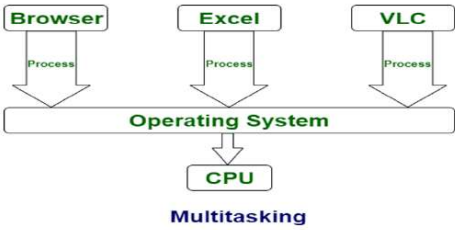
MULTI-TASKING AND MULTI-THREADING

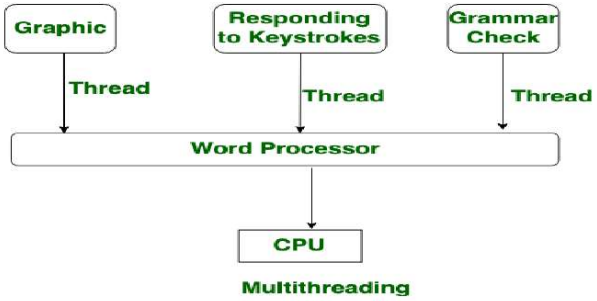
Introduction

- Multi-tasking and multi-threading are two techniques used in operating systems to manage multiple processes and tasks.
- **Multi-tasking** is the ability of an operating system to run multiple processes or tasks concurrently, sharing the same processor and other resources.
- In multi-tasking, the operating system divides the CPU time between multiple tasks, allowing them to execute simultaneously.
- Each task is assigned a time slice, or a portion of CPU time, during which it can execute its code.
- Multi-tasking is essential for increasing system efficiency, improving user productivity, and achieving optimal resource utilization.



Multitasking

- **Multi-threading** is a technique in which an operating system divides a single process into multiple threads, each of which can execute concurrently.
- Threads share the same memory space and resources of the parent process, allowing them to communicate and synchronize data easily.
- Multi-threading is useful for improving application performance by allowing different parts of the application to execute simultaneously.



Multithreading

DIFFERENCE BETWEEN MULTI-TASKING AND MULTI-THREADING

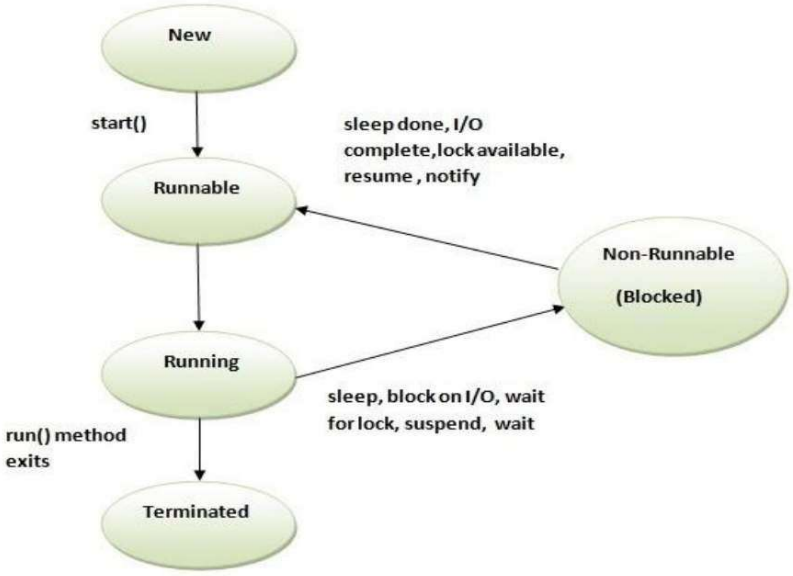
S.NO	Multitasking	Multithreading
1.	In multitasking, users are allowed to perform many tasks by CPU.	While in multithreading, many threads are created from a process through which computer power is increased.
2.	Multitasking involves often CPU switching between the tasks.	While in multithreading also, CPU switching is often involved between the threads.
3.	In multitasking, the processes share separate memory.	While in multithreading, processes are allocated the same memory.
4.	The multitasking component involves multiprocessing.	While the multithreading component does not involve multiprocessing.
5.	In multitasking, the CPU is provided in order to execute many tasks at a time.	While in multithreading also, a CPU is provided in order to execute many threads from a process at a time.
6.	In multitasking, processes don't share the same resources, each process is allocated separate resources.	While in multithreading, each process shares the same resources.
7.	Multitasking is slow compared to multithreading.	While multithreading is faster.
8.	In multitasking, termination of a process takes more time.	While in multithreading, termination of thread takes less time.
9.	Isolation and memory protection exist in multitasking.	Isolation and memory protection does not exist in multithreading.
10.	It helps in developing efficient programs.	It helps in developing efficient operating systems.
11.	Involves running multiple independent processes or tasks	Involves dividing a single process into multiple threads that can execute concurrently
12.	Multiple processes or tasks run simultaneously, sharing the same processor and resources	Multiple threads within a single process share the same memory space and resources
13.	Each process or task has its own memory space and resources	Threads share the same memory space and resources of the parent process
14.	Used to manage multiple processes and improve system efficiency	Used to manage multiple processes and improve system efficiency
15.	Examples: running multiple applications on a computer, running multiple servers on a network	Examples: splitting a video encoding task into multiple threads, implementing a responsive user interface in an application

JAVA THREAD MODEL (LIFE CYCLE OF A THREAD)

- In java, a thread goes through different states throughout its execution.
- These stages are called thread life cycle states or phases.
- A thread can be in one of the five states in the thread.
- The life cycle of the thread is controlled by JVM.

The thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



1. New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

Example

```
Thread t1 = new Thread();
```

2. Runnable

- When a thread calls start( ) method, then the thread is said to be in the Runnable state.
- This state is also known as a Ready state.

Example

```
t1.start( );
```

3. Running

When a thread calls run( ) method, then the thread is said to be Running. The run( ) method of a thread called automatically by the start( ) method.

4. Non-Runnable (Blocked)

- This is the state when the thread is still alive, but is currently not eligible to run.
- A thread in the Running state may move into the blocked state due to various reasons like sleep( ) method called, wait( ) method called, suspend( ) method called, and join( ) method called, etc.
- When a thread is in the blocked or waiting state, it may move to Runnable state due to reasons like sleep time completed, waiting time completed, notify( ) or notifyAll( ) method called, resume( ) method called, etc.

Example

```
Thread.sleep(1000);
wait(1000);
wait();
suspend();
notify();
notifyAll();
resume();
```

5. Terminated

- A thread in the Running state may move into the dead state due to either its execution completed or the stop( ) method called.
- The dead state is also known as the terminated state.

CREATING THREADS

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

1. By extending Thread class

The java contains a built-in class Thread inside the java.lang package. The Thread class contains all the methods that are related to the threads.

To create a thread using Thread class, follow the step given below.

Step-1: Create a class as a child of Thread class. That means, create a class that extends Thread class.

Step-2: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.

Step-3: Create the object of the newly created class in the main( ) method.

Step-4: Call the start( ) method on the object created in the above step.

Example: By extending Thread class

```
class SampleThread extends Thread
{
        public void run()
        {
                System.out.println("Thread is under Running...");
                for(int i= 1; i<=10; i++)
                {
                        System.out.println("i = " + i);
                }
        }
}
public class My_Thread_Test
{

        public static void main(String[] args)
        {
                SampleThread t1 = new SampleThread();
                System.out.println("Thread about to start...");
                t1.start();
        }
}
```

Output:

```
Thread about to start...
Thread is under Running...
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

2. By implementing Runnable interface

- The java contains a built-in interface Runnable inside the java.lang package.
- The Runnable interface implemented by the Thread class that contains all the methods that are related to the threads.

To create a thread using Runnable interface, follow the step given below.

Step-1: Create a class that implements Runnable interface.
Step-2: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.
Step-3: Create the object of the newly created class in the main( ) method.
Step-4: Create the Thread class object by passing above created object as parameter to the Thread class constructor.
Step-5: Call the start( ) method on the Thread class object created in the above step.

Example: By implementing the Runnable interface

```
class SampleThread implements Runnable
{

        public void run()
        {
                System.out.println("Thread is under Running...");
                for(int i= 1; i<=10; i++)
                {
                        System.out.println("i = " + i);
                }
        }
}
public class My_Thread_Test
{
        public static void main(String[] args)
        {
                SampleThread threadObject = new SampleThread();
                Thread thread = new Thread(threadObject);
                System.out.println("Thread about to start...");
                thread.start();
        }
}
```

Output:

```
Thread about to start...
Thread is under Running...
    i = 1
    i = 2
    i = 3
    i = 4
    i = 5
    i = 6
    i = 7
    i = 8
    i = 9
    i = 10
```

CONSTRUCTORS OF THREAD CLASS

1. Thread()
2. Thread(String name)
3. Thread(Runnable r)
4. Thread(Runnable r,String name)

METHODS OF THREAD CLASS

1. **public void run():** is used to defines actual task of the thread.
2. **public void start():**It moves the thread from Ready state to Running state by calling run( ) method.
3. **public void sleep(long milliseconds):** Moves the thread to blocked state till the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long milliseconds):** waits for a thread to die for the specified milliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void suspend():** is used to suspend the thread(deprecated).

SLEEP() & JOIN()

```
class SampleThread extends Thread
{
		public void run()
		{
			System.out.println("Thread is under Running...");
			for(int i= 1; i<=10; i++)
			{
				try
				{
					Thread.sleep(1000);
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
```

```
                        System.out.println("i = " + i);
                }
        }
}
public class My_Thread_Test
{
        public static void main(String[] args)
        {
                SampleThread t1 = new SampleThread();
                SampleThread t2 = new SampleThread();
                SampleThread t3 = new SampleThread();
                System.out.println("Thread about to start...");
                t1.start();
                        try
                        {
                                t1.join();
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }


                t2.start();
                t3.start();
        }
}
```

THREAD PRIORITIES

- In a java programming language, every thread has a property called priority.
- Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling).
- The thread with more priority allocates the processor first.
- But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

Three constants defined in Thread class:

1. MIN_PRIORITY
2. NORM_PRIORITY
3. MAX_PRIORITY

- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.
- The java programming language Thread class provides two methods setPriority(int), and getPriority( ) to handle thread priorities.

setPriority( ) method

The setPriority( ) method of Thread class used to set the priority of a thread.

It takes an integer range from 1 to 10 as an argument and returns nothing (void).

Example

```
threadObject.setPriority(4);
                    or
threadObject.setPriority(MAX_PRIORITY);
```

getPriority( ) method

The getPriority( ) method of Thread class used to access the priority of a thread.

It does not takes any argument and returns name of the thread as String.

Example

```
String threadName = threadObject.getPriority();
```

Example1

```
class SampleThread extends Thread
{
    public void run()
    {
        System.out.println("Inside SampleThread");
        System.out.println("CurrentThread: " + Thread.currentThread().getName());
    }
}
public class My_Thread_Test
{
        public static void main(String[] args)
        {
            SampleThread threadObject1 = new SampleThread();
            SampleThread threadObject2 = new SampleThread();
            threadObject1.setName("first");
            threadObject2.setName("second");
            threadObject1.setPriority(4);
            threadObject2.setPriority(Thread.MAX_PRIORITY);
            threadObject1.start();
            threadObject2.start();
        }
}
```

Output:

```
Inside SampleThread
Inside SampleThread
CurrentThread: second
CurrentThread: first
```

Example2

```
class MultiThread extends Thread
{
    public void run()
    {
        System.out.println("running thread name is:"+Thread.currentThread().getName());
        System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
    }
    public static void main(String args[])
    {
        MultiThread m1=new MultiThread ();
        MultiThread m2=new MultiThread ();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

Output

```
running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1
```