MultiThreading

What's a Process ?

- **Process is nothing but an instance of program in execution (Unit of work in modern time-sharing systems)**
- **Modern operating systems can concurrently execute multiple processes.**

What's Multitasking ?

- ***"The ability to have more than one program working at what seems like the same time"***

Examples :

- **You can edit while printing a document**
- **Web page may be loading multiple images while accepting user inputs**

Multitasking Ways

- **Preemptive Multitasking → OS simple shifts the control to other process without consulting the user (Windows 3.1, Mac OS)**
- **Cooperative Multitasking → OS shifts the control to other process only when the currently running process yields control. (Non-preemptive multitasking) Linux, Windows NT,95**

What's Multithreading?

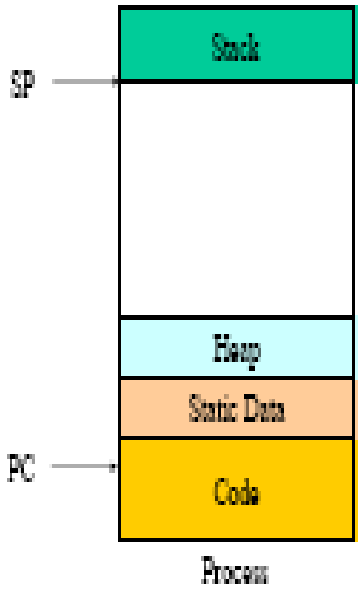
- **Basis of Multi-Tasking at a Program Level**
- **An Individual Program will Appear to do Multiple Tasks at the Same Time. Each Individual Task is Handled by a Thread**
- **"A thread of execution is a program unit that is executed independently of other parts of the program"**

Process VS Thread

S.No	Process	Thread
1	No Sharing of Memory	Sharing of Memory and other data structures
2	Can not Corrupt Data structures	Can Corrupt Data Structures
3	Context switching is Expensive	Context Switching is Chaeper

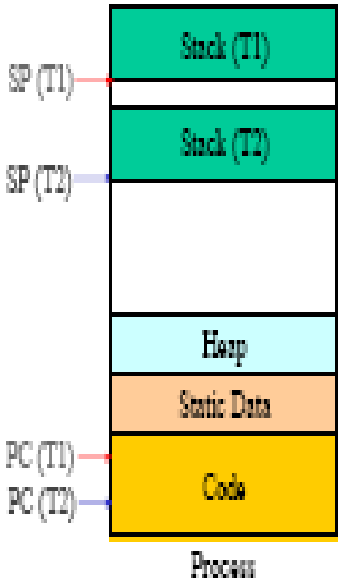
What is a Process?

- Execution context
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers
- Code
- Data
- Stack



What is a Thread?

- Execution context
  - Program counter (PC)
  - Stack pointer (SP)
  - Data registers



Multithreading In Java

1. **Java Supports multithreaded based multitasking**
2. **Virtual Machine Executes each Thread for short period of time (Time Slice)**
3. **Thread Scheduler Activates, deactivates threads**

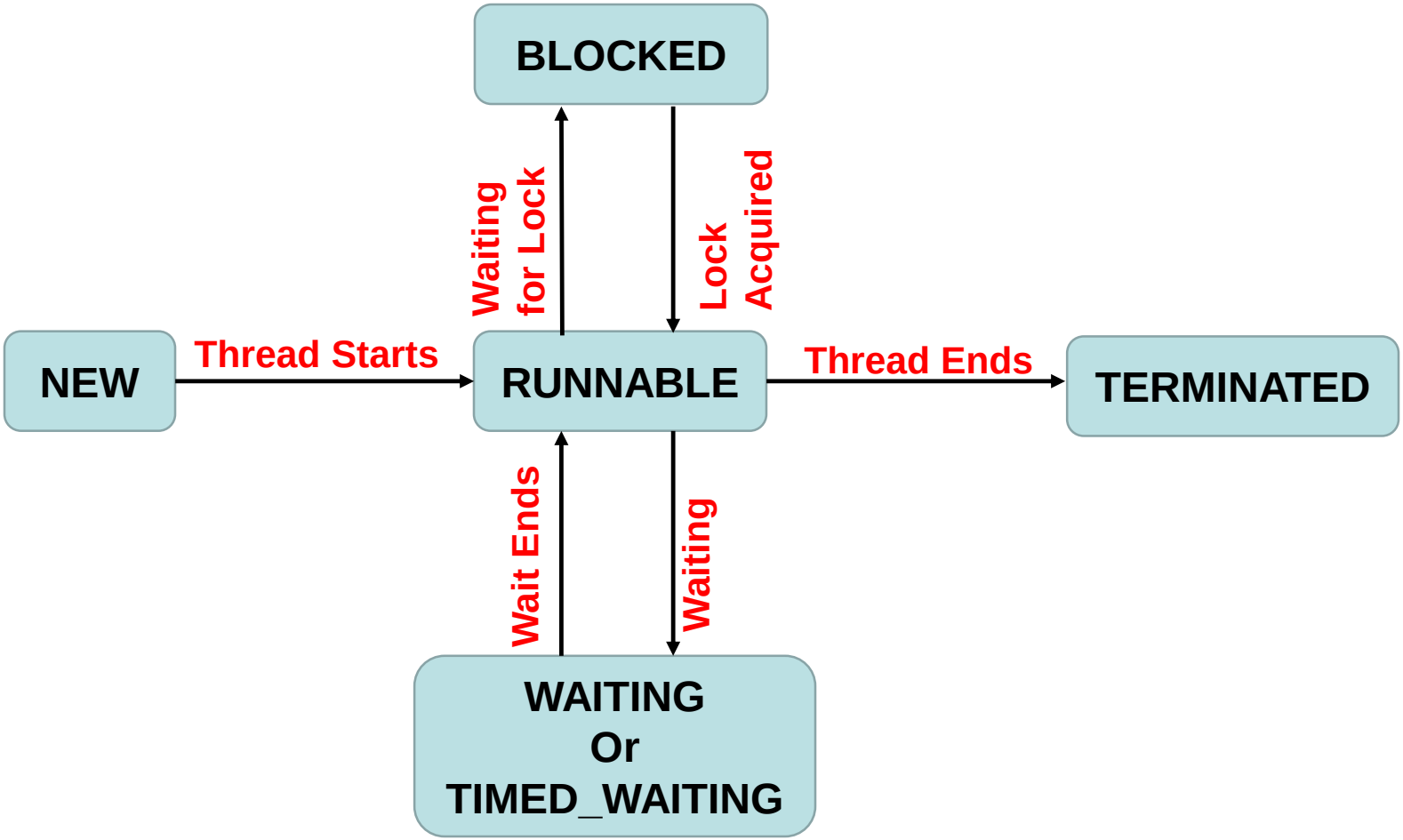
Java Thread Basics (Thread Priorities)

1. Java assigns a thread priority that determines how that thread should be treated with respect to other threads
2. Thread priorities are integers (1 to 10) MIN_PRIORITY =1, MAX_ PRIORITY = 10, NORM_ PRIORITY =5

3. Thread priorities are used by the thread scheduler to decide when each thread should be allowed to run

4. Higher priorities threads gets more CPU time than the lower one

Thread Basics (Thread States)



Thread Basics (Scheduling Threads)

Scheduler activates a new thread if

- **a thread has completed its time slice**
- **a thread has blocked itself**
- **a thread with higher priority has become runnable**

Scheduler determines new thread to run

- **looks only at runnable threads**
- **picks one with max priority**

Thread Basics (Terminating a Thread)

- **Thread Terminates when its run() Method Exits**
- **Sometimes necessary to terminate running thread**
- **Don't use deprecated stop() method**
- **Interrupt thread by calling interrupt() Method [Note : interrupt() Method Throws a Checked Exception named InterruptedException]**
- **Interrupted thread must sense interruption and exit its run() method**
- **Interrupted thread has chance to clean up**

Thread Class In Java

- **Defined in java.lang package**
- **Constructors :**
    1. **Thread()**
    2. **Thread(String name)**
    3. **Thread(Runnable r)**
    4. **Thread(Runnable r, String name)**
- **Important Methods : (Next Slide)**

Important Thread Class Methods

1. getName() → Returns name of the thread
2. getPriority() → Returns priority of the thread
3. isAlive() → Checks if thread is alive or not
4. run() → Entry point for the thread
5. sleep(long millis) → Causes the current thread to sleep for mentioned no of milliseconds
6. sleep(long millis, long nano) → Causes the current thread to sleep for mentioned no of milliseconds and nano seconds
7. start() → To start a thread
8. stop() → To stop a thread (deprecated method)
9. interrupt() → To interrupt a thread
10. isInterrupted() → Checks whether the thread is interrupted or not.
11. currentThread() → Static Method. Returns a reference of the currently executing thread

Main Thread

- main() Thread begins execution immediately
- Other child threads will be spawned from the main thread
- Generally the last thread to terminate
- This thread is created automatically when you start executing your program

How To Create a Thread

- Two Ways of Creating a Thread
  1. **By Extending a Thread class**
  2. **By Implementing a Runnable interface**

Runnable interface

```
public interface Runnable
{
        public void run();
}
```

How To Create a Thread

- **To Create a Thread Your Thread class should either**
  1. Extend a Thread Class OR
  2. Implements a Runnable Interface

```
class MyThread extends Thread
{
  public void run()
 {
   ……….
   ……….
 }
}
```

```
class MyThread implements Runnable
{
  public void run()
 {
   ……….
   ……….
 }
}
```

Example

```
class TH1 extends Thread
{
public void run()
{
try
{
        for(int i=101;i<999;i=i+2)
        {
        System.out.println(i+"By Thread TH1");
        Thread.sleep(10);
        }
}// End of try
catch(InterruptedException e){}
}// End of run
} //end of TH1
```

Thread printing Three digit Odd Numbers

Thread printing Three Digit prime Numbers

```
class TH2 extends Thread
{
public void run()
{
int i,j;
try
{
        for(i=100;i<=999;i++)
        {
        for(j=2;j<i/2;j++)
        {
        if(i % j != 0) continue;
        else
        break;
        }
        if( j >= i/2)
        {
        System.out.println(i+"By Thread TH2");
        Thread.sleep(100);
        }
        } // end of for
}// end of try
catch(InterruptedException e){}
}// End of run
} //end of TH1
```

```
class THtest
{
public static void main(String args[])
{
Thread t1 = new TH1();
Thread t2 = new TH1();
Thread t3 = new TH2();
Thread t4 = new TH2();

t1.start();
t4.start();
System.out.println("Main Method Exited");
}
}
```

Thread Synchronization

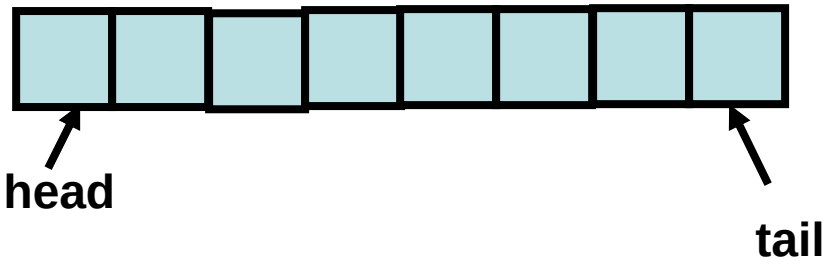
- **Process for Sharing shared resources**
- **Key for synchronization is monitor or semaphore.**
  1. **Monitor is an object that is used as mutually exclusive lock**
  2. **Only one thread can own a monitor at a given time.**
  3. **Acquiring a lock or monitor is said to have entered the monitor.**
  4. **All other threads attempting to acquire the lock or enter the monitor are blocked.**
- **Java implements synchronization thru**
  1. **Synchronized methods (Object Locks)**
  2. **Synchronize Statement**

Data Structure Corruption

- **Threads share access to common objects and hence can conflict with each other and corrupt data structure.**
- **Race Condition may occur**

  “*Race Condition occurs if the effect of multiple threads on shared data depends on the order in which threads are scheduled.*”

Example Corrupting Data Structures




```
public class Queue
{
private Object[ ]  elements;
private int head;
private int tail;
private int size;

public Queue(int capacity){}
public Object removeFirst() {} // head++, size--
public void add(Object anotherObject){} // tail++, size++
public boolean isFull() { }
public boolean isEmpty() { }
} // End of class Queue
```

Producer Thread
Adding Greetings into Queue

```
class Producer implements Runnable
{
private Queue queue;
private int repetitions;
private static final int DELAY =10;
public Producer(String greeting, Queue queue , int reps){}
public void run()
{
try
{
        int i =1;
                while(i <= repetitions)
                {
                        if(!queue.isFull())
                        {
                        queue.add(i+":"+greeting);
                        i++
                        }
                        Thread.sleep(DELAY);
                }
}
catch(InterruptedException e){}
}
}
```

Consumer Thread
Deleting Greetings into Queue

```
class Consumer implements Runnable
{
private Queue queue;
private int repetitions;
private static final int DELAY =10;
public Consumer(Queue queue , int reps){}
public void run()
{
try
{
        int i =1;
        while(i <= repetitions)
        {
                if(!queue.isEmpty())
                {
                Object Obj = queue.removeFirst();
                i++
                }
                Thread.sleep(DELAY);
        }
}
Catch(InterruptedException e){}
}
}
```

Main Class

```
Queue queue = new Queue(10);
final int repetitions = 100;

Runnable r1 = new Producer("Hello, world",queue,repetitions);
Runnable r2 = new Producer("Goodbye, world",queue,repetitions);
Runnable r3 = new Consumer(queue, 2*repetitions);

Thread T1 = new Thread(r1);
Thread T2 = new Thread(r2);
Thread T3 = new Thread(r3);
```

How Queue Can be Corrupted?

Race Conditions

- **How add of Queue class implemented**

```
public void add(Object anObject)
{
        ……..
        elements[tail] = anObject;
        tail++;
        size++;
}
```

- **How removeFirst works**

```
public Object removeFirst()
{
        ……..
        Object r = elements[head]
        head++;
        size--;
}
```

Object Locks

- *Each Object in Java has its associated implicit monitor*
- *A thread can lock a java object*
- *When another thread tries to acquire the lock it is blocked.*
- *Java uses synchronized methods for object locking*
- *By declaring a method as synchronized, we ensure that thread executes method to its completion.*
- *How to declare the method synchronized:*

```
public synchronized void add (Object obj) { }
public synchronized Object removeFirst () { }
```

The synchronized statement

- Syntax

```
synchronized(object-reference)
{
        statements-to-be-synchronized;
}
```

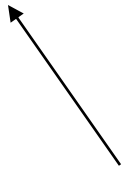
Deadlock

- ***When two threads have a circular dependence on a pair of synchronized objects.***
- ***Declaring add and remove methods synchronized ensures that threads executes them fully not partially. { Not Enough to ensure that program runs correctly}***
- **Example Consider the following code :**

```
if(!queue.isFull()) { queue.add(…….); i++; }
```

- **Move the check inside the add**

```
public synchronized void add(Object obj)
{
 while(queue.isFull())) wait for space
}
```



Can lead to deadlock

Inter Thread Communication

- **Methods Used for Inter Thread Communication**
  1. ***final void wait()*** → Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
  2. ***final void wait(long timeout)*** → Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
  3. ***final void notify()*** → Wakes up a single thread that is waiting on this object's monitor.
  4. ***final void notifyAll()*** → Wakes up all threads that are waiting on this object's monitor.

```
public class Queue
{
private Object[ ]  elements;
private int head;
private int tail;
private int size;

public Queue(int capacity){}
public synchronized Object removeFirst() throws interruptedException
 {
while(size() == 0) wait();
.....................
.....................
notifyAll();
 }
public synchronized void add(Object anotherObject) throws interruptedException
{
while(size == elements.length) wait();
.....................................
.....................................
notifyAll();
}
```

join method

- ***final void join() throws interruptedException***
- ***join() method allows the thread to wait until the thread on which it was called terminates.***
- ***Calling thread waits until the specified thread joins.***

Suspending , Resuming and Stopping Threads

- Prior to Java 2 we can use

  final void suspend();

  final void resume();

- In Java 2(we provide methods in the Thread class itself)

Example Suspending Resuming in Java 2

```
class SampleThread extends Thread
{
…….
boolean suspendFlag=false;
public void run()
{
try
{
syncronized(this) { while(suspendFlag) wait(); }
……..
}
catch(InterruptedException e) { }
}

void mySuspend() { suspendFlag = true; }

synchronized void myResume() { suspendFlag = false; }
} // end of thread class
```