**Basic Problems in Operating System and Programming**

**Race Condition:** A race condition occurs when two processes or threads tend to acquire the same resources at a same time and it causes a problem in the system.

**For example** In computer memory or storage, a race condition may occur if commands to read and write a large amount of data are received at almost the same instant, and the machine attempts to overwrite some or all of the old data while that old data is still being read. The result may be one or more of the following:

* the computer crashes or identifies an illegal operation of the program
* errors reading the old data
* errors writing the new data

A race condition can also occur if instructions are processed in the incorrect order.

There are a few types of race conditions. Two categories that define the impact of the race condition on a system are referred to as critical and noncritical:

* A **critical** race condition will cause the end state of the device, system or program to change. For example, if flipping two light switches connected to a common light at the same time blows the circuit, it is considered a critical race condition. In software, a critical race condition is when a situation results in a bug with unpredictable or undefined behavior.
* A **noncritical**race condition does not directly affect the end state of the system, device or program. In the light example, if the light is off and flipping both switches simultaneously turns the light on and has the same effect as flipping one switch, then it is a noncritical race condition. In software, a noncritical race condition does not result in a bug.

There are two common types of race conditions

* Read-modify-write
* Check-then-act

There are two types of testing tools used to identify race condition

Static Testing Tools identify race condition without running the program that’s why they can produce false reports.

Dynamic Testing Tools identify race condition identify race condition by running the program and they produce fewer false reports by they cannot catch the race conditions that are directly executed in the program.

Two ways programmers can prevent race conditions in operating systems and other software include:

* **Avoid shared states.** This means reviewing code to ensure when shared resources are part of a system or process, atomic operations are in place that run independently of other processes and locking is used to enforce the atomic operation of critical sections of code. Immutable objects can also be used that cannot be altered once created.
* **Use thread** **synchronization.**Here, a given part of the program can only execute one thread at a time.

Preventing race conditions with other types of technology is also possible:

**Storage and memory**

The serialization of memory or storage access will also prevent race conditions. This means if read and write commands are received close together, the read command is executed and completed first by default.

**Networking**

In a network, a race condition may occur if two users try to access a [channel](https://www.techtarget.com/searchdatacenter/definition/channel) at the same instant and neither computer receives notification the channel is occupied before the system grants access. Statistically, this kind of situation occurs mostly in networks with long lag times, such as those that use [geostationary satellites](https://www.techtarget.com/searchmobilecomputing/definition/geostationary-satellite).

To prevent such a race condition, a priority scheme must be devised to give one user exclusive access. For example, the subscriber whose username or number begins with the earlier letter of the alphabet or the lower numeral may get priority when two subscribers attempt to access the system within a prescribed increment of time.