**LAB # 5**

**Parallel Processing**

**Introduction:**

Parallel Processing is the simultaneous execution of a task through multiprocessors attached to the same computer. It involves **taking a process, dividing it into several smaller processes, and then working on each of those simultaneously**. **The** goal of this divide-and-conquer approach is to complete the larger task in less time than it would have taken to do it in one large chunk. In python, the multiprocessing module is used to run independent parallel processes.

By the end of this lab, you’ll be able to:

* Structure the code and enable parallel processing to parallelize any typical logic using python’s multiprocessing.
* Implement synchronous and asynchronous parallel processing.

Before moving on to the lab tasks, let’s revise some basic concepts.

**There are two main ways to handle parallel programs:**

**Shared Memory:**

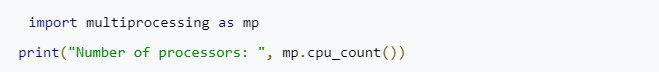
All the processors can access all memory locations and can read or write shared variables. The advantage is that you don’t need to handle the communication explicitly. But here a problem arises when multiple processors access and change the same memory location at the same time. To avoid this conflict, synchronization techniques are used.

**Distributed memory**

In distributed memory, each process is totally separated and has its own memory space. In this scenario, communication is handled explicitly between the processes, so it is costlier compared to shared memory.

**Multiprocessing for parallel processing**

Using the standard multiprocessing module, we can efficiently parallelize simple tasks.

Maximum parallel processes you can run is limited by the number of processors in your computer.

## Synchronous Execution: A synchronous execution is one in which the processes are completed in the same order in which it was started. It means that the first task in a program must finish processing before moving on to executing the next task. This is achieved by locking the main program until the respective processes are finished.

## Asynchronous Execution: When you run something asynchronously it means it is non-blocking, you execute it without waiting for it to complete and carry on with other things. As a result, the order of results can get mixed up but usually gets done quicker.

## Python multiprocessing module provides pool class and process class.

**Process class:**

## There are two important functions that belongs to the Process class – start() and join() function.

## Start():

## To start a process, we use start method of Process class.

## Join():

## The join method blocks the execution of the main process until the process whose join method is called terminates.

## First we need to instantiate a process object. Process can be instantiated with or without arguments. If we want to pass any argument through the process use args keyword.

## 

When a process object will be created, nothing will happen until we tell it to start processing via start() function.

## The process will then run and return its results.

## After that we tell the process to complete via join() function.

## (So if you create many processes and don’t terminate them, you may face scarcity of resources.)

## Let’s implement the following code to understand the Process class.

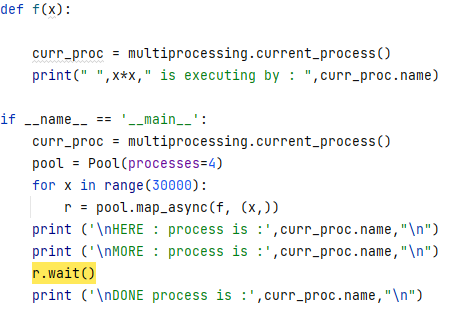
**Pool class**

The multiprocessing.Pool() class produces a set of processes called workers and can submit tasks using methods given in the table below.

|  |  |
| --- | --- |
| **Synchronous Execution** | **Asynchronous Execution** |
| apply() | apply\_async() |
| map() | map\_async() |
| starmap() | starmap\_async() |

For parallel mapping, you should first initialize a multiprocessing.Pool() object. The first argument is the number of workers; if not given, that number will be equal to the number of cores in the system.

## apply ():

**apply\_async ():**

**TASK :**

* Check the same code for the remaining methods i.e. map/map\_async and starmap/starmap\_async.
* Also note down the execution time (for all methods)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**