Embedded Systems Programming - PA8001

http://bit.ly/15mmqf7

Lecture 12

Mohammad Mousavi

m.r.mousavi@hh.se

Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

A Multi-Threaded Server

```
public class EchoServer {
  public static void main(String[] args)  {
    int port = 4443;
    ServerSocket serverSocket = new ServerSocket(port);
    System.err.println("Started server on port " + port);
    while (true) {
      Socket clientSocket = serverSocket.accept();
      System.err.println("Connected to a new client");
      new Thread(new OneClientConnection(clientSocket)).
                              start();
   }
}
```

OneClientConnection

```
public void run() {
  in = new Scanner(new BufferedInputStream(clientSocket.get
  out = new PrintWriter(clientSocket.getOutputStream(), tru
  String s;
  int i = 0;
  while(in.hasNextLine()){
    s = in.nextLine();
    System.err.println(s);
    out.println("["+i+"]  " + s);
    i++;
  }
  ...
}
```

Services

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

Services: to be created and started from other components by passing Intents.
Run in the background and do not provide a UI.
May generate Notifications to start an Activity (with a UI)

Services

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

Services: to be created and started from other components by passing Intents.
Run in the background and do not provide a UI.
May generate Notifications to start an Activity (with a UI)

Services or worker threads?

Worker thread: to increase interaction with the use, while performing something time consuming

Service: to perform tasks in background with limited/no user interaction

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)`

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)`

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)`

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)`

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)`

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)`

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)`

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.

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.

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.

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.

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

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.

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 we distribute with this lecture!