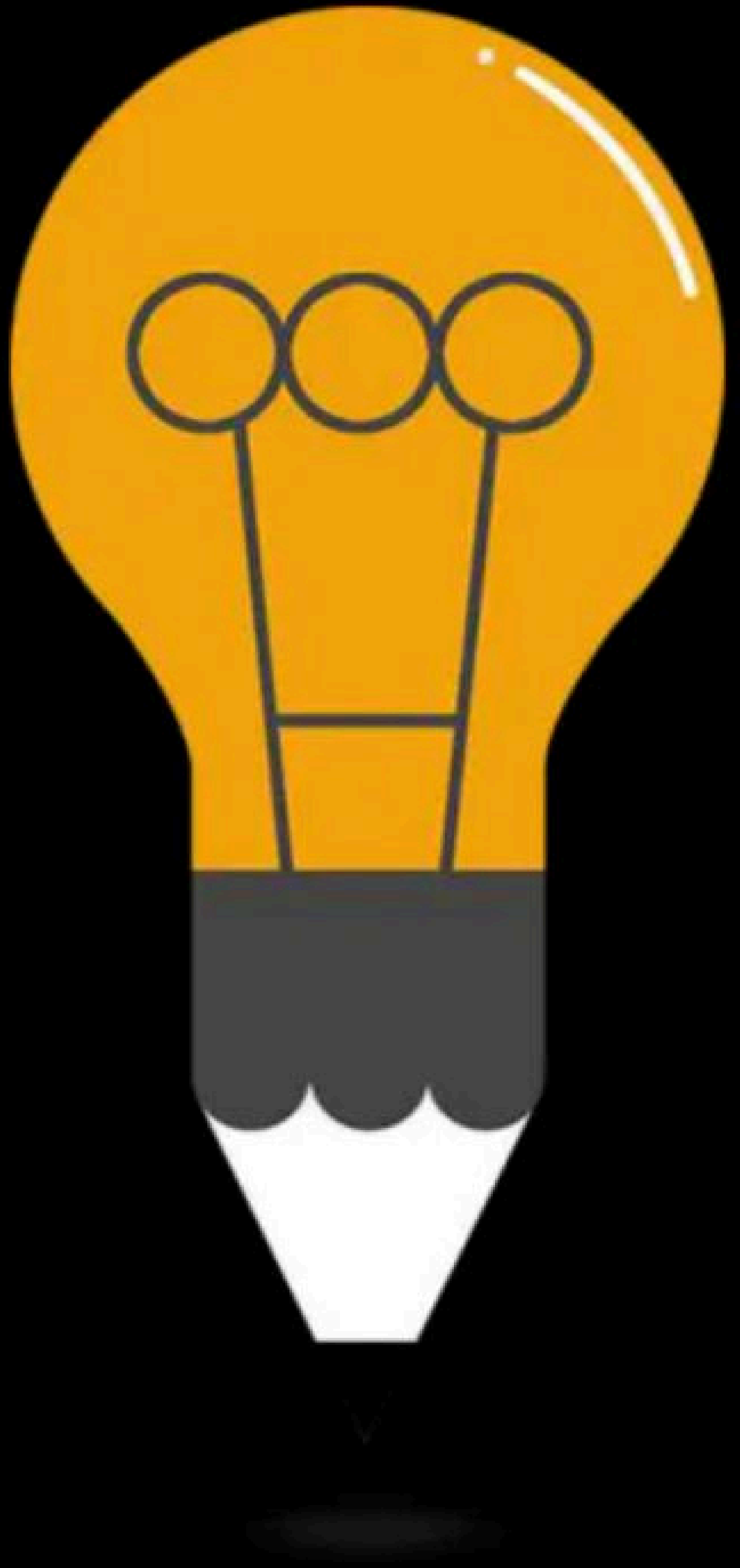
Multithreading, System Call & Fork Call

Comprehensive Course on Operating System for GATE - 2024/25

Vishvadeep Gothi • Lesson 15 • May 6, 2023



Operating System

Process Synchronization

By: Vishvadeep Gothi

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;
}
```

Synchronization Hardware

↳ inst^{ns} provided in cpu arch.,

so that inst^{rs} can be used fore synch..

✓1. TestAndSet()

✓2. Swap()

Flag is boolean

TestAndSet()

Returns the current value flag and sets it to true.

TestAndset (Flag)

if Flag == false

↳ Flag = True

Return false

if flag == True

↳ flag = True

Return True

TestAndSet()

Shared

Boolean Lock=~~False;~~ True

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

P1 , P2

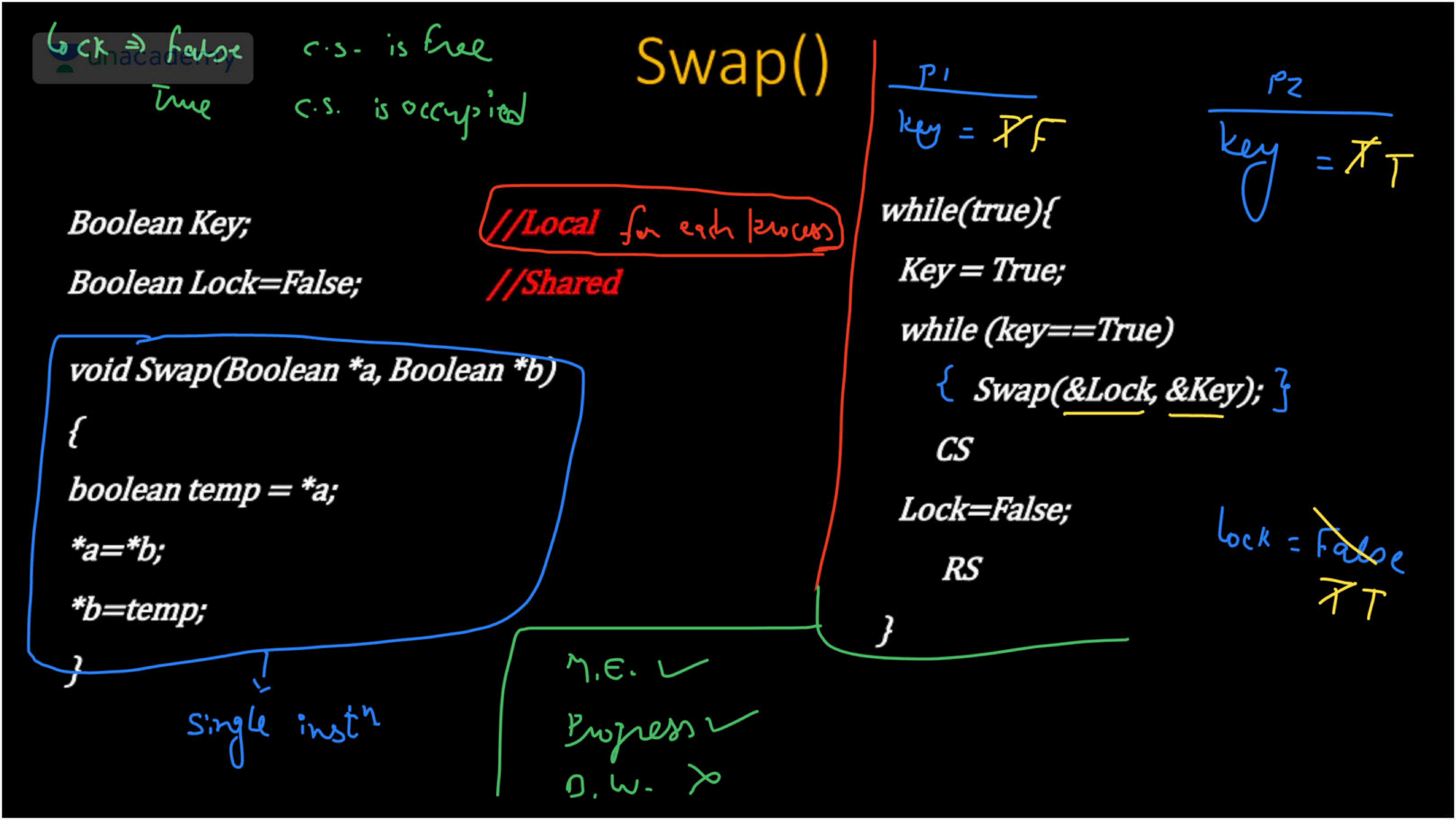
```
while(true)
{
  while(TestAndSet(&Lock));
        CS
  Lock=False;
}
```

M.E. ✓

Progress ✓

Bounded waiting ✗

Confusion → clear ✗



lock ⇒ false   c.s. is free

True   c.s. is occupied

Swap()

```
Boolean Key;              //Local for each process
Boolean Lock=False;       //Shared

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

single inst^n

P1

key = ~~T~~ F

P2

key = ~~T~~ T

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

lock = ~~False~~ ~~T~~ T

M.E. ✓

Progress ✓

B.W. ✗

Synchronization Tools

↳ provided by os,

↳ to be used to provide synch.

✓ 1. Semaphore

✗ 2. Monitor

Semaphore

- unsigned Integer value which can be accessed using following functions only
  - wait() / P() / Degrade() → decreases semaphore value by 1
  - signal() / V() / Upgrade() → increases — " —— " ——

atomic

wait() & signal()

S is semaphore here

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

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

Types of Semaphore

Binary Semaphore

value can be either 0 or 1

0, $\underline{1}$

Counting Semaphore

value can be any +ve integer

0, $\underline{1}$, 2, 3, 4, 5, 6, 7, ...

Types of Semaphore

Binary Semaphore	Counting Semaphore
It is used to implement the solution of critical section problems with multiple processes	It is used to control access to a resource that has multiple instances

↓

mutual exclusion

Critical Section Solution

Binary semaphore

S = ~~1~~ ~~0~~ ~~1~~ 0

P1

```
while(True)
{
 wait(S)
     C.S.
 signal(s)
}
```

P2

```
while ( True )
{
  wait ( s )
     c.s.
  signal(s)
}
```

Characteristics of Semaphores

- Used to provide mutual exclusion ✓
- Used to control access to resources ✓
- Solution using semaphore can lead to have deadlock
- Solution using semaphore can lead to have starvation
- Solution using semaphore can be busy waiting solutions
- Semaphores may lead to a priority inversion
- Semaphores are machine-independent ✓

Happy Learning.!

wed 8:30 pm

sat 2-3 howes