Multithreading

Unit-3

Topics:-

- Multithreading, purpose, multithreading vs multi processing
- life cycle of thread ***
- Creating threads, thread synchronization ***.
- Thread priority, thread sleep,
- isAlive, join()
- inter-thread communication - wait(), notify(), ***
- Producer & Consumer problem ***
- Deadlock.

Q) What is a thread & Multithreading?

A)
- A thread is a part of same program that can run concurrently/parallely
- A thread has separate path of execution.
- Multithreading is a feature of java that allows concurrent execution of 2 or more threads.

Purpose / benefits of M/threading :-

1) To run ≥ 2 or more tasks.

eg A text editor has separate threads for print & typing.

- These 2 tasks can be done parallely

2) To achieve better efficiency using parallel processing.

eg Search in a large dataset can be divided into 2 parts - searching can be done parallely in both parts to improve efficiency.

Q How to create threads?

A 2 ways - by extending **Thread class**
or " implementing **Runnable interface.**

Q Thread vs Process?

Thread	Process
• Threads r part of same pgm.	• Process r separate pgm.
eg Text editor has 2 threads ↙ for print ↘ for typing	• eg text editor, browser, media player r diff processes.
• Threads r light weight	• Processes r larger pgm.

• Threads have same shared m/m	• Process have diff. m/m.
• Context switching takes less time, so m/th. is faster	• takes more time, so multi process is slower.

Q) What r ways to achieve Multi tasking?

A) 2 ways :- multiprocessing & multithreading (m/th)

- M/process means several process r run^g at same time parallely.
- eg text editor, browser, player etc.

- M/th. means several threads r run^g at same time like text editor has mul threads.

M/th.	M/processing
• ~~Each~~ Mul threads r there	• Mul processes r there.
• th. r light weight.	• Heavy wt.
• cost is less	• high
• th. ~~have~~ share same m/m	• Separate m/m.
• Context switching faster	• slower.

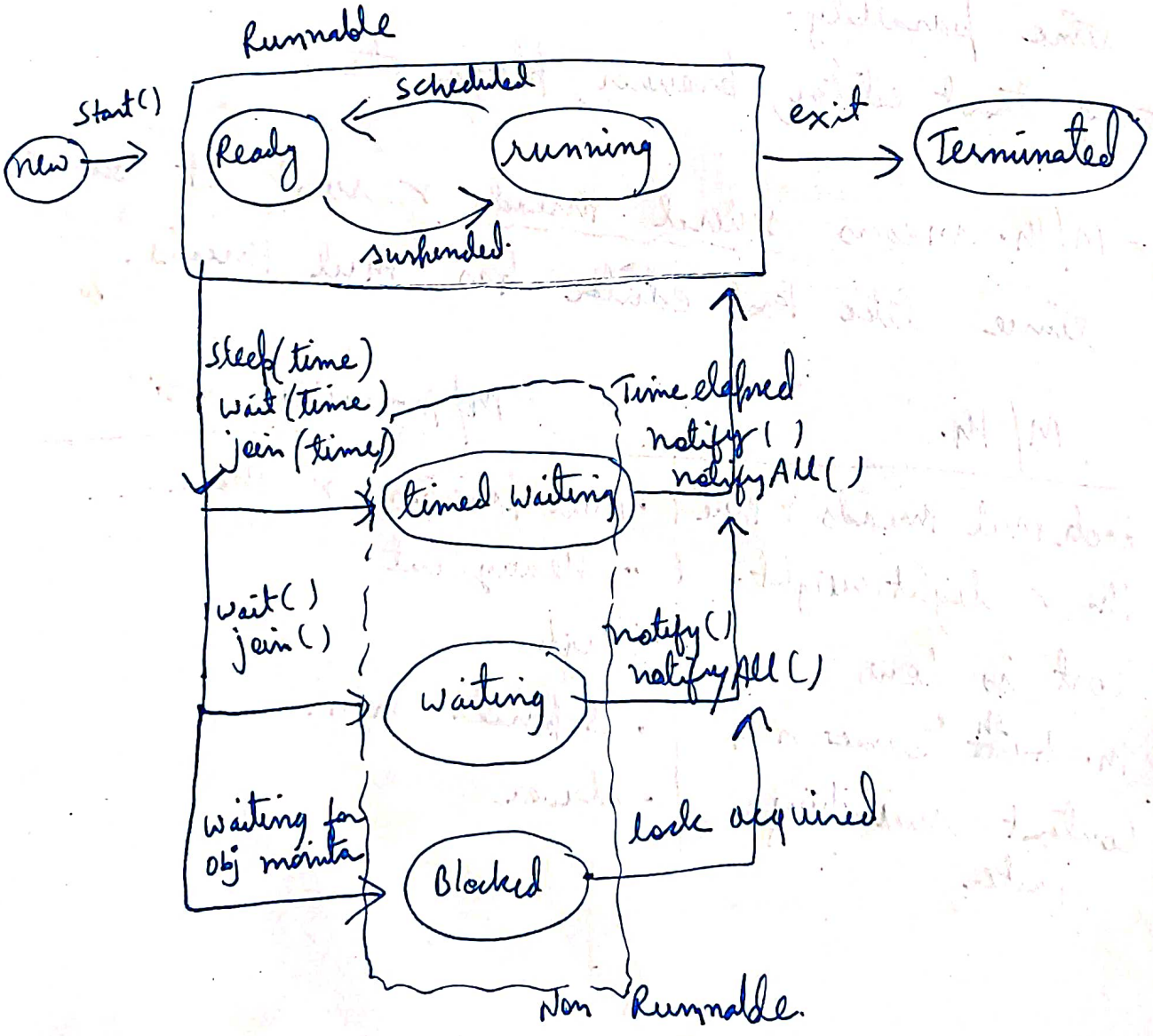
Adv. of java M/th :-

1) Don't block user

2) Several tasks can be done parallely

3) independent - if ~~ex~~ Exception occurs in 1 thread others r not affected.

Life Cycle of a thread ****.



1) New :-
- When a new th. is created it remains new state until started()
- th. is started by calling start() fun.

2) ~~Runnable~~ : Ready :
- when a th. is started, it moves to ready state.
- th. is waiting for CPU allocation wh. is done by OS
- A running th. may also move to ready state when it is suspended by OS

3) Running :-
- When th. gets CPU allocation by the OS, it is moved to running state.

4) timed waiting :-
- th. moves to timed waiting state when sleep(), wait() or join() is called.
- It is now waiting for other th. to complete or for times to elapse.
- th. continues after it receives signal from other th. using notify() or when timeout occurs.

5) Waiting()

- Similar to timed waiting, but there is no timeout.
- th. is waiting for signal from other th. to continue
- th. goes to waiting state when join(), wait() is called.
- th. continues when notify() is called by other th.

6) Blocked

- th. is waiting for a lock wh. is held by some other th.
- when the other th. releases the lock, this th. acquires that lock & continues.

Creating threads

Q WAP to create 2 threads.
Create 2 threads - 1 will print 1 to 100
& other " " a to z

A See code 29a

Thread Synchronization ★★★.

Problem w/ multi/thg :-

1) interference :-

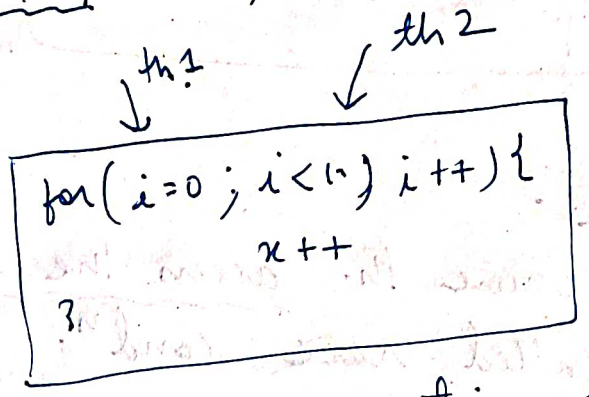
- 2 th. runng at same time can interfere w/ each other.

eg 2 th. writing on same file.

2) Inconsistency :-.

- when 2 th. share same resource then inconsistency may arise

eg



. here 2 th. ~~th~~ r incrementing the value of same var x, wh. is shared b/w 2 th.

. when th. context switching occurs inconsistency may arise.

. expected o/p ~~may~~ should be 20
but actual " may be less than 20.

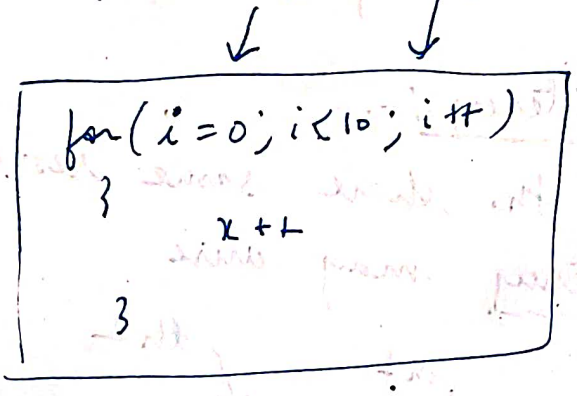
. To resolve these issues, th. synch is needed

Critical section

- A part of pgm wch is accessed by 2 or more threads.

eg the for loop is shared by 2 th. wch r inc the value of same x.

- So this is a c/sec.



Race condition:-

- When 2 or more th. access the same pgm then it is called race condn.

th. synch: **

- Thread synchronization means to allow only 1 th. to enter the critical section (C/S) at a time.
- So only 1 th. is allowed to accen the shared re/src
- this prevents inconsistency.

How is th. syn achieved

- three v 2 ways

1) using synchronized Keyword bf fun

- eg
```
synchronized public void run() {
        for(int i=0; i<10; i++) {
            x++
        }
}
```

- Synchronized kw is ~~put~~ used bf the fun whn. contains c/sec. So only 1 th. enters the fun.

2) using synchronized block

- a syn. block is created & the fun call is placed inside the syn. block.

```
synchronized (task_obj) {
        run(); // run fun call is placed
               inside a syn block.
}
```

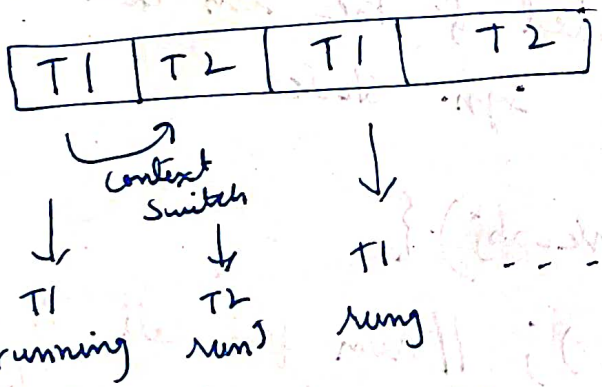
Q) WAP to show th. synch

- In the following 2 th. r inc the value of same var X.
- th. syn is done using Synch. block.
- When no block is used - o/p may be wrong.
- With the block, correct o/p is displayed.

See code - 296

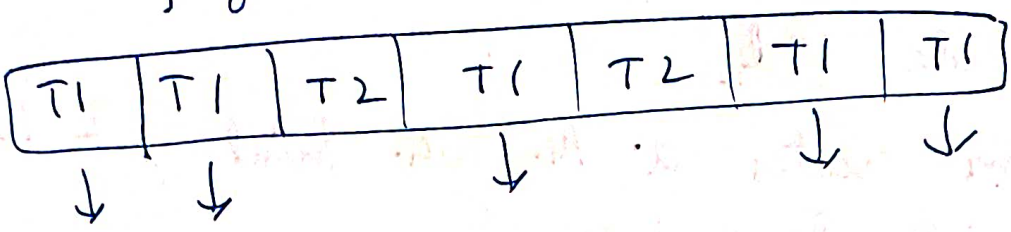
Thread Priority

- When several th. r created, they can't run at same time bcz there is only 1 CPU.
- So they r constantly ~~switching~~ switching - this is called context switching.



- The amount of time a th. ~~see~~ runs is decided by the OS.
- However, we can change th. priorities to allow it run for more time.

eg Priority of T1 is increased



T1 runs for 5 cycles
T2 " " 2 cycles

WAP to demonstrate th. priority :-

- 2 th. r created runs a loop & inc a var
- T1 priority is set higher than T2.
- So value of T1's var will be higher than T2.
- Th. priority can be set by

`Thread.NORM_PRIORITY ± inc`

See code 29C

Thread sleep()

- sleep() method allows to put a th. to sleep for a given duration of time.
- It throws InterruptedException, wh. is a checked Exception, so it must be handled using try, catch block.

WAP to demo sleep()

See code 29d

- The pgm creates a th. which print 1 to 10 at a interval of 1sec.
- To put a th. to sleep for 1 sec

```
Thread.sleep(1000);
```

↳ time in msec.

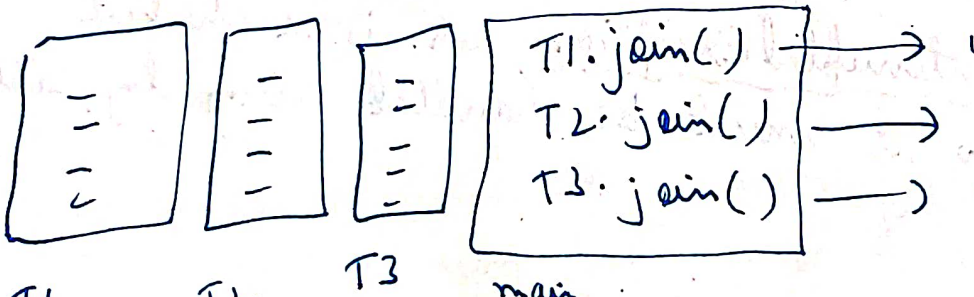
---

isAlive() & join() methods

- isAlive() – used to check whether a th. alive or not.
- join() – used to wait for a th. to finish.

eg Suppose there r 3 th., ~~I want the main thread to finish~~ they can finish in any order. However, I want the main th. to finish at last, then call the join() fun in main th.

- main th. will wait for all the other th. to finish b4 continuing



main will finish at last.

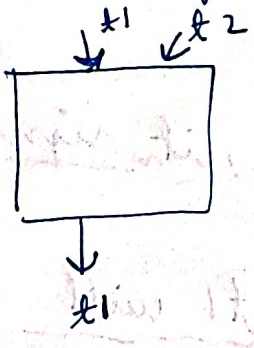
WAP to demo isAlive() & join()

See 29e

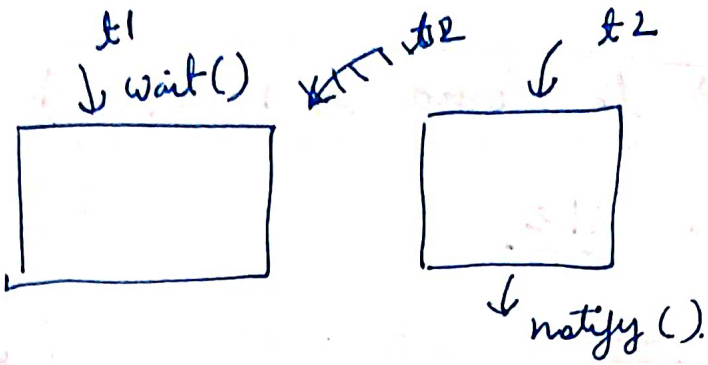
Inter thread Communication ***

- inter th. comm is done using wait(), notify(), notifyAll().
- when th. syn is done using synch block/fun there is no way for i/th. comm.

eg when t1 has leaves the synch block, t2 has no way to know that t1 has exited the syn. block.



- So, i/th. comm means when t1 finishes the syn. block, it informs the t2 th. wh. can enter the block after that.
- i/th. comm is done using wait(), notify(), notifyAll().



Wait()

- when wait() is used, the th. releases the lock & goes to sleep.
- It wakes up when ~~the th t~~ some other th. calls notify()

eg. t1() releases the lock & goes to sleep.
t2() acquires " ", finishes & calls notify
t1() wakes when t2 calls notify.

notify()

- when a th. calls notify(), it signals the other th. to wake up.
- when t2 calls notify(), t1 will wake up.

notifyAll()

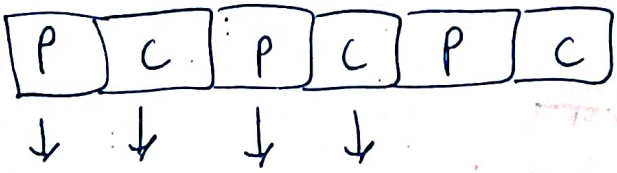
- When a th. calls notifyAll(), it signals all other th. to wake.

- Use of wait(), notify is demonstrated in producer, consumer problem.

Producer Consumer Problem ***

V. imp

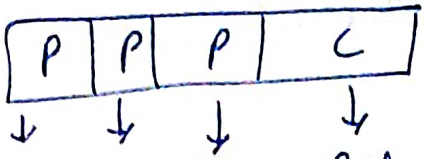
- It's a classic th. sync problem wh. also requires i/th. comm.
- Suppose one th. acts as a producer, it gens some values & assign to a var x.
- Another th. acts as consumer, it reads the value of x & print it.
- Producer P can't assign next value to x until Consumer C has printed it.
- So correct sequence shud be



Set x=1 Print x=1 Set x=2 Pri x=2 .....

- If P runs more than once than consumer will miss some values.

eg:- This seq. is incorrect



Set x=1 x=2 x=3 Print x=3

C didnot print x=1, 2

So this is incorrect.

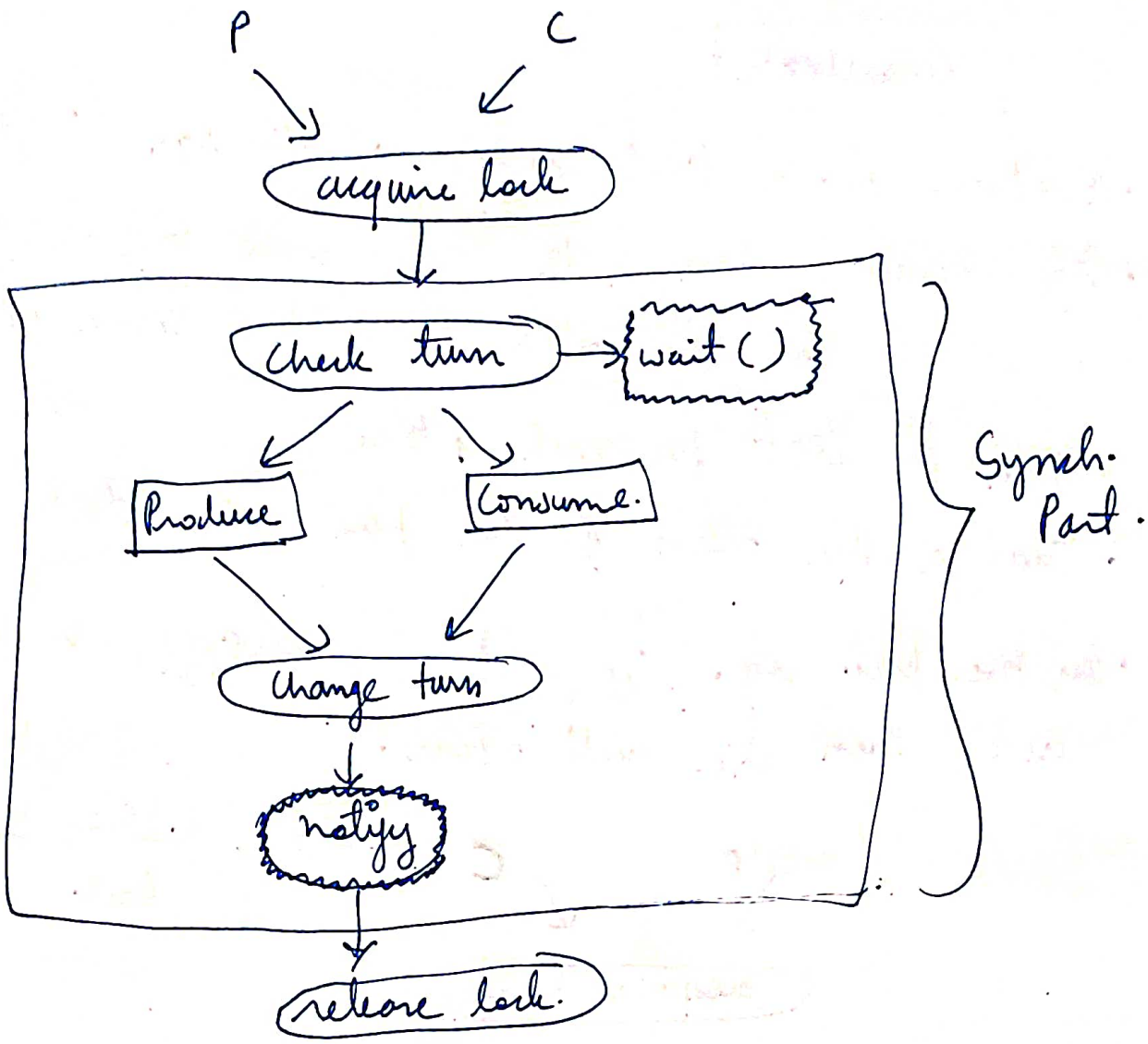
P & C must run 1 by 1.

How to resolve:-

- Use a turn var, whs. specifies whether P will run or C will run.
- If turn = 'p', then P will run
  " " = 'c', " C " "
- During P's turn, C will call wait() & goes to sleep. When P completes, it will call notify() & signal C to wake.

~~Thus P will inform~~

- Thus when its P's turn:-
  - C will sleep
  - P will complete (produce 1 value)
  - P will change turn & wake C
  - so C will run next
- So above steps will be followed by C

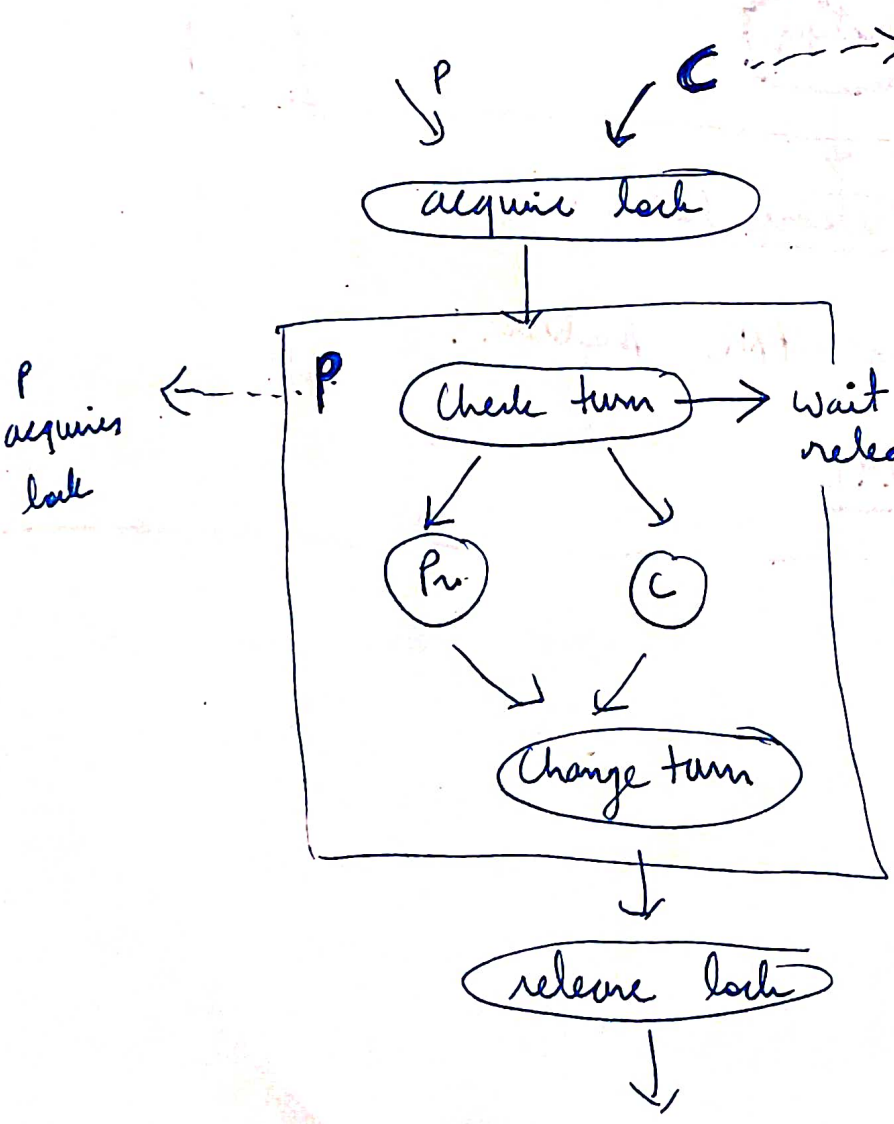


WAP to demo PNC problem :-

see code - 29f

Deadlock

- Another issue that arises w/ th. syn is deadlock.
- D/L occurs when a th. holds some re/src & it is waiting for the re/src held by some other th.
- Both th. wait for each other
- Thus a d/L occurs & the pgm gets stuck.
- In the prev eg., if wait() & notify() r not used, then d/L will occur.



- Suphore P acquires lock & enters the critical section.
- But it was suphared to be C turn.
- Now C is waiting for lock wch. is held by P
- P is waiting for C to run, so that its turn can come.
- Both r waiting for each other, so the pgm will be stuck.
- This is a d/L

Code :-

- Remove wait() & notify from code 2nd.

The o/p will be stuck.