Multi Threading (threading module)

Applications are two types

1. Single tasking applications
2. Multitasking applications

Single Task application

An application which allows performs one operation or task is called single task application.

Multitasking applications

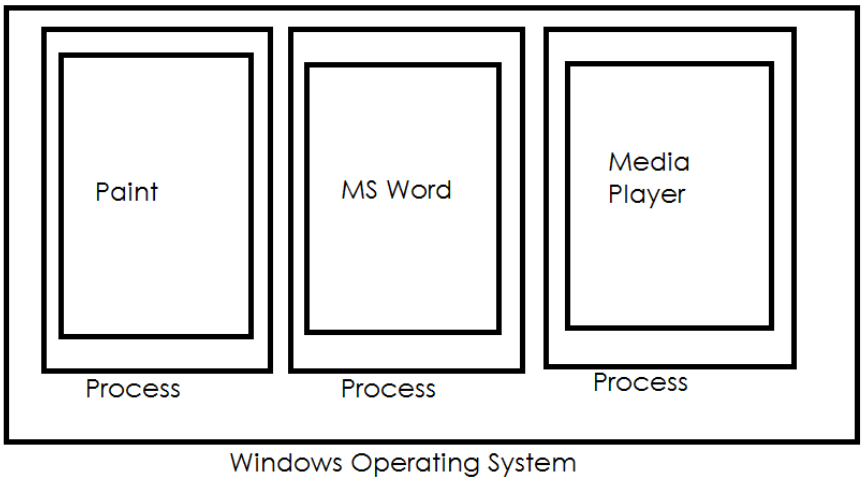
An application which allows performs more than one operation concurrently or simultaneously is called multitasking application.
Multitasking applications are two types

1. Process based multitasking
2. Thread based multitasking

Process based multitasking

What is process?

A process is instance of program (OR) program under execution is process.
Simultaneous execution of more than one program is called process based multitasking.



Thread based multitasking

What is thread?
Thread is an independent of part/path of execution within program. Thread performs operation independent of other operations. Simultaneous execution of more than one thread is called threading based multitasking.
Thread is an instance of process or child process
Thread is process under execution.

Threads execution simultaneously by scheduling, this scheduling is done using thread schedulers provided PVM (Python Virtual Machine).

In order work with threads python provided a module called “threading”.

How to create a thread?
Thread can be created in two ways,

1. By inheriting Thread class
2. Using thread callable object

Threading module provides a predefined class “Thread”, using this class we can create thread object.

class threading.Thread(group=None, **target**=None, name=None, args=(), kwargs={})

Using thread callable object
In this approach we create a thread object by defining target. Target is **function** or callable object. This function is used by thread to perform operation.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

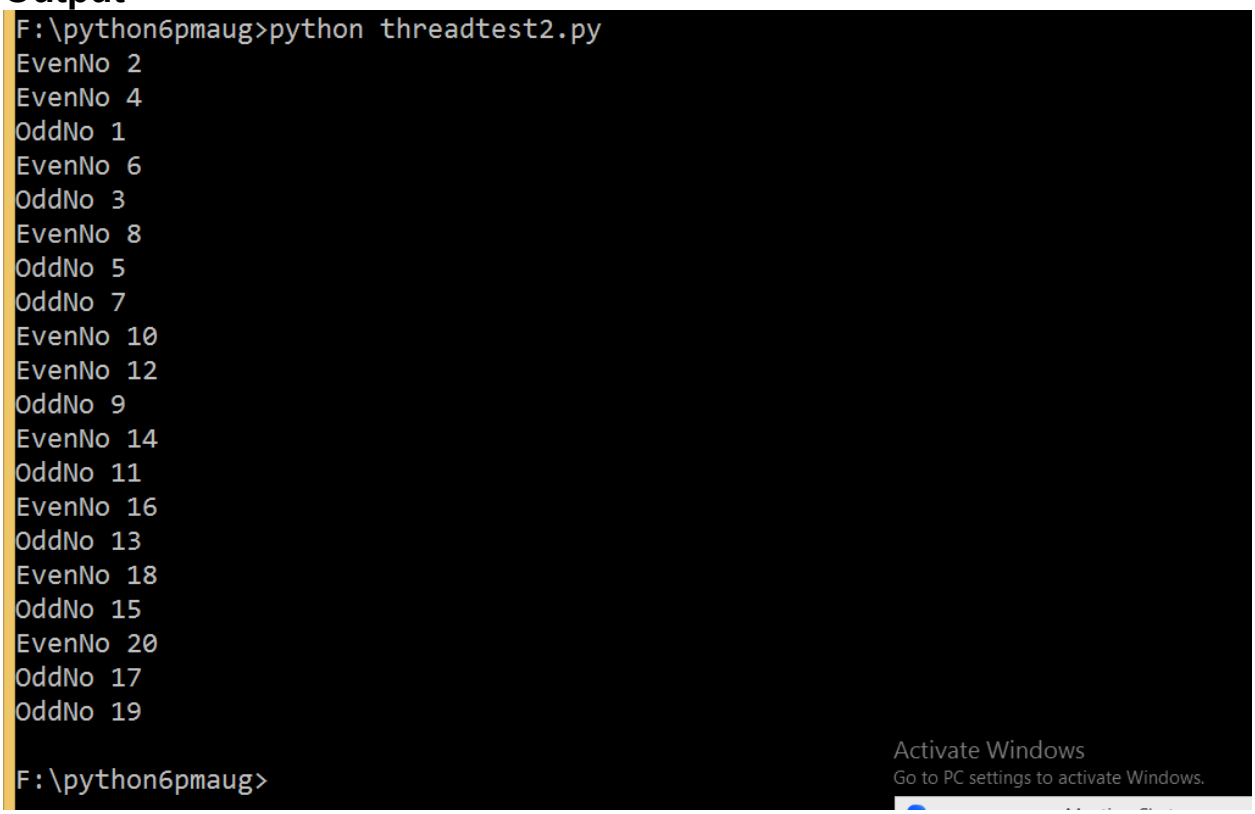
Example:

```
import threading
def even():
    for num in range(1,21):
        if num%2==0:
            print(f'EvenNo {num}')
def odd():
    for num in range(1,21):
```

```
        if num%2!=0:
            print(f'OddNo {num}')

def main():
    t1=threading.Thread(target=even)
    t2=threading.Thread(target=odd)
    t1.start()
    t2.start()
main()
```

Output



Thread is executed by calling or invoking start() method. By start() method, thread is given to scheduler, which scheduler execution of thread.
Thread run() method calls target(function/callable object).

Example:

```
import threading
def even(start,stop):
    for num in range(start,stop):
        if num%2==0:
```

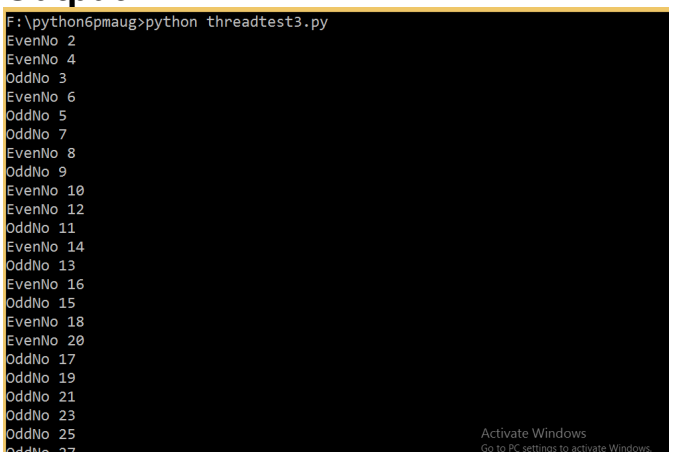
```
            print(f'EvenNo {num}')

def odd(start,stop):
    for num in range(start,stop):
        if num%2!=0:
            print(f'OddNo {num}')


def main():
    t1=threading.Thread(target=even,args=(2,21))
    t2=threading.Thread(target=odd,args=(2,30))
    t1.start()
    t2.start()

main()
```

Output

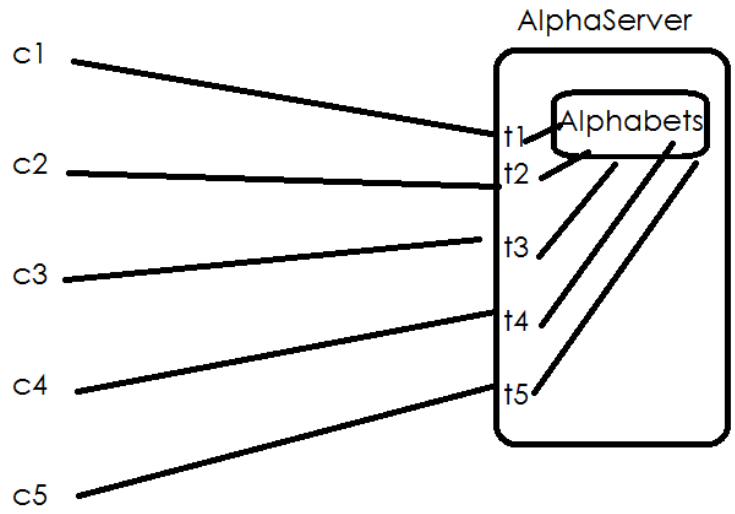


Creating thread by inheriting Thread class

Syntax:

```
class <thread-class-name>(threading.Thread):
        def __init__(self):
                super().__init__()
        def run(self):
                statement-1
                statement-2
```

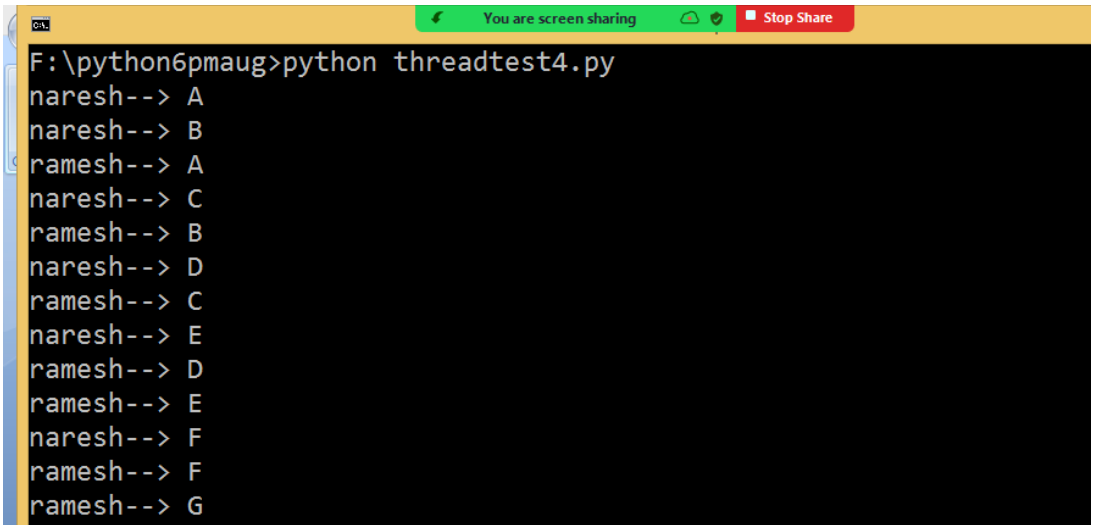
The operation performed by Thread is written inside run method. Every thread class must override run() method.



Example:

```
import threading
class AlphaThread(threading.Thread):
    def __init__(self):
        super().__init__()
    def run(self):
        for n in range(65,91):
            print(f'{self.name}--> {n:c}')
def main():
    t1=AlphaThread()
    t2=AlphaThread()
    t1.name="naresh"
    t2.name="ramesh"
    t1.start()
    t2.start()
main()
```

Output:



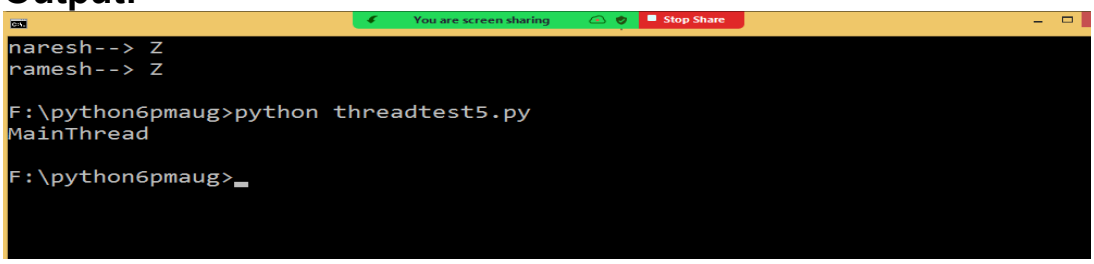
Every python program is executed within PVM as a thread. Default thread created by PVM is MainThread.

Example:

```
# finding thread which is currently in execution
import threading
def main():
    t1=threading.current_thread()
    print(t1.name)

main()
```

Output:



Thread Synchronization

Synchronization process of acquiring the lock of the object

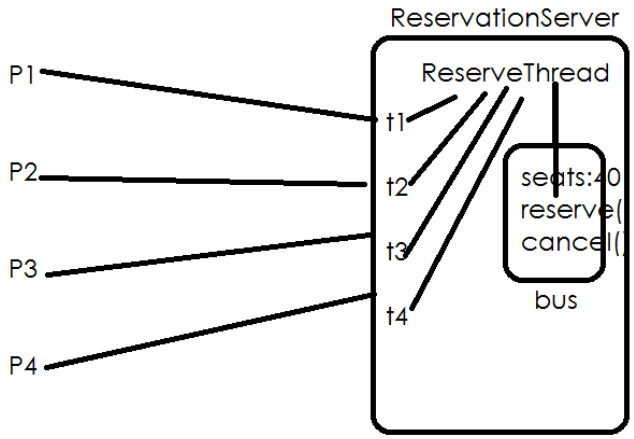
When more than one thread operates on shared resource, when one thread is operating on shared resource not allowing another thread to operate is called synchronization.

Thread synchronization allows develop thread safe applications.

Lock object

Threading module provides Lock class to implement locking
Lock class provides 2 methods

1. acquire : Acquire a lock, blocking or non-blocking.
2. release: Release a lock. This can be called from any thread, not only the thread which has acquired the lock.



Example:

```
import threading
class Bus:
    def __init__(self):
        self.seats=40
        self.l=threading.Lock()
    def reserve(self,name,n):
        self.l.acquire()
        for i in range(n):
            self.seats=self.seats-1
        print(f'{name} your seats are reserved')
        print(f'available seats {self.seats}')
        self.l.release()

class ReserveThread(threading.Thread):
    def __init__(self,b,name,n):
        super().__init__()
        self.bus=b
        self.name=name
```

```
        self.n=n
    def run(self):
        self.bus.reserve(self.name,self.n)

def main():
    bus1=Bus()
    t1=ReserveThread(bus1,"naresh",10)
    t2=ReserveThread(bus1,"suresh",5)
    t3=ReserveThread(bus1,"kishore",10)
    t1.start()
    t2.start()
    t3.start()
main()
```

Output:

```
F:\python6pmaug>python threadtest6.py
naresh your seats are reserved
available seats 30
suresh your seats are reserved
available seats 25
kishore your seats are reserved
available seats 15
```