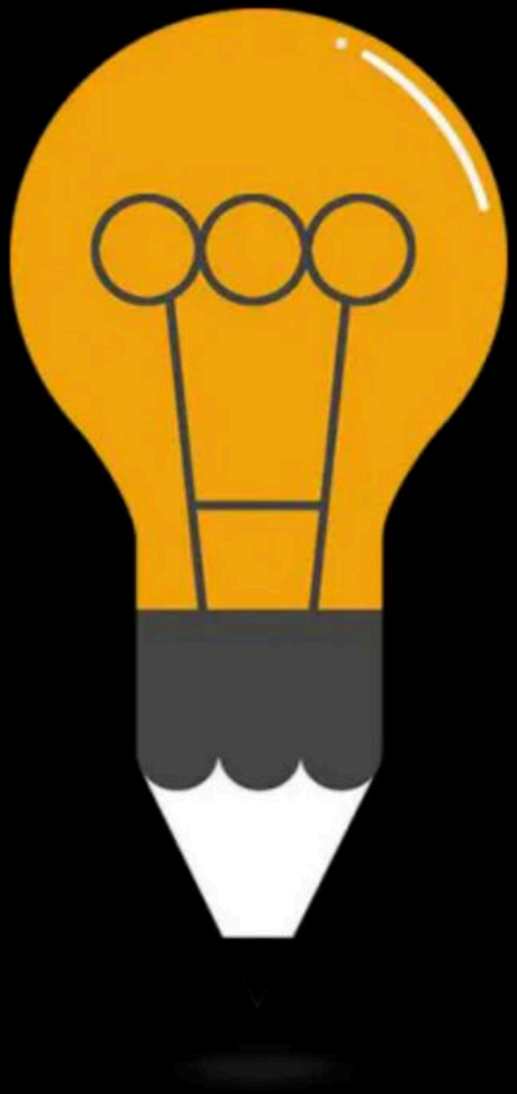
unacademy

Two-Process Solution of Critical Section

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 13 • May 4, 2023

unacademy



Operating System
Process Synchronization

By: Vishvadeep Gothi

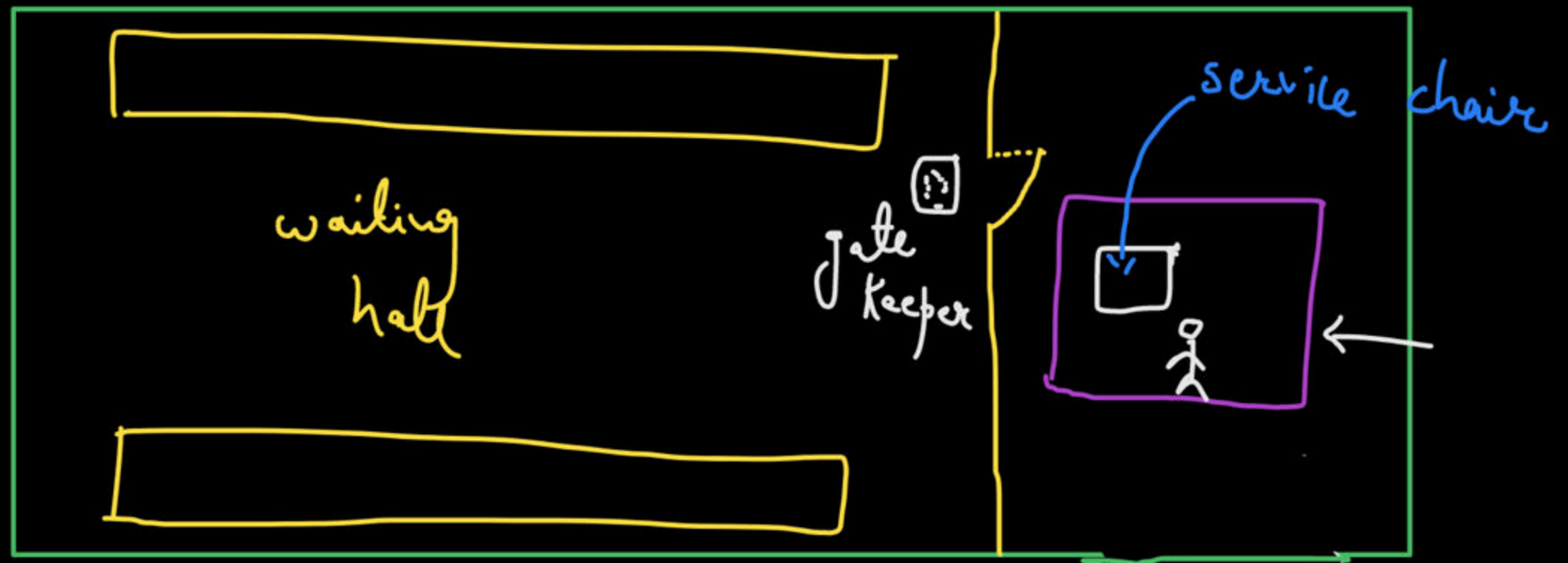
while (condition);

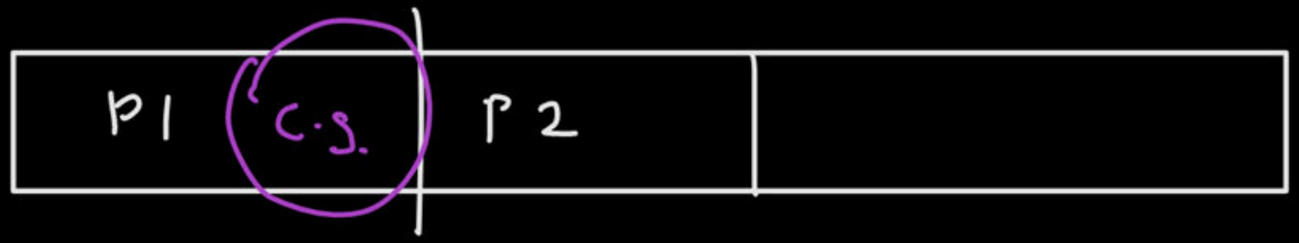
while (condition)
{

}

Critical Section

The critical section is a code segment where the shared variables can be accessed





Solution of Critical Section Problem

Requirements of Critical Section problem solution:

1. Mutual Exclusion
2. Progress
3. Bounded Waiting

---

1. Mutual Exclusion :-

If one process is executing critical section, then other process should not be allowed to enter & execute the critical section

2. Progress :-

If no any process is in C.S. and a process wants to enter into C.S., then the process should be allowed to enter into C.S.

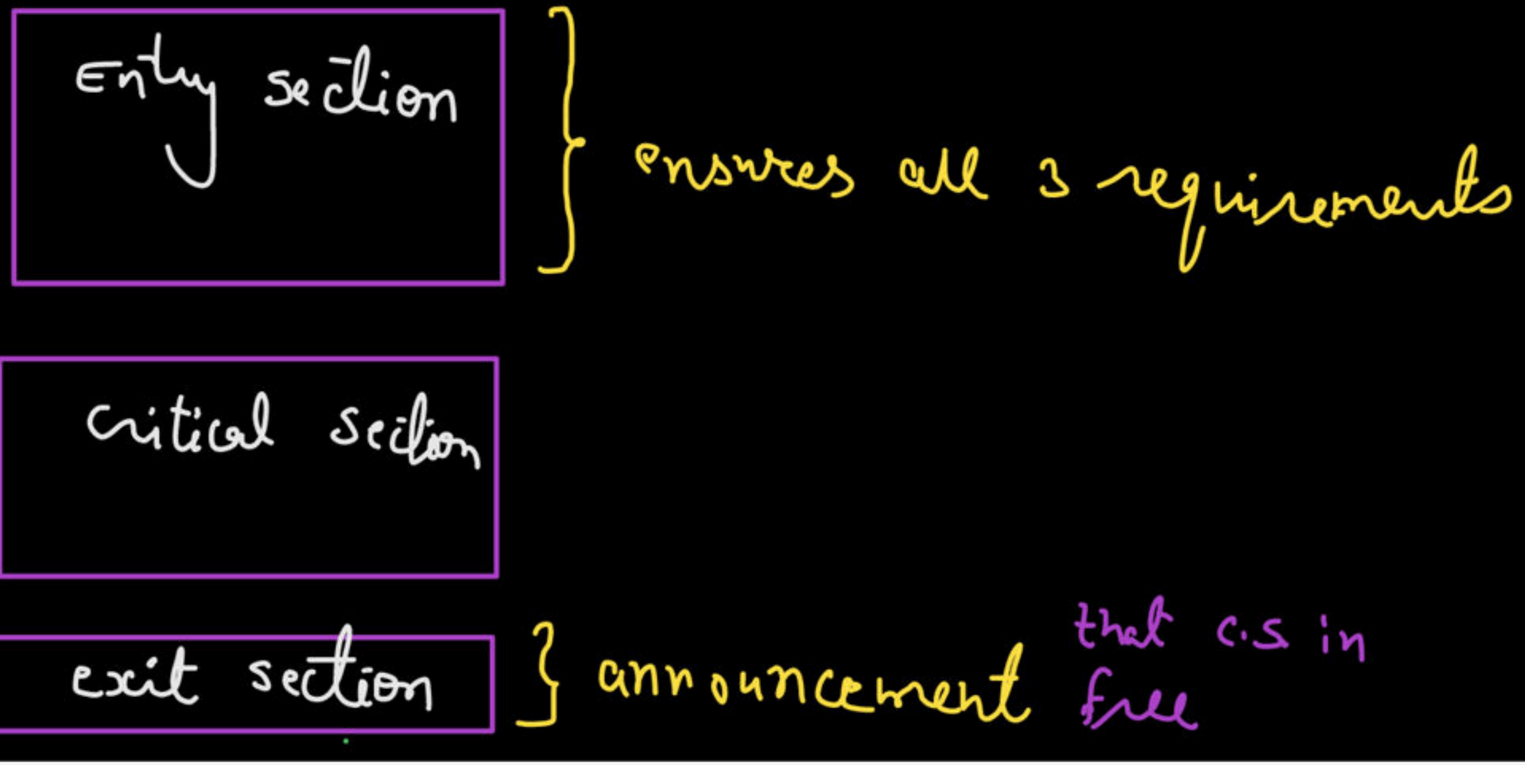
---

3. Bounded - waiting _ :- (fairness)

If a process is in C.S., and other process is waiting for C.S., then the first process should not be allowed to enter into C.S. again by keeping second process waiting for C.S.

2-Process Solution

① s/w solution :- Soln through Code



Solution 1

lock => false => no any process in C.S.
↳ true => process in C.S.

```
Boolean lock=false;
```

P1

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

P2

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

1. M.E ✗
2. Progress ✓
3. Bounded waiting ✗

lock = ~~false~~ ~~True false~~ True

Mutual Exclusion:- lock = ~~false~~ ✗ T

P1	P2	P1	P2	
while (lock);	while(lock);	lock = true		
	lock = true	C.S.	C.S.	

Progress:- (P1) lock = ~~false~~ ~~true~~ false

(P2) lock = ~~false~~ ~~true~~ false

(3) Bounded waiting :- Situat^n => P1 in C.S. | Preempt^{ed} | P2 waiting for C.S. |

lock = ~~false~~ ~~True~~ ~~false~~

True

True

Preempt | P1 comes out of C.S. & tries to enter into C.S. again

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;
}
```

unacademy

Happy Learning.!