Lecture 31 — Android IV

Jeff Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

December 29, 2014

Part I

Networking Basics

Android Networking

One more lecture on advanced Android topics: networking.

Not necessary for the labs, but still examinable.

Network Permissions

To access the network, permissions need to be assigned:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Network Availability

Good practice: check availability

```
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
    ... // Do useful work
    }
```

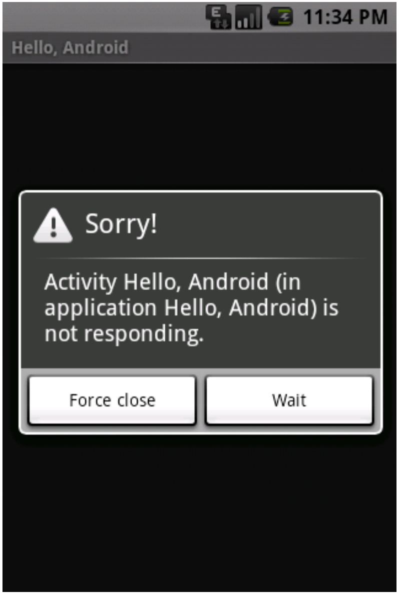
Networking

Networks are inherently unreliable.

They have unknown latency / bandwidth.

If we do network operations in the UI thread, we might get...

Not Responding


11:34 PM

Hello, Android

Sorry!

Activity Hello, Android (in application Hello, Android) is not responding.

(From linuxtopia.org)

Background Task

The not responding dialog doesn’t appear because the program is hung.

It is doing a network operation and not redrawing the screen.

Solution: do work in the background with an `AsyncTask`.

AsyncTask

```
private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
        @Override
        protected String doInBackground(String... urls) {

            // params comes from the execute() call: params[0] is the url.
            try {
                return downloadUrl(urls[0]);
            } catch (IOException e) {
                return "Unable to retrieve web page. URL may be invalid.";
            }
        }

        // onPostExecute displays the results of the AsyncTask.
        @Override
        protected void onPostExecute(String result) {
            textView.setText(result);
       }
    }
```

URL Downloader Task

This is a private class inside a `MainActivity` but it could be defined as an inner class.

Like a collection, such as `List`, the `AsyncTask` takes parameter types inside angle brackets for:

```
doInBackground
onProgressUpdate
onPostExecute.
```

URL Downloader Task

AsyncTask execution goes through four stages:

1. `onPreExecute()`
2. `doInBackground(Params...)`
3. `onProgressUpdate(Progress...)`
4. `onPostExecute(Result)`

Starting Tasks

To actually execute the Download Webpage task, use `new DownloadWebpageTask().execute(url);`

A task can be executed only once; to do something again, create another instance.

Cancelling Tasks

A task can be cancelled while it is running by calling `cancel(boolean)`.

This only makes `isCancelled()` return true.

The `doInBackground` method should check and see if the task has been cancelled.

Instead of `onPostExecute`, `onCancelled` runs.

Excerpt from Networking Code

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setReadTimeout(10000 /* milliseconds */);
conn.setConnectTimeout(15000 /* milliseconds */);
conn.setRequestMethod("GET");
conn.setDoInput(true);
// Starts the query
conn.connect();
int response = conn.getResponseCode();
```

Full code in the lecture notes.

Using the Network

`HttpURLConnection` is the key to making the connection.

Data can be of any type.

Not necessary to know in advance the length of the data.

HttpsURLConnection

Uses of this class follow a pattern:

1. Obtain a new `HttpURLConnection`.
2. Prepare the request.
3. Optionally upload a request body.
4. Read the response.
5. Disconnect.

HttpsURLConnection + SSL

Calling `openConnection()` on a URL with the "https" (HTTP with SSL, security) scheme will return an `HttpsURLConnection`.

We are not going to cover this.

For more detail and including things like posting content and authentication, take a look at the Android guidelines.

Part II

Battery Life

Battery Usage

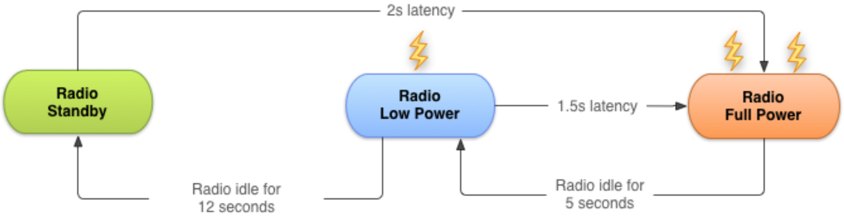
Other than the screen, the next biggest user of battery is likely the wireless radio.

The radio for a typical 3G divide has three states:

1. Full Power
2. Low Power
3. Standby

Radio State Transitions

Transition from one state to another is not instant.



Battery Usage

Creating a new network connection puts the radio in the full power state.

A 1-second transfer is followed by:
- 5 seconds of “tail time” in the high power state
- 12 seconds in the low power state;

Then the radio returns to a standby state.

Total “on” time: 18 seconds.

Unbundled Data

Transfer data for 1 second every 18 seconds.

Radio never goes into standby $\rightarrow$ battery drain.

Out of every 60 seconds:
  18 will be in the high power state;
  42 will be in the low power state.

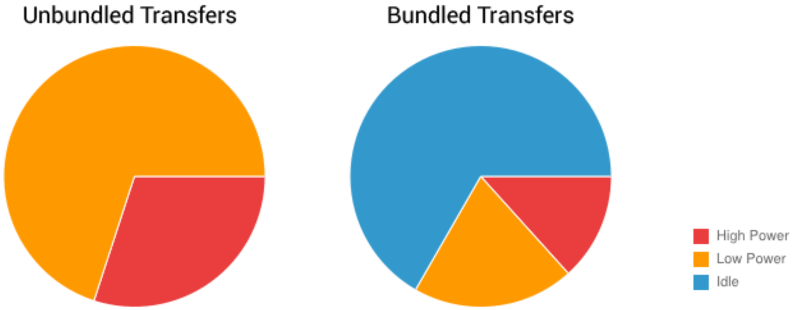
Bundled Data

Idea: bundle data (do transfers in bulk)

3 consecutive seconds of transfer means:
- 8 seconds in the high power state
- 12 seconds in the low power state.

40 seconds in the idle state!

Bundled vs. Unbundled



Part III

Cloud Sync

Cloud Sync

Sync your app with the cloud so data is not lost.

Even if the user reinstalls the app or changes device.

Google provides backup API for storing a small amount of data.

Cloud Sync

Register for the service with Google.

Then implement Backup Agent:

```
<application android:label="MyApp"
             android:backupAgent="TheBackupAgent">
    ...
    <meta-data android:name="com.google.android.backup.api_key"
    android:value="ABcDe1FGHij2KlmN3oPQRs4TUvW5xYZ" />
    ...
</application>
```

Backup Agent

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.FileBackupHelper;


public class TheBackupAgent extends BackupAgentHelper {
   // The name of the SharedPreferences file
   static final String HIGH_SCORES_FILENAME = "scores";

   // A key to uniquely identify the set of backup data
   static final String FILES_BACKUP_KEY = "myfiles";

   // Allocate a helper and add it to the backup agent
   @Override
   void onCreate() {
       FileBackupHelper helper = new FileBackupHelper(
             this, HIGH_SCORES_FILENAME);
       addHelper(FILES_BACKUP_KEY, helper);
   }
}
```

Cloud Sync

This `BackupAgentHelper` takes backups of the user's high scores file.

What if we used `SharedPreferences` instead of a file?

Backup Agent

```
import android.app.backup.BackupAgentHelper;
import android.app.backup.SharedPreferencesBackupHelper;

public class TheBackupAgent extends BackupAgentHelper {
    static final String PREFS_DISPLAY = "displayprefs";
    static final String PREFS_SCORES = "highscores";

    // An arbitrary string used within the BackupAgentHelper implementation to
    // identify the SharedPreferencesBackupHelper's data.
    static final String MY_PREFS_BACKUP_KEY = "myprefs";

    // Simply allocate a helper and install it
    void onCreate() {
        SharedPreferencesBackupHelper helper =
                new SharedPreferencesBackupHelper(
                    this, PREFS_DISPLAY, PREFS_SCORES);
        addHelper(MY_PREFS_BACKUP_KEY, helper);
    }
}
```

Backup Semantics

To request a backup, create an instance of the `BackupManager`.

Call its `dataChanged()` method.

If you call `dataChanged()` more than once before the backup actually takes place, the backup will occur only once.

Restoring from backup happens automatically when the user reinstalls the application.

Can force it with `requestRestore()`.

Sync Conflicts

It's possible that when you save data to the cloud you end up with conflicts.

Some simple approaches to fixing it:

- Strategy 1: Newer is better.
- Strategy 2: Value Judgement.
- Strategy 3: Merge.

Sync Conflicts

These strategies work if the conflict and data are simple, but we might also have some more complex situations.

If we are tracking something important like money, choosing the higher of the two values is an incorrect solution.

Consider the following scenario where we just store the total:

Sync Conflicts

1. Starting condition: the user has 0 coins on Device A, 0 on Device B.
2. Player collects 10 coins on A.
3. Player collects 15 coins on B.
4. Device B saves.
5. Device A saves - conflict detected.
6. Conflict resolution: choose the largest of the two.

Error occurred: player collected 25 coins but the value of 15 was chosen, so the user has “lost” 10 coins.

Sync Conflicts

Idea: send the delta instead of the values (e.g. “+10”)

Android will send only the most recent update if network connectivity is not available.

Imagine that the user collected 5 coins on A while on an airplane (network off).

Then in another session, collected another 5 coins.

When the synchronization occurs, only the second update will be sent, so only 5 coins will be added to the user’s total.

Still incorrect.

Sync Conflicts

Solution: store sub-totals per device.

Have a separate “account” for each device.

When the user collects 10 coins on device A, write it into a value for coins collected on A.

Total: simply sum up the coins collected on A and B.