Experiment No. 12
Demonstrate the concept of Multi-threading
Date of Performance:
Date of Submission:

Experiment No. 12

Title: Demonstrate the concept of Multi-threading

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

Objective: To introduce the concept of Multithreading 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


# Define a function to be executed by each thread

def print_numbers():

    for i in range(5):

        print(f"{threading.current_thread().name}: {i}")
    time.sleep(1)

thread1 = threading.Thread(target=print_numbers, name="Thread 1")
thread2 = threading.Thread(target=print_numbers, name="Thread 2")

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print("Both threads have finished execution.")
```

Output:

```
Thread 1: 0
Thread 2: 0
Thread 2: 1
Thread 1: 1
Thread 2: 2
Thread 1: 2
Thread 2: 3
Thread 1: 3
Thread 2: 4
Thread 1: 4
Both threads have finished execution.
```

Conclusion: The Python program demonstrates multi-threading using the threading module. Two threads are created to execute a function concurrently. By starting both threads and using join(), we ensure that the main program waits for their completion. This showcases the concurrent execution capability of multi-threading.