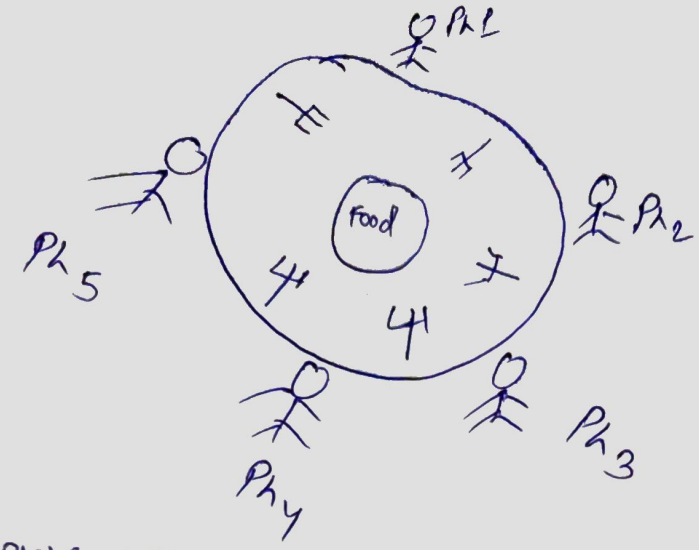
LEC-201 Dining Philosophers problem.



- We have 5 Philosophers
- They spend their life just being in two states:
  (a) Thinking (b) Eating.
- They sit on a circular table surrounded by 5 chairs (1 each), in the centre of table is a bowl of noodles, and the table is laid with 5 single forks.
- Thinking State: When a Ph. Thinks, he does'nt interact with others.
- Eating State: When a Ph. Gets hungry, he tries to pick up the 2 forks adjacent to him (Left & Right). He can pick only one at a time.
- One can't pick up a fork if it is already taken.
- When Ph. has both forks at the same time, he eats without releasing them

Sol^n → ~~Semaphores~~.

→ Semaphores
- Each fork is a binary semaphore
- A Ph calls wait() operation to acquire a fork
- Release fork by calling signal().
- Semaphore fork[5] {1};

– Although the semaphore sol, makes sure that no two neighbours eating simultaneously but it could still create Deadlock.

– Suppose that all 5 Ph. become hungry at the same time & each pick up their left fork, then all fork semaphores would be 0.

– When each Ph. tries to grab his right fork, he will be waiting for ever (Deadlock.

⇒ We must use some methods to avoid deadlock & make sol^n correct.

(A) Allow at most 4 Ph. to be sitting simultaneously.

(B) Allow a ph. to pick up his fork only if both forks are available & do this, he must pick them up in a CS. (atomically).

(C) odd-even rule

an odd Ph picks up first his left fork & then his right fork, whereas even Ph, picks up his right fork & then his left fork.

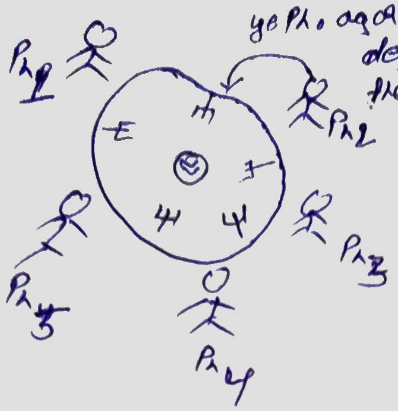
⇒ Only semaphores are not enough to solve this Problem.

We must add some enhancement rules to make deadlock free sol^n.

Sol^n → using Semaphore ~~(binary sema)~~

① each fork – Semaphore (binary sema)

sema fork[5] {1} → initialize.



ye Ph. agar ye fork lega to wait call kr dega
then Ph4 wo fork pick nhi kr skta
b/c Ph1 has took this fork

② wait() → fork[i] ⇒ Ph[i] → acquire

③ Release() → fork[i] → fork releases.

initial
let i = 1
(1+1) % 5
→ 2

Sol^n
```
do {
    wait (fork[i]);
    wait (fork[(i+1)%5]);
    // eat
    signal (fork[i]);
    signal (fork[(i+1)%5]);
    // think
} while (1)
```

Sol^n ✗

this sol^n has deadlock
assume sare ph. ne apne
left wali fork uthali
e.g.- Ph2 wait kr h
ki Ph5 apni ~~right~~
left fork release kare
jo Ph1 ki right fork h

This can be be avoided by

– Allow atmost 4 ph. sim.

~~P_5~~

ab Ph_5 ki absence h to Ph_1 apni right wali fork
utha skta h

- ab Ph4 eating kar rha h
- ab jab Ph4 kha lega tb apne dono fork release kr dega. then Ph2 ko apna right fork mil jaega or Ph2 bhi execute ho jaegi or apne dono fork release kr dega.

& so on Ph3 Ph4 ...
Lekin humne Ph5 ko baithne nhi diya then this solⁿ (X)

(Sol) Allow

Allow a ph. to pick up his only if both forks are available & to do this, he must pick them in a CS (atomically)

i.e.

```
wait(fork[i]);
wait(fork[(i+1)%5]);
```

→ C.S then can make mutex & mutex release

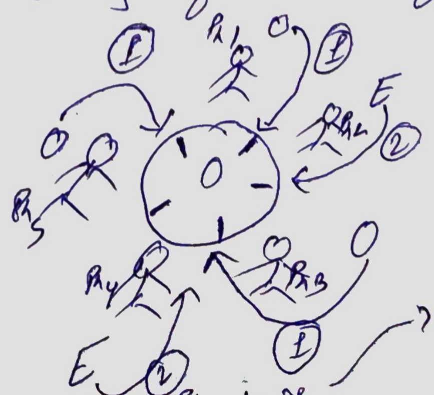
& signal portion code → CS

(✓)

Solⁿ

jitne bhi odd Phi h wo apne left fork ko pick kare & then right fork

& even Ph, picks right fork then left fork.



→ but no deadlock Ph5 apni dusri fork le skta h wo Ph4 bhi wo fork le skta h but condⁿ doesn't fit