BITS, PILANI – K. K. BIRLA GOA CAMPUS

Operating Systems

by

Dr. Shubhangi

Synchronization

PROCESS SYNCHRONIZATION

Some Interesting Problems

THE READERS/WRITERS PROBLEM:

```
BINARY_SEMAPHORE   wrt       = 1;
BINARY_SEMAPHORE   mutex     = 1;
int                readcount = 0;
```

```
Writer:
do {
        wait( wrt );
        /*   writing is performed    */
        signal(  wrt  );
} while(TRUE);
```

```
Reader:
do {
        wait( mutex );                          /* Allow 1 reader in entry*/
        readcount = readcount + 1;
        if readcount == 1  then  wait(wrt );    /* 1st reader locks writer */
        signal( mutex );
                /*   reading is performed  */
         wait( mutex );
         readcount = readcount - 1;
         if readcount == 0  then  signal(wrt );   /*last reader frees writer */
         signal( mutex );
} while(TRUE);
```

```
WAIT ( S ):
    while   ( S  <=  0 );
    S  =  S  -  1;
SIGNAL ( S ):
    S  =  S  +  1;
```

PROCESS SYNCHRONIZATION

Some Interesting Problems

THE DINING PHILOSOPHERS PROBLEM:

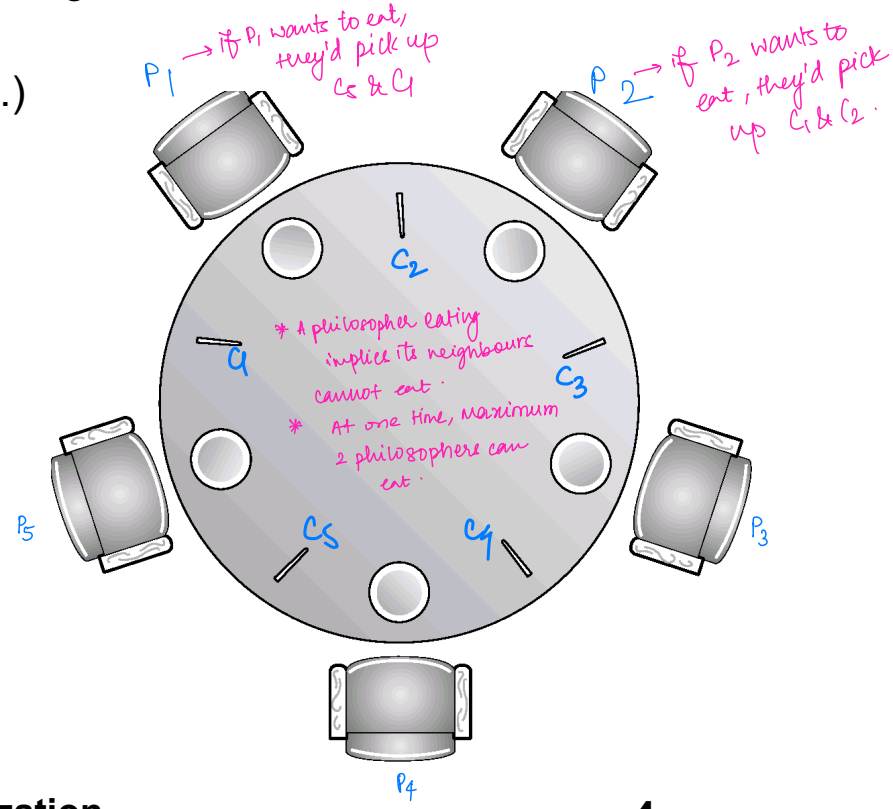
Philosophers → Thinking state / Eating state } exclusively.

5 philosophers with 5 chopsticks sit around a circular table. They each want to eat at random times and must pick up the chopsticks on their right and on their left.

Clearly deadlock is rampant (and starvation possible.)

Several solutions are possible:

- Allow only 4 philosophers to be hungry at a time.
- Allow pickup only if both chopsticks are available. (Done in critical section)
- Odd # philosopher always picks up left chopstick 1st, even # philosopher always picks up right chopstick 1st.



DINING PHILOSOPHER PROBLEM

- Data structure support needed
  - semaphore chopstick [N] ;

- Data structure Initialization
  - for(int i=0; i< N; i + +)
    chopstick [i] = 1;

BINARY SEMAPHORE

↳ At a time, only one philosopher will be accessing this chopstick

DINING PHILOSOPHER PROBLEM

- The structure of Philosopher *i*:

```
do {
        wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );

            // eat

        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] );

            // think

} while (TRUE);
```

can be generalised for n when n = odd.

releases both left and right chopsticks.

Analogy :-

* Philosophers here are processes.
* The chopsticks are the semaphores (common thing that the philosophers want to use and enter the critical section (eating))

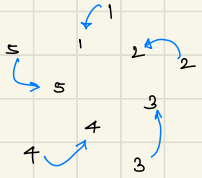
Why discussing equal no of chopsticks and philosophers?

→ The Operating System has limited resources.
(discussing the ideal solution)

```
do {
        wait ( chopstick[i] );
        wait ( chopStick[ (i + 1) % 5] );
            // eat
        signal ( chopstick[i] );
        signal (chopstick[ (i + 1) % 5] )
            // think
} while (TRUE);
```

6: Process Synchronization

Assume that all philosophers pick up their left chopstick first



∴ chopstick[1] = 0
chopstick[2] = 0
chopstick[3] = 0
chopstick[4] = 0
chopstick[5] = 0

Now nobody will be able to proceed further since 1 will make chopstick[2] = -1 when they pick up their right chopstick.

∴ DEADLOCK SITUATION

SOLUTION #1: Allow at max 4 people to pick up any chopsticks at once. so that atleast one philosopher ∴ atleast one philosopher will have the ability to pick up their right chopstick and eventually release their left chopstick for some other process

We can also keep N philosophers and N+1 chopsticks.

SOLUTION #2 :- Pick up a chopstick if only both chopsticks are available otherwise, do not hold even one.

This is not leading to deadlock since this is only waiting, it is not holding from others (holding + waiting → deadlock, waiting will never lead to deadlock)

SOLUTION #3: Write separate codes for even and odd numbered philosophers.

even : picks left chopstick first (Assymmetrical solution)

odd : picks right chopstick first

even:

```
do{
  wait[chopstick[i])
  wait ( chopstick[ (i+1) % N]
    // eat
  signal (chopstick[i])
  signal ( chopstick [ (i+1) % N]
    // think
} while (TRUE)
```

odd:

```
do{
  wait ( chopstick[ (i+1) % N]
  wait[chopstick[i])
    // eat
  signal ( chopstick [ (i+1) % N]
  signal (chopstick[i])
    // think
} while (TRUE)
```

DINING PHILOSOPHER PROBLEM

- Draw back
  - Dead lock is possible → if everyone picks up left/right chopstick
- Solution
  - The philosopher can pick up a chopstick if both the left & right one are free.
  - Let the odd philosopher pick the left chopstick and even philosopher pick the right chopstick.
  - Take only even number of philosophers