Process Synchronization

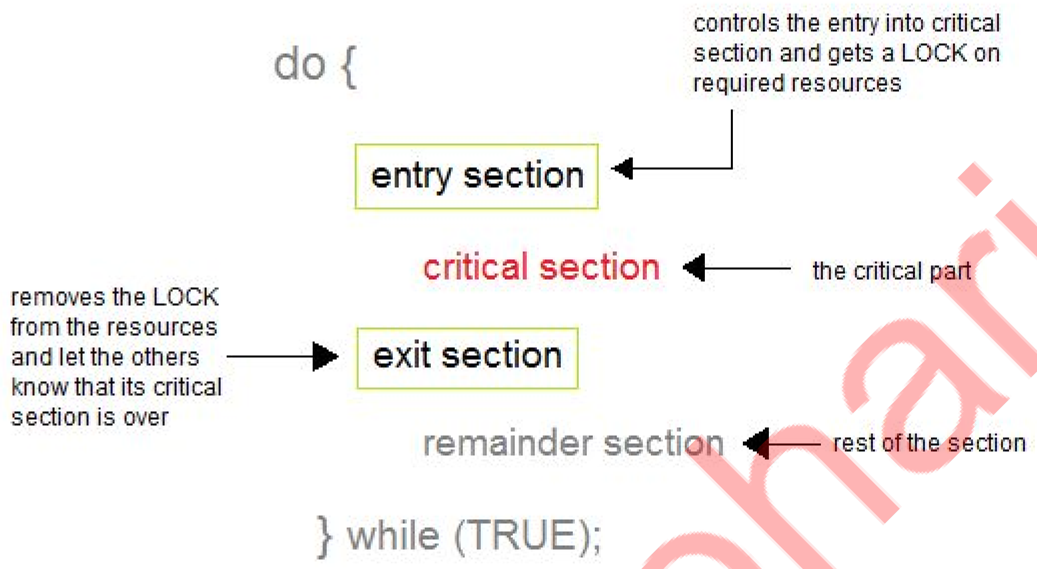
- Process Synchronization means sharing system resources by processes in such a way that Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.
- On the basis of synchronization, processes are categorized as one of the following two types:
  - Independent Process : Execution of one process does not affect the execution of other processes.
  - Cooperative Process : Execution of one process affects the execution of other processes.
- Process synchronization problem arises in the case of Cooperative processes also because resources are shared in Cooperative processes.
- Process Synchronization was introduced to handle problems that arose while multiple process executions such as Critical Section Problem.

A **race condition** is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.
A simple example of a race condition is a light switch. In some homes there are multiple light switches connected to a common ceiling light. When these types of circuits are used, the switch position becomes irrelevant. If the light is on, moving either switch from its current position turns the light off. Similarly, if the light is off, then moving either switch from its current position turns the light on. With that in mind, imagine what might happen if two people tried to turn on the light using two different switches at exactly the same time. One instruction might cancel the other or the two actions might trip the circuit breaker.

- Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.

Critical Section Problem

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.


do {
controls the entry into critical section and gets a LOCK on required resources
entry section
critical section
the critical part
removes the LOCK from the resources and let the others know that its critical section is over
exit section
remainder section
rest of the section
} while (TRUE);

* In the entry section, the process requests for entry in the **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.