Raja N.L.Khan Women's College(Autonomous)
Department of BCA
2nd Semester
Paper: C3T
Prepared By: Moumita Mondal

Multithreaded Programming

Multithreading:

Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel. 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.

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.

Benefits of Multithreading:

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

There are two distinct types of **Multitasking** i.e. Processor-Based and Thread-Based multitasking.

Q: What is the difference between thread-based and process-based multitasking?

Ans: As both are types of multitasking there is very basic difference between the two. **Process-Based multitasking** is a feature that allows your computer to run two or more programs concurrently. For example you can listen to music and at the same time chat with your friends on Facebook using browser.

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

Q: Why multitasking thread requires less overhead than multitasking processor?

Ans: A multitasking thread requires less overhead than multitasking processor because of the following reasons:

- Processes are heavyweight tasks where threads are lightweight.
- Processes require their own separate address space where threads share the address Space.
- Interprocess communication is expensive and limited where Interthread communication is inexpensive, and context switching from one thread to the next is lower in cost.

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. At this state, we can do the only one of the following things with it
   i) Scheduling it for running using **start()** method.
   ii) Kill it using **stop()** method.
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 in round robin fashion.
   If we want a thread to relinquish control another thread of equal priority before its turns comes, we can do so with **yield()** method.
3. **Running State:** It means that the processor has given its time to the thread for execution. A thread keeps running until the following conditions occurs
   a. Thread give up its control on its own and it can happen in the following situations
      i. A thread gets suspended using **suspend()** method which can only be revived with **resume()** method
      ii. A thread is made to sleep for a specified period of time using s**leep(time)** method, where time in milliseconds
      iii. A thread is made to wait for some event to occur using **wait ()** method. In this case a thread can be scheduled to run again using **notify()** method.
   b. A thread is pre-empted by a higher priority thread.
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.

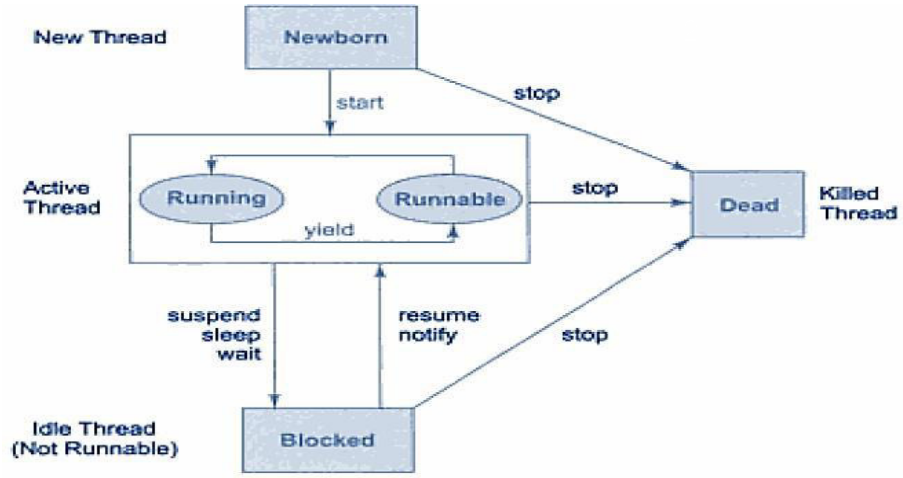

Fig: Life Cycle of Thread

Main Thread:

Every time a Java program starts up, one thread begins running which is called as the *main thread* of the program because it is the one that is executed when your program begins.

- Child threads are produced from main thread
- Often it is the last thread to finish execution as it performs various shut down operations

How to Create a Thread:

Java defines two ways in which this can be accomplished:

- You can implement the Runnable interface.
- You can extend the Thread class, itself.

Create Thread by Implementing Runnable :

The easiest way to create a thread is to create a class that implements the Runnable interface. To implement Runnable, a class need only implement a single method called **run( )**, which is declared like this:

public void run( ) :

You will define the code that constitutes the new thread inside run() method. It is important to understand that run() can call other methods, use other classes, and declare variables, just like the main thread can.

After you create a class that implements Runnable, you will instantiate an object of type

Thread from within that class. Thread defines several constructors. The one that we will use is shown here:

Thread(Runnable threadOb, String threadName);

Here threadOb is an instance of a class that implements the Runnable interface and the name of the new thread is specified by threadName. After the new thread is created, it will not start running until you call its start( ) method, which is declared within Thread. The start( ) method is shown here:

void start( );

Example to Create a Thread using Runnable Interface :

```
class t1 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t1 obj1 = new t1();
        Thread t = new Thread(obj1);
        t.start();
    }
}
```

Output:

```
C:\NIEC Java>javac t1.java

C:\NIEC Java>java t1
Thread is Running

C:\NIEC Java>
```

Create Thread by Extending Thread:

The second way to create a thread is to create a new class that extends Thread, and then to create an instance of that class. The **extending class must override the run( ) method**, which is the entry point for the new thread. It must also call start( ) to begin execution of the new thread.

Example to Create a Thread by Extending Thread Class :

```
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
    }
}
```

Output:

```
C:\NIEC Java>javac t2.java

C:\NIEC Java>java t2
Thread is Running

C:\NIEC Java>
```

Thread Methods with Description:

1. **public void start()**
   Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2. **public void run()**
   If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3. **public final void setName(String name)**
   Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4. **public final void setPriority(int priority)**
   Sets the priority of this Thread object. The possible values are between 1 and 10.
5. **public final void setDaemon(boolean on)**
   A parameter of true denotes this Thread as a daemon thread.
6. **public final void join(long millisec)**
   The current thread invokes this method on a second thread, causing the current thread

Difference between multithreading and multitasking:

Multithreading	Multitasking
It is a programming concept in which a program or a process is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is an operating system concept in which multiple tasks are performed simultaneously.
It supports execution of multiple parts of a single program simultaneously.	It supports execution of multiple programs simultaneously.
The processor has to switch between different parts or threads of a program.	The processor has to switch between different programs or processes.
It is highly efficient.	It is less efficient
A thread is the smallest unit in multithreading.	A program or process is the smallest unit in a multitasking environment.
It helps in developing efficient programs.	It helps in developing efficient operating systems.
It is cost-effective in case of context switching.	It is expensive in case of context switching.