CS434/534: Topics in Networked (Networking) Systems

Mobile System: Android (Single App/Process)

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Admin.

- Project planning
  - Start: Apr. 5
    - A meeting w/ instructor; Due: team, potential topics
  - Checkpoint I: Apr. 13 (updated; Thursday)
    - Initial ideas; survey of related work; feedback from instructor/TF
  - Checkpoint II: Apr. 20 (updated)
    - Initial design; feedback from instructor/TF
  - Checkpoint III: Apr. 27 (updated)
    - Design refined; key components details
  - Final report: May 4
    - No more than 6 page write up
Recap: TinyOS (Sensors)

- Simple clean software framework
  - component
  - interface
  - configurations

- Aggregation (framework)
  - an app (configuration) at a time, linking only necessary components

- Execution model
  - one for event
  - one for task

```c
interface ADC {
    async command result_t getdata();
    async command result_t getContinuousData();
    event result_t dataReady(uint 16_t data);
}

configuration SenseTask {
    // this module does not provide any interfaces
}
implementation {
    components Main, SenseTaskM, LedsC,
    TimerC, DemoSensorC as Sensor;
    Main.StdControl -> TimerC;
    Main.StdControl -> Sensor;
    Main.StdControl -> SenseTaskM;
    SenseTaskM.Timer ->
        TimerC.Timer[unique("Timer")];
    SenseTaskM.ADC -> Sensor;
    SenseTaskM.Leds -> LedsC;
}
```
Recap: Mobile UI Framework

- Limited screen real estate
  - one thing at a time

- Limited resources: more dynamic system management on app life cycle
  - give app chances to adapt, better mem management
Recap: Java ME

- To accommodate heterogeneous mobile devices, define **configurations and profiles**
- The Mobile Information Device Profile (MIDP) provides a simple, representative programming framework

Lifecyle callbacks
- startApp
- pauseApp
- destroyApp

```
d=Display.getDisplay(this)
d.setCurrent(disp)
```

```
d.addCommand()
d.setCommandListener()
```

A set of command

Command listener

Current Displayable
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorldMIDlet extends MIDlet
               implements CommandListener {

    private Command exitCommand;
    private Display display;
    private TextBox t;

    public HelloWorldMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.EXIT, 2);
        t = new TextBox("CS434", "Hello World!", 256, 0);
        t.addCommand(exitCommand);
        t.setCommandListener(this);
    }

    public void startApp() { display.setCurrent(t); }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}

HelloWorldMIDlet.java
Recap: General Mobile GUI App Structure

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite

Display

Display Composite

Display

Display

Display Composite

Display

Display

Display

Display

Display Composite

Display

Display

Display Composite

Display

Display

Display Composite

Display

Display

Display
Mobile GUI App Workflow: Handle Events

App lifecycle callbacks/custom
-start
-pause
-...

App

Display Composite

Display

Display Composite

Display

Display Composite

Display

Display

Event Handler

Event Handler

Data/Model

Data/Model
Recap

- Key design points (so far) for mobile GUI app framework
  - How to specify app life cycle customization
  - How to specify display
  - How to specify event scheduling
    - How to link event, display, handler, data
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
  - Sora
  - TinyOS
  - Java ME
  - Android
Android

- A mobile OS based on Linux
  - Customized Linux kernel 2.6 and 3.x (Android 4.0 onwards), e.g.,
    - default no X Windows, not full set of GNU libs
  - Apply OS concepts for mobile contexts, e.g.,
    - each app is considered one Linux user
    - by default, every app runs in its own Linux process
    - each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
  - New key components, e.g.,
    - binder for IPC
    - power management wakelock
Android Architecture

**System Apps**
- Dialer
- Email
- Calendar
- Camera
- ...

**Java API Framework**
- Content Providers
- View System
- Activity
- Location
- Package
- Notification
- Resource
- Telephony
- Window

**Native C/C++ Libraries**
- Webkit
- OpenMAX AL
- Libc
- Media Framework
- OpenGL ES
- ...

**Android Runtime**
- Android Runtime (ART)
- Core Libraries

**Hardware Abstraction Layer (HAL)**
- Audio
- Bluetooth
- Camera
- Sensors
- ...

**Linux Kernel**
- Drivers
  - Audio
  - Binder (IPC)
  - Display
  - Keypad
  - Bluetooth
  - Camera
  - Shared Memory
  - USB
  - WIFI

**Power Management**
Android Tools

- Android SDK Manager (android)
- Android emulator
- Android debug bridge (adb) can connect to an Android device and start a shell on the device

Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
  - Sora
  - TinyOS
  - Java ME
  - Android
    - Platform overview
    - Basic app/view/event handling
App lifecycle callbacks/custom
- start
- pause
- ...

How to specify the customized callbacks: extend Activity class

How to link the callbacks defined in view to listener/controller: View.set…Listener()

Application Framework (Android): Key Concepts

- Activity
- View/ViewGroup (Layout)
- External resources
Activity

- A single, focused thing that the user can do

- A typical app consists of multiple activities (why?)

- Typically organized as a Stack
  - Top Activity is visible
  - Other activities are stopped
  - Back button to traverse the Activity Stack
Simple Activity Example

- Create Hello434
Activity: Manifest File

- To facilitate launching and managing Activities, each activity is announced in a manifest file.

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.example.heliotest">
  <application>
    android:label="@string/app_name">
    <activity>
      android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Instead of a hardcoded string in code, defines in res/strings.

Manifest the activity.
Android Project Resources

- AndroidManifest.xml
- assets
- bin
- default.properties
- gen
- res
  - drawable-hdpi
    - icon.png
  - drawable-ldpi
    - icon.png
  - drawable-mdpi
    - icon.png
  - layout
    - main.xml
  - layout-large
    - main.xml
  - layout-small
    - main.xml
  - values
  - src
- assets
- bin
- default.properties
- gen
- res
  - drawable-hdpi
  - drawable-ldpi
  - drawable-mdpi
  - layout
  - layout-large
  - layout-small
  - values
  - values-en-UK
  - values-en-US
  - values-fr
  - src
- strings.xml
Activity: Example

// MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // savedInstanceState holds any data that may have been saved
        // for the activity before it got killed by the system (e.g.
        // to save memory) the last time

        super.onCreate(savedInstanceState);
        setContentView(…); // set a View
    }
}

Android Activity Life Cycle
Simple Activity Example

- Create Hello434
- Show life cycle
View

- A view component is a building block for user interface components.

- Two types of views
  - Leaf: TextView, EditText, Button, Form, TimePicker... ListView
  - Composite (ViewGroup): LinearLayout, RelativeLayout, ...

View

https://developer.android.com/guide/topics/ui/overview.html
Programmatic Definition of View

// MainActivity.java
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        // savedInstanceState holds any data that may have been saved
        // for the activity before it got killed by the system (e.g.
        // to save memory) the last time

        super.onCreate(savedInstanceState);

        TextView tv = new TextView(this);
        tv.setText("Hello!“);
        setContentView(tv);
    }
}
Simple Activity Example

- Create Hello434
- Show life cycle
- Use programmatic definition of view
Define View by XML

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
</LinearLayout>
```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
}

TextView myTextView = (TextView)findViewById(R.id.myTextView);

TextView myTextview = (TextView)findViewById(R.id.myTextview);

main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/myTextview"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello World, HelloWorld"
    />
</LinearLayout>
External Resources

- Compiled to numbers and included in R.java file

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>res/anim/</td>
<td>XML files for frame-by-frame animation</td>
</tr>
<tr>
<td>res/drawable/</td>
<td>images compiled and optimized</td>
</tr>
<tr>
<td>res/layout/</td>
<td>XML files for screen layouts</td>
</tr>
<tr>
<td>res/values/</td>
<td>compiled XML files into different resource</td>
</tr>
<tr>
<td>res/xml/</td>
<td>arbitrary XML files</td>
</tr>
<tr>
<td>res/raw/</td>
<td>raw, uncompilled files</td>
</tr>
</tbody>
</table>
Linking Views and Handlers/Controllers

- onKeyDown, onKeyUp
- onTrackBallEvent
- onTouchEvent

```
myEditText.setOnKeyListener(new OnKeyListener() {
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (event.getAction() == KeyEvent.ACTION_DOWN)
            if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
                ...
                return true;
            }
        return false;
    }
});
```

```
registerButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View arg0) {....}
});
```
public class TipCalcActivity extends Activity {

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

        Button calc = (Button) findViewById(R.id.calculate);
                calc.setOnClickListener(
                    new View.OnClickListener() {
                        @Override
                        public void onClick(View view) {
                            ...
                        }
                    });
    }
}
Example: TipCalc

Handler:

```java
@Override
public void onClick(View arg0) {
    EditText amountText =
        (EditText) findViewById(R.id.amount_value);

    // get input
    double amt = Double.parseDouble(amountText.getText().toString());

    // compute output
    double tipD = amt * 0.15;

    // set UI
    String tipT = String.format("%.2f", tipD);
    TextView tipText = (TextView) findViewById(R.id.tip_value);
    tipText.setText(tipT);
}
```
Simple Activity Example

- Create Hello434
- Show life cycle
- Use programmatic definition of view
- Add event handling
Summary: Most Basic Android UI App Software Concepts

- Activity

- UI (View/ViewGroup)
  - External definition of views in XML
  - findViewById() to reduce coupling

- Link view events to event handlers
  - set...Listener()

- Other concepts such as fragments which we skip
Outline

- Admin and recap
- Mobile/wireless development framework
  - GNURadio
  - Sora
  - TinyOS
  - Java ME
  - Android
    - Platform overview
    - Basic concepts
      - Activity, View, External Resources, Listener
    - Execution model and inter-thread communications
      - Handler, AsyncTask
Android Execution Model

- When an application component starts and the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution.
- The very important thread is also called the "main" thread:
  - in charge of dispatching events to the appropriate user interface widgets, including drawing events.
  - interacts with components from the Android UI toolkit.

Event Handler Execution

- Event handler executed by the main/UI thread

http://www.java2s.com/Open-Source/Android/android-core/platform-frameworks-base/android/os/Looper.java.htm
Event Handler and Responsiveness

- Event handler blocks events in the msg queue from being processed => slow running handler leads to no UI response

Responsiveness: Numbers (Nexus One)

- ~5-25 ms - uncached flash reading a byte
- ~5-200+(!) ms - uncached flash writing tiny amount
- 100-200 ms - human perception of slow action
- 108/350/500/800 ms - ping over 3G. varies!
- ~1-6+ seconds - TCP setup + HTTP fetch of 6k over 3G
Event Handler and ANR

- Android system detects no response
  - Main thread ("event"/UI) does not respond to input in 5 sec
Discussion

ɐ What are some events that may take a while to be processed?
   ▪ Time consuming loading process, e.g., slow onCreate
   ▪ Heavy computation, e.g., voice recognition, update map display
   ▪ Networking access
   ▪ ...

ɐ What are some design options if an event may take a while to be processed?
Typical Design Guidelines

- Notify user
  - E.g., progress bar, progress dialog
  - A splash screen

- If possible, non-blocking, incremental update UI
  - E.g., gradual add items to map

- Whenever possible, release UI thread ASAP
  - Keep event handler simple
  - Post heavy processing off the UI thread
Example: Background Thread

- Use a background thread to do the task
- Background thread behavior controlled by state
- State controlled by event handler
public class LoadingScreen extends Activity implements Runnable {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.load);
        // start a new thread to load
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run(){
        longRunningTask();
        setContentView(R.layout.main);
    }

    ...
Background Thread vs UI Thread

Problem:

- Background thread and UI thread are running concurrently and may have race conditions if they modify UI simultaneously (e.g., UI switches to a different orientation)

- A major sin of programming: concurrency bugs
Solution

- Background thread does not directly modify UI: send msg to UI thread, who processes the msg

- Android Methods
  - `Activity.runOnUiThread(Runnable)`
  - `View.post(Runnable)`
  - `View.postDelayed(Runnable, long)`
Example

public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            // a potentially time consuming task
            final Bitmap bitmap = processBitMap("image.png");

            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            }); // post
        } // run
    }).start();
}
Android Handler

- Android’s mechanism 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
  - A handler is bound to the thread / message queue of the thread that creates it
  - from that point on, it will deliver messages and runnables to that message queue
  - That thread processes msgs
Android Handler
Using Handler: Examples

There are two main uses for a Handler

- to schedule messages and runnables to be executed as some point in the future
  - `postDelayed(Runnable, delayMillis)`

- to enqueue an action to be performed on a different thread than your own.
  - `post(Runnable)`
public class MyActivity extends Activity {

[...]

// Need handler for callbacks to the UI thread
final Handler mHandler = new Handler();

// Create runnable task to give to UI thread
final Runnable mUpdateResultsTask = new Runnable() {
    public void run() {
        updateResultsInUI();
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

[...]
protected void startLongRunningOperation() {

    // Fire off a thread to do some work that we shouldn't do directly in the UI thread
    Thread t = new Thread() {
        public void run() {
            mResults = doSomethingExpensive();
            mHandler.post(mUpdateResultsTask);
        }
    };
    t.start();
}

private void updateResultsInUi() {

    // Back in the UI thread -- update our UI elements based on the data in mResults
    . . .
}
public class LoadingScreen extends Activity implements Runnable {

    private Handler mHandler = new Handler(); // UI handler

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

        // start a new thread to load
        Thread thread = new Thread(this);
        thread.start();
    }

    public void run() {
        longTask();
        mHandler.post(mSetFinalViewTask);
    }

    private Runnable mSetFinalViewTask = new Runnable() {
        public void run() {
            setContentView(R.layout.main);
        }
    };
}
private Handler mHandler = new Handler(); // UI handler
private Runnable longTask = new Runnable() {

    // processing thread
    public void run() {
        while (notFinished) {
            // doSomething

            mHandler.post(taskToUpdateProgress);
        }

        // mHandler.post(taskToUpdateFinalResult)
    }

    Thread thread = new Thread(longTask);
    thread.start();
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) { // on some background thread
        int count = urls.length; long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) { // on UI thread!
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) { // on UI thread!
        showDialog("Downloaded " + result + " bytes");
    }
}

new DownloadFilesTask().execute(url1, url2, url3); // call from UI thread!