Introduction to Distributed and Parallel Computing
CS-401

Dr. Sanjay Saxena
Visiting Faculty, CSE, IIIT Vadodara
Assistant Professor, CSE, IIIT Bhubaneswar
Post doc – University of Pennsylvania, USA
PhD – Indian Institute of Technology(BHU), Varanasi

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Advantages of Java Multithreading

- It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- You **can perform many operations together, so it saves time**.
- Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

- Java Multithreading is mostly used in games, animation, etc.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

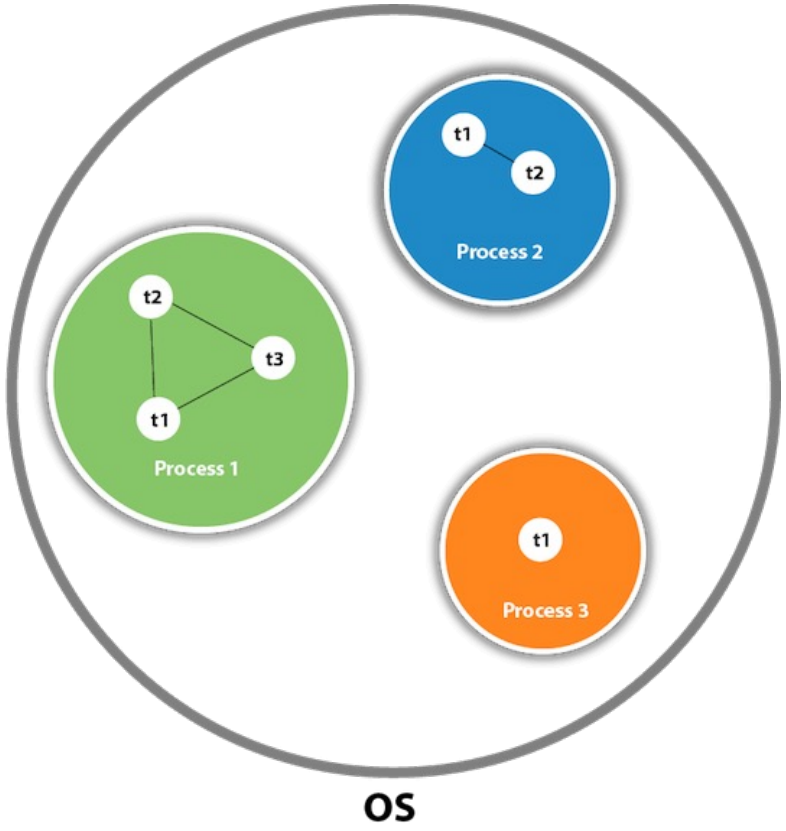
2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



As shown in the figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Life cycle of a Thread (Thread States)

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

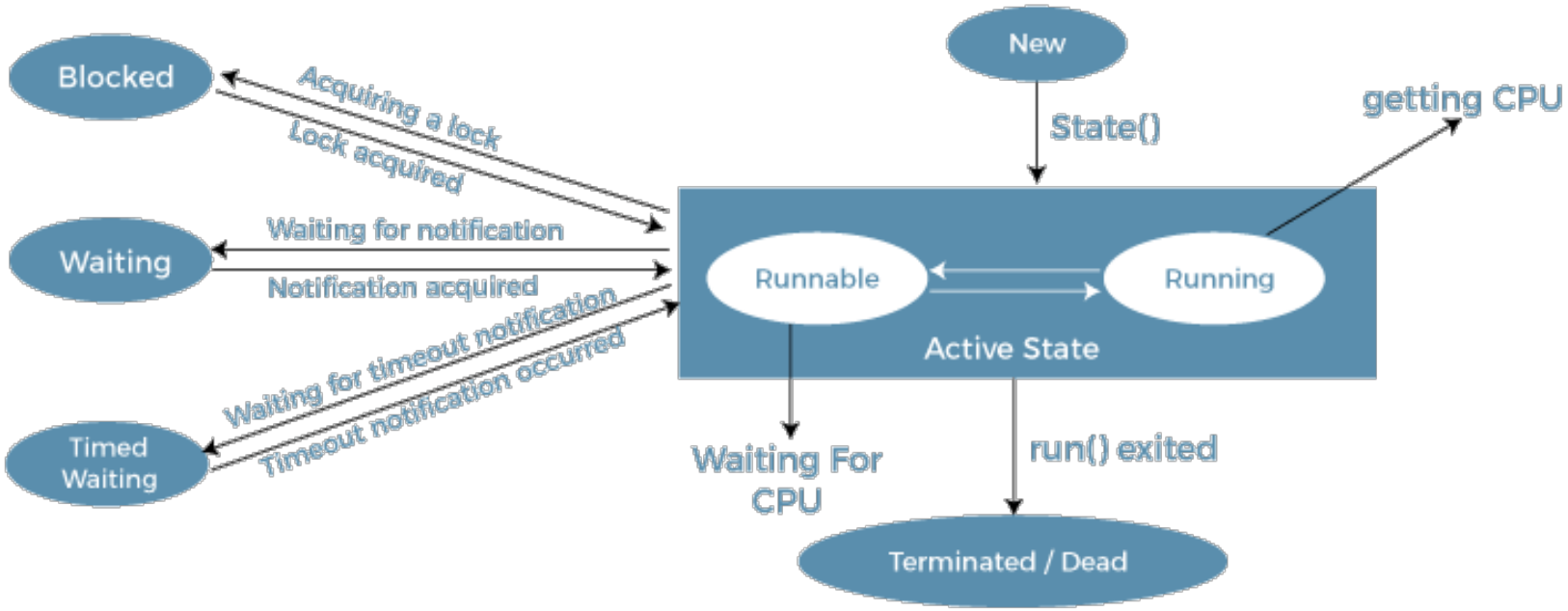
1.New

2.Active

3.Blocked / Waiting

4.Timed Waiting

5.Terminated



Life Cycle of a Thread

Different Thread States

- **New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.
- **Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.
- **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.
  A program implementing multithreading acquires a fixed slice of time to each individual thread.
- **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

- **Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

- **Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.
- **Termination**

Java Threads | How to create a thread in Java

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

1) Java Thread Example by extending Thread class

```
1.class Multi extends Thread{
2.public void run(){
3.System.out.println("thread is running...");
4.}
5.public static void main(String args[]){
6.Multi t1=new Multi();
7.t1.start();
8. }
9.}
```

2) Java Thread Example by implementing Runnable interface

```
1.class Multi3 implements Runnable{
2.public void run(){
3.System.out.println("thread is running...");
4.}
5.
6.public static void main(String args[]){
7.Multi3 m1=new Multi3();
8.Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
9.t1.start();
10. }
11.}
```

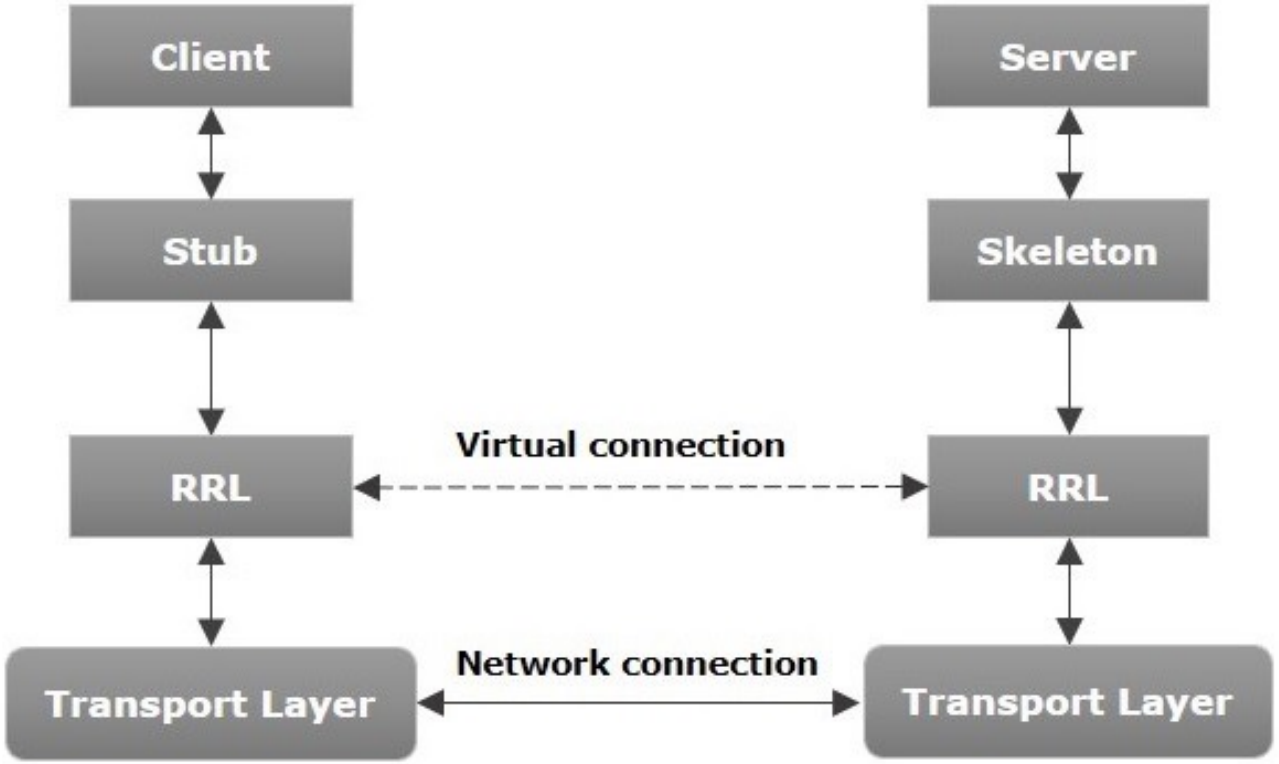
Java RMI: Remote Method Invocation

- It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.
- RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.



- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.

Working of an RMI Application

- When the client makes a call to the remote object, it is received by the stub which eventually passes this request to the RRL.
- When the client-side RRL receives the request, it invokes a method called **invoke()** of the object **remoteRef**. It passes the request to the RRL on the server side.
- The RRL on the server side passes the request to the Skeleton (proxy on the server) which finally invokes the required object on the server.
- The result is passed all the way back to the client.

Marshalling and Unmarshalling

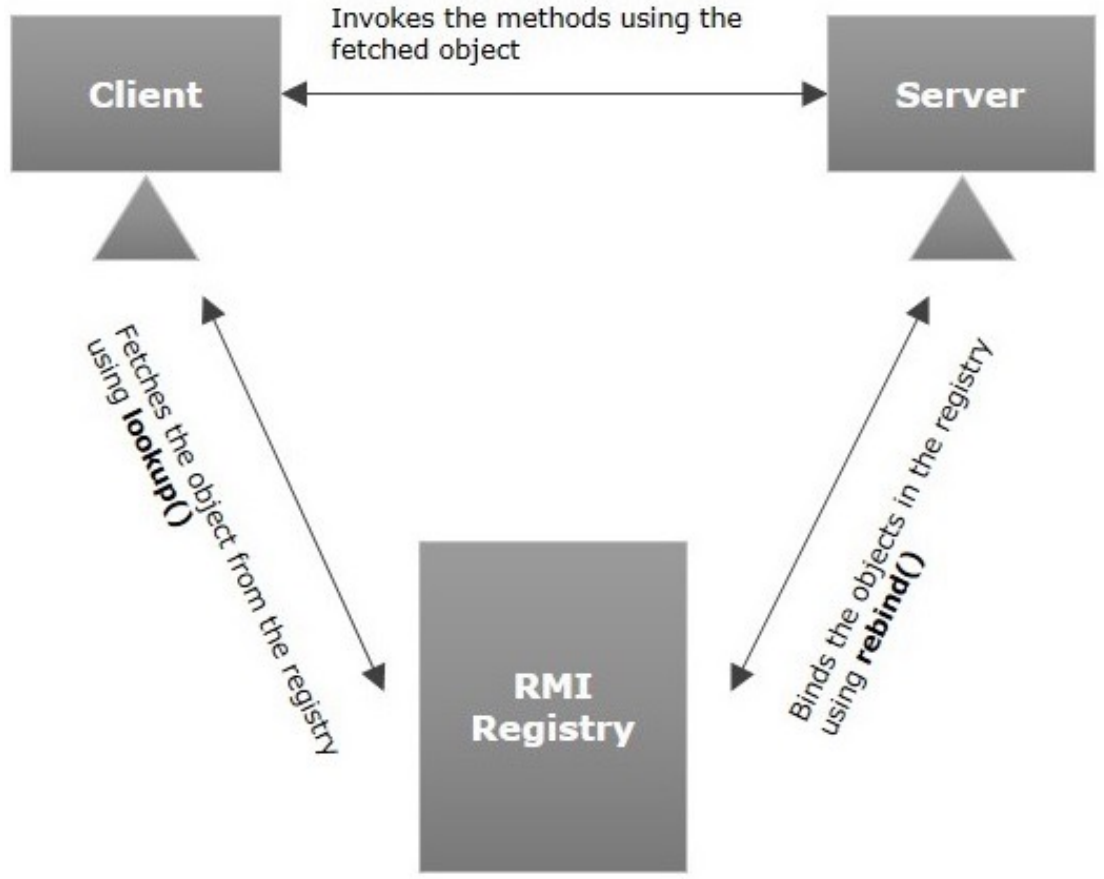
- Whenever a client invokes a method that accepts parameters on a remote object, the parameters are bundled into a message before being sent over the network. These parameters may be of primitive type or objects. In case of primitive type, the parameters are put together and a header is attached to it. In case the parameters are objects, then they are serialized. This process is known as **marshalling**.

- At the server side, the packed parameters are unbundled and then the required method is invoked. This process is known as **unmarshalling**.

RMI Registry

RMI registry is a namespace on which all server objects are placed. Each time the server creates an object, it registers this object with the RMIregistry (using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).



Goals of RMI

- To minimize the complexity of the application.
- To preserve type safety.
- Distributed garbage collection.
- Minimize the difference between working with local and remote objects.

Thanks & Cheers!!

Small aim is a crime; have great aim.
Bharat-Ratan A. P. J. Abdul Kalam