Notes of Unit-3

Race Condition in Operating Systems (OS)

In this tutorial we are going to learn about Race Condition in Operating Systems.

Today, we are going to learn about the most important concept in Operating Systems. The Race Condition is a condition which usually occurs in Multi Threading concept which occurs in an Operating System.

The Race Condition usually occurs at the case of Tube Light which has multiple switches. This Tube Light with multiple switches is the biggest example for the Race Condition which has occurred in Operating Systems.

The Race Condition also occurs in the case of processes also. If we do not take care of this Race Condition well then we might get stuck in a Deadlock too.

Now, let us understand the basics of Multi Threading first.

Multi Threading in Operating Systems (OS)

Multi Threading is a procedure by which a task is divided into multiple single threads. These threads are executed one by one in a sequential or non sequential manner to complete the task.

The same process or job can be carried out by a number of threads in multithreading where threads does the same process in other tasks where we have requirement of such kind of threads for task completion, or we can say that the task is being carried out by more than one thread. It is possible to multi tasked by using multithreading.

There are three types of Threads in Multi Threading Concept. They are:

1. Many to Many Multi Threading style
2. Many to One Multi Threading style

3. One to One Multi Threading style

Race Condition in Operating Systems (OS)

Race Condition in Multi Threading Scenario

A race condition is a situation that develops when many threads share a resource or execute the same piece of code in a multithreaded context. Inappropriate handling of this might result in an unfavorable scenario where the output state depends on the threads execution order.

Race Condition in Multi Processing Scenario

The Race Condition is a situation that is developed when a device or system tries to do two or more operations simultaneously when, due to the nature of the device or system, the actions must be performed in the right order to be performed successfully; a race condition is an unpleasant circumstance that results.

The most frequent associations of race with programming and computer science are these. When two computer program processes try to access the same resource simultaneously, they happen and disrupt the system.

Race Condition in Critical Section Area Problem

A race condition is a potential scenario that might happen within a critical section area. This occurs when different results from the execution of numerous threads in a crucial region are obtained depending on the execution order of the threads.

If the critical section area is regarded as an atomic instruction, race conditions in certain areas can be avoided. Race problems can also be avoided by employing locks or atomic variables to properly synchronize threads.

Race Condition is Inter Disciplinary approach. It can occur in both the concepts of Multi Threading, Process Execution, and Critical Section Area too.

Examples of Race Condition:

Example 1: Tube Light ON and OFF

The Race Condition usually occurs at the case of Tube Light which has multiple switches. This Tube Light with multiple switches is the biggest example for the Race Condition which has occurred in Operating Systems.

Explanation

Consider a Tube Light, two switches. Let the two switches are connected to the Tube Light and the Tube Light is in OFF State. Here, if on switch 1 the tube light gets switched on. Then, if we on the switch 2 when switch 1 is in ON State, the Tube Light get switched off. But, both the switched present are in ON State and the Tube Light is in OFF State.

Now, let us consider that the Tube Light is in OFF State and switch 1 and switch 2 are in OFF State. Now, if we turn ON both the switches at a time then the Tube Light would be ON State only. This is because of circuit breaker. One of the switch actions is tripped by the circuit breaker present in the circuit. This is to prevent the functioning of switch going irrelevant.

What if same condition is occurred in the computer? Here, the ON and OFF Conditions gets replaced by Read and Write Operations. Here, the Tube Light is replaced with a computer. Just imagine what happens if we are re writing the data on the computer while the old data is being read. Because of this state, these conditions might occur:

1. Errors and Faults in Newly written data
2. Errors and Faults in Old written data
3. The Computer might gets crashed
4. The Data Corruption might occur.
5. The Data Stored might not be in order.

Race conditions result in inconsistent output and degrade our application's endurance and confidence.

In operating systems, concurrency is achieved via threads. The capacity to carry out many operations concurrently is known as concurrency. Concurrency is achieved in the OS via threads. We may encounter

circumstances where the threads processing the shared data provide different results each time if many threads access shared data without being synced with one another.

Race Conditions Identification or Detection in Operating Systems (OS)

The need for finding the Race Conditions is very important. If we fail to identify them, then we are going to lose so much data and data already present is also going to go corrupt. So, it is very important for the user and the computer to find out the occurrence of Race Condition in the Operating Systems (OS).

Finding and detecting racial conditions is thought to be challenging. They are a semantic issue that might result from several potential coding errors. It is preferable to write code that avoids these issues from the outset.

Tools for static and dynamic analysis are used by programmers to find race conditions. Without starting the software, static testing tools scan everything. However, they tends to generate a lot of inaccurate reports. Although dynamic analysis methods provide fewer false positives, they could miss race conditions that don't occur within the program itself.

Data races, which happen when two threads simultaneously target the same memory region and at least one of them performs a write operation, can occasionally result in race conditions. Data races are simpler to spot than race conditions since they need particular circumstances to manifest. Data race scenarios are kept an eye out for by tools like the Data Race Detector from the Go Project. Race situations provide more significant issues and are more strongly related to application semantics.

Race Condition in Producer and Consumer Problem

The Race Condition usually occurs in the case of Producer and Consumer Problem. This Producer and Consumer Problem is the biggest example for the Race Condition which has occurred in Operating Systems.

Let us know what is Producer and Consumer Problem.

Producer and Consumer Problem

The Producer and Consumer problem is another name for Bound Buffer Problem. In this issue, there are n slots in a buffer, and each slot may hold one data unit. Producer and Consumer are the two operations that are using the buffer. The Producer and Consumer problem comes under the category of Classical Problems of Synchronization.

Here, Producer tries to create new data or may change the existing data.

The Consumer tries to read the data only. It does not try to alter the already present data.

There are two cases in this Producer and Consumer Problem. Let us explain these two cases with the help of Producer Threads and Consumer Threads.

The two cases are:

1. The Consumer Threads are working faster than Producer Threads

Here, in this scenario there would be fast reading of data, but the change in data as expected by the computer is not achieved.

Example:

We have just deposited an amount of Two Thousand Two Hundred and Twenty Two Rupees (2, 222) in out Bank Account today. We wanted to draw Two Thousand Rupees (2, 000) from the bank on the same day after six hours of depositing money. Our Previous Balance before depositing Two Thousand Two Hundred and Twenty Two Rupees (2, 222) was Fifteen Hundred Rupees.

Now, as the Producer Threads are slow, the Account Balance was not updated to Three Thousand Seven Hundred and Twenty Two Rupees (1500 + 2222 = 3722).

The Account Balance is Fifteen Hundred only now. So, if we draw money the cash would not be updated and we cannot draw Two Thousand from the bank. Because of The Consumer Threads are working faster than Producer Threads Scenario we are facing this problem.

2. The Producer Threads are working faster than Consumer Threads

Here, in this scenario we are reading the data in the computer, suddenly the values of the data gets changed during the process of reading of data.