Producer Consumer Problem using Semaphores

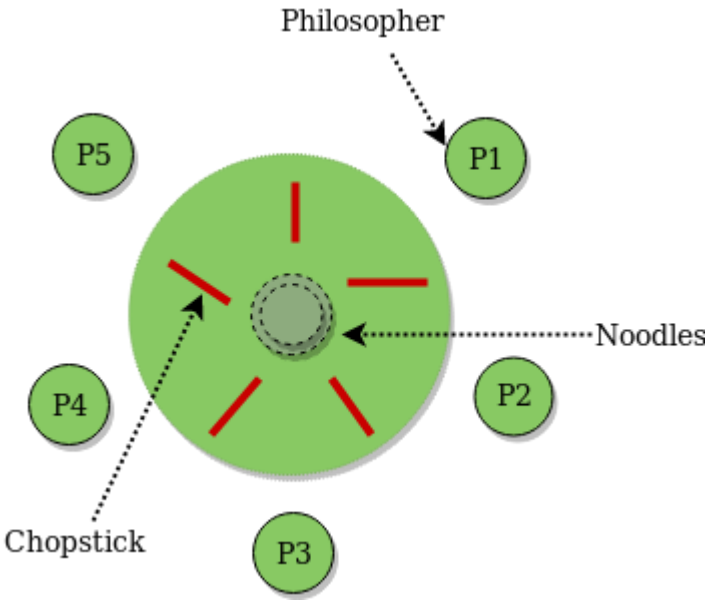
- **Problem Statement –**
    - We have a buffer of fixed size.
    - A producer can produce an item and can place in the buffer.
    - A consumer can pick items and can consume them.
    - We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item.
    - In this problem, buffer is the critical section.
    - To solve this problem, we need two counting semaphores – Full and Empty. "Full" keeps track of number of items in the buffer at any given time and "Empty" keeps track of number of unoccupied slots.
- **Initialization of semaphores –**
  mutex = 1
  Full = 0 // Initially, all slots are empty. Thus full slots are 0
  Empty = n // All slots are empty initially
- **Solution for Producer**

```
do{

//produce an item

wait(empty);
wait(mutex);

//place in buffer

signal(mutex);
signal(full);

}while(true)
```

- **Solution for Consumer**

```
do{

wait(full);
wait(mutex);

// remove item from buffer

signal(mutex);
signal(empty);

// consumes item

}while(true)
```

Dining Philosopher Problem Using Semaphores



- There are three states of the philosopher: **THINKING, HUNGRY, and EATING**.
- Here there are two semaphores: Mutex and a semaphore array for the philosophers.
- Mutex is used such that no two philosophers may access the pickup or putdown at the same time.

- The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

Semaphore Solution to Dining Philosopher

Each philosopher is represented by the following pseudocode:

```
process P[i]
 while true do
   {  THINK;
      PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
      EAT;
      PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
   }
```