Multithreaded Programming using Java Threads

CSC207 – Software Design

Slides are kindly provided by: 
Professor Rajkumar Buyya  
University of Melbourne, Australia  
http://www.buyya.com

Agenda

- Introduction
- Thread Applications
- Defining Threads
- Java Threads and States
  - Priorities
- Accessing Shared Resources
  - Synchronisation
- Assignment 1:
  - Multi-Threaded Math Server
- Advanced Issues:
  - Concurrency Models: master/worker, pipeline, peer processing
  - Multithreading Vs multiprocessing
A single threaded program

class ABC
{
    ....
    public void main(..)
    {
        ...
        ..
    }
}

A Multithreaded Program

Main Thread

Thread A

Thread B

Thread C

Threads may switch or exchange data/results
Single and Multithreaded Processes

threads are light-weight processes within a process

Modern Applications need Threads (ex1): Editing and Printing documents in background.
Multithreaded/Parallel File Copy

```c
reader()
{
    lock(buff[i]);
    read(src,buff[i]);
    unlock(buff[i]);
}
```

```c
writer()
{
    lock(buff[i]);
    write(src,buff[i]);
    unlock(buff[i]);
}
```

Cooperative Parallel Synchronized Threads

Multithreaded Server: For Serving Multiple Clients Concurrently

Client 1 Process

Client 2 Process

Server Process
Web/Internet Applications: Serving Many Users Simultaneously

Sockets/ PVM/MPI

Threads

Compilers

CPU

Sockets/ PVM/MPI

Threads

Compilers

CPU

Levels of Parallelism

Code-Granularity
Code Item
Large grain (task level)
Program

Medium grain (control level)
Function (thread)

Fine grain (data level)
Loop (Compiler)

Very fine grain (multiple issue)
With hardware
What are Threads?

- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.
- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

Java Threads

- Java has built in thread support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread start setPriority
  - yield run getPriority
  - sleep stop suspend
  - resume
- Java Garbage Collector is a low-priority thread.
Threading Mechanisms...

- Create a class that extends the Thread class
- Create a class that implements the Runnable interface

1st method: Extending Thread class

- Create a class by extending Thread class and override run() method:
  ```java
class MyThread extends Thread {
    public void run() {
      // thread body of execution
    }
  }
```
- Create a thread:
  ```java
  MyThread thr1 = new MyThread();
  ```
- Start Execution of threads:
  ```java
  thr1.start();
  ```
- Create and Execute:
  ```java
  new MyThread().start();
  ```
An example

class MyThread extends Thread {
    public void run() {
        System.out.println(" this thread is running ... ");
    }
}

class ThreadEx1 {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();
    }
}

2nd method: Threads by implementing Runnable interface

- Create a class that implements the interface Runnable and override run() method:

  class MyThread implements Runnable {
    ....
    public void run() {
      // thread body of execution
    }
  }

- Creating Object:
  MyThread myObject = new MyThread();
- Creating Thread Object:
  Thread thr1 = new Thread( myObject );
- Start Execution:
  thr1.start();
An example

class MyThread implements Runnable {
    public void run() {
        System.out.println(" this thread is running ... ");
    }
}

class ThreadEx2 {
    public static void main(String[] args) {
        Thread t = new Thread(new MyThread());
        t.start();
    }
}
A Program with Three Java Threads

- Write a program that creates 3 threads

```java
class A extends Thread {
    public void run() {
        for(int i=1;i<=5;i++) {
            System.out.println("From ThreadA: i= " + i);
        }
        System.out.println("Exit from A");
    }
}

class B extends Thread {
    public void run() {
        for(int j=1;j<=5;j++) {
            System.out.println("From ThreadB: j= " + j);
        }
        System.out.println("Exit from B");
    }
}
```
class C extends Thread
{
    public void run()
    {
        for(int k=1;k<=5;k++)
        {
            System.out.println("\t From ThreadC: k= "+k);
        }
        System.out.println("Exit from C");
    }
}

class ThreadTest
{
    public static void main(String args[])
    {
        new A().start();
        new B().start();
        new C().start();
    }
}

Run 1

raj@mundroo threads [1:76] java ThreadTest
From ThreadA: i= 1
From ThreadA: i= 2
From ThreadA: i= 3
From ThreadA: i= 4
From ThreadA: i= 5
Exit from A
From ThreadC: k= 1
From ThreadC: k= 2
From ThreadC: k= 3
From ThreadC: k= 4
From ThreadC: k= 5
Exit from C
From ThreadB: j= 1
From ThreadB: j= 2
From ThreadB: j= 3
From ThreadB: j= 4
From ThreadB: j= 5
Exit from B
Run2

- [raj@mundroo] threads [1:77] java ThreadTest
  - From ThreadA: i = 1
  - From ThreadA: i = 2
  - From ThreadA: i = 3
  - From ThreadA: i = 4
  - From ThreadA: i = 5
  - From ThreadC: k = 1
  - From ThreadC: k = 2
  - From ThreadC: k = 3
  - From ThreadC: k = 4
  - From ThreadC: k = 5

Exit from C
  - From ThreadB: j = 1
  - From ThreadB: j = 2
  - From ThreadB: j = 3
  - From ThreadB: j = 4
  - From ThreadB: j = 5

Exit from B
Exit from A

Process Parallelism

- int add (int a, int b, int & result)
- // function stuff
- int sub(int a, int b, int & result)
- // function stuff

MISD and MIMD Processing
Data Parallelism

- sort( int *array, int count)
- \(/\ldots../\)
- \(/\ldots../\)

```
 pthread-t, thread1, thread2;
```
```
 pthread-create(& thread1, sort, array, N/2);
 pthread-create(& thread2, sort, array, N/2);
 pthread-par(2, thread1, thread2);
```

**SIMD Processing**

Thread Priority

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (NORM_PRIORITY) and they are served using FCFS policy.
- Java allows users to change priority:
  - ThreadName.setPriority(intNumber)
    - MIN_PRIORITY = 1
    - NORM_PRIORITY=5
    - MAX_PRIORITY=10
class A extends Thread {
    public void run() {
        System.out.println("Thread A started");
        for(int i=1;i<=4;i++) {
            System.out.println("\t From ThreadA: i= "+i);
        }
        System.out.println("Exit from A");
    }
}

class B extends Thread {
    public void run() {
        System.out.println("Thread B started");
        for(int j=1;j<=4;j++) {
            System.out.println("\t From ThreadB: j= "+j);
        }
        System.out.println("Exit from B");
    }
}

class C extends Thread {
    public void run() {
        System.out.println("Thread C started");
        for(int k=1;k<=4;k++) {
            System.out.println("\t From ThreadC: k= "+k);
        }
        System.out.println("Exit from C");
    }
}

class ThreadPriority {
    public static void main(String args[]) {
        A threadA=new A();
        B threadB=new B();
        C threadC=new C();
        threadC.setPriority(Thread.MAX_PRIORITY);
        threadB.setPriority(threadA.getPriority()+1);
        threadA.setPriority(Thread.MIN_PRIORITY);
        System.out.println("Started Thread A");
        threadA.start();
        System.out.println("Started Thread B");
        threadB.start();
        System.out.println("Started Thread C");
        threadC.start();
        System.out.println("End of main thread");
    }
}
Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account.
  - Can the following operations be done at the same time on the same account?
    - Deposit()
    - Withdraw()
    - Enquire()
Shared Resources

- If one thread tries to read the data and another thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use “Synchronized” method:
  ```java
  public synchronized void update()
  {
      ...
  }
  ```
class InternetBankingSystem {
    public static void main(String[] args) {
        Account accountObject = new Account();
        Thread t1 = new Thread(new MyThread(accountObject));
        Thread t2 = new Thread(new YourThread(accountObject));
        Thread t3 = new Thread(new HerThread(accountObject));
        t1.start();
        t2.start();
        t3.start();
        // DO some other operation
        } // end main()
    } // end class InternetBankingSystem

class MyThread implements Runnable {
    Account account;
    public MyThread(Account s) { account = s; }
    public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable {
    Account account;
    public YourThread(Account s) { account = s; }
    public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable {
    Account account;
    public HerThread(Account s) { account = s; }
    public void run() { account.enquire(); }
} // end class HerThread

the driver: 3rd Threads sharing the same object

Shared account object between 3 threads
Monitor (shared object access): serializes operation on shared object

class Account {
   // the 'monitor'
   int balance;

   // if 'synchronized' is removed, the outcome is unpredictable
   public synchronized void deposit() {
      // METHOD BODY: balance += deposit_amount;
   }

   public synchronized void withdraw() {
      // METHOD BODY: balance -= deposit_amount;
   }

   public synchronized void enquire() {
      // METHOD BODY: display balance.
   }
}

Producer and Consumer Problem

- Classical multithread synchronization problem
  - two threads, the producer and the consumer, who share a common, fixed-size buffer.

- The producer’s job is to generate a piece of data and put it into the buffer.

- The consumer is consuming the data from the same buffer simultaneously.

- The problem is
  - to make sure that the producer will not try to add data into the buffer if it is full
  - that the consumer will not try to remove data from an empty buffer.
The solution for this problem involves two parts.

- The producer should wait when it tries to put the newly created product into the buffer until there is at least one free slot in the buffer.
- The consumer, on the other hand, should stop consuming if the buffer is empty.
Assignment 1: Multithreaded MathServer – Demonstrates the use of Sockets and Threads

- A Client Program
  - What is sqrt(10)?
  - sqrt 4.0
  - “2.0”

- A Client Program
  - What is sin(10)?
  - A Client Program in “C”
  - What is sin(10)?

- A Client Program in “C++”
  - What is sin(10)?
A Multithreaded Program

\[ p = \sin(x) + \cos(y) + \tan(z) \]

```java
/* MathThreads.java: A program with multiple threads performing concurrent operations. */
import java.lang.Math;
class MathSin extends Thread {
    public double deg;
    public double res;

    public MathSin(int degree) {
        deg = degree;
    }
    public void run() {
        System.out.println("Executing sin of " + deg);
        double Deg2Rad = Math.toRadians(deg);
        res = Math.sin(Deg2Rad);
        System.out.println("Exit from MathSin. Res = " + res);
    }
}
```
class MathCos extends Thread {
    public double deg;
    public double res;

    public MathCos(int degree) {
        deg = degree;
    }

    public void run() {
        System.out.println("Executing cos of "+deg);
        double Deg2Rad = Math.toRadians(deg);
        res = Math.cos(Deg2Rad);
        System.out.println("Exit from MathCos. Res = "+res);
    }
}

class MathTan extends Thread {
    public double deg;
    public double res;

    public MathTan(int degree) {
        deg = degree;
    }

    public void run() {
        System.out.println("Executing tan of "+deg);
        double Deg2Rad = Math.toRadians(deg);
        res = Math.tan(Deg2Rad);
        System.out.println("Exit from MathTan. Res = "+res);
    }
}

class MathThreads {
    public static void main(String args[]) {
        MathSin st = new MathSin(45);
        MathCos ct = new MathCos(60);
        MathTan tt = new MathTan(30);
        st.start();
        ct.start();
        tt.start();
        try {
            // wait for completion of all thread and then sum
            st.join();
            ct.join(); //wait for completion of MathCos object
            tt.join();
            double z = st.res + ct.res + tt.res;
            System.out.println("Sum of sin, cos, tan = "+z);
        } catch(InterruptedException IntExp) {
        }
    }
}
Run 1:
[raj@mundroo] threads [1:111] java MathThreads
Executing sin of 45.0
Executing cos of 60.0
Executing tan of 30.0
Exit from MathSin. Res = 0.7071067811865475
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Sum of sin, cos, tan = 1.7844570503761732

Run 2:
[raj@mundroo] threads [1:111] java MathThreads
Executing sin of 45.0
Executing tan of 30.0
Executing cos of 60.0
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Exit from MathSin. Res = 0.7071067811865475
Sum of sin, cos, tan = 1.7844570503761732

Run 3:
[raj@mundroo] threads [1:111] java MathThreads
Executing cos of 60.0
Executing sin of 45.0
Executing tan of 30.0
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Exit from MathSin. Res = 0.7071067811865475
Sum of sin, cos, tan = 1.7844570503761732

References

- Sun Java Tutorial – Concurrency: