Android Processes and Threads

Process

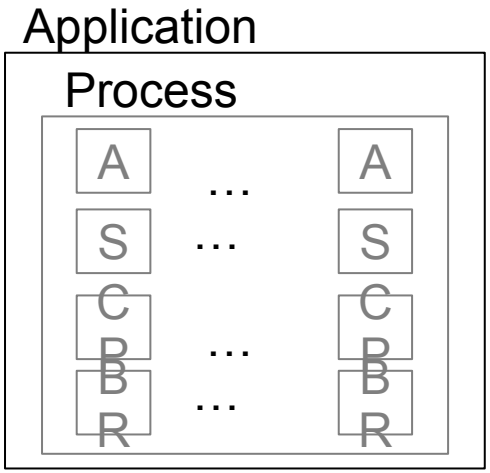
- A program in execution
  - OS creates a process to be served by the processor
  - Multi-tasking unit
    - multiple processes competing for system resources simultaneously
    - OS executes processes using some scheduling strategy
  - Memory isolation
    - Process requires memory for data, code, stack, etc
    - Multiple processes may be present in the memory simultaneously but each has its own address space independent of the others
    - Process cannot directly access memory of any other process, for better protection and security
  - Thread
    - Execution stream within the process
    - Process may have multiple threads running concurrently
    - All threads have access to the process memory

Android Process

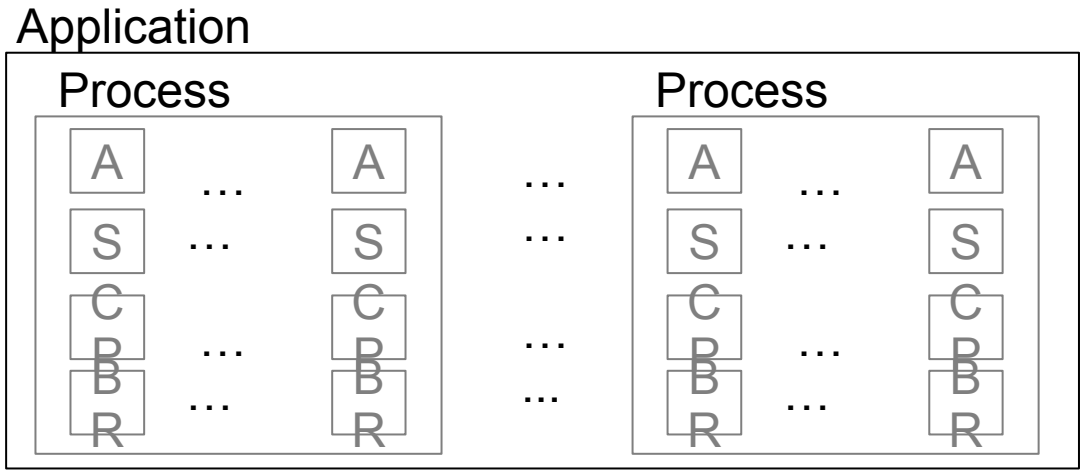
- Implementation
  - Based upon Linux Process with Android related variations
  - Processes may be killed due to low memory and restarted later
  - Process life-cycle differences

- Application architecture
  - Application by default has a single process and within that a single thread
  - All application components run in the same process and thread by default
    - Memory shared among all components
  - An application component may specify a separate process for execution
    - Memory not shared in this case
    - Inter-process communication required for interaction

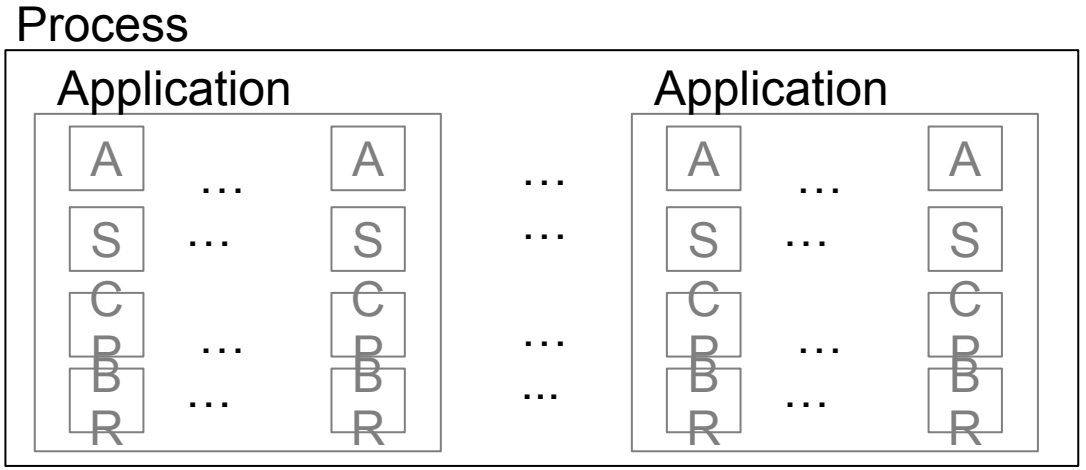
Architectural Possibilities



(a)



(b)



(c)

(a) All application components run in a single process by default
(b) Application can be split into multiple processes – generally suitable for services
(c) Multiple applications can be made to run in a single process – rarely done

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smd"
    android:versionCode="1"
    android:versionName="1.0">

  <application android:label="@string/app_name"
          android:icon="@drawable/ic_launcher"
          android:theme="@style/Notes">

    <activity android:name="NotesActivity"
          android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name="com.example.smd.ListActivity"
          android:label="ListActivity"
          android:parentActivityName="com.example.smd.NotesActivity" >

          <meta-data android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.smd.NotesActivity" />
    </activity>
    <provider android:name="NotesProvider"
          android:authorities="com.example.smd.notesprovider"
          android:exported="true" />

    <service android:name="SpellChecker"
          android:process="com.example.smd.spellchecker"/>

  </application>
</manifest>
```

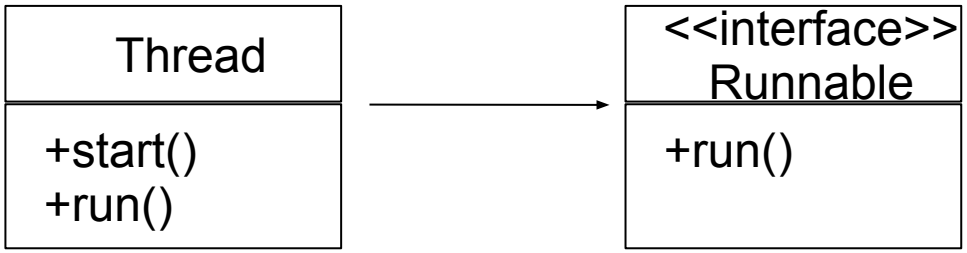
Application components running in default process (identified by package name)

Service running in separate process (identified by process attribute)

Thread

- Lightweight process
  - Independent execution stream running concurrently
  - Shares memory of same process with other threads
  - OS schedules threads for execution within a process
- Multi-threaded programming
  - Allows better responsiveness
    - long running tasks (e.g. network access, database queries, etc) can be delegated to worker threads to keep the main thread responsive
  - Synchronization issues arise
- Android Threading
  - Programming based on Java Threads
  - Underlying implementation used is Linux

```
Thread thread = new Thread (new
Runnable(){
   public void run(){
      // do task
      // …
   }
});

thread.start();
```



Asynchronous Programming

- Asynchronous
  - Execution order is non-deterministic
  - Multiple threads lead to asynchronous execution
  - Callbacks to rescue

```
Thread thread1 = new Thread (new
Runnable(){
   public void run(){
      System.out.println("thread-1");
   }
});

thread1.start();
```

```
Thread thread2 = new Thread (new
Runnable(){
   public void run(){
      System.out.println("thread-2");
   }
});

thread2.start();
```

Output: (a) thread-1
thread-2

(b) thread-2
thread-1

```
// synchronous program

if(authenticate()){
   welcome();
}



fetchNews();
```

```
// asynchronous program

authenticate(new Runnable(){
   public void run(){
      welcome();
   }
});

fetchNews();
```

```
// authenticate implementation

public bool authenticate(){
   // extract parameters from UI fields
   // do authentication task
   return result;
}
```

```
// authenticate implementation using thread

public void authenticate(Runnable callback){
   Thread thread = new Thread(new Runnable() {
      public void run(){
         // extract parameters from UI fields
         // do authentication task …

         if (result == true ){
            // call the callback on task completion
            callback.run();
         }
      }
   });
   thread.start();
}
```

Concurrency and Synchronization

- Issue
  - Multiple threads accessing a shared resource simultaneously
  - Unless synchronized shared resource is not **thread-safe**

- Solution
  - Shared Memory model
    - Lock-based
      - One thread acquires the lock and others wait till the lock is released
    - Efficient but programmatically complicated and error-prone
  - Message Passing model
    - Queue-based
      - One thread actually access the shared resource while others send messages (for resource access) that are handled in a queue
    - Easier to implement

Android Threading Model

- Model
  - Each process has a main thread
    - Also known as UI thread
  - Looper managing event loop
  - Long running tasks can block main thread leading to application not responding issues
  - Worker threads may be created for long running tasks

- Considerations
  - Do not block the main thread
    - use worker threads for long running tasks
  - Do not access the Android UI outside the UI (main) thread
    - Android UI is not thread-safe and requires synchronization
    - Worker threads can send messages to UI thread for UI manipulation

Handler

- Associated with Message Queue
- Used to handle incoming messages
  - Runnable
  - Message

```
public class ThreadActivity extends Activity
{
    …

    Handler handler = new Handler();

    public void buttonClick(View view)
    {

      Thread thread = new Thread (new Runnable(){
        public void run(){
          try{

            // do task ...

            handler.post(new Runnable(){
              public void run(){
                  // update UI
              }
            });

            } catch (Exception ex) { }

        }
      });

      thread.start();
    }
}
```

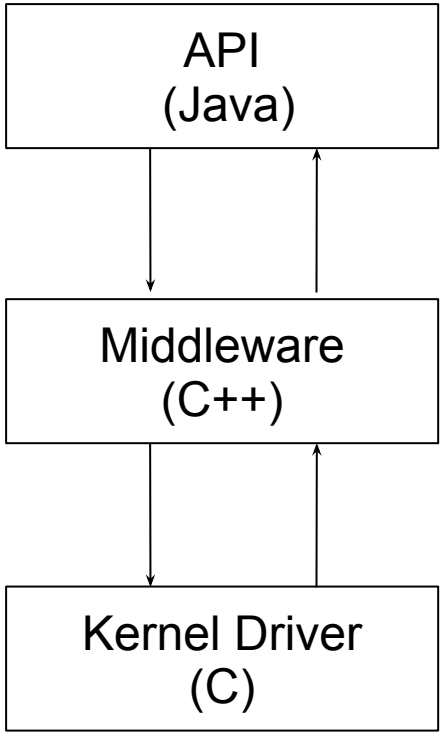
Callback

```
public class ThreadActivity extends Activity
{
    …

    public void buttonClick(View view)
    {

        AsyncTask<Void,Void,Void> task = new
AsyncTask<Void,Void,Void>(){
            protected Void doInBackground(Void... params){
                // do task
            }

            protected void onPostExecute(Void result){
                // update UI
            }
        };

        task.execute();

    }
}
```

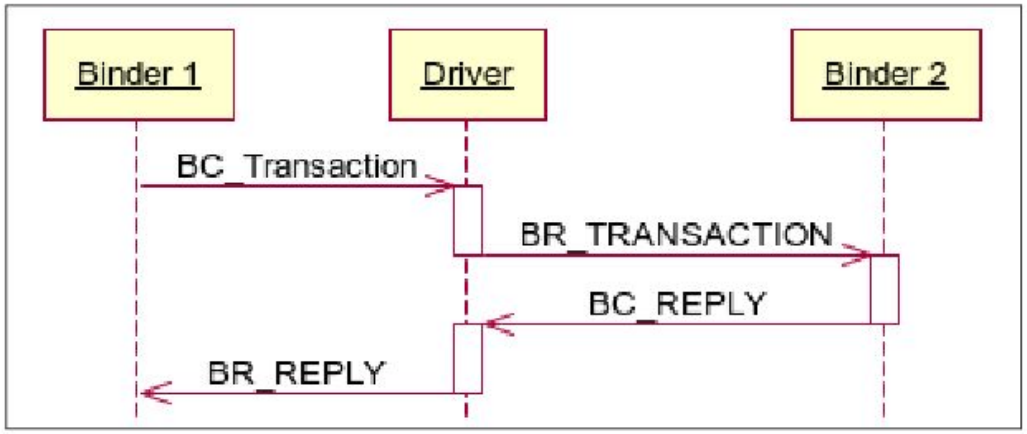
Inter-process Communication

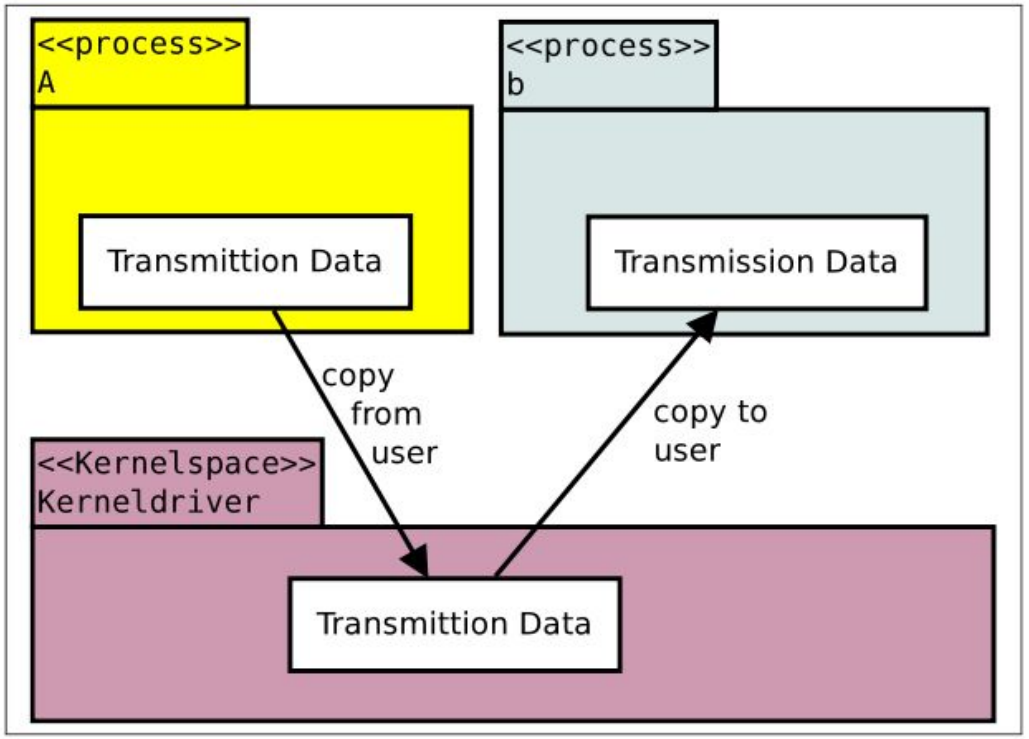
- Need
  - Processes are isolated – each having its own address space
  - Communication mechanism required for cooperation
- Techniques
  - Shared resources
    - Memory
    - Files, Memory-mapped files, Databases, etc
  - Message passing
    - Pipes
    - Sockets
    - **RPC**
- Android support
  - Binder Framework
    - Programming model based upon message passing and RPC
    - OS support using Binder kernel driver based upon OpenBinder

High-level Architecture

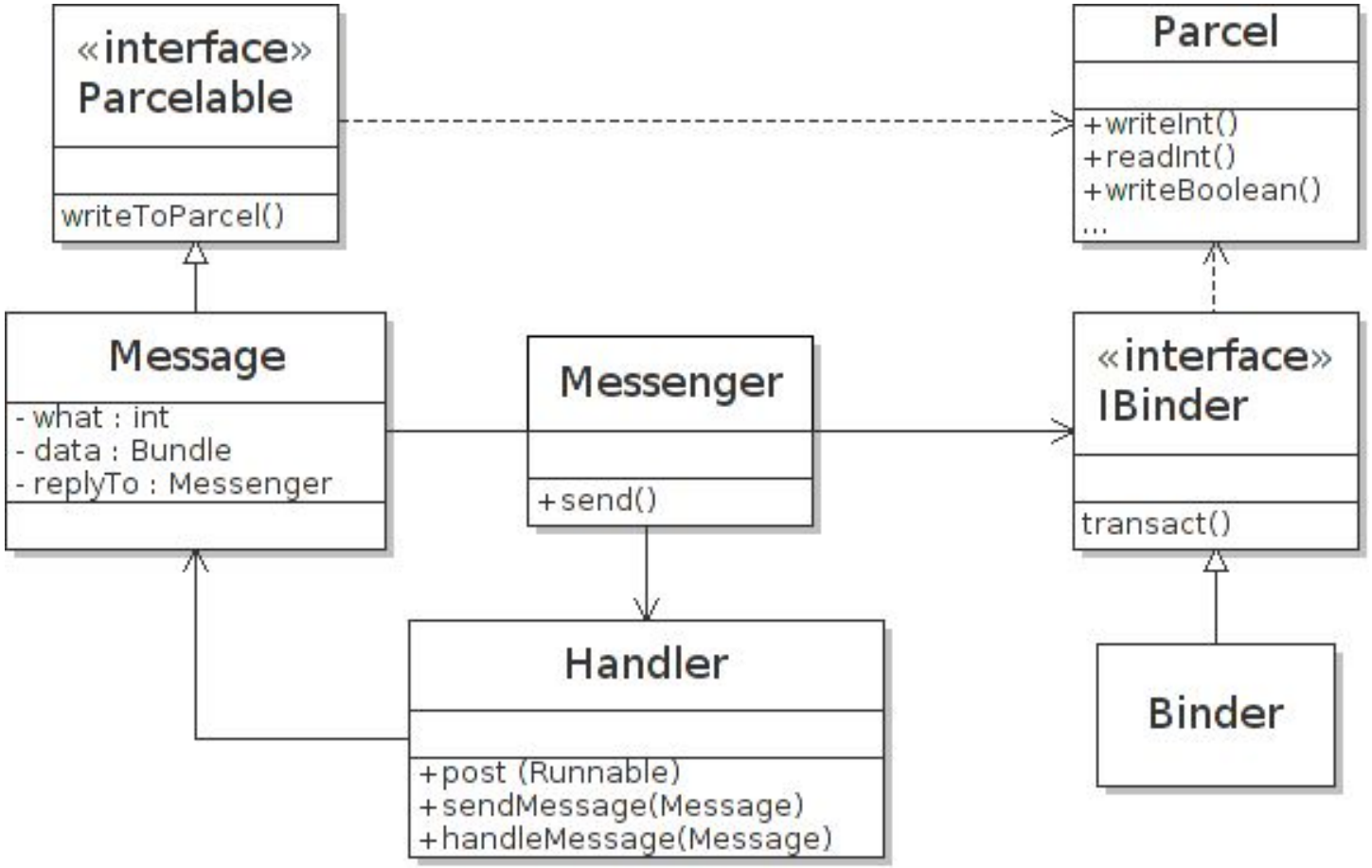


Binder IPC and Transactions

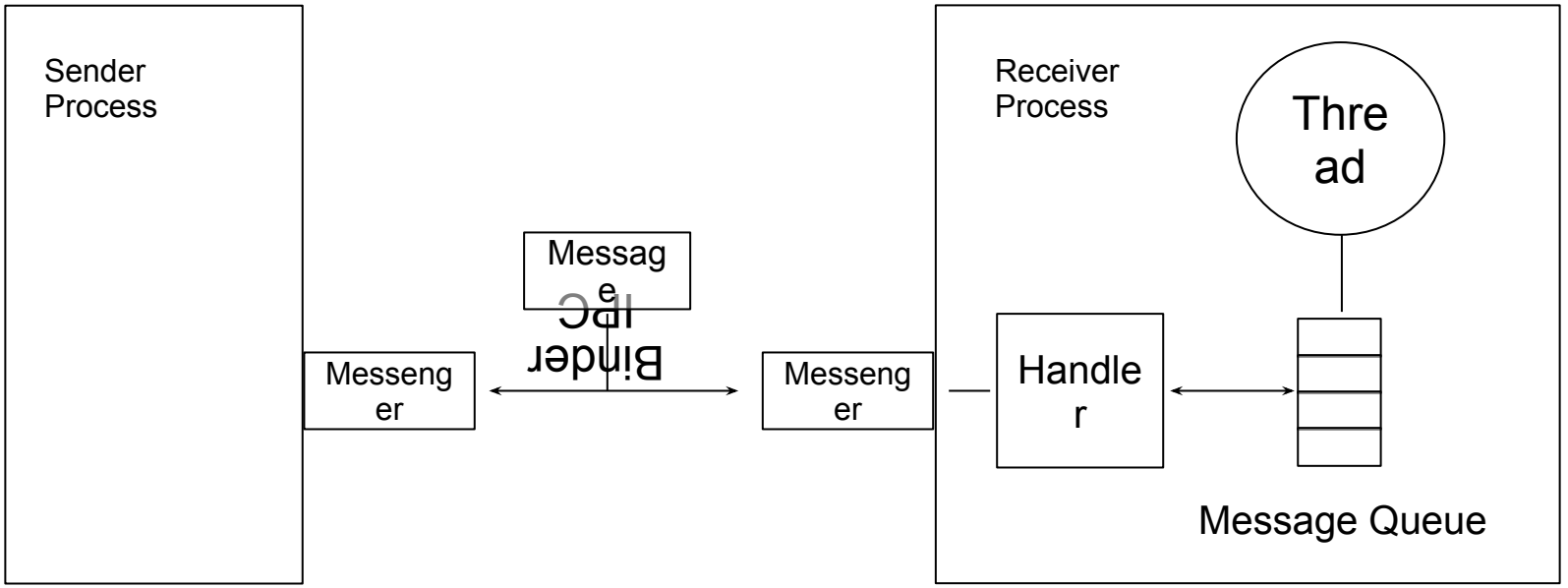




Java API


«interface»
Parcelable
writeToParcel()
Parcel
+writeInt()
+readInt()
+writeBoolean()
...
Message
- what : int
- data : Bundle
- replyTo : Messenger
Messenger
+send()
«interface»
IBinder
transact()
Handler
+post (Runnable)
+sendMessage(Message)
+handleMessage(Message)
Binder

Communication Architecture



- **Messenger** encapsulates **Binder** object capable of remote communication
- **Message** is a **Parcelable** that can be serialized and sent from one process to other
- **Messenger** (on the receiving side) is also associated with **Handler**
- Messengers in sender and receiver processes exchange **Message** through Binder IPC
  - Sender process sends the **Message** through its **Messenger**
  - Receiver process **Messenger,** on receipt of **Message**
    - delegate to **Handler** that puts the **Message** in **Message Queue**
    - handles the **Message** on its turn, in the main thread, once again through **Handler**

Sender Process
Activity

```
public class NotesActivity extends BaseActivity
{

   private Messenger messenger;

   private ServiceConnection connection = new ServiceConnection(){

      public void onServiceConnected(ComponentName className,IBinder binder) {

        messenger = new Messenger(binder);
        Message message = Message.obtain(null,1);
          // may send some data along in form of Bundle
        try {
           messenger.send(message);
        } catch (RemoteException e) { }

        bound = true;
      }

      public void onServiceDisconnected(ComponentName className){
         bound = false;
      }
   };

   protected void onStart(){
      super.onStart();
      Intent intent = new Intent(this,SpellChecker.class);
      bindService(intent,connection, Context.BIND_AUTO_CREATE);
   }

   protected void onStop(){
      super.onStop();
      if(bound){
         unbindService(connection);
      }
   }
}
```

Receiver Process
Service

```
public class SpellChecker extends Service {

   Messenger messenger = new Messenger(new IncomingHandler());

   public IBinder onBind(Intent intent){
      return messenger.getBinder();
   }

   public class IncomingHandler extends Handler{

      public void handleMessage(Message msg){
         switch(msg.what){
            case 1:
               processMessage(msg,msg.replyTo);
            default:
               super.handleMessage(msg);
         }
      }
   }

   public void processMessage(final Message msg, final Messenger replyTo){

      // process the message

      // may also send a reply back using replyTo messenger, if that is not null

   }

}
```