Dining Philosopher’s problem

The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively. There is a bowl of rice for each of the philosophers and 5 chopsticks. A philosopher needs both their right and left chopstick to eat. A hungry philosopher may only eat if there are both chopsticks available. Otherwise a philosopher puts down their chopstick and begin thinking again.

The dining philosopher is a classic synchronization problem as it demonstrates a large class of concurrency control problems.

First and most important consequence we face within solving Dining Philosopher’s problem is:

- **Deadlock**

- When do deadlock happens ?

Deadlock happening in dinning philosophers problem when every philosopher takes the right or left fork at same time.

it is considered as deadlock cause all philosophers took only one for and they will wait any of the other’s to drop it , but no one will as they are all hungry.

- How we solved deadlock ?

When a philosopher takes the left fork he will check for the right fork , if the right fork isn’t available , he will drop the left fork.

- Other solution for deadlock in dinning philosophers:

-To make odd numbered philosophers starts with right forks and even numbered philosopher start with left fork , so there will be always at least 1 eats at a time.

-To make maximum four philosophers hungry at a time.

Second consequence we face is :

• Starvation:

- When do starvation happens ?

Starvation happening in dinning philosophers problem when a hungry philosopher takes too much time to eat due to the adjacent philosophers take his chopsticks

- How we solved starvation ?

- by giving a randomize time interval for each state (thinking, hungry, eating) and put a constant maximum boundaries that time intervals can’t exceed.

- Other solution for deadlock in dinning philosophers:

- maintain a queue of philosophers. When a philosopher is hungry, he/she gets put onto the tail of the queue. A philosopher may eat only if he/she is at the head of the queue, and if the chopsticks are free.

- **The pseudocode :**

```
typedef enum { THINKING, HUNGRY, EATING } state;
phil_state state[N];
semaphore mutex =1;
semaphore s[N];
void get_forks(int i) {
state[i] = HUNGRY;
while ( state[i] == HUNGRY ) {
P(mutex);
if ( state[i] == HUNGRY &&
state[LEFT] != EATING &&
state[RIGHT(i)] != EATING ) {
state[i] = EATING;
V(s[i]);}
V(mutex);
P(s[i]); }
}
void put_forks(int i) {
P(mutex);
state[i]= THINKING;
if ( state[LEFT(i)] == HUNGRY ) V(s[LEFT(i)]);
if ( state[RIGHT(i)] == HUNGRY) V(s[RIGHT(i)]);
V(mutex); }
void philosopher(int process) {
while(1) {
think();
get_forks(process);
eat();
put_forks(); }
}
```

• Dining Philosopher’s Problem as real life application

Imagine 5 users working in project on “Github” (table) and each user pulls from the adjacent user, The users are the forks. Admin and the user who pushes to origin (push = philosophers), So the pushing can not only sit at the edge of the table between two users (forks), but also on a line cutting across the table, connecting any two users (forks).