Chapter 12

Concurrency – Java Threads

Concurrency in Java

All Java programs run in threads

When a Java application begins execution, a new (single) thread is created and **`main`** is called

A program becomes multithreaded if it constructs and starts additional threads of execution

Each thread has its own execution stack and instruction pointer

Concurrency in Java

The concurrent units in Java are `Thread` objects that include a `run` method (similar to a `main`)

The `run` method is inherited and overriden in subclasses of the `Thread` class

Code for `run` can execute concurrently with other such methods and with `main`

Java `Thread` Class Essentials

`run` – inherited and overridden in subclasses

`start` – calls `run`, after which control immediately returns to `start`

`yield` – pauses execution of the thread and puts it in the task ready queue, allowing other threads to execute

`sleep` – blocks execution of the thread for a specified amount of time

Java Thread Example

```
import java.util.Date;

public class MessageThread extends Thread {

  private String message;
  private static final int REPS = 10;
  private static final int DELAY = 1000;

  // Construct a thread object with message = m
  public MessageThread(String m) {
     message = g;
  }
```

Java Thread Example, continued

```
public void run() {
    try {
      for (int i = 1; i < REPS; i++) {
         Date now = new Date();
         System.out.println(now + " " + message);
         sleep(DELAY);
      }
    } catch (InterruptedException e) {}
  }
}
```

Java Thread Example, driver

```
public class MessageTest {

  public static void main(String[] args) {
    MessageThread t1 =
          new MessageThread("Thread 1");
       MessageThread t2 =
          new MessageThread("Thread 2");
       MessageThread t3 =
          new MessageThread("Thread 3");
    t1.start();
    t2.start();
    t3.start();
  }
}
```

Competition Synchronization with Java Threads

A method that includes the `synchronized` modifier disallows any other method from running on the object while it is being executed

Java associates a monitor with each object that has a synchronized method

If only a part of a method must be run as a critical section, just that part can be synchronized

Competition Synchronization with Java Threads

Communication between threads is provided by the `wait` and `notify` methods

- Defined in `Object` so all objects inherit them
- `wait` – suspends the thread until `notify` is called for the object on which `wait` is called
- `notify` – resumes one of the threads waiting on this object
- The `wait` method must be called in a loop

Shared Buffer in Java

```
public class DataBuf {
  private int[] queue;
  private int nextIn, nextOut,
              filled, qSize;

  public DataBuf(int size) {
    queue = new int[size];
    filled = 0;
    nextIn = nextOut = 1;
    qSize = size;
  }
```

Shared Buffer in Java, cont.

```
public synchronized void deposit(int item) {
  try {
    while (filled == qSize) {
       wait();
    }
    queue[nextIn] = item;
    nextIn = (nextIn + 1) % qSize ;
    filled++;
    notify();
  } catch (InterruptedException e) {}
}
```

Shared Buffer in Java, cont.

```
public synchronized int fetch() {
  int item = 0;
  try {
     while (filled == 0) {
       wait();
     }
     item = queue[nextOut];
     nextOut = (nextOut + 1) % qSize;
     filled--;
     notify();
  } catch(InterruptedException e) {}
  return item;
}
```

Producer in Java

```
public class Producer extends Thread {
  private DataBuf buffer;
  private String name;
  public Producer(DataBuf db, String n) {
    buffer = db;  name = n;
  }
  public void run() {
    int newItem = initialvalue;
    while (true) {
       buffer.deposit(newItem);
       // generate next newItem here
     }
  }
}
```

Consumer in Java

```
public class Consumer extends Thread {
  private DataBuf buffer;
  private String name;
  public Consumer(DataBuf db, String n) {
     buffer = db;  name = n;
  }
  public void run() {
    int item;
    while (true) {
       item = buffer.fetch();
       // process item here
    }
  }
}
```

Producer/Consumer Test Driver

```
public class Driver {
  public static void main(String[] args) {
     DataBuf buff1 = new DataBuf(10);
     Producer prod1 =
              new Producer(buff1, "Producer 1");
     Consumer[] cons =
              {new Consumer(buff1, "Consumer 1"),
                new Consumer(buff1, "Consumer 2"),
                new Consumer(buff1, "Consumer 3")};
     prod1.start();
     cons[0].start();
     cons[1].start();
     cons[2].start();
  }
}
```

Evaluation

Relatively simple, but effective

(C# threads are a bit more advanced)

C# Threads

Basic thread operations

- Any method can run in its own thread
- A thread is created by creating a **Thread** object
- Creating a thread does not start its concurrent execution - it must be requested through the **Start** method
- A thread can be made to wait for another thread to finish with **Join**
- A thread can be suspended with **Sleep**
- A thread can be terminated with **Abort**

C# Threads

Synchronizing threads

- The **Interlock** class (thread-safe increment/decrements, assignment)
- The **lock** statement (marks a critical section)
- The **Monitor** class (similar to lock)

```
class C {
      int[] q = new int[100];
      int i = 0;
      void Insert(int n) {
            lock (this) q[i++] = n;
      }
}
```

C# Threads

Evaluation

- An advance over Java threads, e.g., any method can run its own thread
- Thread termination is cleaner than in Java
- Synchronization is more sophisticated

Potential for Concurrency in Scheme

AND – parallelism

```
(f a1 a2 a3 a4)
```

- Create 4 processes to evaluate each argument concurrently
- Suspend `f` until all 4 processes are done
- Resume `f` when the 4 arguments are available