unit-3

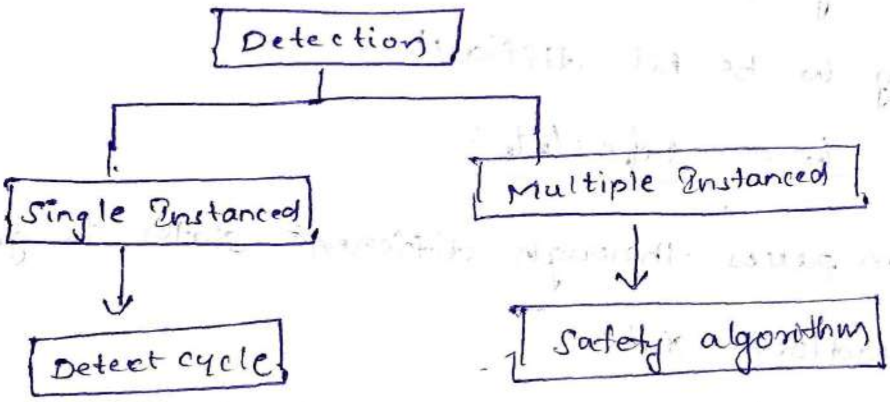
1) Deadlock detection and its recovery:-

→ In this approach, the os doesnot apply any mechanism to avoid or prevent the deadlocks.

→ Therefore the system considers that deadlock will definetly occur.

→ In order to get rid of deadlocks, The os periodically checks the system for any dead lock.

→ In case, it finds any deadlock then the os will recover the system using some recovery techniques.



→ In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock.

→ In multi-instanced resource type graph, detecting a cycle is not just enough. we have to apply safety algorithm on the system.

In order to recover the system, for deadlocks either OS considers resources or processes.

For Resource :-

Preempt the resource :-

We can snatch one of the resource from the owner of the resource and give it to the other process with the expectation that it will complete the execution and will release this resource sooner.

→ Choosing a resource which will be snatched is going to be bit difficult.

Rollback to a safe state :-

→ System passes through different states to get into the deadlock state.

→ The operating system can rollback the system to previous state. which is safe.

→ For this purpose, OS needs to implement check pointing at every state

For process :-

Kill a process :-

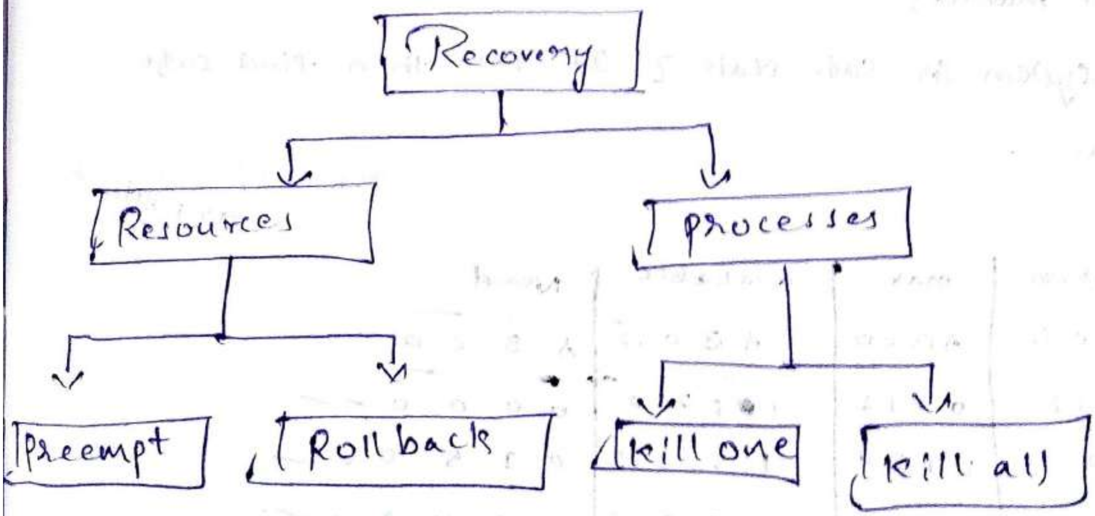
→ Killing a process can solve our problem but the bigger concern is to decide which process to kill.

→ Generally, operating system kills a process which has done least amount of work till now.

kill all process :-

→ This is not a suggestible approach but can be implemented if the problem gets very serious.

→ Killing all process will lead to inefficiency in the system because all processes will execute again from starting.



2)

Banker's Algorithm:-

→ If resources are having multiple instances, Banker's algorithm is used to prevent or avoid the dead lock.

Ex:-

	Allocation				Max				available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

~~Need = Allo - max~~

1) Need matrix?

2) Is system in safe state? If yes then find safe sequence.

Sol:-

av + Alloc

	Allocation				max				available				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0 ✓	0	0	1	2	0	0	1	2	1	5	2	0	0	0	0	0
P_1 ✓	1	0	0	0	1	7	5	0	1	5	3	2	0	7	5	0 ×
P_2 ✓	1	3	5	4	2	3	5	6	2	8	8	6	1	0	0	2 ×
P_3 ✓	0	6	3	2	0	6	5	2	2	14	11	8	0	0	2	0 ✓
P_4	0	0	1	4	6	6	5	6	2	14	12	12	0	6	4	2 ×
									3	14	12	12				

Total = Allo + Avail.

need = max - allo

allo = max - need

~~allo = max~~

max = allo + need

Safe sequence :-

P_0 P_2 P_3 P_4 P_1

algorithm :-

Input :- processes

- any 2 out of 3 would be given (max, need, allocation)
- avaliable or total no. of resources.

step-1 :- flag[i] = 0, for i = 0 to (n-1) & find need [n][m] = max [n][m] - allocation[n][m]

step-2 :- find a process P_i such that :- flag [i] = 0 & Need <= avaliable

step-3 :- if such process exist then,

flag [i] = 1, avaliable = avaliable + allocation

go to step 2.

Step-4 :- If flag [i] = 1 for all i then system is in safe state otherwise unsafe state.

Ex:- 2:-

Process	Max				Allocation				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	6	0	1	2	4	0	0	1	3	2	1	1
P_1	2	7	5	0	1	1	0	0				
P_2	2	3	5	6	1	2	5	4				
P_3	1	6	5	3	0	6	3	3				
P_4	1	6	5	6	0	2	1	2				
	12	22	21	17	6	11	9	10				

Using Banker's algorithm, answer the following questions

- i) How many resources of type A, B, C, D are there?

  ii) What are the contents of need matrix?

  iii) Find if the system is in safe state? If it is, find the safe sequence

Sol:-

i) No. of resources = ~~max~~ alloc + available

$$
\begin{array}{rrrrr}
= & 6 & 11 & 9 & 10 \\
+ & 3 & 2 & 1 & 1 \\
\hline
 & 9 & 13 & 10 & 11 \\
\hline
 & A & B & C & D
\end{array}
$$

ii)

Process	max				allocation				avaliable				Need			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	6	0	1	2	4	0	0	1	3	2	1	1	2	0	1	1
P_1	2	7	5	0	1	1	0	0	7	2	1	2	1	6	5	0
P_2	2	3	5	6	1	2	5	4	8	4	6	6	1	1	0	2
P_3	1	6	5	3	0	6	3	3	8	10	9	9	1	0	2	0
P_4	1	6	5	6	0	2	1	2	8	12	10	11	1	4	4	4
									9	13	10	11				

<u>Safe Sequence:-</u>

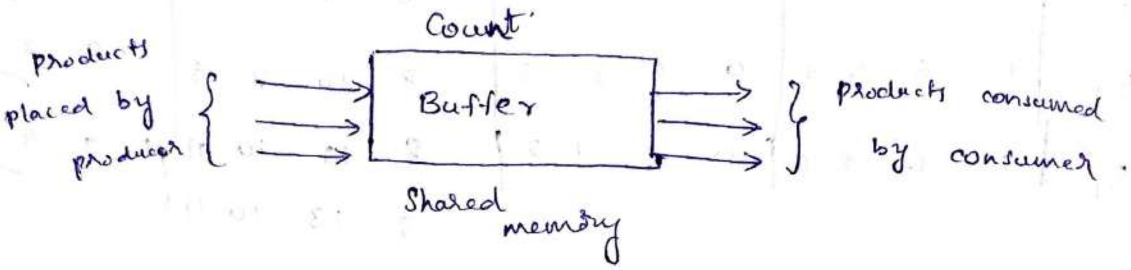
P_0, P_2, P_3, P_4, P_1

$\therefore$ yes, the system is in safe state & the safe sequence is P_0, P_2, P_3, P_4, P_1 //.

3)
Producer- consumer problem :-

→ 2 process :- 1) producer :- produces product & place it in a buffer (memory).

2) Consumer :- Consumes the product which was placed in buffer by the producer.



Condition at producer side :- (count ++)

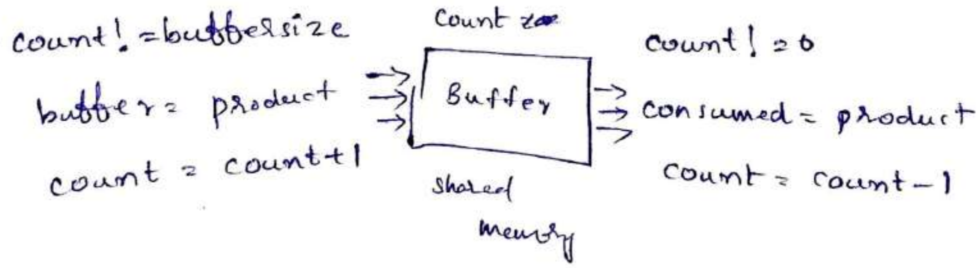
→ Before placing a product into a buffer, producer have to check whether buffer is full (or) not.

Condition at consumer side :- (count --)

→ Before consuming product from buffer, consumer have to check buffer is Empty (or) not.

Producer:

consumer



Producer

$I_1 : R_1 = count$

$I_2 : R_1 = R_1 + 1$

$I_3 : count = R_1$

Consumer

$I_4 : R_2 = count$

$I_5 : R_2 = R_2 - 1$

$I_6 : count = R_2$

Instruction Execution:-

Let us assume count = 4.

$I_1 : R_1 = count \Rightarrow R_1 = 4$

$I_2 : R_1 = R_1 + 1 \Rightarrow 4 + 1 = 5$

$I_4 : R_2 = count \Rightarrow R_2 = 4$

$I_5 : R_2 = R_2 - 1 \Rightarrow R_2 = 3$

$$\left.\begin{array}{l} I_6 : count = R_2 \Rightarrow count = 3 \\ I_3 = count = R_1 \Rightarrow count = 5 \end{array}\right\} \text{Data inconsistent.}$$

Race condition

Process Synchronization:-

Process Synchronization is the coordination of execution of multiple processes ~~in a~~ such that no two processes access the same shared resources and data. There are 4 sections:-

1. Entry section
2. Critical section
3. Exit Section
4. Remainder section

Race condition:

A race condition is an undesirable situation that occurs when a device (or) system attempts to perform two (or) more operations at the same time

4) Dining philosopher problem:-

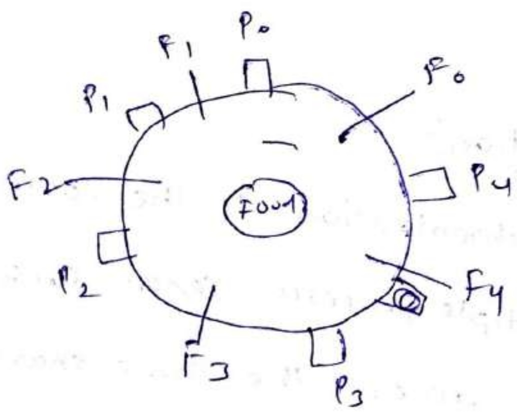
-> Assume that 5 philosophers P_0, P_1, P_2, P_3, P_4 are sitting around a circular dining table.

-> Dining table has 5 chopsticks and 2 bowl of rice in the middle.

-> philosopher has to either eat (or) think.

-> When philosopher wants to eat, he use 2 chop-sticks.

-> when he wants to think he keeps down both chopsticks

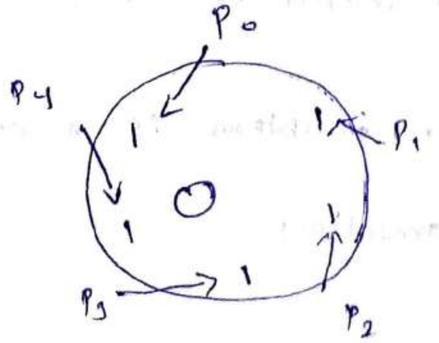


Problem:- Develop an algorithm where no philosopher starves i.e., every philosopher should eventually get a chance to eat

=> chopstick [5] initialized to 1

structure of philosopher i;

```
do
{
  wait (chopstick [i]);
  wait (chopstick [(i+1)] % 5);
  //eat;
  signal (chopstick [i]);
  signal (chopstick [i+1] % 5);
  // think;
} while (true);
```



Here, P_0 has taken one fork and waiting for another fork from P_1 until it releases.

∴ All are in deadlock condition.

Scanned with ACE Scanner

Solution of dining philosopher problem:

Shared data:

state [5]

self [5] : semaphore; {initialize to 0}

mutex : semaphore; {intialize to 1}

left : (i+4) % 5 ; {left neighbour}

right : (i+1) % 5 ; {right neighbour}

5) Reader-writer problem:-

Reader-writer problem is one of the famous operating system problems related to synchronization where multiple processes can be found reading or writing on the same shared resource

→ The reader-writer problem is a classical problem of process - synchronization

Ex :

Case	process 1	process 2	Allowed / not allowed
Case-1	writing	writing	Not allowed
case-2	Reading	writing	Not allowed
case-3	writing	Reading	Not allowed
case-4	Reading	Reading	allowed

→ The Solution of Readers and writers can be implemented using binary semaphores

1. wait(s)

```
{
while (s<=0);
s--;
}
```

2. signal(s)

```
{
s++;
}
```

Code for reader process:

```
static int readcount=0;
wait(mutex);
read count++;
if (readcount==1)
{
   wait (write);
}
signal(mutex);
wait(mutex);
read--;
if (readcount==0)
{
 signal(write);
}
signal(mutex);
```

Code for writer process :-

```
wait(write);
    WRITE INTO THE FILE
    Signal(wrt);
```

→ If a writer wishes to access the file, wait operation is write semaphore which decrements write to 0.

2-Marks :-

1) operations on process :-

Process operations refer to the activities performed on processes in an operating system.

→ These operation include :- 1) creating
2) Terminating
3) Suspending
4) Resuming
5) Communicating

2) Co-operating processes :-

The processes that depend on other processes. They work together to achieve a common task.

3) Inter- process communication :-

It is a type of mechanism usually provided by the operating system. The main goal of this mechanism is to provide communications in between several processes

4) Critical section :-

The critical section is a code segment where the shared variables can be accessed.

5) Mutual Exclusion :-

It is a program object that prevents multiple threads from accessing the same shared resource.

6.) Message passing :-

It refers to the sending of a message to process which can be an object, parallel process, subroutine, function or thread

7) Strict alternation :-

The software mechanism implemented at user mode

Scanned with ACE Scanner