**Tutorial No. 7**

**Title: Program on Multithreading using Python**

**Batch: B1 Roll No: 1914078 Tutorial No.:7**

### Aim: Program on implementation of multithreading in Python

**Resources needed:** Python IDE

### Theory:

### What is 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)

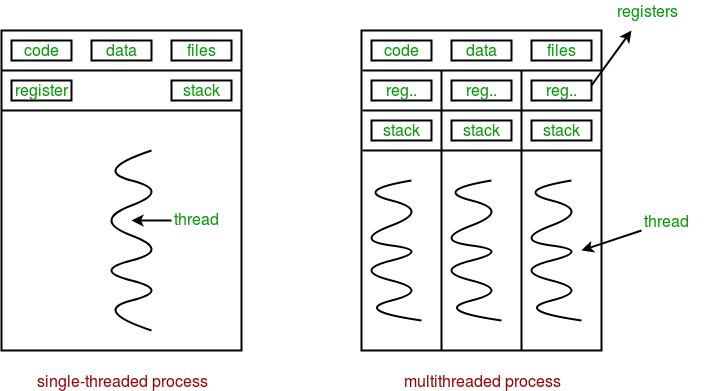
### A thread is an entity within a process that can be scheduled for execution independently. Also, it is the smallest unit of processing that can be performed in an OS (Operating System).

**What is Multithreading?**

Multiple threads can exist within one process where:

* Each thread contains its own **register set** and **local variables (stored in stack)**.
* All thread of a process share **global variables (stored in heap)** and the **program code**.

Consider the diagram below to understand how multiple threads exist in memory:



**Multithreading** is defined as the ability of a processor to execute multiple threads concurrently.

**Multithreading in Python**

In Python, the **threading** module provides a very simple and intuitive API for spawning multiple threads in a program.

# Python program to illustrate the concept of threading

# importing the threading module

import threading

def print\_cube(num):

"""

function to print cube of given num

"""

if \_\_name\_\_ == "\_\_main\_\_":

# creating thread

t1 = threading.Thread(target=print\_square, args=(10,))

# starting thread 1

t1.start()

# wait until thread 1 is completely executed

t1.join()

# both threads completely executed

print("Done!")

**Creating a new thread and related methods: Using object of Thread class from threading module.**

To create a new thread, we create an object of Thread class. It takes following arguments:

target: the function to be executed by thread

args: the arguments to be passed to the target function

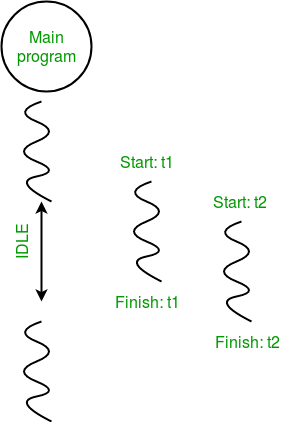
Once the threads start, the current program (you can think of it like a main thread) also keeps on executing. In order to stop execution of current program until a thread is complete, we use join method.

t1.join()

t2.join()

As a result, the current program will first wait for the completion of t1 and then t2. Once, they are finished, the remaining statements of current program are executed..

Diagram below depicts actual process of execution.



Threading.current\_thread().name this will print current thread’s name and threading.main\_thread().name will print main thread’s name. **os.getpid()** function to get ID of current process.

Thread synchronization is defined as a mechanism which ensures that two or more concurrent threads do not simultaneously execute some particular program segment. Concurrent accesses to shared resource can lead to race condition.

A race condition occurs when two or more threads can access shared data and they try to change it at the same time. As a result, the values of variables may be unpredictable and vary depending on the timings of context switches of the processes.

For example two threads trying to increment shared variable value may read same initial values and produce wrong result.

Hence there is requirement of acquiring lock on shared resource before use. **threading** module provides a **Lock** class to deal with the race conditions.

Lock class provides following methods:

**acquire([blocking]) :** To acquire a lock. A lock can be blocking or non-blocking.

When invoked with the blocking argument set to True (the default), thread execution is blocked until the lock is unlocked, then lock is set to locked and return True.

When invoked with the blocking argument set to False, thread execution is not blocked. If lock is unlocked, then set it to locked and return True else return False immediately.

**release() :** To release a lock.

When the lock is locked, reset it to unlocked, and return. If any other threads are blocked waiting for the lock to become unlocked, allow exactly one of them to proceed.

If lock is already unlocked, a ThreadError is raised.

Firstly, a Lock object is created using:

lock = threading.Lock()

Then, lock is passed as target function argument:

t1 = threading.Thread(target=thread\_task, args=(lock,))

t2 = threading.Thread(target=thread\_task, args=(lock,))

In the critical section of target function, we apply lock using lock.acquire() method. As soon as a lock is acquired, no other thread can access the critical section (here, increment function) until the lock is released using lock.release() method.

### Activities:

1. Write a program to create three threads. Thread 1 will generate random number. Thread two will compute square of this number and thread 3 will computer cube of this number

### Result: (script and output)

### Script:

import threading

import random

def print\_random(x, y):

    return (random.randint(x, y))

def print\_cube(x, y):

    num = print\_random(x, y)

    print("The Cube of {}: {}".format(num, num \* num \* num))

def print\_square(x, y):

    num = print\_random(x, y)

    print("The Square of {}: {}".format(num, num \* num))

if \_\_name\_\_ == "\_\_main\_\_":

    t0 = threading.Thread(target=print\_random, args=(0, 9))

    t1 = threading.Thread(target=print\_square, args=(0, 9))

    t2 = threading.Thread(target=print\_cube, args=(0, 9))

    t0.start()

    t1.start()

    t2.start()

    t0.join()

    t1.join()

    t2.join()

### Output:

### 

### Outcomes: Apply Object Oriented Programming Concepts in Python

### Questions:

### a)What are other ways to create threads in python? Give examples.

### Ans)

### 1.Creating Threads using Class

### import threading

### class thread(threading.Thread):

### def \_\_init\_\_(self, thread\_name, thread\_ID):

### threading.Thread.\_\_init\_\_(self)

### self.thread\_name = thread\_name

### self.thread\_ID = thread\_ID

### # helper function to execute the threads

### def run(self):

### print(str(self.thread\_name) +" "+ str(self.thread\_ID));

### thread1 = thread("GFG", 1000)

### thread2 = thread("GeeksforGeeks", 2000);

### thread1.start()

### thread2.start()

### print("Exit")

### 2.Creating Threads using Functions

### from threading import Thread

### from time import sleep

### # function to create threads

### def threaded\_function(arg):

### for i in range(arg):

### print("running")

### 

### # wait 1 sec in between each thread

### sleep(1)

### if \_\_name\_\_ == "\_\_main\_\_":

### thread = Thread(target = threaded\_function, args = (10, ))

### thread.start()

### thread.join()

### print("thread finished...exiting")

### b) How wait() and notify() methods can be used with lock in thread?

### Ans: The wait() method releases the lock, and then blocks until it is awakened by a notify() or notifyAll() call for the same condition variable in another thread. Once awakened, it re-acquires the lock and returns. It is also possible to specify a timeout.

**Conclusion: (Conclusion to be based on the objectives and outcomes achieved)**

understood and implemented the activity based on threading

**References:**

* 1. **Reema Thareja , “Python Programming: Using Problem Solving Approach”, Oxford University Press, First Edition 2017, India**
  2. **Sheetal Taneja and Naveen Kumar,” Python Programing: A Modular Approach”, Pearson India, Second Edition 2018, India**