Lecture 5. Process Synchronization

Lecturer: Aidana Karibayeva

Process Synchronization is a way to coordinate processes that use shared data. It occurs in an operating system among cooperating processes. **Cooperating processes** are processes that share resources. While executing many concurrent processes, process synchronization helps to maintain shared data consistency and cooperating process execution. Processes have to be scheduled to ensure that concurrent access to shared data does not create inconsistencies. Data inconsistency can result in what is called a **race condition**. A race condition occurs when two or more operations are executed at the same time, not scheduled in the proper sequence, and not exited in the critical section correctly.

On the basis of synchronization, processes are categorized as one of the following two types:

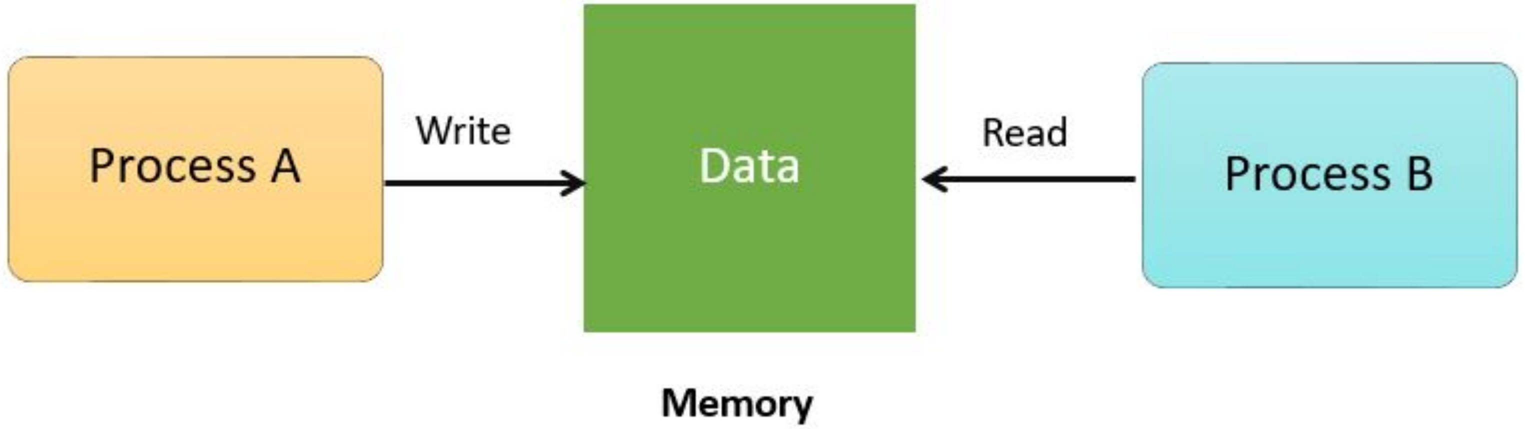
- **Independent Process** : Execution of one process does not affects the execution of other processes.
- **Cooperative Process** : Execution of one process affects the execution of other processes.

Process Synchronization

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process executions.

For Example, process A changing the data in a memory location while another process B is trying to read the data from the **same** memory location. There is a high probability that data read by the second process will be erroneous.



Critical section

A **critical section** is a segment of code that can be accessed by only one signal process at a certain instance in time. This section consists of shared data resources that need to be accessed by other processes. The entry to the critical section is handled by the **wait()** function, represented as P(). The exit from a critical section is controlled by the **signal()** function, represented as V(). Only one process can be executed inside the critical section at a time. Other processes waiting to execute their critical sections have to wait until the current process finishes executing its critical section.

Any solution to the critical section problem must satisfy three requirements:

- **Mutual Exclusion** : If a process is executing in its critical section, then no other process is allowed to execute in the critical section.
- **Progress** : If no process is executing in the critical section and other processes are waiting outside the critical section, then only those processes that are not executing in their remainder section can participate in deciding which will enter in the critical section next, and the selection can not be postponed indefinitely.
- **Bounded Waiting** : A bound must exist 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.

Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.

Summary

- Process synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
- Four elements of critical section are 1) Entry section 2) Critical section 3) Exit section 4) Reminder section
- A critical section is a segment of code which can be accessed by a signal process at a specific point of time.
- Three must rules which must enforce by critical section are : 1) Mutual Exclusion 2) Process solution 3)Bound waiting
- Mutual Exclusion is a special type of binary semaphore which is used for controlling access to the shared resource.

Summary

- Process solution is used when no one is in the critical section, and someone wants in.
- In bound waiting solution, after a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section.
- Peterson's solution is widely used solution to critical section problems.
- Problems of the Critical Section are also resolved by synchronization of hardware
- Synchronization hardware is not a simple method to implement for everyone, so the strict software method known as Mutex Locks was also introduced.
- Semaphore is another algorithm or solution to the critical section problem.

Thank you for your attention!