CSE 486/586 Distributed Systems
Android Programming

Steve Ko
Computer Sciences and Engineering
University at Buffalo

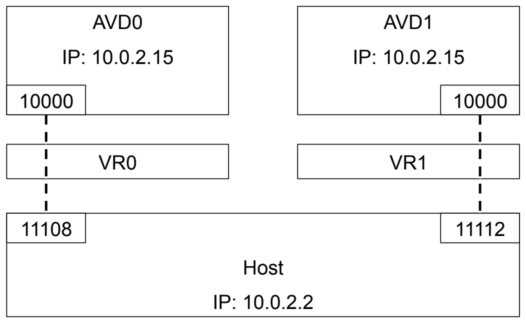
Recap

- What to put on top of physical networks?
  - Layers providing survivability
- Where to put functionalities?
  - Fate-sharing & end-to-end arguments
  - IP layer doesn't provide much
  - TCP handles most of the survivability issues
- TCP & UDP: the two transport protocols of the Internet
- What interface do applications see?
  - Socket API

Today

- Basic Android programming interleaved with a review of PA1
- Mainly programming model and components

The Hack: Emulator Port Forwarding



Three Most Important Things

- Read the documentation.
  - You will not be able to do anything without reading the documentation.
  - Learn how to use the APIs.
  - Learn how to use the constructs, e.g., AsyncTask, Messenger, etc.
- Do it; write your code.
  - No learning without doing
- Learn how to debug.
  - Using LogCat, DDMS, etc.

Android Programming Model

- No main()
- Four main components: Activity, Service, ContentProvider, BroadcastReceiver
  - You need to implement at least one of them to write an Android app.
- Event-driven
- Permissions
  - For certain APIs, you need to request permissions in AndroidManifest.xml.
  - These APIs are called protected APIs or sensitive APIs
  - Many permissions, e.g., internet, external storage, etc.

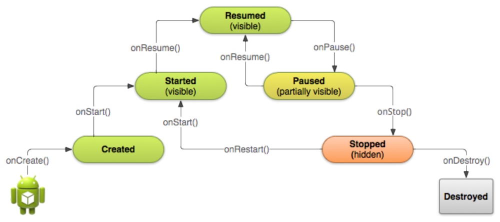
What? No main()?

- There is a main()! It's just that it's hidden.
- Zygote starts at boot.
- Launcher sends a message to start an activity.
- Zygote forks a new VM instance that loads ActivityThread.
  - ActivityThread has the real main() for an app.
- ActivityThread calls the app's onCreate(), onStart(), etc.

Example - Activity

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Example - Activity



Declare in AndroidManifest.xml

```
<manifest ... >
  ...
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application>
</manifest>
```

CSE 486/586 Administrivia

- PA 2 will be out by the end of this week.
- Please use Piazza; all announcements will go there.
- Please come to my office during the office hours!
  - Give feedback about the class, ask questions, etc.

Services

- A service runs in the background with no UI for long-running operations.
  - Playing music, sending/receiving network messages, …
  - Subclass of android.app.Service
- Started service
  - A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
- Bound service
  - A service is "bound" when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

How to Write a Service

- Declare in AndroidManifest.xml
- Implement necessary methods in *Service*

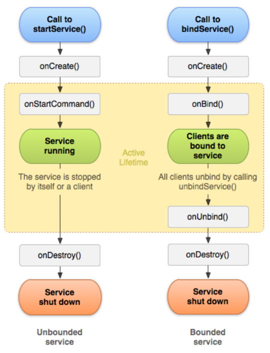
Declare in AndroidManifest.xml

```
<manifest ... >
  ...
  <application ... >
     <service android:name=".ExampleService" />
     ...
  </application>
</manifest>
```

Necessary Methods

- onStartCommand()
  - The system calls this method when another component, such as an activity, requests that the service be started, by calling startService().
- onBind()
  - The system calls this method when another component wants to bind with the service (such as to perform RPC), by calling bindService().
- onCreate()
  - The system calls this method when the service is first created, to perform one-time setup procedures (before it calls either onStartCommand() or onBind()).
- onDestroy()
  - The system calls this method when the service is no longer used and is being destroyed.

Service Lifecycle



Content Providers

- A content provider provides a table view of data.
- If you write a content provider, any client application with the permission can enter/read/update/delete data items in your content provider.
- A client application (that uses your content provider) uses *ContentResolver* to interact with your content provider.
- You need to extend *ContentProvider* and implement necessary methods.

How a Client Interacts

- Table identification → URI (android.net.Uri)
  - E.g., content://user_dictionary/words
- Insert
  - public final Uri ContentResolver.insert (Uri url, ContentValues values)
- Update
  - public final int ContentResolver.update (Uri uri, ContentValues values, String where, String[] selectionArgs)
- Query
  - public final Cursor ContentResolver.query (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
- Delete
  - public final int ContentResolver.delete (Uri url, String where, String[] selectionArgs)

How to Write a Content Provider

1. Declare in AndroidManifest.xml
2. Define a URI that client apps will use
3. Define permissions
4. Implement necessary methods in *ContentProvider*
5. When implementing *ContentProvider*, use either the Android file system or SQLite as the actual data storage.

Declare in AndroidManifest.xml

```
<manifest ... >
 ...
 <application ... >
    <provider android:name=".ExampleProvider" />
    ...
 </application>
</manifest>
```

Defining a URI

- Typical format
  - content://<authority>/<table name>
  - Authority: a global (Android-wide) name for the provider
    - » E.g., edu.buffalo.cse.cse486.proj1.provider
  - Table name: the name of a table that the provider exposes
    - » Note: a provider can expose more than one table.
- Should be added to AndroidManifest.xml
  - E.g., <provider android:authorities="edu.buffalo.cse.cse486.proj1.provider" …>…</provider>

Define Permissions

- Should define permissions (for others) in AndroidManifest.xml
- android:permission: Single provider-wide read/write permission.
  - E.g., <provider android:permission="edu.buffalo.cse.cse486.proj1.provider.permission.USE_PROJ1_PROVIDER" …>…</provider>
- android:readPermission: Provider-wide read permission.
- android:writePermission: Provider-wide write permission.

Necessary Methods

- query()
  - Retrieve data from your provider.
- insert()
  - Insert a new row into your provider.
- update()
  - Update existing rows in your provider.
- delete()
  - Delete rows from your provider.
- getType()
  - Return the MIME type corresponding to a content URI.
- onCreate()
  - Initialize your provider. The Android system calls this method immediately after it creates your provider. Notice that your provider is not created until a ContentResolver object tries to access it.
- These need to handle concurrent accesses (need to be thread-safe)