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 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!

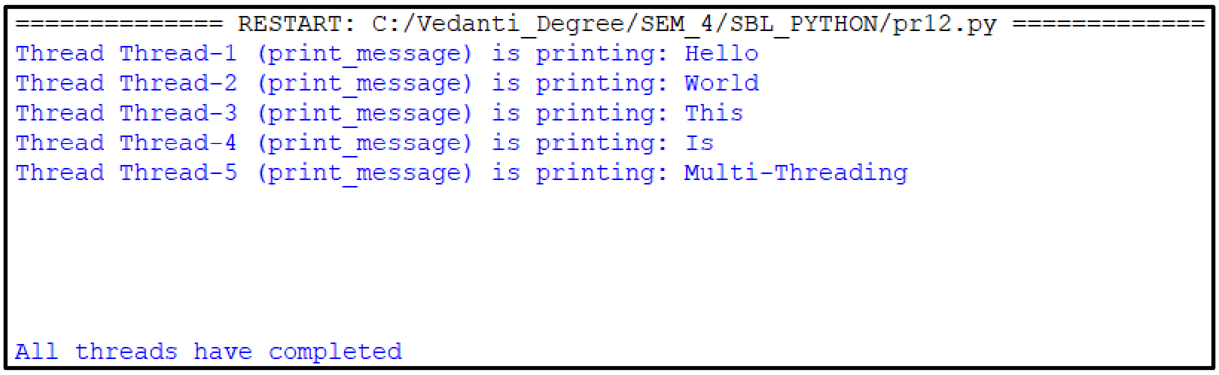
A thread contains all this information in a **Thread Control Block (TCB)**:

- **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 print_message(message, delay):
   print(f"Thread {threading.current_thread().name} is printing: {message}")
   time.sleep(delay)

messages = [
   ("Hello\n", 2),
   ("World\n", 3),
   ("This\n", 1),
    ("Is\n", 4),
    ("Multi-Threading\n", 2)
]

threads = []
for message, delay in messages:
    thread = threading.Thread(target=print_message, args=(message, delay))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()

print("All threads have completed")
```

OUTPUT:



```
============== RESTART: C:/Vedanti_Degree/SEM_4/SBL_PYTHON/pr12.py =============
Thread Thread-1 (print_message) is printing: Hello
Thread Thread-2 (print_message) is printing: World
Thread Thread-3 (print_message) is printing: This
Thread Thread-4 (print_message) is printing: Is
Thread Thread-5 (print_message) is printing: Multi-Threading





All threads have completed
```

CONCLUSION:

In conclusion, the implementation of multi-threading showcases the power and efficiency of concurrent execution in software development. By leveraging multiple threads, tasks can be executed simultaneously, thereby improving performance and responsiveness of applications, especially in scenarios where tasks can be parallelized. Through this program, we have witnessed how multi-threading enables tasks to execute concurrently, utilizing the available resources efficiently.