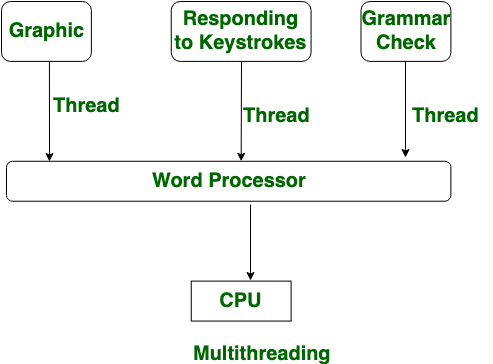
**MULTI-THREADING vs MULTI-TASKING**

Let’s start with the initials, which is introduction:

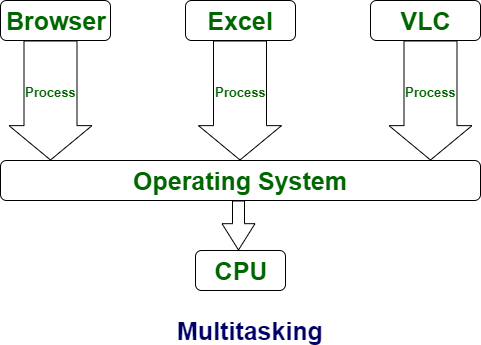
**MULTI-THREADING:**

Multithreading is a procedure in which many threads are produced by the CPU in order to perform task in multiple threads as it will be able to execute the task simultaneously. Multi-threading provides the same memory and resources to the processes for execution.



**MULTI-TASKING:**

Multitasking or Asynchronous programming is a technique to run simultaneous operations in an application on a single OS thread at a time. Multitasking involves often CPU switching between the tasks, so that users can collaborate with each program together. Multi-tasking provides separate memory and resources to the processes for execution.



**COMPARISION**

**MULTI-TASKING**

In multitasking, users are permitted to perform various tasks by CPU in a single OS thread at a time.

Multitasking often provide CPU switching between the tasks, which makes it more simple for user’s to work.

In multitasking, the processes share separate memory.

Multitasking module consist up of multiprocessing.

In multitasking, CPU is provided in order to execute many tasks at a time with in a single thread.

In multitasking, processes don’t share same resources, each process is allocated separate resources for execution.

**MULTI-THREADING**

While in multithreading, many threads are created from a process through which computer power is increased.

While in multithreading also, sometimes CPU switching is often involved between the threads and sometimes user switching is involved.

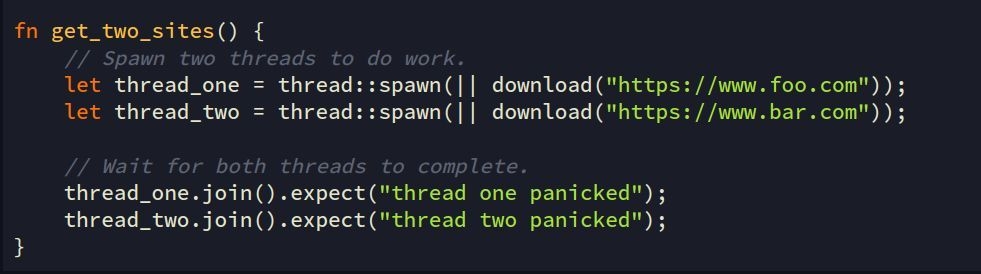
While in multithreading, processes are allocated with same memory.

While multithreading module does not consist of multiprocessing.

While in multithreading, CPU is provided in order to execute many threads from a process at the same time.

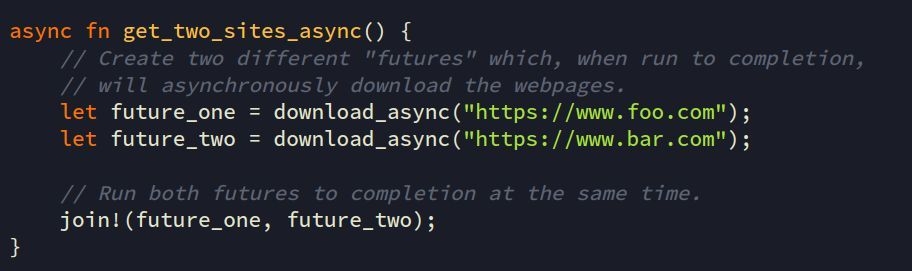
While in multithreading, each process share same resources for execution.

**COMPARISION W.R.T CODE**

**MULTI-THREADING:**

The above code will work fine as the threads are designed to do the same, run multiple tasks at once. However, there will be some restrictions in this code, which includes the overhead in the process of shifting between different threads and sharing of data between threads. Even if a thread is in ideal state it would be consuming some precious system resources.

**MULTI-TASKING:**

These above were the costs that asynchronous code is designed to eliminate. We can rewrite the above function using Rust's “async/.await” notation, which will allow us to run multiple tasks simultaneously without creating multiple threads.

Overall, asynchronous applications have the potential to be faster than multi-threading and use less resources than a corresponding threaded implementation. However, there is a cost. Also asynchronous functions require special support from the language and libraries. In Rust, “async fn” creates an asynchronous function which returns a Future. To execute the body of the function, the returned Future must be run to completion.