Experiment No. 10
Demonstrate the concept of Multi-threading
Date of Performance:
Date of Submission:

Experiment No. 10

Title: Demonstrate the concept of Multi-threading

Aim: To study and implement the concept of Multi-threading

Objective: To introduce the concept of Multi-threading in python

Theory:

Thread

In computing, a **process** is an instance of a computer program that is being executed. Any process has 3 basic components:

- An executable program.
- The associated data needed by the program (variables, work space, buffers, etc.)
- The execution context of the program (State of process)

A **thread** is an entity within a process that can be scheduled for execution. Also, it is the smallest unit of processing that can be performed in an OS (Operating System).

In simple words, a **thread** is a sequence of such instructions within a program that can be executed independently of other code. For simplicity, you can assume that a thread is simply a subset of a process!

A thread contains all this information in a **Thread Control Block (TCB)**:

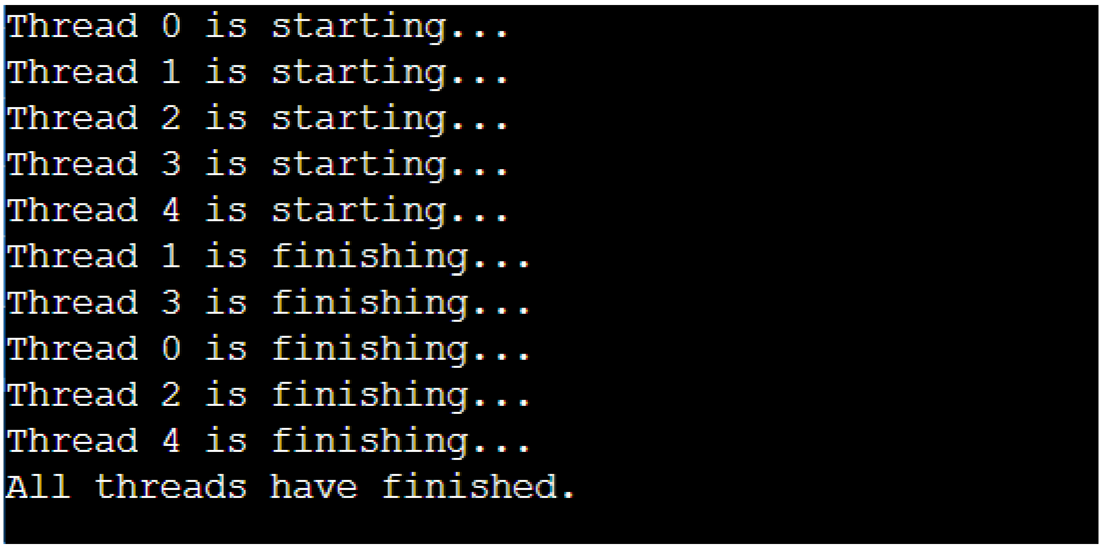
- **Thread Identifier:** Unique id (TID) is assigned to every new thread
- **Stack pointer:** Points to thread's stack in the process. Stack contains the local variables under thread's scope.
- **Program counter:** a register which stores the address of the instruction currently being executed by thread.
- **Thread state:** can be running, ready, waiting, start or done.
- **Thread's register set:** registers assigned to thread for computations.

- **Parent process Pointer:** A pointer to the Process control block (PCB) of the process that the thread lives on.

Code:

```
import threading

import time

def task(name):

    print(f"Thread {name} is starting...")

    time.sleep(2)

    print(f"Thread {name} is finishing...")

threads = []

for i in range(5):

    t = threading.Thread(target=task, args=(i,))

    threads.append(t)

    t.start()

for t in threads:

    t.join()

print("All threads have finished.")
```

Output:



```
Thread 0 is starting...
Thread 1 is starting...
Thread 2 is starting...
Thread 3 is starting...
Thread 4 is starting...
Thread 1 is finishing...
Thread 3 is finishing...
Thread 0 is finishing...
Thread 2 is finishing...
Thread 4 is finishing...
All threads have finished.
```

Conclusion: In this example, we've demonstrated the concept of multi-threading in Python using the threading module. By creating multiple threads, each executing a separate task concurrently, we can improve the overall performance of our program by leveraging the available CPU resources more effectively. However, it's important to handle synchronization and communication between threads carefully to avoid potential issues such as race conditions or deadlocks.