Classical Problems of Synchronization

1) Bounded buffer Problem / Producer Consumer Problem

- Here we have n buffers, capable of holding 1 item.
- We need a symmetry bet^{n} producer & consumer
  i.e producer produces full buffer for consumer
  or consumer produces empty buffer for producer.
- Producer needs atleast 1 empty buffer to produce
  ↳ no overflow

  & consumer needs atleast 1 full buffer to consume
  ↳ no underflow

Initial Values:-

- Semaphore
  S = 1 ∈ At a time only P or C
  E = n ∈ Initially all buffers empty
  F = 0 ∈ Initially no full buffer

Producer

```
                          while (true)
Produce an                {
  item                  ← produce ()
∴ no of
E, 1 less               ← wait (E)
Only producer
  will work             ← wait (S)
Add item to
  buffer                ← append ()
may allow
  consumer              ← Signal (S)
As 1 item
produced, F 1
  more                  ← Signal (F)
```

Consumer

```
while (true)
{
  consume () → consume an item
  wait (F) → ∴ no of F, 1 less
  Wait (S) → Only C will work
  take () → remove item from buffer
  signal (S) → may allow producer
  Signal (E) → ∴ no of E, 1 more
  use () → finally use item
}
```

∴ When producer work,
E one reduce
F one increase

∴ When consumer works,
E one increase
F one reduces

Now if you remember, wait has a while loop. while (S ≤ 0) & no such checking in signal.
∴ To stop someone from entering CS, you have to impose wait//.

Now if all buffer full ∴ no job for producer needed

∴ E = 0, F = n → 1st line of Producer is wait (E) ⇒ E ⇒ -1
∴ Producer code will be stuck in while loop.
∴ Overflow avoided//

Now if all buffer empty ∴ no job for Consumer needed

∴ E = n, F = 0 → 1st line of consumer code wait (F) ⇒ F = -1
∴ consumer code stuck in while loop
∴ no underflow//

2) Reader Writer Problem

Reader : Only wants to read contents of database

Writer : Update the database ~~& read~~ d
(can both read d write)

* If 2 readers access the shared data simultaneously
→ no problem.

* If a write d some other process want to access the database simultaneously → problem may occur

∴ writer must have exclusive access to shared database while writing.

Rules / Priorities suggested

writer starved ← ○ No reader must wait for other readers to finish simply because a writer is waiting

○ Once writer is ready, it must perform the write asap. (In this case, no new reader will start reading)
↳ reader starved.

Initial values :-

○ ME semaphore for write. ← Semaphore wrt = 1 → common to both R d W.

mutex = 1 → Ensure ME when read-count is updated.

○ Used by first / last Reader that enters / exits CS

int read-count = 0
↳ no. of process currently reading.

○ Not used by readers who enter / exit while other readers in CS

Reader - Writer Lock

Acquiring this lock ⇒ Specifying the mode ⇒ read / write.

- ○ If process wants only to read ⇒ R-W lock in read mode
- ○ If process wants to modify ⇒ R-W lock in write mode
- ○ Multiple process may work concurrently if read mode.
- ○ Only 1 process if write mode

Writer :-

```
wait ( wrt)                    Simple hasslefree
    write operation               writing by
signal (wrt)                   1 writer only
```

Reader :-

```
wait (mutex)            ← 1st R comes, acquire lock
                          ∴ other R can't access read-count
read-count ++           ← No. of R increase
if (read-count == 1)    ← If you are first
                          reader, you got
    wait (wrt)            to stop any writer
signal (mutex)          ← Now he may allow
                          other readers

    reading operation
wait (mutex)            ← Again restrict read-count
read-count --           ← No. of R reduces
if (read-count == 0)    ← If last reader,
                          you got to allow
                          writers.
    signal (wrt)
signal (mutex)          ← Remove restriction
                          on read-count.
```

* Writer in CS :- nobody Else allowed.

Reader in CS :- Other reader allowed but not writer

Now read-count is a shared variable ie used by multiple readers ∴ we need a semaphore to make sure at a time only 1 reader accesses it.

3) Dining Philosopher Problem

- Consider 5 philosopher who spend their lives thinking | eating
- They are seated on a circular table in 5 chairs
- Centre of table has a bowl of rice & 5 chopsticks.
- When a philosopher thinks, he doesn't interact with colleagues.
- When he is hungry, he tries to pick 2 closet chopstick (one on left hand, other on right)
- 1 philosophe may pick only 1 chopstick at a time
  ∴ he can't pick a chopstick i.e in his neighbour's hand.
- When a hungry philosopher has both chopsticks, he eats without releasing the chopsticks.
- When she finishes eating, she puts down both chopsticks & starts thinking again.

Need Deadlock free & Starvation free solution

↓ If each philosopher picks 1 chopstick each, none can eat.

↓ If only few philosopher repeatedly get to eat, rest are starving.

* The easiest solution is to represent each chopstick with a semaphore. ∴ Philosopher takes chopstick using wait() & release using signal().

* Also 5 similar Semaphores needed. (Guess what to use???)

```
Semaphore   chopstick[5];  ← All initialized to 1
  do {
      wait (chopstick [i]);           ← Acquire 1st chopstick
      wait (chopstick [(i+1) % 5]);   ← Acquire 2nd "
           eat ()
      Signal (chopstick [i]);
      Signal (chopstick [(i+1) % 5]);
           think ()
  } while (true);
```

.) No 2 neighbours are eating simultaneously.

.) Here we first pick left chopstick then Right

.) Philosophers first pick left & get preempted, each get 1 chopstick each, however second chopstick is held by neighbour ∴ none can proceed ∴ deadlock ∴ no progress.

To get deadlock free solution :-

1*) Make a separate CS for the 2 line acquiring both chopstick & consider an additional Semaphore for the same ← Expensive.

2*) Allow 4 philosophers only to be seated at a time, Still 5 chopsticks ∴ atleast 1 will get both & eat ← Starvation
↓

3*) get one more chopstick ∴ 6 chopsticks.

4*) ***
Asymmetric solution -

Odd numbered philosopher pick L chopstick & then R chopstick.

Even numbered " " R " then L ".

← Too complex.

* 5*) Allow a philosopher to pick chopstick only if both available

← Get rid of hold & wait
∴ No Deadlock

∴ Each picking 1 chopstick situation no more possible

However Starvation may still occur

But in all these somewhere or the other we change the main problem Statement !!!

Errors in Semaphores :—

Semaphores will be affected if order of wait & signal is violated. by 1 / more processes.

1) Suppose 1 process interchanges the order of wait () & signal ().
Σ multiple P. in CS.

2) Suppose 1 process replaces signal() with wait () ← dealock will occur.

3) Suppose 1 process omits wait () / signal() or both. ← can you imagine what will happen!

∴ we need high level synchronization construct => MONITOR.