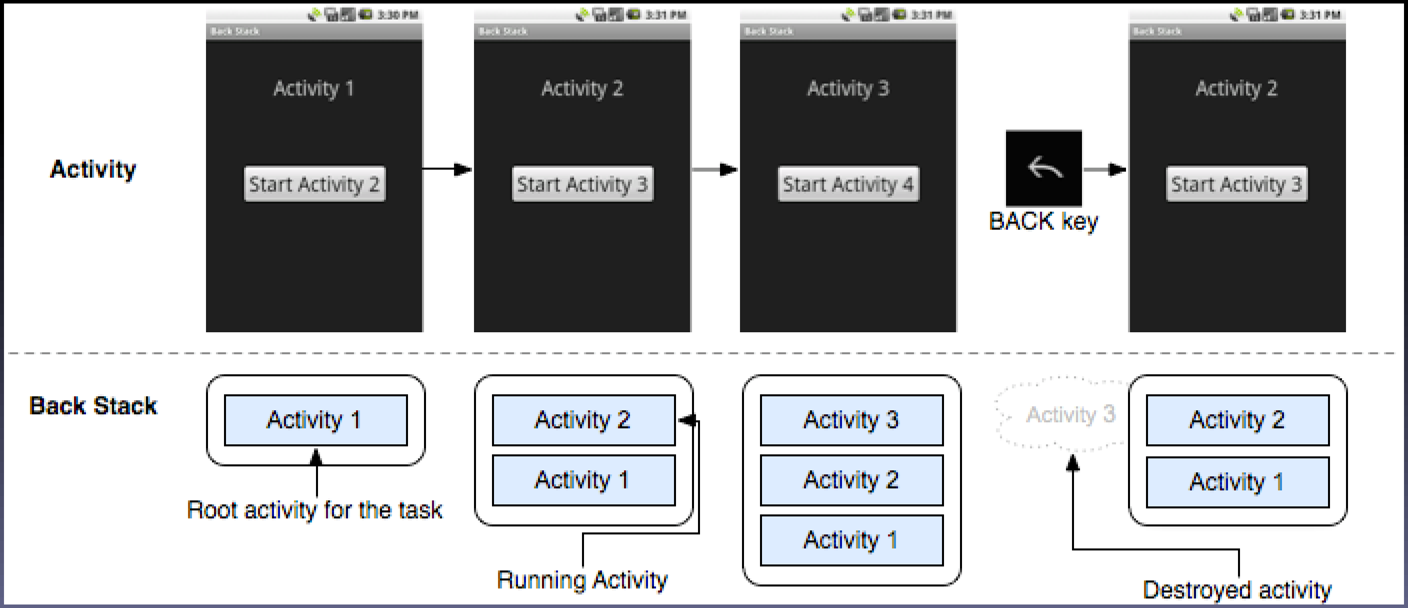
PROCESSES AND THREADS

Threads, and processes, and tasks! Oh, my!

- Process – linux process
  - Usually one process per application
- Thread – linux thread
  - May be multiple per process
  - May encompass multiple application components
- Tasks
  - Android stack of activities
  - May cross process boundaries
  - One task stack for each "job"

Task stack

- Stack of activities rooted at initial activity
- Multiple tasks may exists at once
  - Background task stacks
- Back button pops activity from stack



Processes

- When an application component is launched, if no other component is running, a new process with single thread is started
- Separate process can be specified for component(s) using android:process attribute in manifest file
- Processes may be killed due to low memory
    - importance hierarchy from process lifecycle

Threads

- When application is launched, creates “main” thread
  - UI thread
- Long operations should go in extra threads
  - Background / worker threads
- Single-threaded model for UI
  - 2 rules
    - Do not block UI thread
    - Do not access the android UI toolkit from outside the UI thread
  - UI toolkit not thread-safe, must always be manipulated in UI thread
  - Several ways to access UI thread from extra threads
    - Activity.runOnUiThread(Runnable)
    - View.post(Runnable)
    - View.postDelayed(Runnable, long)
    - Handler

Threads

- Threads and Runnables created using standard Java syntax
- Example new thread creation

```
new Thread (new Runnable() {
  public void run() {
    // implementation . . .
  }
}).start();
```

UI helper thread example

```
public void onClick (View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork
              ("http://exmpl.com/img.png");

            mImageView.post (new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Looper

- Android class for providing message queue for threads
- UI thread has a Looper created for it implicity
    - Can connect to this queue and handle messages by declaring new handler in main thread
- HandlerThread
    - Handy class for starting a new thread that has a looper

Handler

- Handles messages and runnables passed to a thread’s message queue
  - Connects to thread’s Looper
- Thread safe
- Extend Handler and override handleMessage(Message msg)

Message

- Members
  - public int what
  - public int arg1
  - public int arg2
  - public object obj
  - public Messenger replyTo
- Typically constructed using Message.obtain()
  - returns message object from global pool to avoid allocating new objects

Handler

- Example implementation

```
class MyHandler extends Handler {
    @Override
    public void handleMessage (Message msg) {
      switch (msg.what) {
        case 1:
          // do something;
          break;
        case 2:
          // do something else;
          break;
        default:
          super.handleMessage(msg);
      }
    }
}
```

Looper/Handler example

AsyncTask

- Simplifies creation of long-running tasks that need to communicate with the UI
- Must be subclassed
- Instance must be created on UI thread
- Instance can only be executed once
- Automatically invokes
  - doInBackground() on worker thread
  - onPreExecute(), onPostExecute(), onProgressUpdate() on UI thread
  - Return value of doInBackground() passed to onPostExecute()
- publishProgress() can be called in doInBackground() to execute onProgressUpdate()
  - Useful for progress bar updates

AsyncTask example

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://exmpl.com/img.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {

    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

Interprocess communication

- Remote procedure calls can be accomplished two ways
  - Android interface definition language (AIDL)
  - Messenger service
- AIDL provides custom defined interface
  - Requires other applications to have AIDL files
- Messenger service has standard format but less flexible
- Binder of AIDL or Messenger interface can be returned to clients in onBind()

AIDL

- Defines an interface for interprocess communication
- Needs to be thread-safe
  - Calls from local process are handled in caller thread
  - Calls from remote process are handled from thread pool
- Calls to interface are direct function calls (synchronous), unless *oneway* keyword specified (asynchronous)
- Interface define in .aidl file
  - Android SDK tools automatically generate IBinder interface

Example .aidl file

```
// IRemoteService.aidl
package com.example.android;

// Declare any non-default types here with import statements

/** Example service interface */
interface IRemoteService {

    /** Request the process ID of this service */
    int getPid();

    /** Demonstrates some basic types that you can use as
     *  parameters and return values in AIDL.
     */
    void basicTypes(int anInt, long aLong, boolean aBoolean,
            float aFloat, double aDouble, String aString);
}
```

Messenger

- Pointer to Handler
- Allows handler to be called from other processes
- Can be used for interprocess message passing
- To expose handler
  - Initialize Messenger with Handler to share

    Messenger mMessenger = new **Messenger** (new MyHandler());
- To connect to remote handler
  - Initialize Messenger with IBinder of remote interface

    Messenger mMessenger = new **Messenger**(serviceIbinder);

IPC example

Thread safe

- Interprocess communication with an IBinder performed using a pool of threads in IBinder process
- ContentProvider methods called from a pool of threads in content provider’s process
  - query()
  - insert()
  - delete()
  - update()

LogCat

- Android logging system mechanism used to view system debug output
- Can be used to view stack trace of emulator errors
  - Useful for locating line of code were error initiated
- LogCat viewable in realtime in Debug or DDMS view of Eclipse
- Common logging methods
  - v - verbose
  - d - debug
  - i - information
  - w - warning
  - e - error
- Usage example
  - Log.i(“MyActivity”, “MyClass.memberfunction – error message”);