Services & Background

Tasks
Introduction

• Last week we did various storage-related tasks
  • Reading Files, Databases
• These tasks can potentially take time to execute
• Let’s see how and why to handle such tasks in background
Process Management Recap

• By default, components of a single app run in the same Linux process

• When the system wants to run a new component
  • If app doesn’t yet have any running components, a new process with a single execution thread is started
  • Otherwise, component is started within that process
  • By default, all components also run on the same thread!
    • “Main thread”

• It’s possible to control which process certain components run in
  • Manifest `android:process` XML attribute

Threads

• Main thread – handles UI
  • Drawing views
  • User interaction
  • Lifecycle events

• Also other system calls will run on the main thread
  • Lifecycle callbacks

• Too much work on Main/UI thread – slowdowns & hiccups

Threads

• As there is 1 main execution thread, both application components and UI interactions are done synchronously.

• Long computations, I/O background tasks on the main thread will block the UI.

• If the UI thread hangs for more than a few seconds (5s), the “Application Not Responding” dialog is presented.
  • Poor user experience.

• All long-running tasks should be run on a separate background thread.
val thread = Thread {
    // a time consuming task
    val bmp: Bitmap = processBitmap("image.png")

    // TODO: do something with the result - show on UI, ...
}

thread.start()
Threads continued

• The Android UI functions are NOT thread-safe
• All UI manipulation must be done from UI thread

Thus
• On one hand you should do work on worker threads
• On the other hand, you can’t update UI from those threads
• What to do? Use messages and/or helper methods:
  • `Activity.runOnUiThread(Runnable)`
  • `View.post(Runnable)`
  • `View.postDelayed(Runnable, long)`
Example – Sending results from Worker Thread to UI thread

Thread {
    // a time consuming task, run on new thread
    val processedImage : Bitmap = processBitmap("image.png")

    // show in UI
    my_image_view.post { // code to be run on UI thread:
        my_image_view.setImageBitmap(processedImage)
    }
}

}.start()

Background work options

- Several ways for implementing background work:
  - Standard Java (Kotlin) threads, extending Runnable
    - Use the helper methods from prev. slides
    - Do messaging with Handlers
  - Android AsyncTask
    - Useful for one-off tasks, e.g. download image
- AlarmManager
  - For timing jobs precisely
- DownloadManager
  - Downloading URI-s to a file in the background
- For more complex applications, consider using WorkManager API
  - Schedule tasks, while balancing system resources

https://developer.android.com/guide/background/
https://developer.android.com/training/multiple-threads/communicate-ui
AsyncTask

• A helper class designed to perform work in the background and publish results to UI
• Manages threads and inter-thread messaging for you
• An asynchronous task is defined by
  • 3 types (can also be void):
    • Params, Progress, Result
  • 4 Steps
    • onPreExecute
    • **doInBackground**
    • onProgressUpdate
    • onPostExecute
• **doInBackground** is the minimum necessary part to use, and the only one run in background!

AsyncTask

private inner class DownloadFilesTask : AsyncTask<URL, Int, Long>() {

  override fun doInBackground(vararg urls: URL): Long? {
    val count = urls.size
    var totalSize: Long = 0
    for (i in 0..count) {
      totalSize += Downloader.downloadFile(urls[i])
      publishProgress((i / count.toFloat() * 100).toInt())
    }
    return totalSize
  }

  protected override fun onProgressUpdate(vararg progress: Int) {
    setProgressPercent(progress[0])
  }

  override fun onPostExecute(result: Long?) {
    showDialog("Downloaded $result bytes")
  }
}
Services
Services

• Faceless components that run in the background
  • E.g. music player, network download manager, chat service notifies user even if they are not using the chat app

• Have higher priority than background activities, thus suitable for long-running tasks
  • Recall runtime memory management

• Services are started by app components using Intent
  • startService( .. )

https://developer.android.com/guide/components/services
Bound vs Started

• Services can take 3 forms:
  • Background service
    • Called with startService()
    • Run indefinitely
      • Until manually stopped
  • Foreground service
    • A notification is shown while the service keeps running
  • Bound
    • App components bind to the service with bindService()
    • Runs as long as there are components bound to it
    • Provides a binding – a client-server interface

• The type is determined by which callback methods you implement
  • onStartCommand() – starts as background service
  • onBind()
Services - continued

- Implementing your own Service
  - Extend Service class and declare in Manifest
  - Override callback methods
    - onStartCommand(), onBind()

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ee.ut.cs.week6">
  <application>
    ...
    <service
      android:name=".FileLoadingService"
      android:enabled="true"
      android:exported="true" />
  </application>
</manifest>
```

https://developer.android.com/guide/components/services
class MyService : Service() {

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        // get extras from intent, etc
        val foo = intent?.getStringExtra("key1")

        // do some work

        return START_STICKY // stay running
    }

    override fun onBind(intent: Intent): IBinder? {
        // should return a communication channel to the service
        return null // if not using bound service
    }
}

• Bound service example:
  • [https://developer.android.com/guide/components/bound-services.html#Binder](https://developer.android.com/guide/components/bound-services.html#Binder)
Starting, Stopping a service

• Start via Intent object
  • Can attach extras
  • Can set action

```kotlin
val intent = Intent(context, FileLoadingService::class.java)
startService(intent)
```

• Stop using
  • stopSelf() – from the service itself
  • stopService() -- from other components

• Bound services starts/stop based on components binding to them

https://developer.android.com/guide/components/services#StartingAService
Demo
Results of services with broadcasts

• Using broadcasts, your service can notify interested apps about some task finishing

    // In service:
    var done = Intent()
    done.setAction("ee.ut.cs.dl_file")
    done.putExtra("downloadedFilePath", filePath)
    sendBroadcast(done)

    // In activity
    val filter = IntentFilter()
    filter.addAction("ee.ut.cs.dl_file")
    registerReceiver(bmpReceiver, filter)
Services & Threads

• Services still run in the main thread of host process by default!

• Generally, you want to create a new thread inside the Service

• There is an extension of Service – IntentService, which creates a thread for you
  • Also includes a sequential work queue
Notifications

- Notifications – messages displayed to user outside of app’s main UI in the top “navigation drawer”
  - System events, updates, new messages, tasks finishing

- Notifications have
  - Icons (1,4), titles (5), text (6), timestamps(3)
  - Interactive behavior (actions upon click)

- May also appear in lock screen

Images: https://developer.android.com/guide/topics/ui/notifiers/notifications
https://developer.android.com/guide/topics/ui/notifiers/notifications
Building a basic notification

• Notification.Builder – to build the object
• NotificationManager – to send out the notification

```kotlin
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(android.R.drawable.ic_dialog_info)
    .setContentTitle("Done")
    .setContentText("My Service finished")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)

val notification = builder.build()
with(NotificationManagerCompat.from(this)) {
    // notificationId is a unique int for
    // each notification that you define!!
    notify(notificationId, builder.build())
}
```
## Configuring notifications

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setAutoCancel( bool: Boolean)</td>
<td>Whether clicking hides the notification</td>
</tr>
<tr>
<td>setLargeIcon(resourceld)</td>
<td></td>
</tr>
<tr>
<td>setContentTitle(string)</td>
<td></td>
</tr>
<tr>
<td>setContentText(string)</td>
<td></td>
</tr>
<tr>
<td>setPriority()</td>
<td>Used on &lt;= 7.1, for 8.0+, use channels</td>
</tr>
<tr>
<td>setStyle()</td>
<td>Various extra options: multiple texts, images, media playback buttons</td>
</tr>
<tr>
<td>setColor()</td>
<td></td>
</tr>
<tr>
<td>setContentIntent()</td>
<td>Intent action which is run when clicked</td>
</tr>
<tr>
<td>setSound()</td>
<td></td>
</tr>
<tr>
<td>setTicker()</td>
<td>Scrolling text</td>
</tr>
<tr>
<td>addAction()</td>
<td>Extra button with action attached</td>
</tr>
<tr>
<td>setGroup()</td>
<td>Which group to appear in</td>
</tr>
</tbody>
</table>

For more information, refer to the following links:

- [https://developer.android.com/training/notify-user/expanded.html](https://developer.android.com/training/notify-user/expanded.html)
Notification Channel

• Since Android API 26 (Android 8.0), all notifications must be assigned a channel

```kotlin
val name = "My Channel"
val descriptionText = "My Channel Description"
val importance = NotificationManager.IMPORTANCE_DEFAULT
val channel = NotificationChannel(CHANNEL_ID, name, importance).apply {
    description = descriptionText
}

val notificationManager: NotificationManager =
    getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
notificationManager.createNotificationChannel(channel)
```

https://developer.android.com/training/notify-user/channels
Adding actions to notification

createNotificationChannel()
var builder = ...

// Create an explicit intent for an Activity in your app
val intent = Intent(this, MainActivity::class.java)
intent.putExtra("processedImage", image)
...

// Wrap it in a PendingIntent
val pending: PendingIntent = 
    PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT)

// Set the intent fires when user taps the notification
builder.setContentIntent(pending)

val notification = builder.build()
Foreground Service

• A service which the user is aware of – via notification.
  • startForeground(id : Int, notification: Notification)
  • stopForeground()

• Has higher priority from system perspective

• Need permission android.permission.FOREGROUND_SERVICE

• Must provide a notification for the status bar
  • cannot be dismissed unless the service is either stopped or removed from the foreground.

https://developer.android.com/guide/components/services#Foreground
To conclude

• UI can be updated only from UI thread!
• Normal Java threads are fine to use for simple cases
• Services
  • Started, Foreground, Bound
  • Still need to manage threading!
• Notifications- another entry point into your app
Next week

• Automated Android testing
  • Unit tests
  • UI tests

• Home Assignment 1 will be announced