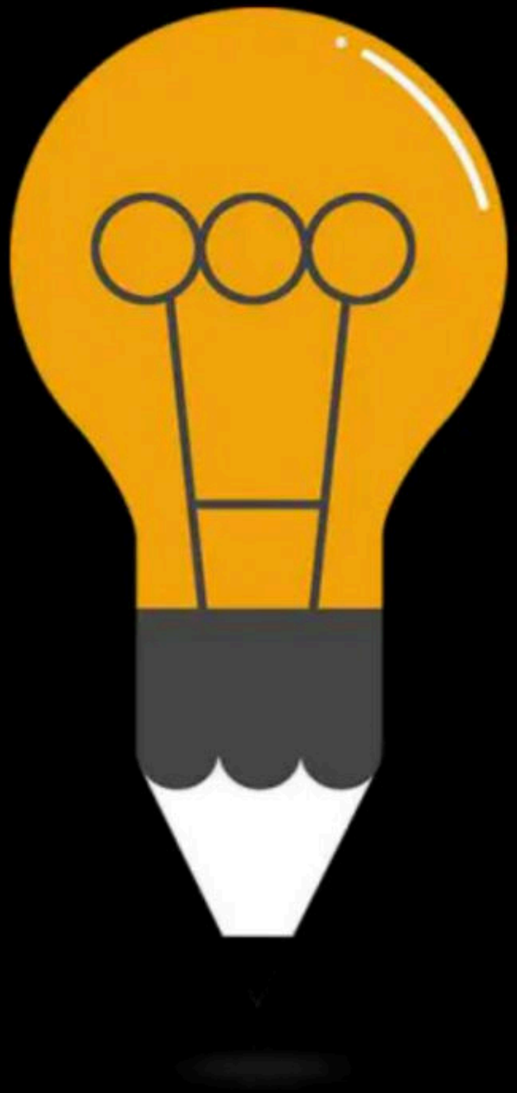
unacademy

Doubt Clearing Session & Classical Synchronization Problems

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 18 • May 9, 2023

unacademy



Operating System

Doubts & Classical Problems of Synchronization

By: Vishvadeep Gothi

while ( ) ;

while (a > b)
a = x;

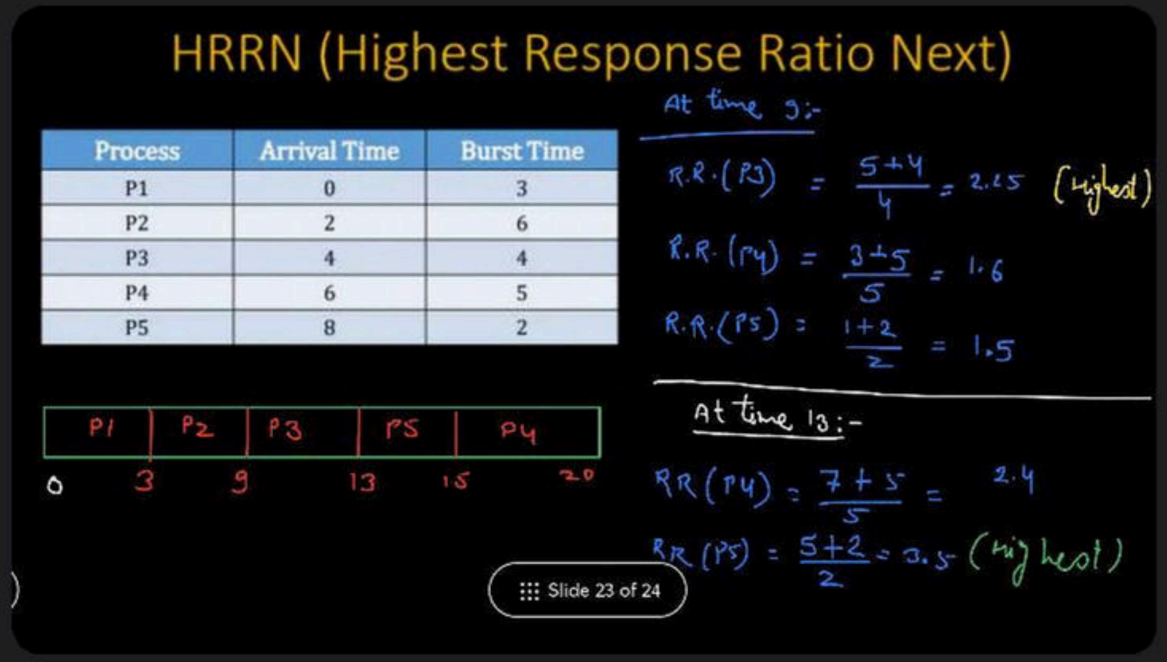
while (_)
{
}

while (——)
{
= = }
}

unacademy

▲ 1 • Asked by Anish

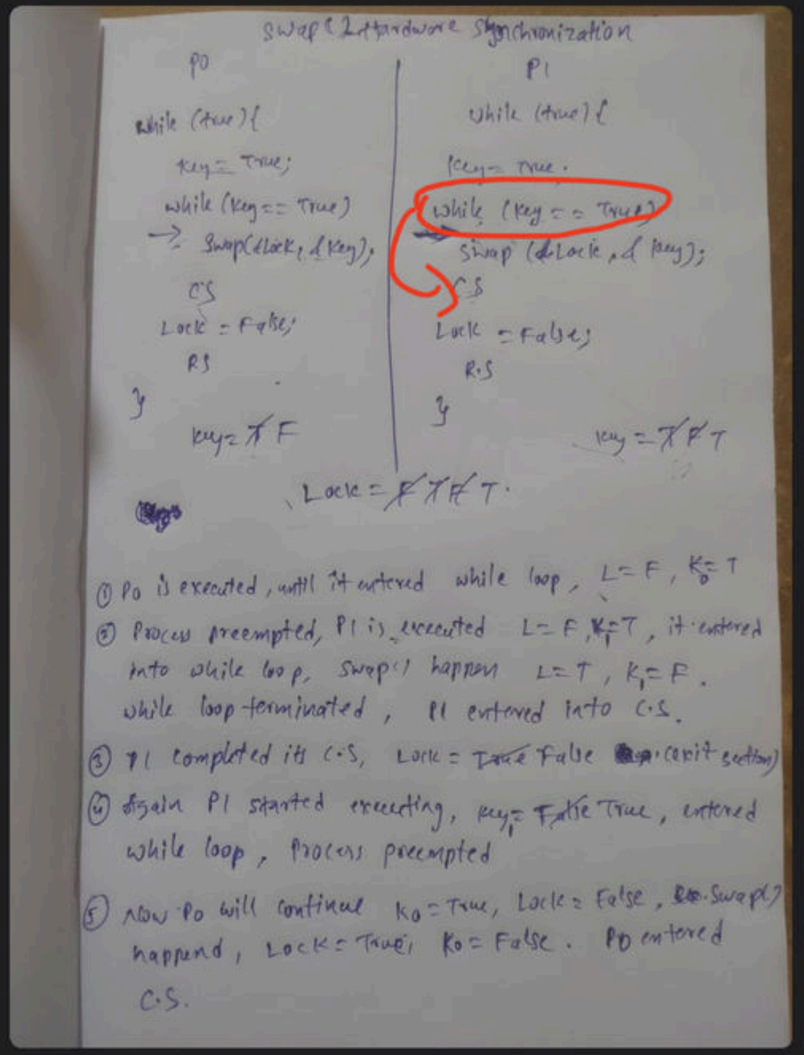
Doubt hai isme?



unacademy

▲ 1 • Asked by Rakesh

swap hardware synchronization....bounded waiting



P0

Lock = ~~False~~ ~~True~~ False ~~True~~

key = ~~True~~ False ~~True~~

P1

key = ~~True~~ ~~False~~ True

P1 in C.S.

unacademy

Quiz - 2

Ques (1)

If P_0 then P_1

B = 7 21 19

If P_1 then P_0

B = 7 8 15

Ans = 2

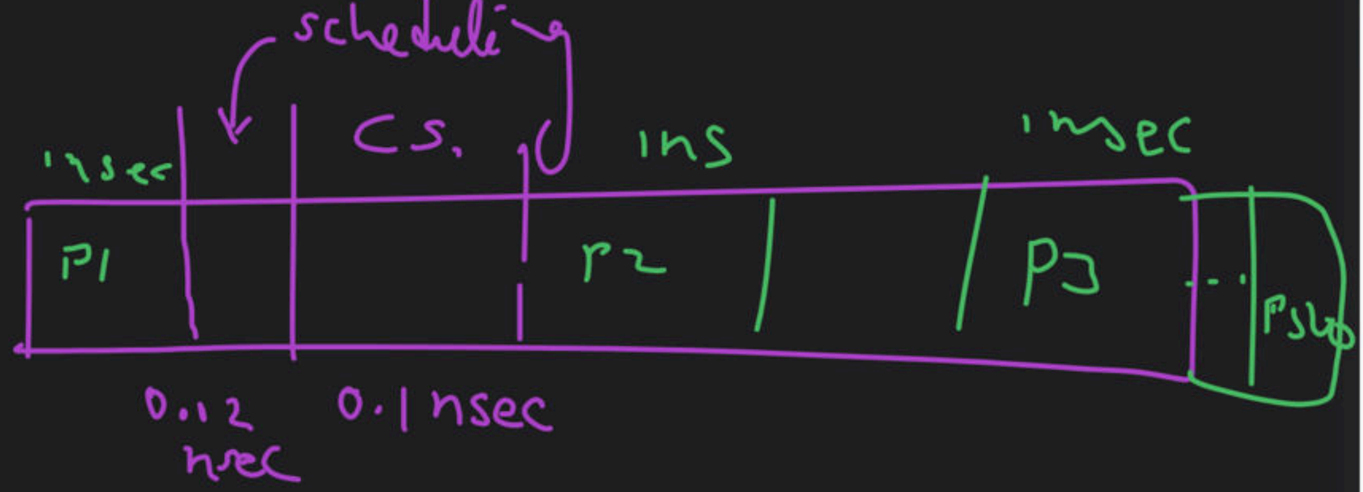
---

(2) $n > 1$ processes

FCFS
SRTF
} no preempt^n

3) $Q = t$

	B.T.	
P1	b	1
P2	b	1
⋮	b	1
Pn	b	1



499 times $\Rightarrow (0.02 + 0.1) =$

$0.12 * 499 = 59.88$

$b = t$

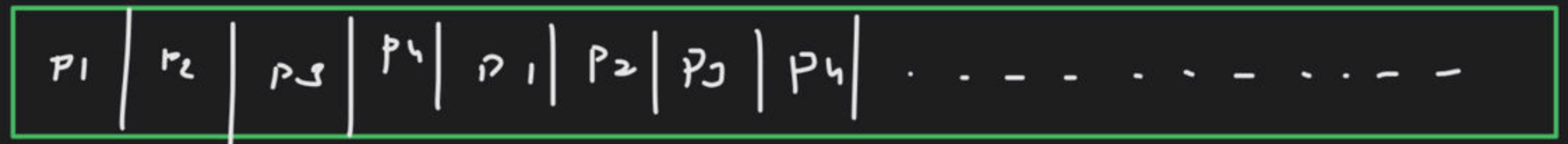
$n = 500$

$t = 1ns$

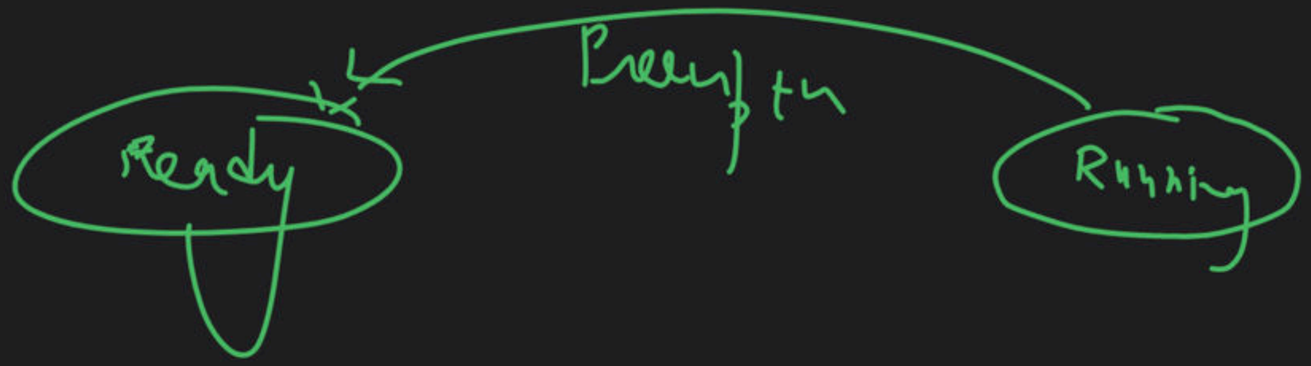
Process running time $= 500 * 1$ nsec $= 500$ nsec

Total time $= 559.88$ nsec

$\Rightarrow \frac{500}{559.88} * 100\% = 89.3\%$



$$\text{Avg WT} = \frac{0 + 25 + 98 + 106 + 123}{5} = 70.4$$



5)

P1, P2, P3, P4

Process

{ while (lock)

Run C.S.

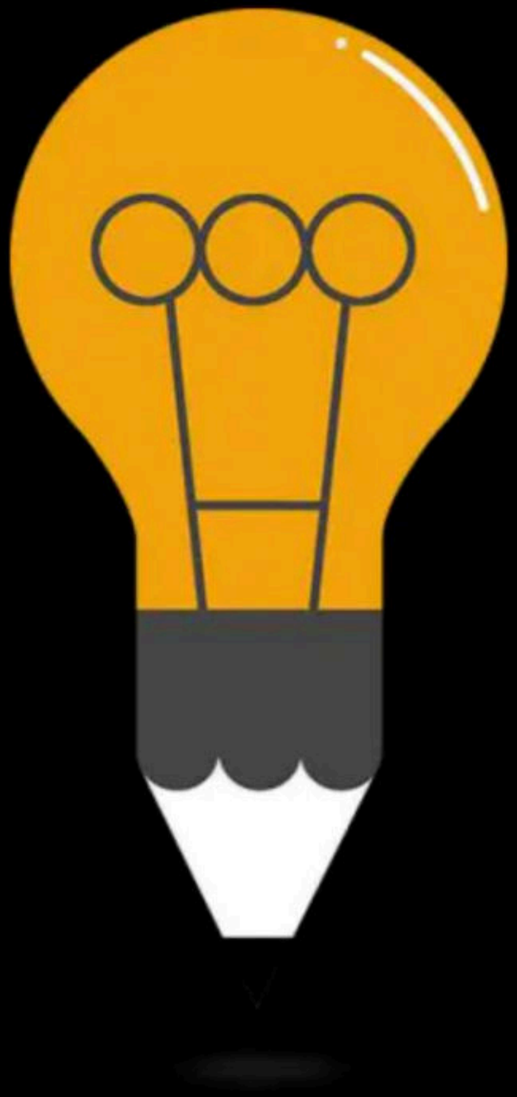
lock = True

}

6)

lock = ~~False~~ True.

unacademy



DPP

By: Vishvadeep Gothi

unacademy

Question 1

Similar to turn variable

```
boolean Lock = False;

while(true)
{
   while(Lock != False);
     CS
   turn=True;
     RS;
}
```

```
while(true)
{
   while(Lock! = True);
     CS
   turn=False;
     RS;
}
```

M.E. ✓
Progress ✓
B.W. ✓

unacademy

Question 2

```
Boolean lock=0;

while(true)
{
   while(! Lock);
   lock=1;
     CS
   lock=0;
     RS;
}
```

```
while(true)
{
   while(! Lock);
   lock=1;
     CS
   lock=0;
     RS;
}
```

Both process will run while (!lock); forever

M.E. ✓

Progress ✗

B.W. ✗

Deadlock ✓

Question GATE-2023

5 threads => inc.
3 — " — => dec

X = 10

Consider the two functions **incr** and **decr** shown below.

```
incr(){                 decr(){
    wait(s);                wait(s);
    X = X+1;                X = X-1;
    signal(s);              signal(s);
}                       }
```

There are 5 threads each invoking **incr** once, and 3 threads each invoking **decr** once, on the same shared variable X. The initial value of X is 10.

Suppose there are two implementations of the semaphore $\mathbf{s}$, as follows:

I-1: s is a binary semaphore initialized to 1.
I-2: s is a counting semaphore initialized to 2.

Let $\mathrm{V1}$, $\mathrm{V2}$ be the values of X at the end of execution of all the threads with implementations **I-1**, **I-2**, respectively.

Which one of the following choices corresponds to the minimum possible values of $\mathrm{V1}$, $\mathrm{V2}$, respectively?

A. $15, 7$
B. $7, 7$
C. $12, 7$ ✓
D. $12, 8$

I-1:- Mutual Exclusion

$10 + 5 - 3 = 12$

I2

$S = 2$

$10 + (1 + 1 + 1 + 1 + 1) - 1 - 1 - 1$

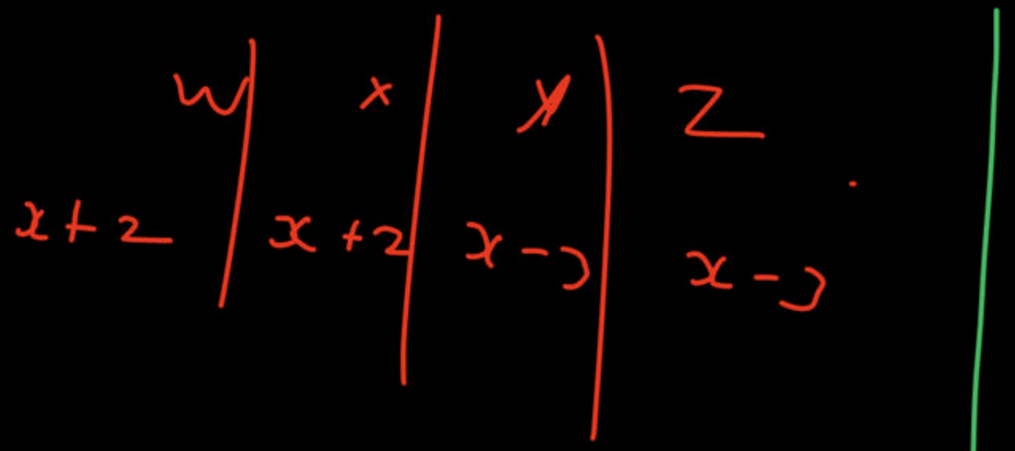
9

$10 - 3 = 7$

$S = \phi$ 1

Question 1

A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the process W and X reads x from memory, increments by 2, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory , decrements by 3, stores it to memory and then terminates. Each processes before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes *the* V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What are the minimum and maximum possible values of x after all processes complete execution?



max = 4

min = -6

unacademy

Question 2

A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the process W and X reads x from memory, increments by 2, stores it to memory and then terminates. Each of the processes Y and Z reads x from memory , decrements by 3, stores it to memory and then terminates. Each processes before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes *the* V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What are the total distinct possible values of x after all processes complete execution?

$0 + 2 + 2 - 3 - 3$

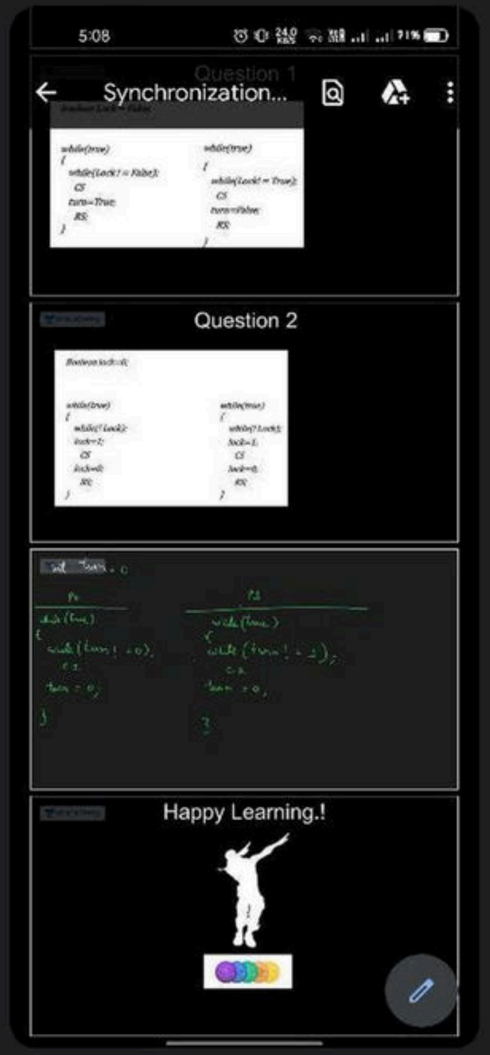
Ans = 8

4, 2, −3, −6, −2, −7, 1, −1

unacademy

▲ 1 • Asked by Riya

Please help me with this doubt



turn = 0

P0

while(turn != 0);

C.S

turn = 0

P1

while (turn != 1);

C.S.

Turn = 0

M.E. ✓ only P0 can enter in C.S always

Progress ✗ P1 can never enter in C.S.

B.W. ✗

Question GATE-2004

- Given below is a program which when executed spawns two concurrent processes:

```
Semaphore X:=0;
/* Process now forks into concurrent processes P1 & P2 */
P1 : repeat forever          P2:repeat forever
        V(X);                        P(X);
        Compute;                     Compute;
        P(X);                        V(X);
```

Consider the following statements about processes P1 and P2:

I. It is possible for process P1 to starve.

II. It is possible for process P2 to starve.

unacademy

Solution 1

```
Boolean lock=false;
```

```
while(true)
{
   while(lock);
   lock=true;
     CS
   lock=false;
     RS;
}
```

```
while(true)
{
   while(lock);
   lock=true;
     CS
   lock=false;
     RS;
}
```

Solution 2

```
int turn=0;
```

```
while(true)
{
   while(turn!=0);
     CS
   turn=1;
     RS;
}
```

```
while(true)
{
   while(turn!=1);
     CS
   turn=0;
     RS;
}
```

Solution 3: Peterson's Solution

```
Boolean Flag[2];
int turn;
```

```
while(true) {
    Flag[0]=true;
    turn=1;
    while(Flag[1] && turn==1);
        CS
    Flag[0]=False;
        RS;
}
```

```
while(true){
    Flag[1]=true;
    turn=0;
    while(Flag[0] && turn==0);
        CS
    Flag[1]=False;
        RS;
}
```

TestAndSet()

```
Boolean Lock=False;

boolean TestAndSet(Boolean *trg){
boolean rv = *trg;
*trg = True;
Return  rv;
}
```

```
while(true)
{
  while(TestAndSet(&Lock));
        CS
  Lock=False;
}
```

unacademy

Swap()

```
Boolean Key;                    //Local
Boolean Lock=False;             //Shared

void Swap(Boolean *a, Boolean *b)
{
boolean temp = *a;
*a=*b;
*b=temp;
}
```

```
while(true){
  Key = True;
  while (key==True)
        Swap(&Lock, &Key);
     CS
  Lock=False;
     RS
}
```

wait() & signal()

```
wait(S)
{
   while(S<=0);
   S--;
}
```

```
signal(S)
{
   S++;
}
```

Busy waiting :- (spin lock)

Process runs on CPU & keeps it busy ; but still can not proceed further.

lock = ~~false~~ True

```
while (lock);
```

```
while (lock);
```

Solutions Without Busy Waiting

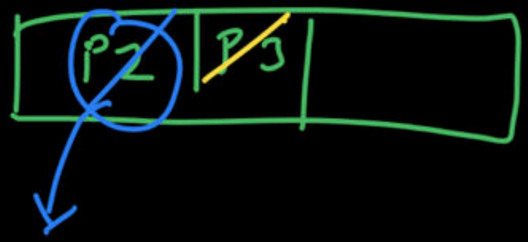
Semaphore can be negative

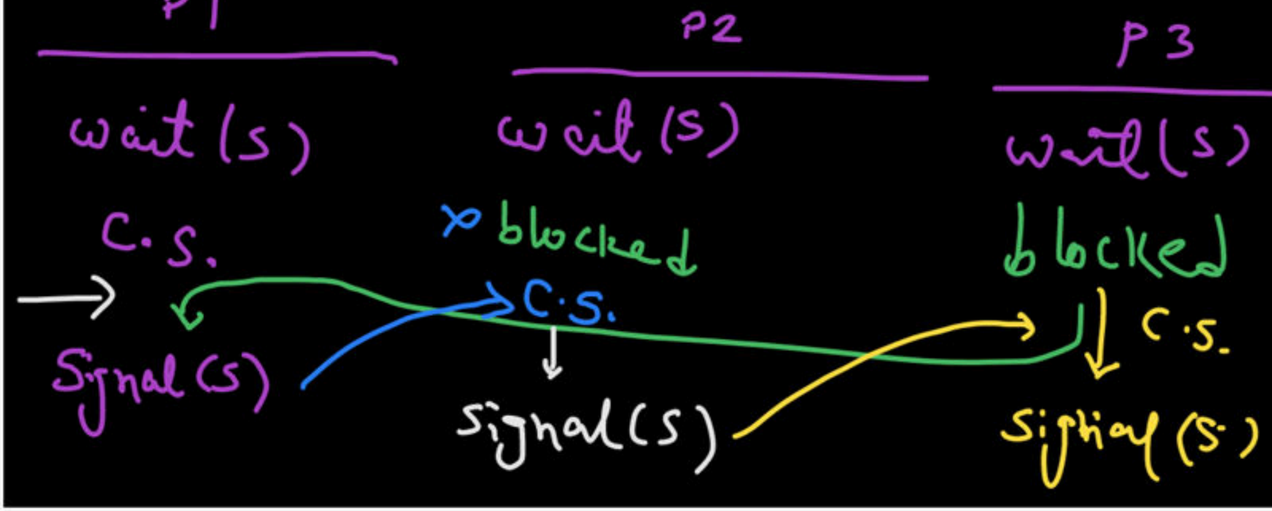
```
wait(Semaphore s){
   s=s-1;
   if (s<0) {
      // add process to queue
      block();
   }
}
```

```
signal(Semaphore s){
   s=s+1;
   if (s<=0) {
      // remove process p from queue
      wakeup(p);
   }
}
```

S = 1 0 -1 -2 -1 0 1

Queue

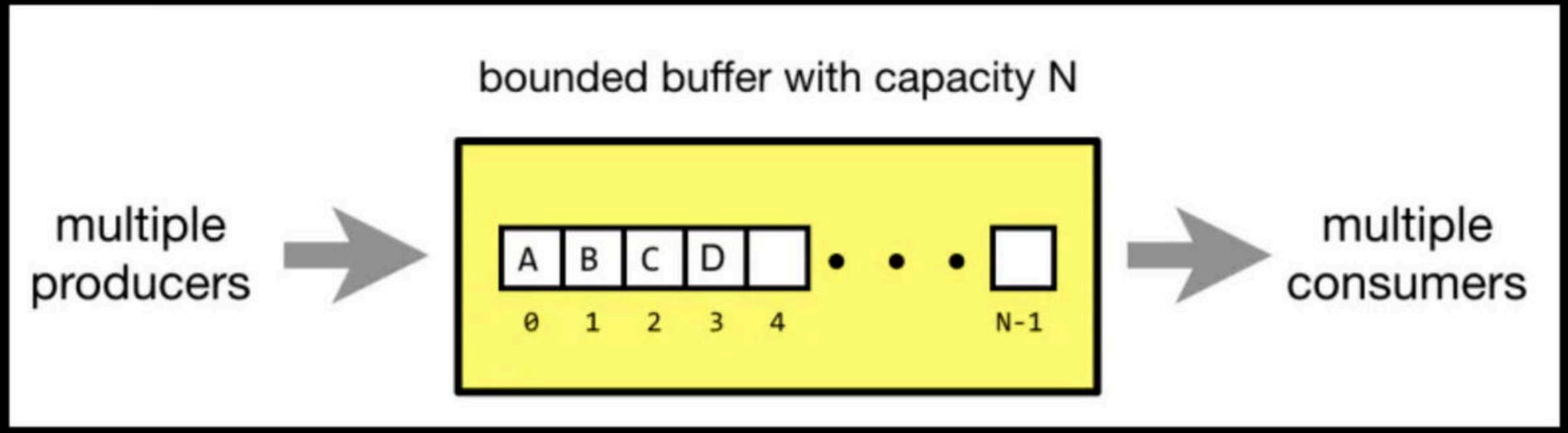


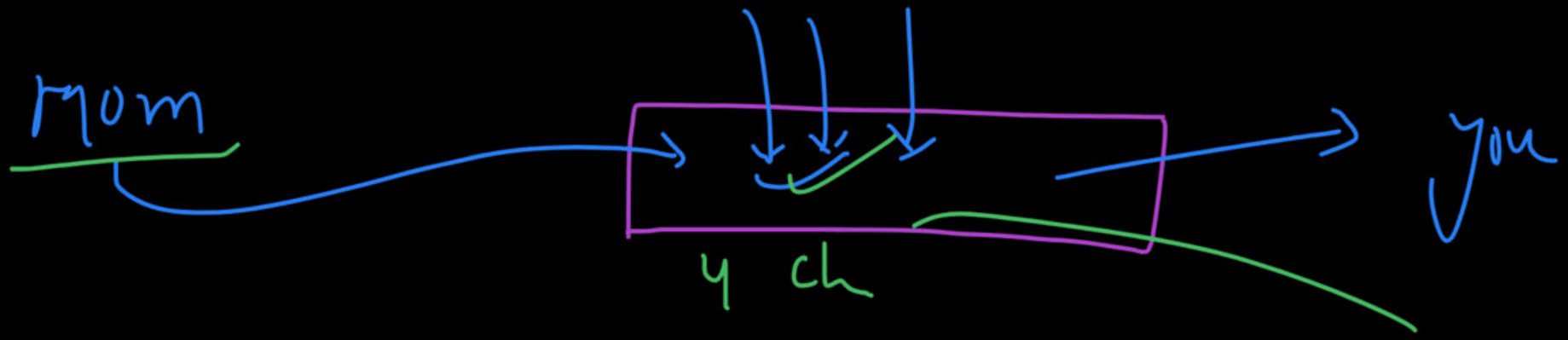


Classical Problems of Synchronization

1. Producer - consumer Problem (Bounded - buffer)
2. Reader - writer problem
3. Dining - philosopher problem

Bounded Buffer Problem





Bounded Buffer Problem

- Known as producer-consumer problem also
- Buffer is the shared resource between producers and consumers

Bounded Buffer Problem: Solution

- Producers must block if the buffer is full (no any space empty in buffer)
- Consumers must block if the buffer is empty (no any item in buffer)

Bounded Buffer Problem: Solution

- Variables:
  - Mutex: Binary Semaphore to take lock on buffer (Mutual Exclusion)
  - Full: Counting Semaphore to denote the number of occupied slots in buffer
  - Empty: Counting Semaphore to denote the number of empty slots in buffer

Initially when buffer is empty

mutex = 1

full = 0

empty = n

Producer()

```
wait(Empty)
wait(mutex)

//add item on buffer

signal(mutex)
signal(full)
```

Consumer()

```
wait (full)
wait (mutex)
// remove item from buffer
signal (mutex)
signal (Empty)
```

If first 2 statements (wait() statements) of consumer process are swapped, then there can be deadlock when buffer is empty.

If first 2 statements of producer process are swapped, then there can be deadlock when buffer is full.

Reader-Writer Problem

Consider a situation where we have a file shared between many people:

- If one of the people tries editing the file, no other person should be reading or writing at the same time, otherwise changes will not be visible to him/her
- However, if some person is reading the file, then others may read it at the same time

Reader-Writer Problem: Solution

- If writer is accessing the file, then all other readers and writers will be blocked
- If any reader is reading, then other readers can read but writer will be blocked

Reader-Writer Problem: Solution

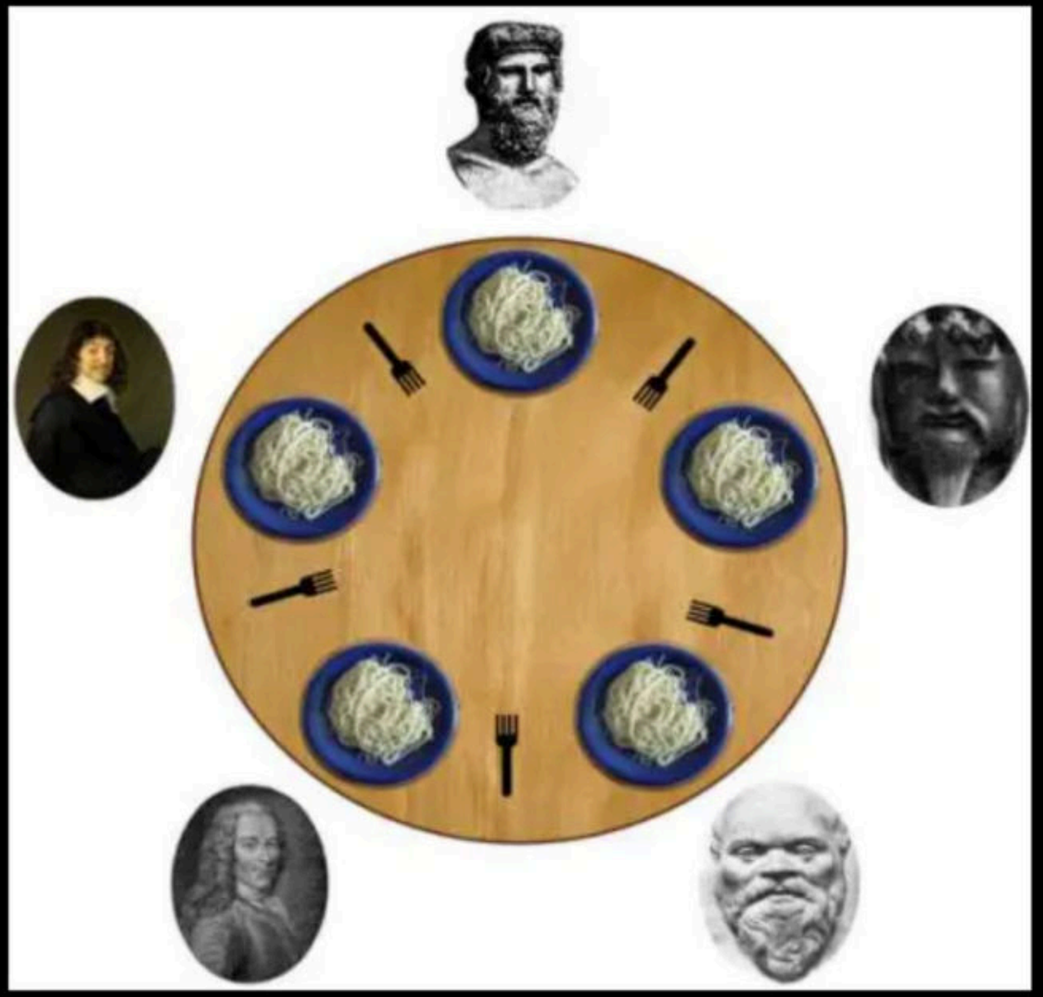
- Variables:
  - mutex: Binary Semaphore to provide Mutual Exclusion
  - wrt: Binary Semaphore to restrict readers and writers if writing is going on
  - readcount: Integer variable, denotes number of active readers

- Initialization:
  - mutex: 1
  - wrt: 1
  - readcount: 0

Writer() Process

Reader() Process

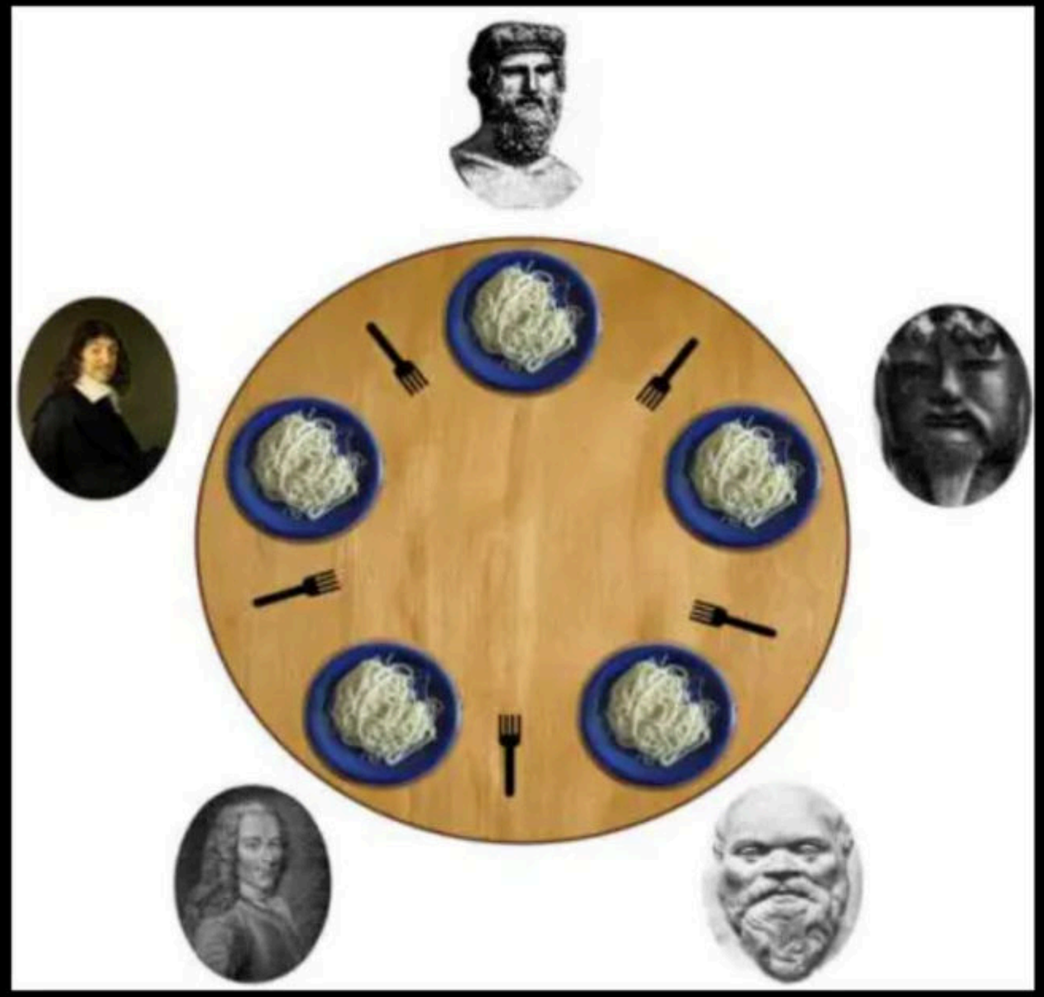
Dining Philosopher Problem



Dining Philosopher Problem

- K philosophers seated around a circular table
- There is one chopstick between each philosopher
- A philosopher may eat if he can pick up the two chopsticks adjacent to him
- One chopstick may be picked up by any one of its adjacent followers but not both

Dining Philosopher Problem: Solution



Dining Philosopher Problem: Solution

Dining Philosopher Problem: Solution

Dining Philosopher Problem: Solution

Some of the ways to avoid deadlock are as follows –

1. There should be at most (k–1) philosophers on the table

Dining Philosopher Problem: Solution

Some of the ways to avoid deadlock are as follows −

1. There should be at most (k–1) philosophers on the table
2. A philosopher should only be allowed to pick their chopstick if both are available at the same time

Dining Philosopher Problem: Solution

Some of the ways to avoid deadlock are as follows –

1. There should be at most (k–1) philosophers on the table
2. A philosopher should only be allowed to pick their chopstick if both are available at the same time
3. One philosopher should pick the left chopstick first and then right chopstick next; while all others will pick the right one first then left one

Dining Philosopher Problem: Solution

unacademy

Happy Learning.!

→ clan Problem of sync.
→ Threads
→ Sys call
→ Fork ()

Deadlock