Android Services & Local IPC: Programming Started Services

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand how to program Started Services
Implementing a Service is similar to implementing an Activity, e.g.:

Inherit from Android Service class

```java
public class MusicService extends Service {
    public void onCreate() {
        ...
    }
    public int onStartCommand(Intent intent,
                               int flags, int startId) { 
        ...
    }
    protected void onDestroy() {
        ...
    }
    public IBinder onBind(Intent intent) {
        return null;
    }
    ...
}
```
Programming a Started Service

- Implementing a Service is similar to implementing an Activity, e.g.:
  - Inherit from Android Service class
  - Override lifecycle methods
  - May need to determine the concurrency model to use in onStartCommand()

```java
public class MusicService extends Service {
    public void onCreate() {
        ...
    }
    public int onStartCommand(Intent intent, int flags, int startId) {
        ...
    }
    protected void onDestroy() {
        ...
    }
    public IBinder onBind(Intent intent) {
        return null;
    }
    ...
}
```
Implementing a Service is similar to implementing an Activity, e.g.:

- Inherit from Android Service class
- Override lifecycle methods
  - May need to determine the concurrency model to use in onStartCommand()
- The onBind() method & onUnbind() aren’t used for Started Services
  - You still need to provide a no-op implementation for onBind(), however

```java
public class MusicService extends Service {
    public void onCreate() {
        ...
    }
    public int onStartCommand(Intent intent,
                               int flags, int startId) {
        ...
    }
    protected void onDestroy() {
        ...
    }
    public IBinder onBind(Intent intent) {
        onBind = (Intent intent) { return null; }
        ...
    }
}
```
Implementing a Service is similar to implementing an Activity, e.g.:

- Inherit from Android Service class
- Override lifecycle methods
- Include the Service in the AndroidManifest.xml config file

```xml
<application ... >
  <activity android:name=".MusicActivity"
    ...
  </activity>

  <service android:exported="false"
    android:name=".BGLoggingService"
    ...
  </service>

</application>
```

[www.vogella.com/articles/AndroidServices/article.html](http://www.vogella.com/articles/AndroidServices/article.html) has more on Services
Music Player Service Example

- Client Activity can play music via a Started Service
Music Player Service Example

- Client Activity can play music via a Started Service
- To start the Service a user needs to push the “Play” button
Music Player Service Example

• Client Activity can play music via a Started Service

• To start the Service a user needs to push the “Play” button

• If music is playing when the client Activity leaves the foreground, the Music Service will continue playing
Music Player Service Example

- Client Activity can play music via a Started Service
- To start the Service a user needs to push the “Play” button
- If music is playing when the client Activity leaves the foreground, the Music Service will continue playing
- To stop the Service a user needs to explicitly push the “Stop” button
public class MusicActivity extends Activity {
  ...
  public void play (View src) {
    Intent intent = new Intent(MusicActivity.this,
                                MusicService.class);
    intent.putExtra("SongID", R.raw.braincandy);

    startService(intent);
  }

  public void stop (View src) {
    Intent intent = new Intent(MusicActivity.this,
                                MusicService.class);

    stopService (intent);
  }
}
public class MusicService extends Service {
    MediaPlayer player;

    public int onStartCommand(Intent intent, int flags, int startid) {
        player = MediaPlayer.create(this, intent.getIntExtra("SongID", 0));
        player.setLooping(false);
        player.start();
        return START_NOT_STICKY;
    }

    public void onDestroy() { player.stop(); }
}

developer.android.com/reference/android/media/MediaPlayer.html has more
Inherit from Service class
Extract the resid from the "extra" & create a MediaPlayer
Start playing the song (doesn't block)
Don't restart Service if it shuts down
Stop player when Service is destroyed
<application ... >

<activity android:name=".MusicActivity"
          android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>

<service android:exported="true"
         android:name=".MusicService" />

</application>

Service is usable by components external to this application
Analysis of the Music Player Service Example

- This is a very simple example of a Started Service
  - In particular, it runs in the UI Thread, but doesn’t block due to the behavior of MusicPlayer.start()
  - Also, there’s no communication from the Service back to the Activity that invoked it!
Analysis of the Music Player Service Example

• This is a very simple example of a Started Service

• Services with long-running operations typically need to run in separate Thread(s)
Download Service Example

- Client Activity requests a Started Service to download a file from a server
Download Service Example

- Clients send Intents via calls to `startService()`
Download Service Example

- Clients send Intents via calls to startService()
- The DownloadService is started on-demand
- Based on the Activator pattern

www.dre.vanderbilt.edu/~schmidt/PDF/Activator.pdf has more info
Download Service Example

- Clients send Intents via calls to `startService()`.
- The DownloadService is started on-demand.
- The DownloadService handles each Intent in turn the worker thread in a ServiceHandler & stops itself when it runs out of work.
- This implementation of the Command Processor pattern offloads tasks from an app’s main thread to a single worker thread.

www.dre.vanderbilt.edu/~schmidt/PDF/CommandProcessor.pdf has more info
public class DownloadActivity extends Activity {
    ...
    public void onClick(View v) {
        Intent intent = new Intent(DownloadActivity.this,
                                 DownloadService.class);
        ...
        intent.setData(Uri.parse(editText.getText().toString()));

        Add the URL to the download as data
        startService(intent);
    }
    ...
}
Download Service Implementation

```java
public class DownloadService extends Service {
    private volatile Looper mServiceLooper;
    private volatile ServiceHandler mServiceHandler;

    Handler that receives messages from the thread

    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) { super(looper); }

        public void handleMessage(Message msg) {
            downloadImage((Intent) msg.obj);
            stopSelf(msg.arg1);
        }

        Dispatch a callback hook
        method to download a file

        public void downloadImage(Intent intent) {
            /* ... */
        }

        Stop the service using the startId, so that we don't stop
        the service in the middle of handling another job

        public void downloadImage(Intent intent) {
            /* ... */
        }

        Download the image & notify the client

        ...
    }
}
```
Download Service Implementation

```java
public class DownloadService extends Service {
    ...
    public void onCreate() {
        super.onCreate();

        HandlerThread thread = new HandlerThread("DownloadService");
        thread.start();

        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }
```

Start up the thread running the service, which we create a separate Thread because the Service normally runs in the process's UI Thread that we don't want to block.

```java
HandlerThread thread = new HandlerThread("DownloadService");
thread.start();
```

Get the HandlerThread's Looper & use it for our Handler.

```java
mServiceLooper = thread.getLooper();
mServiceHandler = new ServiceHandler(mServiceLooper);
```
Download Service Implementation

```java
public class DownloadService extends Service {
    ... 
    public int onStartCommand(Intent intent, int f, int startId) {
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        msg.obj = intent;
        mServiceHandler.sendMessage(msg);
        return START_NOT_STICKY;
    }

    public void onDestroy() {
        mServiceLooper.quit();  // Shutdown the looper
    }
}
```

For each start request, send a message to start a job & deliver the start ID so we know which request we're stopping when we finish the job.

Message msg = mServiceHandler.obtainMessage();
msg.arg1 = startId;
msg.obj = intent;
mServiceHandler.sendMessage(msg);
return START_NOT_STICKY;

It's instructive to consider how to extend this example to run in a thread pool.
Analysis of the Download Service Example

- The worker thread solution shown here is a common Android Service idiom that implements the *Command Processor* pattern.
- In fact, it’s so common that Android provides a reusable framework that simplifies the use of this pattern!
Logging Service Example

• The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread

```java
public class LoggingService extends IntentService {
    protected abstract void onHandleIntent(Intent intent);
}
```

<table>
<thead>
<tr>
<th>L.</th>
<th>Time</th>
<th>PID</th>
<th>TID</th>
<th>Application</th>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>612</td>
<td>course.examples.Ser...</td>
<td>Logging...</td>
<td>Service destroyed</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>631</td>
<td>course.examples.Ser...</td>
<td>Logging...</td>
<td>Log this message</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>631</td>
<td>course.examples.Ser...</td>
<td>Logging...</td>
<td>Service created</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>612</td>
<td>course.examples.Ser...</td>
<td>Logging...</td>
<td>Service command started</td>
</tr>
</tbody>
</table>
The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread.

Clients send commands (expressed as Intents) via calls to startService().

```java
Intent intent = new Intent (this, LoggingService.class));
intent.putExtra("LogMsg", "hello world");
startService(intent);
```
Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread
- Clients send commands (expressed as Intents) via calls to startService()
- The LoggingService subclass handle intents in a worker thread asynchronously

```java
public class LoggingService extends IntentService {
    void onHandleIntent(Intent intent) {
        ... }
}
```
The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread.

Clients send commands (expressed as Intents) via calls to startService().

The LoggingService subclass handle intents in a worker thread asynchronously.

```java
public class LoggingService extends IntentService {
    void onHandleIntent(Intent intent) {
        ... 
    }
}
```

Android starts the Service as needed, which internally spawns a worker thread that handles a queue of intents.
Douglas C. Schmidt

Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread.
- Clients send commands (expressed as Intents) via calls to startService().
- The LoggingService subclass handle intents in a worker thread asynchronously.

```java
public class LoggingService extends IntentService {
    void onHandleIntent(Intent intent) {
        ... 
    }
}
```

The IntentService calls this hook method from the worker thread to handle each intent that started the Service.
Logging Service Example

- The Logging Service extends the IntentService to offload logging operations from an app’s UI Thread.
- Clients send commands (expressed as Intents) via calls to startService().
- The LoggingService subclass handle intents in a worker thread asynchronously.

```java
public class LoggingService extends IntentService {
    void onHandleIntent(Intent intent) {
        ... 
    }
}
```

When there are no more intents to handle the IntentService stops itself automatically.
Logging Activity Implementation

public class BGLoggingActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        ...
        buttonStart.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(BGLoggingActivity.this,
                                           BGLoggingService.class);

                intent.putExtra("LogMsg",
                               "Log this message");

                Add the message to log as an “extra”
                startService(intent);
            }
        });
    }
}
Logging Service Implementation

```java
public class BGLoggingService extends IntentService {
    ...

    Inherit from IntentService class

    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        return START_NOT_STICKY;
    }

    Don't restart this Service if it's shutdown

    protected void onHandleIntent(Intent intent) {
        ...
        Log.i(TAG, intent.getCharSequenceExtra("LogMsg").toString());
    }

    This hook method runs in a worker thread & logs the data
}
```

Note the “inversion of control” in the Android Service framework.
<application ... >

<activity android:name=".BGLoggingActivity"
        android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service android:exported="false"
        android:name=".BGLoggingService" />

</application>

Service is only usable by components in this application

developer.android.com/guide/topics/manifest/service-element.html#exported
AndroidManifest.xml File

```xml
<application ... >

<activity android:name=".BGLoggingActivity"
          android:label="@string/app_name">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<service android:exported="false"
         android:name=".BGLoggingService"
         android:process=":myProcess"/>
</application>
```

Instruct Android to run the BGLoggingService in its own process

[developer.android.com/guide/topics/manifest/service-element.html#proc]
Analysis of the Logging Service Example

- The LoggingService is an intentionally simplified example
Analysis of the Logging Service Example

• The LoggingService is an intentionally simplified example
• You don’t need to implement it as an IntentService (or even as a Service)
  • You could simply do the logging in a new Thread or ignore concurrency altogether!
Analysis of the Logging Service Example

- The LoggingService is an intentionally simplified example
- You don’t need to implement it as an IntentService (or even as a Service)
- In general, use a Service (or IntentService) when you want to run a component even when a user is not interacting with the app that hosts the Service

<table>
<thead>
<tr>
<th>L...</th>
<th>Time</th>
<th>PID</th>
<th>TID</th>
<th>Application</th>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>612</td>
<td>course.examples.Service</td>
<td>Logging...</td>
<td>Service destroyed</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>631</td>
<td>course.examples.Service</td>
<td>Logging...</td>
<td>Log this message</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>612</td>
<td>course.examples.Service</td>
<td>Logging...</td>
<td>Service created</td>
</tr>
<tr>
<td>I</td>
<td>09-19 12:...</td>
<td>612</td>
<td>612</td>
<td>course.examples.Service</td>
<td>Logging...</td>
<td>Service command started</td>
</tr>
</tbody>
</table>
Summary

• Programming Services & Intent Services is relatively straightforward
• Though they sometimes can be overkill…
Summary

- Programming Services & Intent Services is relatively straightforward
- The Service class uses the app’s UI Thread by default
  - Implementing a multi-threaded service should therefore often be made by extending Service class directly & spawning one or more threads
Summary

- Programming Services & Intent Services is relatively straightforward
- The Service class uses the app’s UI Thread by default
- IntentService creates a worker thread & uses that thread to run the service
- IntentService also creates a queue that passes one intent at a time to onHandleIntent()
Summary

- Programming Services & Intent Services is relatively straightforward.
- The Service class uses the app’s UI Thread by default.
- IntentService creates a worker thread & uses that thread to run the service.
- The Service class needs a manual stop via stopSelf() or stopService().
- Conversely, IntentService automatically stops itself when there are no more intents in its queue.
Android Services & Local IPC: Programming Started Services with Messengers

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

- Understand how to use Messengers to communicate from Started Services back to their invoking Activities
- Provides an interface for IPC with remote processes without using AIDL
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it.

[Diagram showing sender process associate Handler with Messenger, sending message to Messenger, and Messenger handling messages through handleMessage().]

developer.android.com/reference/android/os/Messenger.html has more info
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it.
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process.

Diagram:
- Sender Process
  - Messenger
  - sendMessage()
  - pass Intent
- Receiver Process
  - Intent (with Messenger extra)
  - onCreate()
  - onStartCommand()
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it.
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process.
- The receiver then does several things:
  - Obtains the Messenger.
Overview of Messengers

- A Messenger provides a reference to a Handler that others can use to send messages to it.
- An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process.
- The receiver then does several things:
  - Obtains the Messenger.
  - Returns the results back to the sender process.
Overview of Messengers

• A Messenger provides a reference to a Handler that others can use to send messages to it

• An Activity can create a Messenger pointing to a Handler in one process & then pass that Messenger to another process

• The receiver then does several things

• You can use Messengers with both Bound & Started Services to implement the Command Processor pattern

www.dre.vanderbilt.edu/~schmidt/PDF/CommandProcessor.pdf has more info
Using Messenger in Download (Started) Service

• DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService

• DownloadService uses the Messenger to reply back to the Activity

```
DownloadActivity
  onCreate()

ReplyHandler
  handleMessage()
```

Diagram:
- DownloadActivity
  - onCreate()
  - initiateDownload()

- ReplyHandler
  - handleMessage()
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity

Download Activity

1. onCreate()
2. create messenger

ReplyHandler

initiateDownload()

The Messenger stores a reference to the Handler
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity

The Intent stores a reference to the Messenger as an “extra”
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity

Diagram:

1. Download Activity
   - onCreate()
   - initiateDownload()
   - handleMessage()
2. Intent
   - startService()
3. Messenger
4. Download Service
   - onCreate()
   - onStartCommand()

Crossing process boundary

DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService. DownloadService uses the Messenger to reply back to the Activity.
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity

Diagram:

1. Download Activity
   - onCreate()
   - initiateDownload()
2. Messenger
   - handleMessage()
3. Intent
   - send intent (to DownloadService)
4. Download Service
   - onCreate()
   - onStartCommand()
   - Activate Service if it’s not already running
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity

![Diagram showing the interaction between DownloadActivity, Intent, Messenger, and DownloadService]

1. Download Activity
   - onCreate()
   - initiateDownload()
   - onClick()
   - handleMessage()
   - ReplyHandler

2. Messenger
   - onCreate()
   - startService()

3. Intent
   - onCreate()
   - startActivity()
   - sendMessage()

4. Download Service
   - onCreate()
   - onStartCommand()
   - create a worker thread & Handler
   - sendMessage()
   - handleMessage()
   - downloadImage()
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService.
- DownloadService uses the Messenger to reply back to the Activity.
Using Messenger in Download (Started) Service

- **DownloadActivity** passes Messenger as an “extra” to the Intent used to activate the **DownloadService**
- **DownloadService** uses the Messenger to reply back to the Activity

**Diagram:**

1. **Download Activity**
   - `onCreate()`
   - `initiateDownload()`

2. `onCreate()`

3. **Intent**
   - `startService()`

4. **Messenger**
   - `handleMessage()`

5. **Download Service**
   - `onCreate()`
   - `onStartCommand()`

6. **Service Handler**
   - `sendMessage()`
   - `handleMessage()`
   - `downloadImage()`

- **downloadActivity** passes Messenger as an “extra” to the Intent used to activate the **DownloadService**

- **DownloadService** uses the Messenger to reply back to the Activity
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService.
- DownloadService uses the Messenger to reply back to the Activity.

**Diagram:**
- **Download Activity**
  - `onCreate()`
  - `initiateDownload()`
  - `handleMessage()`
  - `ReplyHandler`

- **Intent**
  - `startService()`

- **Download Service**
  - `onCreate()`
  - `onStartCommand()`
  - `Service Handler`
  - `sendMessage()`
  - `handleMessage()`
  - `downloadImage()`

**Process Boundary Crossing:**
- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService.
- DownloadService uses the Messenger to reply back to the Activity.
Using Messenger in Download (Started) Service

- DownloadActivity passes Messenger as an “extra” to the Intent used to activate the DownloadService
- DownloadService uses the Messenger to reply back to the Activity
Programming a Messenger in Download Service

- Service replies to Activity via Messenger’s send() method

```java
public class DownloadService extends Service {
    ...
    private final class ServiceHandler extends Handler {
        ...
        public void downloadImage(Intent intent) {
            // ...
            Code to downloading image to pathname goes here

            Message msg = Message.obtain();
            msg.arg1 = result;
            Bundle bundle = new Bundle();
            bundle.putString("PATHNAME", pathname);
            msg.setData(bundle);
            Messenger messenger = (Messenger)
                intent.getExtras().get("MESSENGER");
            messenger.send(msg);
        }
    }
    ...
    Return pathname to the client
```
Programming a Messenger in Download Service

- Client Activity receives Message via its Handler event looper

```java
public class DownloadActivity extends Activity {
    ...
    Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            Bundle data = msg.getData();
            String pathname = data.getString("PATHNAME");

            if (msg.arg1 != RESULT_OK || path == null) {
                Toast.makeText(DownloadActivity.this, "failed download", Toast.LENGTH_LONG).show();
            }
            displayBitmap(path);
        }
    };
    ...
```

Get pathname from Download Service

Display the image
Summary

- Messengers provide a flexible framework for communication between processes in Android
- Asynchrony is straightforward, though can be complex for non-trivial usages