Scripting Language

UNIT-IV

K.RaviKanth, Ast.Prof., Dept. of CSE, IIIT- RGUKT- Basara, T.S, IND.

Multi Threading

Multi Tasking

- In multi tasking environment, several tasks are given to processor at a time.
- Multi tasking can be done by scheduling algorithms.
- In this, most of the processor time is getting engaged and it is not sitting idle.
- Multi-Processing
- Multi-Threading

Uses:

- To use the processor time in a better way.
- To achieve good performance.
- To make processor not to sit idle for long time.

Multi Processing	Multi Threading	Multi Tasking
More than one process running simultaneously	More than one thread running simultaneously	More than one program running simultaneously
Its part of a Program	Its part of a Program	Two or more programs running concurrently on a computer
It's a heavy wait process	It's a light wait process	It's a heavy wait process
Process is divided into threads	Threads are divided into sub threads	Different tasks are scheduled based on algorithms
There is no communication between processors directly	Within the process, threads are communicated	There is no communication between programs directly

All the above are meant for optimum utilization of CPU resources.

Python pgms that we have seen so far contain only a single sequential flow of control. The program begins, run through a sequence of executions & finally ends. At any given point of time, there is only one statement under execution

Multi Threading

- Multithreading is nothing but concurrent programming
- A multithreaded pgm contains two or more parts that can run concurrently (at the same time).
- Each part of such pgm is called a “thread” & each thread defines a specific path of execution.
- Multithreading is a specialized form of Multitasking
- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a thread, and each thread defines a separate path of execution.
- Thus, multithreading is a specialized form of multitasking.
- There are two distinct types of multitasking:
  - → Process-based and
  - → Thread-based.

A single sequential flow of control in a process (pgm under execution) is called a thread

(or)

An independent path of execution in a program is a thread. It has a beginning, a body & an end

Every pgm will have atleast one thread

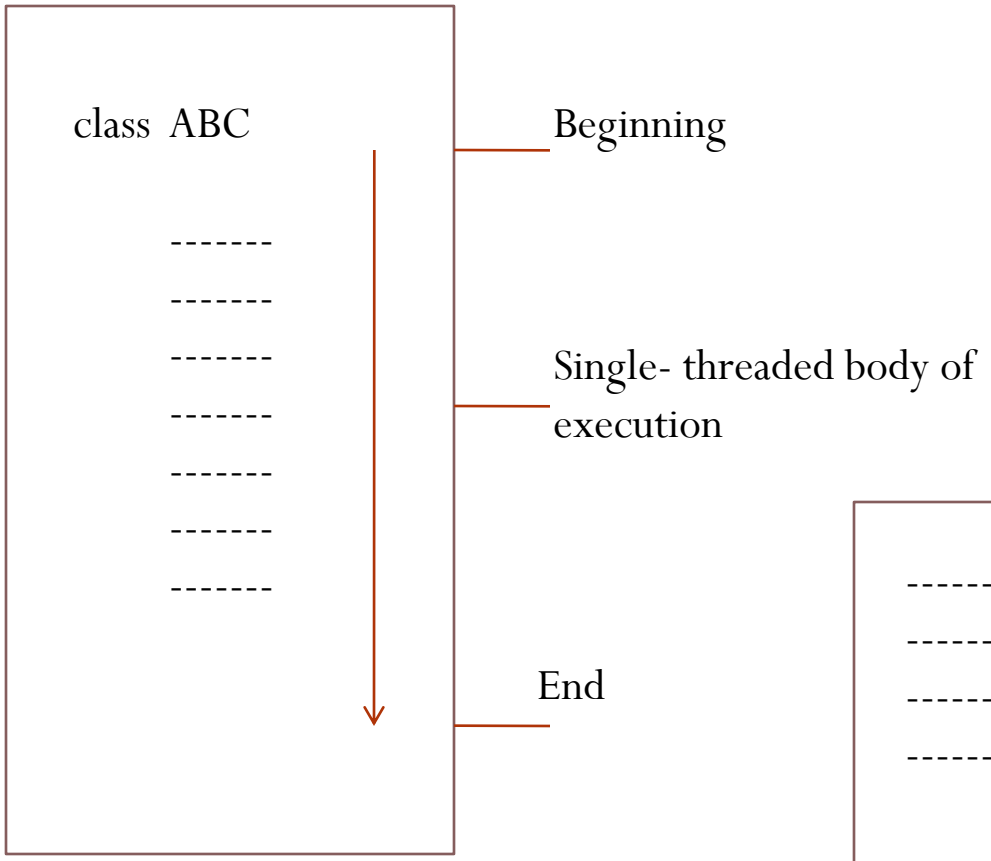
Based on the concept of threads, applications may be classified into two kinds

1. Single-threaded application
2. Multi-threaded application

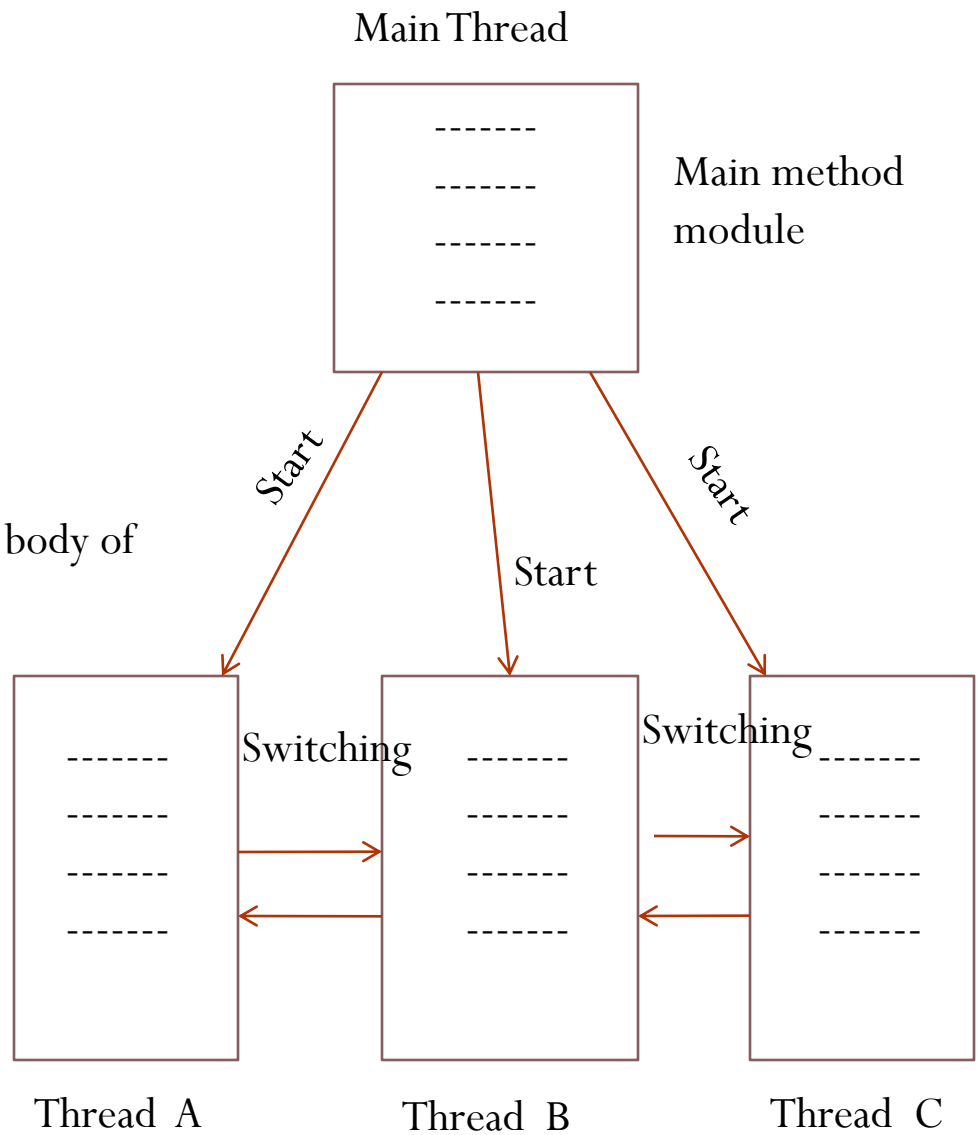
A single threaded appl. Can perform only one job at-a-time, as they have only one flow of control

An appl. is said to be multithreaded, if it has multiple flows of control. In each flow, we can perform one job.

Whenever a single appl. has to perform multiple concurrent jobs, that appl. is a right on to be made as multithreaded.



A single –threaded pgm



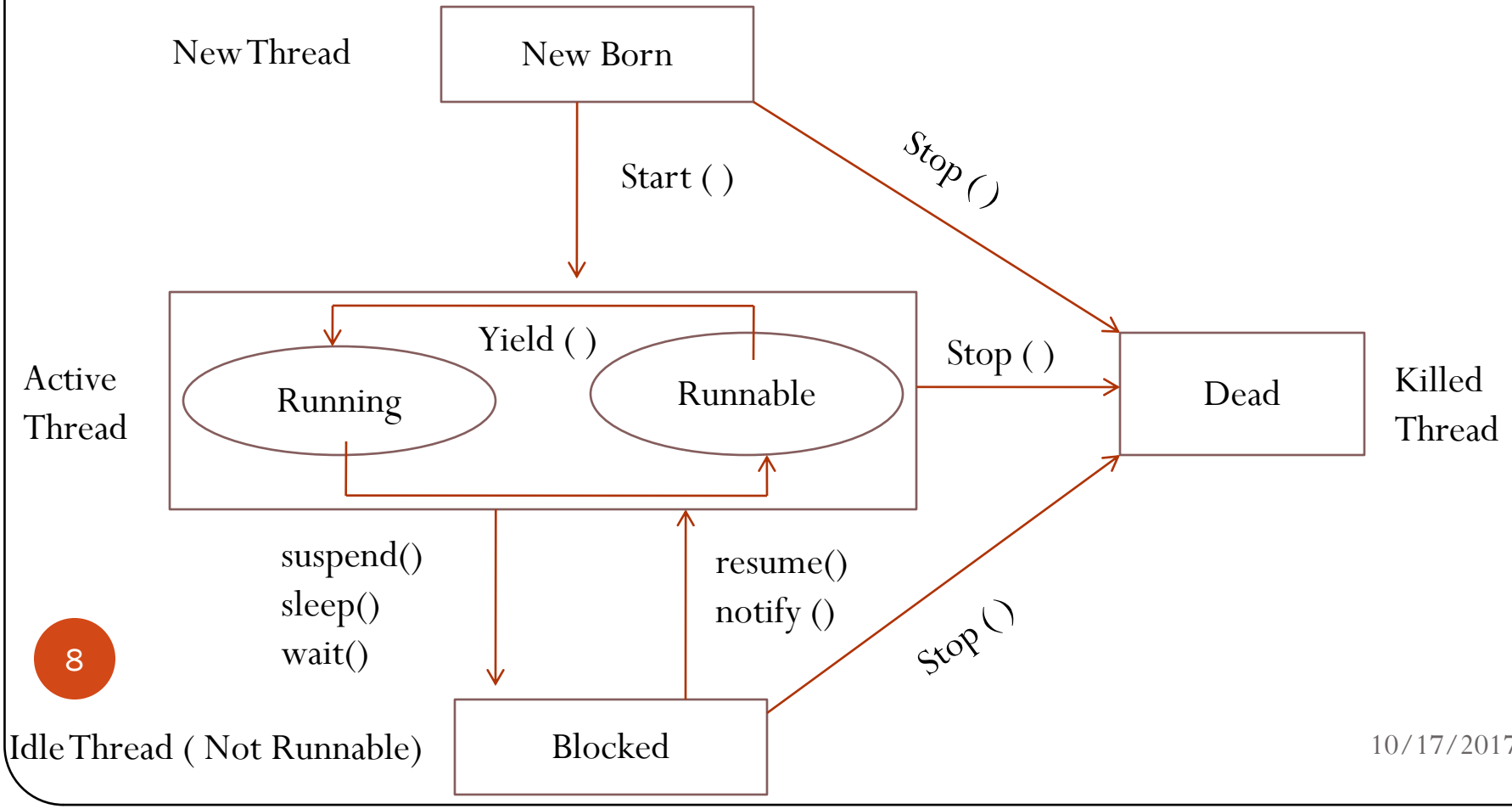
A Multi-threaded Program

Life Cycle of a Thread:

During the life time of a thread, there are many states it can enter. The important states are,

1. **Newborn state** **2. Runnable State** **3. Running State**

4. Blocked State **5. Dead State**



A thread is always in any of these states & it can move from one state to another via a variety of ways

Newborn State:

when we create a thread object, the thread is born & is said to be in "Newborn State". The thread is not yet scheduled for running

At this state, we can do only one of two operations

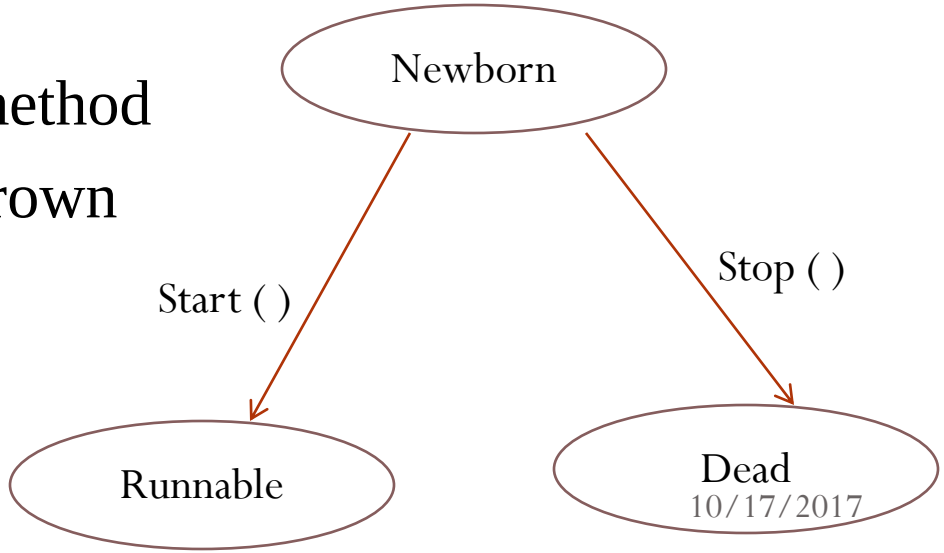
1. Schedule it for running using "start( ) method".

   Then it moves to the runnable state

2. Kill it using "stop( ) method".

Then it moves to the Dead state

If we attempt to use any other method at this state, an exception will be thrown



Blocked State:

- A thread is said to be blocked, when it is prevented from entering into the Runnable state & subsequently the Running state.
- This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements
- A blocked thread is considered “not runnable” but not dead & therefore fully qualified to run again

Dead State:

- Every thread has a life cycle.
- A thread ends its life when it has completed executing its “run() method”. It is a natural death
- We can kill the thread by sending the “stop() method” to it at any state, thus causing a premature death to it.

Introduction:

- A THREAD, also known as thread of execution is defined as the smallest unit that can be scheduled in an operating system
- They are usually contained in processes
- A Process can have more than one thread that can run concurrently
- These threads share the memory and the state of the process
- Thus a single set of code or program can perform two or more tasks simultaneously i.e, it is used by several processors at different stages of execution is known as Multi-threading
- A thread is a light weight process which do not require much memory
- Multi-threading or running several threads is similar to running diff. programs simultaneously
- The main benefit of MT is that multiple threads within a process share the same data space with the main thread

- A thread can be pre-empted(interrupted),suspended,resumed and blocked before it finally terminates
- There are two different kinds of threads
  - Kernal threads
  - User-space threads (or User threads)
- The Kernal threads are a part of the operating system and the user cannot manipulate them from their code
- A User-space thread is similar to a function call. Every process has at least one thread i.e., the process itself
- It can even start multiple threads which are executed by OS executes as parallel processors
- On a single processor machine, parallelism is achieved by thread Scheduling or Time-slicing
- There are two modules which help in multi-threading namely **_thread and threading**

Advantages of Multi-Threading

- Multi-threaded programs can execute faster on computers with multiple CPUs. Each thread can be executed by a separate CPU
- The program can remain responsive to input
- Besides having local variable, threads of a process can share the memory of global variables. If a global variable is changed in one thread, this change is valid for all threads
- Handling of threads is simpler than handling of processes for an operating system. This is why they are sometimes called lightweight process(LWP)

Starting a new THREAD using the _thread Module

- To begin a new thread we need to call **start_new_thread()** method of the _thread module
- **Syntax:**

  _thread .start_new_thread(function_name,args[, kwargs])

  Here,

  args is a tuple of arguments

  kwargs is an optional dictionary of keyword arguments

  The start_new_thread( ) method returns immediately

  It starts_new_thread( ) method returns immediately. It starts the child thread and calls function with the passed list of args. When function returns ,the thread terminates
- **Note:** In the start_new_thread() method use an empty tuple to call function without passing any arguments

- **# Program to implement multi-threading. The threads print the current time**

```
import _thread
import time
#Define a function for the thread
def display_time(threadName, delay):
    count=0
    while count<5:
        time.sleep(delay)
        count +=1
        print("%s: %s" % (threadName, time.ctime(time.time())))
# Create two threads as follows
try:
    _thread.start_new_thread(display_time,("ONE",1,))
    _thread.start_new_thread(display_time,("TWO",2,))
except:
    print("Error : unable to start thread")
```

OUTPUT:

```
ONE: Mon Oct  9 08:49:21 2017
TWO: Mon Oct  9 08:49:22 2017
ONE: Mon Oct  9 08:49:22 2017
ONE: Mon Oct  9 08:49:23 2017
TWO: Mon Oct  9 08:49:24 2017
ONE: Mon Oct  9 08:49:24 2017
ONE: Mon Oct  9 08:49:25 2017
TWO: Mon Oct  9 08:49:26 2017
TWO: Mon Oct  9 08:49:28 2017
TWO: Mon Oct  9 08:49:30 2017
```

The Threading Module

- The threading module, which was first introduced in python 2.4, provides much more powerful, high-level support for threads than the thread module

- It defines some additional methods as follows:
- **threading.activeCount( )** : Returns the number of active thread objects
- **threading.currentThread( )** : Returns the count of thread objects in the caller's thread control
- **threading.enumerate( )** : Returns a list of all active thread objects

- Besides these methods, the threading module has the Thread class that implements threading. The methods providing by the Thread class include:

- **run( )** : This method is the entry point for a thread
- **start( )**: This method starts the execution of a thread by calling the run method
- **join([time])**: The join() method waits for threads to terminate
- **isAlive( )**: The isAlive() method checks whether a thread is still executing
- **getName( )**: As the name suggests, it returns the name of a thread
- **setName( )**: This method is used to set name of a thread

Creating a Thread using Threading Module

- Follow the steps to implement a new thread using the Threading module
  - Define a new subclass of the Thread class
  - Override the _init_() method
  - Override the run()method. In the run() method specify the instruction that the thread should perform when started
  - Create a new Thread subclass and then use its object to start the thread by calling its start()method which in turn calls the run() method.

```
# MT_ex2: Program to create a thread using the threading module

import threading
import time
class myThread(threading.Thread): #create a sub class
    def _init_(self,name,count):
        threading.Thread._init_(self)
        self.name =name
        self.count =count
    def run(self):
        print("\n Starting" +self.name)
        i=0
        while i<self.count:
            display(self.name,i)
            time.sleep(1)
            i+=1
        print("\n Exiting "+self.name)
def display(threadName,i):
    print("\n",threadName,i)
#create new threads
thread1 = myThread("ONE",5)
thread2 = myThread("TWO",5)
#Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("\n Exiting Main Thread")
```

Output:

```
Starting ONE    Starting TWO
ONE   0
TWO  0
ONE    1
TWO  1
ONE   2
TWO  2
ONE   3
TWO  3
ONE   4
TWO  4
Exiting  ONE
Exiting   TWO
Exiting Main Thread
```

Synchronizing Threads

- When two or more threads need access to a shared resource, they need a way to ensure that the resource will be used only by one thread at a particular time. Else, may lead to serious problems
- The Threading module also includes a locking mechanism to synchronize threads
- It supports the following methods:
- **lock()** method when invoked returns the new lock
- **acquire([blocking])** method of the new lock object forces threads to run synchronously. The optional blocking parameter is used to control whether the thread waits to acquire the lock
  - If the value of blocking is 0, the thread returns 0 if the lock cannot be acquired and 1 if the lock was acquired
  - If blocking is set to 1, the thread blocks and waits for the lock to be released
- **release()** method of the new lock object releases the lock when it is no longer required
- **Note :** The output of the following program may vary on your PC subject to the processors speed and number of applications running currently

- **# Program to synchronize threads by locking mechanism**

```
import threading
import time
class myThread(threading.Thread):
    # create a sub class
  stopFlag=0
  def __init__(self,name,msg):
      threading.Thread.__init__(self)
      self.name=name
      self.msg=msg
  def run(self):
      print("\n Starting" +self.name)
      self.display()
      time.sleep(1)
      print("\n Exiting "+self.name)
def display(self):
      while self.stopFlag!=3:
          Lock.acquire()
          print("[",self.msg,"]")
          Lock.release()
          self.stopFlag+=1
Lock=threading.Lock()
#Create new threads
thread1=myThread("ONE","HELLO")
thread2=myThread("TWO","WORLD")
#Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("\n Exiting Main Thread")
```

Output:

- StartingONE
- StartingTWO
- [HELLO]
- [HELLO]
- [HELLO]
- [WORLD]
- [WORLD]
- [WORLD]
- Exiting ONE
- Exiting TWO
- Exiting Main Thread