3. Critical section problem

* A critical section is defined as part of the program where the shared memory is to be acess.
* Here, the shared memory can be a variable or file
* if we take producer consumer problem then acessing count variable is Ntg but a critical section.
  - It denotes How many variables are present in the buffer
* suppose, in the buffer contain 2 ~~variables~~ items. the count variable value is 2
* suppose the buffer contains No items. the count variable value is 0.
* in producer consumer problem the producer produce an item, and the corresponding item stored in the buffer and after that the count variable will be incremented by 1 (count++) count = count + 1.
* in similar, The consumer problem consume an item from the buffer, and it decrements the value of the count by 1
  so we can write the statement of count--
  itmeans count = count - 1
* so, in Producer consumer problem acessing count variable is Ntg but critical section
* so, why it is denoted as critical section because if Producer process tries to acess count variable, Then consumer process should not acess count variable.

* Here accessing count variable is Ntg but critical section. that means if producer process tries to acess critical section, Then consumer process not allowed to acess critical section
* that means the acessing to the critical section must be mutual exclusive
* it means the critical section should contains only 1 process at a time.
* if producer process is in the critical section, The consumer process is not allowed to ~~attain~~ enter the critical section
* likewise, when consumer process is in the critical section, Then producer process is not allowed to enter in the critical section.
* if both producer and consumer process acces count variable at a time, at it produces wrong results.
* so inorder to solve that problem we are using critical section.
* This is about what is critical section
* Next, how ~~to~~ we can solve critical section
* inorder to solve the critical section problem. we have 2 solutions
  1. Software solution —is Ntg but→ Peterson's solution
  2. Hardware solution —is Ntg but→ synchronization Hardware
* So we can solve the critical section in 2 ways. The 1st way software, 2nd way is Hardware approach

* Any solution to the critical section must satisfy 3 requirements. it may be either Peterson's or synchronization Hardware.

* it has to satisfy 3 requirements

1. mutual exclusion
2. Progress
3. Bounded waiting

* So, any solution to critical section Problem, must satisfy These 3 requirements.

mutual exclusion : it means at a time only one process is to be in the critical section.

* when one process is in the critical section, Then other process should not be allowed to attain the critical section
* only when the first process comes out from the critical section Then only the second process enter into the critical section.
* So, we have to solve mutual exclusion border to solve critical section problem.

Progess : let critical section doesn't contain any process at a set of process such as $P_0, P_1, P_2, P_3 \ldots$ like several processes

* All the several processes waiting for entering into critical section.
* So, The decision about which will enter into the critical section should be taken in finite amount of time.

time. indefinate amount of time

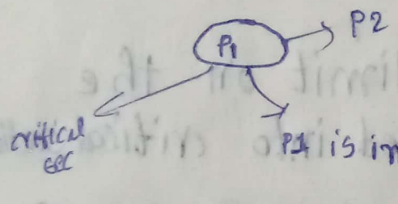
* This is Ntg but 2nd Requirement for solving a critical section Problem.
* let us assume that a critical section is free. The set of processes $P_0, P_1, P_2, P_3 \ldots$ are interested to entire in crit
* Then the decision about which Processes will enteres into the critical section. that decision should be taken in a finite amount of time
* it is there should be Progress → like order

3. Bounded waiting : Bounded means waiting. There should be a bound or limit on The No. of times a process can enter into critical

limited ← → The size of buffer is fixed

There 2 types bounded and unbounded waiting → size of the buffer infinite ∞

* when other process request to entering into critical Section. let's take an example



P2 try to enter critical section, but critical sec have only 1 Process

* assume P1 process completed it's execution. so, P1 comes out of the critical section.

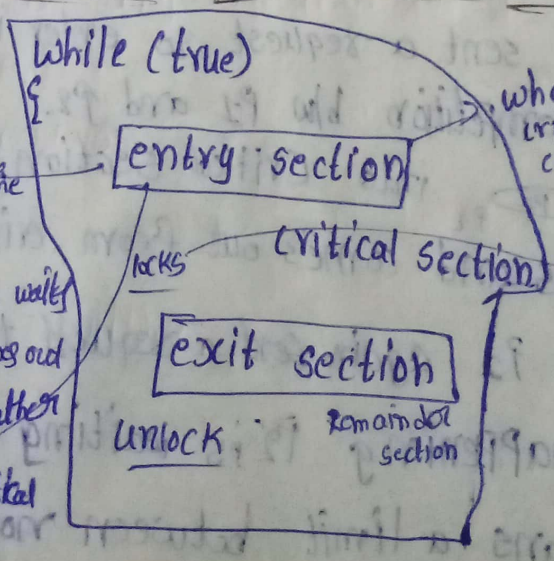
P1

* and P1 again sent a request to enter into the critical
* so, there is a competition b/w P1 and P2.
* and P1 enter P2 into Critical Section. so P1 completed all it's operations and come's out from critical section.
* let us assume P1 again sent a request to enter into crit
* Here what is happening P2 is waiting so much time
* Here Bounded means a limit between no. of times a process

have to be entered into critical section.

* when other process puts a request to enter into the critical section
* Here P2 sends a request to enter into critical section. but P1 entered into critical section
* So Here we set a limit to process P1 enter into a critical section
* so, it has to be entered into critical section for a Bounded (or) limited no. of times.
* let us, assume the limit is given as 3 times. so that P2 has to be entered into the critical section only 3 times after that
* we have to given the chances to P2 so that, is Nothing but bounded waiting
* So there should be a bound or limit on the no of times a process can entered into critical section
* when other process puts a request to enter into the critical section

General structure of process P1



Suppose critical section is free it enters into the critical section

If it is Not free, the process waits to that corresponding comes out

it contains code whether the process enter into critical section or not

whenever a process enter critical section, first it check critical section (or) having

process

set process is enter in critical

it locks

* inorder to solve the critical section problem there are 2 solution
1. ~~Peter~~ software solution → it means Peterson's solution
2. Hardware solution → it means synchronization hardware

* peterson's solution is important to solve critical section problem
* software means a program, program means collection of instructions
* we can uses the peater son solution only when there are 2 processes which involves critical section.
* it means the 2 process shares the common code.
* suppose the system contain more than 2 processes the peterson's solution is not possible.
* let us assume that we have 2 processes P0, P1
* Here we using 2 datastructures.

① int turn → it is an integer variable.
→ it indicates whose turn it is to enter into critical sec
it store value suppose value is 0, P0 enter 1, P1 enter

② boolean flag [2] → is an array size is 2
mean true or false
flag[0] is true it is enter int critical sec
flag[0] is false it is not intrested to enter into crit

Structure of process Pi

```
while (true)
{
    flag [i] = true;      // intrested to enter
    turn = j;             // changes given to j, Pi gives to j
    while (flag [j] == true && turn == j);   // entry section
        critical section
    flag [i] = false;     // exit section
}
```

condition is F. The control comes from critical section
if enter section it completes operation it comes out sec

both are True P(j) enter critical otherwise Pi enter into critical

Pi comes out critical sec, Pj can enter into critical sec

It execute in exit section.

P0 :-

```
flag[0] = true;
turn = 1;
while (flag[1] == true && turn == 1);
    critical section
    flag[0] = false;
```

P0 interested to critical sec

It gives chances to P1

P0 is checking whether P1 is interested to the critical section (or) whether P1 is already in critical sec or not.

if both are true P1 is in critical section (or) P1 is intrested to enter critical section

→ we have to wait P1 complete operation and P1 comes out.

* suppose
→ if one of the condition is false

*suppose P1 is not interseted to enter

*so, the control enters into critical sec

*next, P0 completed all it's operations

*while P0 is compled all operation. Then

in exit section it executes an operation called flag[0] = false

it specifies P0 is No longer belongs to critical sec

*Now process P1 is enter into csec like write code P1

P1

```
flag[1] = true;
turn = 0;
while (flag[0] == true && turn = 1);
    critical section;
    flage[1] = false;
```

* we know that any solution for a critical section problem we Satisfy3 requirements.

1) Mutual exclusion →
2) Progress ——→ out of all Process. it enters only interseted flag[2] → initial
3) Bounded waiting.

to P1, P0 are not interested

flag[0] = false → P0 no longer belongs to crit
→ P1 enter into critsec.

Synchronization Hardware

*) The critical section solution have 2 ways
1) Pealerson's solution
2) Synchronization Hardware soln

* Here we are using 2 instructions
1) TestAndSet() instruction
2) swap instruction

Definition of TestAndSet() instruction : → machine instruction

TestAndSet() returns a boolean value

```
boolean TestAndSet(boolean *target)
{
    boolean rv = *target;
    *target = true;
    return rv;
}
```

it mean using call by reference

it perform any modification on formal argument it reflected on actual argument also.

it is pointer. so will pass adress as the Parameter Here

test and set accept boolean as argument

return value

test and set accepts true value it return true other w false

that value is written if its true written True otherwise False

false → critical sec.

critical sec.

comes out critical sec.

Mutual Exclusion with TestAndSet()

```
do
{
    while (TestAndSet(&lock));
        // critical section

    lock = false;
}
```

semicolon

condition is true, control is stay on loop

condition is F, control comes out from loop to critical

initial value of lock

False
1000

suppose adress

it executed it goes to test And set function

adress = 1000

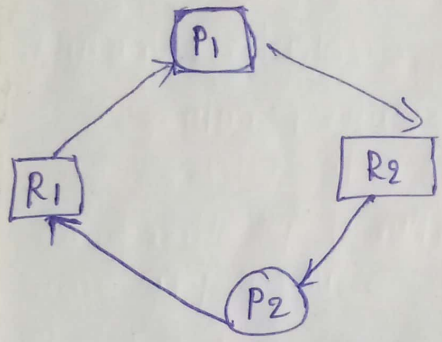
This 1000 Passed to above Target Pointer

it is perform any modification. It is reflected on above code also.

rv = false

Deadlocks

* The main task of operating system is it allocates resources to the processes
* we have several type of resources such as printer, scanner, harddisk, Ram, cache likewise several type of resources are available
* operating system allocates resources among available processes in the system
* lets take an example, we have 2 processes P_1, P_2, 2 resources in the system such as R_1, R_2
* assume P_1 requires R_1 and R_2 for it's completion
* Process P_2 also requires R_1 and R_2 for it's execution/completion



process represents circle ○
Resources represents square □

* R_1 is free, so its OS allocate R_1 to P_1
* Here we are executing P_1 and P_2 simultaneously.
* P_2 is sent a request to R_2
* R_2 is free, so OS allocate R_2 to P_2
* P_1 acquire R_1 resources and P_2 has acquire Resource R_2.
* P_1 Put a Request to R_2. but R_2 is allocate some others. not free
* so, OS not allocate R_2 to P_1
* let as assume P_2 put a Request to R_1. but R_1 is allocate to some other processes. not free
* so, OS not allocate R_1 to P_2.

* P_1 has ac
is acquir
* P_2 can not
not releas
* P_2 has
R_1 is a
* P_1 can
completed
* P_1 ca
* because
* process

→ This
we can't e
call this

definition

a set of
processes
additiona
if we take
* P1 is
by P2
for it's
* as sa
R1. bu
P_1 requir
afte ge

* P_1 has acquired R_1 and P_1 is waiting for R_2. but R_2 is acquired by P_2

* P_2 can not releases R_2 only when it's completion. P_2 can not releases R_2

* P_2 has acquired R_2 and P_2 is waiting for R_1. but R_1 is acquired by P_1

* P_1 can not releases R_1 until and unless it does completed it's execution.

* P_1 can not releases R_1 as well as P_2 can not Releases R_2

* because in execution process P_1 requires both resources R_1 and R_2

* process P_2 requires both resources R_1 and R_2

→ This situation is called deadlock. because in this situation we can't execute P_1 Process as well as P_2 Process so, we can call this situation called deadlock.

definition of deadlock

a set of processes are said to be in deadlock. if every processes in holding some resources and is waiting for additional resources which are acquired by another process

if we take the example

* P1 is holding R_1 and it is waiting for R_2. but R_2 is acquired by P_2. P_2 can't releases R_2 because P_2 requires R_1 also for it's execution. after getting R_1 only P_2 releases R_2.

* as same as P_2. P_2 is holding R_2 and it is waiting for R_1. but R_1 is acquired by P_1. P_1 can't releases R_1 because P_1 requires R_2 also for it's execution.

after getting R_2 then only P_1 releases R_1.

Deadlock characterization

Neccesary conditions for deadlock :-

1. mutual Exclusion
2. Hold and wait
3. No preemption
4. circular wait.

* If these four conditions occur simultaneously in the system then deadlock may occur in the system.
* Here we are not giving 100% chance to deadlock occurs
* These four conditions occurs simultaneously There may be a chance to deadlock may occur.

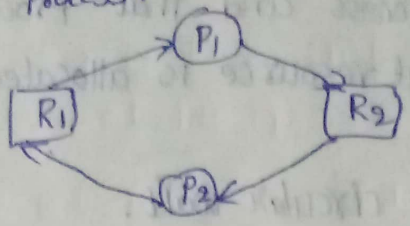
Mutual Exclusion :

mutual Exclusion means a process can uses only one resources at a time that means all the resources are no-sharable by default. by default it is not possible to share any resource among multiple processes.

* if shared a single resource simultaneously among multiple processes it will leads to wrong results.
* assume that we shared a printer b/w 2 process simultaneously Then we will get the printing as the combination of two processes
* it produces P1 and P2 combination output. so that is wrong output
* so, then there mutual exclusion occur. Then there a chance to deadlock may occur.
* if we eliminate mutual exclusion. there is no deadlock in the system.

2. Hold and wait

A process is holding some resources and is wait for additional resources. but those resources are acquired by some other process.



* P_1 is holding R_1 and it is waiting for R_2. but R_2 is acquired by P_2. P_2 releases R_2 only after it's completion

* if we consider about P_2. P_2 holding R_2 and it is waiting for R_1. but R_1 is acquired by P_1. so because of Hold and wait deadlock may occur in the system.

* if there is no hold and wait. there is No deadlock in the system.

3. No premption

A process can not releases a resourceses. untill and unless it has completed it's execution.

* so during the execution a processes can not releases the reasource.

* A process can releases the resources only after completing it's execution.

* P_1 realeases R_1 only P_1 completes its execution

* P_2 can releases R_2 only when P_2 completes it's execution.

* because of No Premption deadlock may occur in the system.

* if we eliminate No premption. it means if the process release the resource during the execution then we can say that there is no deadlock in our system.

circular wait

A set of processes are said to be in deadlock state if one process is waiting for a resource which is allocated to some other process and that process is also waiting for a resource. that resource is allocated to some other process

* This is best example for circular wait.
* because circular wait there have dead occurs.
* if we eliminate circular wait then we can say that there is no deadlock.

Deadlock characterization

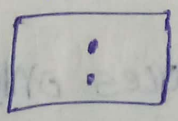
Resource - Allocation graph → it is ntg but a graph with help of this we represent process

we know that graph is represented by G = (V, E) → set vertices, set of Edges

→ Processes and resource types

* let us assume Resource type is RAM. 2 Rams are connected to computer

RAM

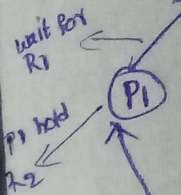


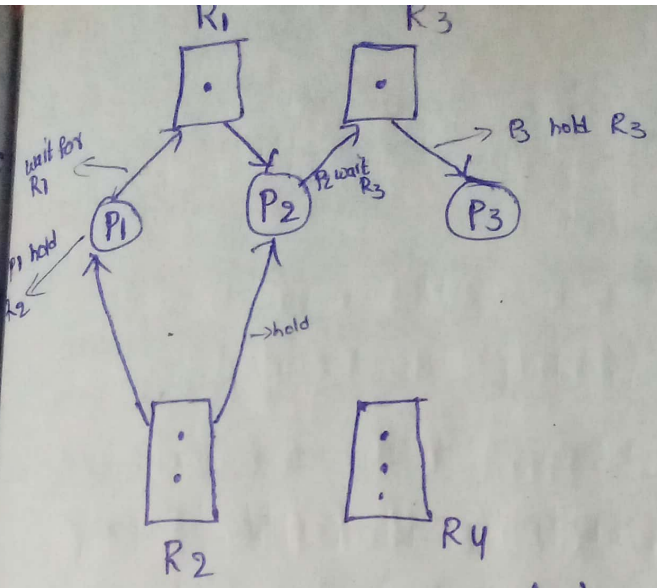
→ Each instance is represented by dot. They are

Edges are 2 types like 5 printers

1. Request Edge $P_i \rightarrow R_j$
2. Assignment Edge $P_i \leftarrow$

* In the below 3 processes in the system and 4 Resource type in the system.





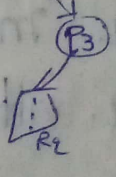
→ check this graph producing any deadlock or not.

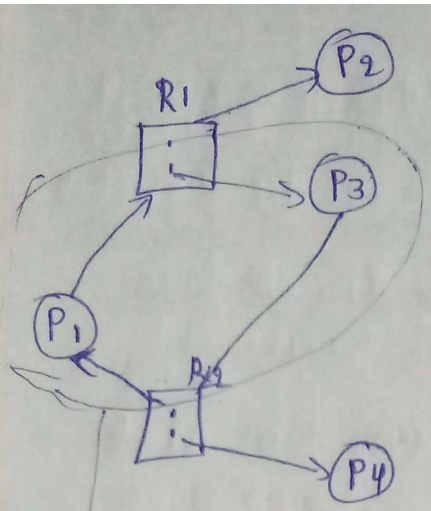
* P_3 requires only one process that is R_3, already R_3 is allocated to P_3. so P_3 can completes execution with R_3.
* once P_3 execution P_3 releases R_3 And that R_2 is allocated to P_2 now
* Now P_2 have R_2 as well as R_3. P_2 completes it's execution with R_2 and R_3. so once P_2 execution is over it releases R_1 and R_2
* so P_1 contains R_1 and R_2. so P_1 completes it's execution with R_1 and R_2.
* Here there is a no deadlock

→ check wheather this graph it have any cycle or not?

if the graph deocsn't contain cycle. it having no deadlock

* if we join P_3 to R_2. it may occur deadlock.
* so that graph contain cycle and deadlock.





* so there is No deadlock
* Resource allocation having cycle. but No deadlock
* if resource allocation graph doesn't contain cycle. it doesn't get deadlock.
* suppose if resource allocation graph contains cycle. and if all the resource type contain single instance then there must be a deadlock.
* Whereas the resource type contain multiple instances then there may be a possibility that deadlock may occur (or) may not occur

Deadlock - Prevention

example for
* Deadbck Prevention means taking deases before the medicine.

1. mutual exclusion
2. Hold and wait
3. No premption
4. circular wait

* It may occur simultaneously it may occur deadlock
* so, we elimnate one by one of this 4 condition. then

we can say that no deadlock occur.

Mutual

* It is Not possible to eliminate mutual exclusion because by default all the resources are Non-sharable
* if we share single resource simultaneously among multiple processes Then we can say that mutual exclution is eliminated
* but that is Not possible to share single resource simultanesly among multiple processes.
* so, it is Not possible to eliminate.

Hold and wait

Hold and waity occurs. it may occur deadlock. so let us check to eliminate Hold and wait are not

* There are 2 approaches inorder to eliminate hold and W

1. A process can request for a resource only when that process is holding No resources.

   * So in this we have only wait is there. No hold are there. so in this way to eliminate Hold and W

2. first the process has to Put request for all the resources

   * let us assum A process requires 5 resources for it's execution.
   * first the correspond process put the request for 5 resources
   * once the process acquired all 5 resources. The only it starts execution
   * Here also No wait here

disadvantage

but the disadvantage of this way are,

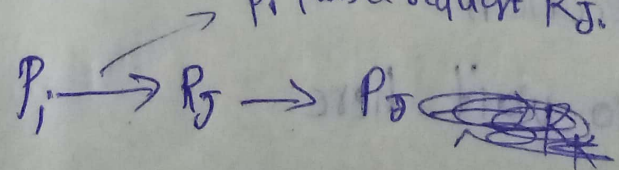
* 1) Poor utilization of resource
  * let us process acquire 5 resources . in 5 resources 1 is printer. but when printer is needed?
  * the last page only the printer is needed. let us assume inorder to execute that process we are taking 1 hour of time
  * during the 1 hour Printer sets as idle.
  * because we use printer ending only.
  * Not utilizing an effective manner.
  * There may be a possibility that some other process May wait for printer that 1 hour time
  * → so That problem is Non as starvation.

2. starvation

* but we can eliminate hold and wait.

No premption

* we check to eliminate premption or not.
* it means a process can not releases untill and unless its execution.



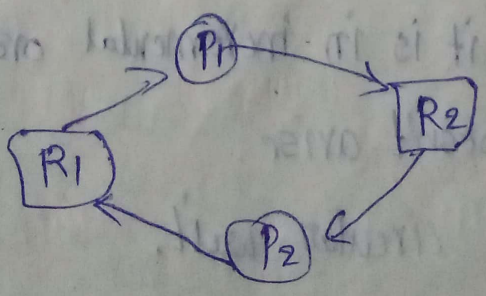
* operating system will check P_i has any acquired resources or not.
* if P_i hold any resources. OS forcebly releases those resources and allocate resources to those processes which needs them.
* after sometim P_i may request R_J.
* This is the 1st solution.
* 2nd $P_i \rightarrow R_J \rightarrow P_J \rightarrow R_K$
* so operating system close that P_J is waiting for R_K
* so OS forcebly releases the resources from P_J and it allocates to P_i

$P_i \rightarrow R_J$, $P_J \rightarrow R_K$

* So P_i completes execution with R_J.
* any any accquired resources of P_i that also completes exe
* And once it's execution is over. -
* Then OS allocate This R_J to P_J. $R_J \rightarrow P_J$
* so in this way we can eliminate No premption.

Circular wait

* Now wheather check to eliminate circular wait or not.



* we can eliminate circular wait.

* inorder to eliminate circular wait OS uses a mapping function called F

$F : R \rightarrow N$ (R: Resource type, N: integer)

* Here an OS represent integer for each resource type.

* let for Printer the value is 1

Printer = = 1

Scanner = = 2 → like that

* Here the process holding resources R_i and the process Request for R_j. only if $F(R_j)$ greater than $F(R_i)$

$$F(R_j) > F(R_i)$$

P_1 → Need Three resources R_3, R_6, R_4

R_3, R_6, R_4

* for first it puts Request for R_3

$F(R_3)$ → it means suppose it returns 3.

$F(R_4) > F(R_3)$ so 4 is greater than 3. Nxt put request

$F(R_6) > F(R_4) > F(R_3)$

P_1

P_6, R_3, R_2 → it is in incremental order

* So, circular wait doesn't arise.

* So, we eliminate circular wait.