Multi-Threading

Threads

A thread is a unit of concurrent execution.

In Android a thread of execution results from an Activity forking into two or more concurrently running tasks.

Multiple threads can exist within the same process and share memory resources (on the other hand, processes have each its own process space).
Multi-Threading

Advantages of Multi-Threading

1. Threads share the process’ resources but are able to execute independently.

2. Applications generally remain responsive to user interaction (main thread runs UI, and slow tasks are sent to background threads).

3. Threading provides an useful abstraction of concurrent execution.

4. Particularly useful in the case of a single process that spawns multiple threads on top of a multiprocessor system. In this case real parallelism is achieved.

5. Consequently, a multithreaded program operates faster on computer systems that have multiple CPUs.
Multi-Threading

Disadvantages of Multi-Threading

1. Code tends to be more complex

2. Need to detect, avoid, resolve Deadlocks

Android’s Approach to Slow Activities

An application may involve a time-consuming operation, however we want the UI to be responsive to the user. Android offers two ways for dealing with this scenario:

1. Do expensive operations in a background service, using notifications to inform users about next step

2. Do the slow work in a background thread.

Android’s threading uses Handler objects and posting Runnable objects to the View.
Multi-Threading

Handler Class


When a process is created for your application, its main thread is dedicated to running a message queue that takes care of managing the top-level application objects (activities, intent receivers, etc) and any windows they create.

You can create your own threads, and communicate back with the main application thread through a Handler.

A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue.

Each Handler instance is associated with a single thread and that thread's message queue.

When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.
Multi-Threading

Handler Class

There are two main uses for a Handler:

1) to schedule messages and runnables to be executed as some point in the future; and

2) to enqueue an action to be performed on a different thread than your own.

Multi-Threading

Threads and UI

Warning
Background threads are not allowed to interact with the UI. Only the main process can access the activity’s view.
**Multi-Threading**

**Handler’s MessageQueue**

A secondary thread that wants to communicate with the main thread must request a message token using the `obtainMessage()` method.

Once obtained, the background thread can fill data into the message token and attach it to the Handler’s message queue using the `sendMessage()` method.

The Handler uses the `handleMessage()` method to continuously attend new messages arriving to the main thread. A message extracted from the process’ queue can either return some data to the main process or request the execution of runnable objects through the `post()` method.
### Multi-Threading

#### Using Messages

<table>
<thead>
<tr>
<th>Main Thread</th>
<th>Background Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>... Handler myHandler = new Handler(){</td>
<td></td>
</tr>
<tr>
<td>@Override</td>
<td></td>
</tr>
<tr>
<td>public void handleMessage(Message msg){</td>
<td></td>
</tr>
<tr>
<td>// do something with the message...</td>
<td></td>
</tr>
<tr>
<td>// update GUI if needed!</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>} //handleMessage</td>
<td></td>
</tr>
<tr>
<td>} //myHandler</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Thread backgJob = new Thread(new Runnable(){ |
@Override |
public void run(){ |
//...do some busy work here ... |
//get a token to be added to |
//the main's message queue |
Message msg = myHandler.obtainMessage(); |
... |
//deliver message to the |
//main's message-queue |
myHandler.sendMessage(msg); |
} //run |
} //Thread |
//this call executes the parallel thread |
backgroundJob.start(); |
...

### Using Post

<table>
<thead>
<tr>
<th>Main Thread</th>
<th>Background Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>... Handler myHandler = new Handler();</td>
<td></td>
</tr>
<tr>
<td>@Override</td>
<td></td>
</tr>
<tr>
<td>public void onCreate(Bundle savedInstanceState) {</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Thread myThread1 = new Thread(backgroundTask, &quot;backAlias1&quot;);</td>
<td></td>
</tr>
<tr>
<td>myThread1.start();</td>
<td></td>
</tr>
<tr>
<td>} //onCreate</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>//this is the foreground runnable</td>
<td></td>
</tr>
<tr>
<td>private Runnable foregroundTask</td>
<td></td>
</tr>
<tr>
<td>= new Runnable() {</td>
<td></td>
</tr>
<tr>
<td>@Override</td>
<td></td>
</tr>
<tr>
<td>public void run() {</td>
<td></td>
</tr>
<tr>
<td>// work on the UI if needed</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>} //backgroundTask</td>
</tr>
</tbody>
</table>

private Runnable backgroundTask |
= new Runnable() { |
@Override |
public void run() { |
... |
myHandler.post(foregroundTask); |
} //run |
} //backgroundTask |
Multi-Threading

Messages

To send a Message to a Handler, first invoke `obtainMessage()` to get the Message object out of the pool.

There are a few flavors of `obtainMessage()`, allowing you to just create empty Message objects, or ones populated with message identifiers and arguments.

Then, you send the Message to the Handler via its message queue, using one of the `sendMessage...()` family of methods, such as...

sendMessage Methods

- `sendMessage()` puts the message on the queue immediately
- `sendMessageAtFrontOfQueue()` puts the message at the front of the queue immediately (versus the back, as is the default), so your message takes priority over all others
- `sendMessageAtTime()` puts the message on the queue at the stated time, expressed in the form of milliseconds based on system uptime (`SystemClock.uptimeMillis()`)
- `sendMessageDelayed()` puts the message on the queue after a delay, expressed in milliseconds
Multi-Threading

Processing Messages
To process messages sent by the background threads, your Handler needs to implement

```
handleMessage()
```

which will be called with each message that appears on the message queue.

There, the handler can update the UI as needed. However, it should still do that work quickly, as other UI work is suspended until the Handler is done.

Example 1. Progress Bar
Assume the main thread will display a progress bar widget showing the progress of a slow background operation.

```xml
<LinearLayout
 xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:background="#ff0000ff">
    <TextView android:id="@+id/TextView01"
 android:layout_height="wrap_content"
 android:layout_width="fill_parent"
 android:text="Working"
 android:textSize="18sp" android:textStyle="bold" />
    <ProgressBar android:id="@+id/progress"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content" />
</LinearLayout>
```
Multi-Threading

Example 1. Progress Bar

```java
package cis493.threads;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.widget.ProgressBar;
import android.widget.TextView;

public class ThreadDemo1ProgressBar extends Activity {
    ProgressBar bar;
    TextView msgWorking;
    boolean isRunning = false;

    Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            bar.incrementProgressBy(5);
            if (bar.getProgress() == bar.getMax()) {
                msgWorking.setText("Done");
                bar.setVisibility(View.INVISIBLE);
            } else {
                msgWorking.setText("Working... " + bar.getProgress());
            }
        }
    } // handleMessage
}
```
**Example 1. Progress Bar**

```java
@override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    bar = (ProgressBar) findViewById(R.id.progress);
    bar.setMax(100);
    msgWorking = (TextView) findViewById(R.id.TextView01);
}
```

```java
public void onStart() {
    super.onStart();
    bar.setProgress(0);
    Thread background = new Thread(new Runnable() {
        public void run() {
            try {
                for (int i = 0; i < 20 && isRunning; i++) {
                    Thread.sleep(1000);
                    handler.sendMessage(handler.obtainMessage());
                }
            } catch (Throwable t) { // just end the background thread
                isRunning = true;
                background.start();
            }
        }
    });
    isRunning = true;
    background.start();
} // onStart
```
Example 1. Progress Bar

```java
public void onStop() {
    super.onStop();
    isRunning = false;
}
``` // ThreadDemo1ProgressBar

Example 2. Using Handler post(...) Method
We will try the same problem presented earlier (a slow background task and a responsive foreground UI) this time using the posting mechanism to execute foreground runnables.
Multi-Threading

Example 2. Using Handler post(...) Method

```java
// using Handler post(...) method to execute foreground runnables

package cis493.threads;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.text.Editable;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

public class ThreadsPosting extends Activity {

    ProgressBar myBar;
    TextView lblTopCaption;
    EditText txtBox1;
    Button btnDoSomething;
    int accum = 0;
    long startingMills = System.currentTimeMillis();
    String PATIENCE = "Some important data is been collected now. Please be patient. ";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        lblTopCaption = (TextView) findViewById(R.id.lblTopCaption);
        txtBox1 = (EditText) findViewById(R.id.txtBox1);
        btnDoSomething = (Button) findViewById(R.id.btnDoSomething);
        myBar = (ProgressBar) findViewById(R.id/myBar);
    }

    public void DoSomething(View v) {
        Accumulator accum = new Accumulator();
        accum.doSomething();
    }

    public class Accumulator extends Handler implements Runnable {
        public void run() {
            accum = AccumulatorAccumulator.getInstance();
            accum.doSomething();
        }
    }
}
```
Multi-Threading

Example 2. Using Handler `post(...)` Method

```java
Handler myHandler = new Handler();

@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    lblTopCaption = (TextView)findViewById(R.id.lblTopCaption);
    myBar = (ProgressBar) findViewById(R.id.myBar);
    myBar.setMax(100);
    txtBox1 = (EditText) findViewById(R.id.txtBox1);
    txtBox1.setHint("Foreground distraction. Enter some data here");
    btnDoSomething = (Button) findViewById(R.id.btnDoSomething);

    btnDoSomething.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Editable txt = txtBox1.getText();
            Toast.makeText(getBaseContext(), "You said >> " + txt, 1).show();
        //onClick
    }); //setOnClickListener
    //onCreate
}

@Override
protected void onStart() {
    super.onStart();
    // create background thread were the busy work will be done
    Thread myThread1 = new Thread(backgroundTask, "backAlias1");
    myThread1.start();
    myBar.incrementProgressBy(0);
}
    // this is the foreground "Runnable" object responsible for GUI updates
    private Runnable foregroundTask = new Runnable() {
        @Override
        public void run() {
            try {
                int progressStep = 5;
                lblTopCaption.setText(PATIENCE + "\nTotal sec. so far: " +
                        (System.currentTimeMillis() - startingMills) / 1000 ) +
                        myBar.incrementProgressBy(progressStep);
                accum += progressStep;
                if (accum >= myBar.getMax()){
                    lblTopCaption.setText("Background work is OVER!");
                    myBar.setVisibility(View.INVISIBLE);
                }
            }
        } catch (Exception e) {
        e.printStackTrace();
    }
    //run
}; //foregroundTask
```
13. Android – Multi-Threading

Multi-Threading

Example 2. Using Handler post(...) Method

```java
//this is the "Runnable" object that executes the background thread
private Runnable backgroundTask = new Runnable () {
    @Override
    public void run() {
        //busy work goes here...
        try {
            for (int n=0; n<20; n++) {
                //this simulates 1 sec. of busy activity
                Thread.sleep(1000);
                //now talk to the main thread
                myHandler.post(foregroundTask);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    //run
    };
//backgroundTask
};//ThreadsPosting
```

Thread States

Try to match classic Java Thread states into Android’s Thread-States

<table>
<thead>
<tr>
<th>Thread State</th>
<th>BLOKED</th>
<th>The thread is blocked and waiting for a lock.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread State</td>
<td>NEW</td>
<td>The thread has been created, but has never been started.</td>
</tr>
<tr>
<td>Thread State</td>
<td>RUNNABLE</td>
<td>The thread may be run.</td>
</tr>
<tr>
<td>Thread State</td>
<td>TERMINATED</td>
<td>The thread has been terminated.</td>
</tr>
<tr>
<td>Thread State</td>
<td>TIMED_WAITING</td>
<td>The thread is waiting for a specified amount of time.</td>
</tr>
<tr>
<td>Thread State</td>
<td>WAITING</td>
<td>The thread is waiting.</td>
</tr>
</tbody>
</table>

10/11/2009
Multi-Threading

Questions?