Real-Time Embedded Systems

DT8025, Fall 2016
http://goo.gl/AZfc9l

Lecture 9

Masoumeh Taromirad
m.taromirad@hh.se

Center for Research on Embedded Systems
School of Information Technology

Processes in Android

Upon starting an application: a new Linux process with a single thread of execution.

By default, all components in the same process and thread (called the "main" thread).

Upon starting a component within an existing application: component is started within that process and uses the same thread of execution.

Keeping the UI reactive

Single thread model: challenge for reactivity.
Time consuming operations: separate threads ("background" or "worker" threads).

Note: the Android UI toolkit should not be accessed from outside the UI thread (not thread-safe UI methods)

The main thread

Run event listener and rest otherwise!

```
public void onClick(View v) {
   new Thread(new Runnable() {
     public void run() {
        // Do something ...

     }
   }).start();
}
```

Runnable

The interface `java.lang.Runnable` represents a command that can be executed.
A class that implements `Runnable` has to provide a method

```
public void run()
```

Threads in Java

Also used to start new threads in a Java program. This is how:

- Create a `Thread` passing a `Runnable` in the constructor.
- To start the thread use the method `start()`; it calls the `run()` method in the runnable.

Example

See the ManyThreads program to illustrate the constructs.

Another example

The prime calculator

We input a number N and get the prime number of order N. We use an extra button to test whether the UI is reactive even when calculating large prime numbers.

Calculating in the same thread

What we want to do when a number is given

```
int nr = Integer.parseInt(nrText.getText().toString());

long prime = primeNr( nr );

showtext.setText(""+ prime);
```

Place this code in the `OnKeyListener` for the `EditText`.

For large values it will make the UI unusable: the calculation takes a long time and the main thread cannot take care of other events.

Starting another thread to calculate

What we could do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(nrText.getText().toString());
   final long prime = primeNr( nr );
   showtext.setText(""+prime);
  }
}).start();
```

Place this code in the `OnKeyListener` for the `EditText`.

The main UI thread and this new worker thread take turns to execute.

Starting another thread to calculate

But it does not work!

We are not allowed to update the UI from other threads!

What we could do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(nrText.getText().toString());
   final long prime = primeNr( nr );
   showtext.setText(""+prime);
  }
}).start();
```

Posting to the UI thread

Posting *runnables*

Views have a couple of methods that allow you to ask the UI thread to run some code:

```
public boolean post (Runnable action)
public boolean postDelayed (Runnable action,
                            long delayMillis)
```

The prime calculator again

What we do when a number is given

```
new Thread(new Runnable() {
  public void run() {
   int nr = Integer.parseInt(nrText.getText().toString());
   final long prime = primeNr( nr );
   showtext.post(new Runnable() {
       public void run() {
          showtext.setText(""+ prime);
       }
   });
  }
}).start();
```

Services

Applications might need to do work even when the user is not interacting with the app.

Services

- Run in the background and do not provide a UI.
- May generate Notifications to start an Activity (with a UI)
- Remains running until it stops itself with `stopSelf()` or or another component stops it by calling `stopService()`.
- Created and started from other components by passing Intents.

Services or worker threads?

If you need to perform work outside your main thread, but only while the user is interacting with your application, then you should probably instead create a new thread

Define A Service

```
public class HelloService extends Service {
  ...
  // override the followings
  public void onCreate() { ... }

  public void onStartCommand() { ... }

  public void onBind() { ... }

  public void onDestroy() { ... }
  ...
}
```

Define A Service

```
public class HelloService extends Service {
  ...
  public void onCreate() { ... }

  public void onStartCommand() { ... }

  // always must be implemented; no binding: return null
  public void onBind() { return null; }

  public void onDestroy() { ... }
  ...
}
```

The echo client app

The launcher Activity

1. A button to start a Service to handle a TCP connection.
2. The Activity finishes directly after calling the Service.
3. An Intent is passed to the method that starts the service.
4. All this is done in the listener for the button.

The Service

1. Runs in the main thread.
2. We have to define the methods that are used by the system:
   - `void onCreate()`
   - `int onStartCommand(Intent i, int flags, int id)`
   - `void onDestroy()`
3. Must be terminated explicitely
   - `stopSelf`
   - or `stopService(Intent i)`

Services and threads

Activities and Services of an app run in the same main thread. If we want to do things in other threads we have to do it explicitly.

We would also like the worker thread of a Service to be very much like the main thread: doing nothing but waiting for messages to work on.

Loopers, Handlers, HandlerThreads

1. Every thread in android can be associated with a Looper (listens to messages)
2. We can associate Handlers to Loopers: they can receive messages that are put in a queue and dealt with in order.
3. HandlerThreads are already associated to a looper.

A Service with a ServiceHandler

The following is the program structure we suggest for a Service that can be asked to do several things.

Define a ServiceHandler inside your Service class

```
private final class ServiceHandler extends Handler{
  public ServiceHandler(Looper looper){
    super(looper);
  }
  // override handleMessage:
  public void handleMessage(Message msg){
    // Normally we would do some work here!
    // switch on  msg.what (integer)
    // to distinguish between different things to do!
  }
}
```

A Service with a ServiceHandler (ctd.)

onCreate starts a HandlerThread and associates a ServiceHandler to its Looper

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

  public void onCreate() {
    HandlerThread thread =
      new HandlerThread("TheServiceWorkerThread",
                        Process.THREAD_PRIORITY_BACKGROUND);
    thread.start();
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

A Service with a ServiceHandler (ctd.)

onStartCommand just sends messages to the ServiceHandler

```
public class TheService extends Service{
  public int onStartCommand(Intent intent,
                            int flags,
                            int startId) {
    Message msg = mServiceHandler.obtainMessage();
    msg.what = intent.getExtras().getString("WhatToDo"));
    mServiceHandler.sendMessage(msg);
    return START_STICKY;
  }
}
```

How does a Service start an Activity?

Services that have done what was required of them and want the app to start an Activity to interact with the user should not start the Activity themselves! (the user might be using some other app!).

Instead they should produce a Notification that the user can select in order to start an Activity.

An application with two Activities and a Service

Check the code distributed with this lecture!