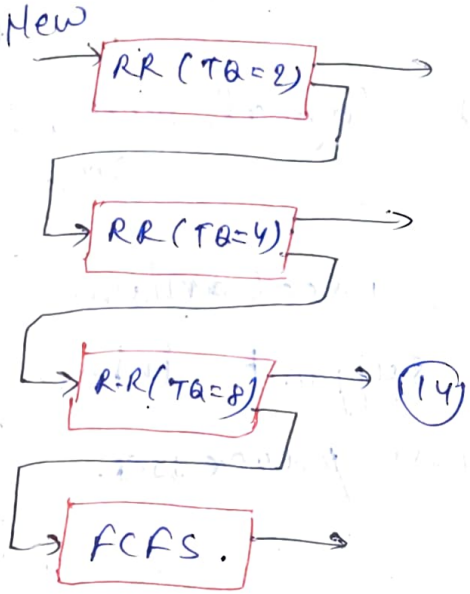
Multilevel feedback Qu

This is with the fe

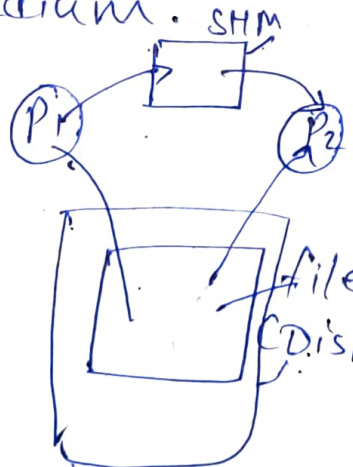


→ starvation ↓ prevent here.

→ But in this p we are providing a chance to get execution.

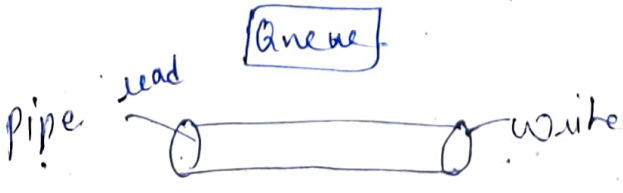
I.P.C & Synchronization ⟹

Process must communicate each other for this the basic requirement is Medium.



→ Generally we don't prefer a file b'coz it required much time for the access. But file is good medium.

→ Apart from local address space of the process we are providing shared memory.



→ Pipe is logical file in main memory. It has two en for read & write. There are in mechanism queue, monitor, file, pipe, etc.

→ Hence these are the diff. IPC mechanism are provided.

→ We are going to deal with logical mechanism.

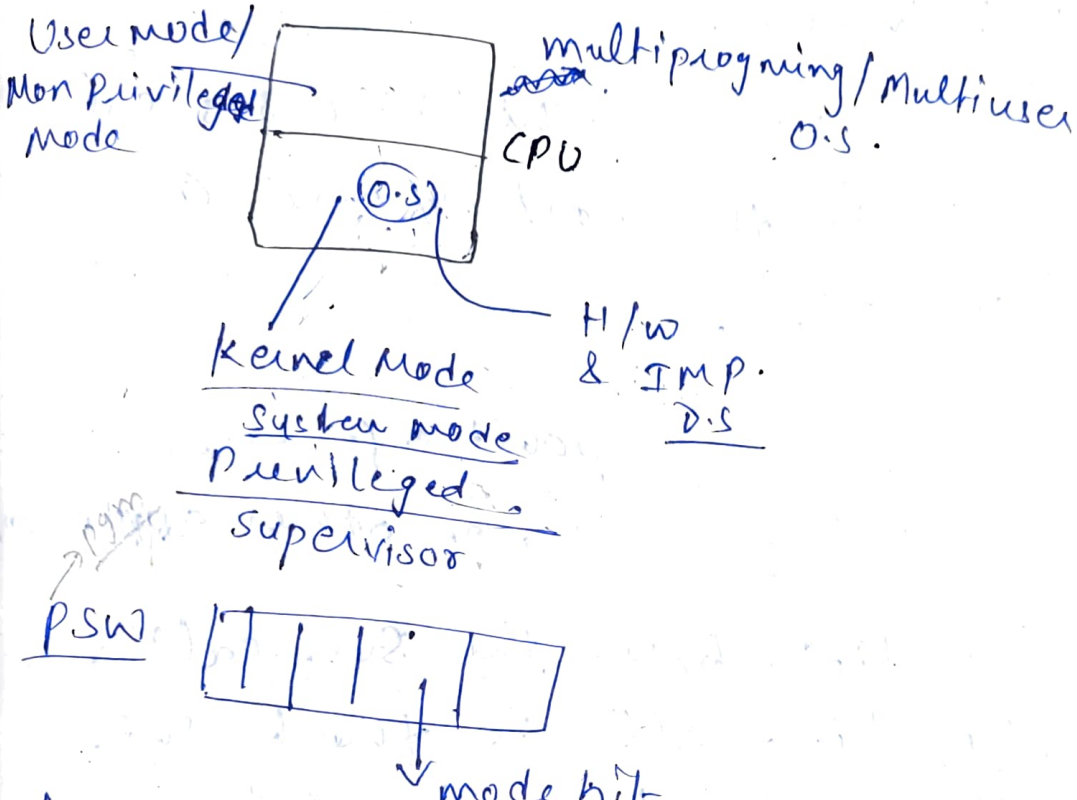
IPC mechanism → Is a mechanism that support inter process comm. b/w the processes.

Synchronizat^n Problems →

When we are communicating, the pro which we are facing in communicat on is known as synchronization problems.

System call Vs Library call

These two both are basically a fun^{cs}.



→ In kernel certain Priveleges will only run & no other pgms will run.

→ The pgms that runs on kernel mode coill give out H/w & implements of O.S. will be given.

→ So certain pgms must be run.

→ The multiuser / multiuser O.S provide the Sys. call Interface which provides the list of all the services.

→ fork() → This is a sys-call. This is the part of the O.S. (kernal).

→ Sqrt();
sin();
Scanf(); } part of the C, or other applica

* The diff. b/w. the Sys. call & the Lib. call is that sys. call run in kernal mode & Lib. call run in user mode

→
```
main()
{ fork();
  printf("Hello");
}
```

When fork(); It gives it prints
Hellow) Hello
"Hello"

two times Hellow
Hello, Hello.

Pgm status word →

PSW has several field and ith mode bit

User mode – 0
Kernal mode – 1 > mode bit

→ In the user pgm if we want some O.S. services then to access the O.S. service is said to be sys call.

→ It is a high level pgm and a call to the operating system for service.

→ It provides a interface.
i.e. API
SCI – List of all the services.

→ The implementation of system call is in operating sys. but; a library call a call to the func which is implemented in user library in user mode.

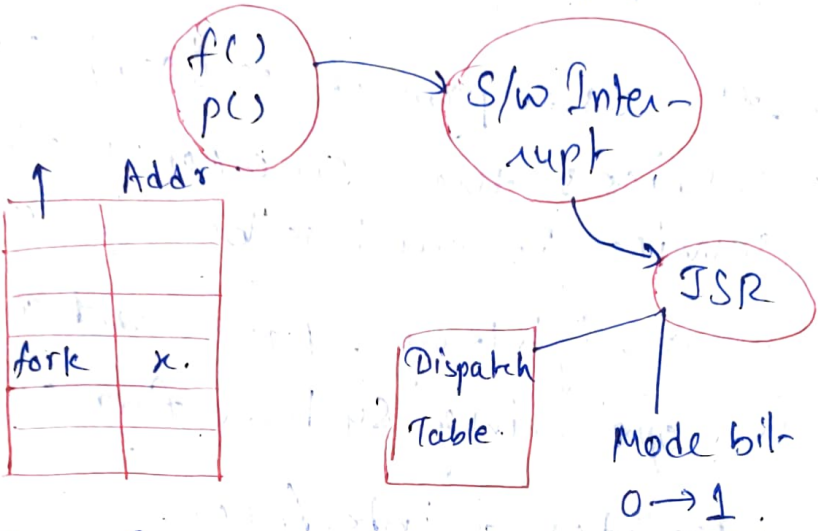
fork(); → O.S. system call

sqrt()
sin()
scanf() → User library (or) user mode library call.

S/W Interrupt → An Interrupt which is generated to execute an instruction is called s/w interrupt.

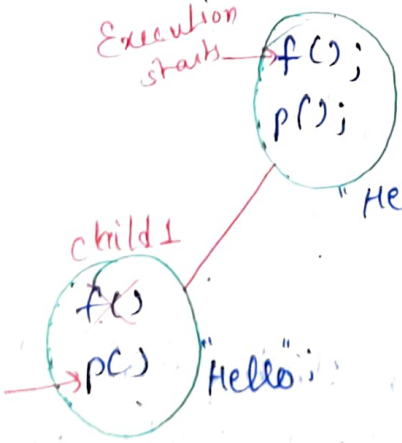
→ Every interrupt has its service interrupt routine (ISR).
It will change mode bit- 0 → 1

→ It consults a table called dispatch table will have no. of entries.



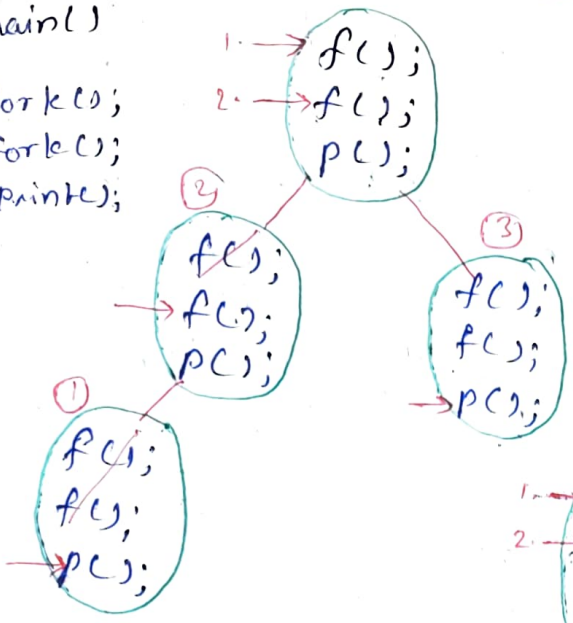
→ When fes, fork function is called it creates child process ch1 another pro The code of the child is same as parent process.

→ In the child process execution sta from instruction next to the fork. But the code is same as parent



→ A single pgm is associate with 2 processes i.e. instance of the pgm.

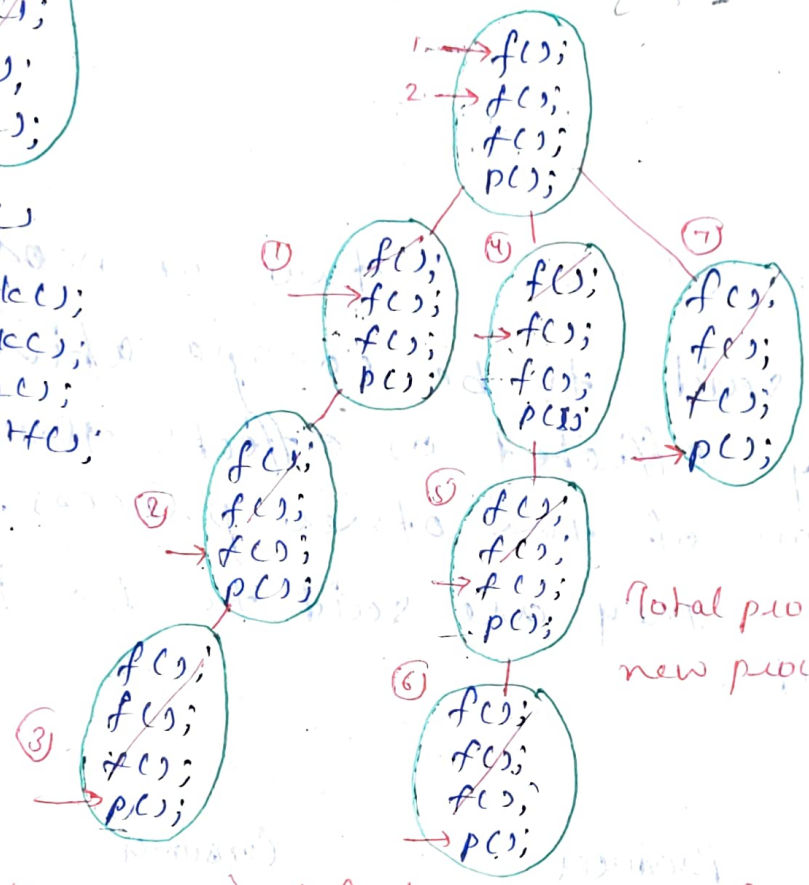
1) main()
{
fork();
fork();
printf();
}



Total processes = 4
new processes = 3

2^n = 4 total
$2^n - 1$ child

2) main()
{
fork();
fork();
fork();
printf();
}



Total processes = 8
new processes = 7

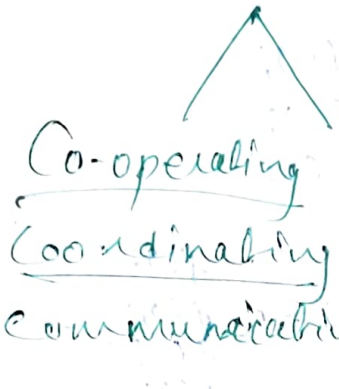
* If there are 'n' forks, then total no. of processes = 2^n.
new no. of processes = $2^n - 1$

Que
main
{ int i,n;
for(i=1; i<=n; ++i) n+1
fork();
}
new process?

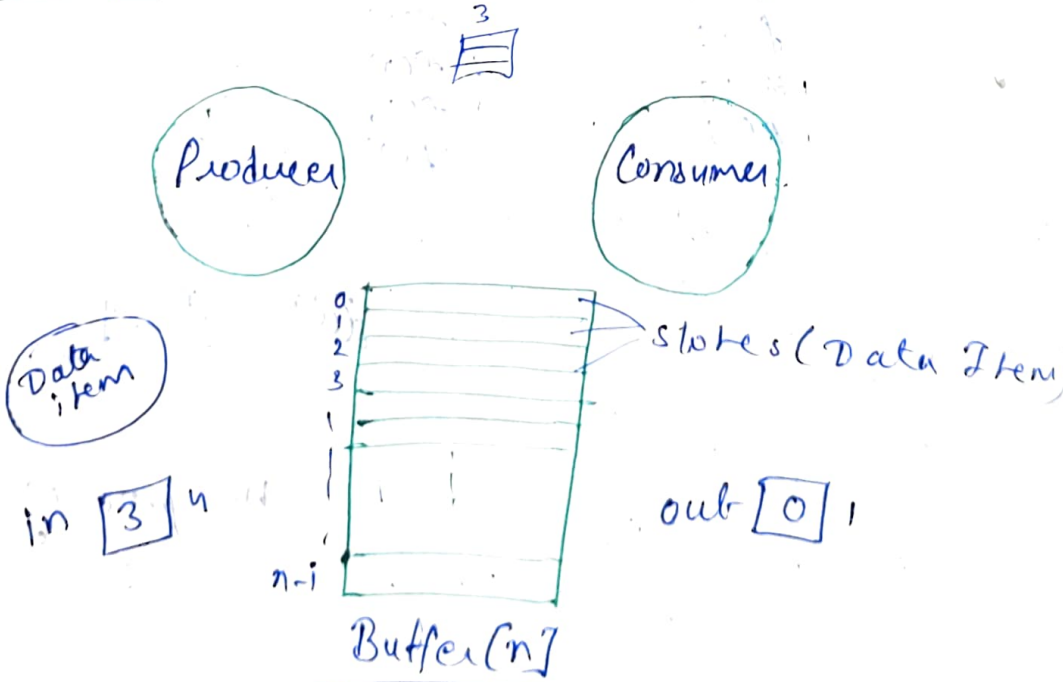
a) n!-1 b) 2n-1 c) $2^n - 1$ d) $n^2 - 1$

Producer Consumer Problem

Processes (IPC & shms).



Co-operating → two or more proce are said to be co-operative if they effected or effects the execution of the other process. Otherwise they are said to be independent.



→ Each slots contains the data items.
→ The Producer & consumer both are processes.

→ If the process are running continuously is said to be in wind server.

UNIX WIND

Daemon Servers.

* producer - produces data item & place it in the buffer in the slot.

→ If slot is free the producer produces the data item.

→ If slot is not free, the producer has to wait until buffer is free.

→ Same prob. with consumer, if there is no data items in the buffer means buffer is empty then the consumer has to wait.

Synchronization points →

1) When buffer is full, the producer has to wait until consumer consume.

2) When buffer is empty, the consumer waits until producer produces data item.

In this producer & consumer are said to co-operating process from the above cond^n.

Producer's code (Pgm).

```
int count;  ⟶ (Global shared variable int count. It will maintain the status of the buffer.
void Producer(void)
{int itemP;
while (True)
{
    Produce-item(itemP);
    while(count == n);  ← Busy waiting
    Buffer[in] = itemP;
    in = (in+1) mod n;
    count = count + 1;
}
}
```

I) Load Rp, M[count];
II) INC Rp
III) store M[count], Rp.

Consumer's code (Pgm)

```
int count;
void consumer(void)
{ int itemc;
  while(True)
  {
    while (count == 0);
    itemc = Buffer[out];
    out = (out + 1) mod n;
```

count = count - 1 ; → I) load Rc, M[count],
Process-item (itemc) ; II) DCR Rc
III) store M[count], Rc

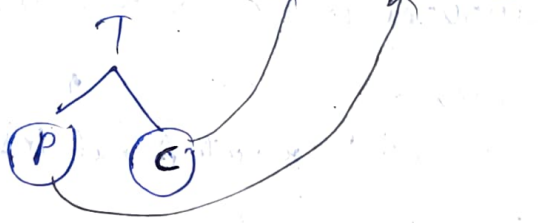
}

}.

eg $n = 100;$

(P) (C)

T. sec. Count 4 5 6



RQ (P) (C)

CPU (priority)

Rp 5 6 Rc 4 5

T_1 : Prod : exec : I

T_2 : Prod : " : II

Interrupt Occur.

T_5 : Consumer : exec : I

T_6 : — " — : " II

Interrupt Occur

: Prod : " : III

: Consu : " : II

* Lack of comm. is arises the Inconsistency prob.

* Lack of Synchronizat^{n} is arises loss of data.

* B'coz of inconsistency we have the Deadlocks.

* Consider the execut^{n} cycle given —

At first producer will execute 2 inst. & interrupt Occu

At 2nd Consumer —— " ——

At 3rd Producer will execute final inst. — " ——

→ The prob. is that we are strictly in the Queue is to be 5, but in the above the queue is maintaining only 4 & 6. This lead to inconsistency.

* Inconsistency producing incorrect results or integrity is lost.

→ ~~This~~ But there is chance to get correct result if interrupt doesn't takes place b/w execution. This term can be said as atomically (i.e. preemption) ie. interrupt occurs only after execut^{n} cycle

→ But by the Von new mann Arch. the interrupt occurs due to many reasons when executing the instructions.

Probs occurs in P & C →

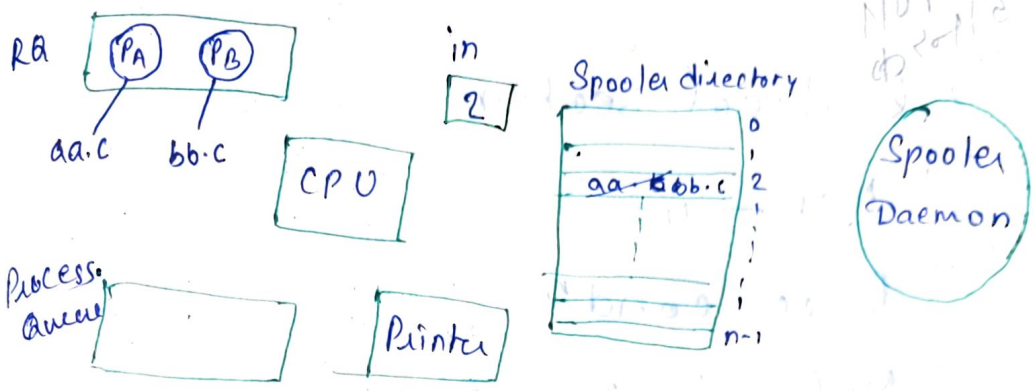
1) Inconsistency 2) Loss of data 3) Deadlocks

Deadlocks occurs ~~when~~ b'coz of improper synchroniz^{n} takes place b/w processes.

Synchroniz^{n} mechanism are developed b'coz of due to the above reasons.

Printer Daemon Problem (UNIX).

→ No user is allowed to access h/w. Even Printer is under ctrl of O.S.



To print the file name into directory —

→ Printer uses spooler directory to make entries by the prints.

→ Spooler Daemon make to printer to print the table into the directory.

load R1M,

T_1 : PA : exe — I
T_2 : PA : exe — II
T_3 : PA : exe — III

When the first three instruction is executed then the file aa.c is stored in the spooler directory. Then the interrupt is occur. Then the next three inst. executed then the bb.c is overwrite the aa.c in the spooler directory then we give the print command then the file bb.c is printed & the aa.c is lost.

T_4 : PB : exe — I
T_5 : PB : exe — II
T_6 : PB : exe — III
T_7 : PB : exe — IV

* The prob. arises is not only inconsistency but also loss of data.

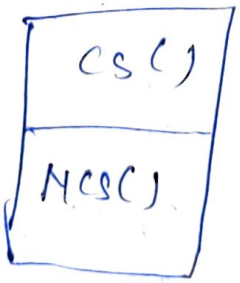
Terminology

1) Critical section
2) Non — " ——
3) Race cond^n (concurrency) (preemption)
4) Mutual Execution

Critical Section → There r sequence of instructn in the CS which has to execute by the process.

Critical sect^n is that part of the pgm text where shared resources are accessed.

Non Critical section → where no shared resources accesses.



Race cond^n → (concurrency)
If two processes are want to access the shared resources.

1) Shared resources
2) Race cond^n.
3) Preemption.

This three arises the prob. of inconsistency, loss of data & deadlock.

Mutual Exclusion → No 2 processes simultaneously present inside the crical sect^n at any time. Means the using the shared resources.

Sol^n for Mutual Sol^n —

- M.E with Busy waiting
- ME without Busy waiting.

H/w S/w.

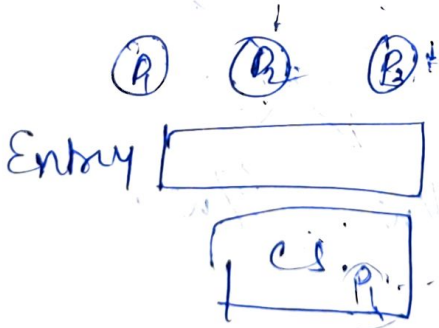
H/w Sol^n →

1) Disabling Interrupts →

* Any soln we develope for the entry must be satisfy the following 4 condns →

(1) M.E. → Under all condn it satisfy the M.E condn.

(2) Progress → No process running the outside the CS should block other interested process from entering the CS.



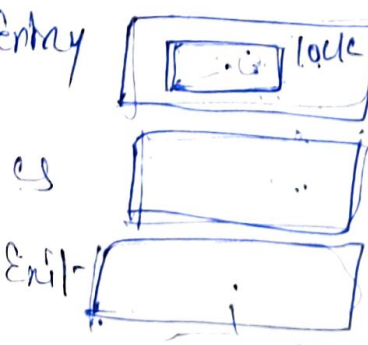
(3) Bounded Waiting → There should be a bound for the timing to access the CS, Means some amt of bound must be allowed to about access critical sect^n.

(4) Arch. Neutral → No assumption about the speed & no. of CPUs may be assumed.

* The Disabling Interrupts has a disadv. that its implementation is diff.. The most of the O.S. can not be able to implement this. So it is non implementable soln in the user level it is possible only in the kernel level. B'coz if it is possible at the user level the user can interact with the H/w & may affect the OS & H/w. & security.

2) Lock Variables

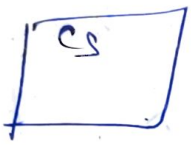
- Binary
  - 0 (cs is free)
  - 1 (cs is busy).



1) When the process enter it check whether lock is '0' or '1'.
2) If it is '0' it enter into C.S. make the lock at entry.
3) After utilizing the CS. it comes into exit at the exit the unlock is done & set '0' at the entry.

Code for the soln (in M/c lang)

Entry

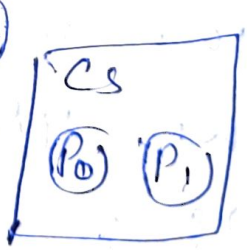
1. Load Ri, M[lock];
2. Comp Ri, #0;
3. JNZ step 1.
4. store M[lock], #1
5. 

6. Exit section.
   store M[lock], #0;

Analysis of code -

Lock [0 1]

R0 [0] [0] R1 (P0) (P1)



T1 : P0 : exec : 1, 2, 3,

P1 : exec : 1, 2, 3, 4, 5,

P0 : Exec : 4, 5.

It is not suitable soln b'coz it don't ensure the M.E.

Hint of Analysis

→ Read, Comp. (optional.)
→ Update action.

If this seq. is given then the given code is incorrect.

If we read the value of shared variable then comp. it. The cmp is optional case. But when we update the value of shared variable after reading then there must be the code the incorrect.

- shared variables (Resources)
- Read
- (Comparisions)
- Updata on the shared variable action.

99.99% Incorrected.

Date 28/10/06

Synchronization Mechanisms

(i) Mutual Exclusion with Busy Waiting → (Spinlock). Wastage of CPU time

a) Disabling Interrupts } H/W soln.
b) Lock Variables }
c) Strict- Alternations → S/W soln

```
P0:
While (true)
{   Non-CS();
    Entry  [ While (turn != 0); ]
           [ CS ]
    Exit   [ turn = 1; ]
}
```

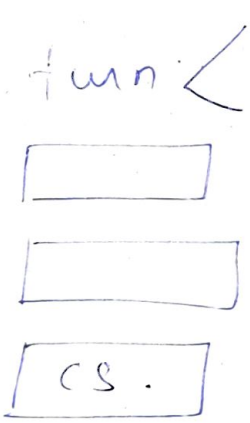
when the turn = 1 the busy waiting & when turn = 0 then CS

```
P1:
While (true)
{   Non-CS();
    Entry  [ While (turn != 1); ]
           [ CS ]
    Exit   [ turn = 0; ]
}
```

when turn = 0 then busy waiting when turn = 1 then CS

* It is a s/w sol^n & comes under the category of Busy waiting & having two processes P_0 & P_1

* the value of turn is tells us that which process really able to enter the CS. When the value of '0' then the process 'P_0' is able to enter the CS. & when the value of turn is '1' then the process 'P_1' is able to enter the CS.



Analysis —

1) M.E is garentees. bcoz the value of the value of the turn is not updated but there is another probs. —

1) Do not garentees Progress bcoz the one process distaict the other to enter the CS.
2) It restricted in 2 processes.
3) It Busy waiting which leads

the prob of CPU wastage CPU time.

d) Petersons Solutions.

- This a s/w soln.
- Restricted to 2 processes.
- Busy waiting soln.

```
# define N 2
# define TRUE 1
# define FALSE 0
int turn;                          // shared variables
int interested[N];                 // (current interest of Process)
void entry-section()
{ 1. int other;                    // other process
  2. other = 1-Process;            // current process
  3. interested[Process] = TRUE;
  4. turn = Process;               // (setting turn value)
  5. while (turn == Process && interested[other] == TRUE)
  }
6. [ CS. ]
```

// If both cond' is true then it go to in the busy waiting.
// When one is true, other take turn if makes that is false so cond' is false so the process can enter in CS.

Simple Case

turn [1]

int [0] = F

int [1] = F

CPU (P1)

CS: P0

T0 : P0 : 1, 2, 3, 4, 5, [6]

P1 : 1, 2, 3, 4, 5

When the process P0 is Busy Waiting starts executing the statements 1, 2, 3, 4, 5, as well as 6 also executed so the P0 in the CS. Then the interrupt is occured. Then the P1 is starts executing then statements 1, 2, 3, 4, & 5 executed. In the (5) the condn is true so busy waiting. so the P0 is only in CS. so ME is

Que Which process will enter CS. guaranteed

a) Whichever process start entry code seq^ first.

b) — " — interest [process]

c) — " — turn.

d) Randomly some processes.

turn

0	1

RQ.

P_0	P_1

int [0] = F T

int [1] = F T

CS
P_1

→ which process is executing first it is upon the CP scheduler.

→ Suppose P_0 start first the 1, 2, 3 statement executi then the interu is occure. * means the va of int [0] = F T.

T_0 : P_0 : 1, 2, 3

P_1 : 1, 2, 3, 4, 5 ↺

P_0 : 4, 5 ↺

P_1 : 5, 6.

P_0 : 5 ↺

The P_1 starts execu 1, 2, 3, 4, 5, now th (5) cond^n is ~~false~~ true so bu waiting. Again ctr is given to the P_0 & it executed 4, & 5 & cond is true so busy waiting

& &

* Which Process sets the value of turn first is enters the critical section is first.

1) Here the mutual exclusion is garantees.

2) In this sol^n the progress is als garanteed. b'coz If any pro

is interested to enter the CS that can enter the CS. B'coz the in the (5) statement we check only for the interested processes. Not for the not interested processes.

) It also garanteed the Bounded waiting B'coz. the Both process has the same priority

) It is also Arch. neutral.

Disadvantage → This soln is restricted only for the two processes. When the 3rd process is there then the 3rd process is not allowed. So the Bounded waiting is restricted to the 2 processes.

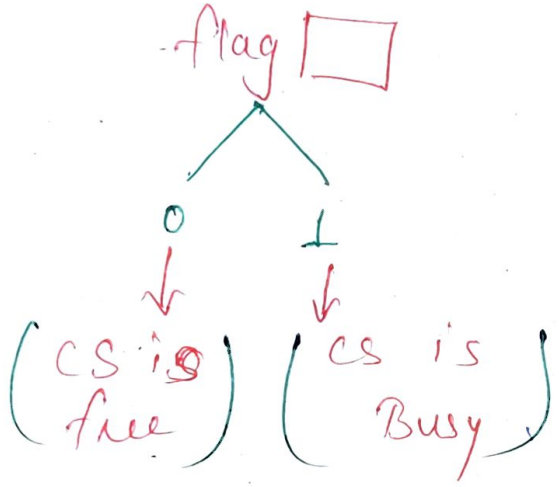
5) Test & set lock Instruction (TSL)

→ H/w soln.

→ Multiple process.

→ Busy waiting.

```
TSL Register, flag;
```



1) Copies the current value of flag into register. (Memory Read)

2) sets the value of flag always to "one"; (1); (Memory write)

→ TSL will automatically locks when it begin.

Entry Section

I. TSL Reg; , M[flag] ;

II. Cmp Reg; , # ∅

III. JNZ step I

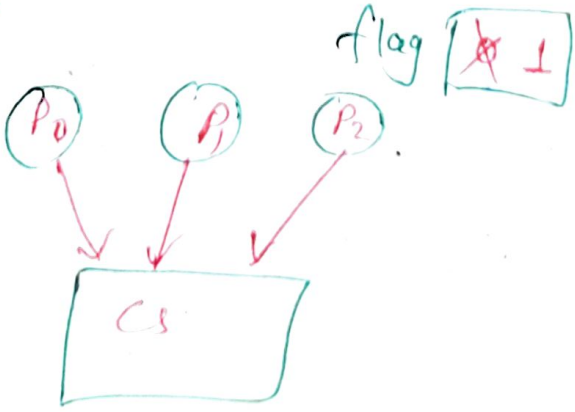
IV. [CS]

Jump not-equal to zero.

V Exit-Section

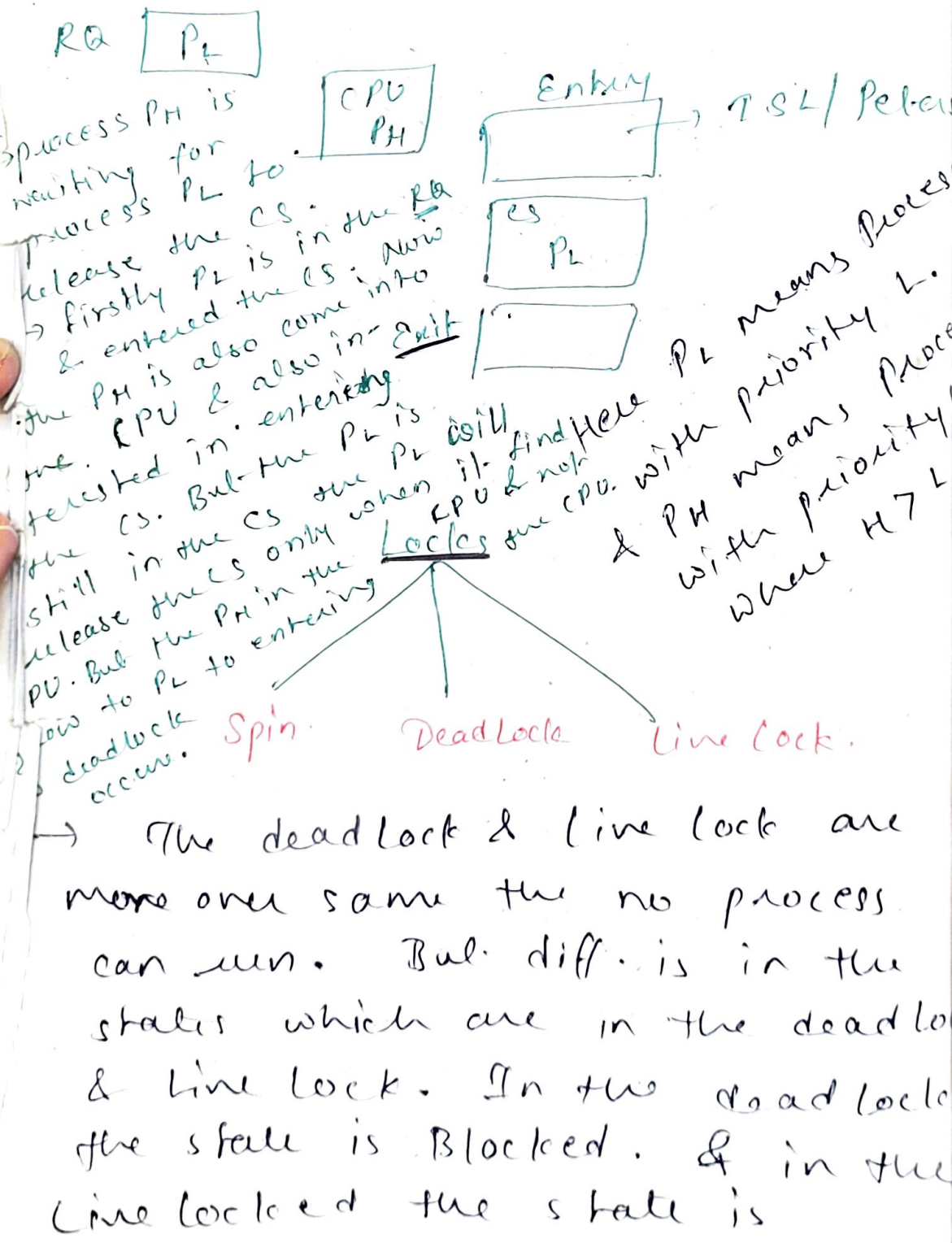
MOVE ME[flag], # ∅ ;

It garanteed M.E.. It garanteed the progress & bounded waiting & Arch. Natral. But this has the disadvantage of Busy waiting Which process execute true(I) statement TSL Reg; , M[flag]; true process is enter the CS. & other may perform the busy waiting.



Priority - Inversion Problem

- Peterson / TSL (B.W.)
- Preemptive Priority based scheduling.



Process P_H is waiting for process P_L to release the CS. → Firstly P_L is in the RQ & entered the CS. Now the P_H is also come into the CPU & also in- terested in entering the CS. But the P_L is still in the CS the P_L will release the CS only when it find the CPU. But the P_H in the CPU & not flow to P_L to entering deadlock occurs.

P_L means Process with priority L. & P_H means Process with priority where H > L

→ The deadlock & live lock are more over same the no process can run. But diff. is in the states which are in the deadlock & Live lock. In the deadlock the state is Blocked. & in the Live locked the state is Ready / Running.

I) Mutual Exclusion without Busy waiting / Blocking →

In this the problem of Busy waiting is solved. In this the process when execute the one iteration it finds that the ~~b~~ busy waiting so it leaves the CPU & gives the chance to the other processes. & suppose that when the others process complete the execution in the C.S. it gives the ctr to me.

(i) sleep() & wakeup();

Code of Producer.

```
Void Producer (void)
{ int itemp;
  while (True)
  {
    Produce_item(itemp);
    if (count == n) sleep();
    Buffer[in] = itemp;
    in = (in + 1) mod n;
```

```
    count = count + 1;
    if (count == 1) wakeup(consumer);
  }
}
```

Code of Consumer

```
void consumer(void)
{ int itemc;
  while(true)
  { if (count == 0) sleep();
    itemc = Buffer[out];
    out = (out+1) mod n;
    count = count - 1;
    if (cont == n-1) wakeup(Producer)
    Process-item(itemc);
  }
}
```

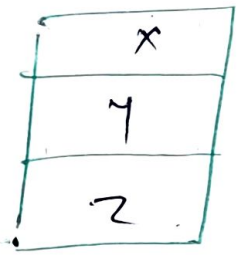
Preemption

This has some probs —

1) Inconsistency — B'coz there is a shared variable count.

2) Deadlock —

n=3 count

(P) 

(C) x x 3

Consider there r 3 processes. At first consumer starts & find the no. process it makes sleep. Initially no. process are there in buffer. So producer produces the items & the buffer get fulled. When the stmt count == n it makes sleep. And the no wakeup signal has been sent to the consumer. In this situation it going to be in the deadlock.

Means there is no guarantee for the process to be in the deadlock or not.