Contents

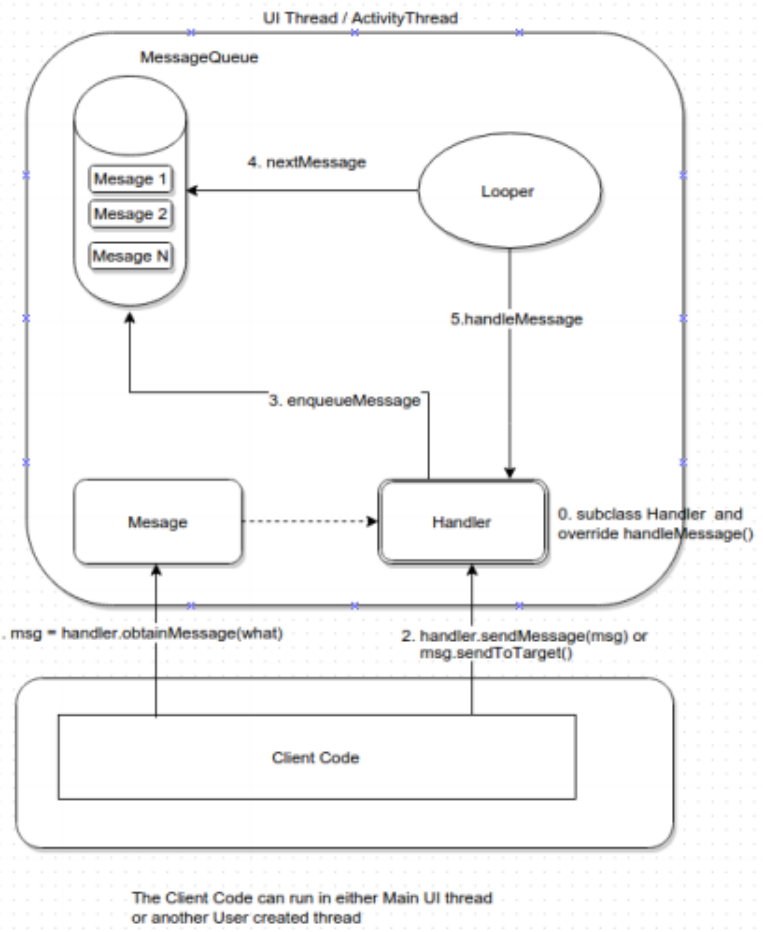
1 Thread of Execution

Android applications use a **single thread model**. A single thread of execution called **main**. It is started when a process is created.

- Handles and dispatches user interface events: drawing the interface, responding to interactions. E.g. onClick...()
- Handles activity lifecycle events: onCreate(), onDestroy. For all components in an application
- HandlerThread

2 Looper and handler

- HandlerThread
  - Extension of Thread with support for a Looper
- Looper
  - Each HandlerThread can have one Looper
  - A Java thread dies when the run method returns
  - Maintains a MessageQueue
  - Looper.loop(): loops through the MessageQueue and processes waiting Messages
- Message
  - A task to be completed
  - Might contain data, reference to a Runnable object
- Handler
  - Attached to a Looper
  - Enqueues messages in the Looper MessageQueue
  - Configurable delivery
  - Handles messages from the MessageQueue
  - Threadsafe
  - One Looper can have many Handlers associated with it



2.1 Splitting threads

- Long (ish) running code that does not involve the UI
  - E.g. an image download
  - Occurs in a separate thread of execution
  - Still tightly coupled to an activity
  - Not allowed to do network communication in the UI thread
- Instantaneous code that does involve the UI
  - E.g. drawing the image that has been downloaded
  - posted to the UI thread responsible for a particular View to execute, logically parceled up as a Runnable object
  - Risk of orphaned threads

3 AsyncTask

A convenience class for making complex asynchronous worker tasks easier. Worker / blocking tasks are executed in a background thread. Can get data back using **results callback**, and it's executed in the UI thread. With each AsyncTask that is spun off, a thread is created and destroyed, which might be a performance issue. We can solve this by implementing implement a thread pool.

4 Services

An Application Component that

- Has no UI
- Represents a desire to perform a longer-running operation. I.e. longer than a single-activity element of the task
- Threads are associated with the activity that started them i.e. could be orphaned

Activities are loaded/unloaded as users move around app, where as services **remain for as long as they are needed**. Can expose functionality for other apps: one service **may be used by many applications**, which allows to avoid duplication of resources

4.1 What services are not

- Not a separate process
  - Runs in the same process as the application in which it is declared (by default)
- Not a thread
  - One thread per Application
  - Handles events for all components
  - If you need to do things in the background, start your own thread of execution

4.2 Uses of Services

- MP3 Playback: Want to play audio while the user is doing other things
- Network Access: long download, sending email, polling email server for new mail.
- Anything that you dont want to interrupt the user experience for

4.3 Creating a Service

Services are designed to support communication with

- Local Activities (in the same process). For example: within VM
- Remote Activities (in a different process). For example IPC
- Multiple components
  - System services underpin much of Android core OS, but wrapped with various APIs

Reference section

atomic

Something that "appears to the rest of the system to occur instantaneously" (One operation at a time).**Atomic operation** means an operation that appears to be instantaneous from the perspective of all other threads. You don't need to worry about a partly complete operation when the guarantee applies.