# Difference Between Multitasking and Multithreading

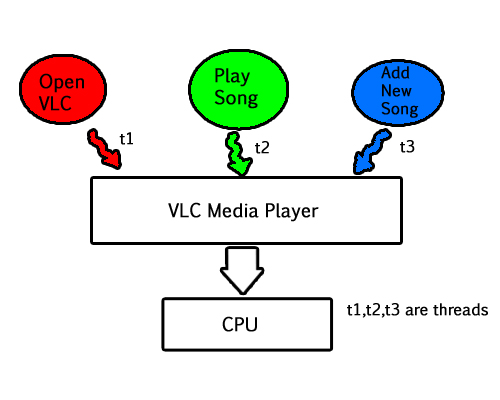
# In this article, we will discuss the differences between Multitasking and Multithreading in Rust. People generally get confused between these terms. We will discuss them in detail.

### What is Multi****threading**** Operating System?

A thread is a basic unit of CPU utilization. Multi threading is an execution model that allows a single process to have multiple code segments (i.e., threads) running concurrently within the “context” of that process.

e.g. VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist.

The image below completely describes the VLC player example:



**Multi threading system’s working –**

* Say there is a web server which processes client requests. Now if it executes as a single threaded process, then it will not be able to process multiple requests at a time. Firstly one client will make its request and finish its execution and only then, the server will be able to process another client request. This is really costly, time consuming and tiring task. To avoid this, multi threading can be made use of.
* Now, whenever a new client request comes in, the web server simply creates a new thread for processing this request and resumes its execution to hear more client requests. So the web server has the task of listening to new client requests and creating threads for each individual request. Each newly created thread processes one client request, thus reducing the burden on web server.

**Multi threading in Rust**

* To create a new thread, we call the thread::spawn function.
* The calls to thread::sleep force a thread to stop its execution for a short duration, allowing a different thread to run
* Join function is for execution of download without any interrupt.

the main calls function of get\_two\_sites\_async, which calls other two functions names as download\_asyn1 and download async\_2 which download 2 files and store them in variable of future\_one and future\_two.Output store in function of get\_two\_sites\_async and block\_on in main display the results.

Filename: src/main.rs

****

**Output :**

Important thing to notice is that Bar thread printed first, even though the print statement from the foo thread appears first in the code. Because download\_asyn1 associated with delay so as thread run multiple task at a time so it complete execution of bar first and display it result and then foo.

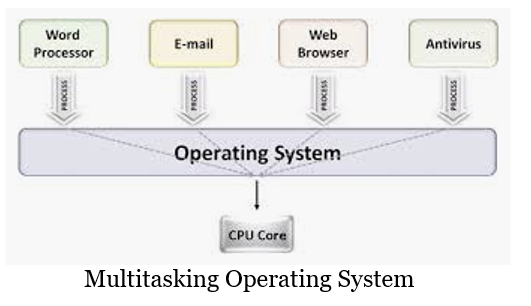
****

## What is Multitasking Operating System?

When any program is running it is referred to as a task. In a **Multitasking Operating System**, two or more tasks are active simultaneously.

In this, we stop one task temporarily to work one another. It also means that the computer can work with more than one program at a time.

For instance, you can fetch information from one database on the screen analyzing data, while the computer is sorting data from another database, on the other hand performing calculations on a separate worksheet.



Another example is, In modern operating systems, we can play MP3 music, edit documents in Microsoft Word, surfing the Google Chrome all are simultaneous.

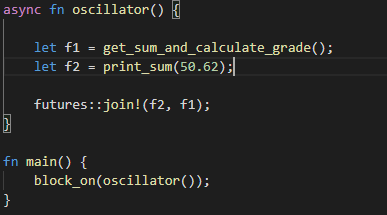
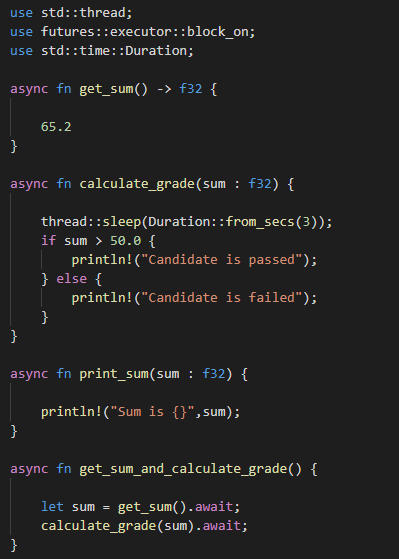
It is not same as "multiple loading" of applications, also referred to as "context switching" or "task switching," In Context switching, several applications can be open, but only one application is working at a time. **Multitasking** based on the time-sharing along with context switching.

**Multi tasking in Rust**

In a typical threaded application, if you wanted to download two different webpage’s at the same time, you would spread the work across two different threads.

Rust's async/.await notation, which will allow us to run multiple tasks at once without creating multiple threads:

Filename: src/main.rs



**Output :**



Notice that “sum is 50.62” thread printed first, even though the print statement “candidate pass/fail” appears first in the code. This happens because passing and failing of candidate take some time of execution as it is associated with delay so its execution occur after execution of f2 variable in oscillator.

Lets see the difference between two and decide which one is better for our task

| **S.NO** | **MULTITASKING** | **MULTITHREADING** |
| --- | --- | --- |
| 1. | In multitasking, users are allowed to perform many tasks by CPU. | While in multithreading, many threads are created from a process through which computer power is increased. |
| 2. | Multitasking involves often CPU switching between the tasks. | While in multithreading also, CPU switching is often involved between the threads. |
| 3. | In multitasking, the processes share separate memory. | While in multithreading, processes are allocated same memory. |
| 4. | Multitasking component involves multiprocessing. | While multithreading component does not involve multiprocessing. |
| 5. | In multitasking, CPU is provided in order to execute many tasks at a time. | While in multithreading also, CPU is provided in order to execute many threads from a process at a time. |
| 6. | In multitasking, processes don’t share same resources, each process is allocated separate resources. | While in multithreading, each process share same resources. |

### Conclusion

Multitasking is similar to multiprogramming whereas, Multithreading is thread-based multitasking. Multithreading is less costly than multitasking as threads are easy to create then a process but it can lead to problems, such as:

* Race conditions, where threads are accessing data or resources in an inconsistent order
* Deadlocks, where two threads are waiting for each other to finish using a resource the other thread has, preventing both threads from continuing
* Bugs that happen only in certain situations and are hard to reproduce and fix reliably