Dynning Philosopher Algorithm

Problem Statement :-

1 There are N philosopher sitting arround a round dynning table.
2 There are N plates ; placed Each plate is infront in philosopher.
3. There are N chopsticks placed between the plates.
4 There is a bowl of food placed at the centre of the table
5. whenever a philosopher is willing to have his food, he will be attempting to pick up two chopsticks, which are shared with his nearest neighbours.

If any of his neighbours happens to be eating at that time then the philosopher has to wait until his nei completion.

6. When the philosopher able to pick up two chopsticks he puts food from the centre bowl into his own plate and eats. After he finishes he puts the chopsticks back to table and the chopsticks are available for his neighbours.

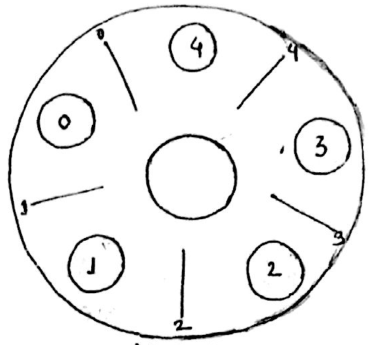


fig:-1

Philosopher	Chopstick sought
0	0
1	2
2	2
3	4
4	4

Statement:-

Let the philosopher be numbered 0, 1, 2, ... (N-1) and let the chopsticks placed at the left of philosopher i be numbered as C_i and at the right be numbered as C_{i+1} % N.

Algorithm to solve the synchronisation problem.

```
Do
 if (i%2 = 0)
  {
   wait (Ci) ;            // Pick up the left chopstick and generate wait signal.
   wait(Ci+1 % N);        // Pick up the right chopstick and generate wait signal
  }
else
{  wait (Ci+1 % N);       // Pick up the right chopstick and generate wait signal.
   wait (Ci) ;            // Pick up the left chopstick and generate wait signal.
 }.
 eat                      // critical Section to the problem.
 Signal (Ci)              // Put left chopstick back
 Signal (Ci+1 % N)        // Put right chopstick back.
```

```
// function "Pick up" will be used when Pi is willing eat
Void Pickup (int i)
{
    A[i] = Waiting ;
    test (i) ;
    if(A[i] ! = Eating)
    {
        c[i] . wait ( ) ;
    }
}
// function 'put down' to replace chopsticks once Pi has done with eating
Void Put down (int i)
{
    A[i] = thinking ;
    if (A[i] == . Waiting )
        test (i) ;
    if (A[i] == eating)
        C[i] . signal ;
}
void initialize ( )
{
    int i ;
    for (i = 0 ; i < N ; i++)
    {
        A[i] = thinking ;
    }
}
}
// for ith philosopher the follow cs - solution will be found
do
{
    dp . {Pick up (i) ;
    <EAT>
    dp . putdown (i) ;
    <Think>
} while (1);
```

{ while(true)

An even numbered philosopher will pick the left Chopstick first then the right one whereas an odd numbered philosopher will pick up the right chopstick first then the left.

According to the fig: 1 and the table;

Now chopstick 0 will definately go to philosopher 0 then none else accure it.

Philosophers Chop stick 2 will go to either philosopher 1 or 2.

Chopstick 4 will go to either philosopher 3 or 4

Therefore chopsticks 1 and 3 are available.

Sleeping Barbar's problem :-

• Problem Statement :-

1) A hair cutting saloon has N chairs in its waiting room and 1 chair in barbar cabin.

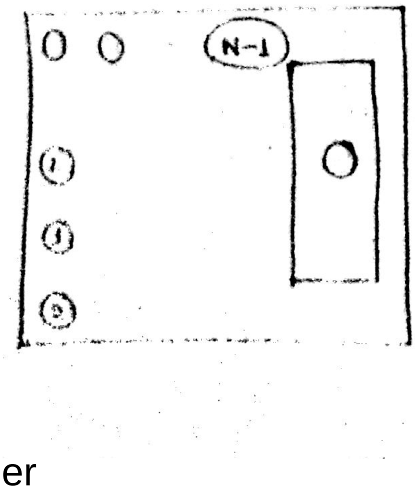
2. If there is no customer in the saloon then the barbar goes to sleep.

3. When a customer arrives at the saloon ~~the customer walks into~~ one of the following may occur —

i) If there is no customer in the saloon then he goes to barbar's cabin wakes up the barbar and starts getting haircut.

ii) If a customer is already getting haircut then the arrived customer looks for an empty chair in the waiting room. If a chair is empty then he occupies it in the order of his arrival and waits for his turn. If there are no empty seats then the customer walks away without getting a haircut.

4. When a customer finishes getting his haircut he sents another customer from the ready queue.



Algorithm for Sleeping barbar's problem:-

```
do
{
  if (capacity_full = TRUE)
  {
    deleted ( );        // If there is no space in the ready queue then n+1th
    or                  customer and so on will be aborted.
    aborted ( );
  }
  else
  {
    capacity = capacity - 1 ; // Whenever capacity_full = FALSE or 0
                              1 ≤ capacity ≤ N then a newly arrived
  }                           customer can be accomodated in the waiting room
  Wait (ci);            // The ith arrived customer, i=0,1,2, ..., n-1 will
  getting haircut       Proceed to barbar's cabin and generate wait signal to others
  Signal (ci);          // once ith customer is completed he will generate
                        signal to others and i+1th customer will get his turn
} while (TRUE)
```

Producer Consumer Problem :-

Problem Statement :-

Consider there are two concurrent processes — A process is producing some item and another one is consuming those items. There may be the following three situations —

i) Producer is producing at a faster rate than the consumer is consuming.

ii) Producer is producing at a lesser/slower rate than the consumer is consuming.

iii) Producer is producing in almost at a same speed at which the consumer is consuming.

In situation 1 There arises a synchronisation problem for which a temporary storage is used by the producer, known as a buffer which can be either bounded (limitted size) or unbounded (unlimitted size)

How to use a bounded buffer to utilize the overwhelming production produced by the producer ~~for~~ for the consumer.

Semaphore used : mutex = 1 ;
Empty = N ;
full = 0 ;

```
Void producer ( )
  {int item
   while(true)
    {
     Produce _ item (& item);
     wait (empty ) ;
     Wait (mutex) ;
     enter_item (item);
     Signal (mutex);
     Signal (full);
    }
  }
Void consumer ( )
{
  while (true )
  {
   wait (full ) ;
   wait (mutex);
   remove (& item) ;
   Signal (mutex) ;
   Signal (empty) ;
  consume _ item ( item) ;
  }
}
```