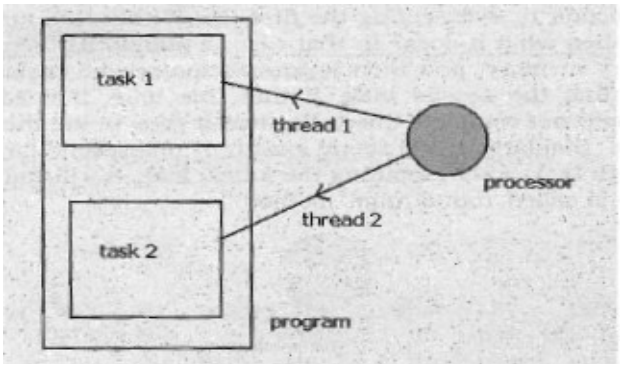
Multithreading

- Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms , that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

- A ***process*** consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process.

- In Thread-based multitasking, thread is the smallest unit of code, which means a single program can perform two or more tasks simultaneously.
  Example :- a text editor can print and at the same time you can edit text file with provided text.That means those two tasks are perform by separate threads.

- In multi thread program several part s of the same program is executed at a time, by microprocessor.



- In the above process there are two programs. These parts may represent two separate blocks of code or two separate methods containing processes. Each part may perform a separate task. The processor should execute the two parts simultaneously. So the processor uses 2 separate threads to execute these two parts.

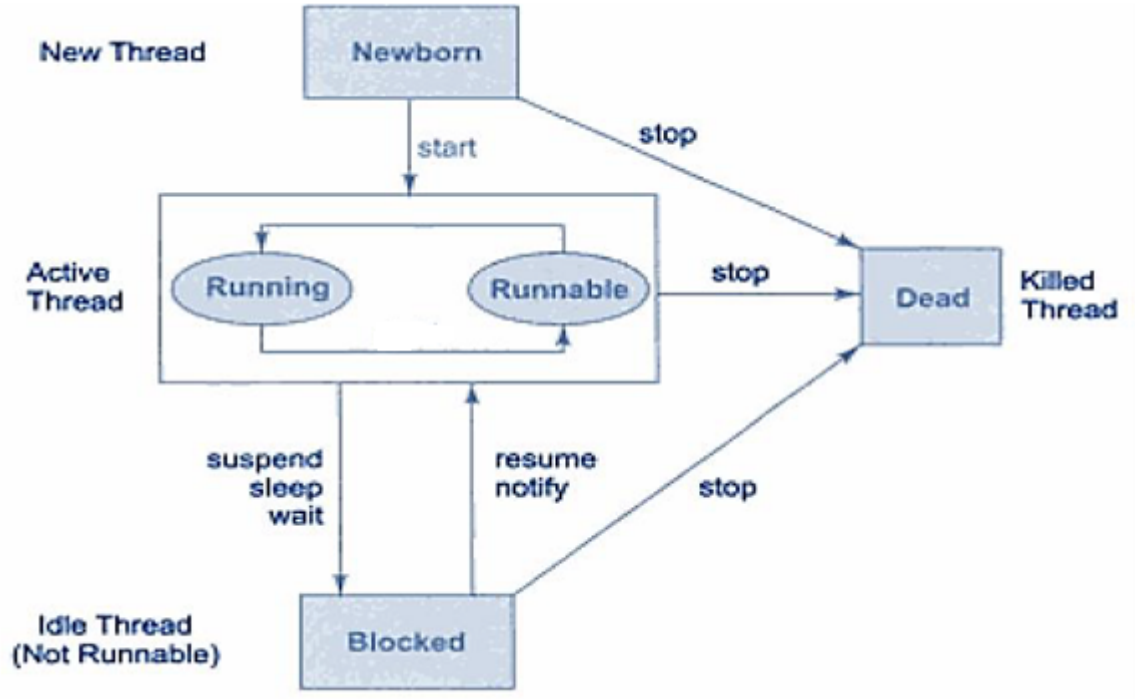
- Each thread can be imagined as an individual process that can execute separate statements. We can imagine these threads as hands of the microprocessor.

Benefits of Multithreading

- Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the program execution
- Improved performance and concurrency
- Enables programmers to do multiple things at one time
- Simultaneous access to multiple applications

Life Cycle of Thread

A thread can be in any of the five following states



1. **Newborn State:** When a thread object is created a new thread is born and said to be in Newborn state.
2. **Runnable State:** If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor. If all threads in queue are of same priority then they are given time slots for execution
3. **Running State:** It means that the processor has given its time to the thread for execution. A thread keeps running until the following condition occurs
   Thread give up its control when a thread is made to sleep for a specified period of time using s**leep(time)** method, where time in milliseconds.
4. **Blocked State:** If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blocked state.
5. **Dead State:** A runnable thread enters the Dead or terminated state when it completes its task or otherwise terminates.

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])

- **args** is a tuple of arguments
- **kwargs(** keyword arguments) is an optional
- **The start_new_thread( )** 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

Example

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

```
import thread
```

```
import time

# Define a function for the thread
def print_time( threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print "{}: {}".format( threadName, time.ctime(time.time()) )

# Create two threads as follows
thread.start_new_thread( print_time, ("Thread-1", 2, ) )
thread.start_new_thread( print_time, ("Thread-2", 4, ) )
while True:
          pass
```

Output

```
Thread-1: Sun Mar 18 18:26:54 2018

Thread-1: Sun Mar 18 18:26:56 2018

 Thread-2: Sun Mar 18 18:26:56 2018

Thread-1: Sun Mar 18 18:26:58 2018

Thread-2: Sun Mar 18 18:27:00 2018

 Thread-1: Sun Mar 18 18:27:00 2018

Thread-1: Sun Mar 18 18:27:02 2018

Thread-2: Sun Mar 18 18:27:04 2018

Thread-2: Sun Mar 18 18:27:08 2018

Thread-2: Sun Mar 18 18:27:12 2018
```

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
- **hreading.currentThread( )**: Returns the count of thread objects in thecaller’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

Example

```
import time
import threading
def calc_square(numbers):
        print("calculate sqaure numbers")
        for n in numbers:
                time.sleep(0.2)
                print('sqaure:',n*n)
def calc_cube(numbers):
        print("Calcualte cube of numbers")
        for n in numbers:
                time.sleep(0.2)
                print('cube:',n*n*n)
arr=[2,3,8,9]
t=time.time()
print "Thread active count",threading.activeCount( )
t1=threading.Thread(target=calc_square,args=((arr,)))
t2=threading.Thread(target=calc_cube,args=((arr,)))
t1.start()
print "Thread active count",threading.activeCount( )
t2.start()
print "Thread active count",threading.activeCount( )
print "Active thread are",threading.enumerate( )
t1.join()
t2.join()
print("done in ;",time.time()-t)
```

Output

```
Thread active count 1
calculate sqaure numbers
Thread active count 2
Calcualte cube of numbers
Thread active count 3
Active thread are [<_MainThread(MainThread, started -1211128064)>, <Thread(Thread-1, started
-1214276800)>, <Thread(Thread-2, started -1224737984)>]
('sqaure:', 4)
('cube:', 8)
('sqaure:', 9)
 ('cube:', 27)
('cube:', 512)
```

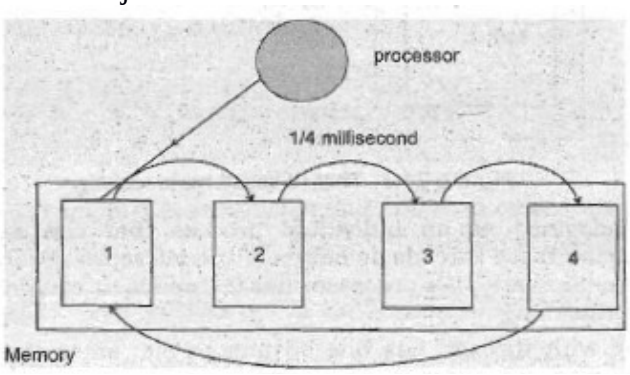
('sqaure:', 64)
('cube:', 729)
('sqaure:', 81)
('done in ;', 0.8023691177368164)

Multiprocessing

Multiprocessing refers to the ability of a system to support more than one processor at the same time. Applications in a multiprocessing system are broken to smaller routines that run independently. The operating system allocates these threads to the processors improving performance of the system.

Consider a computer system with a single processor. If it is assigned several processes at the same time, it will have to interrupt each task and switch briefly to another, to keep all of the processes going.Multiprocessing will dividing work between multiple processes. Multiprocessing is a feature that allows your computer to run two or more programs concurrently. **Example:-** you can listen to music and at the same time chat with your friends on Facebook using browser.

Suppose we want to execute 4 jobs at a time. We load them into memory. Now the micro processor has to execute them all at a time. The memory is divided into 4 parts and the jobs are loaded there. Within the time slot, it will try to execute each job. So the processor will take small amount of time duration. This small part of the processor time is called " Time Slice". Within in this time slot the processor try to execute the jobs.



In process based multi tasking, several programs are executed at a time, by the microprocessor.

Advantages of multiprocessing

- The main advantage of multiprocessor system is to get more work done in a shorter period of time. These types of systems are used when very high speed is required to process a large volume of data. Multi processing systems can save money in comparison to single processor systems because the processors can share peripherals and power supplies.
- It also provides increased reliability in the sense that if one processor fails, the work does not halt, it only slows down. e.g. if we have 10 processors and 1 fails, then the work does not halt, rather the remaining 9 processors can share the work of the 10th processor. Thus the whole system runs only 10 percent slower, rather than failing altogether.

Example 1

```
import time
import multiprocessing
def calc_square(numbers):
        print("calculate sqaure numbers")
        for n in numbers:
                time.sleep(0.2)
                print('sqaure:',n*n)
def calc_cube(numbers):
        print("Calcualte cube of numbers")
        for n in numbers:
                time.sleep(0.2)
                print('cube:',n*n*n)
arr=[2,3,8,9]
t=time.time()
t1=multiprocessing.Process(target=calc_square,args=((arr,)))
t2=multiprocessing.Process(target=calc_cube,args=((arr,)))
t1.start()
t2.start()
t1.join()
t2.join()
print("done in ;",time.time()-t)
```

Difference between Multithreading and Multiprocessing

Basis for Comparison	Multiprocessing	Multithreading
Execution	Multiple processes are executed concurrently.	Multiple threads of a single process are executed concurrently.
Creation	Creation of a process is time-consuming and resource intensive.	Creation of a thread is economical in both sense time and resource.
Communication	There is no communication between processors directly	With in the process threads are communicated
Memory	Processes require their own separate address space	Threads share the address space only

Normal Code

```
import time
def calc_square(numbers):
        print("calculate sqaure numbers")
        for n in numbers:
                time.sleep(0.2)
                print('sqaure:',n*n)
def calc_cube(numbers):
        print("Calcualte cube of numbers")
        for n in numbers:
                time.sleep(0.2)
                print('cube:',n*n*n)
arr=[2,3,8,9]
t=time.time()
calc_square(arr)
calc_cube(arr)
print("done in ;",time.time()-t)
```