SSE3052: Embedded Systems Practice

Jinkyu jeong
jinkyu@skku.edu
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Android Application

- Activity Lifecycle
- Intents
- Intent Filters
- Services

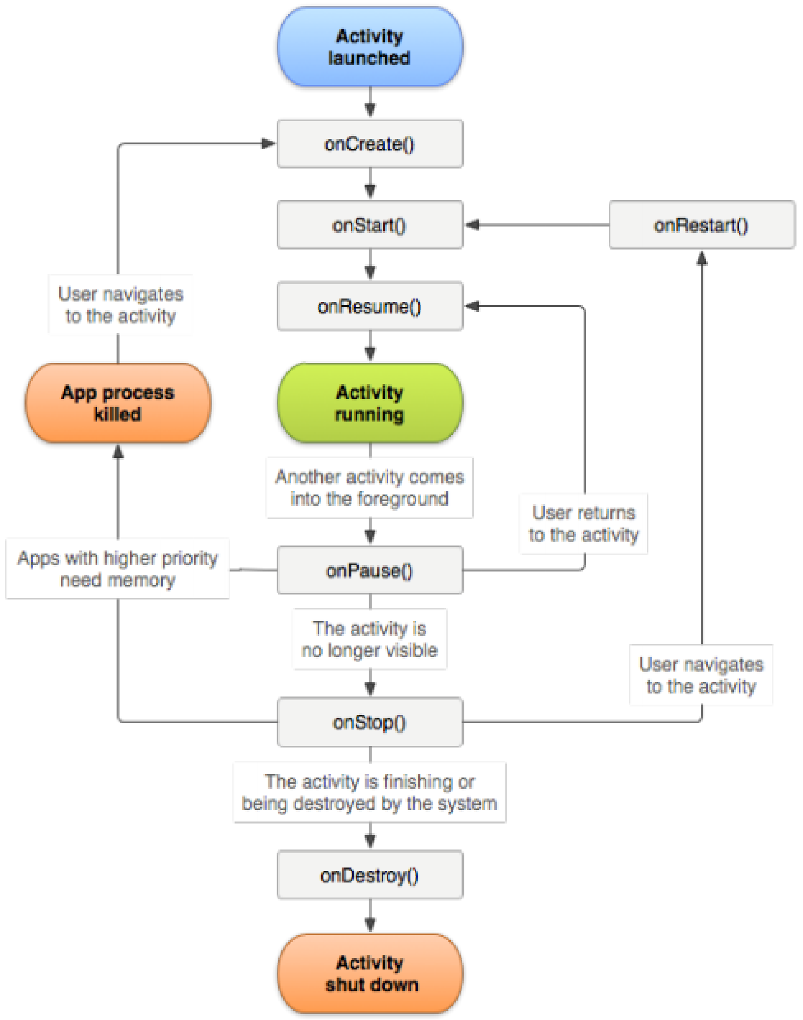
Activity States

- Resumed (=running)
  - Foreground
  - Has user focus
- Paused
  - Another activity is in foreground
  - Still visible (completely alive)
- Stopped
  - Completely obscured by another activity
  - No longer visible (but still alive)
- Killed

Creating Activity

- onCreate()
  - Must implement
  - System calls this when creating activity
  - Initializes essential components of activity
    - Ex) setContentView() to define layout of user interface
- onPause()
  - Indication that user is leaving activity
  - Usually, commit any changes that should be persisted

Activity Lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()

Example Activity

```
public class ExampleActivity extends Activity {
        //Override
        public void onCreate (Bundle savedInstanceState) {
                super.onCreate (savedInstanceState);
                // The activity is being created.
        }
        //Override
        protected void onStart() {
                super.onStart();
                // The activity is about to become visible.
        }
        //Override
        protected void onResume() {
                super.onResume();
                // The activity has become visible (it is now "resumed").
        }
        //Override
        protected void onPause() {
                super.onPause();
                //Another activity is taking focus (this activity is about to be "paused").
        }
        //Override
        protected void onStop() {
                super.onStop();
                // The acitivty is no longer visible (it is now "stopped")
        }
        //Override
        protected void onDestory() {
                super.onDestory();
                // The acitivity is about to be destroyed.
        }
}
```

Intent

- Facilitates communication between components
- 3 fundamental use cases:
  - Starting an activity
  - Starting a service
  - Delivering a broadcast

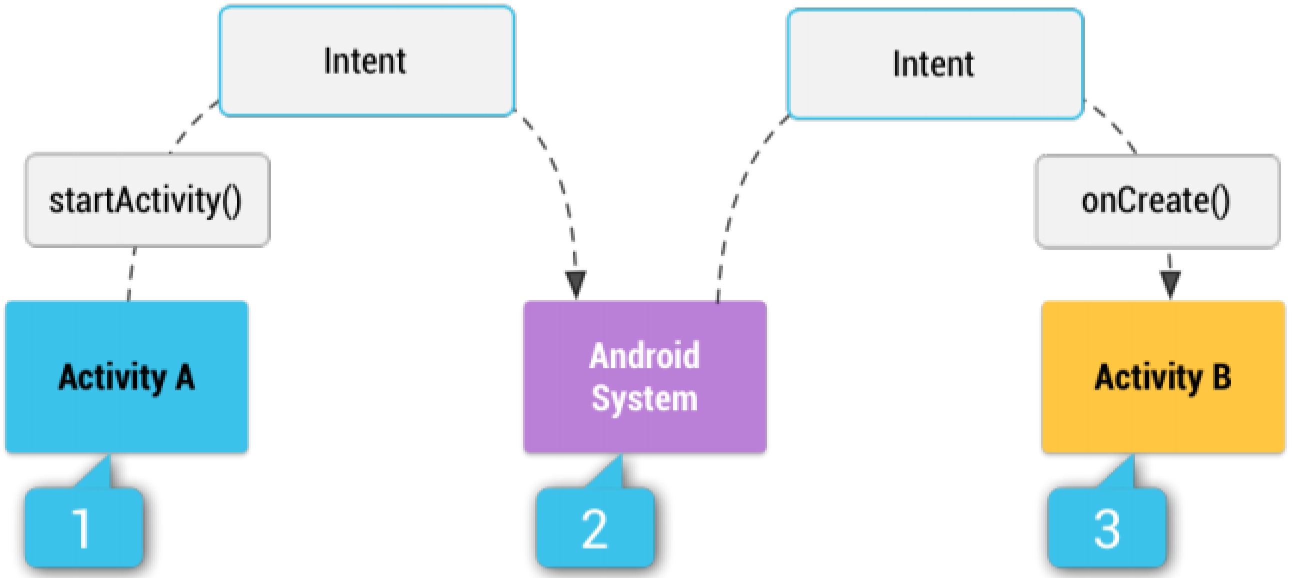
Starting Activities

- Use *startActivity()* method

Ex)

Intent intent = new Intent (this, ActivityB.class);

StartActivity (intent);



Intent Types

- Explicit – specify the component name
  - Used to start a component in the same application
  - *Intent intent = new Intent (this, ActivityB.class);*

- Implicit – do not name a specific component
  - Instead, declare a general action to perform
  - *Intent intent = new Intent (Intent.ACTION_VIEW, Uri.parse(http://csl.skku.edu”));*

Building Intent

- Constructors:
  - Intent (String action, Uri uri)
  - Intent (Context packageContext, Class<?> cls)

- Information contained in Intent:
  - Component name: Class name of target component
  - Action: Generic action to perform
  - Data: URI that references data to be acted on
  - (Category, Type, and Extras)

Action/Data Pair

- Examples:
  - ACTION_VIEW content://contacts/people/1
  - ACTION_DIAL content://contacts/people/1
  - ACTION_VIEW tel:123
  - ACTION_DIAL tel:123
  - …

https://developer.android.com/reference/android/content/Intent.html

Data Transfer to Target

- An intent can contain data via a *Bundle*
- Add data directly with *putExtra()* methods
  - 2 parameters: key-value pair
  - Key is always *String*
  - Values can be *int*, *float*, *String*, *Bundle*, *Parceable*, etc.
  - Ex) *Intent.putExtra("Val1", "This is for ActivityB");*
- Or, create a Bundle object with all extra data, then pass the object with *putExtras()*

Example: Building Intent

- Explicit

```
Intent downloadIntent = new Intent (this, DownloadService.class);
downloadIntent.setData (Uri.parse(fileUrl));
startService (downloadIntent);
```

- Implicit

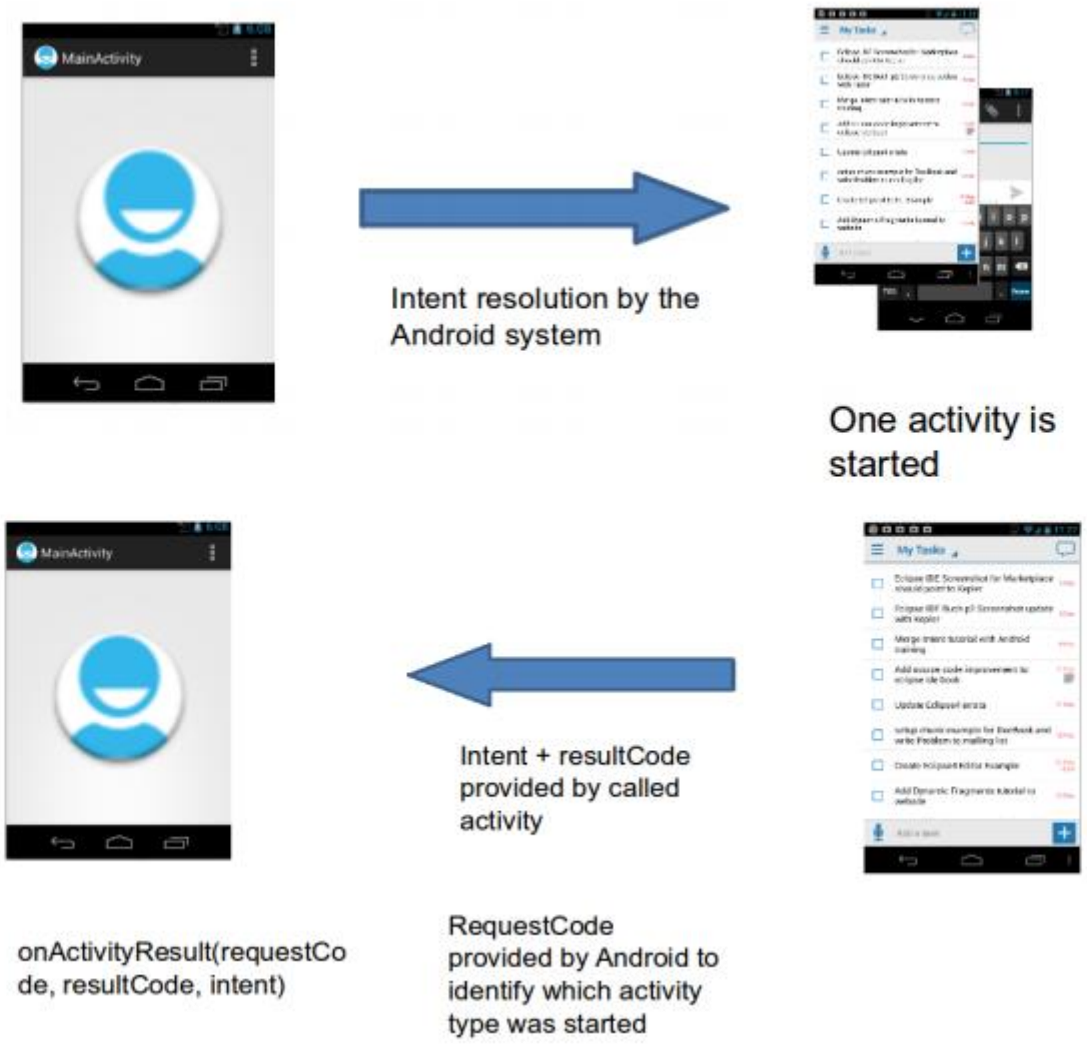
```
Intent sendIntent = new Intent();
sendIntent.setAction (Intent.ACTION_SEND);
sendIntent.putExtra (Intent.EXTRA_TEXT, textMessage);
sendIntent.setType ("text/plain");

if (sendIntent.resolveActivity (getPackageManager()) != null) {
      startActivity (sendIntent);
}
```

Receiving Data

- *getIntent()* – to retrieve *Intent* object
  - getAction()
  - getData()
  - getExtras() – to retrieve *Bundle* object (extra data)

```
Bundle extras = getIntent().getExtras();
if (extras == null) {
      return;
}
// get data via the key
String value1 = extras.getString (“Val1”);
String value2 = extras.getString (Intent.EXTRA_TEXT);
```

Retrieving Result Data


MainActivity
Intent resolution by the Android system
One activity is started
My Tasks
MainActivity
Intent + resultCode provided by called activity
onActivityResult(requestCode, resultCode, intent)
RequestCode provided by Android to identify which activity type was started

Retrieving Result Data

- Instead of *startActivity()*, use *startActivityForResult()* method
- Once sub-activity ends, *onActivityResult()* is called
- Sub-activity uses *finish()* method to go back to caller
- Sets a result using *setResult()* method

Intent Example

Caller Activity

```
Intent i = new Intent (this, ActivityTwo.class);
i.putExtra ("Value1", "This value one for AcitivityTwo");
i.putExtra ("Value2", "This value two ActivityTwo");
// set the request code to any code you like,
// you can identify the callback via this code
startActivityForResult (i, REQUEST_CODE);
```

Callee Activity

```
public void finish() {
        // Prepare data intent
        Intent data = new Intent();
        data.putExtra ("returnKey1", "Swinging on a start.");
        data.putExtra ("returnKey2", "You could be better then you are.");
        // Activity finished ok, return the data
        setResult (RESULT_OK, data);
        super.finish();
}
```

Intent Example (cont’d)

Caller Activity

```
Protected void onActivityResult (int requestCode, int resultCode, Intent data) {
        if (resultCode == RESULT_OK && requestCode == REQUEST_CODE) {
                if (data.hasExtra(“returnKey1”)) {
                        //…
                }
        }
}
```

Exercise 1

- http://web.archive.org/web/20161122015542/http://www.vogella.com/tutorials/AndroidIntent/article.html#exercise-starting-activities

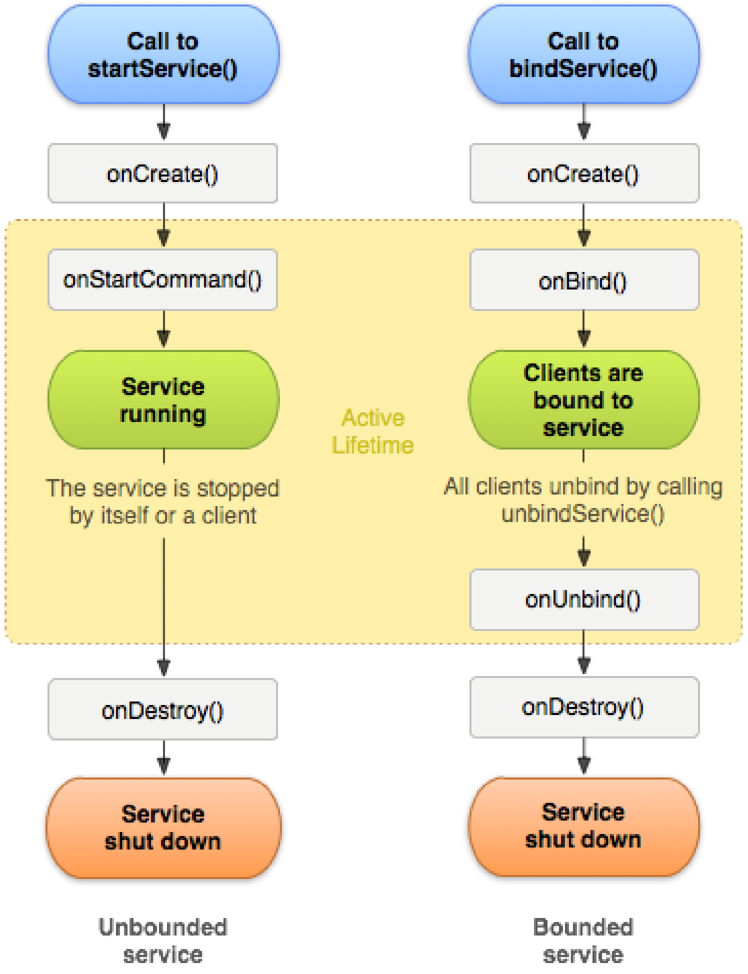
Services

- Run in background
- Do not provide a user interface
- Not bound to lifecycle of an activity
- Used for long-running operations
  - Handling network transactions
  - Playing music
  - Perform file I/O

Types of Services

- Scheduled
  - When `JobScheduler` launches the service
- Started
  - When application component calls `startService()`
- Bound
  - When application component binds to it by calling `bindService()`
  - Offers client-server interface that allows to interact with the service

Service Lifecycle



Service Lifecycle

- `onStartCommand()`
  - Invoked by calling startService()
- `onBind()`
  - Invoked by calling bindService()
  - Must implement
  - If you don't want to allow binding, return `null`
- `onCreate()`
  - To perform one-time setup procedures
- `onDestroy()`
  - To clean up any resources such as threads, registered listener s, or receivers

```
public class ExampleService extends Service {
    int mStartMode;       // indicates how to behave if the service is killed
    IBinder mBinder;      // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // A client is binding to the service with bindService()
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // All clients have unbound with unbindService()
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // A client is binding to the service with bindService(),
        // after onUnbind() has already been called
    }
    @Override
    public void onDestroy() {
        // The service is no longer used and is being destroyed
    }
}
```

Unlike activity lifecycle callback methods, you are *not* required to call the superclass implementation of these callback methods.

Declaring Service in Manifest

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

https://developer.android.com/guide/topics/manifest/service-element?hl=ko

Implementing Service

Two classes you can extend to create a service:

- `Service`
  - Base class for all services
  - Important to create a new thread
  - (Uses application's main thread by default)
- `IntentService`
  - Subclass of `Service`
  - Uses a worker thread to handle all of start request
  - Recommended if service does not handle multiple requests si multaneously
  - Must implement `onHandleIntent()`

Extending `IntentService`

```
public class HelloIntentService extends IntentService {

  /**
   * A constructor is required, and must call the super IntentService(String)
   * constructor with a name for the worker thread.
   */
  public HelloIntentService() {
      super("HelloIntentService");
  }

  /**
   * The IntentService calls this method from the default worker thread with
   * the intent that started the service. When this method returns, IntentService
   * stops the service, as appropriate.
   */
  @Override
  protected void onHandleIntent(Intent intent) {
      // Normally we would do some work here, like download a file.
      // For our sample, we just sleep for 5 seconds.
      try {
          Thread.sleep(5000);
      } catch (InterruptedException e) {
          // Restore interrupt status.
          Thread.currentThread().interrupt();
      }
  }
}
```

Extending `Service`

```
public class HelloService extends Service {
  private Looper mServiceLooper;
  private ServiceHandler mServiceHandler;


  // Handler that receives messages from the thread
  private final class ServiceHandler extends Handler {
      public ServiceHandler(Looper looper) {
          super(looper);
      }
      @Override
      public void handleMessage(Message msg) {
          // Normally we would do some work here, like download a file.
          // For our sample, we just sleep for 5 seconds.
          try {
              Thread.sleep(5000);
          } catch (InterruptedException e) {
              // Restore interrupt status.
              Thread.currentThread().interrupt();
          }
          // Stop the service using the startId, so that we don't stop
          // the service in the middle of handling another job
          stopSelf(msg.arg1);
      }
  }
```

Extending `Service` (cont'd)

```
@Override
  public void onCreate() {
    // Start up the thread running the service.  Note that we create a
    // separate thread because the service normally runs in the process's
    // main thread, which we don't want to block.  We also make it
    // background priority so CPU-intensive work will not disrupt our UI.
    HandlerThread thread = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();

    // Get the HandlerThread's Looper and use it for our Handler
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
  }

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
      Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

      // For each start request, send a message to start a job and deliver the
      // start ID so we know which request we're stopping when we finish the job
      Message msg = mServiceHandler.obtainMessage();
      msg.arg1 = startId;
      mServiceHandler.sendMessage(msg);

      // If we get killed, after returning from here, restart
      return START_STICKY;
  }

  @Override
  public IBinder onBind(Intent intent) {
      // We don't provide binding, so return null
      return null;
  }

  @Override
  public void onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
  }
}
```

Service Restart Behavior

- `START_STICKY`
  - Service is restarted if it gets terminated
  - Intent data passed to `onStartCommand` is null
- `START_NOT_STICKY`
  - Service is not restarted
- `START_REDELIVER_INTENT`
  - Similar to `START_STICKY` but original intent is re-delivered

Starting Service

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Stopping Service

- Service stop itself by calling `stopSelf(int)`
  - Pass the ID of start request (`startId` delivered to `onStartCommand()`)
- Another component can stop it by calling `stopService()`

https://developer.android.com/guide/components/services?hl=ko#java

Exercise 2

1. http://www.vogella.com/tutorials/AndroidServices/article.html#exercise-using-services-and-service-communication

Submission

- Format: YourStudentID_lab10.pdf
- Upload it on iCampus
- Due: 5/3 (Mon.) 23:59