AGENDA

1.) semaphores

2.) Atomic data types

Int / concurrent Hashmaps

3.) deadlock

Producer Consumer Problem:

*) SEMAPHORE :

Mutex with upperbound >1

Semaphore s = new semaphore (2);

At max (2) threads can acquire lock at a time.

HOW TO USE :

Locks:		semaphore	
lock()	→	acquire().	(−1)
unlock()	→	release	(+1)

↳ ON Producer - Consumer side

Producer	Consumer
① s.acquire()	s.acquire()
② s.add()	s.remove()
③ s.release	s.release();

↳ # size of semaphore → ?? (let's discuss)

Case 1.) (NO Producer)

⑥

(store is fully filled)

∴ No Producer

Allowed consumers → (6)

(6)

case 2:)

2 occupied
4 empty

Producers allowed : 4
Consumers allowed : 2

} = 5

total count remains = 5

→ if we apply this condition would it work ??

semaphore s = new semaphore(5);

Another case: (1) semaphore var with (5) value

store

0 Producers
5 semaphore
i.e. No. of consumers → (5)

IS THIS GOOD ??

NO.....

Problem: We're using (1) semaphore to track both Producer / Consumers [2 tasks]

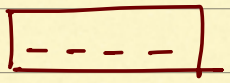
Ideally: 1 semaphore for Producer
1 semaphore for consumer

Eg:

	Producer	Consumer
①	P.acq() · // lock	c · acquire()
②	s.add(item) //	· s.remove(item)
③	c · release(item) //rele·	· p · release();

4 → 3 → 4

∴ we have ② semaphores now, we might need diff LIMITs on them



Semaphore c → Initial value = 0
Semaphore P → Initial value = 4 (Initial state)

S1.) acq. lock in Producer

4 → 3

∴ 3 More Producers are Allowed

(P)→

P update → ③

(unless released)

execute > s.add(item)

should we Release (P) ???

Releasing Ⓟ → values goes back to ④

Ideally → Producer count = 3
Consumer count = 1

Hence : `c.release();`

↓

Instead P.release() → ✗ c.release() ✓

Note: A Thread can be released even if they didn't acquire

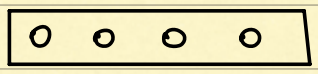
release → basically adds (1)

P_allowed = 4
c_allowed = 0

Producer	Consumer
·) reduce (P)	·) reduce (C) count
·) add item	·) remove item
·) increase (C)	·) increase (P) count

Release += 1
Acquire -= 1

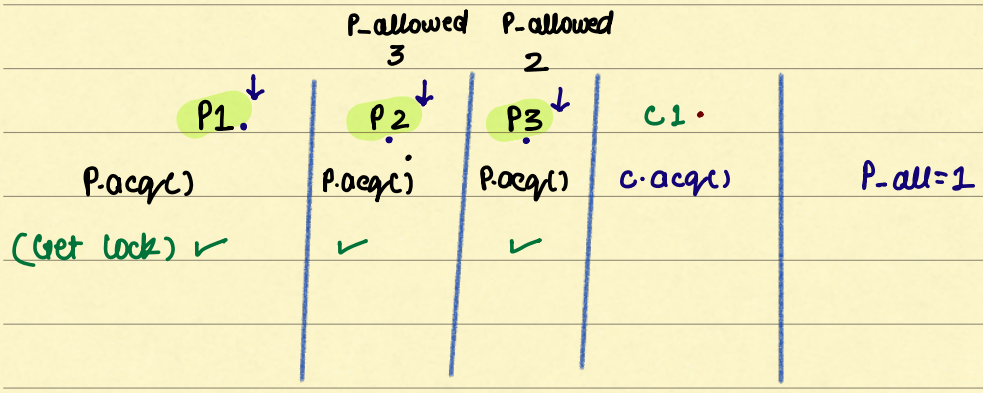
DAY RUN:



Maxsize = 4

c-allowed = 0 semaphore

P_allowed = 4 semaphore



step 1

·) P1, P2, P3 → Get lock.

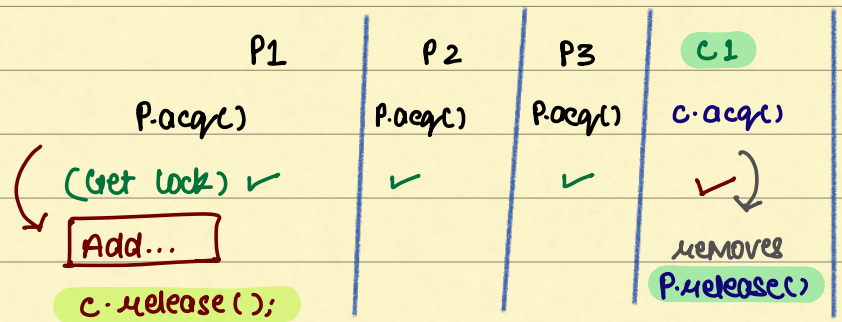
·) They have Not released lock.....

i.e. (C1) still CANNOT Get lock

(WAIT)

---

step 2.)



C-allowed = 1

P-allowed = 1

Now, (C1) can Resume

·) Now, (C1) acquires lock → value again becomes (0)

C-allowed = 0

P- allowed = 1

Step 3.) C4 Now Release (P)

P.release();

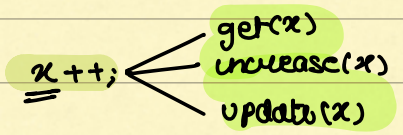
C-allowed = 0

P-allowed = 2 }

DEMO

*) CONCURRENT DATA STRUCTURES:

1.> Atomic data types



·) Multiple threads on objects / data caused issue.

·) The statements were NOT ATOMIC

Atomic operations — execute in (1) GO

·) using Atomic datatypes - No Need to handle locks
(Provide way to Perform operation on NON-atomic variants)

a.) Atomic Int:

Provide extra methods to do common operations on int

DEMO · slower than Normal int

b.) Concurrent HashMaps:

<K,V>

allow datatype to Perform better In concurrent Env.

Eg: Implement cache

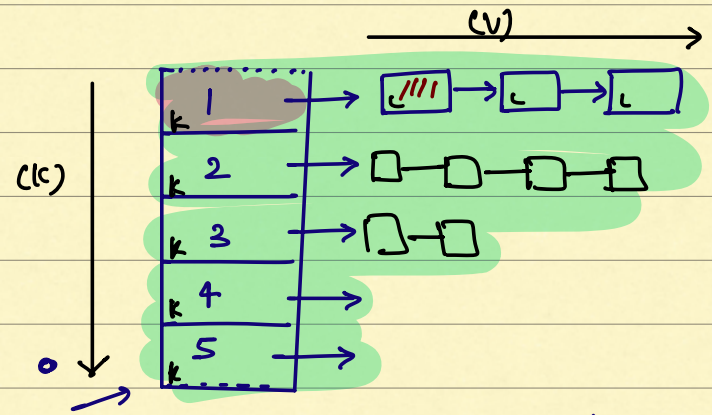
```
class cache {
  Map<Int, String> m;

  add() {
    ....
  }
}
```

NOT WORK IN MUL. ENV.

HOW HASHMAP WORK IN JAVA ...

Internally uses → LL ⊕ Array



get/ put *

- add()
- get()

func.

(t_1) m.get(1)

(t_2) m.get(5)

t_3 m.put(6)

WORKING:

-) Java takes lock on full map
-) Perform operation

PROBLEM

② threads

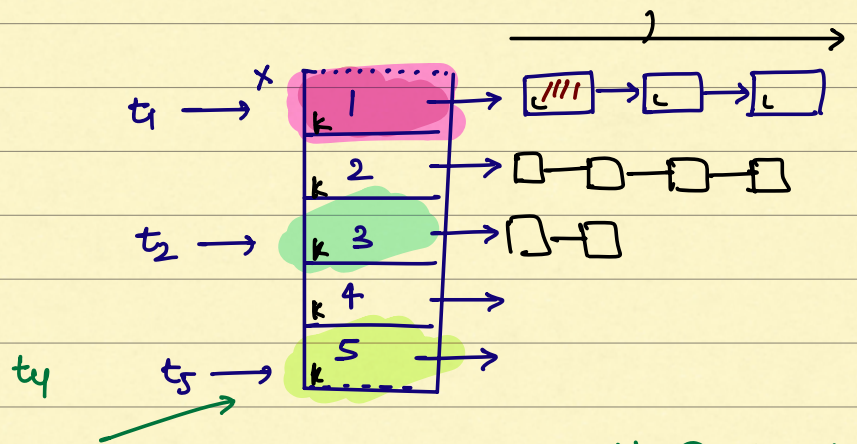
(t_1)	(t_2)
m.get(1)	m.get(4)

ONLY (1) is Allowed even both work on diff buckets.

SOLUTION: CONCURRENT HashMaps

Provide faster Perf. in concurrent env.

:) takes lock on individual bucket



No ② threads are allowed to access same bucket at same time

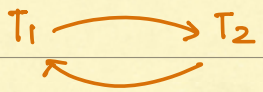
*) **DEADLOCK :**

two threads waiting on each other only.

case: two locks (l_1, l_2)
two threads t_1, t_2

(T_1)	(T_2)
① $lock_1$.lock() ✓	② $lock_2$.lock() ✓
③ $lock_2$.lock() ✗	④ $lock_1$.lock()

both waste time. system won't make progress



only.

deadlock: situation when complete system comes to halt because of threads waiting on locks they will never get.

SOLUTION:

a.) Preventing deadlock / Avoid
b.) Recovery
3.) Ignorance

① Prevention / Avoidance:

take locks in order

(dev. can control when to take lock in system)

Eg: booking seat in BMS

always ascending order

solution → take locks in order

② RECOVERY / IGNORE:

↳ S1.) Deadlock Identification:

1.> wait for timeout

-) call method . wait
-) if NO response
  ↳ assume deadlock
-) kill thread and restart

2.> Graph cyclic Detection

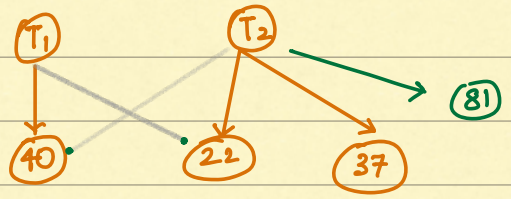
Identify deadlock by building Graph & kill.

T_1 → lock 40

T_2 → 22, 37

T_1 → (Tries) (22) ✗

T_2 → (Tries) (40, 81) ✗



S2.) after identification

-) IGNORE.... (do anything)
-) Try to solve (Eg. in MYSQL)