DB = 1

Ex

W1 : down(DB)
Database;

W2 : down(DB)
↳ Busy wait.

```
void writer (void)
{ while (1)
  { down (DB);
    Database;
    up (DB);
  }
}
```

```
int rc = 0;
void reader (void)
{   down (mutex);
    rc = rc + 1;
    up (mutex);
    down (DB);
    Database;          C.S.
    up (DB);
    down (mutex);
    rc = rc - 1;
    up (mutex);
}
```

```
void reader (void) { while (1).
{   down (mutex)
    rc = rc + 1;
    if (rc == 1)
      down (DB) // No writer usage.
    up (mutex)
    DB  // CS.

    down (mutex);
    rc = rc - 1;
    if (rc == 0) up (DB) // (writer can access)
                 up mutex).
```

$r_c = 0$ ✗ ✓ ✗ 0 DB = ✗ 1 mutex = ✗ 0.

R-W:

r_1 : down (mutex)

r_2 : down (mutex) ⇒ sleep / BW

r_1 : $r_c = r_c + 1 = 0 + 1 = 1$

if ($r_c == 1$)

W_1 : down (DB)

Database

r_1 : down (DB) // BW (r_1 r_2 in side).

W_1 : up (DB)

r_1 : down (DB) — ✗ 0

up (mutex) ✓

Database

r_2 down (mutex) ✗ 0

$r_c = r_c + 1 = 1 + 1 = 2$

W_2 = down (DB) 0 → BW

r_2 : Database.

up (mutex)

Database. ✓

r_1 : down (mutex).
$r_c = r_c + 1$
→ —

r_2 : down (mutex) → BW.
→ —

r_1 : if ($r_c == 0$) // false.
→ — up (mutex).

r_2 : down (mutex).
$r_c = (r_c == 0)$ up (DB) ✓
up (mutex)

Ques.) What happen if you interchange down mutex and $r_c = r_c + 1$ in the reader code?

(A) No problem, the solution will work fine.
(B) Multiple reader are not allowed
(C) Reader and writer both can access database at the same time.
(D) None of the above.

$r_c = 0$ 1
$r_1 = r_c + 1$ → Not atomic.
→ —

Alternate : Both r_1 & r_2 execute $r_c = 2$ and then got preempted and then writer executed (down (DB) But readers without checking r_c, directly jump to DB along with writer).

Ques int R=0 , W = 0 ;
Semaphore mutex = 1

```
void Reader (void)
{   L1: down (mutex)
    if (W==1)
    {  _____ (1)  up(mutex)
       go to L1 ;
    }
    else
    {  R = R+1 ;
       _____(2)  up (mutex)
    }
    Database
    down (mutex)
    R = R-1
    up (mutex)
}
void Writer (void)
{   L2 : down (mutex) .
    if (_____) (3)   W==1 or R != 1
    {  up (mutex)
       goto L2 ;
    }
    W=1 ;
    up (mutex);
    Database
    down(mutex); W=0 up(mutex); }
```

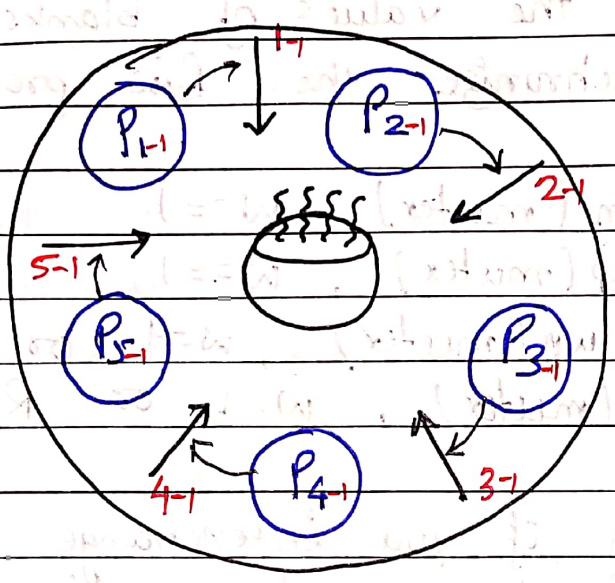
what should be the values of blanks ① ② ③ in order to synchronize the R/W problem.

(A) up(mutex), down(mutex) W==1
(B) down(mutex), up(mutex), W==1
(C) down(mutex), up(mutex), W=1 or R≥1
(D) up(mutex), up(mutex), W==1 or R≥1.

Ques 2.) What happen if you interchange W==1 and up(mutex) in above problem. in writer code.

(A) No problem
(B) Multiple Reader are not allowed
(C) only Multiple writer are allowed in DB
(D) Reader and writer access DB at the same time

Dining Philosopher's Problem



Process	fork
P_0	(0 1)
P_1	(1 2)
P_2	(2 3)
P_3	(3 4)
P_4	(4 5)

Deadlock ✓
(as there is no Progress)

```
void Philosophers (int i)
 {
    while (1)
    { thinking ();
      take fork (i); // take left fork
      takefork ((i+1)%N); take right fork
      eat ();
      put fork (i);
      put fork ((i+1)%N);
    }
 }
```

Solution : T = 0 H = 1 E = 2.

(Thinking) (Hungary) (Eating)

	0	1	2	3	4
state	0	0	0	0	0

- initialy philosophers are in thinking state.
- when they feel hungary, go in eating state.
- when all philosophers get hungary at the same time, all will ~~become~~ ~~if~~ pick up the left fork first and when they try to attempt to take the right fork, the right fork is not available. Then all the philosophers will wait for each other for the right fork and this results in deadlock position.

- The solution of the above problem is:

```
#define N 5          // No of philosophers
#define LEFT (i+N-1) % N    // Left philosopher.
#define RIGHT (i+1) % N     // Right philosopher.
#define THINKING 0
#Define HUNGARY 1
#Define EATING 2
Semaphore mutex = 1
Semaphore S[N] = ; // all initialized to 0
int state [N] ; an array to keep track of states

void philosopher (int i)
{
    while (1)
    {
        Thinking ();
        take forks (i);
        eat ();
        put forks (i);
    }
}
```

```
take forks ( int i)
{       down (mutex);
        state[i] = HUNGARY;
        test (i) ;
        up(mutex) ;
        down ( S[i]);
}

put forks ( int i)
{       down (mutex) ;
        state [i] = THINKING ;
            test ( LEFT);
            test( RIGHT) ;
        up(mutex);
}

void  test ( int i)
{   if state [i] == HUNGARY && ( state[LEFT] ≠ EATING
                                && state[RIGHT] ≠ EATING
    { state[i] = EATING ;
        up ( S[i]);
    }
}
```

OPERATING SYSTEM

MUTEX : mutex is binary semaphore used by philosopher in mutual exclusion manner.

state[N] Integer array used to keep track of every philosopher's state. Initially all philosophers will be in thinking state.

S[N] Array of binary semaphore initially all are assigned to zero.

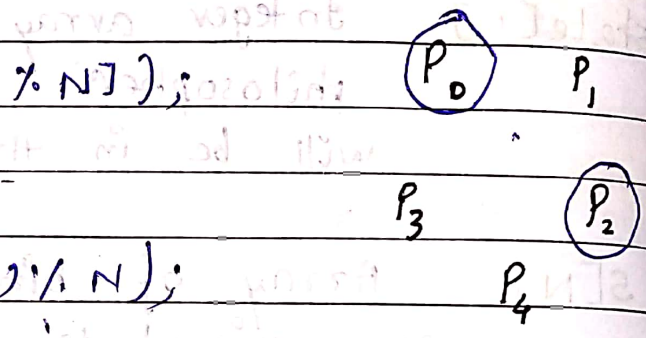
Ques Assume the philosopher P_1 and P_2 are in the eating state then if the philosopher P_3 is time to go into eating state then which statement in the above code ~~try to go into eating~~ is trying to control the above condition ?

(A) if condition
(B) test (i)
(C) down (mutex)
(D) down (S[i])

Ques) What happen if you interchange up (mutex) and down S[i] in the take fork function?
(A) No problem, solution will work fine
(B) More than two philosopher will go into eating state.
(C) It is possible for deadlock.
(D) None of the above.

Ques) Let P_0 to P_4 be process and m_0 to m_4 be binary semaphore. Mutex initialized to 1. Each Process execute below code.

```
wait ( m[i] );
wait ( m[(i+1) % N] );
C.S
signal (m[i])
signal (m[(i+1)% N]);
```



Consider the following

I ME is satisfied

II ME is Not satisfied

III It is possible for Deadlock.

(A) only I

(B) only II

(C) only I & III

(D) only II & III

Ques) Suppose we want to synchronize concurrent process P and Q using binary semaphores S_1 and S_2, the code for the process P and Q is shown below.

Process P

```
while (1)
{ P(S1);
  P(S2);
  CS
  V(S1);
  V(S2);
}
```

Process Q

```
while (1)
{ P(S1);
  P(S2);
  CS
  V(S1);
  V(S2)
}
```

This results ensure
(A) ME
(B) starvation but Not DL
(C) DL
(D) None.

Ques If P(S1) and P(S2) were interchanged, then
(A) ME
(B) DL
(C) Both A & B
(D) None.

Ques) Three concurrent Process executes code segment X Y Z executes, that access a b c. Process a b c Y execute P operation on semaphore b, c, d.

Process Y : P (b c d)
Process Z : P (C d a)

before entering respective code segment. After completing the execution of its code segment, each process invoke the V operations on its 3 semaphores. All semaphores are binary semaphores and initialized to 1. Which one of the following represent a deadlock free order of invoking P operation.

(a) X : P (a b c)

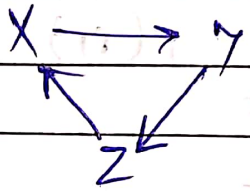
(a) X : a b c Y : b c d Z : c d a

(b) X : b a c Y : b c d Z : a c d

(c) X : b a c Y : c b d Z : a c d

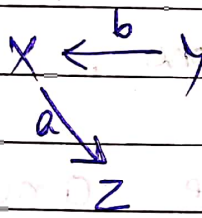
(d) X : a b c Y : b c d Z : c d a

(a) X a b c
Y b c d
Z c d a



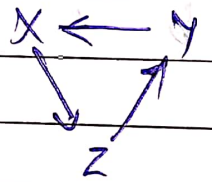
DL

(b) X b a c
Y b c d
Z a c d



✓ No DL

(c) X : b a c
Y : c b d
Z : a c d



DL

Ques.) Suppose we want to synchronize two process

Process P
```
while (1)
{   W : ____
    print '0'
    print '0';
    X : ____
}
```

Process Q
```
while (1)
{   Y : ____
    print '1';
    print '1';
    Z : ____
}
```

Sync statements can be inserted at points W, X, Y, Z which of the following will lead to output string.. 00 11 00 11 00 11

	W	X	Y	Z	S	T	
(A)	P(S)	V(S)	P(T)	V(T)	1	1	X
(B)	P(S)	V(T)	P(T)	V(S)	1	0	✓
(C)	P(S)	V(T)	P(T)	V(S)	1	1	X
(D)	P(S)	V(S)	P(T)	V(T)	1	0	X

Ques Consider 6 concurrent process S=0

A: Down(s) CS up(s)
B: D(s) CS U(s)
C: D(s) CS U(s)
D: U(s) CS D(s)
E: U(s) CS D(s)
F: U(s) CS D(s)

what is maximum value of S. 3

Ques Consider a schedule with 3 Process A, B, C binary semaphore are.

```
Process A          Process B          Process C
{ while(1)         { P(T)             { P(Z);
  P(S)               V(Z)               V(S);
  **
  V(T)
}                  }                  }
}
```

** Prints 2 times.

Ques Consider two process :

P_0

```
while (1)
{ down (mutex);
  print '$';
  up (mutex);
}
```

P_1

```
while (1)
{ down (mutex);
  print '#';
  up (mutex);
}
```

Which of the following is possible pattern

I $ # * *

II # * $ *

III (# / $)*

(A) I
(B) I & II
(C) All
(D) None.

Ques:

A : D(S)
D(Q)
print ("Akash");
V(Q)
V(S)

B : D(Q)
D(S)
print "Gupta"
V(Q)
V(S)

Initially Q = 1 S = 1

In above situation :

(A) DL may occur
(B) Print Akash Gupta
(C) DL never occur
(D)

Que Counting semaphore value is 10 in a specific time. S=10. 15 Do con P up

10 − 15 + P = 35 If the new value of semaphore is 35 find value of P.

P = 40 ✓

MONITORS

* Monitor is a programming language compiler support to acheive synchronization. It is a collection of variable, procedure ~~an~~ combined together, ~~&~~ a in a special kind of module or a package.

- The process running outside the monitor cannot directly access the internal variable of monitor. But however they can call the procedure of the monitor.

- The monitor are an important property that only one process can be inside the monitor at any point of time.

- Syntax of the monitor :

Keywords → Monitor Example name of Monitor

```
{ variables ;
  condition variables;
   Procedure P1
     { = }.
   Procedure P2
     { = }.
}.
```

Condition Variable :

eg. condition X, Y;

- The two different operations are performed on the monitors.
  1. wait ( ) — X.wait() or wait(x);
  2. signal ( ) — Y.wait( ) or wait(y);
- Process performing wait operation only on C.V will be suspended and the suspended proces is placed in the block Queue of respective condition variable.
- Every condition variable have its own Queue.
- Signal :

```
signal ( Full );
signal ( empty );
```

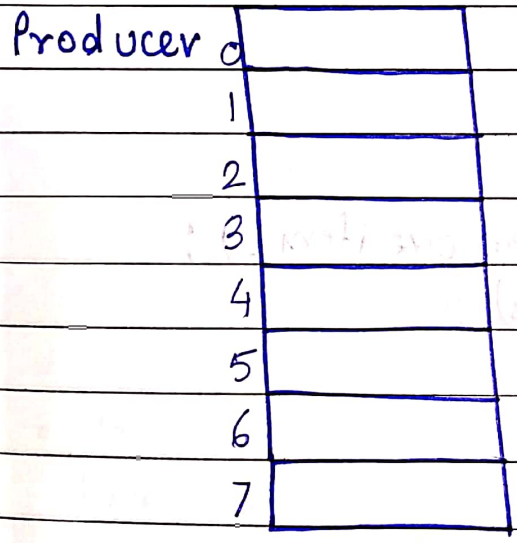
X. Signal ( ) or signal(x);
Y. signal ( ) or signal (y);

- There are two cases :

(i) The block Queue of a respective condition variable is empty. (No suspended process on respective block Queue). The signal has no effect and the signal will be lost.

(ii) The block Queue of respective condition variable is not empty. (Already there is some suspended process in block Queue) one process will be resumed from the block queue and one process will continue the execution.

Producer-Consumer with monitors




```
Consumer    Monitor Producer-Consumer
            {  integer count = 0;
               condition FULL, EMPTY;
            procedure Enter_items
               {  if (count == N)
                     { wait(FULL); }
                  Enter(item);
                  count = count + 1;
                  if (count == 1)
                  signal (EMPTY);
               }
```

```
Procedure Remove_item
{   if (count == 0)
        { wait (EMPTY); }
    Remove (item);
    count = count - 1;
    if (count == N-1)
        signal (FULL);
}
}
```

```
Procedure → Producer
{   while (true)
    {   produce_item (item);
        producer_consumer.enteritem();
    }
}
```

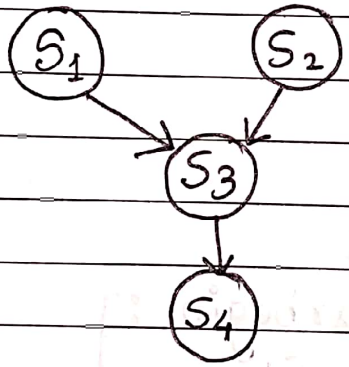
```
Procedure_Consumer
{   while (true)
    {   producer_consumer.Removeitem();
        process_item (items);
    }
}
```

CONCURRENT PROGRAMMING

- Consider following instruction

S_1 : $a = b + c$
S_2 : $d = e + f$
S_3 : $g = a / d$
S_4 : $h = g * i$.

Read Set = { b, c, e, f, a, d, g, i }
Write set = { a, d, g, h }



Dependency Graph

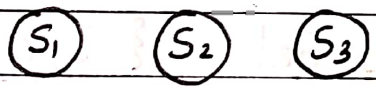
(i) $R(i) \cap W(j) = \phi$
(ii) $W(i) \cap R(j) = \phi$
(iii) $W(i) \cap W(j) = \phi$.

} Independent

- S_1 and S_2 execute concurrently because they do not have any common variable.
- Any two statements Si and Sj can be executed concurrently or parallely if and only if it satisfy the three rules.

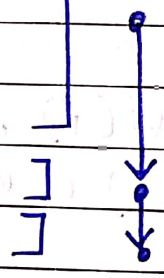
- Concurrent program will be written by using the below statement

begin	parbegin / co-begin
S_1	S_1
S_2	S_2
S_3	S_3
end	co-end



For previous example:

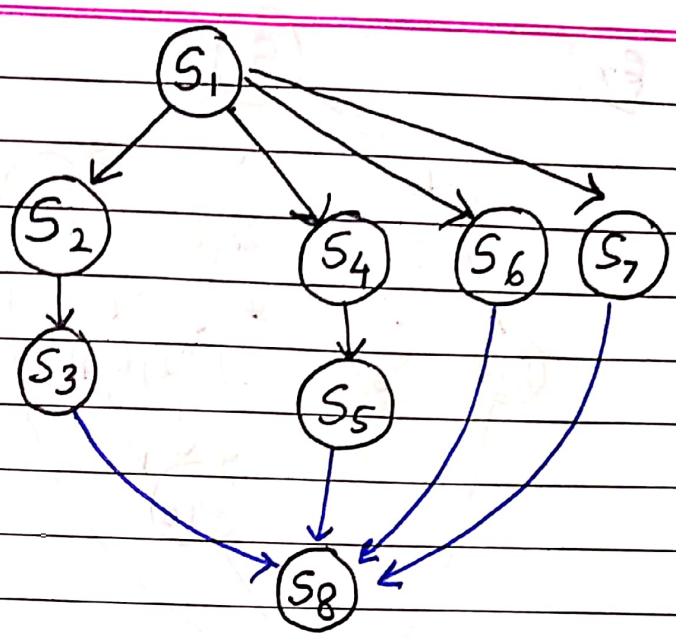
```
begin :
    parbegin :
        S1
        S2
    parend.
    S3
    S4
end
```



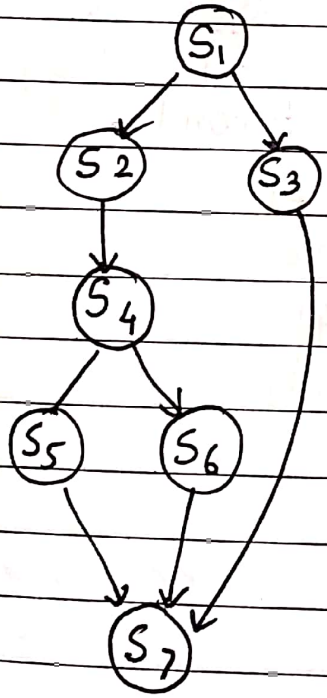
- If nothing is given, by default use begin.

Ques Consider a program:

```
S1 ;
parbegin :
begin S2
      S3
end
begin
      S4
      S5
end
      S6
      S7
Parend
      S8
```

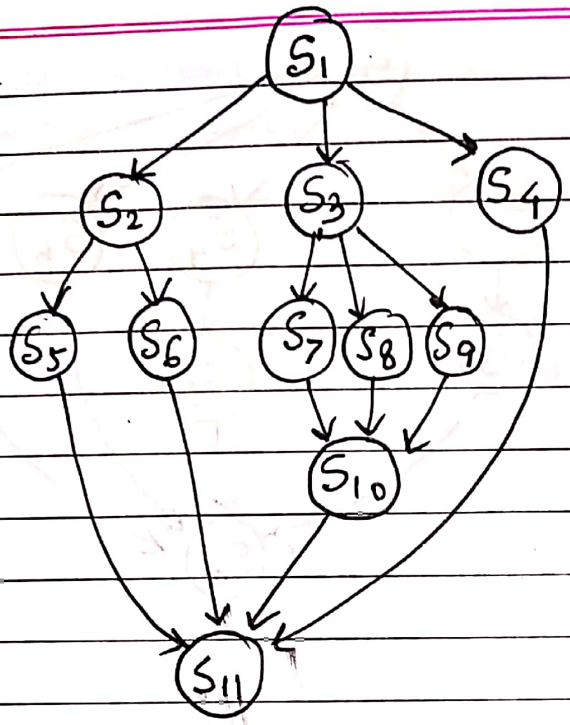


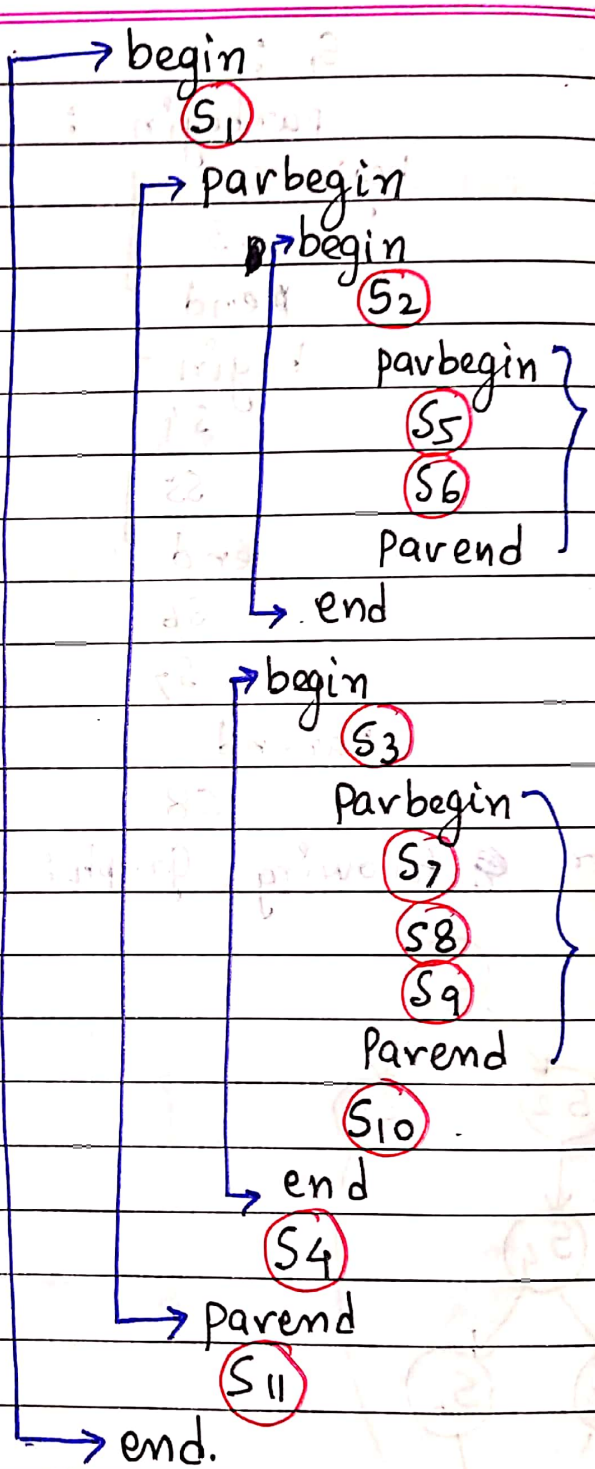
Consider following Graph:



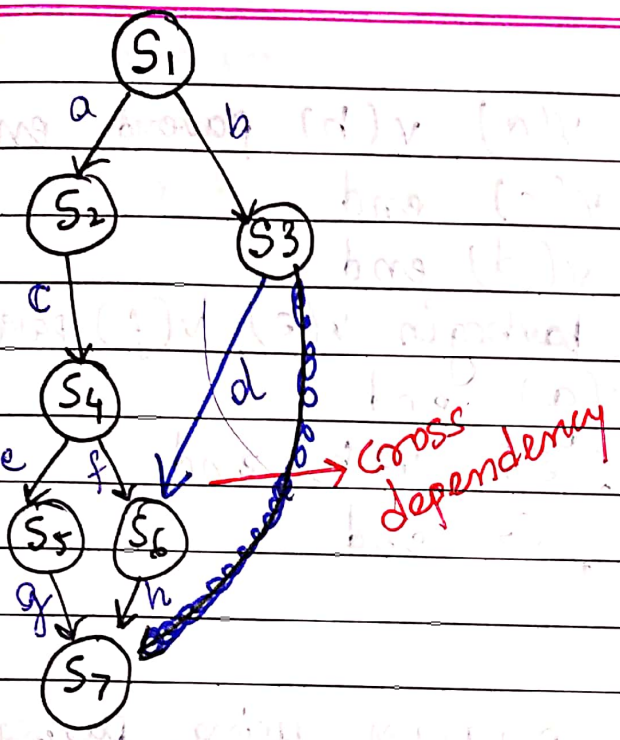

```
begin S1
   parbegin
      S3
      begin
         S2
         S4
            parbegin
               S5
               S6
            Parend
      end
   Parend
      S7
end
```

Ex

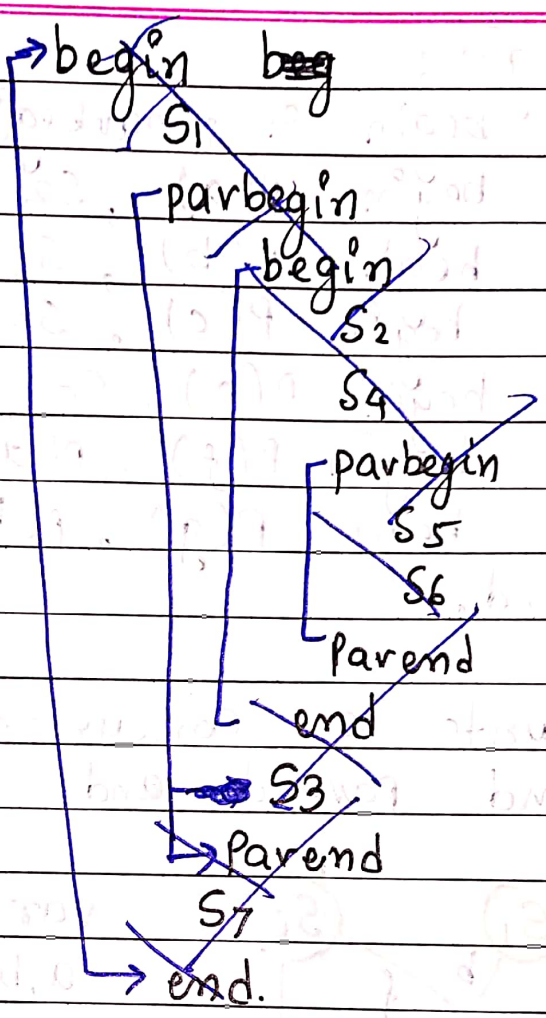

S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
S11


begin
S1
parbegin
begin
S2
parbegin
S5
S6
Parend
end
begin
S3
Parbegin
S7
S8
S9
Parend
S10
end
S4
Parend
S11
end.

Ex



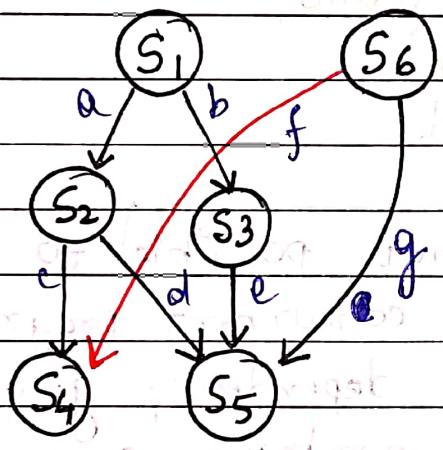

```
begin
    S1
    parbegin
        S3
        begin
            S2
            S4
            parbegin
                S5
                S6
            parend
        end
    parend
    S7
end
```



It is **not possible** to write concurrent programming for all dependency graph using parbegin and parend. But it is very much possible with help of semaphore operation.

parbegin :
begin S1 ; parbegin V(a), V(b) parend end
begin P(a) , S2 , V(c) end
begin P(b) , S3 V(d) end
begin P(c) , S4 , parbegin V(e), V(f) parend e
begin P(e) , S5 , V(g) end
begin P(f) , P(d) , S6 , V(h) end
begin P(g) , P(h) , S7 end
parend.

Ques write a concurrent program using parbegin and parend and semaphore.

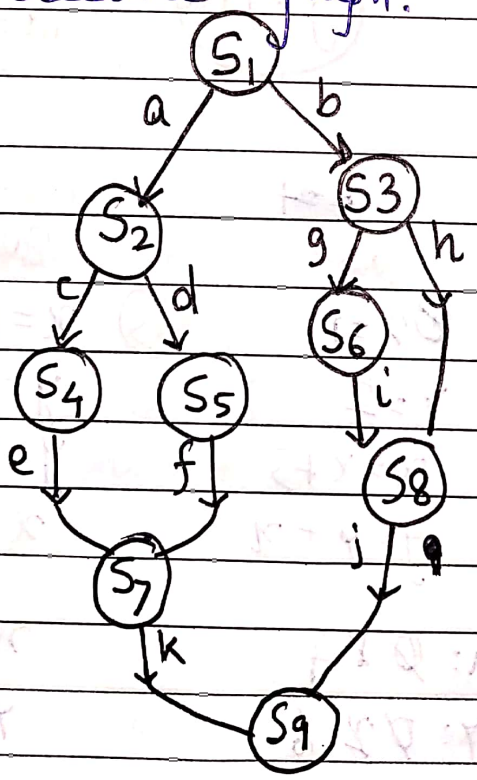



```
var
a,b,c,d,e,f,g : Semaphore ;
begin
co-begin                          ↗ Not mandatory to write
 begin S1 ; parbegin V(a); V(b); parend
 begin P(a); S2 ; parbegin V(c); V(d) end pe, e
 begin P(b); S3 ; V(e) end
 begin P(c) P(f) ; S4 end
 begin P(d) P(e) ; S5 end
 begin S6 , V(f) , V(g) end
coend
end
```

Ques Consider following concurrent programming.

```
var
    a, b, c, d, e, f, g, h, i, j, k : Semaphore
begin
  cobegin
      begin  S1  V(a)  V(b)  end
      begin  P(a)  S2  V(c)  V(d)  end
      begin  P(c)  S4  V(e)  end
      begin  P(d)  S5  V(f)  end
      begin  P(e)  P(f)  S7  V(k)  end
      begin  P(b)  S3  V(g)  V(h)  end
      begin  P(g)  S6  V(i)  end
      begin  P(h)  P(i)  S8  V(j)  end
      begin  P(j)  P(k)  S9  end
  coend
end.
```

Draw the precedence graph.

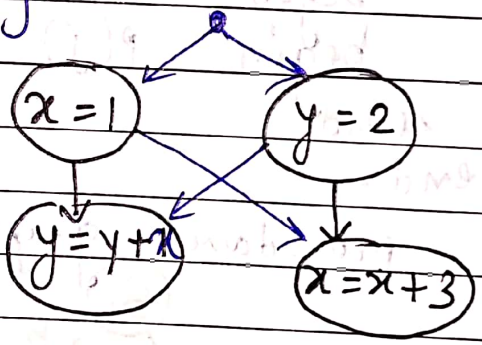


Ques int x=0, y=0;

```
parbegin
    begin
        x = 1 ;
        y = y+x;
    end
    begin
        y = 2
        x = x+3
    end
parend.
```

What can be the final values of x, y after running concurrent program.

x = 0 y = 0



3

I. x=1 y=2
II. x=1 y=3 ✓
III. x=4 y=6 ✓

case 1

① x = 1
y = y + x
y = 2
x = x + 3

x : 0 ✗ 4
y : 0 ✗ 2

case 2

② x = 1
y = 2
x = x + 3
y = y + x

x : 0 ✗ 4
y : 0 ✗ 6

case 3

③ x = 1
y = 2
y = y + x
x = x + 3

x : 0 ✗ 4
y : 0 ✗ 3

Case 4 | Case 5 | case 6

④ y = 2
x = x + 3
x = 1
y = y + x

x : ∅ ⊄ 1
y : ∅ ⊄ 3

⑤ y = 2
x = 1
y = y + x
x = x + 3

x : ∅ ✗ 4
y : ∅ ⊄ 3

⑥ y = 2
x = 1
x = x + 3
y = y + x

x : ∅ ✗ 4
y : ∅ ≠ 6.

Q Consider following program :

```
void P( )
{
    A ;
    B ;
    C ;
}
void Q( )
{ D ;
  E ;
}
void main( )
{ parbegin
    P( );
    Q( );
  parend
}
```

Possible output :

(A) A B C D E ✓
(B) D E A B C ✓
(C) A D B C E ✓
(D) D A C E B ✗
(E) E C B D A ✗.

Ques Consider the following code

$x = 0 \quad S = 2$

Process Z	process W	Process X	Process Y
wait(s)	Wait(s)	wait(s)	wait(s)
Read(x)	Read(x)	Read(x)	Read(x)
x = x-2;	x = x+1;	x = x+1;	x = x-2;
write(x)	write(x)	write(x)	write(x);
Signal(s)	signal(s);	signal(s)	signal(s);

what is maximum possible value of x?

(A) −2 (B) −1 (C) 1 (D) 2

Minimum value = −4. ✓

Ques The following two process P_1 and P_2 that share a variable B with an initial value of 2 execute concurrently.

P_1

```
P1( )
{  C = B-1;
   B = 2* C;
}
```

P_2

```
P2( )
{  D = 2xB
   B = D-1;
}
```

what # values B can possibly take? after execution (distinct.

B = 2

1 C = B-1
2 B = 2*C
4 D = 2xB
3 B = D-1

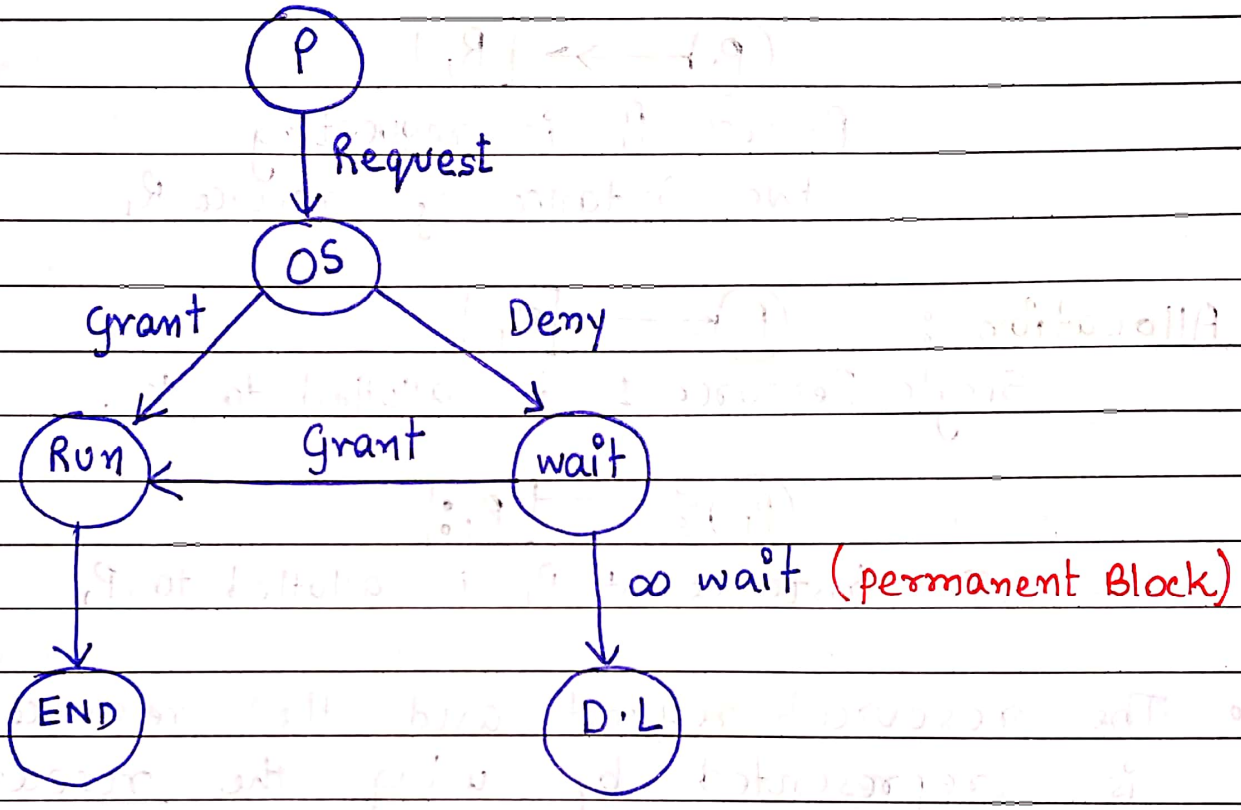
1 C = B-1
4 D = 2xB
2 B = 2*C
B = D-1
3

4 D = 2xB
3 B = D-1
2 C = B-1
B = 2*C
4

4 D = 2xB
3 C = B-1
6 B = 2*C
B = D-1
3

(2), (3), (4) ✓ 3 distinct values.

DEADLOCK

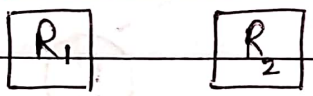


- If two or more processes are waiting on some event to happen which never happed then those process are said to be involved in the deadlock.

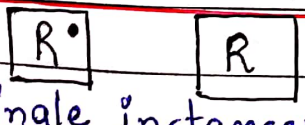
- **Basics of deadlock :**

Process

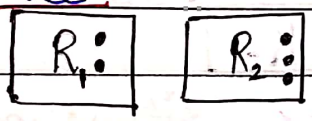
Resource



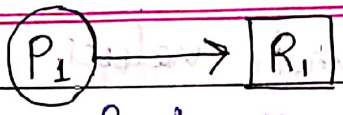
Resource with instance



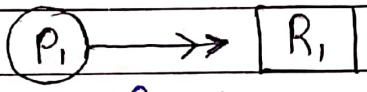
Single instances



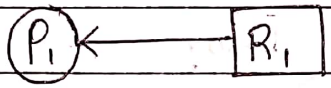
Multiple Instance

- **Requesting :** 

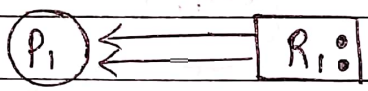
Process P_1 is requesting one instance of resource R_1



Process P_1 is requesting two instance of resource R_1

Allocation : 

Single Resource 1 is allotted to P_1.

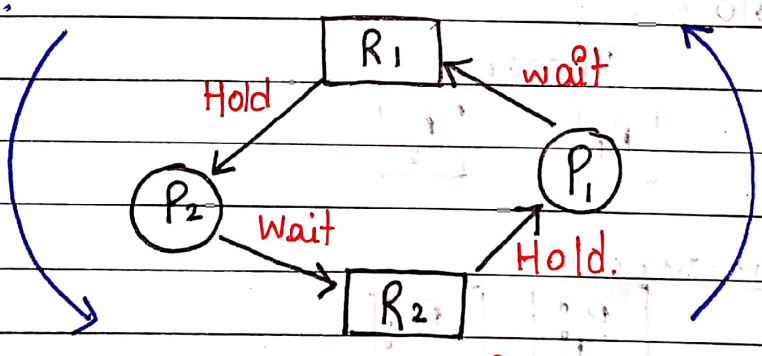


Two instances of R_1 is allotted to P_1

- The resource request and the resource allocation is represented by using the **resource allocation graph.**

Resource allocation Graph :

- It constist of set of vertices and set of edges.



DeadLock situation

- The resource request or release life cycle :

(i) The process will request from resource available in the system.

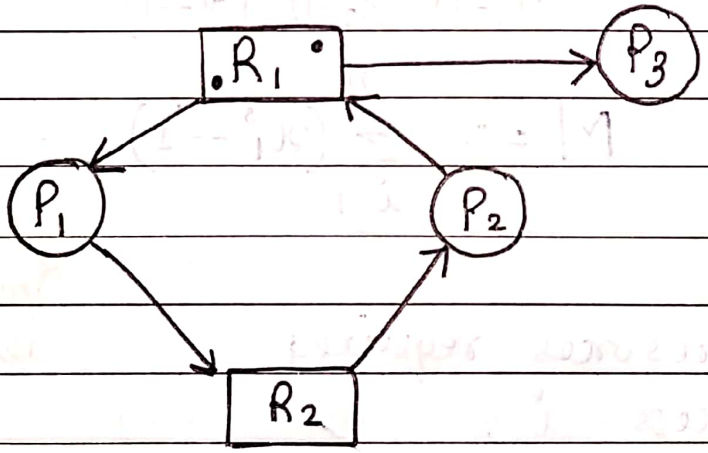
(ii) OS clearly validate the request of the process.

(iii) If the request made by the process is valid then the operating system check for the availibility of the resource.

(iv) If the resource is fairly available then it will be allocated to the process, otherwise the process has to wait.

(v) If all the resource ~~requested~~ required by the process are allocated, then it will go into execution.

(vi) Once the execution of the process is completed, then it will release the resource.



(Assuming P_3 runs in finite amount of time)

This is deadlock free R.A.G.

Q)1) Consider a system with n-process and 6 tape drives. Each process require 2 tape drives to complete there execution. Then what is the maximum value of n which ensure deadlock free operation.

P_1	P_2	P_3	P_4	P_5	P_6 .....
1	1	1	1	1	1

if $n = 6$, chance of deadlock.

$\therefore n = 5$ ✓

Ex. P_1 x_1
P_2 x_2
P_3 x_3
P_4 x_4
P_n x_n

Max unit of resource that ensures deadlock.

P_1 P_2 P_3 ... P_n
(x_1-1) (x_2-1) (x_3-1) (x_n-1).

$$M = \sum_{i=1}^{n} (x_i - 1) = \sum_{1}^{n} x_i - n$$

P_i = process i

x_i = No of resorces required by process i.

Rmax that ensure deadlock.

Ex Minimum no of unit of resource that ensure no deadlock.

$$m = \sum_{1}^{n} x_i - n + 1$$

$m + n > \sum_{1}^{n} x_i$ (Deadlock)

Ex2: Consider a system with 3 process P_1, P_2 and P_3. Each require 2 unit of resource to complete the execution. What is the minimum number of resource required to ensure deadlock free operation.

P_1	P_2	P_3	
1	1	1	1

Note: $\left[\sum_{i=1}^{n} x_i < m + n\right]$ Deadlock Free.

- $\sum_{i=1}^{n} x_i$ → Total Demand
- m → No of Resource
- n → No of Processes.

Example 1: $2n < 6 + n$

$n < 6$ $\quad n = 5$ //

Example 2: $2 \times 3 < m + 3$

$m > 3$. $\quad m = 4$. //

Ques.) Consider the system with n process and six tape drive. If each proces require 3 tape drive to complete its execution. What is max value of n that ensure DL free.

$3n < 6 + n$

$2n < 6$

$n < 3$ $\quad \boxed{n = 2}$.

Quec) Consider a system with 3 process P_1, P_2 and P_3, the peak demand for each process for a perticular resource type R is 5, 9, 13 then what is the min number of resource req to ensure DL free operation.

$$(5+9+13) < m+3.$$
$$24 < m$$
$$\therefore \quad m = 25.$$

Ques) A system is having 3 user process P_1, P_2, P_3 where P_1 require 21
P_2 require 31
P_3 req 41
Minimum number of unit of r ensure No deadlock. is.

$$93 < m+3.$$
$$90 < m$$
$$\boxed{m = 91}$$

Ques (i) If there are 100 unit of resource in the system and each Process in the systems require 4 unit of resource

$$3n < 100$$
$$n < 33.3$$
$$\boxed{n = 33}$$

for Deadlock free.

(ii) Min number of process that ensure deadlock.
$n = 33 + 1 = 34$ ✓

Necessary Condition : Deadlock may or may not be possible.
- Sufficient condition: Deadlock surely, never be possible

Necessary : There are 4 necessary condition of occurrence of deadlock,
(i) Mutual Exclusion
(ii) Hold and wait
(iii) No preemption (No release)
Sufficient → (iv) Circular wait.

Mutual execution : By this condition there must exist atleast one resource in the system which can be used by only one process at a time. If there exist no such resource then deadlock will never occur. Resource have been allocated only one process or it is to be freely available. There should be 1-1 relation-ship between

- Printer is example of resource that can be used by only one process at a time.
- Hold and wait. The process is holding the resource and waiting on some other resource simultaneously.

- **No preemption :** By this condition, once the resource have been allocated to the process, it cannot be preempted.
- The resource has to be released by the process after execution. It is not allowed to forcefully preempt the resource from the process.

- **Circular wait :** The process are circularly waiting on each other for the resource.
- If all other 4 condition exist simultaneously in the system, then definitely there exist a deadlock.

Not only 1, 2, 3, All 4 reasons are must for causing deadlock. All 4 conditions are not independent, because the circular wait includes hold and wait.

SOLUTION FOR DEADLOCK

P A D I

- P → Prevention
- A → Avoidance
- D → Detection
- I → Ignorance.

Deadlock prevention

- **Mutual exclusion :** It is not possible to dissatisfy mutual exclusion always because of shareable and non shared resource. for example, file read by multiple resource. But printer is non-shareable resource. May result data inconsistency.

- **Hold and wait :** Allocate all the resource required by the process before start of execution. But this results in low utilizat" of resource. Ex. P_i is unnecessarily holding a resource which is required only at the end of execution.

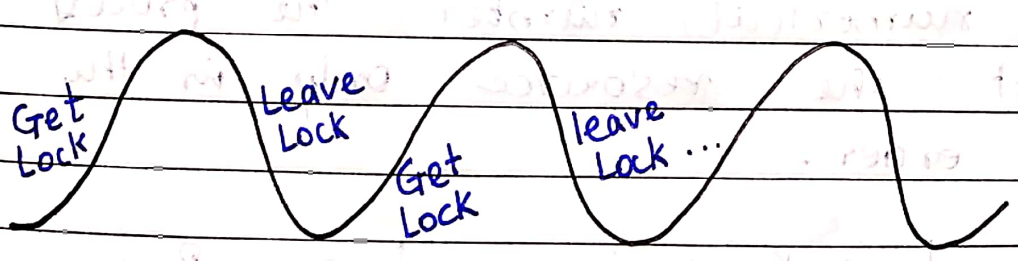
No-preemption :(1)

- The process should release all the existing resource before making the new request. Ex. the process P_1 requires 100 resource to complete the execution,

  Partial allocate: $P_1 \rightarrow 100$. but $P_1 \leftarrow 85$ only

  next time $P_1 \rightarrow 100$ but $P_1 \leftarrow 55$

  P_1 may move in starvation.



- If the above condition is hold to dissalisfy hold and wait then it may go into starvation problem.

No-preemption (2): Let The process P_1 is requesting for resource R_1,

Case (i) If R_1 is free then it will be allocated to process P_1.

Case (ii) If the resource R_1 is not free and if it is allocated to other process P_2

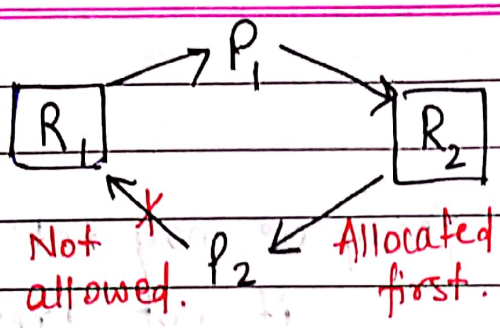
(ii)(a) If Process P_2 is in execution, then the process P_1 has to wait.

(b) If the process P_2 is not in execution and it is requesting some other resource R_2, preempt the resource R_1 from R_2 and allocate to P_1.

- Resource preemption guarrantees the process completion.

Circular wait: The resource will be assigned with the numerical number. The process can request the resource only in the increasing order.

$P_1 \rightarrow R_8$	$P_1 \rightarrow R_6$	$P_1 \rightarrow R_{10}$
(allocate)	(x not allowed)	allowed.



Deadlock Avoidance :

- Deadlock avoidance can be avoided by Bankers algorithm.

Total Available

A	B	C
10	5	7

safe → No deadlock
unsafe → Deadlock.

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

Current Available

A	B	C
3	3	2

Current Need

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

safe sequence- $< P_1, P_3, P_0, P_2, P_4 >$
(Multiple possible) $< P_1, P_3, P_0, P_4, P_2 >$
etc.

- If we can satisfy the remaining need of all the process with the current available resource then the system is said to be safe state. otherwise the system is said to be in the unsafe state.
  - If the system is in the unsafe state then it is possible for deadlock.
  - The order in which we satisfy the remaining need of all the process, is called as safe sequence.
  - The safe sequence may not be unique. We can have multiple safe sequence. the unsafe state purely depend on the behaviour of the process. (unpredictable)
  - Each and every time when the process is requesting for any resource, the banker algorithm will be used to identify if the system is in the safe state or unsafe state.

- If the system is in the safe state, then the request of the process will be guarranted. If system is in ~~[illegible]~~ unsafe state then the request of the process will be denied.

(Ques.) Consider following table ; which Process executes Last?

Total Available

X	Y	Z
5	5	5

Current Available.

X	Y	Z
0	1	2 →

Current Allocation

	X	Y	Z
P_0	1	2	1
P_1	2	0	1
P_2	2	2	1
	5	4	3

Current Need.

	X	Y	Z
P_0	1	0	3
P_1	0	1	2 ✓
P_2	1	2	0
	0	1	0

P_1 P_0 P_2

3 2 0
2 2 1
5 4 1

0 1 2
+ 2 0 1
2 1 3
+ 1 2 1
3 3 4
2 2 1
5 5 5 ✓

Ans P_2

Current Avail = Avail + ~~Need~~ Allocated.

Current Need = Max Need - Allocated.

Ques.) Consider following table:

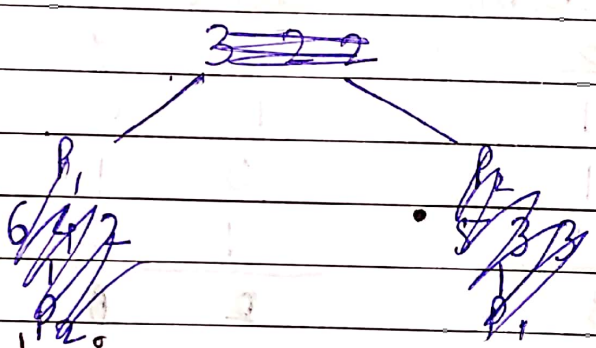
	Allocation			Max Need			current need		
	X	Y	Z	X	Y	Z	X	Y	Z
P_0	0	0	1	8	4	3	8	4	2
P_1	3	2	0	6	2	0	3	0	0
P_2	2	1	1	3	3	3	1	2	2
	5	3	2						

Current Available

X	Y	Z
3	2	2

Total Available

8	5	4



Extra requests:

REQ1: P_0

X	Y	Z
0	0	2

current Need.

X	Y	Z
8	4	4
5	0	0
1	2	2

Req Q2 : P_1

X	Y	Z
2	0	0

3 2 2
2 1 1
5 3 3
3 2 0
8 5 3.

Current Need

X	Y	Z
8	4	2
3	0	0
1	2	2

5 3 3
8 5 3.

current Need Q_2 only

X	Y	Z	
8	4	2	✓
5	0	0	
1	2	2.	

Only Q_2 can be fullfilled.

Ques) A system has 4 process

Allocated					Max					Need [2, 0, 0]				
X	Y	Z	P	Q	X	Y	Z	P	Q	X	Y	Z	P	Q
1	0	2	1	1	1	1	2	1	3	0	1	0	0	2
2	0	1	1	0	2	2	2	1	0	0	2	1	0	0
1	1	0	1	1	2	1	3	1	1	1	0	3	0	0
[	1	1	1	0	1	1	2	2	0	0	0	1	1	0

If Available = X Y Z P Q

0 0 X 1 1

what is smallest value of X for which this is safe state ?

silly mistake

~~X = 3~~

0 0 3 1 1

1 1 0 1 0

1 1 2 2 1

1 1 1 1 0

2 2 4 3 2

~~X = 3~~ ✓

X = 2

0 0 2 1 1

1 1 1 1 0

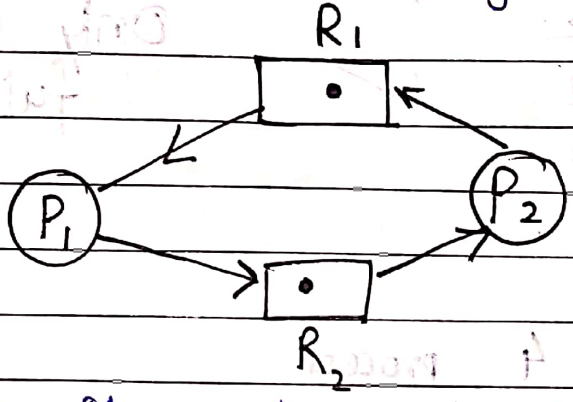
1 1 3 1 1

1 1 0 1 1

2 2 3 2 2

✓

Question:) Consider the following Graph:

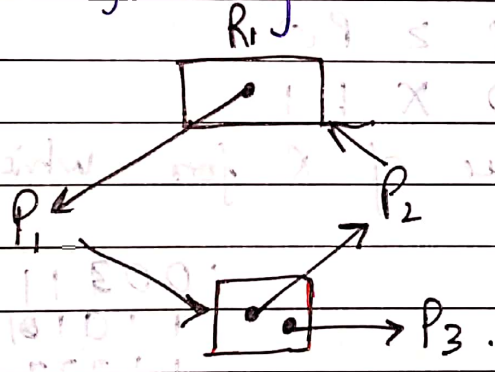


Q Find if system is in a deadlock state or else find a safe sequence.

	Allocation		Need		available	
P_1	1	0	0	1	0	0
P_2	0	1	1	0		

Deadlock.

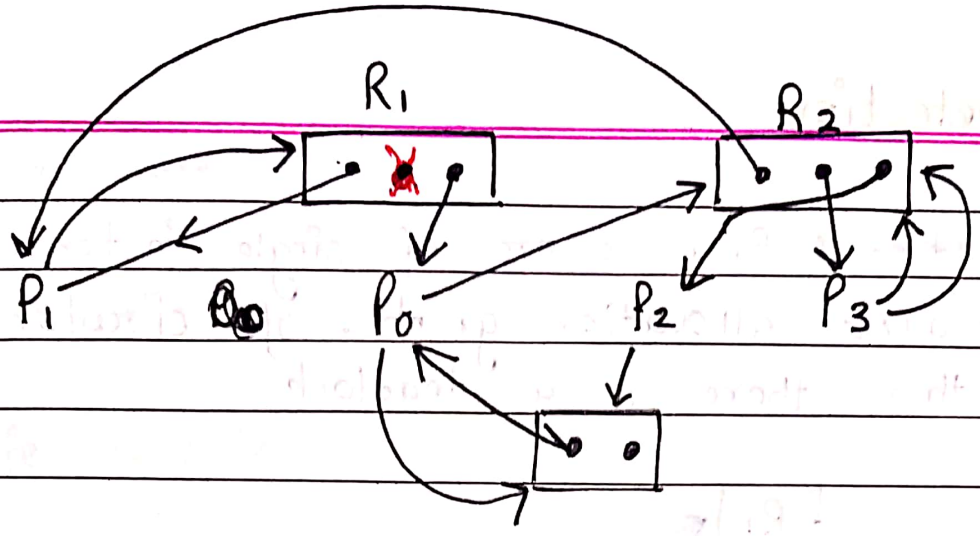
Ques Consider following



safe state ✓

P_3 P_1 P_2 ✓

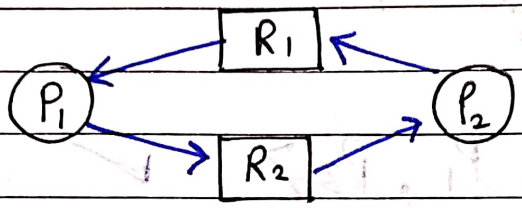
Ques) Consider following graph :



< P2 , P0 , P1 , P3 > ✓

Deadlock Detection

- Deadlock Detection : Resource are of single instanee type.
  ↳ Draw resource allocation graph. If circular wait cycle exist then there is a deadlock.



- If all resources are of single instance type then the cycle in RAG is necessary and sufficient condition for the occurance of deadlock.

- If all the resource are not single instance type Then the cycle in the resource allocation graph is just a necessary condition but not a sufficient condition for deadlock.

- The resources are of multiple instance type.

Ex.

	Allocate	Need	Available
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	3 1 3
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

$< P_0, P_2, P_1, P_3, P_4 >$ No deadlock.

- What happens if the process P_2 is requesting for additional resources 0 0 1 ? Then the system will go to deadlock.

< Unsafe state > : Possibility of Deadlock.

- unsafe state is unpredictable. If it is totally dependent on behaviour of process. Means changing nature of their remaining need, if remaining need is constant (not changing), surely deadlock will occur.

- If the remaining need is changed, it may come again in some state.

Deadlock Recovery :

- No traditional OS use this deadlock algorithm because it is time consuming and take more space.
- Real time OS use deadlock algorithm,
  (i) Kill in the process
  (ii) Resource preemption.

(i) Killing the process : Kill process which are involved in the deadlock. Kill one after the other. Kill one process which is involved in the

deadlock and again check for deadlock by using deadlock detection. If deadlock exist, then apply again, kill one more process, then again apply deadlock. detection and so on.

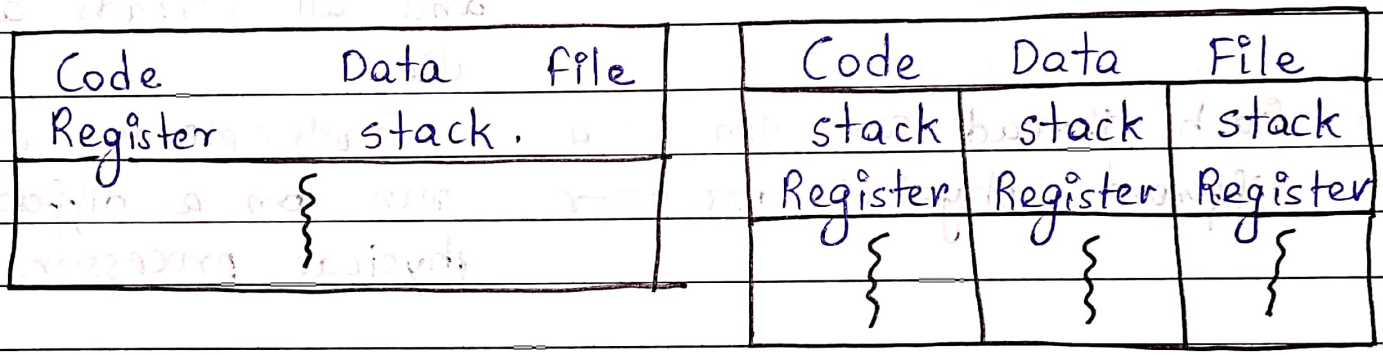
- on what basis do we delete a process?
  (i) Low Priority Process.
  (ii) Number of resource a process is holding
  (iii) % of completion.
  (iv) Number of resource the process is requesting

(ii) Resource Preemption: Resources will be preempted from the process that are involved in the deadlock, and the preempted resource will be allocated to other process, so that there is a possibility of recovery, from deadlock.

- If the resource preemption is followed to recover system from the deadlock. Then it may be possible for the process to go into starvation.

THREADS

- A thread represent an abstract entity that execule the sequence of instruction.
- It has its own set of CPU registers.
- It has its own stack.
- There is no thread specific heap. or data segment.



- Thread is a light weight process (less number of instruction)
- Thread is just like a process having light weight

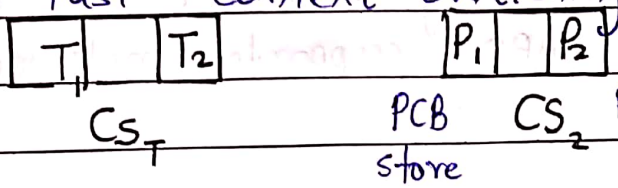
THREAD	PROCESS
• Thread has no data segment or heap.	• Process has everything
• Thread cannot live on its own. It requires a process	• There must be atleast one thread in a process
• There can be more than one thread in a process.	• First thread is called main and has process stack.

THREAD	PROCESS
• Thread within a process share code data and file but have its own set of stack and register	• Process does not usually share its code, data or file with other processes.
• If a thread die, stack is reclaimed.	• If a process die, the resources are reclaimed and all threads also die.
• Each thread can run on a different physical processor	• Each process can run on a different physical processon
• Inter thread communication is via memory.	• Inter Process Comm is via data copy.

Advantage of Thread :

- Responsiveness : If the process is divided into multiple threads, then if one thread is completed, its execution then the output will respond immediately. This response will be very fast compared to the response of process. Hence the thread will improve the responsiveness.

- Advantage : Fast context switching.



- Context switching time between thread is very less than context switching time between the process. Because the thread will have less context compared to the context of process.

- Effective utilization of Multiprocessor system : If the process is divided into multiple threads then the different threads can be scheduled onto the different processor. So that the process execution will be faster.

- Resource sharing : Resource like code, data, file and memory will be shared among all the threads between the process. The stack and register cannot be shared between the thread. Every thread will have its own stack and register.

Thread = stack + set of CPU reg + CPU time

Process = 1 thread + PCB + code segment + Data + Memory.

Multithread Process = n threads + PCB + Code + Data + Memory

n process = n PCB + n code + n data.

- Economical - The implementation of a thread does not require extra hardware. Java API supports implementation of threads.

- Throughput - Enhanced throughput of the system. If the process is divided into multiple threads, if it consider multiple threads and divided into one job, then the number of jobs completed for unit time will increase and hence the throughput of the system will advance

- Type of threads - There are two level of threads.
  (i) user level (ii) Kernel level,

User Level Threads	Kernel Level Threads.
• ULT are implemented by user or programmer.	• KLT are implemented by OS
• OS does not know about the ULT or OS view the ULT as a process only.	• KLT are recognised by OS.
• If one ULT is performing, blocking system call, then the entire process will be blocked.	• In one KLT is perform blocking system calls the another thread will continue execution.
• ULT are dependent	• KLT are independent
• Implementation of ULT is easy.	• Implementation of KLT is complex.
• Has less context	• Has more context
• No H/w support needed	• H/w support needed.

Ques : Which of the following is not supported by the thread?

(A) stack (B) file (C) Address space (D) MQ.

Memory Management

- The main functionality of memory management in allocating and deallocating the memory to the Process The main goal of memory management is efficient utilization of memory by minimizing the internal and external fragmentation.

Memory Management Technique:

(i) Contiguous
  - ↳ Fixed partition
  - ↳ variable partition

(ii) Non-contiguous.
  - ↳ Paging
    - ↳ Multilevel Paging
    - ↳ Segmentation
    - ↳ Inverted Paging
    - ↳ Segmented Paging.
    - ↳ Demand Paging.

- Continuous Memory Allocation Technique:
  It is of two types : (Static) & (Dynamic).

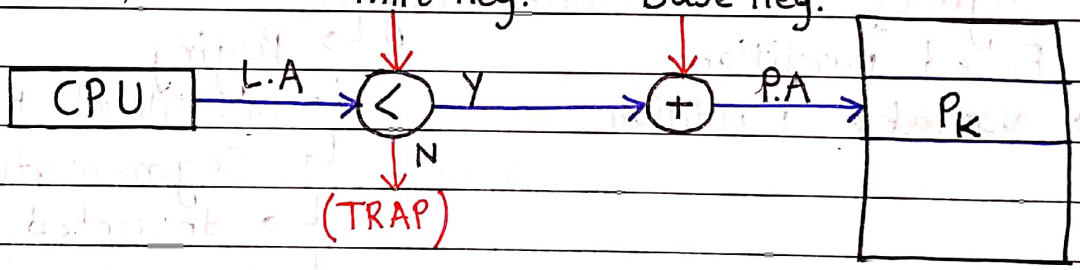
(1) Fixed Partition : Static Partition is a fixed size partition scheme.
→ 1 partition - 1 process.
→ Degree of multiprogramming is fixed.

(Size may not be equal but sizes are fixed)

- In this technique, main memory is predivided into fixed size partitions
- Size of each partition is fixed and cannot be changed.
- Each partition allow only one process at a time. Degree of multiprogramming is restricted by number of partition
- Maximum size of process is restricted by maximum size of partition

STRUCTURE

- Every process is associated with the limit Registers.
- Lower limit registers contain starting address of partition.
- upper limit registers contain ending address of the partition.



Advantage: Implementation of this scheme is simple.

- Overhead of processing is also low.
- It support multiprogramming.
- No special hardware is required.
- Only one memory access is required, which reduce access time.

Disadvantage: No single program may be reduceded to the size of the largest partition, in a given system.

- It does not support dynamic datastructures, such as stack, queues, heap.
- It utilize memory inefficiently.
- The degree of multiprogramming is limited and equal to the number of partition.
- It suffers from both internal fragmentation and external fragmentation.

Consider below memory allocation:

		Process
A	10 KB	A - 10 KB
	15 KB	B - 12 KB
C	25 KB	C - 20 KB
D	8 KB	D - 5 KB.
B	12 KB	

internal Fragmentation } Free space within partition.

Variable Partition or Dynamic Partition: It is a variable size partitioning scheme. It perform the allocation at runtime, when a process arrive, a partition of size equal to the size of Process is created. Then that partition is allocated to the process.

A	10 KB	B. Freed ⟹	A
B	12 KB		12 KB
C	20 KB		C
D	5 KB		D
/// 3 KB /// → Free space			3 KB

E.F

- External Fragmentation (E.F) may occur because of contiguous allocation to a process. Free partition are not contiguous.

- In variable partition scheme, initially the main memory will be single continuous free block.
- Whenever the request by the process come, partition will be met.

Advantage : It does not suffer from internal Fragmentation.
- Degree of multi programming is Dynamic.
- There is no limitation on the size of Process.

Disadvantage : It suffers from external fragmentation, allocation and dellocation of memory is complex.
- Compaction : To avoid the external fragmentation, Compaction technique is used. Moving all the process towards the top or towards the bottom to make the free available memory in a same contiguous space.
- The compaction is undesireable to implement because it interrupt all the running process in the memory.
- To implement non-contiguous memory management technique to avoid problem of external fragmentat

Partition Allocation Methods:

- When more than one partition is freely available to accomodate the request of the process then decision making of selecting a partition will be done by partitioning allocation method:

- **First Fit:** Allocate the process in a partition which is the first sufficient partitioning from the top of the memory.

- **Best Fit:** Allocate the process in a partition which is the smallest sufficient partition in the memory.

- **Worst Fit:** Allocate the process in a partition which is the largest sufficient among the free available partition. To find out the largest sufficient partition, it is also required to search for free available partition in the memory.

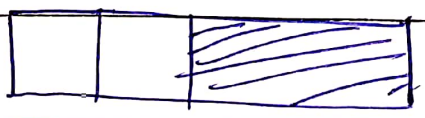
- **Next Fit:** It also work like first fit but it will be searched for first sufficient partition from the last allocation point.

Difference between Internal and External Fragments

	Internal	External
Basic		
	occurs when fix size memory blocks are allocated to a process	occurs when variable size partitions are allocated to a process
	• when the memory assigned to the process is larger than the memory required by the process. This create free space in the allocated block.	• When the process is removed from the memory, it creates free space in the memory causing externa fragmentation.
	• Solution of internal fragmentation is that memory must be partitioned into variable size.	• Solution is compactio paging, and segment -ation.

Ques A 1000 KB memory is mapped using variable partition but no compaction. It currently has two partitions of size 260 KB and 200 KB The smallest allocation request in kilobytes that could be denied is for

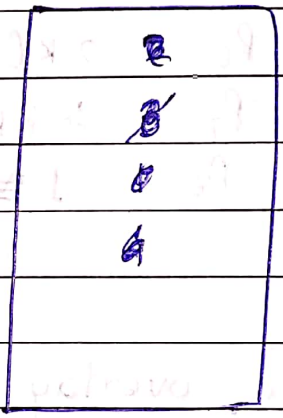
(A) 151 (B) 181 (C) 231 (D) 541.



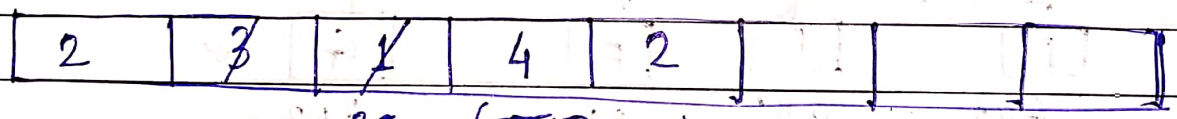
Ques:) Consider six memory locations of size 200, 400, 600, 500, 300, 250 KB. These partition need to be allocated to 4 process: 357, 210, 468 on 491 in that order. If the best fit algorithm is used. which partition are not allocated? Ans. 200 & 300.

Ques:) Given 32 MB for allocation and the following event of memory allocation occurs.

P_1 = 2 MB
P_2 = 3 MB
P_3 = 5 MB → External fragment
P_4 = 1 MB
P_5 = 4 MB
P_2 – finish – Release.
P_6 – 2 MB
P_4 – finish & Release.



Assume fixed and equal size partitions. Calculate m/o of total 8 partition. (4 MB each) wasted.



∴ 32 – (7)
= 24 MB free.

Fragments : 2 + 2 = 4 MB ∴ 28 MB wasted.

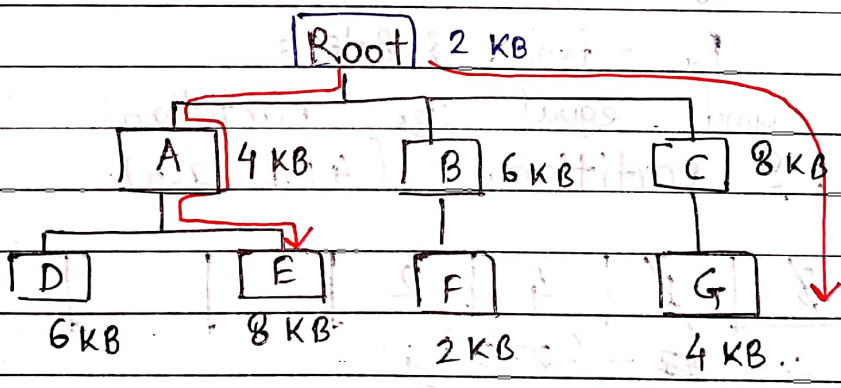
wasted memory = Internal Fragments + External fragment
= 24 + 4 = 28 MB

Q.) Let a system has 32 KB space and divided into 8 partition of fixed size. If the following process req. Compute memory wasted if any.

P_1	3 KB	1
P_2	4 KB	0
P_3	1 KB	3
P_4	2 KB	2
P_5	3 KB	1
P_6	2 KB	2
P_7	4 KB	2
P_8	1 KB	3

→ 14 — 14 KB wasted due to internal frag.

Ques.) The overlay tree for a program given below.



What will be size of partition in physical memory required to load and run this program?

(A) 12 KB
(B) 14 KB
(C) 10 KB
(D) 8 KB

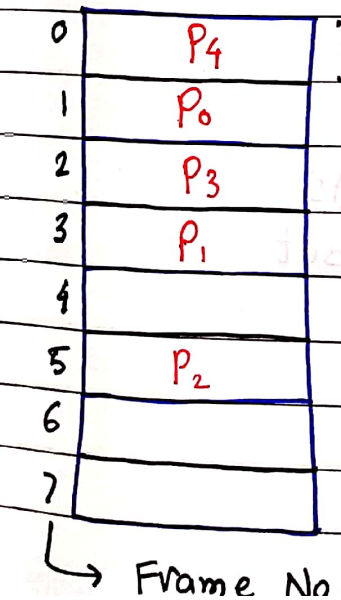
2 + 4 + 8 = 14 KB minimum required.
(Root) (A) (E) (G) (C)

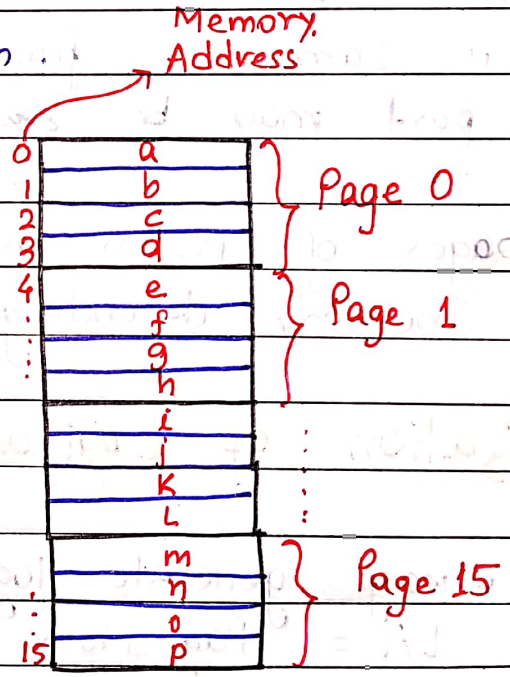
- Maximum space that can be needed at a time is 14 KB.
- Non contiguous memory locatio

Non-contiguous memory allocation :

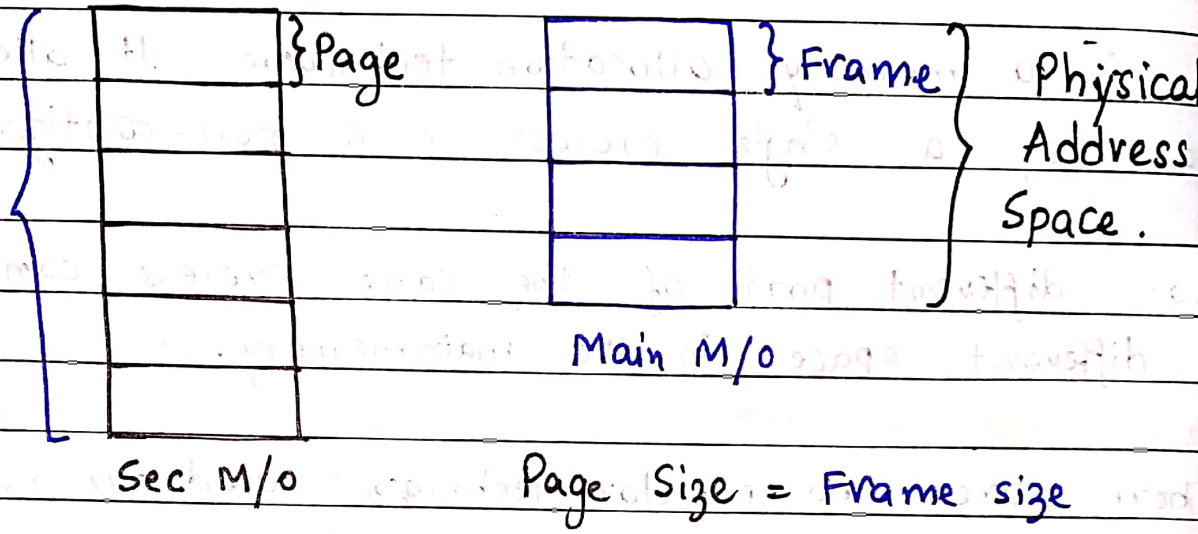
- It is a memory allocation technique. It allow to store part of a single process in a non-contiguous manner.
- Thus different part of the same process can be stored at different space in the main memory.
- There are two popular techniques used for non-contiguous memory allocation.
  (i) Paging
  (ii) segmentation.

Ex Program - 64 B
Page size - 4 B.
Pages - $\frac{64}{4} \Rightarrow 16$ pages.





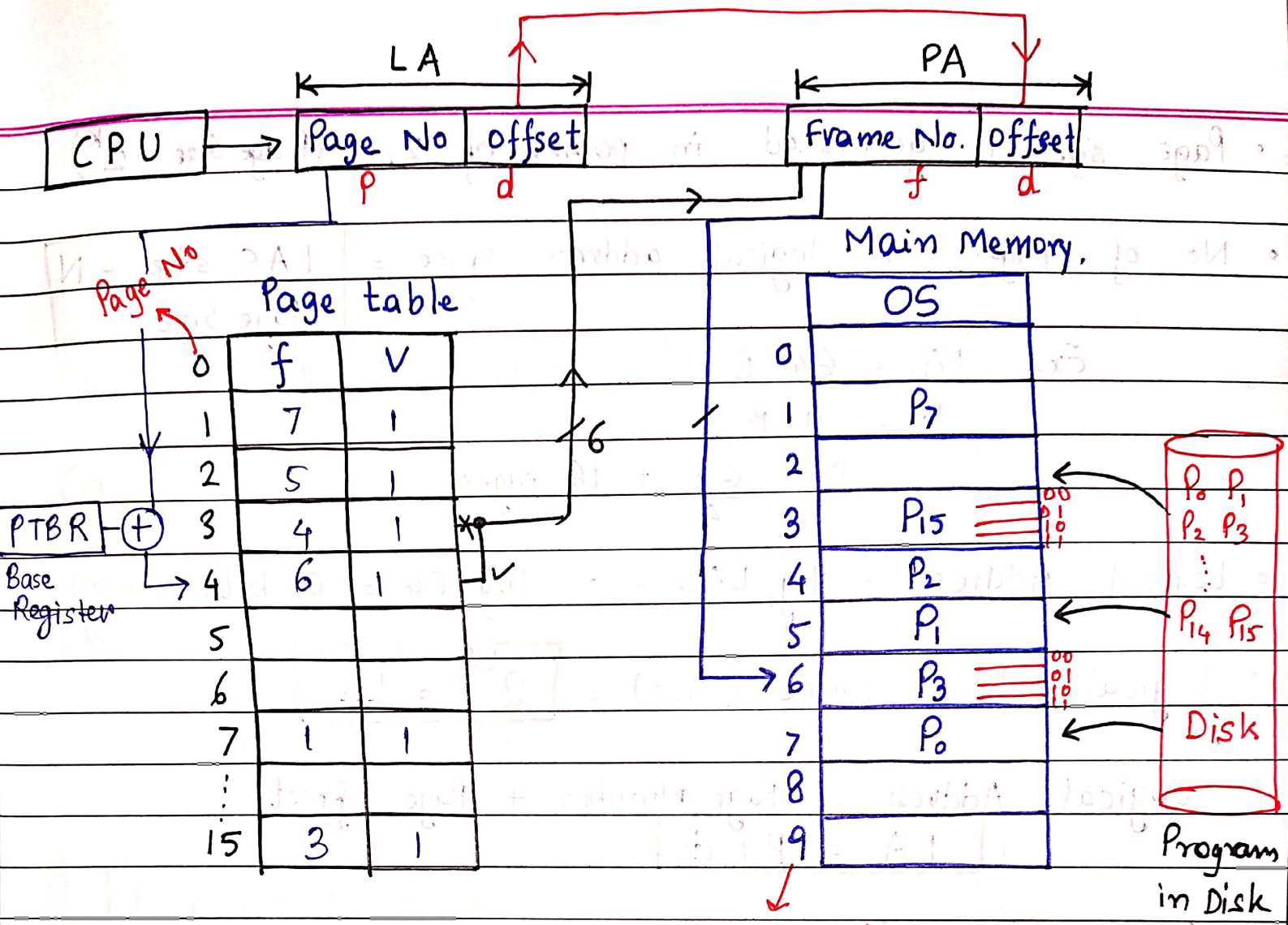
- Paging is a fixed size partitioning scheme. In paging, memory is divided into fixed size partition. The partition of secondary memory are called as pages. The partition of main memory are called as frames.



- Each process is divided in pages where size of each part is same as frame size. The size of the last part may be ~~same~~ less than page size.
- The pages of process are stored in the frame of main memory depending upon their availibility.

Organization of Logical Address Space.

- CPU always generate logical address for LAS.
  LA = Page No x Pagesize + Word offset
- LAS is divided into equal size pages.
- Pages are always fixed size units.



Page entry = # Page in Program

$$\#\text{Pages} = \frac{\text{Program Size}}{\text{Page size}}$$

$d = \log_2(\text{Page Size})$ $\quad p = \log_2 \#\text{Pages}$

0011	01

Page 3 offet 1

⇓

Frame 6th offset 1.

110	01

- Page size is generated in power of 2. (Page Size = 2^k)
- No of pages in logical address space = $\frac{\text{LAS size}}{\text{Page Size}} = N$

Ex. LAS = 64 B

PS = 4 B.

$\therefore N = \frac{64}{4} = 16$ pages.

- Logical Address = $\log_2 LAS = \log_2 64 = 6$ bits.
- Logical Address Space (size) = $2^{LA} = LAS$
- Logical Address = Page Number + Page offset.

$LA = P + d$

- $P = \log_2 N$ where N is # pages in LAS.
- $d = \log_2 P_S$ where P_S is page size.

Organization of Physical Address Space:

- Physical Address space is divided into equal size frame. Frame size is always divided into page size.

$PA = \log_2 PAS$

$PA = f + d$ (offset)

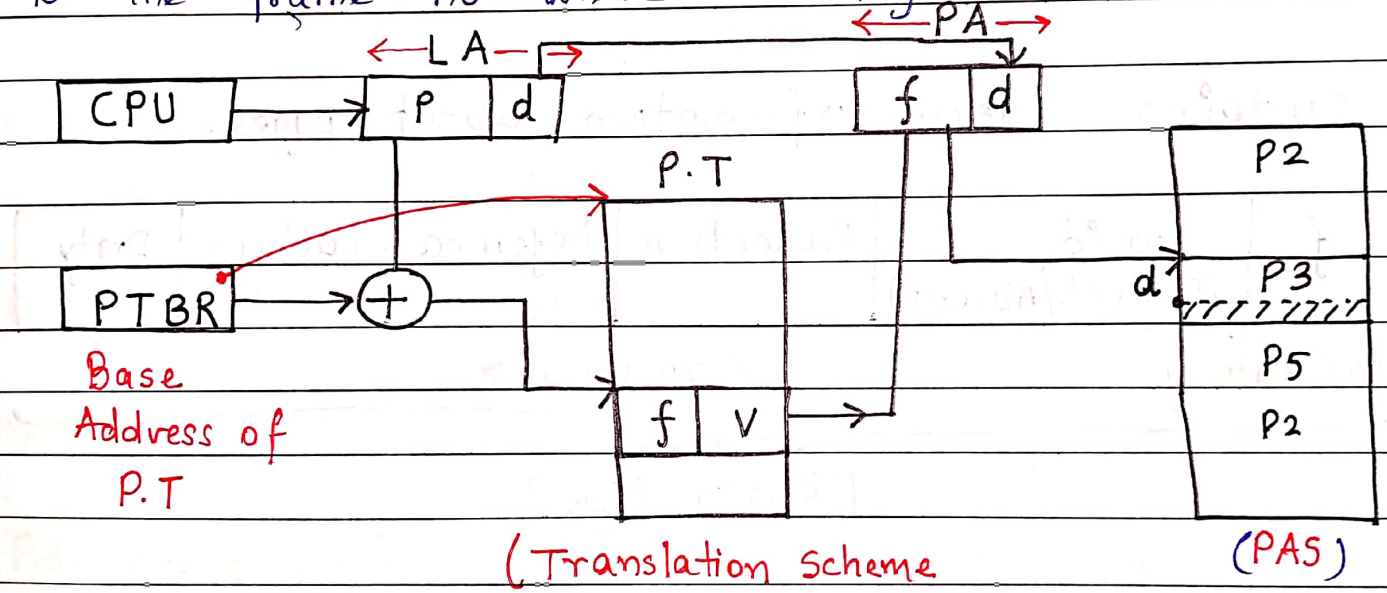
(frame No)

M = # frames in PAS = $\frac{\text{PAS (size)}}{P_s \text{ (Page size)}}$.

$f = \log_2 M$ $d = \log_2 P_s$.	$M = 2^f$ $P_s = 2^d$

Organization of Page Table :

- Data structure that mapps the Page No referenced by CPU to the frame no where that page is stored.



- Page table is stored in the main memory. Number of entries in the page table is equal to number pages a process is divided into.

- Page table base Register contain the base address of page table.

- Each process has its own independent page table.

Page Table Size = # entries x Page Entry Size

$$PT_s = N \times e_s$$

$$P_{Ts} = \frac{LAS \times e_s}{P_s}$$

OPERATING SYSTEM

Page table entry:

- It contains several information about pages.

1 entry	f	valid (Present/Absent)	Protection	Reference	caching	Dirty

Mandatory — < optional >

[Entry Size]

- The information contained in the page table entry is changed from OS to OS.
- Most important info is frame number.
- It is also called as valid / invalid and indicate if page is present in the memory or not.
- Protection bit : Also called as read or write bit. If only read operation is allowed to be performed and no writing is allowed, then th

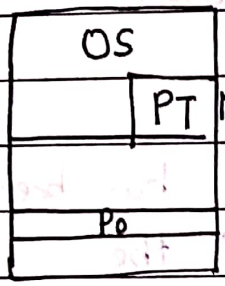
bit is set to zero. It specifies the permission of read or write on pages. If both read/write is allowed to be performed, then this bit is set to 1.

- Reference bit : It specify whether the page has been referenced in the last clock cycles or not. If the page has been set referenced recently, it is set as 1.
- Caching : Enable or disable information. This bit enables or disable the caching of pages. Whenever freshness in the data is required, then caching is disabled using this bit. If caching of the page is disabled, then it is set one.
- Dirty bit. It is also known as modify bit. This bit specifies if that page is modified or not. If the page has been modified then set 1, otherwise 0.
- Advantages of Paging : It allows to store part of single process in a non-contiguous fashion.
- It solves problem of external fragmentation.

Disadvantage : suffers from internal fragmentation. on average P/2 (last page is half full).

- There is an overhead of maintaining a page table for a process.
- The time taken to fetch the instruction increases.

since now - two memory access are required.



Effective Access time = PTA time + Page Access time
= m + m

EAT = 2m time.
(NO TLB)

Ques) Consider a system where Number of pages N = 2K
Page size = 4 KW
PA = 18 bits
cal LA and No of frame in PAS.

PAS = 2^{18} bits. # frames = $\frac{2^{18}}{4\,KW}$ = 64

LA = P + d.
$\log_2 N + \log_2 P_s$.
= 11 + 12
= 23 bits.

Ques) Consider a system with logical address space of 128 MW and PA = 24 bits. PAS is divided into 8K frames. Then what is page size and # pa in LAS.

LAS = 2^{27} word.
PAS = 2^{24} bits.
$m = 8K = \frac{2^{24}}{P_s}$
$P_s = 2^{24-13} = 2^{11}$ word. ✓

No of pages in LAS = $\frac{2^{27} \text{ words}}{2^{11} \text{ words}} = 2^{16}$ pages.

(Ques.) Consider a system with logical address 32 bits and physical address space 64 MB. P_s = 4 KB. Memory is Byte addressible.

e_s = 2 B

what is PTs?

LA = 32 bits.

PAS = 64 MB.

P_s = 4 KB.

Pages = $\frac{2^{32}}{2^{12}} = 2^{20}$ pages. × ~~Bytes~~

e_s = 2 B.

$\therefore$ 2×2^{20} bytes PTs.

2 MB Page table size.

(Ques.) Consider a system with logical address space = PAS = 2^{16} byte. P_s = 512 bytes M/o is byte addressible. e_s = 2 B if Page table entry contain 1 V, 1 R, 3 P, 1 d How many bits are still available in the page table entry, to store paging information?

frames = $\frac{2^{16}}{2^9} = 2^7$ frames.

16 − (7 + 1 + 1 + 3 + 1) = 3 bits remaining.

OPERATING SYSTEMS

Ques.) Consider a system, LAS = PAS = S bytes
M/o is byte addressable. P_s = P bytes
PTes = e bytes.
what is optimal value of page size by minimizing memory overhead of maintain the page table size and internal fragmentation.

(A) $P = \sqrt{2se^2}$
(B) $P = \sqrt{2se}$
(C) $P = \sqrt{2s^2e}$
(D) $P = \sqrt{2(se)^2}$

Page size = # Pages x es. # pages = $\frac{S}{P}$

= $\frac{S}{P} \times e_s$

Page size = $\frac{Ses}{P}$

M/o = PTs + Internal frag.

= $\frac{Se_s}{P} + \frac{P}{2}$.

M/o = $\frac{2e_sS + P^2}{2P}$.

$\frac{d(M/o)}{dp} = -\frac{Se_s}{P^2} + \frac{1}{2} = 0$

$-\frac{Se_s}{P^2} = -\frac{1}{2}$.

$P^2 = 2Se_s$. $P = \sqrt{2Se_s}$

Ques.) In a paging scheme, virtual address space is 4 KB and es = 8 KB what should be the optimal page size?.

$$P = \sqrt{2^{12} \times 2^{13} \times 2} = \sqrt{2^{26}} = 2^{13} \text{ bytes.}$$

$$P = 8 \text{ KB.}$$

Ques.) In a paging scheme, LAS = 16 MB. es = 2 B. what is optimal Page size.

$$P = \sqrt{2 \times 2^4 \times 2^{20} \times 2} \text{ Bytes.}$$

$$= 2 \times 2^2 \times 2^{10}$$

$$= 8 \times 2^{10} \text{ Bytes}$$

$$P = 8 \text{ KB.}$$

Ques.) Consider a single level paging scheme. VAS = 4 MB. P = 4 KB. what is the max Pe size?. Entire PT fits in one Page.

$$8 \text{ KB} = \sqrt{\frac{4 \text{ MB}}{4} \times 2 \times es}.$$

$$es = \frac{8 \times 4 \text{ KB} \times \text{KB}}{4 \times 4 \text{ MB}}$$

$$es = 8 \text{ Bytes.} \quad //$$

PTs = # entries * es

$PTs \leq Ps.$

entries * $es \leq Ps.$

$N * es \leq Ps.$

$$e_s \leq \frac{P_s}{N}$$

$$e_s <= \frac{P_s}{LAS/P_s}$$

$$\boxed{e_s \leq \frac{(P_s)^2}{LAS}}$$ when entire PT fits in one page.

$$e_s \leq \frac{4KB \times 4KB}{4MB}$$

$$e_s \leq 4B$$

$$\boxed{e_{max} = 4B}$$

Ques.) Consider single level paging scheme.
VAS = 4GB
P_s = 128 KB.

$$e_s \leq \frac{(128 \times 128)^2 \; KB \times KB}{4 \; GB}$$

$$e_s \leq \frac{2^{14} \times 2^{20}}{2^2 \; 2^{30}}$$

$$e_s = 2^{34-32} = 4B \; \checkmark$$

Ques.) Consider a single level paging scheme. VAS = 256 MB. The Page table entry size is 4 B. What is the minimum P_s possible such that entire PT fit in one page.

$$4B \leq \frac{(P_s)^2}{256 \; MB}.$$

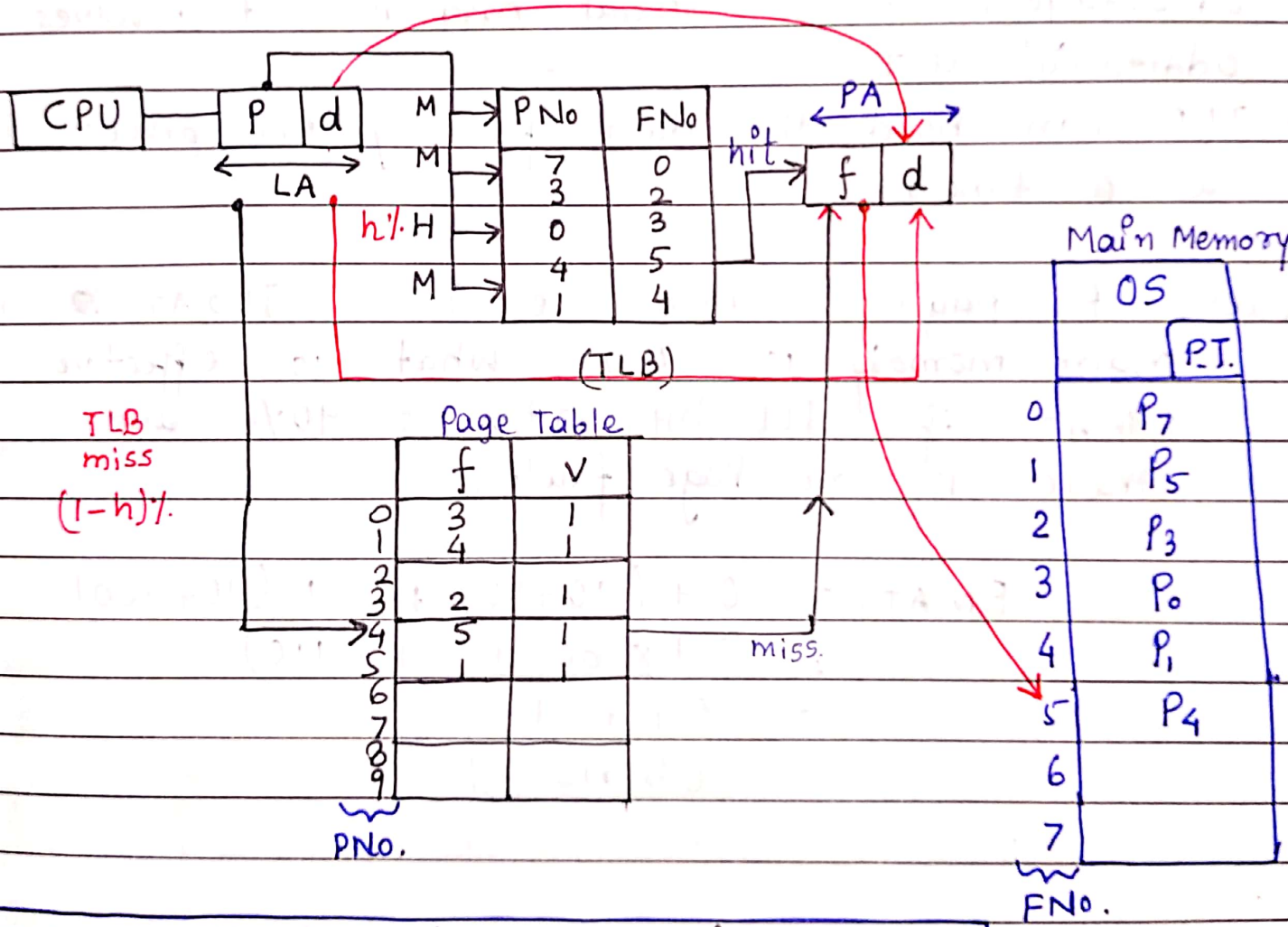
$2^2 \times 2^{28} B \leq (P_s)^2$

$P_s \geq 2^{15}$

$\boxed{P_s \geq 32\,KB}$

TRANSLATION LOOK ASIDE BUFFER

- By using TLB, we can improve performance of memory access. Means we can reduce Effective Memory access time.



$$EMAT = \%h\,(c+m) + (1-h)\%\,(c+m+m)$$

hit ratio → TLB Acc, Inst. Acc.

miss Ratio → TLB Acc, PT Acc., Inst. Acc.

- TLB contain 2 fields : PNo , Frame No
- Hardware device implemented using associative register . It is faster as compared to main memory.
- TLB will be placed before PT.

Advantage : TLB reduces effective memory access time.
Only one memory access is required.

Disadvantage : It is special hardware, it involves additional cost.
- TLB can hold the data of only one process at a time.

Ques.) A paging scheme uses TLB . TLB AT = 10 ns main memory is 50 ns. what is effective EMAT if TLB hit ratio is 90% and there is no Page fault.

$$\begin{aligned} EMAT &= 0.9\,(10+50) + 0.1\,(10+100) \\ &= 0.9 \times 60 + 0.1\,(110) \\ &= 54 + 11 \\ &= 65 \text{ ns} \quad // \end{aligned}$$

Q:) Consider a paged virtual memory system with 32 bit virtual address. 1KB pages. Each page table entry requires 32 bits. It is desired to limit page table size to 1 page size.

(i) How many levels of page table are required?

Sol:) LAS = 2^{32} # Pages = $\frac{2^{32}}{2^{10} \cdot 2^3} = 2^{22-3} = 2^{22}$

$\therefore$ 2^{22} entries x 2^2 = 2^{24} Bytes.

2^{10} Bytes

$\frac{2^{21} \text{ Bytes}}{2^{10}} = 2^{11}$ pages. (level 2).

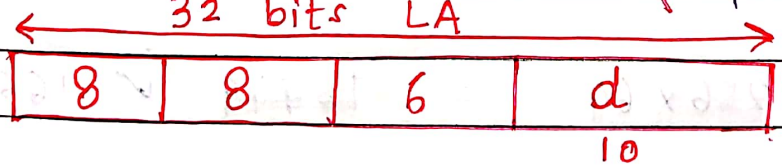
Size $\therefore$ $2^{11} \times 2^5 = 2^{16}$ Bytes.

$\frac{2^{16}}{2^{10}} = 2^6$ pages. (level 3)

Size = $2^{6+5} = 2^{11}$ Bytes.

$\therefore$ $\frac{2^4}{2^{10}} = 2$ pages.

$2 \times 2^5 = 2^6$ Bytes. (level 4).

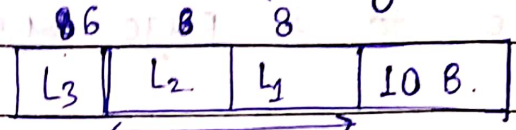


$\frac{2^{32}}{2^{10}} = 2^{22}$ pages. $\frac{2^{24}}{2^{10}} = 2^{14}$ pages $\Rightarrow \frac{2^{16}}{2^{10}} = 2^6$ pages. $= 2^8$ Bytes ✓

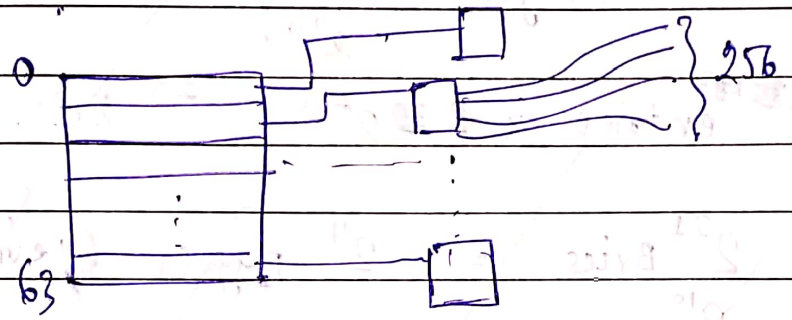
level 1 — level 2 (address of level 3 stored here) — level 3 (frames stored here)

(ii) The page table have 2 level have 2^8 entries. Table at 1 level 2^6 entries. If top level page table has 2^6 entries then how many pages are possible on page table?

$P_s = 4\ B$



$2^8 \times 2^2 = 2^{10}$



1 page 64 pages 256 pages × 64

$\therefore 2^{14} + 2^6 + 1 = 16384 + 64 + 1$

$= 16449$ ✓

(ii) A middle level has 2^6 entries then how many pages are possible?

$1 + 256 + 256 \times 64 =$ ~~16449~~ ✓ 16641

(iii) If bottom level has 2^6 entries how many pages?

$1 + 256 + 256 * 256 = 65793$ ✓

Question) A computer has a 64 bit virtual Address and 2048 Bytes (2KB pages). The page table entry size is 4 Byte. A multilevel page table is used and each page table size = 1 page. How many levels are required?

8	9	9	9	9	9	11 bits.

53 bits.

6 levels

entries in P.T.

$\frac{2048\ B}{4\ B} = 512$

$\therefore 2^9$

Ques.) Consider a system with 2 level paging. PT is divided into 2K pages. Each of size 4 Kilo word. PT entry size is 2 words. Memory is word addressib PAS is 64 MW which is divided into 16 K frames. Then calculate :

(i) length of logical address.
(ii) length of Physical address
(iii) outer page table size.
(iv) Inner page table size.

L_2	L_1	d

2^{16}

0

$(2^{16}-1)$

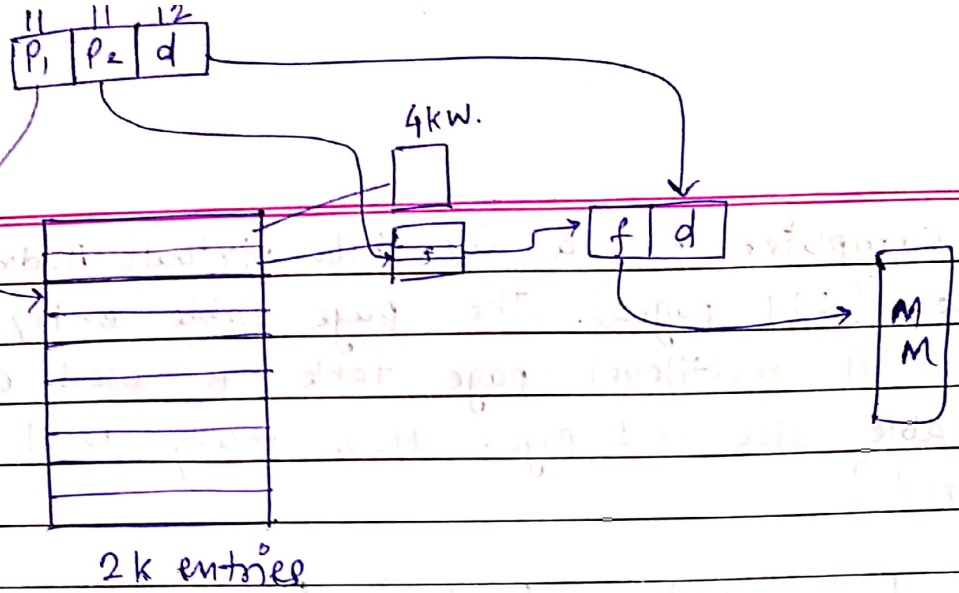
PAS.

$2K = 2^{11}$

$\frac{2^{26}}{2^{14}} = 2^{12}$ pages.

size of frames $= \frac{2^{26}}{2^{14}} = 2^{12} = 4K$ words.

$\therefore$ Page size = 4KW.



SYSTEM

entries in 1 page = $\frac{4\text{ kw}}{2\text{ w}}$ = 2K entries.

$d = \log_2 P_s$ = 12 bits.

LA = $P_1 + P_2 + d$ = 11 + 11 + 12 = 34 bits.

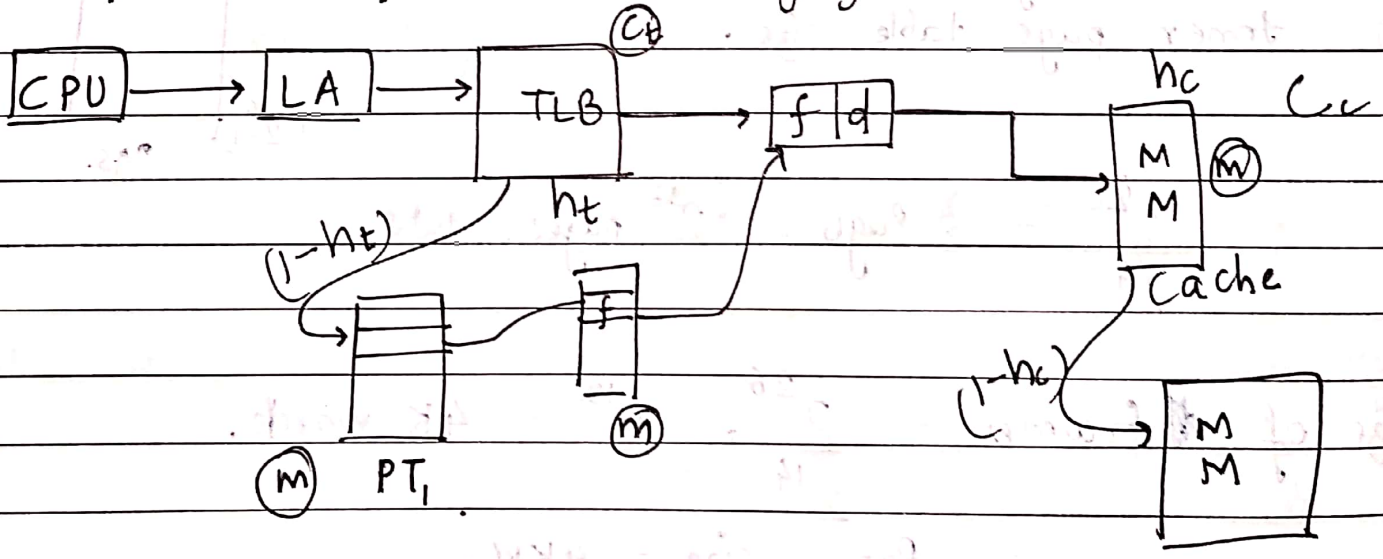
PA = $f + d$ = $\log_2(16K) + 12$

= 14 + 12 = 26 bits.

outer page table size = 2K x 2W = 4K w.

inner page table size = 2K x 2W = 4KW

Performance of Multilevel paging : For two levels.



$$Emat = h(c+m) + (1-h)(c+m+m+m)$$

Hit — TLB — Main Memory — Miss — TLB — PT_1 — PT_2 — Main Memory.

In general,

$$Emat = h(c+m) + (1-h)(c+(n+1)m)$$

(n = number of levels.)

Ques.) TLB hit ratio is 0.95. $C = 0$.
cache memory access time 10 ns
MM access time = 100 ns.
CM hit ratio = 90 %.
Find effective memory access time for 2 level paging system.

$$Emat = 0.95(0.9\times10 + 0.1\times100) + 0.05(0.9\times10 + 0.1\times100 + 0.9\times10 + 0.1\times100)$$
$$= 0.95(9+10) + 0.05(3\times19)$$
$$= \cancel{1} + \cancel{2}$$
$$= \cancel{3\ ns}.$$

$$Emat = h_{TLB}(C_t + h_{cache} * C_c + (1-h_{cache})\cdot C_c + m) + (1-h_{TLB})[C_t + m + m + h_{cache} * C_c + (1-h_{cache})(C_c + m)]$$

$$Emat = 30\ ns \checkmark$$

Ques) Consider a system using multilevel paging scheme Page size is 16 KB. Memory is Byte addressible.

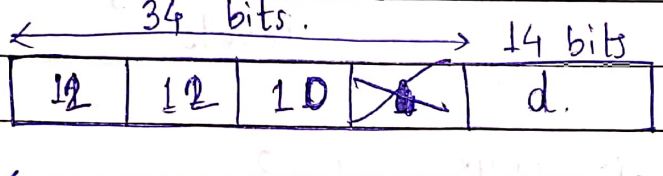
$P_s = 16\ KB$

$VA = 48$ bits.

$es = 4\ B$.

(i) How many levels of Page table are required

(ii) Give the divided PA and VA.

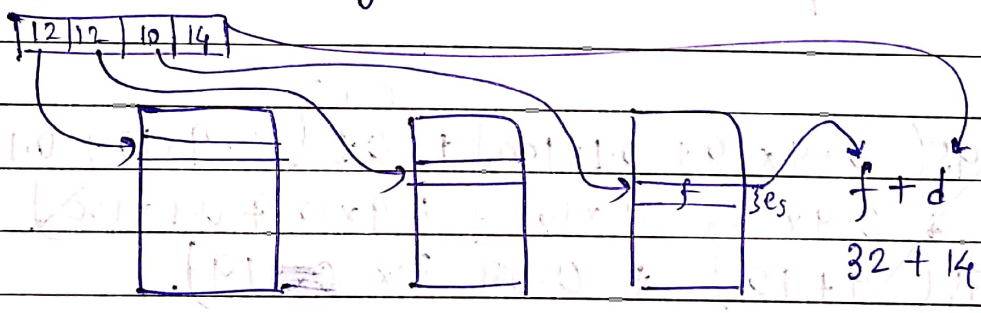


1 word = 1 B

48 bits.

$d = \log_2 P_s = \log_2 2^{14} = 14$ bits.

entries in level L = $\frac{16\ KB}{4\ B}$ = 4K entries



32 + 14 = 46 bits.

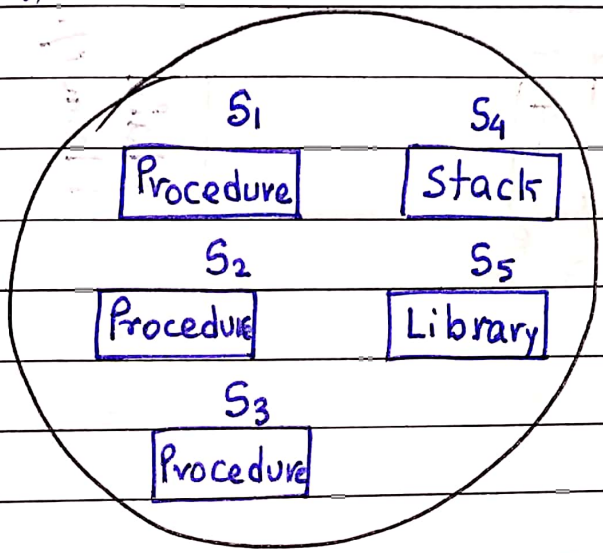
∴ PAS = 2^{46} B = 64 TB.

frame bits = es.

Segmentation

- Like paging, segmentation is another non-contiguous, memory location technique.
- In segmentation, process is not divided into fixed sized pages. The process is divided into modules for better utilization.

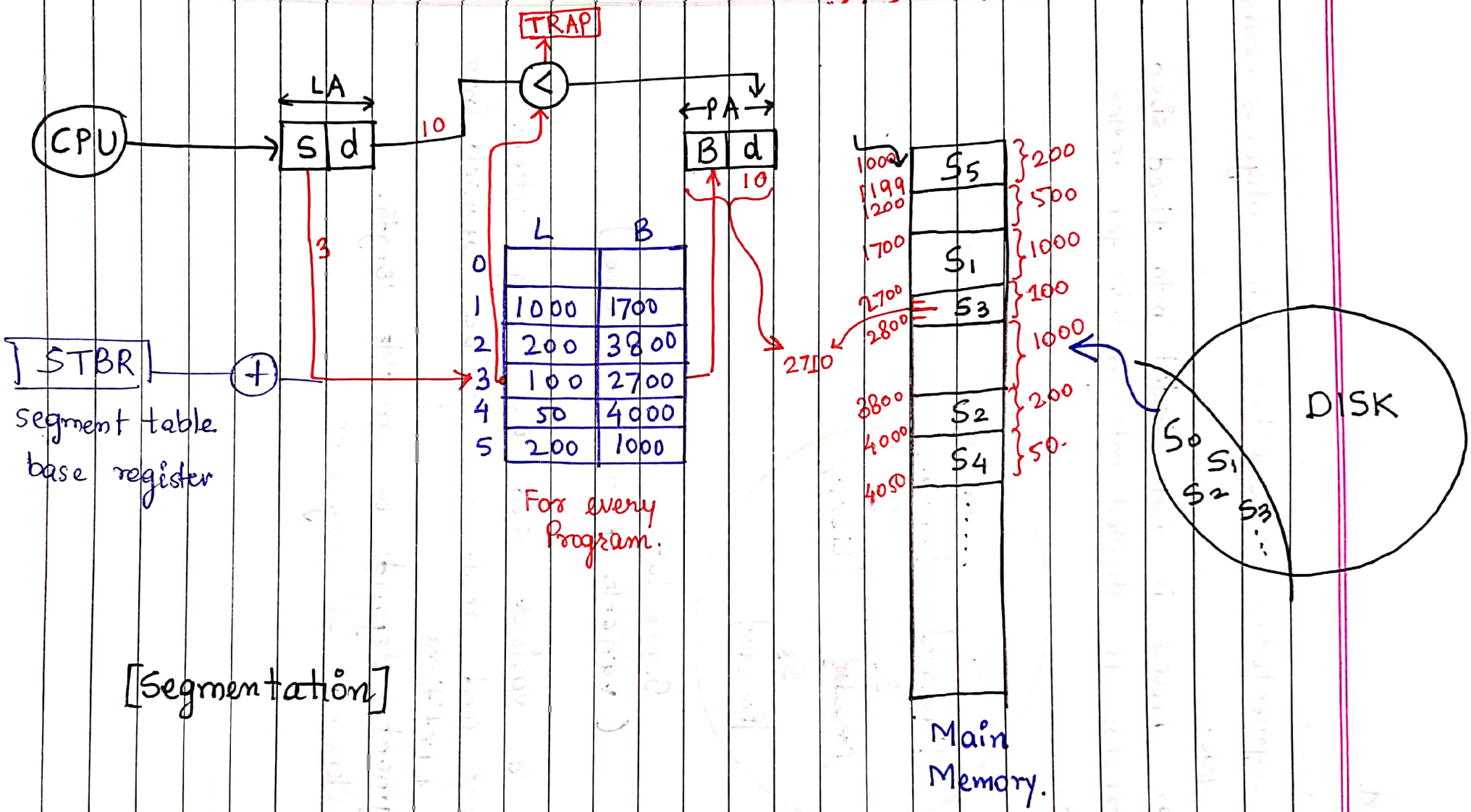
```
void add ()  } S1
{            }
}            }

void sub ()  } S2
{            }
}            }
```



Segmentation
(variable partition scheme)

- Segmentation is a variable sized partitioning scheme.
- Secondary memory and main memory are divided into variable size partitions.
- Size of segment depends on module length.
- Partion is known as segments.

OPERATING SYSTEM



[Segmentation]

Ques:) Consider following segment table.

S_0	1219	700
S_1	2300	14
S_2	90	100
S_3	1327	580
S_4	1952	96

which of the following produces trap?

(A) 0, 430 → 1649
(B) 1, 11 → 2311
(C) 3, 425 → 1752
(D) 4, 95 → 2047
(E) 2, 100 ✓ → 190

)) In segmentation, Logical Address space, segment No / segment address / Number of bits required to represent segment size or word No of segment, or word offset or word address of segment.

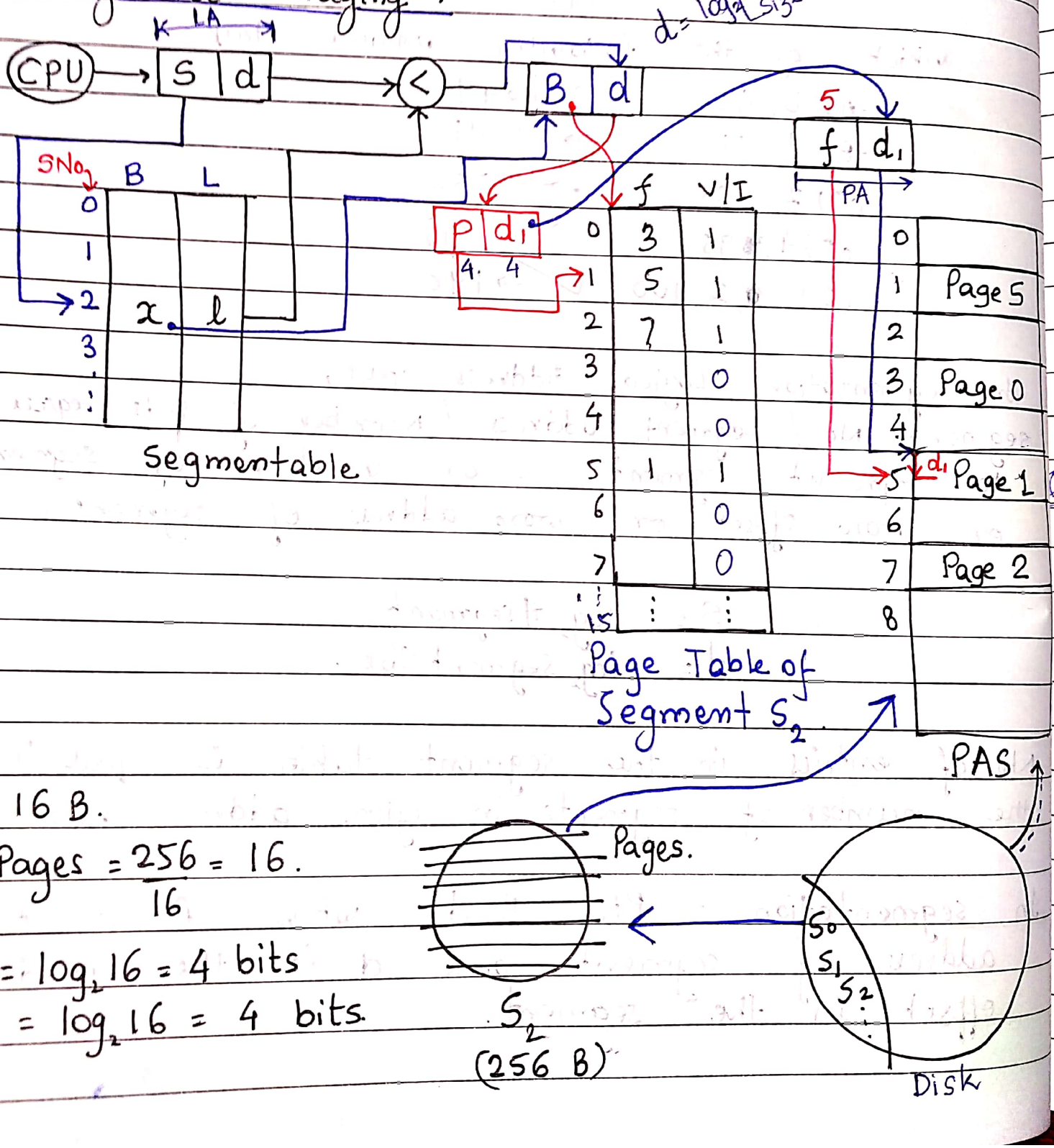
$$S = \log_2 \#\text{segment}$$
$$d = \log_2 \text{segment Size}.$$

No of entries in the segment table is equal to the number of segments in logical address space.

In segmentation = PA = B+d where B is starting address of segment and d is the word offset in the segment.

- The variable size fragments transfered from logical address space to physical address space. Hence it is similar to variable partitioning scheme. Thus it suffers from external fragmentation.

Segmented – Paging :



$P_s = 16$ B.

$\#\text{Pages} = \frac{256}{16} = 16.$

$P_1 = \log_2 16 = 4$ bits

$d_1 = \log_2 16 = 4$ bits.

- To avoid overhead of transferring large size segment into memory, the segmented paging will be applied. In segmented paging, the paging will be applied on segment.

- Logical Address = s + d.
  d = p + d_1
  where p is Pagenumber of segment.
  d_1 is word number of page.

  d = $\log_2$ Segment size
  d_1 = $\log_2$ Page size.
  p_1 = d - d_1 = $\log_2$ #Pages in Segment.

  f = $\log_2$ m

Ques Given, # segments = 2^{18}
Ps = 1 kW
SS = 64 kW
Find s, d, p_1, d_1 ?

[s = $\log_2 2^{18}$ = 18 bits.]
[d_1 = $\log_2$ Ps = 2^{10} = 10 bits]
Pages = $\frac{64\,K}{1\,K}$ = 64 pages. = 2^6
∴ [p = $\log_2 2^6$ = 6 bits.]

d = 2^{16} = 16 bits.

Advantages of Segmented Paging:

- Reduces memory usage.
- Size of page table is limited by segment size.
- It solves the Problem of external fragmentation.

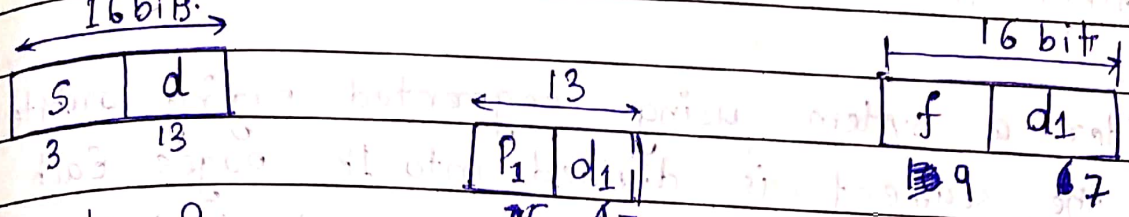
Disadvantages:

- Internal fragment
- Complexity level is much higher as compared to paging.

OPERATING SYSTEM

(Ques.) A certain Computer System has segmented paging architecture for virtual memory. The memory is byte addressible. Both virtual and Physical Address space contain 2^{16} Bytes. The virtual address space divided into 8 non overlapping equal size segments. The memory management unit has a hardware segment table. Each entry of which contain the physical address of page table for a segment. Page table are stored in the main memory and consist of 2 Bytes page table entry. What is the minimum page size in bytes so that the page table of a segment require atmost one page to store it?

LAS = PAS = 2^{16}.



segments = 8.

Segment size = 2^{13} Bytes.

$d = \log_2 2^{13} = 13$ bits.

d is divided into P_1 and d_1.

Page table entry = 2 bytes. = 16 bits.

∴ frame No = 16 bits.

Total P_s = 2 Bytes × # pages.

$$\#\text{ pages} = \frac{2^{13}}{P_s}$$

entries in page table = N * e_s.

$$PT_s < P_s$$

$$N * e_s \leq P_s$$

$$\frac{2^{13}}{x} \cdot 2 \leq x.$$

$$x^2 \geq 2^{14}$$

$$x \geq 2^7.$$

∴ $x = 128$ Bytes.

(ii) Consider above problem, get the division of virtual address.

d_1 = $\log_2 2^7 = 7$ bits.

P_1 d_1 = 13 − 7 = 6 bits.

Paging on Segment table and segment.

(see snapshot in one note).

Ques) Consider a system using segmented paging architecture where the segment is divided into 1k pages. Each of size 512 words. And the segment table is divided into 1K pages, each of size 256 words. The frame number require 18 bits to represent all the frame of physical address space. Page table entry size is 2 words. Memory is word addressable. Calculate, LA, PA, PT size of segment, PT size of segment table.

Solution: Given,

Segment size = 1024×512 words

$= 2^{10} \times 2^9$ words

$= 2^{19}$ words.

$\therefore$

$d_1 = \log_2[\text{Page size}] = \log_2 2^9 = 9$ bits

$P_1 = 19 - 9 = 10$ bits.

$\therefore$ # Page table (for segment table) $= 2^{10}$ entries.

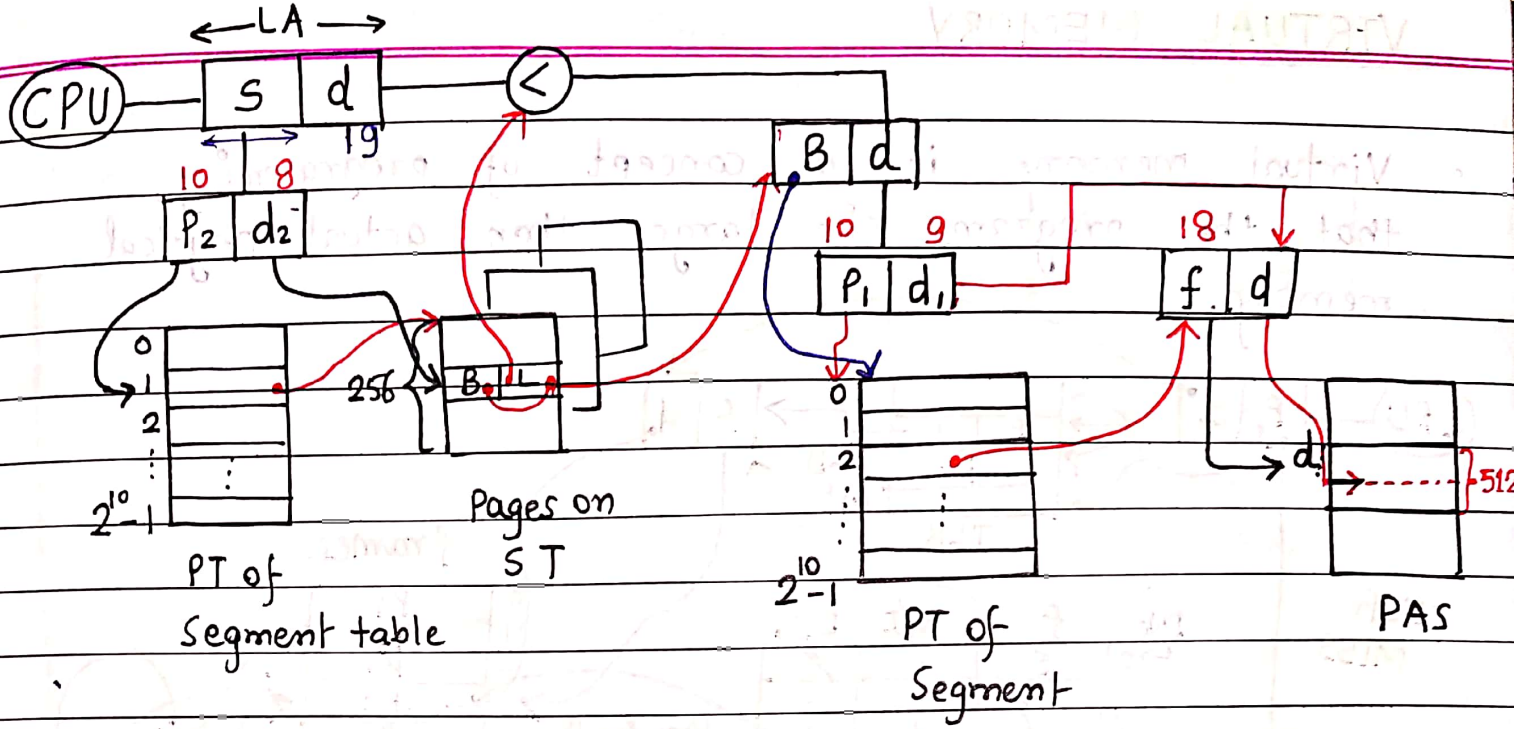
[See side diagram]

$S + d = 18 + 19 = 37$

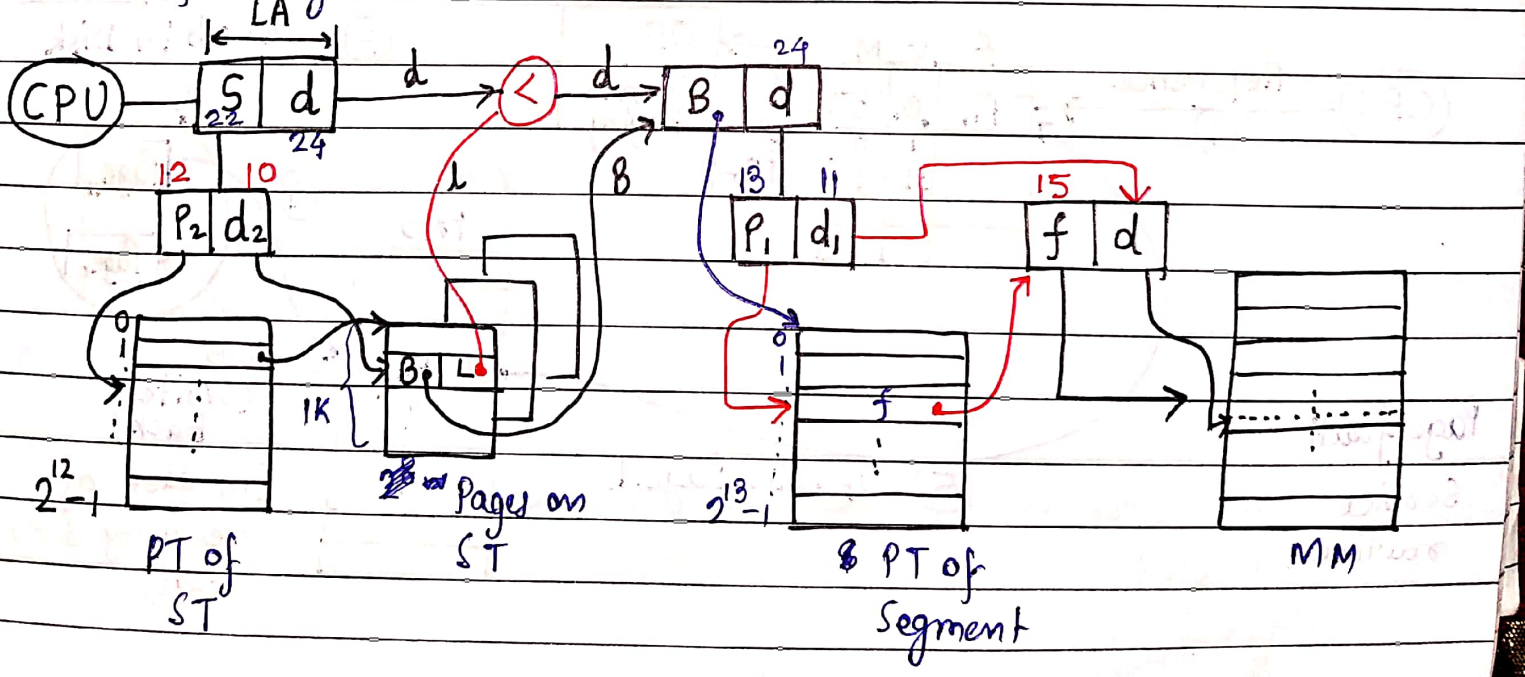
$P_2 + d_2 = 10 + 8$

$P_1 + d_1 = 10 + 9$

$f + d = 18 + 9$.

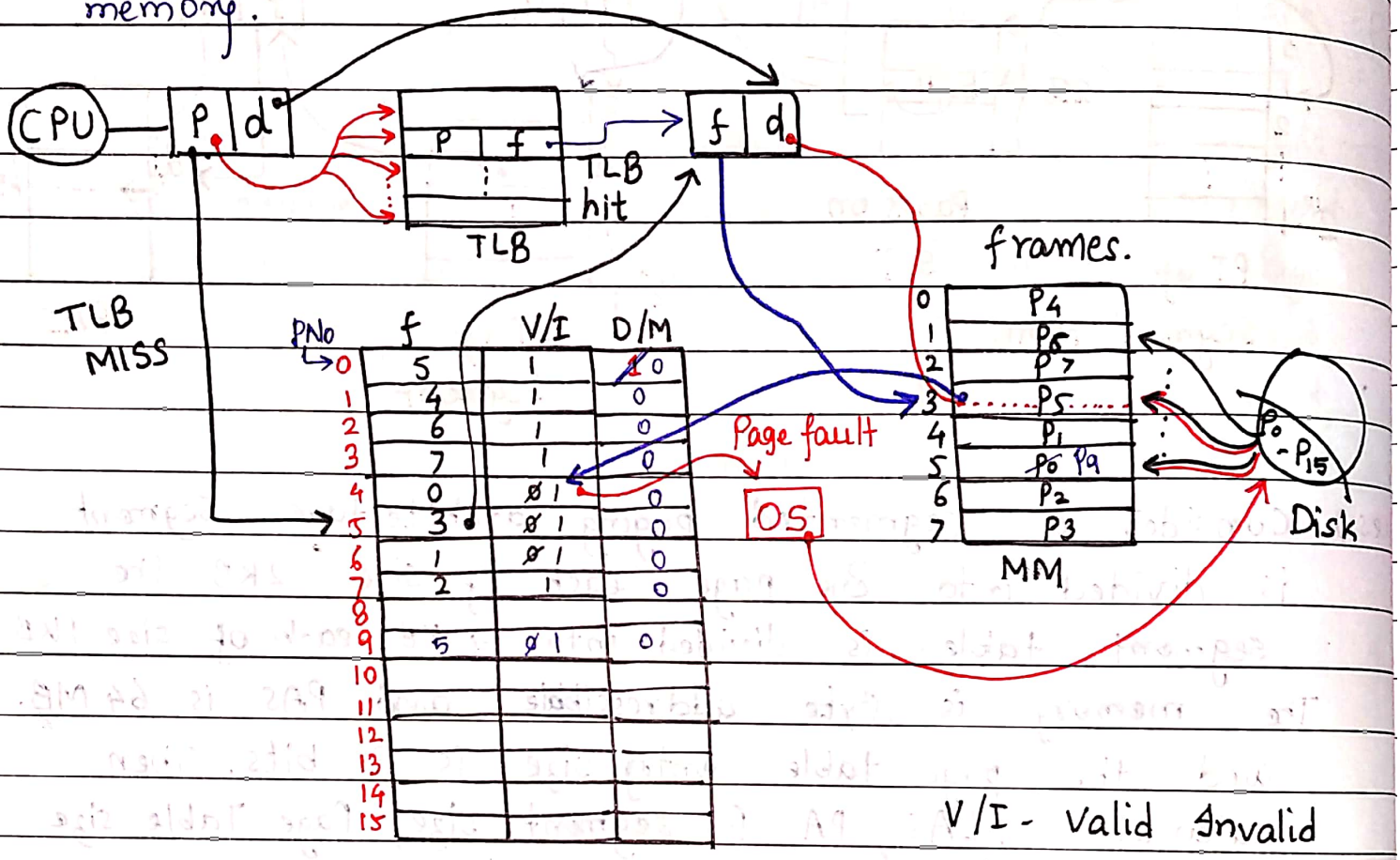


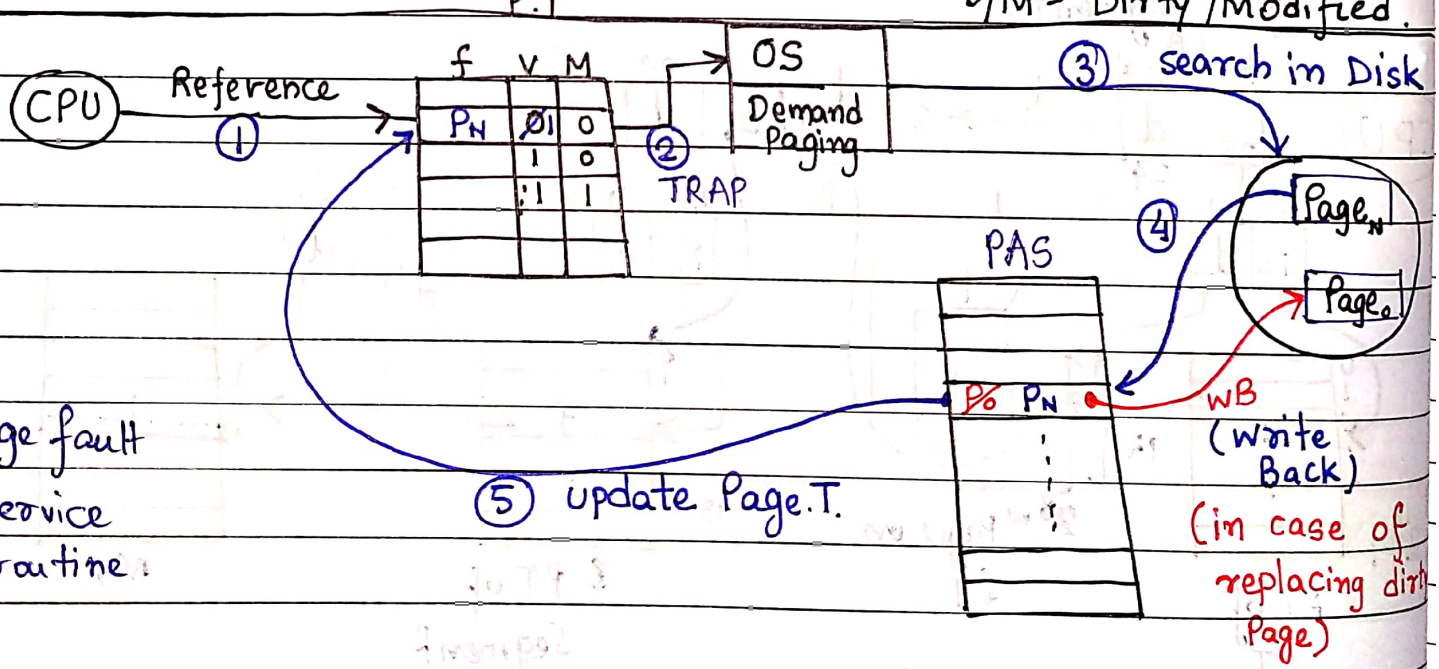
Ques.) Consider a segmented paging architecture. Segment is divided into 8K pages each of size 2KB. The segment table is divided into 4 KB each of size 1KB The memory is Byte addressible and PAS is 64 MB. and the page table entry size is 32 bits. Then calculate , LAS, PA, PT segment size, Page Table size of segment.



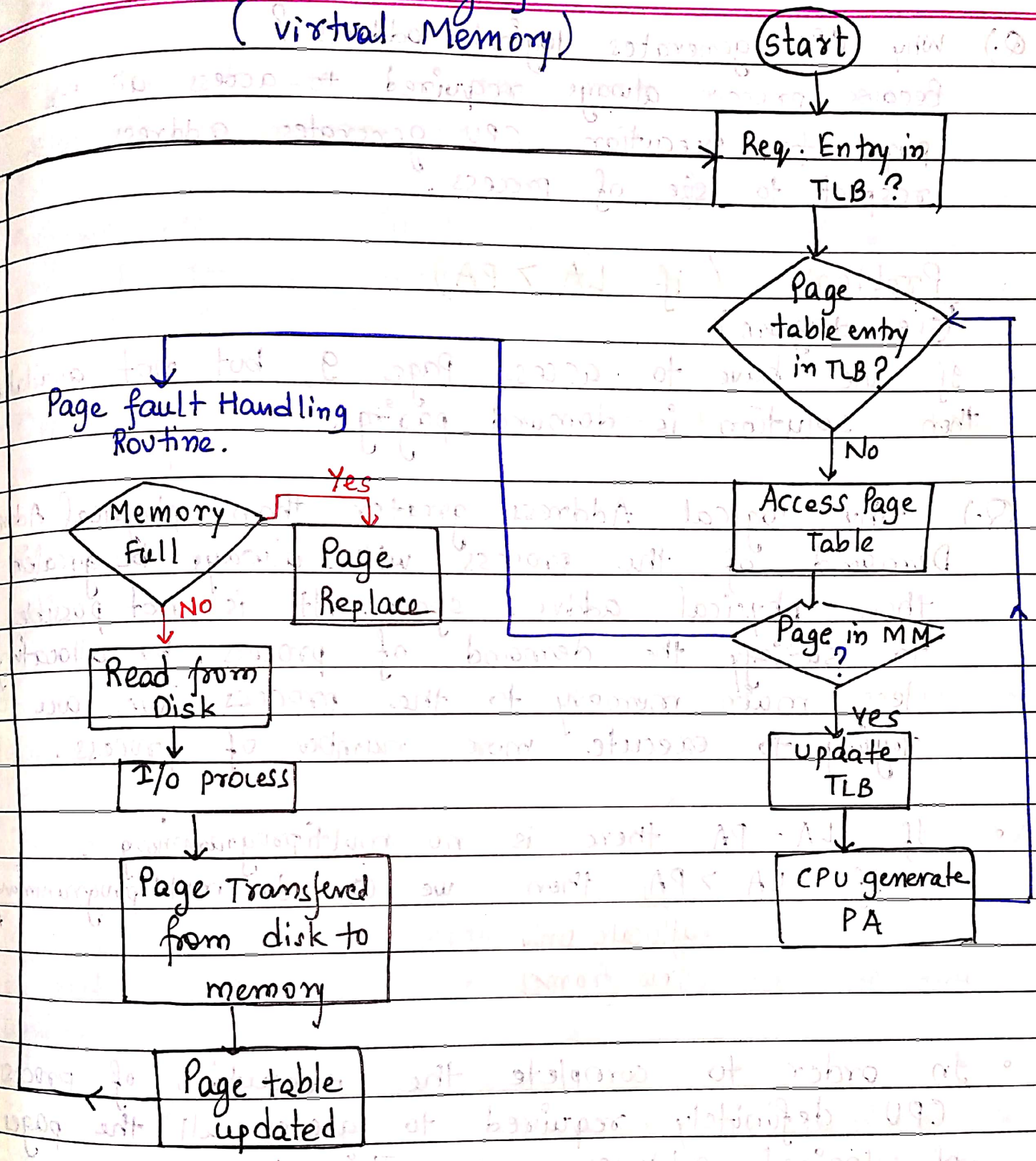
VIRTUAL MEMORY

- Virtual memory is a concept of programming, that the program size larger than actual physical memory.





TLB and Paging (virtual Memory)



operation of Page table and TLB while implementing VM.

Q.) Why CPU generates logical address ?

- Because process always required to access all the pages for execution. CPU generates address with respect to size of process.

Problem (if LA > PA)
(see diagram)

- If we have to access Page 9, but not available then solution is demand paging.

Q.) Why logical Address greater than physical Address
Demand of the process will always be greater than physical address space. It is not possible to satisfy the demand of process by allocating less main memory to the process, we are trying to execute more number of process.

- If LA = PA there is no multiprogramming.
- If LA > PA, then we can do multiprogramming
  allocate only few frames

- In order to complete the execution of process, CPU definitely required to access all the pages of logical address space. This is the reason when CPU is generating the address, it will generate with respect to size of the process.

PAGE FAULT :

- CPU tries to access a page, but the page is not available in memory, then it is called as page fault.
- Program execution will be stopped and signal will be sent to the OS recording page fault. Then the OS validates the detail of the process.
- Operating System will search for the required page in Logical Address space.
- Required page will be transferred from logical Address space to physical address space.
- If main memory is full, then apply Page replacement algorithm.
- Decision making algorithm for replacing the page in PAS.
- Page fault will be updated in page table. If P.T is not updated, then it will result in page fault again.
- Signal will be sent to the CPU to continue the program execution. Then the CPU will access the required page in the memory, and will continue the program execution.

- Because of this process, program execution will stop it cannot be observed by user.

DEMAND PAGING:

- Loading a page into main memory from LAS to PAS on demand.
- The time taken to service a page fault, is called as page fault service time.
- Page fault service time include the time required to perform all the six steps.
- If page fault service time = S
  main memory access time = m
  page fault rate = p $\longrightarrow \frac{\#miss}{\#miss + \#ref\ hit}$

$$\boxed{Emat = (1-p)\,m + Sp}$$ (using TLB)

$m \ll S$.

Ques) Consider a system with page fault service time be 200 ns. Page hit ratio is 95 %. mm access time is 20 ns.

$= 0.95 \times 20 + 0.05 \times 200$
$= 19 + 10$
$= 29$ ns.

Ques) suppose an instruction take, i° microseconds, and additional j microsecond. If page fault occur. what is effective memory access time, if page fault occurs on an average every k instructions ?

page fault rate = $\frac{1}{k}$

Emat = $i^\circ + \frac{j^\circ}{k}$

Ques) Let the page fault service time be 10^{-2} s with average memory access time be 20 ns. If 1 page fault is generated for every 10^6 memory access. then what is effective MAT.

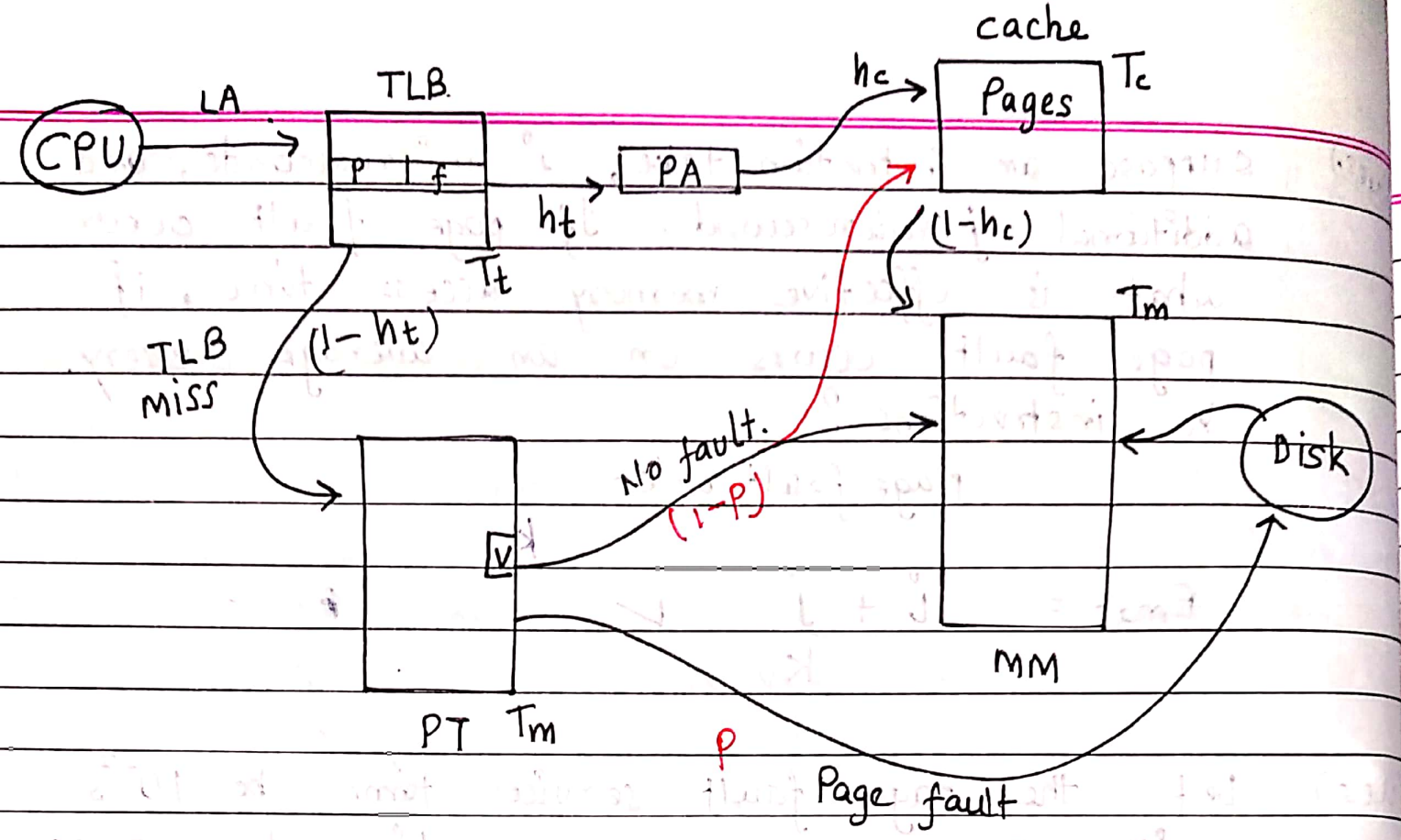
$$Emat = \left(1 - \frac{1}{10^6}\right) \times 20ns + \frac{10^{-2+9}}{10^6} ns \times 1$$

$$= \frac{(10^6 - 1)}{10^6} \times 20 + 10$$

$$= 19.999 + 10$$

$$\approx 29.999$$

$$\approx 30 ns.$$



$$EMAT = h_t\left[T_t + h_c T_c + (1-h_c)(T_c + T_m)\right] + (1-h_t)\left[T_t + T_m + (1-P)\left[h_c T_c + (1-h_c)(T_c + T_m)\right] + P \times S\right]$$

Negligible as compared to S

Ques) Consider a system using demand paging, the page fault service time is 200 ms and main memory access time is 10 ms. the TLB is added to improve the performance. 80% reference are found in TLB, and that of remaining, 10% cause page fault. TLB access time and Page table access time both are negligible. What is effective MAT

EMAT = 0.8 x 10 ms +
0.2 x [0.8 x 200 ms + 0.2 x 10 ms]

EMAT = 33.8 ms

Ques.) Consider demand paging environment where if it takes 8 ms to service a page fault, if either an empty frame is available or replaced page is not to be modified. If it takes 20 ms to service a page fault if a page is modify. Main memory access time is 1 ms and further assume page to be replaced is modified 70 % of the time. Then what is the maximum acceptable page fault rate to get an Emat not more than 2 ms.

Emat ≤ 2 ms.

2 ms ≥ (1 − p) · 1 ms + p [0.7 x 20 ms + 0.3 x 8 ms]

16.4

2 ms ≥ 1 ms − p ms + 16.4 ms x p

2 ms ≥ 15.4 p ms + 1 ms

15.4 p ms ≤ 1 ms.

p ≤ 1 / 15.4

p ≤ 0.0649

or

p ≤ 6.4 %.

Page Replacement Algorithm

(1) FIFO : It replace the oldest page that has been present in the main memory for the longest time. It is implemented by keeping track of all the pages in a queue.

Ques For eg. Consider number of frames in memory = 3 What is the total number of page fault that is occurred in memory using these reference.

×	×	×	×	✓	✓	✓	×	×	✓	
4	7	6	1	7	6	1	2	7	2	

4 1
7 2
6 7

miss = 6 # hit = 4 ✓

Ques Number of frame = 4

7	0	1	2	0	3	0	4	2	3	0	
×	×	×	×	✓	×	✓	×	✓	✓	×	
3	2	1	2	0	1	7	0	1			
✓	✓	×	×	✓	×	×	✓	✓			

4 7 3 2
0 4 7
1 0
2 1

miss = 10 # hit = 10

If No of frame = 3.

7	0	1	2	0	3	0	4	2	3	0	
×	×	×	×	✓	×	✓	×	×	×	✓	
3	2	1	2	0	1	7	0	1			
✓	✓	×	✓	✓	×	×	✓	×			

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

miss = 15 # hit = 5

Ex 1 2 3 4 1 2 5 1 2 3 4 5.

Total = 12

Frame = 3 ⇒ (9)

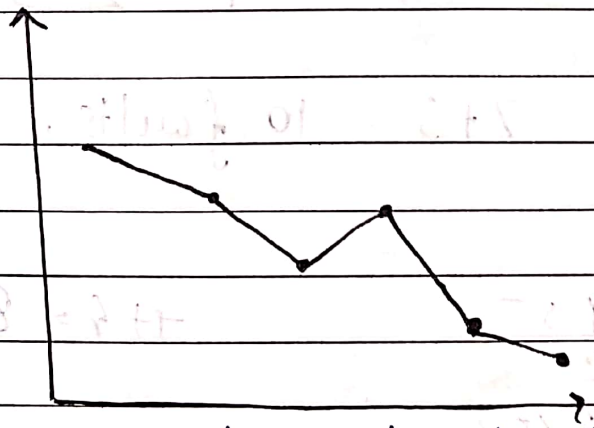
1 4 5
2 1 3
3 2 4

Frame = 4. (10)

1 5 4
2 1 5
3 2
4 3

Effect of increasing no of frames : The no of page fault should either decrease or remain constant on increasing the number of frames in the main memory. But some time, the unusual behaviour is observed.

- Some time by increasing the number of frames in main memory, the number of page fault also increase. this is known as Belady's anomaly. Phenomenon of increasing the page faults on increasing No of frame in main memory.



FIFO Random page replacement algorithm and Second chance algorithm.

(2) LIFO page replacement algo: It replaces the newest page that arrive at last in the main memory. It is implemented by using stack.

Ex 4 7 6 1 7 6 1 2 7 2.

Frames ⇒ 3. 4 7 6 1 7 6 1 2 7 2 4+3 = 7 faults.

Frame = 4. 4 7 6 1 7 6 1 2 7 2 3+4 = 7 faults.

(3) LRU (Least recently used): It replaces the page that has been referred by the CPU for the longest time. Replace the page that has not been used for longest period of time.

Ex 1 2 3 4 1 2 5 1 2 3 4 5.

Frames=3

1 2 3 4 5 7+3 = 10 faults.

Frame=4 4+4 = 8 faults.

Ex. 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0
1 7 0 1.

frame = 3.

7 2 4 0 7 = 10 + 3 ✓
0 3 0 12 faults.
1 3 2 1

Que) MRU : 1 2 3 4 1 2 5 1 2 3 4 5
frame = 3

1
2 3 3 + 5 = 8 ✓
3 4 5

Optimal page Replacement:

- Replace the page that will not be used for longest period of time in future.

Ex 4 7 6 1 7 6 1 2 7 2
frame = 3

4 1
7 2 + 3 = 5 fault.
6 2

1 2 3 4 1 2 5 1 2 3 4 5
frame = 3

4 4 + 3 = 7 faults. ✓
2
3 4 5

Ex. Consider following reference for 460 words program

10 11 104 170 73 309 185 245 246 434 458 364.

Give

(i) the memory reference string assume page size = 100 words.

P_1 0-100
P_2 100-200
P_3 200-300
P_4 300-400
P_5 400-500.

(ii) Find No of page faults. Assuming 200 words available in primary memory at a time (By using LRU).

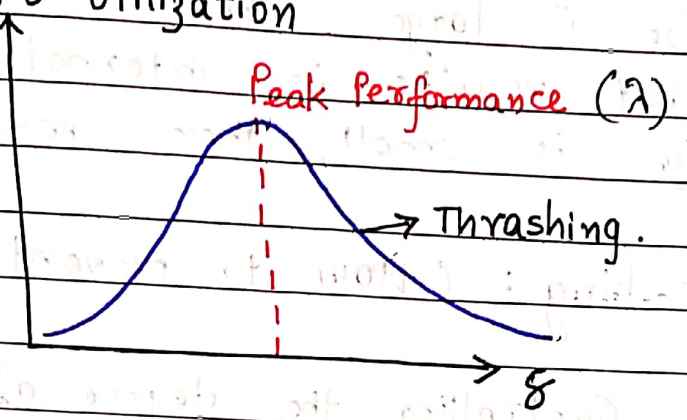
(i) 1 1 2 2 1 4 2 3 3 5 5 4

(ii). 1 2 5 4 2 3 4 — 5 + 2 = 7 faults.

Ques) A system uses FCFS algorithm for page replacement. It has 4 page frames, with no pages loaded to begin. The system first access 100 distinct pages, in some order, and then access same 100 pages but in reverse order. How many page faults will occur. 196 ✓

SYSTEM

Thrashing :



- In the initial δ of multi-programming, the CPU utilization is very high and the system resources are utilized 100 %. If further increase the degree of multiprogram. the CPU utilization will repeatedly fall down and the time taken to complete execution will increase and system spends more time only in page replacement.
- Every page reference will be page fault and CPU remain busy in dealing with page fault. It is called thrashing.
- High paging activity is called thrashing. A process thrashes if it is expanding more time in paging than executing.
- CPU utilization is very low when the page fault rate is very high.

Reason of thrashing :

(i) Primary Reason : High degree of multiprogramming
(ii) Secondary Reason : Lack of frames, Bad page replace algorithm, Page size smaller.

- If page size is large, then chances of thrashing reduces; but results in internal fragmentation
- If page size is small, then more page faults.

Control Thrashing : (How to prevent).

(i) Prevent : Controlling the degree of multiprogramming Don't allow the system to go into thrashing and instruct the long term scheduler, not to bring the process into main memory, After the point of

(ii) Detection : Low CPU utilization, low throughput, high degree of multiprogram, high paging demands from disk.

(iii) Recovery : Process suspension; allow the system to go into thrashing and then instruct the mid term scheduler to suspend some of the process so that the system will recover from thrashing. Or we can increase RAM size.

Ques Consider the system with below condition.
(i) 90 % the page in disk
(ii) CPU utilization is 10 %.
which of the below factor will improve CPU utilization.
(A) Install the bigger disk. - NO

(B) Install the faster CPU - NO.
(C) Increase degree of multi programming. - NO
(D) Decrease degree of multi programming. ✓
(E) Install more RAM. ✓
(F) Decrease page size - NO. (Because pages req. by process will increase.)
(G) Increase page size ✓ (Because pages req by process will be less.

Frame allocation Policy

- If m (number of frame availabe) in main memory.
  n (number of process)
  S_i = number of frame demand of each process.

Allocation of frame for each process = $\frac{S_i}{S} * m$

where S_i is, # frames demanded by i^{th} process
S is $\sum S_i$, total frames demanded
m, # frames in main memory.

This is called proportional frame allocation policy.

Ques For a certain system, total number of frames is 64. The size of two process P_1 and P_2 are

$P_1 = 10$
$P_2 = 127$

How much allocation for each process?

for $P_1 = \frac{10}{137} \times 64 = \frac{640}{137} = 5$

$P_2 = 64 - 5 = 59$.

File and Device Management

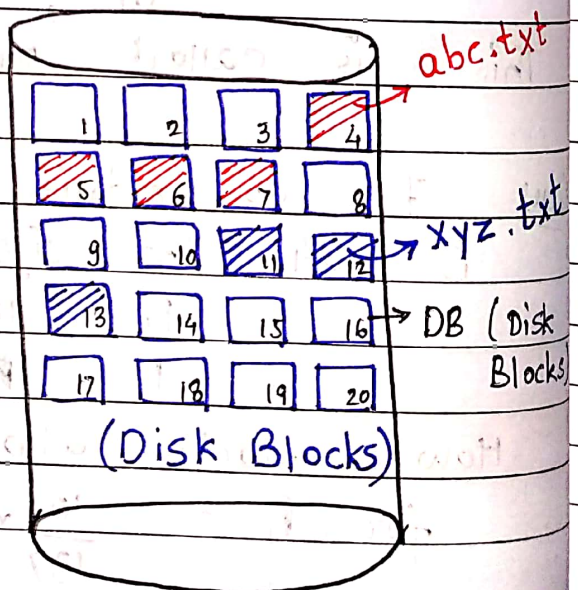
- File is collection of logically related entities (records).
- Attributes : Name, size, type, location, creation date, modification date, owner, password, permission and so on. (File Context).
- All the attributes of file is called as file context. And the file context will be stored in file context block. (FCB)
- Type of Files : .Doc, .txt, .obj, .exe, .class, .C, .java, .js etc.
- various operation performed on file.
  - create, append, close, read, write, save, copy, paste,
  - delete, rename, save as different file.

Disk Space Allocation Methods :

(i) Contiguous allocation:

File Name	Start	length
abc.txt	4	4
xyz.doc	(11)	3

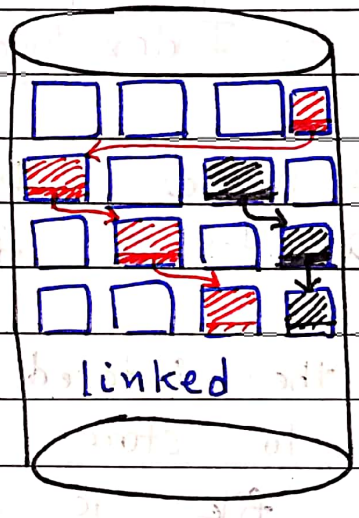
(11) → Disk Block Address (DBA)



- In the contiguous allocation, disk blocks are allocated to the file in a continuous manner.
- Increase the file size is not always possible, because next block must be available.
- It suffers from external fragmentation.
- Internal fragmentation exists in the last disk block of the frame. It supports sequential and random access of the file.

(ii) Non-contiguous allocation :

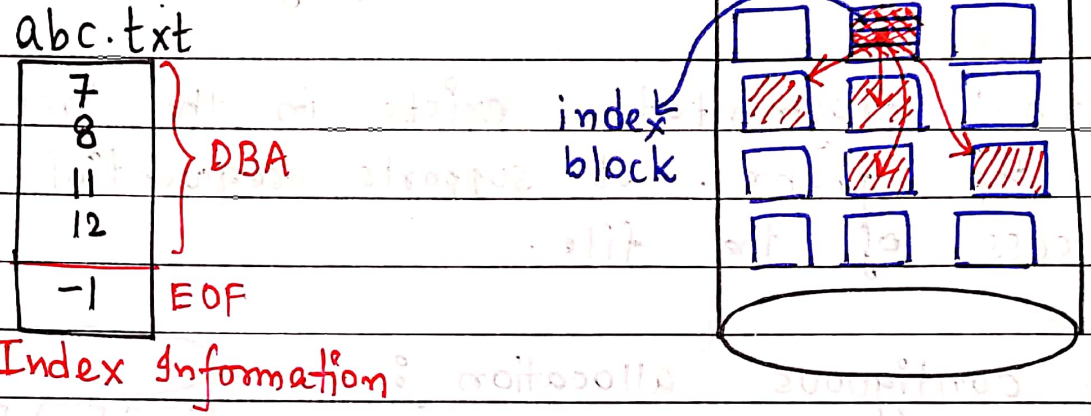
File Name	Disk	Ending
abc.txt	4	15
xyz.doc	11	20



- Increasing the file size is always possible, if the free disk block is available.
- There is no external fragmentation.
- The internal fragmentation may exist in the last disk block of the file.
- It support only the sequential access of the file.
- There is an overhead of maintaining a pointer in every disk block.

- If pointer of any disk block is lost then the file will be terminated. The ending of DBA is block address, confirm that the total file is accessed.

(iii) Indexed Allocation Method:

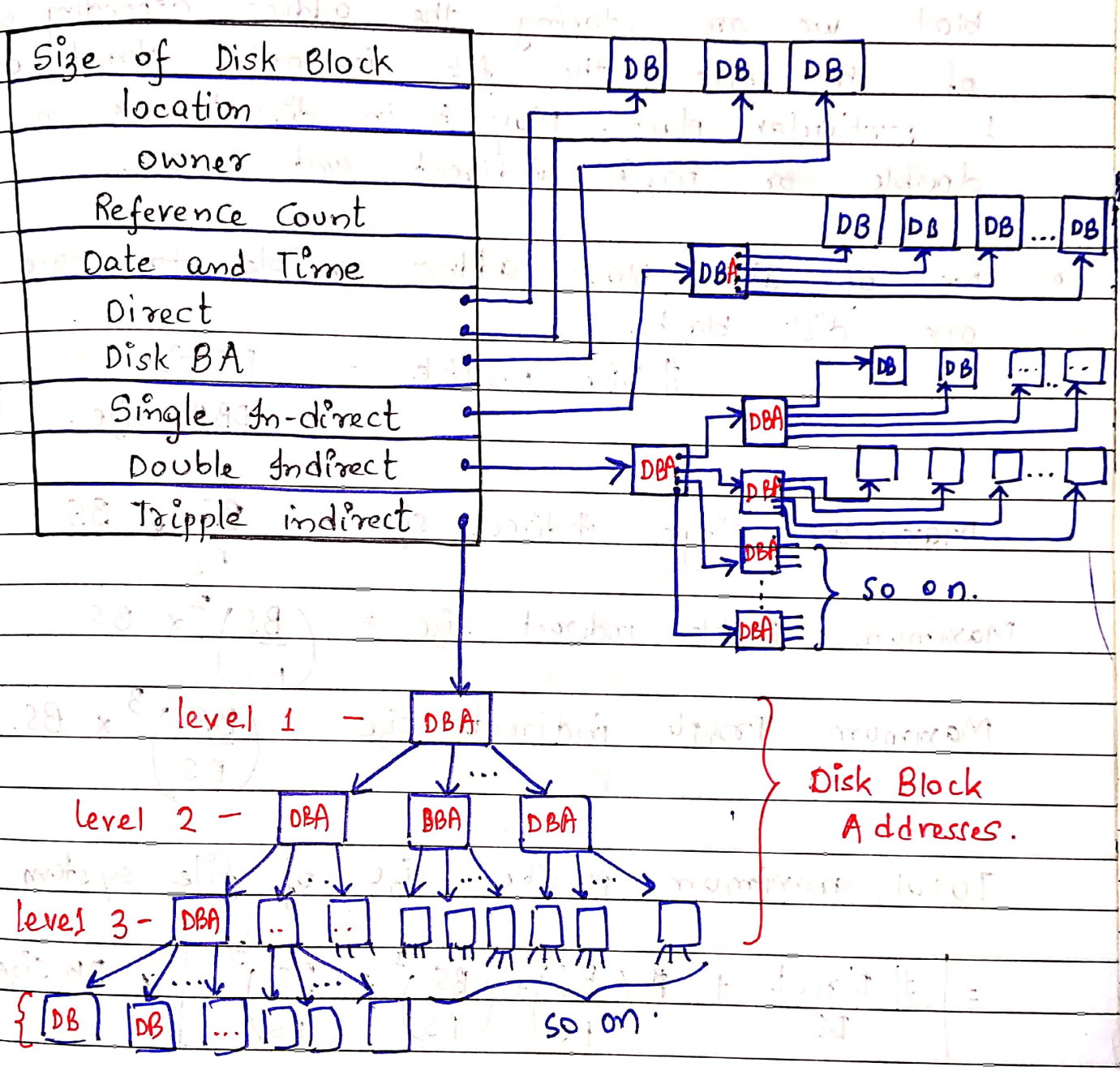


File name	DBA (Index)
abc.txt	4

- In the indexed allocation, atleast one disk block is used just to store the DBA of the file.
- Every file is associated with its own index node
- If the file is very very large, then one disk block may not be sufficient to store the disk block of the file.
- If the file is very very small then its best wasted disk block, just to store DBA.
- ~~unix~~

UNIX INODE IMPLEMENTATION

- Structure of i-Node :



- All the disk blocks are used some for data storage and some for address storage.
- inode is wrt every file and for every file we have its own inode.

- inode is associated with various disk blocks. In some disk blocks we stored the data and in some disk block we are storing the address depending on size of the file, the file is only stored only in 1 perticular place. May be in direct, or in single, double or triple indirect and so on.,

- No of disk block address possible to store in one disk block.

$$\#\text{DBA possible} = \frac{\text{DB size}}{\text{DBA size.}} = \left(\frac{BS}{ES}\right)$$

$$\text{Maximum single indirect size} = \frac{BS}{ES} \times BS$$

$$\text{Maximum Double indirect size} = \left(\frac{BS}{ES}\right)^2 \times BS.$$

$$\text{Maximum Tripple indirect size} = \left(\frac{BS}{ES}\right)^3 \times BS.$$

Total maximum possible size of file system

$$= \left[\#\text{Direct}_{DB} + \left(\frac{BS}{ES}\right) + \left(\frac{BS}{ES}\right)^2 + \left(\frac{BS}{ES}\right)^3 \right] \times \text{DB size}$$

(Ques) Consider the unix iNode 12 direct DBA, one singk indirect, one double indirect one tripple indirect. DBA size is 32 bits and block size is 1KB.

$$\left(\frac{BS}{ES}\right)^3 = \frac{1KB}{32\ bit} = (256)^3$$

$$= (16777216) \times 1\ KB = \approx 16\ GB.$$

Ques. A file system with 300 GB use a file discriptor with 8 direct Block address, 1 indirect block address, 1 double indirect. Size of each Block is 128 B and DBA size is 8B. Max possible file size in this system is.

$$\frac{BS}{ES} = \frac{128}{8} = 16$$

$$\therefore (8 + 16 + 256) \times 2^7\ B = 35840\ B.$$

$$= 35840.$$

$$\approx 35\ KB.$$

Ques. A unix inode has 10 direct, 1 single indirect, 1 double, 1 tripple. BS = 1KB ES = 4B.

$\left(\frac{BS}{ES} = 256\right)$.

Max possible file size = 2^{34} bytes. ✓

or (16 GB).

10 direct

Ques) In a perticular unix OS, each block size is 1024 byte. Each node has one single, one double, one tripple indirect. Also each block can contain addres of 128 Blocks. Which of the following is approx maximum size of a file in the system?

$(128)^3 \times 2^{10}$

$2^{21} \times 2^{10}$

$= 2^{31}$

$= 2$ GB.

Disk Free Space Management

- consider disk of size = 20MB
  Disk Block Size = 1KB
  DBA Size = 16 bits

  # Blocks = 20 K Blocks.
  = 20480 Blocks.
  some used for DBA
  some used for data.

- There are two basic approaches for disk free space management.
- Free list approach.
- Bit map approach.

Free List approach : some disk blocks are used to store the free disk block address.

DBA possible in 1 block = $\frac{DB}{DBA} = \frac{1024\ B}{2\ B}$

= 512 Addresses.

∴ for 20480 entries,

we require $\frac{20480}{512}$ = 40 Blocks required.

Bitmap Approach : only 1 binary bit is used to represent each block.

Binary Bit.

0 (DB is free) 1 (DB is Busy)

1	0	0	1	0
0	0	1	0	0	0
1	1	1	0	0	0

2560 Byte.

Ques Consider disk size = 128 MB.
DB size = 2 KB.
DBA size = 2 Bytes.

Disk blocks that will be consumed for free space management using free list approach is ?

(A) 64
(B) 128
(C) 32
(D) None.

$\frac{2^{27}}{2^{11}}$ Bytes $= 2^{16}$ blocks.

2^{17} Bytes.

$\therefore \frac{2^{16}}{2^{10}} = 2^6 = 64$. A.

Disk consumed to implement free list approx is = 64 x 2 KB

$= \frac{128 \text{ KB}}{128 \text{ MB}} = 0.001$.

= 0.1%

Disk blocks for bitmap:

2^{16} bits = 2^{13} Bytes

8 KB.

∴ 4 Blocks required.

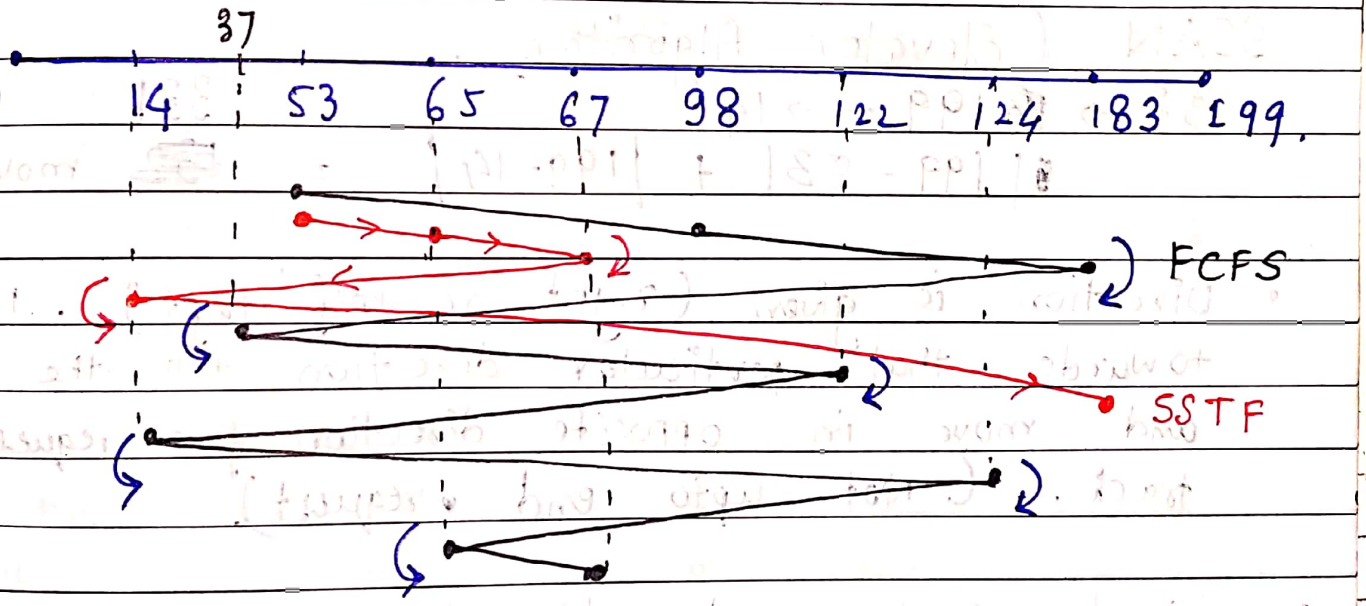
DISK SCHEDULING

Ex. Track Request : 98, 183, 37, 122, 14, 124, 65, 67
Initial head at : 53.

- The main goal of disk scheduling is to minimize the average seek time of the disk. Consider cylinders are numbered from 0 to 199.

FCFS : |98-53| + |98-183| + |37-183| + |122-37| + |122-14| + |124-14| + |65-124| + |67-65|.

= 640 movements.

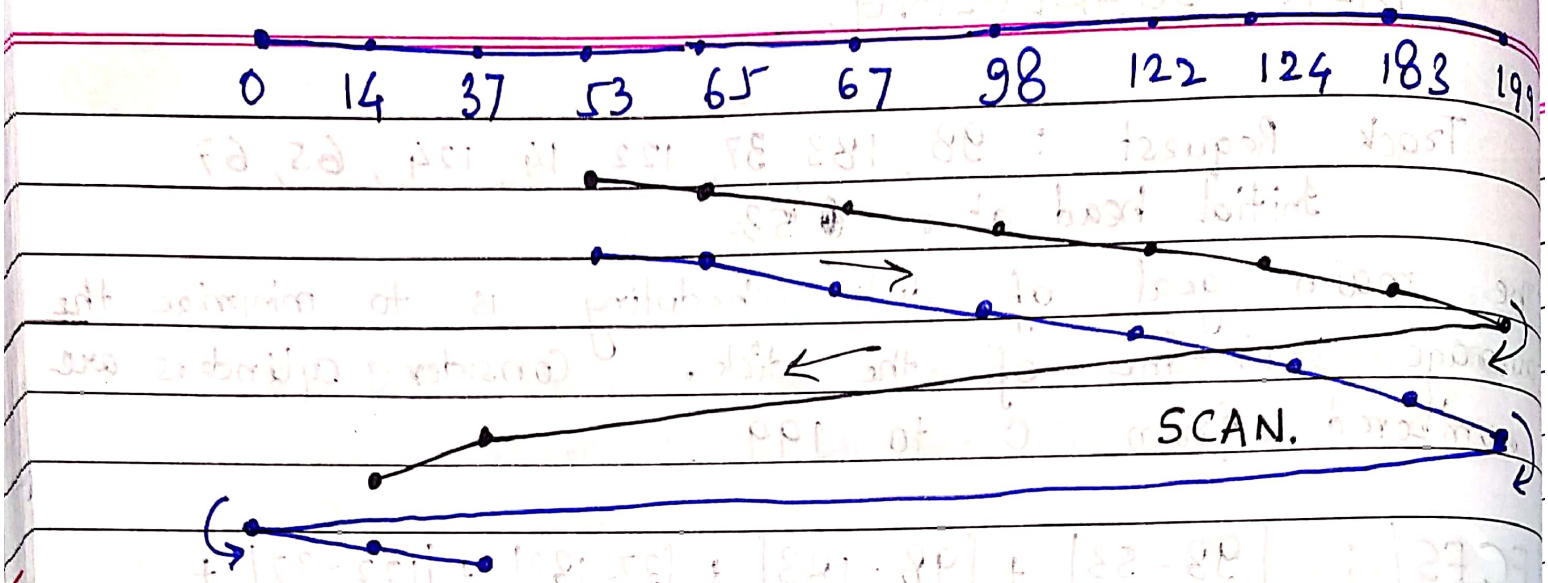


- Easy, starvation free.
- Increased total seek time, inefficient

SSTF : |53-67| + |67-14| + |14-183|

= 236 movements.
2 directional changes.

- Reduced seek time, increased throughput.
- Extra overhead to calculate closest req.
- STARVATION



- In SSTF, the request which are far from the head leads to starvation.

SCAN (Elevator Algorithm)

53 → 199 → 14 331

|199 - 53| + |199 - 14| = movements.

- Direction is given (right or left). Move towards that perticular direction till the end. and move in opposite direction for request track. (Not upto end request).

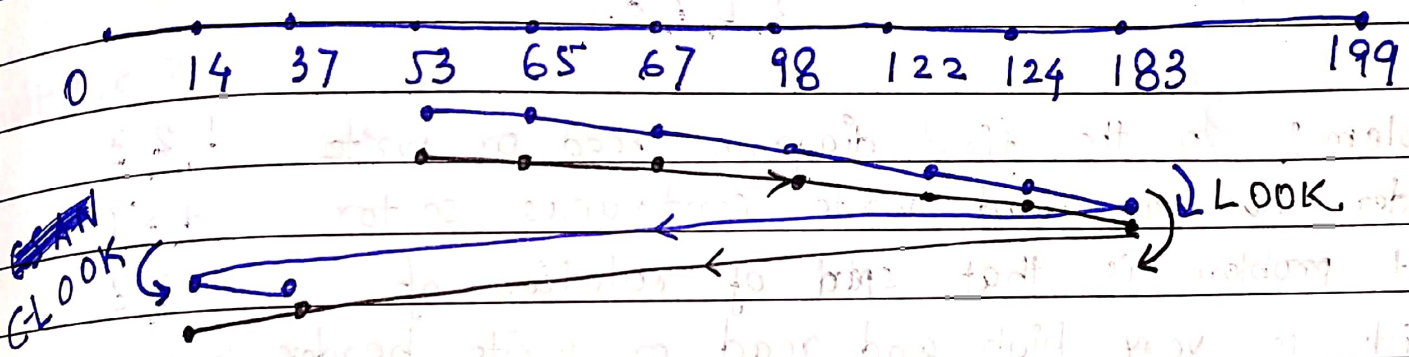
- Simple, easy, starvation free.
- long waiting time for cylinder just visited by the head.
- causes head to move till the end unnecessarily.

C-SCAN (circular scan)

(199 - 53) + (200 - 1) + 37 = 382 movements

- It provide uniform waiting time.
- provides better response time.
- It causes more seek movement as compared to scan algorithm.
- cause the head to move till the end of disk even if there is no request.

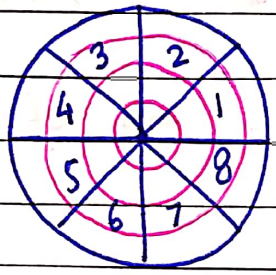
LOOK & C-LOOK (Improved SCAN & C-SCAN)



- (183-53) + (183-14) = 299 movement for LOOK.
- (183-53) + (183-14) + (37-14) = 322 for C-LOOK.
- Does not lead to starvation.
- Overhead of finding the end request.

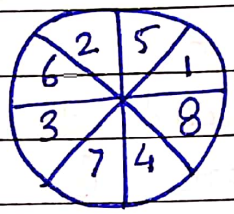
Disk Interleaving

No interleaving



1 rotation

Single interleaving

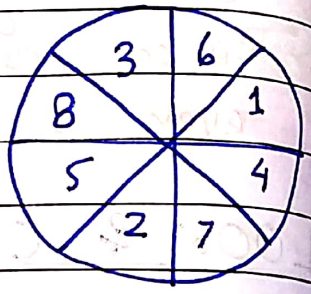


2 rotations

1, 2, 3, 4

5, 6, 7, 8.

Double Interleaving



2.75 rotation

1, 2, 3

4, 5, 6

7, 8

Problem: In the first diagram, read or write header has to read / write continuous sector but problem is that speed of rotation of disk is very high and read or write header is not able to catch next sector. ∴ It has to wait for completion of each rotation.

- Read or write header is not able to catch next sector because header is busy in processing and in calculating CRC and transferring data from previous read sector.

- **Solution:** The above problem is solved by disk interleaving. It makes header to comfortably CRC, transfer data and being able to catch the sector.

- In no interleaving disk, one rotation time is needed to read all its sectors of the track.

- In single interleaving disk, 2 rotation time is required to read all its sector of the track.
- In double interleaving disk, 2.75 rotation time is required to read all its sector of the track.

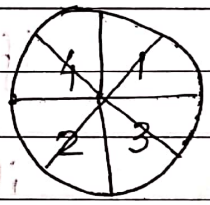
(Ques.) Consider the double interleaved disk whose track is divided into 2 KB sectors. The average seek time of the disk is 30 ms, and the rotation rate of the disk is 3600 rpm. The rotational latency is still considered as half rotation time.

(i) How much time is required to read all its sectors of the track is using double interleaving disk.

$$30 \text{ ms} + \frac{1}{2} \times \frac{60}{3600} + 2 \times \frac{1}{60} \text{ sec.}$$

3600 Rotation $\Rightarrow$ 60 sec.

$$\frac{1}{2} \text{ rotation} = \frac{1}{2 \times 60} = \frac{1}{120} \text{ sec.}$$



$$= 30 \text{ ms} + 8.33 + (2.75 \times 16.66) \text{ ms.}$$

$$= 84 \text{ ms}$$

(ii) What is the time difference to read all 8 sectors of the track if it is non interleaving disk.

$$= 30 + 8.33 + 16.66$$

$$= 55 \text{ ms}$$

$$\therefore \text{ Difference} = 84 - 55 = 29 \text{ ms}$$

what is DTR using double interleaving disk.

(45.84) ms to read 8 sectors.

45.84ms — 16 KB.

1 ms — $\frac{16}{45.84}$ KB.

10^3 ms — $\frac{16}{45.84} \times 10^{+3}$ KB.

= 349 KB.

Fork System call :

The fork is a system call which is used to create the child process.

```
main ()
{   int pid;
    pid = fork ();
  if ( pid < 0 )
      { printf ( "fork fail" ) ; }
  if ( pid == 0 )
      { print ("child process")}
  else { print ("parent process") ;
}
```

- The fork function return a negative value if the child process creation is unsuccesful.
- The fork return a value 0 to the newly created child process.

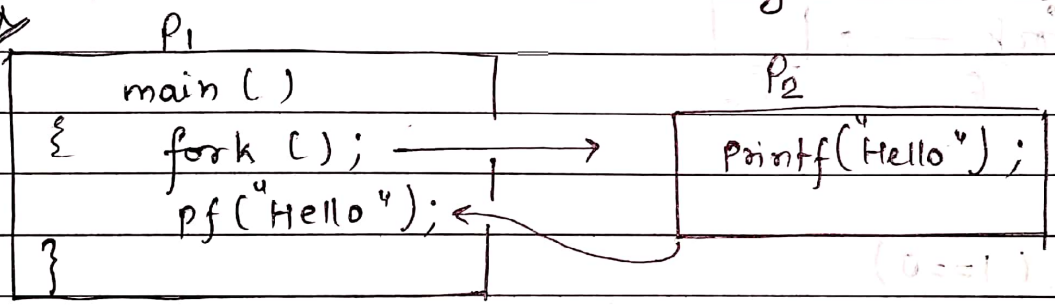
- The fork returns the pid (positive value > 0) of child process to the parent process.
- When child process is created by using the fork system call, the parent and child process will have the below address

(i) Relative address: It will be same for both parent and child.

(ii) Absolute address: It will be different for both.

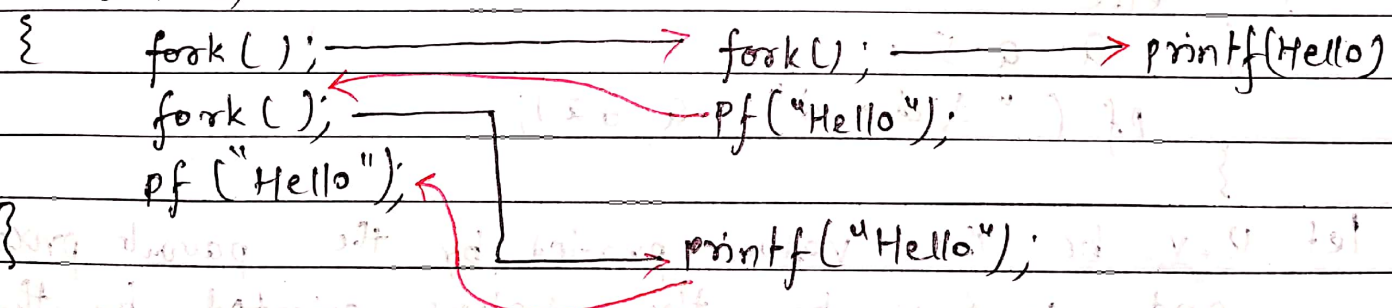
- If we try to print the address using & then it will print relative address (Logical address).

Eg



Prints Hello two times.

Eg



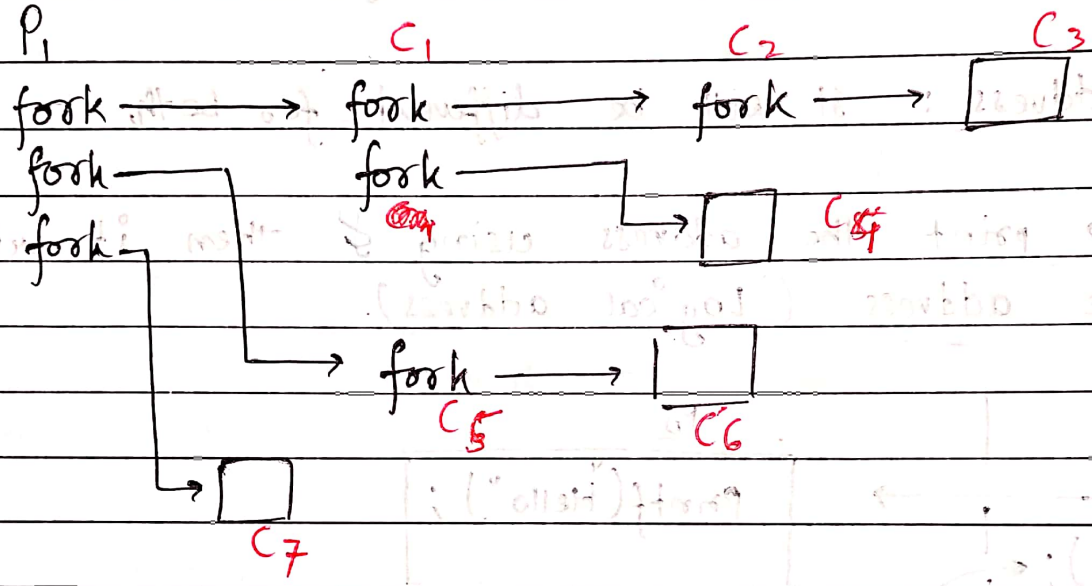
Prints "Hello" 4 times.

child processes = $2^n - 1$ where n = No of fork calls.

Ques.) A process executes the code :

```
fork();
fork();
fork();
```

the total no of child process are $2^3 - 1 = 7$ ✓



Ques.)

```
if ( fork()==0)
{   a = a+5 ;
    pf ("%d %d" , a, &a);
}   else
{   a = a-5 ;
    pf ("\d %d %d" , a, &a);
}
```

let u, v be the values printed by the parent process and x & y be the values printed by the child process.

~~u = 10~~ v = y

x = u+10

Note : The physical address of 'a' in parent and child must be different but our program access virtual address (assuming we are running on OS that uses virtual memory.) The child process gets an exact copy of parent process and virtual address of a does not change in child process. Therefore we get same address in both parent and child.

Ques.) Consider following code: o/p : 1111

```
int main()
{   if (fork() || fork())
        fork();
    printf("1");
}   return 0;
```

Ques.)

```
int main()
{   if (fork())
    {   if (!fork())
        {   fork()
            printf("1");
        }
    else
        {   printf("2");}
    }
else
    {   printf("3");}
printf("4");
}
```

o/p : 2 4 1 4 1 4 3 4.

Date
Page

Ques.)
```
int main()
{
    if (fork() && (!fork()))
        { if (fork() || fork())
            { fork(); }
        }
    else
        printf("2");
        return 0;
}
```

o/p : 2222222.

Ques.) 4 file of size = 11050 Bytes
Need to be stored: 4990 Bytes
for storing these 5170 Bytes
many files on disk, 12640 Bytes.
we can use either
100 disk blocks or 200 Bytes disk blocks.
But cannot mix blocksize. Each block
requires 4 Bytes of Bookkeeping informat
and that also needs to be stored on the disk.
Thus the total space used, to store a file
is the sum of the space taken to store
the file and the space taken to store Bookkee
-ping information, for the block, allocated for
storing the file. The Disk block can store
either book keeping information of a file
or data from a file but not both.
What is the total space required for
storing the files using

Sol.) 100 KB Disk Blocks

111 × 4 = 444 Bytes.
5 Blocks.

116 Blocks.

50 Blocks = 200 Bytes
2 Blocks

52 Blocks.

52 Blocks = 208 Bytes
3 Blocks

55 Blocks.

127 Blocks = 508 Bytes
6 Blocks.

133 Blocks.
356 × 100
35600 Bytes. //

200 KB Disk blocks.

56 Blocks = 224
2 Blocks

58 Blocks

25 Blocks → 100 Bytes
1 Blocks

26 Blocks

26 Blocks = 104 Bytes
1 Block

27 Blocks.

64 Blocks = 256 Bytes
2 Blocks

66 Blocks.
177 × 200
35400 Bytes.

Ques) The address sequence generated by tracing a particular program executing in a pure demand paging system, with 100 records per page. with 1 frame is recorded as follows. What is number of page faults.

0100* 0200* 0430* 0499 0510* 0530
0560 0120* 0220* 0240 0260 0320*
0370

Ques.) A CPU generate 32 bits virtual address. The page size is 4 KB. The processor has a TLB which can hold a total of 128 page table entry.

2^{12} offset

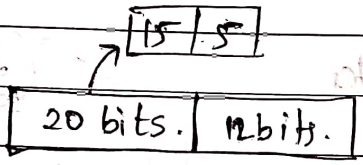
2^7

apoorv135@gmail.com

and is 4 way set associative mapping. The minimum size of the TLB tag is ?

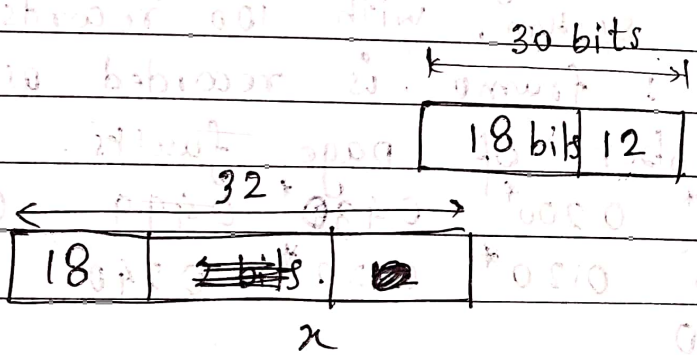
$$\#\text{sets} = \frac{128}{4} = 32 \text{ sets} = 2^5$$

TLB tag = 20 − 5 = 15 tag. //



2^{20} pages.

(Ques.) In a virtual memory system, size of virual address is 32 bits. Size of physical addres is 30 bits. Page size is 4 KB. Size of each page table entry size is 32 bits. The main memory is byte addressible. Which one of the following is the max no of bits that can be used for storing Protection and other information in each page table entry.



memory.

$PS = 2^2 B.$

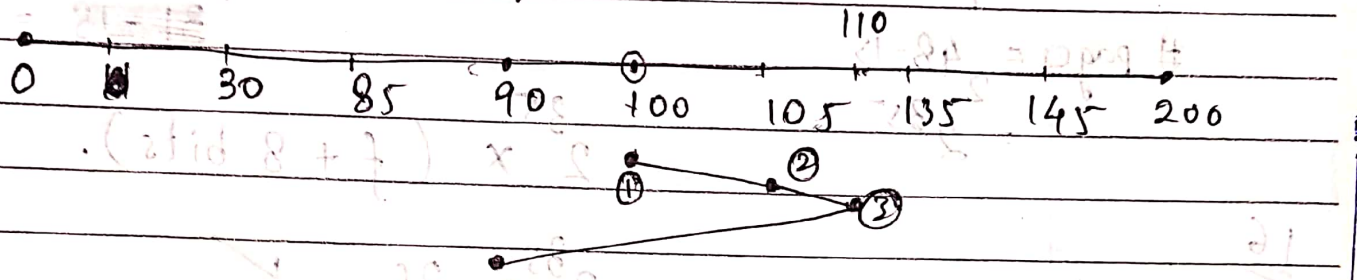
$\#\text{pages} = 2^{28}$ pages.

$x = 32 - 18$

$= 14$ bits.

Date
Page

Ques Suppose a disk has ~~200~~ 201 cylinders numbered from 0-200. At some time the disk arm is at cylinder 100, and the request for cylinder is 30, 85, 90, 100, 105, 110, 135, 145. If SSTF algorithm is being used for disk access the req for cylinder 90 is serviced after servicing the request.



Ques Consider a virtual page system with page size as 2048 Bytes. Each virtual address in the form A, B where A = page number & B is page offset. Find the actual address of the virtual address (3, 512). Consider the following lookup table.

2^{11}

virtual Page	Actual Page
0	2
1	3
2	—
3	4.

Actual page = 4

= 2048 × 8192

8192 + 512

= ~~3584~~ 8704 ✓

Ques.) Consider a logical address space of 2^{48} Bytes. and page size = 8 KB. = 2^{13} Bytes. Physical memory = 2 GB each page table entry also indicate 8 bits for own purpose, then what is the size of page table.

35 bits.	13 bits.

2^{31-13} = 18 bits

pages = $48-13$

$= 2^{35}$

$2^{35} \times (f + 8 \text{ bits})$.

16

$2^{35} \times 26$ ✓

Ques.) A processor uses two level page table both are stored in main memory.

VA = PA = 32 bits.

Memory is byte addressible. for a virtual to physical translation, first 10 bits used for page table, next 10 bit used for level 2 page table. Remaining used for page offset. Page table entry size is 4B each.

LAS

10	10	12

Processor uses TLB with hit ratio 96%. cache memory is used with hit ratio 90%.

$T_m = 10$ ns

$T_c = 1$ ns

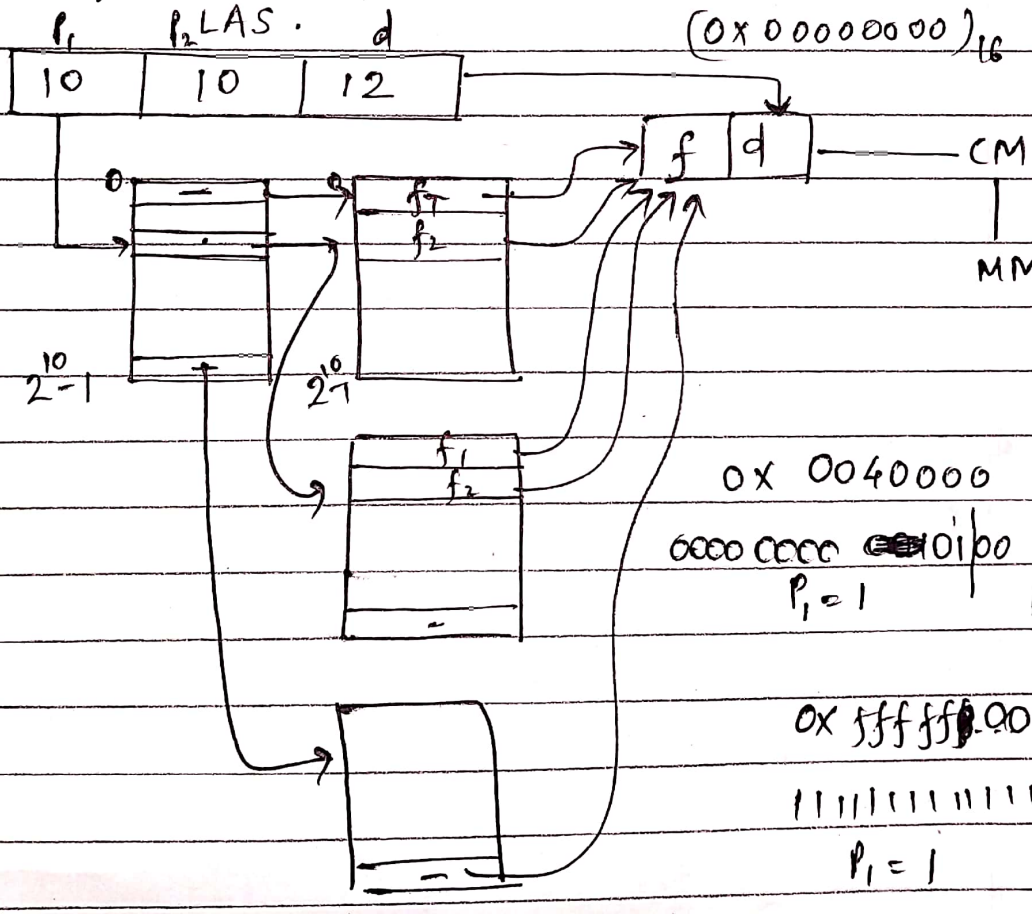
$T_t = 1$ ns

(i) find effective memory access time.

$$EMAT = 0.96 \times (0.9 \times 1 + 0.1 \times 11) + 0.04 \times (1 + 10 + 10 + 0.9 \times 1 + 0.1 \times 11)$$

$$= 1.92 + 0.92 \approx 2.84 \text{ ns.}$$

(ii) Suppose a process access 2 continuous code pages at virtual address 0X00000000
and 2 contiguous Data page at virtual address 0X00400000.
and a stack page starting at virtual address 0X ffffff000
The amount of memory required for storing the PT of this process is ?



page tables = 4
entry size = 4 B each.
∴ Total memory required = 4 × 1024 × 4 B
= 16 KB