**Exercise 4a – Critical Section and Synchronization Techniques**

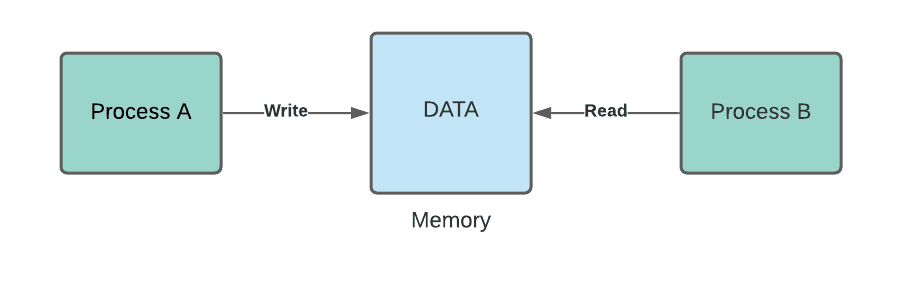
**Notes from Internet – Series-3**

**Process Synchronization in Operating System**

Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data. It is required in a multi-process system where multiple processes run together, and more than one process tries to gain access to the same shared resource or data at the same time.

Changes made in one process aren’t reflected when another process accesses the same shared data. It is necessary that processes are synchronized with each other as it helps avoid the inconsistency of shared data.

For example: A process P1 tries changing data in a particular memory location. At the same time another process P2 tries reading data from the same memory location. Thus, there is a high probability that the data being read by the second process is incorrect.



**Sections of a Program in OS:** Following are the four essential sections of a program:

1. Entry Section: This decides the entry of any process.

2. Critical Section: This allows a process to enter and modify the shared variable.

3. Exit Section: This allows the process waiting in the Entry Section, to enter into the Critical Sections and makes sure that the process is removed through this section once it’s done executing.

4. Remainder Section: Parts of the Code, not present in the above three sections are collectively called Remainder Section.

**Types of process in Operating System**

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

1. Independent Processes: The execution of one process doesn’t affect the execution of another.

2. Cooperative Processes: Execution of one process affects the execution of the other. Thus, it is necessary that these processes are synchronized in order to guarantee the order of execution.

**Critical Section Problem in OS**

Consider a system with n processes (P0, P1, …, Pn-1). Every process has a critical section of code in which the process may change common variables, update a table, write a file, and so on. When one process is running in its critical section, no other process is permitted to run in that area.

The term "critical section" refers to a code segment that is accessed by several programs. The critical section includes shared variables or resources that must be synchronized in order to ensure data consistency. [A segment of code that a signal process can access at a particular point of time is known as the critical section. It contains the shared data resources that can be accessed by other processes.]

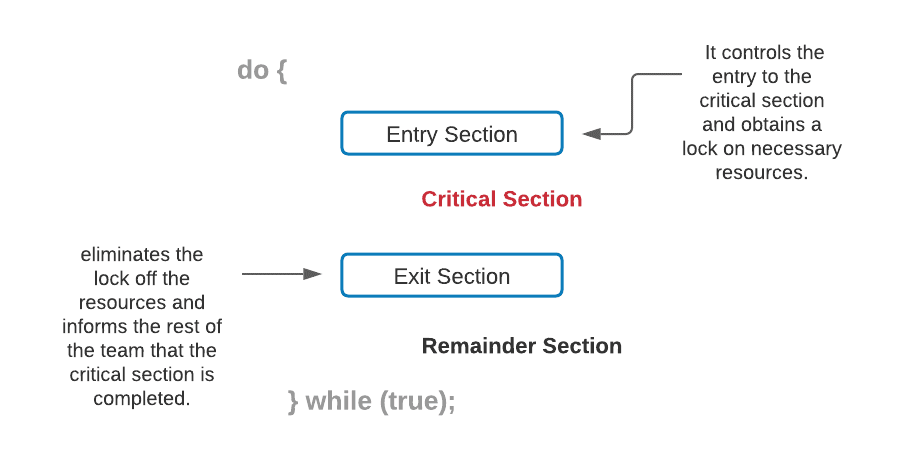
It's a characteristic of a program that seeks to access shared resources. Any resource in a computer, such as a CPU, data structure, any IO device, or memory location.

***There are two methods that control the entry and exit from the critical section:***

**The wait() method controls the entry to the critical section,**

**whereas the signal() function controls the exit.**

A single process executes in a critical section at a time. Other processes execute after the current process is done executing.



**Race Condition in OS**

When more than one processes execute the same code or access the same memory/shared variable, it is possible that the output or value of the shared variable is wrong. In this condition, all processes race ahead in order to prove that their output is correct. This situation is known as race condition.

When multiple processes access and manipulate the same data concurrently the outcome depends on the order in which these processes accessed the shared data. When the output of multiple thread execution differs according to the order in which the threads execute, a race condition occurs.

We can avoid it if we treat the critical section as an atomic instruction and maintain proper thread synchronization using locks or atomic variables.

**Criteria for Synchronization Mechanisms**

Any proposed synchronization technique dealing with the critical section problem must fulfill the following requirements:

**Mutual Exclusion**

The system must make certain that-

* The processes have mutually exclusive access to the critical section.
* At any one time, only one process should be present inside the critical section.

Mutual exclusion is a form of binary semaphore that is used to manage access to a shared resource. To prevent prolonged priority inversion concerns, it features a priority inheritance mechanism.

**Progress**

When no one is in the critical section and someone wants entry, this solution is utilized. Then, in a certain amount of time, those processes not in their reminder section should select who should go in. If no process is in its critical section and one or more threads desire to execute it, any of these threads must be permitted to do so.

**Bounded Waiting**

When a process requests to be placed in the critical section, there is a limit to the number of processes that may be placed in that area. As a result, after the limit has been reached, the system must enable the process to enter its critical section.

Note: The essential requirements are Mutual Exclusion and Progress. All synchronizing mechanisms must meet these requirements. The optional criteria are bounded waiting. However, if feasible, this criterion should be met.

**Solutions To the Critical Section**

In Process Synchronization, the critical section plays an essential part in resolving the problem. The following are the key approaches with respect to solving the critical section problem:

**Peterson’s Solution**

This is a software-based solution to critical section problems that are extensively employed. Peterson's solution was created by a computer scientist named Peterson, as the name itself suggests.

When a process is at a critical section, this approach allows the other process to just execute the remaining code and vice versa. This strategy also assists in ensuring that only one process may execute in the critical section at any given moment.

All three requirements are preserved in this solution:

* Mutual exclusion ensures that only one process has access to the vital area at any one moment.
* Progress is also reassuring because a process that is not in the crucial area cannot prevent other processes from entering it.
* Every process is given a fair chance to join the Critical section, ensuring bounded waiting.

**Example:**

ANY PROCESS Pi

FLAG[i] = true

while( (count != i) AND (CS is !free) ){ wait;

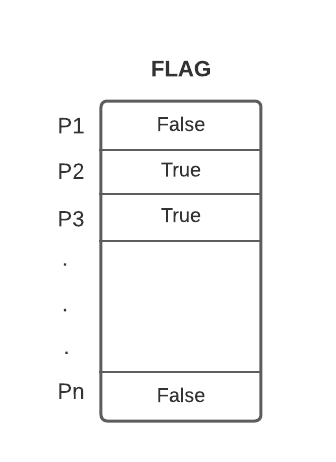
}

FLAG[i] = false

count = j; //after this, choose another process to access CS

**Explanation:**

Suppose there are N processes (P1, P2,... PN) and each process must enter the Critical Section at some point. A FLAG[] array of size N is kept, with the value false by default. As a result, anytime a process needs to reach the critical section, it must set the critical section flag to true. If Process Pi wishes to enter, for example, it will set FLAG[i]=TRUE.



* The count variable shows the process number that is presently waiting to be entered into the CS.
* While quitting, the process that enters the critical section would modify the count to a different number from the list of ready processes.
* Example: When a count equal to 2 is reached, P2 enters the Critical section, and while exiting it sets the count to 3, and hence P3 leaves the wait loop.

### **Synchronization Hardware**

For critical section code, several systems include hardware assistance. If we could prevent interruptions from occurring while a shared variable or resource is being updated, we could easily address the critical section problem in a single-processor system.

We could be certain that the present sequence of instructions would be permitted to run in order without being pre-empted in this way. Unfortunately, in a multiprocessor system, this method is not possible.

In a multiprocessor environment, disabling interrupts might take a long time since the message is sent to all processors. The arrival of threads into the critical section is delayed as a result of the message transmission lag, and system efficiency decreases as a result.

### **Mutex Locks**

Since synchronizing hardware is a difficult approach to install for everyone, Mutex Locks were created as a strict software method.

In this method, a LOCK is gained over the important resources used inside the critical section in the entrance part of the code. This lock is released in the exit section.

### **Semaphore Solution**

Semaphore is a non-negative variable that is shared across threads. It's a different algorithm or solution to the problem of the critical section. It's a thread that's waiting for a semaphore, which can be signaled by another thread.

For process synchronization, it employs two atomic operations: 1) wait() and 2) signal().

### **Uses of Critical section**

#### **Critical sections in data structures**

The read-write conflicting variables are divided between threads, with a copy in each thread. Data structures such as linked lists, trees, and hash tables have connected data variables that cannot be divided across threads, making parallelism difficult to implement.

To avoid this, one technique is to keep the complete data structure in a critical section and handle just one action at a time. Another option is to lock the critical section node in use so that other actions do not utilize the same node. As a result, using the critical section assures that the code produces the intended results.

#### **Critical sections in computer networking**

In computer networking, critical sections are also required. Data may not arrive in an ordered fashion when it reaches network sockets. Let's pretend that program X on the computer needs to take data from the socket, reorganize it, and check for any errors. No other application should access the same socket for that data while this program is working on it. As a result, the socket's data is secured by a critical section, allowing program X to use it exclusively.

#### **Kernel-level critical sections**

Critical section often blocks thread and process migration across processors, as well as interruptions and other processes and threads from preempting processes and threads. Nesting is common in critical parts. Multiple essential parts can be accessed and exited at little cost because of nesting.

Because the critical sections may only operate on the processor into which they are inserted, synchronization is only needed inside the executing processor. This enables important portions to be entered and exited at virtually no cost. There is no need for inter-processor synchronization. All that is required is instruction stream synchronization. Most CPUs offer the necessary synchronization simply by stopping the current execution state.

The usage of critical sections as a long-term locking primitive is not recommended. The software lockout problem is based on kernel-level critical sections.

## **Frequently Asked Questions**

### How to solve critical section problems using the bakery algorithm?

To explain in a few words, each process is assigned a number (which may or may not be unique), with the lowest number being served first.

### What are the two approaches to handling critical section problems?

Preemptive and non-preemptive kernels. A preemptive kernel permits a process to be preempted while in kernel mode. A non-preemptive kernel does not allow a kernel-mode process to be preempted. A kernel-mode process will continue to run until it departs kernel mode, blocks, or willingly surrenders CPU control.

### What are the primary requirements of Synchronizing mechanisms?

Mutual exclusion and Progress are included in the primary requirements to be accomplished necessarily. While the bounded wait is considered as an optional requirement.

## Conclusion

To summarize the article, we discussed every aspect related to the critical section problem in the execution of the process. We discussed various solutions to the critical section problem and its uses also. We learned the three criteria related to process synchronization. But the knowledge never stops, so to better understand the operating systems, you can go through many articles on our platform.