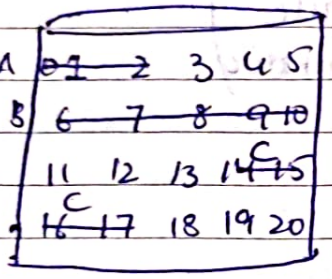
Goals attained using allocation methods :-
1) Efficient disk utilisation
2) Access gets faster

9.00

Contiguous allocation :-
Directory contains all the info related to files.

Directory

file	start	length
A	0	3
B	6	5
C	14	4



Disk

1.00

Advantages :-
Easy to implement. No pointers / no indexes.
Excellent read performance since the data is lying sequentially. ∴ low seek time.

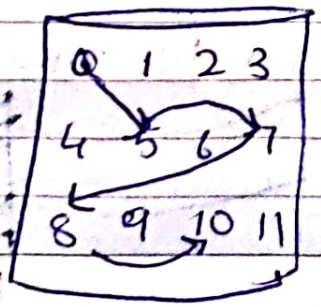
Disadvantages :-
1) Disk will become fragmented (internal and external).
2) Difficult to grow file if edited as next locations maybe filled.

6.00

Linked list allocation :- (Non contiguous)

Directory

file	start
A	2
(5 blocks)	

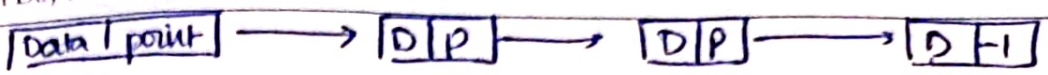


Disk

Pointers being used.

APRIL 2015
Monday 6 13 20 27
Tuesday 7 14 21 28
Wednesday 1 8 15 22 29
Thursday 2 9 16 23 30
Friday 3 10 17 24
Saturday 4 11 18 25
Sunday 5 12 19 26

Important



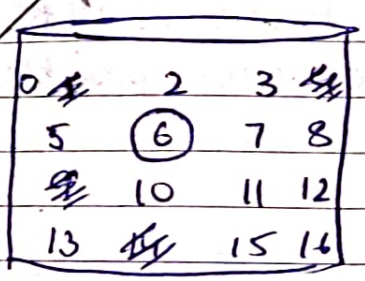
Advantages :-

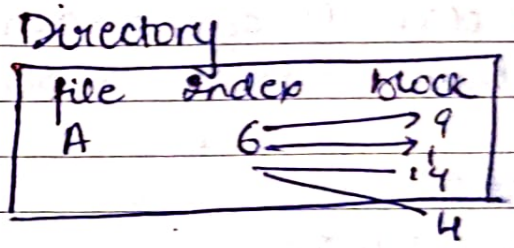
1. No external fragmentation since we can connect blocks with pointers.
2. File size can increase since not continuous.

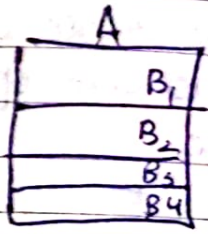
Disadvantages :-

1. Large Seek Time
2. Random access / direct access of data blocks is easier difficult as we need to traverse entire linked list.
3. Overhead of pointers as each pointer also occupies space.

Indexed Allocation :-







The no. of blocks in which data is divided depends upon the physical blocks inside disk's size.

Advantages :-

1. Supports direct access
2. No external fragmentation

Disadvantages :-

1. Pointer overhead
2. Multilevel index as index could be so big that it needs more index to point at it.

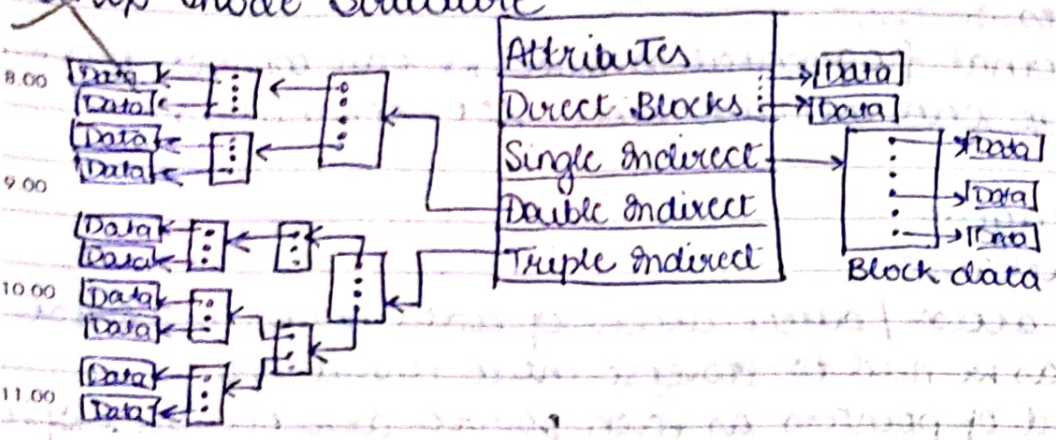
Unix uses I Node to store data.

04 April Saturday 2015

Important

Week 14th Day

Unix Inode Structure



Ques: A file system uses Unix Inode data structure which contains 8 direct block addresses, one indirect block, one double and one triple indirect block. The size of each disk block is 128 B and size of each block address is 8B. Find max. possible file size.

8 direct pointers

Block size of disk = 128 B

Size of each block address = 8B

$\therefore$ no. of pointers in 1 disk block $= \frac{128}{8} = \frac{2^7 B}{2^3} = 2^4$

$= 16$ pointers

16 single indirect = 16 pointers

16 double indirect = 16×16 pointers

16 triple indirect = 16^3 pointers

Given size of each block address = 128 B

$\therefore (8 + 16 + 16^2 + 16^3) \times 128 B$

$8(1 + 2 + 2 \times 16 + 2 \times 16 \times 16)$ KB

$= (35 + 512)$ KB

$= 547$ KB

APRIL 2015

Monday		6	13	20	27	
Tuesday		7	14	21	28	
Wednesday	1	8	15	22	29	
Thursday	2	9	16	23	30	
Friday	3	10	17	24		
Saturday	4	11	18	25		
Sunday	5	12	19	26		

Protection and Security :-

1. In one protection model, computer consists of a collection of objects, hardware or software.
2. Each object has a unique name and can be accessed through a well-defined set of operations.
3. Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Security violation categories :-

1. Breach of confidentiality :-
Unauthorised reading of data. Can be prevented using encryption

2. Breach of Integrity :-
Unauthorised modification of data

3. Breach of availability :-
Unauthorised destruction of data.

4. Theft of service :-
Unauthorised use of resources.

5. Denial of service (DOS) :
Prevention of legitimate use.

Principles of protection :-

Guiding Principle :- principle of least privilege
1. programs, users and systems should be given just enough privileges to perform their tasks.

06 April
Monday

Important

2015
Week 15th Day

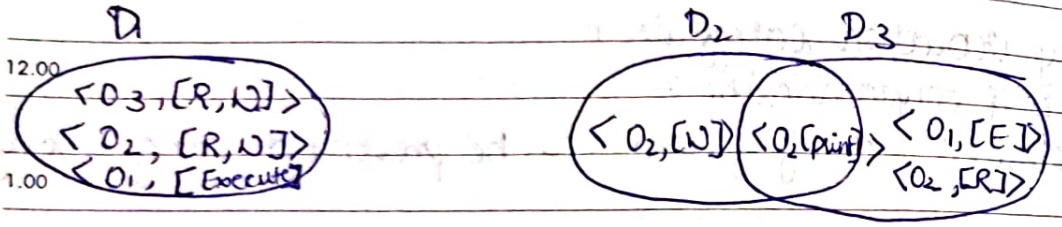
2 limits damage if entity has a bug, gets abused.
3 can be static (during life of system, during life of process)
4 Or dynamic (changed by process as needed) - domain switching, privilege escalation.

9.00

Domain structure

10.00 Domain = set of access-rights.
Access-right = <object-name, right set>

11.00



12.00
1.00

2.00 Access Matrix :
View protection as a matrix (access matrix)
3.00 Rows represent domains
Columns represent objects
4.00 Access(i,j) is the set of operations that a process executing in Domain$_i$ can invoke on object$_j$

5.00

domain \ object	F_1	F_2	F_3	printer
D_1			read	
D_2		read		print
D_3	read/write		execute	

6.00

Special access rights :-
owner of O_i
copy operation from O_i to O_j ("*")
control D_i can modify D_j access rights
transfer :- switch from domain D_i to D_j

APRIL 2015

Monday		6	13	20	27	
Tuesday		7	14	21	28	
	1	8	15	22	29	
	2	9	16	23	30	
	3	10	17	24		
	4	11	18	25		
	5	12	19	26		

When matrix gets too big, we make tables which has the headings of user, object and permissions
8.00 Or else we can make capability model

9.00

Security Problem :-
10.00 System is secure if resources used and accessed as intended under all circumstances. But this condition is quite
11.00 unachievable.
Intruders (crackers) attempt to breach security.
12.00 Threat is a potential security violation.
Attack is attempt to breach security. It can be accidental
1.00 or malicious.

2.00

Security Violation Methods
3.00 1. Masquerading (breach authentication) :
Pretending to be authorised user to escalate privileges.
4.00 2. Replay attack :
As or is or with message modification
5.00 3. Man in the middle attack :
Intruder sits in data flow, masquerading as sender to
6.00 receiver and vice versa.
4. Session hijacking :
Intercept an already-established session to bypass auth.

Security measure levels :
It is impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders

08 April 2015
Wednesday 78 Week 15th Day
Important

Security must occur at 4 levels for effectiveness :-
1 Physical :- Data centers, servers, connected terminals
2 Human :- Avoid social engineering, phishing, dumpster diving
3 Operating system :- Protection mechanisms, debugging.
4 Network :- Intercepted communications, interruptions, DOS etc

Program Threats
Trojan :- It isn't a virus. It is a destructive program that masquerades genuine app. Unlike viruses, they do not replicate. Trojan horses open backdoor entry to your computer which gives malicious users/programs access to system, allowing confidential and personal information to be theft.

Spyware, pop-ups browser windows, convert channels.

Trap Doors :- Specific user identifier or password that circumvents normal security procedures. It could be included in compiler.

Viruses :-
Code fragment embedded in legitimate program
Self replicating, designed to infect other systems
Very specific to CPU architecture, OS, apps.
Usually borne via e-mail or as a macro
A virus attaches itself to a program or file so that it can spread from one system to another, leaving infections as it travels

April
Monday 6 13 20 27
Tuesday 7 14 21 28
Wednesday 1 8 15 22 29
Thursday 2 9 16 23 30
3 10 17 24
4 11 18 25
5 12 19 26

Important

Almost all virus are attached to an executable file, which means virus may exist on your system but it cannot infect your computer unless you run or open malicious program. Thus it cannot spread without human action.

8.00
9.00

Worm:-

10.00 A worm is similar to virus in design (subclass of virus). It can travel without human action. It takes advantage of file or
11.00 information transport features on our system, which allows it to travel unaided. It can replicate itself on the system. It
12.00 would be for a worm to send copy of self to everyone in your email address book

1.00

Port scanning :- Automated attempt to connect to range of
2.00 ports on one or a range of IP addresses.
Detection of answering service protocol.
3.00 Detection of OS and version running on system.
nmap scans all ports in given IP range for response.
4.00 nessus has database of protocols and bugs to apply against a system.
5.00 Frequently launched from zombie systems to decrease traceability.

6.00

Denial of service :-
Overload targeted computer preventing it from doing useful work
Distributed denial of service (DDOS) come from multiple sites at once.
Accidental :- writing bad fork() code
Purposeful :- punishment, extortion.

MAY 2015

Monday		4	11	18	25	
Tuesday		5	12	19	26	
Wednesday		6	13	20	27	
Thursday		7	14	21	28	
Friday	1	8	15	22	29	
Saturday	2	9	16	23	30	
Sunday	3	10	17	24	31	

MAY JUN DEC

10 April 2015
Friday Week 15th Day 10

Important

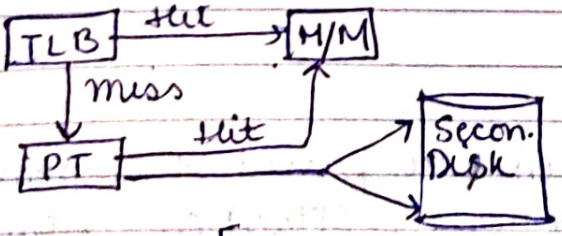
Cryptography:
Way to constrain potential senders and/or receivers of messages based on secret keys.
It enables confirmation of source and builds trust b/w sender and receiver

Encryption:
Constrains set of possible receivers of message.
Symmetric: We use only one key
Asymmetric: We use public and private key.

Firewalling: Placed b/w trusted and untrusted hosts.

Ques:- Consider a paging system uses 1 level page table that resides in M/M and a TLB M/M access time 100ns. TLB lookup takes 20ns. Each page transfer takes 5000ns to/from disk. TLB hit ratio 95%, page fault rate = 10%. Assume that for 20% of total page fault, a dirty page has to be written back to disk before required page is read in from disk. TLB update time is negligible. Find the avg. memory access time?

Solution:-



$0.95(20+100) + .05[((20+100+100)\times 0.90) + 0.10\{(20+100+5000)\times 0.80 + 0.20(20+100+5000+5000)\}]$

without DP

with dirty page

$= 154.$

155

~~EAS of ER should be ckrorsKn~~

First Normal Form :-

- Table should contain no multivalued attributes.

Students Not in 1st Norm form

Roll No	Name	Course
1	Sai	C/C++
2	Harsh	Java
3	Omkar	C/DBMS

Correction 1 Table (1st Normal Form)

Roll No	Name	Course
1	Sai	C
1	Sai	C++
2	Harsh	Java
3	Omkar	C
3	Omkar	DBMS

PK = (Roll No + Course)

Correction 2.

Roll No	Name	Course1	Course2
1	Sai	C	C++
2	Harsh	Java	NULL
3	Omkar	C	DBMS

PK = Roll No.

Correction 3

PK

Roll No	Name
1	Sai
2	Harsh
3	Omkar

R No.	Course
1	C
1	C++
2	Java
3	C
3	DBMS

PK = (R No + Course)

Closure Method

Helps to find all candidate keys in table.

Candidate key is a key that can determine all columns in table.

Candidate key is single - If we add more than one, then it is super key.

MAY 2015

Monday		4	11	18	25
Tuesday		5	12	19	26
Wednesday		6	13	20	27
Thursday		7	14	21	28
Friday	1	8	15	22	29
Saturday	2	9	16	23	30
Sunday	3	10	17	24	31

Prime Attribute : Attribute used in making of candidate key

12 April 2015
Sunday 16 Week 15th Day
Important

$A^+ = BCDA$
$B^+ = CDB$
$C^+ = DC$
$D^+ = D$

Ques: R(ABCD)
FD $\{A \to B, B \to C, C \to D\}$
$A^+ = BCDA$ (C by transitive, A by reflexive)
$B^+ = CDB$
$C^+ = CD$
$D^+ = D$
Also
$(AB)^+ = ABCD$
But it is superkey.

∴ Candidate key = {A}.
prime attribute = A
nonprime = {B, C, D}

Ques: R (ABCD)
FD $\{A \to B; B \to C, C \to D, D \to A\}$.
$A^+ = ABCD$
$B^+ = BCDA$
$C^+ = CDAB$
$D^+ = DABC$

∴ CK = {A, B, C, D}
Prime Attribute :- {A, B, C, D}
Non prime Attribute = {Null}

Q. R(ABCDE)
FD = $\{A \to B, BC \to D, E \to C, D \to A\}$

$AE = BCAE$

Steps: Check attributes on right side. The missing attribute should surely exist on left side to make candidate key.

$AE = ABECD$
$BE = BECDA$
$CE = CE$ ✗

CK = {AE, BE, DE}

First we got AE, check if A, E are on right side now A is on right side of D ∴ replace A with D ∴ DE is CK.
Now D on right side of BC ∴ replace with B ∴ BE is CK

Prime att. = A, B, D, E
Non prime = C

2015
Week 16th Day 103rd

DBMS April 13, 17
Monday

Functional dependency (FD):
It describes the relationship b/w attributes.
$X \rightarrow Y$ ⇒ X determines Y or Y is determined by X.
determinant dependent
$X \rightarrow Y$ cases :-

Sid	Name
1	Ranjit
1	Ranjit

Valid

Sid	Name
1	Ranjit
2	Ranjit

Valid

Sid	Name
1	Ranjit
2	Varun

Valid

Sid	Name
1	Ranjit
1	Varun

Invalid

Types of FD :-
Trivial FD :- If $X \rightarrow Y$ then Y is subset of X.
EG:- Sid → Sid ⇒ $X \rightarrow Y$ is Trivial (Always Reflexive)
This is always valid.
$X \cap Y \neq \phi$ EG: Sid Sname → Sid $\neq \phi$ (Sid)

Non trivial FD :- If $X \rightarrow Y$, then $X \cap Y = \phi$
EG:- Sid → Sname, Sid → Email

Properties of FD (7)
Reflexivity : If Y is subset of X, then $X \rightarrow Y$
Augmentation :- If $X \rightarrow Y$, then $XZ \rightarrow YZ$
Transitive :- If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
Union :- If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
Decomposition :- If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
Pseudo transitive :- If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
Composition :- If $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

14 April Tuesday

partial dependency :- LHS : proper subset of CK and RHS - Non prime attr 2015

Important

LHS = CK & SK AND RHS - PA

Second Normal form :-

Table or relation must be in 1st normal form. All the non-prime attributes should be fully functional dependent on candidate key ie no partial depen. in relation or table.

Not in second Norm form

Cust ID	Store ID	Location
1	1	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bang.
4	3	Mumb.

PK PK

C.ID	SID
1	1
1	3
2	1
3	2
4	3

PK

SID	Location
1	Delhi
2	Banglore
3	Mumbai
2	
3	

Can. Key. C.ID + Store ID

Prime Att. = CID, Store ID

Non prime . location

Here, location depends just on store ID and not CID + SID

∴ it is not 2nd normal form.

Ques : R(ABCDEF)

FD { C → F, E → A, EC → D, A → B }

EC^+ = ECFADB CK = {EC}

prime attributes :- {E, C}

non-prime attributes , {A , B, D, F}

Now we will check that for CK (EC), the proper subsets ie E, C separately determine any non prime att-or not.

C determines F ∴ partial dependency

E determines A

∴ not in 2nd normal form.

April

Monday		6	13	20	27	
Tuesday		7	14	21	28	
Wednesday	1	8	15	22	29	
Thursday	2	9	16	23	30	
Friday	3	10	17	24		
Saturday	4	11	18	25		
Sunday	5	12	19	26		

2015

Week 16th Day 105th

DBMS April 15

Wednesday

LHS = CK or SK OR RHS = PA

Third Normal form :-

Table or relation must be in second normal form.

There should be no transitive dependency in table.

ie CK → Non prime 1 Nonprime 1 → Non prime 2

⇒ CK → Nonprime 2

This is not a third normal form.

No non prime should determine non prime

Ques R(ABCD)

FD : AB → C , C → D

CK = AB

PA = A, B NPA = CD

Here AB → C C → D ∴ AB → D (transitive)

∴ not third normal form

Ques : R(ABCD)

FD ⇔ AB → CD , DB → A

CK : $AB^+ \Rightarrow ABCD$; $DB^+ = ADBC$

∴ CK = {AB ; DB}

PA = A, B, D

NPA = C

BCNF (Boyce Codd Normal Form)

Special case of 3rd normal form :-

All attributes of function dependencies should be dependent on the candidate key.

LHS of each FD should be CK or super key. + 3rd normal form

Monday					
Tuesday					26
Wednesday			6	13	20 27
Thursday			7	14	21 28
Friday		1	8	15	22 29
Saturday		2	9	16	23 30
Sunday		3	10	17	24 31

16 April Thursday 2015

Important

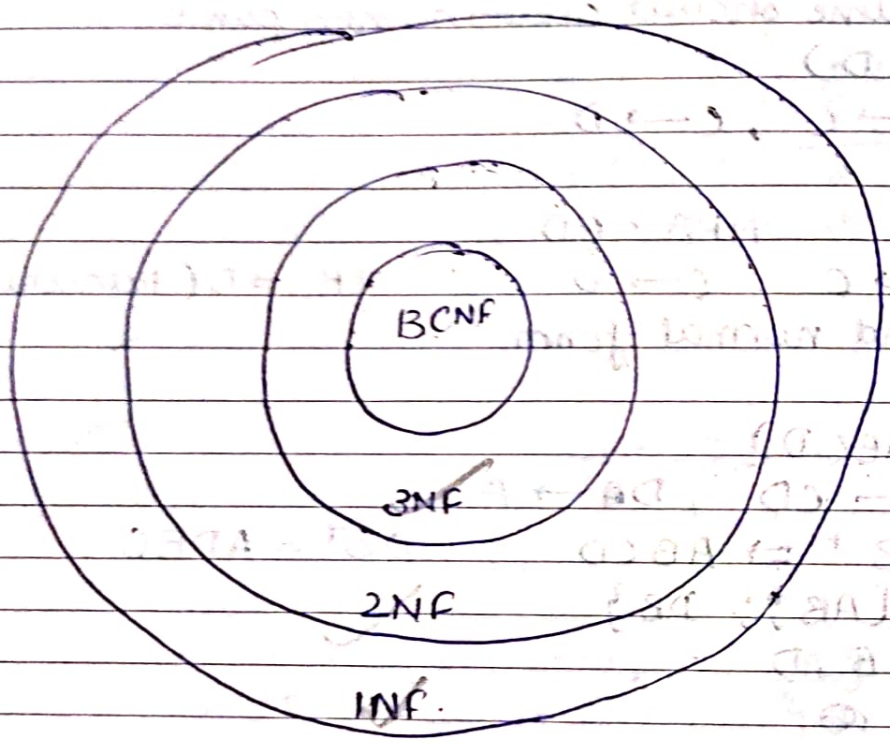
20 Week 16th Day

Ques :- CK = {Roll no., Voter ID}.

FD = { Roll no → name
Roll no → Voter ID
Voter ID → age
Voter ID → Roll no }

PA : Roll, Voter
NPA : name, age

∴ It is BCNF



A table is in 2NF but no compulsion that it is in the 3NF.

2015

April 17
Friday

Week 16th Day 107th

Important

8.00

9.00

10.00

11.00

12.00

1.00

2.00

3.00

4.00

5.00

6.00

MAY 2015

Monday		4	11	18	25
Tuesday		5	12	19	26
Wednesday		6	13	20	27
Thursday		7	14	21	28
Friday	1	8	15	22	29
Saturday	2	9	16	23	30
Sunday	3	10	17	24	31

OCT
NOV
DEC
JAN
JUN