Threading

Before discussing the threading concept, let's try to understand the two terms involved while executing a process in an OS-

- **Concurrency-**

  when we perform multiple tasks intermittently (means periodically), we can say that we are doing them concurrently.

  But, at any given moment actually only one task is being done. For example, you can watch TV, read a newspaper, check your smart phone intermittently.

  But at any given point of time, you are actually doing only one task. Similarly, if you are editing two documents one in Word and another is in Excel, then you are doing two tasks intermittently or concurrently.

- **Parallelism-**

  when we perform tasks simultaneously, then we can say that we are doing them parallelly.

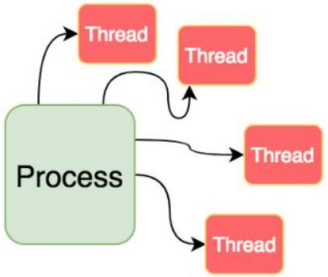
  Means, at any given moment all tasks are being done. For example, while driving your car you can carry out several activities in parallel.

  Like, while driving the car you can listen to the music, you can talk with your co-passengers.

  Another example of Parallelism may be if you are loading 4 or 5 pages in different tabs of your browser, all those pages are being fetched from a web server in parallel. Means, at a given point of time multiple tasks are being executed.

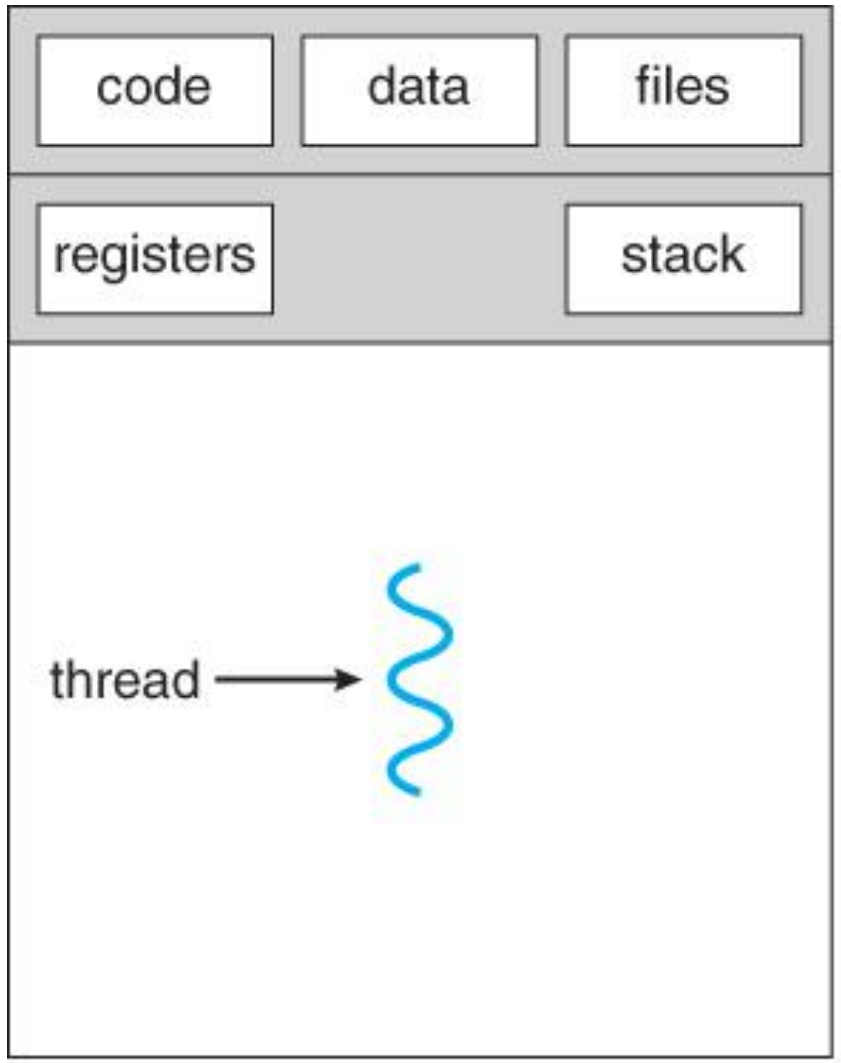
Threads in OS

- A Single Process may have multiple functionalities/ logical units (or parts) that need to be run in parallel.



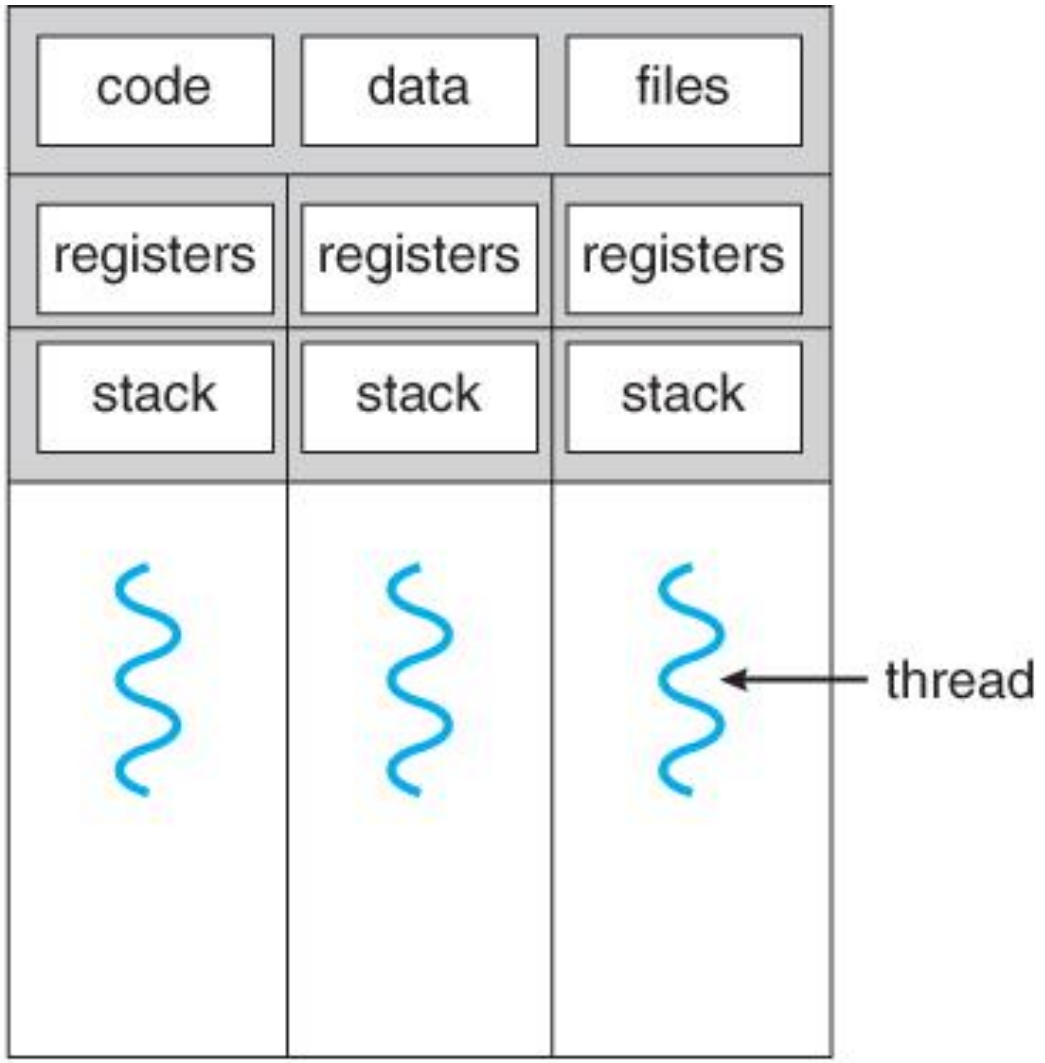
- These logical units are known as **Threads**.
- Each thread has its own set of registers, program counter and stack space.
- There can be multiple threads with same of different functionality in a single process.
- Threads in the operating system provide multiple benefits and improve the overall performance of the system.
- In Multithreading, a single process is divided into multiple threads instead of creating a whole new process. Multithreading is used to achieve parallelism and to improve the performance of the applications as it is faster.
- Through Multithreading, we can achieve Multitasking.

Program -1



single-threaded process

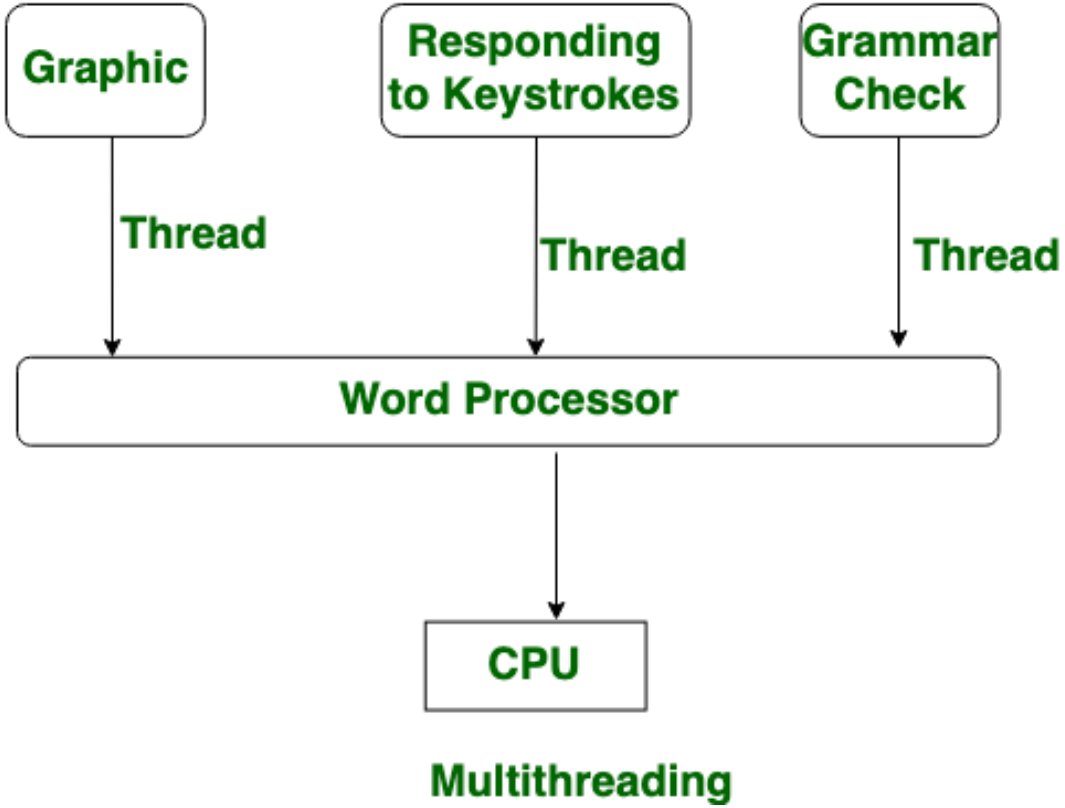
Program -2



multithreaded process

Multithreading

- Multithreading involves two terms- a process and a thread.
- A process is a program in execution.
- A process can be further divided into independent logical units known as threads.
- Multithreading is the capability of OS/CPU to execute multiple tasks simultaneously.



Thread Control Block

- As Thread is an entity within a process that can be scheduled for execution, it contains some information about itself and that information is stored in **Thread Control Block** (TCB).

- A Thread Control Block stores following useful information regarding a thread-

  - **Thread identifier-** Unique id (TID) is assigned to every new thread.

  - **Stack pointer-** Points to the thread's stack in the process and contains the local variables under thread's scope.

  - **Program counter-** It is a register which stores the address of the instruction currently being executed by the thread.

  - **Thread state-** It is the current state of a thread and can be running, ready, waiting, start or done.

Difference b/w Thread & Process

Process	Thread
Processes use more resources and hence they are termed as heavyweight processes.	Threads share resources and hence they are termed as lightweight processes.
Creation and termination times of processes are slower.	Creation and termination times of threads are faster compared to processes.
Processes have their own code and data/file.	Threads share code and data/file within a process.
Communication between processes is slower.	Communication between threads is faster.
Context Switching in processes is slower.	Context switching in threads is faster.
Processes are independent of each other.	Threads, on the other hand, are interdependent. (i.e they can read, write or change another thread's data)
Eg: Opening two different browsers.	Eg: Opening two tabs in the same browser.

Types of Threads

User-level thread-

User-level threads are implemented and managed by the user and the kernel is not aware of it.

- User-level threads are **implemented using user-level libraries and the OS does not recognize these threads**.
- User-level thread is **faster to create and manage compared to kernel-level thread**.
- **Context switching in user-level threads is faster**.
- If one user-level thread performs a blocking operation then the entire process gets blocked. Eg: POSIX threads, Java threads, etc.

- **Kernel-level thread-**

- **Kernel level threads are implemented and managed by the OS**.

- Kernel level threads are **implemented using system calls and Kernel level threads are recognized by the OS**.

- Kernel-level threads are **slower to create and manage compared to user-level threads**.

- **Context switching in a kernel-level thread is slower**.

- Even if one kernel-level thread performs a blocking operation, it does not affect other threads. Eg: **Window Solaris**.