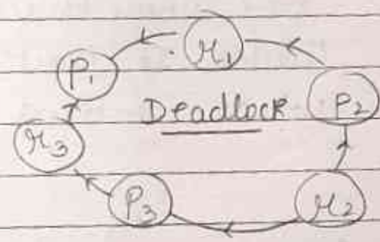
Date 21/9/23
Page

# Unit-3

## Deadlock.

When several process compete for limited no. of res
if the resource is not available then the process en
into waiting state.
If the process gets enable to change its waiting state because the resource requested by it are held by another waiting process i.e. called deadloc




System model for the deadlock
There are 3 steps:
Step 1) Every process will request for the resource
Step 2) If it is available then process will use the resource
Step 3) After use process must release the resource.

Necessary and sufficient condition for deadlock:
1) Mutual Exclusion: Atleast one resource type which be used in non sharable mode i.e. mutual Exclusive

2) Hold and Wait: A process is currently holding atleas resource and requesting additional resource i.e. held other process.

3) Non-preemption: A resource cannot be preempted i.e resource will be released by process after comp

task.

4) Circular wait: Each process must be waiting for a resource which is held by another process which in turn is waiting for the first process to release the resource.

22/23 Resource Allocation Graph (RAG)

The resource allocation graph is set of resource categories $\{R_1, R_2 \ldots R_j\}$ which appear as square nodes on the graph.

- DOTS inside the resource nodes indicate specific instances of the resource.
- A set of process $\{P_1, P_2 \ldots P_i\}$ is represented by circle node
- Request edge: A set of directed arcs from $P_i$ to $R_j$ indicates that $P_i$ has requested $R_j$ and is currently waiting for the resource to become available.
- Assignment edge: It indicates that the resource $R_j$ has been allocated to process $P_i$

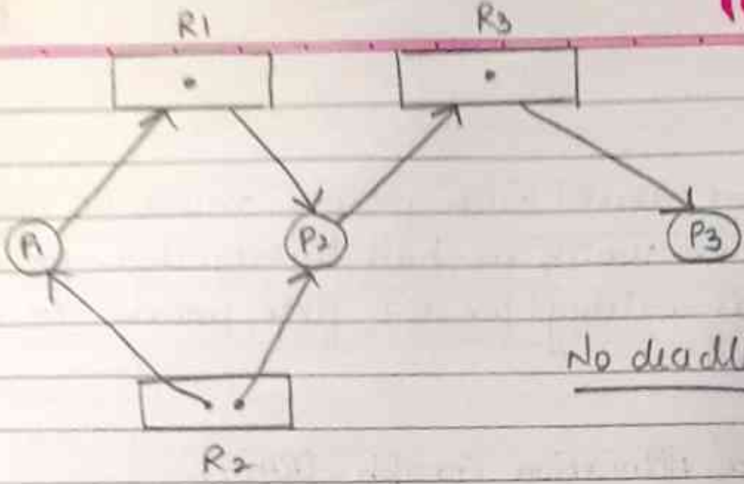
TE1: The request edge can be converted into assignment edge when the request is granted

E2: RAG contains no cycle then the system is not deadlocked

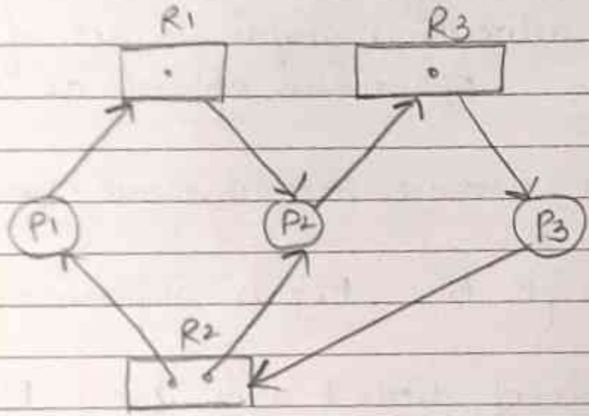
E3: If RAG contains cycle and each resource category contains only a single instance the a deadlock exist.

E4: If a resource category contain more than one instances then the presence of a cycle in RAG indicates the possibility of deadlock but does not give the guarantee of deadlock

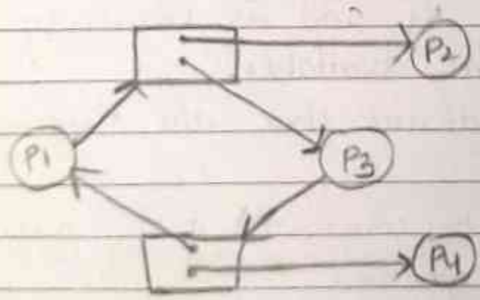
Page No.
Date :

1) 


No deadlock.

2) 


Deadlock

3) 


No deadlock

Deadlock Handling Methods.

1. Prevention
2. Avoidance
3. Detection & Recovery
4. Ignorance.

1. Prevention
It means we need to design such a system which ... atleast one of the four necessary condition of deadlock

ensures that Deadlock should not occur.

i) Mutual Exclusion

If a Resource is assigned more than one process that means the Resource is sharable, then Deadlock will not occur.

Note: There are some Resources that cannot be shared among several processes at a time

Eg: Printer, CD, Recorder etc.

ii) Hold and Wait

Process will acquire only desired Resources but before making any fresh request, it must release all the Resources that is currently held.

iii) ~~Fo~~ Non-preemption:

forceful Preemption: We allow a Process to forcefully preempt the Resource holding by other processes.

This method may be used by higher priority process or other System.

iv) Circular Wait

It can be eliminated by just giving a Natural Number ($f: N \rightarrow R$) to each Resource.

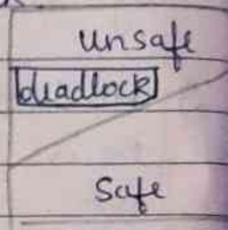
Allow every Process to make request either only in the increasing order or decreasing order of Resource num

If a process requires a Resource of lesser number in case of increasing order, then it must first release all the Resources larger than the required number.

11. Deadlock Avoidance

- The general idea behind Deadlock Avoidance is to pre[vent] Deadlock from ever happening.
- So, we need to find the safe state.
- Safe State: The system can allocate all Resources requested by all process without enter[ing] into Deadlock State

If the safe sequence does not exist then system is [in] Unsafe State, which may lead to Deadlock.




i Banker's Algorithm

- When a process starts up, it must state in advance, the max allocation of resources
- It may request upto the amount of available in Sys[tem]
- when a request would leave system in safe state, if [not] the process must wait until the request is grant[ed] safely

Based on Banker's Algo

/10/23 Q

| Process | Allocation | | | Max need | | | Available (CPU printer tapedriver) | | | Remaini[ng] (max need - | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | | |
| $P_0$ | 0 | 1 | 0 | 7 | 5 | 3 | 3 | 3 | 2 | 7 | 4 |
| $P_1$ | 2 | 0 | 0 | 3 | 2 | 2 | 5 | 3 | 2 | 1 | 2 |
| $P_2$ | 3 | 0 | 2 | 9 | 0 | 2 | 7 | 4 | 3 | 6 | 0 |
| $P_3$ | 2 | 1 | 1 | 4 | 2 | 2 | 7 | 4 | 5 | 2 | 1 |
| $P_4$ | 0 | 0 | 2 | 5 | 3 | 3 | 7 | 5 | 5 | 5 | 3 |

- Remaining need $\leq$ available
  - if condition is true
    - Available = Available + Allocation

$P_0$ 7 4 3 $\leq$ 3 3 2
False

$P_1$ 1 2 2 $\leq$ 3 3 2
True $\Rightarrow$ 3 3 2 + 2 0 0 = 5

P1   6 0 0 $\leq$ 5 3 2
False

P3 : 2 1 1 $\leq$ 5 3 2
True
$\Rightarrow$ 5 3 2 + 2 1 1
= 7 4 3

P4 : 5 3 1 $\leq$ 7 4 3
True
$\Rightarrow$ 7 4 3 + 0 0 2
= 7 4 5

P0 : 7 4 3 $\leq$ 7 4 5
True
$\Rightarrow$ 7 4 5 + 0 1 0
= 7 5 5

P2   6 0 0 ~~7 5 5~~ $\leq$ 7 5 5
True
$\Rightarrow$ 7 5 5 + 3 0 2

$\therefore$ total amount of resources = 10 5 7

Safe sequence : {P1, P3, P4, P0, P2}

Q

| Process | Allocation | | | Max need | | | Available | | | Remain need | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A | B | C | A | B | C | A | B | C | A | B | C |
| P1 | 1 | 0 | 1 | 4 | 3 | 1 | 3 | 3 | 0 | 3 | 3 | 0 |
| P2 | 1 | 1 | 2 | 2 | 1 | 4 | 4 | 3 | 1 | 1 | 0 | 2 |
| P3 | 1 | 0 | 3 | 1 | 3 | 3 | 5 | 3 | 4 | 0 | 3 | 0 |
| P4 | 2 | 0 | 0 | 5 | 4 | 1 | 6 | 4 | 6 | 3 | 4 | 1 |

P1   3 3 0 $\leq$ 3 3 0
True $\Rightarrow$ 3 3 0 + 1 0 1 = 4 3 1

P2   1 0 2 $\leq$ 4 3 1
False

P3   0 3 0 $\leq$ 4 3 1
True $\Rightarrow$ 4 3 1 + 1 0 3 = 5 3 4

P4   3 4 1 $\leq$ 5 3 4
False

P2   1 0 2 $\leq$ 5 3 4
True $\Rightarrow$ 5 3 4 + 1 1 2
= 6 4 6

P4   3 4 1 $\leq$ 6 4 6
True $\Rightarrow$ 6 4 6 + 2 0 0
= 8 4 6

$\boxed{\text{Safe seq : } \{P_1, P_3, P_2, P_4\}}$

III. Deadlock Detection

If deadlocks are not avoided then another approach used to detect when they have occured.

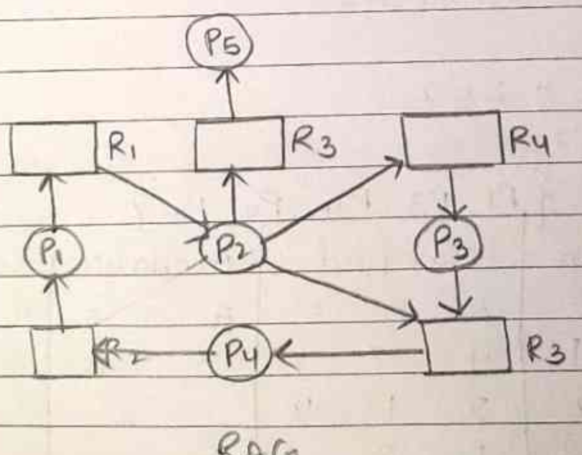
So, there are 2 process to detect the deadlock:

1) Single Instance of Each resource type
2) Several instance of Each resource type

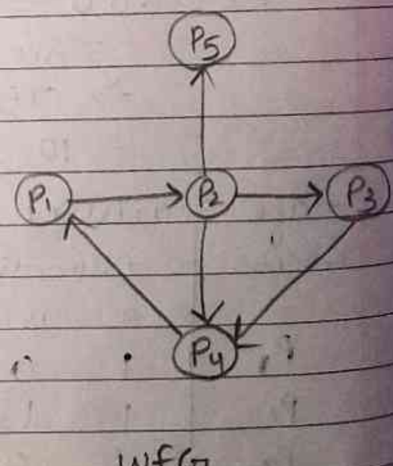
i) Single instance of each resource type:

- If each resource category has a single instance th we can use a variation of the resource allocation Graph that is known as Wait for Graph (WFG)
- A WFG can be constructed from a RAG by eliminat the resources and collapsing the associated edges.

for eg :




RAG




WFG

ii) Several instance of each resource type

The detection algorithm is same as the banker's algorithm

Step 1: The Banker's algorithm set fine

III. Deadlock Recovery.

When a deadlock detection algorithm determines that a deadlock has occured in the system. The system must recover by using 2 approaches for breaking a deadlock

1) Process Termination
2) Resource Preemption

1) Process Termination:

- Abort all the deadlocked process.
- Abort one process at a time until deadlock is eliminated

2) Resource Preemption:

- To eliminate deadlock using preemption we preempt some resources from processes and give these resources to other processes.

Sometimes, this method will raise three issues.

~~1) Selection o~~

Page No.
Date :

1) Selecting a victim : We must determine which resource and which processes are to be preem and also the order to minimize the cost.
2) Rollback : We must determine what should be do with the process from which resources are preempted. Simple idea is total rollback th process and restart it again.
3) Starvation : In a system it may happen that s process is always picked as a victim a result the process will never complete its t So, the solution for this prob is we must picked up a (victims) resource only a finite no. of times.

10/10/23 Process Synchronization

1. Inter - Process Communication (IPC)

It refers to a mechanism which allows commu -ion b/w two or more processes to perform the action simultaneously.

There are two types of processes

1) Independent Processes : which does not depend execution of other processes.
2) Cooperative Processes : These are the process whic can effect or get affected by other process du execution

Cooperative processes can share the data in terms variable, memory, code and resources

2. Race Condition

It is the situation arise due to concurrent executi more than one processes which are accessing a manipulating the same shared data and the of execution depends upon the specific order where the access takes place.

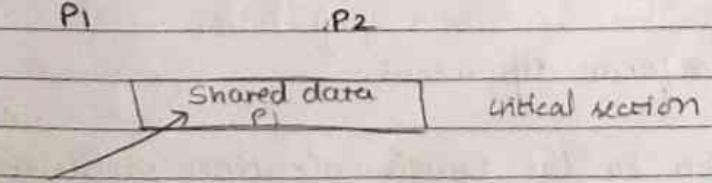
Eg: shared = 5

| P1 ~~5~~ 6 | P2 ~~5~~ 4 |
|---|---|
| x = shared | y = shared |
| x++; | y--; |
| sleep(1) | sleep(1) |
| shared = x | shared = y |

3. Critical Section
This is a piece of code which contains some shared code of variable which is accessible by each process concurrently in order to complete execution.
There must be only one process is allowed at the time in critical section otherwise more than one access may lead to inconsistency.

4. Mechanism to achieve synchronization
i Mutual Exclusion
ii Progress
iii Bounded Wait
iv No assumption related to the hardware

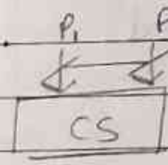
i) Mutual Exclusion.




P1 entered into critical section at the same time if P2 is requesting to enter into critical section then it will not allowed.

ii) Progress.

If no process is in the critical section and if one more process want to execute ~~their~~ their critical section then any one of them must be allowed to get into the critical section.



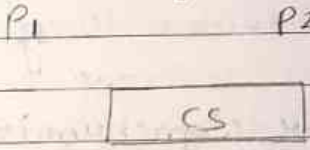

} No progress

iii) Bounded wait

After a process makes a request for getting into critical section their is the limit called how many times other process can get into the critical section before the process ~~of~~ request is granted.
So after the limit is reached system must grant the permission to get into the critical section.

condition: CS use 10 times only

| P1 | P2 |
|---|---|
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| | 0 |
| 200 | 1 |




iv) No assumption related to hardware.

{ mai nahi likh rahi ab... Thak gayi hun... }

No platform dependent.

5. Solution to the Synchronization problem:

There are 3 solution
- → Hardware solution
- → Software solution
- → strict alteration

## Inter Process Communication Synchronization Problem:

↳ Producer-Consumer Problem

```
int count=0;
void produce (void)
{ int itemp;
  while (True)
  { Produce. item (itemp);
    while (count = = n);
    Buffer [in] = item;
    in = (in+1) mod n;
    count = count + 1
  }
}
```

In [ ]

$\longrightarrow$ 
- $I_1$ Load $R_p$, M[Count];
- $I_2$ INCR $R_p$;
- $I_3$ Store M[count], $R_p$

```
void Consume (void)
{ int itemc;
  while (true)
  { while (count = = 0);
    itemc = Buffer(out)
    out = (out + 1) mod n;
    Count = count - 1;
    Process = item (itemc);
  }.
}
```

Load $R_p$, M[count];
DECR $R_p$; ←
Store M[count], $R_p$;

count = 0

out [ ]

Buffer

| | |
|---|---|
| 0 | $x_0$ |
| 1 | $x_1$ |
| 2 | $x_2$ |
| 3 | $\cancel{x_3}$ |
| 4 | |
| 5 | |
| 6 | |

Seq: Produce $I_1, I_2$ consume $I_1, I_2$
Produce $I_3$ consume $I_3$

P $I_1$ $I_2$ $\longrightarrow x_3$    P $I_3$    C = 4    * But; race around condition...

C $I_1$ $I_2$ $\longrightarrow \cancel{x_3}$    C $I_4$    C = 2

(Because 3 in Buffer and c = 4) ✗
(Because 2 in Buffer and c = 2)

* It is a multiprocess sync problem also known as Bounded Buffer Problem. It describes that 2 processes producer-consumer

who share a common fixed size Buffer. Producer process will produce some data and put it into the Buffer and Consumer process will consume the ~~p~~ data and remove the item from Buffer.

* Cond$^n$ for inconsistency:

a) Producer must not try any data item produce to the B, if Buffer size is full.

b) Consumer must not try to consume any data if Buffer size is Empty.

* Sol$^n$ of Producer:

1) Producer either go to sleep or discard the data if Buffer
2) Once the consumer removes an item from Buffer, it ~~notify by~~ notifies the producer to put data into Buffer.

* Sol$^n$ of Consumer:

1) Consumer can go to sleep if Buffer is empty.
2) Once the producer puts the data into Buffer it notifies Consumer to remove data from the Buffer.

16/10/23 Sol$^n$ to sync problem

1) Software Solution

Peterson's algorithm →

$P_0$ interested $\overset{P_0}{\boxed{f}}$ $\overset{P_1}{\boxed{F}}$ → both are false initially

Turn $\boxed{0}$ → 0 means $P_0$'s turn is first &
1 means $P_1$'s turn is first

Entry section

```
Entry Section (Process)                         // Mutual Exclusion ✓
{ int other;                                    // Bounded waiting ✓
  other = 1 - process;                          // Progress ✓
  interested (process) = true
  turn = process
  while (interested [other] == true && turn = process); }
exit section
{ interested [process] = false }
```

Note: It will not work for more than 2 processes.



It is used for implementing any 2 process only & it uses two variable i.e. turn & interested

TE : It does not gives the gurantee to work on multiple core & latest model of computers.

Hardware solⁿ for synchronization problem

1) Lock  2) Test & set  3) swap

* Test & set algorithm

```
lock = false
do{
while(Test and set(&Lock));
                         // critical section.
exit code { lock = false;
~~while (true);~~ }
boolean Test and set (boolean *target)
{ boolean rv = * target;
  * target = true;
  return rv; }
```

// Mutual exclusion
// Bounded waiting
// Progress

| lock True | target | rv |
|---|---|---|
| False | 1000 | False |
| 1000 | | |

* Strict Alternation

Turn variable (Software based, work for 2 processes)

for process $P_0$

```
try  → while(turn != 0);
       CS
exit → { turn = 1;
         exit; }
```

for process $P_1$

```
try  → while( turn != 1);
       CS
exit → {turn = 0;
        exit; }
```

* It is a busy waiting problem solution which can be implemented only for 2 processes

Page No.
Date :

## Semaphore

- They are integer values that are used to solve critical s[ection] problem by using two atomic operation: wait & Signal
- That are used for the process synchronization
- wait () → decrement opⁿ  Signal() → increment opⁿ.
- There are two types of Semaphore: Counting & Binary
  Range : Counting → -∞ to +∞ ; Binary → 0 to 1

17/10/23
- These are integers value Semaphore & have unrestricted value doma[in]
- Binary Semaphore are like Counting Semaphore but their values are r[estricted]
- wait () - P() - Down
- Signal () - V() - Up
- Counting Semaphore.

```
Down (Semaphore S)
{ value = S value -1;
  if (S value ≤ 0)
    { put process in suspended List; }
  else
    return; }
Up (Semaphore S)
{ { value = S value -1 ;
    if (S value ≤ 0)
      { Select a process from Suspended list
        wake up () }
}
```

* Reader - Writer Problem.

| P1 | P2 | |
|---|---|---|
| R | W | causes problem |
| W | R | causes problem |
| W | W | causes problem |
| R | R | — no problem. |

Reader

```
int rc = 0
Semaphore mutex = 1
Semaphore db = 1
void Reader(void)
{ while (true)
  { down(mutex)
      rc = rc+1
  if (rc ==1)
  then
     down(db);
     up(mutex)
       [dB]
   down(mutex)
     rc = rc-1
     if (rc== 0)
    then
     up(db);
     up(mutex);
     process data }
}
```

Writer

```
void write(void)
{ while (true)
   { down(db);
       [dB]
      up(db);
   }
}.
```

Q Consider a system with 'n' processes and 6 tape drives If each process requires 2 tape drives to complete their execution then what is the maximum value of 'n' which always ensures that deadlock free operations  3  4  5  None

Q Consider a system 3 processes each process require two unit to comp. their exetution Then what is the min. no. of resources that ensures deadlock free system.

Page No.
Date :

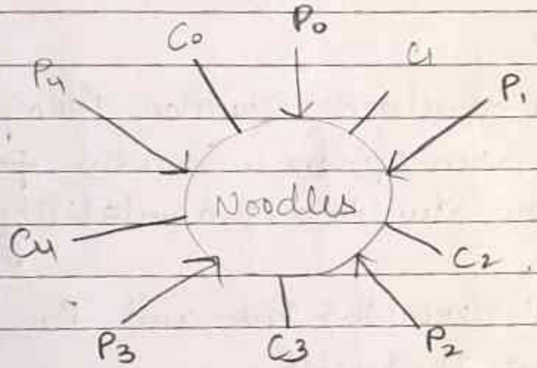
Q Each process $P_i$, where i=1 to 9 executes code
$P_i$
repeat
P(mutex) < CS V(mutex)
forever
The process $P_{10}$ execute the following code
repeat
V(mutex)
$\boxed{CS}$
V(mutex)
forever

$\boxed{Mutex = 1}$

What are max no of process that may present CS any point of time?
2   3   9 ✓   10

Dining Philospher's Problem
In this problem there are 4 philosphers are sitting a dining table and given chopsticks are $C_1$ $C_2$ $C_3$ $C_4$ eat food on the table. Each philospher requires 2 chopsticks at same time to eat.
At any step a philospher is either eating or thin
If philospher is eating, have to pick chopsticks from there left and one from there right
If philospher is thinking, it will keep the chop on the table.
for eg: Philospher $P_0$ wants to eat the food, it requires $C_0$ and $C_1$, as considered here I=0, as $0^{th}$ location.
Case I: In this case one by one philospher is ea

and thinking which is best case. If philosphur $P_0$ is eating with chopstick $C_0$ and $C_1$ and then put the chopsticks on table and then other philosphur $P_1$ has come to eat which requires chopsticks $C_1$ and $C_2$, as we see the chopstick $C_1$ is now available for $P_1$ because $P_0$ already put down the chopstick $C_1$ after eating.
So, in this case there is no problem that can occur.




③ Case 2: In this case, Philosphur $P_0$ is eating with a chopstick $C_0$ and $C_1$ and at the same time Philosphur $P_1$ enter to eat. So it requires $C_1$ and $C_2$. Now, in this case $C_1$ is not available for Philosphur $P_1$ because it already hold by the Philosphur $P_0$.
To deal with this situation binary semaphore is used.
When one Philosphur eat the food with needed chopsticks and case II situation occurs. Then it block that Philosphur otherwise it allow the another philosphur if it is require other chopsticks that are available on the table.
So at sametime, multiple philosphur can eat If their required chopsticks are available on table.

Page No.
Date :

```
Void philospher (void)
{ while (true)
  { thinking ()
    wait (take chopstick (si));
    wait (take chopstick (si+1) % N));
    eat ()
    signal (put. chopstick (i));
    signal (Put. chopstick (i+1) % N);
  }
}
```

NOTE: To avoid the deadlock situation Philospher $P_4$ need to swipe there semaphore. So, when ~~people~~ $P_4$ comes it will not allow due to not availability of their first semaphore.
So, $S_4$ will remains one and $P_3$ will use $S_4$ and $S_3$ and put it back.
So, again $S_3$ and $S_4$ will be available. So further $P_2$ can use $S_2$ and $S_3$ and run further process

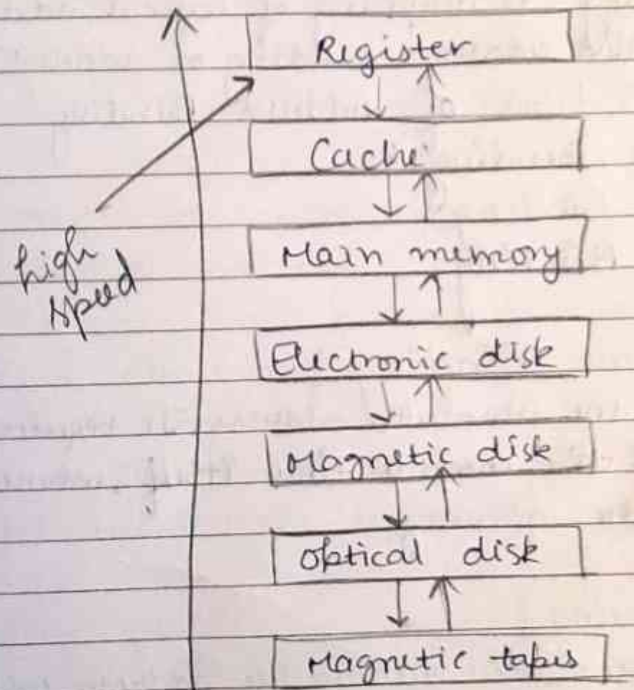
| | | |
|---|---|---|
| $P_0$ | $S_0$ | $S_1$ |
| $P_1$ | $S_1$ | $S_2$ |
| $P_2$ | $S_2$ | $S_3$ |
| $P_3$ | $S_3$ | $S_4$ |
| $P_4$ | $S_0$ | $S_4$ |

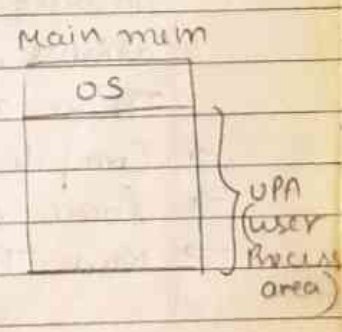
## Unit-4

### Memory management.

- To achieve a degree of multiprogramming and proper utilisation of memory, memory mgmt is required.
- The main memory comprises a large array or group of words or bytes each with its own location
- The primary purpose of the computer system is to execute the program in the main memory

Memory hierarchy




In a multiprogramming OS resides in a part of mem and the rest is used by multiple processes.
The task of subdividing the memory among different process is called memory mgmt and it is a method in OS to manage operations b/w main mem & secondary mem
The main aim of the memory mgmt is to achieve efficient utilisation of memory.

Logical address v/s Physical address

* Logical address i.e. generated by CPU and also k as virtual address.

Absolute address * Physical address is the main memory address.

(LAS) * Logical address space : It is a set of all logical address generated by CPU i

(PAS) * Physical address space : Set of all PA.

Address Binding : A mapping of logical address to a p address is called address binding or address mapping.

There are 3 types of Address Binding :

→ Compile time Binding :
→ Load time Binding
→ Run Time Binding

Compile Time Binding :

It generates the absolute address. It requires that it be known at the compile time itself, where will a p resides in the memory.

Load Time Binding :

The compiler generates relocatable address which are co to absolute address at the load time.

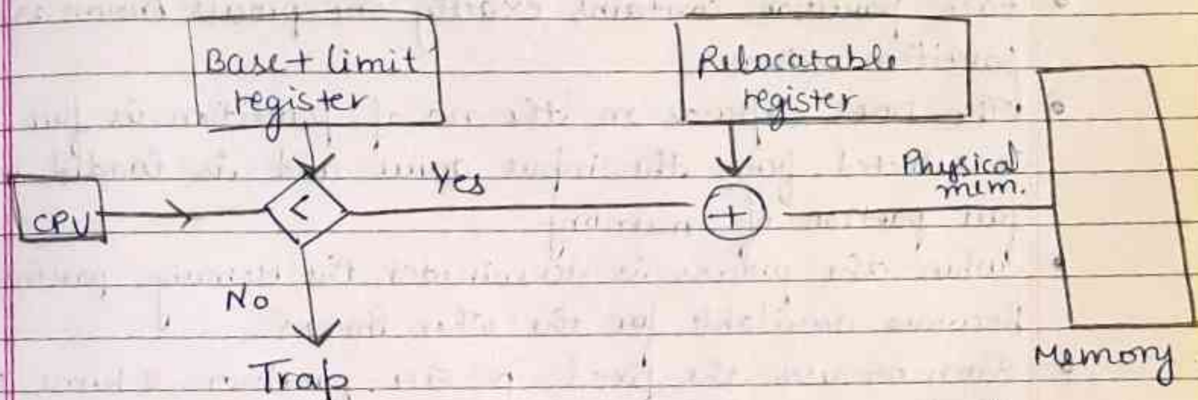
Run Time Binding :

The process may be moved from one mem. segme to other then binding must be delayed ~~a~~ until runti

Memory Protection Hardware

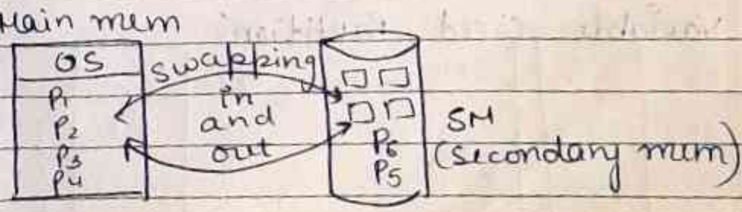
Base address:

It is starting address of any mem. block and limit register gives the limit of any mem. block




- The protection means providing security from unauthorized access or usage of memory.
- The OS can protect the memory with the help of Base & limit register.
- The limit register is always the fencing register
- Base register holds the smallest legal physical mem. address while the limit reg. contain the size of process.

Swapping




Memory allocation

1) Contiguous allocation
   - fixed partition scheme
   - Variable partition scheme
2) Non-contiguous allocation.
   - paging
   - Segmentation
   - paging with segmentation

2/11/2023

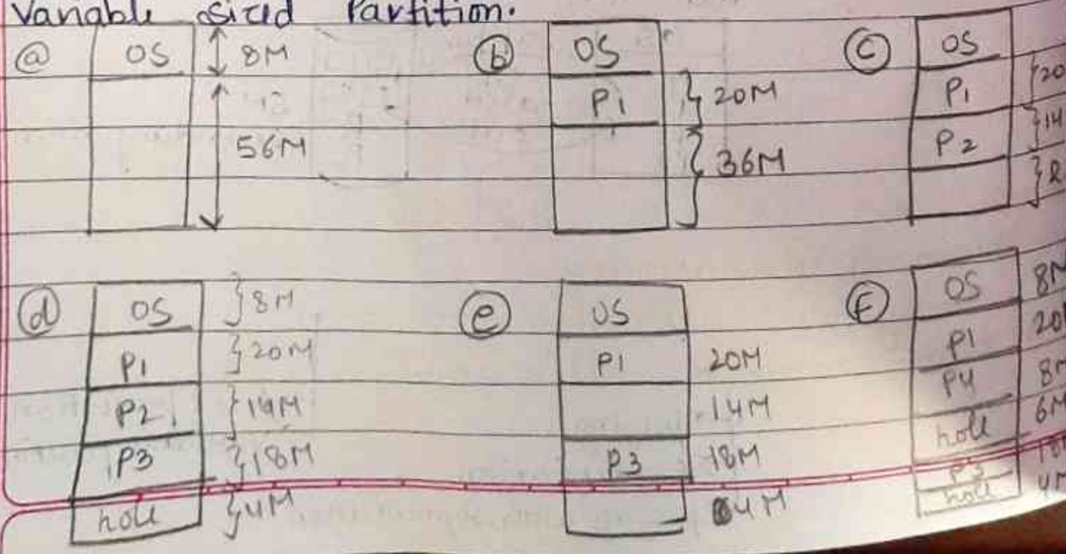
Page No.
Date :

Memory Allocation

1) Contiguous Memory Allocation.

- Single contiguous allocation is a simple memory alloca that requires no special hardware for allocating memory. It is divided into the no. of fixed sized partition.
- Each partition contains exactly one process known as fixe partition scheme
- The DOM depends on the no. of partition is free, the is selected from the input queue and is loaded into free partion of memory.
- When the process is terminated, the memory partition becomes available for the other Process.
- Batch OS uses the fixed 'p' size partition scheme. OS k the record of mem. allocation and de-allocation in t form of table.
- When the Process arrives in the System and needs Me then OS search for the large space for the Process. If i available, the Process is allocated to that Memory.
- There are 2 difficulties with equal size partition:
  - A program may be too big to fit into the Partition.
  - Advantage: Simple to implement and less overhead

Variable sized Partition:



(g)

| | |
|---|---|
| OS | |
| | 20M |
| P4 | 8M |
| | 6M |
| P2 | 18M |
| | 4M |

(h)

| |
|---|
| OS |
| P1 |
| P2 |
| hole |
| P3 |
| P4 |

- for the Memory Allocation, there are 3 schemes: first fit, Best fit, Worst fit.

1) first fit.
Allocating the first hole, that is big enough.
Searching can start either at the begining of the set of holes or where the previous first fit search ended

2) Best fit
Allocate the smallest hole that is fit enough. We must search the entire list, unless the list is ordered by size. This ~~to~~ strategy produces the smallest left-over hole.

3) Worst fit
It allocated the largest hole. This strategy produces the largest leftover hole.

- first fit and Best fit are better than the worst fit, in terms of storage utilization.

fragmentation.
Memory fragmentation is of 2 types.
(1) Internal fragmentation (2) External fragmentation

1) Intenal fragmentation.
There is a wastage of space internal due to the fact that the block of data loaded is smaller than the

Page No.
Date :

partition size.

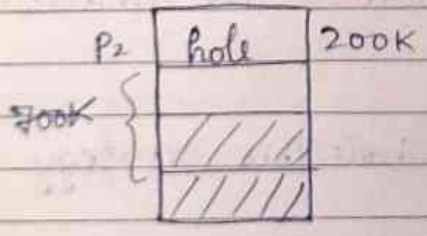
for ex: The size of hole is 1054 bytes and the Process request for 1052 bytes. If we allocate hole then ~~that~~ there is a hole created of 2

External fragmentation.

when in a total mem. space exist to satisfy a re but it is not contiguous storage fragmented into large no. small holes.

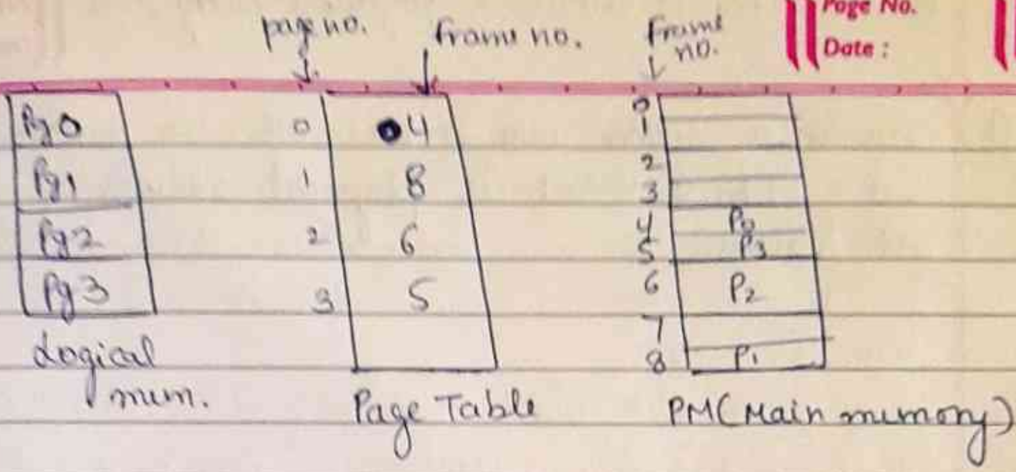
for eg: There is a hole of 200K, 500K in vari size partition scheme and next process for 700K of memory. Actually, 700K of is free which satisfy the request but contiguous. So, this is an External frag of mem. and Best-fit & Worst-fit can by the external fragmentation

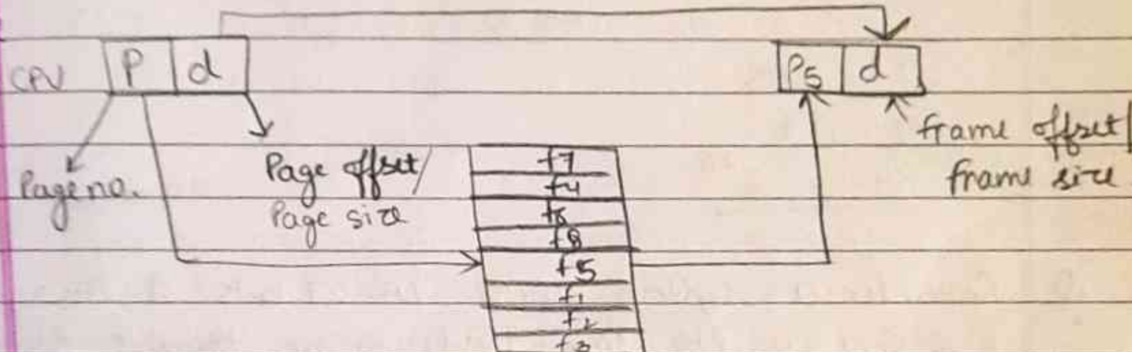
7/12/23 Compaction




Paging

1) It is a non-contiguous memory allocation and phy memory is divided into physical address space and lo memory is divided into logical address space and lo address space is having two fields:
→ no. of pages
→ page offset
and physical memory is divided into 2 parts
→ frame no.
→ frame offset




① $LA = 7$ bit $= 2^7 = 128$

$PA = 6$ bit $= 2^6 = 64$

Page size $= 8$ bytes $= 2^3$

calculate no. of pages & no. of frames

no. of pages $= \dfrac{LAS}{\text{page size}} = \dfrac{2^7}{2^3} = 2^4$ = 4 bits

no. of frames $= \dfrac{PAS}{(\text{frame size})} = \dfrac{2^6}{2^3} = 2^3 = 3$ bits

CPU | 4 | 3 |

Q. $LA = 4GB$ — $2^2 \cdot 2^{30} = 32$ bits

$PA = 64MB$ — $2^6 \cdot 2^{20} = 26$ bits

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

No. of Pages = ? $= \dfrac{2^{32}}{2^{12}} = 2^{20} = 1MB = 20$ bits

No. of frames = ? $= \dfrac{2^{26}}{2^{12}} = 2^{14} = 16KB = 14$ bits

No. of entries in PageTable = ? $2^{20}$ & Size of PT = ? $2^{20} \times 2^{14}$

PT size = No. of entries * pagetable entry size

Page No.
Date :

Q. Consider a system with a PT with 4K entries LA = 29 bits. What is physical address if system 512 frames

LA = 29 bits

no. of frames = 512 = $2^9$

no. of pages = $2^{12} = \frac{LA}{\text{page size}}$

page size = 29 - 12
= 17

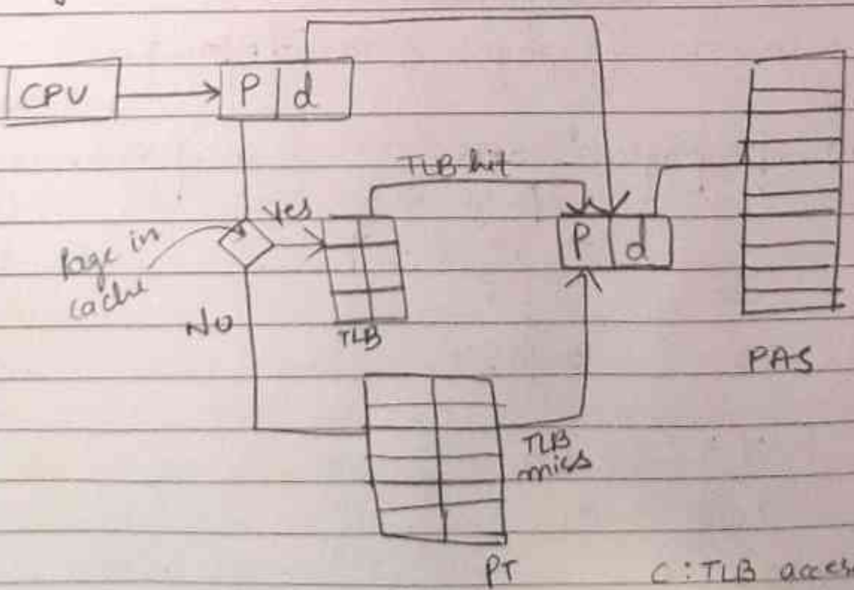
$2^9 = \frac{PAS}{2^{17}}$

Answer

$\underline{PAS = 2^{26}}$

Q. Consider a system with LA = 27 bits & PAS = 128 Page size is 8K. Page Table entry require 32 bits what is the approximate size of PT in bytes

Paging with TLB (Translation look a side Buffer)




C : TLB access time
M : Mem.

Effective memory access time

$EMAT = h * (C + M) + (1 - h) * (C + 2M)$

$h$ ↑ TLB hit ratio

$(1-h)$ ↑ TLB miss ratio

Consider a system which has TLB access time (C) = 20ns and main mem. access time (M) = 100 ns and hit ratio is 85% and the system uses 1 level paging. Then what is EMAT with TLB?

$$EMAT = h^*(C+M) + (1-h)^*(C+2M)$$
$$= 0.85^*(20+100) + 0.15\,(20+200)$$
$$= 0.85\,(120) + 0.15(220)$$
$$= 102 + 33$$
$$= 135\ ns$$

Types of paging (names online search)

Unit-5

Locality of Reference

Demand Paging and Page fault

The process of loading the pages into main mem. on demand.

Page Fault Scheme

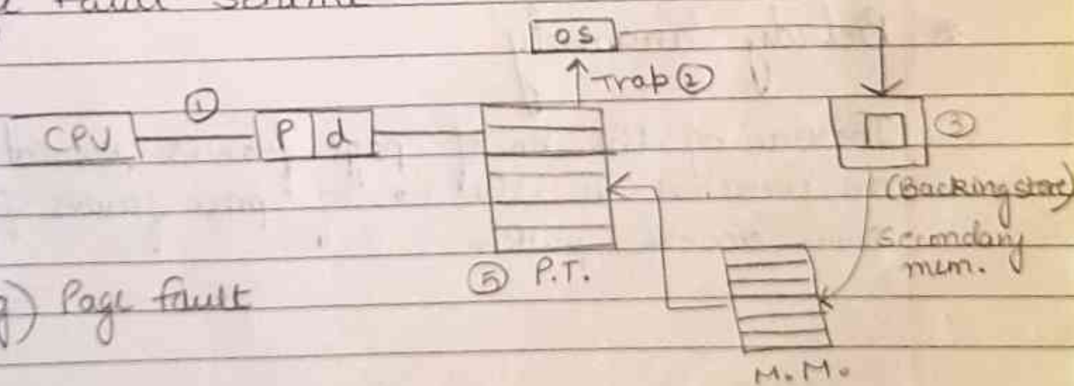



fig) Page fault

Page replacement Algorithm

FIFO (first in first out)

Page No.
Date :

Q1 Ref. string = 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0
1 7 0 1

frame size = 4
Calculate no. of page faults.
Algo FIFO

frame size = 4

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | | |
| | | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 7 | 7 | 7 | | |
| 7 | 7 | 7 | 7 | 7 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | | |
| | | | | * | | * | | | | | | * | * | | |

Page fault = 10

frame size = 3

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | | | |
| | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 0 | 0 | | | |
| 7 | 7 | 7 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 7 | 7 | 7 | | | |

Page fault = 15

* Belady anomoly

Increasing the no. of page frames results in an increase in the no. of page faults for a given access pattern.