

Module: Linux System Programming

Session: Process Synchronization


By

Donkada Mohana Vamsi

C-DAC Hyderabad




Need for Process Synchronization

- Process synchronization is required, when several concurrent processes want to access a resource (CPU, printer etc.) then to get the correct result, synchronization is required.
 - Only cooperative processes need synchronization.
- 

Process Synchronization

Independent Process : Independent processes are not affected by any other process, so no synchronization is required as the processes are not dependent on each other



Process Synchronization

- **Cooperative Process** : Cooperative process is one that can affect or be affected by other processes executing in a system
- Cooperative processes can either directly share a logical address space or be allowed to share data only through files or messages



Process Synchronization

Consider initial value of $X=10$

P1		P2	
Read (x)		Read (x)	
$X=x+1$		$X=x+10$	
$X=11$	Write (x)	Write (x)	$X=20$

The order of execution may change the overall result. It may $P1 < P2$ or $P2 < P1$

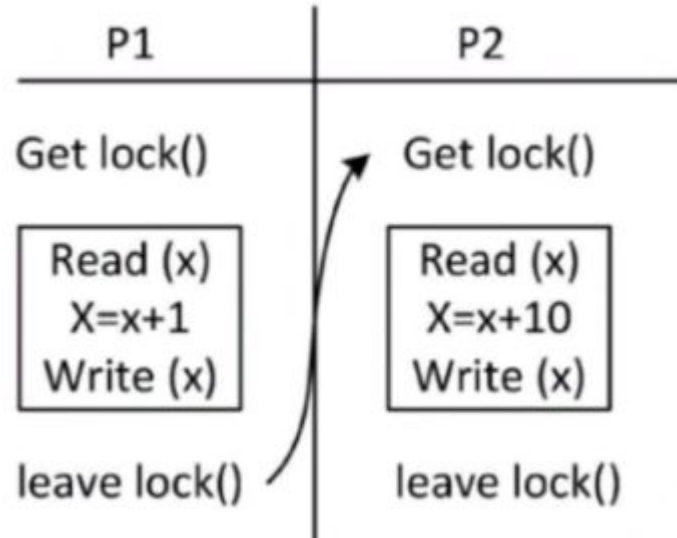
Process Synchronization

The different value of X is stored in both cases because $P1$ and $P2$ are cooperative process and synchronization is not maintained

P1		P2	
X=11	Read (x)	Read (x)	X=20
	X=x+1	X=x+10	
	Write (x)	Write (x)	

Process Synchronization


In order to perform synchronization locks are used



Process Synchronization

- P1 gets the lock initially - So P2 waits still P1 release the lock

OR

- P2 gets the lock initially - So P1 waits till P2 release the lock
 - This kind of approach yields the synchronization(i.e final value of X remains 21)
- 

Process Synchronization

Advantage of lock :

- Consistent updation
- Correct result
- Synchronization is achieved



Process Synchronization

	P1		P2	
	Read (x)		Read (x)	
	X=x+1		X=x+10	
X=11	Write (x)		Write (x)	X=20

A situation like this, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition**

Process Synchronization

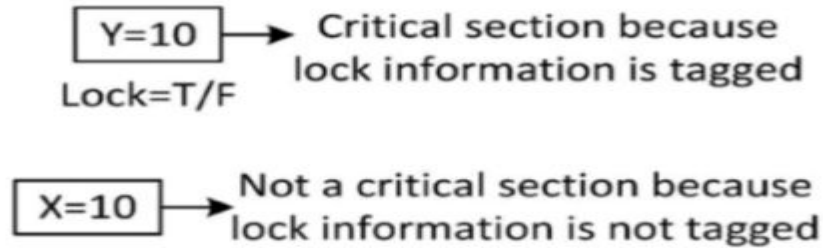
- Consider a system consisting of n processes $\{P_0, P_1, \dots, P_{n-1}\}$. Each process has a segment of code, called a **critical section**, in which the process may be changing common variables, updating a table, writing a file, and so on.



Process Synchronization


- Any shared resource on which lock status information is tagged, such resource is called a **critical section**

Example:



Process Synchronization

Critical section can be defined as :

- A critical section is a section where we perform work related to sharing
 - A portion of program text where shared variables or shared resources will be placed/accessed
- 

Process Synchronization

Critical section problems

- No two process can execute in the critical section
- Lack of decision making regarding which process to wait

The above problems can be solved by fulfilling below requirements

- Mutual Exclusion
 - Progress
 - Bounded Waiting
- 

Process Synchronization

- Each process must request permission to enter its critical section. The section of code implementing this request is the **entry section**
- Entry section followed by **exit section**
- The remaining code is **remainder section**

Structure of process

Do
{

Entry section

Every section have code

CS

CS: Critical section

Exit section

Remainder section


}

While(1)/ while(true);


Process Synchronization

Mutual Exclusion is a property due to which a process agrees to wait until the resource become free and access

Example :

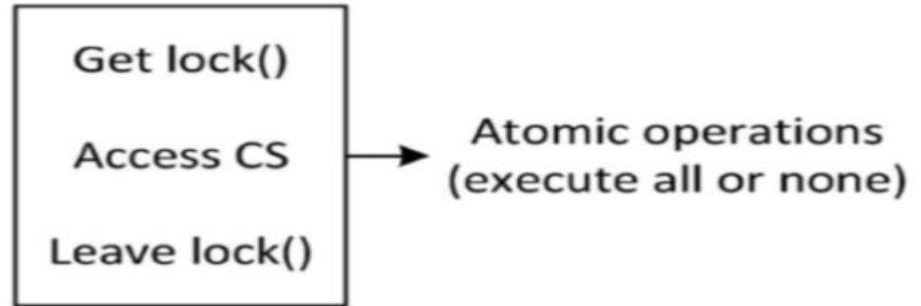
- Initially **lock = F** , means the resource is free.
 - **Lock status** information is **tagged**, so the **resource** is in **critical section**
- 

Process Synchronization

- Now, P1 comes and execute **Get lock()**. P1 acquires lock and start using resources. Now **lock=T**
 - Now, P2 want to use the resource , it will execute **get lock()**, but **lock=T** means the resource is being used by another process
 - Therefore, P2 has to wait till P1 execute and **leave lock()**.
 - In this case, P2 agrees to wait because the resource is not free. This property is called **mutual exclusion**
- 

Process Synchronization


- **Synchronization** is a method to solve the conflict between concurrent process i.e. which process will be executed first.
- In synchronization every process must go through following three steps



Process Synchronization

Mutual exclusion : Two process can't enter simultaneously inside the critical section at any point in time. Only one process is permitted to enter and use resources in critical section at any point in time


Bound waiting : There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.



Process Synchronization

Progress : This solution is used when no one is in the critical section, and someone wants in. Then the selection of the processes that will enter the critical section next cannot be postponed indefinitely

The above said three rules(i.e **mutual exclusion, bound waiting,progress**) must enforce to overcome critical section problem




Process Synchronization

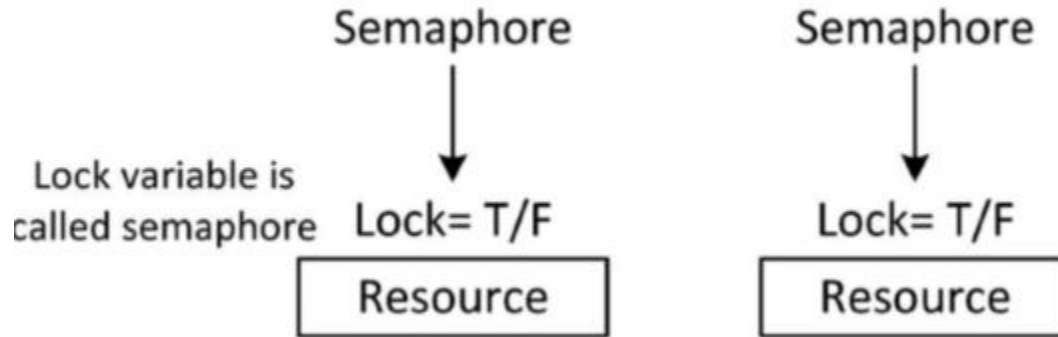
Semaphore is a special integer variable which apart from initialization is accessed through two atomic operations **P** and **V** where

- **P** denotes **wait** operation
- **V** denotes a **signal** operation

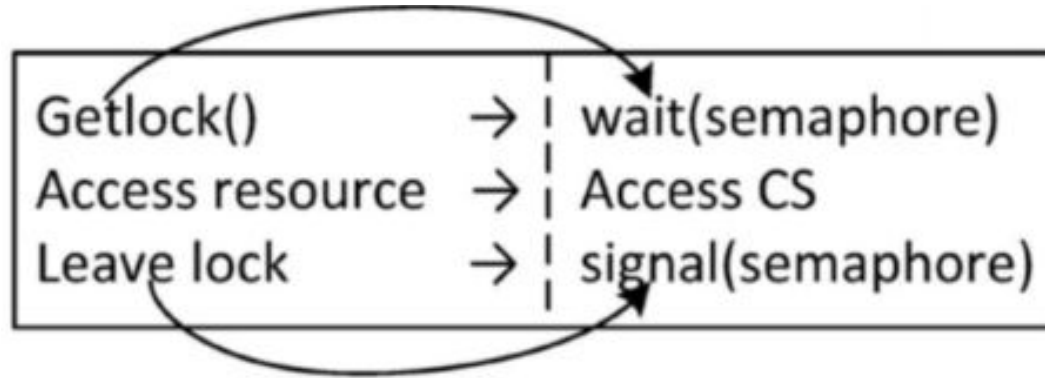
Atomic : The operation which will be executed without any interruption. **P** & **V** both must be executed. **V(s)** can't be executed before **P(s)**



Process Synchronization



Process Synchronization




Process Synchronization

The characteristics of **wait** operation are:

- Wait operation is denoted by P
- It is always placed before the critical section
- It is a software logic, it must be executed
- Wait operation acquires a lock for the current process.
- Wait operation unsuccessful implies block current process because the resource is not available. Therefore, it is unable to acquire the lock
- Wait operation decrement semaphore value by 1
- $P = \text{wait} = \text{block} = \text{down}$


Process Synchronization

The characteristics of **Signal** operation are:

- Signal operation is denoted by V
 - Signal operation is used after the critical section
 - This operation release acquired lock.
 - It is a software logic
 - This operation increment semaphore value by 1
 - $V = \text{Signal} = \text{release} = \text{up}$
- 

Process Synchronization

Types of semaphore: There are two types of semaphores given as follows

- **Binary semaphore:** the value of semaphore is 0 & 1 i.e. Boolean variable. Either value 0 or 1 can be assigned to a variable
 - **Counting semaphore:** the value of semaphore can be anything in between $-\infty$ to $+\infty$. The value of semaphore(s) = integer domain
- 

Thank you

