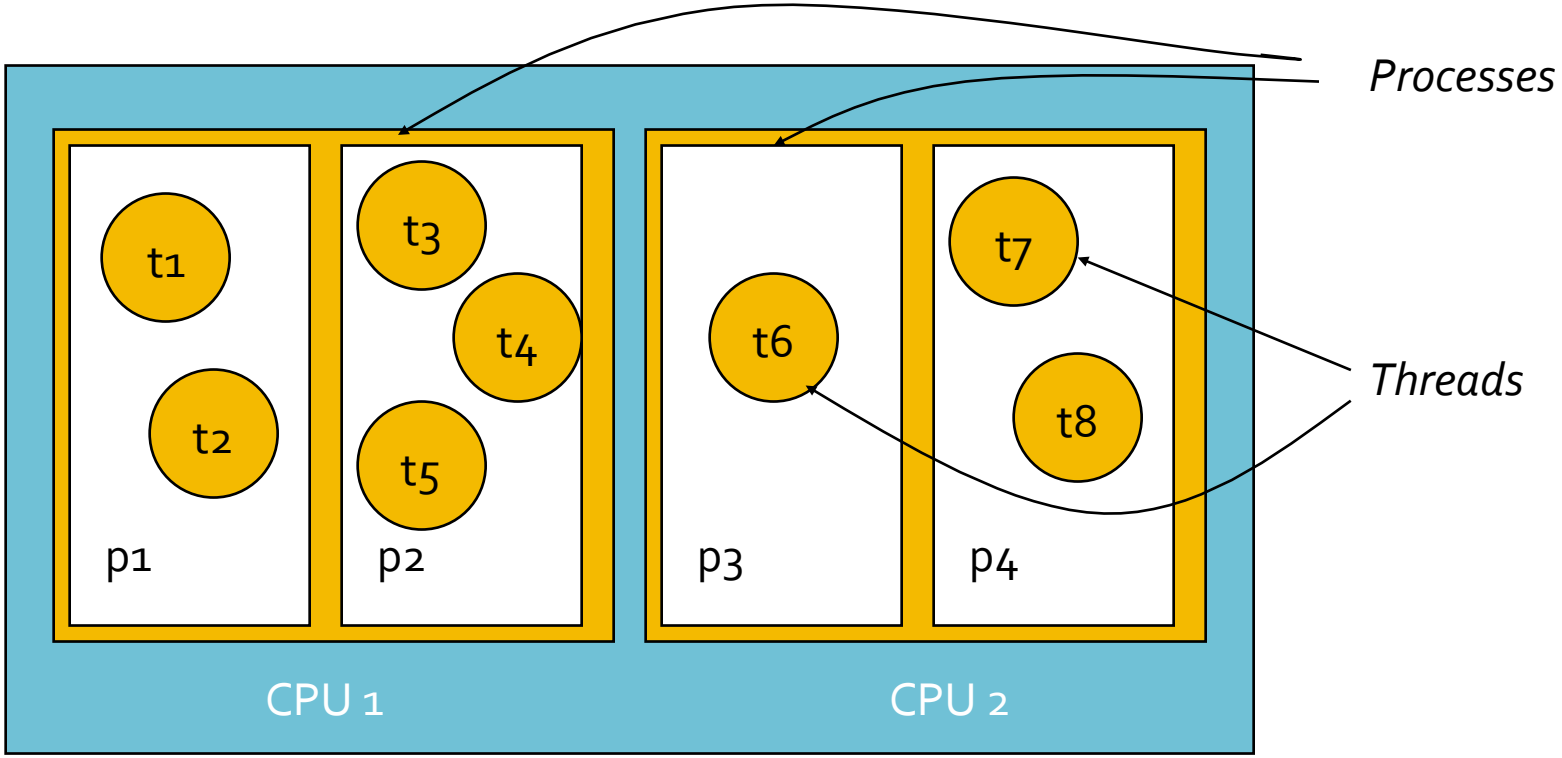
Threads, AsyncTasks & Handlers

Programming the Android Platform

Android Threading

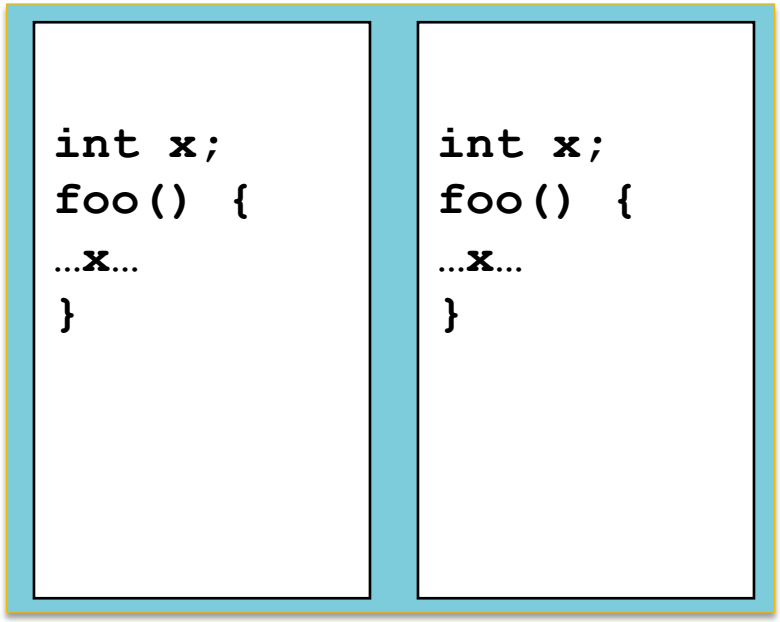
- Android uses Threads
- What is a Thread?
  - Conceptual view
    - Parallel computations running in a process
  - Implementation view
    - Each Thread has a program counter and a stack
    - The heap and static areas are shared across threads

Computation Abstractions

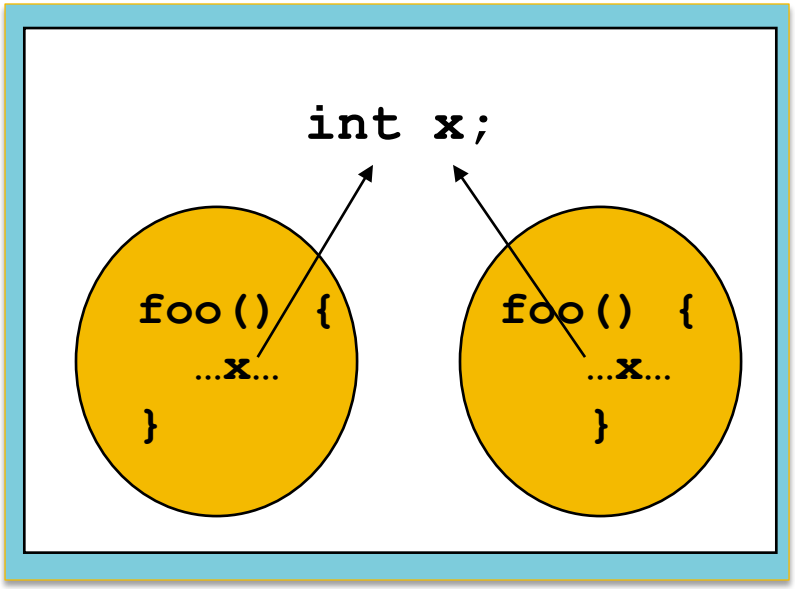


Computer

Processes vs. Threads



Processes don't directly share data



Threads within a process share data

Some Thread Methods

- void start()
  - Starts the Thread
- boolean isAlive()
  - Returns true if the thread has been started, but hasn't yet terminated
- void interrupt()
  - Sends an interrupt request to calling Thread
- void join()
  - Waits for a thread to die

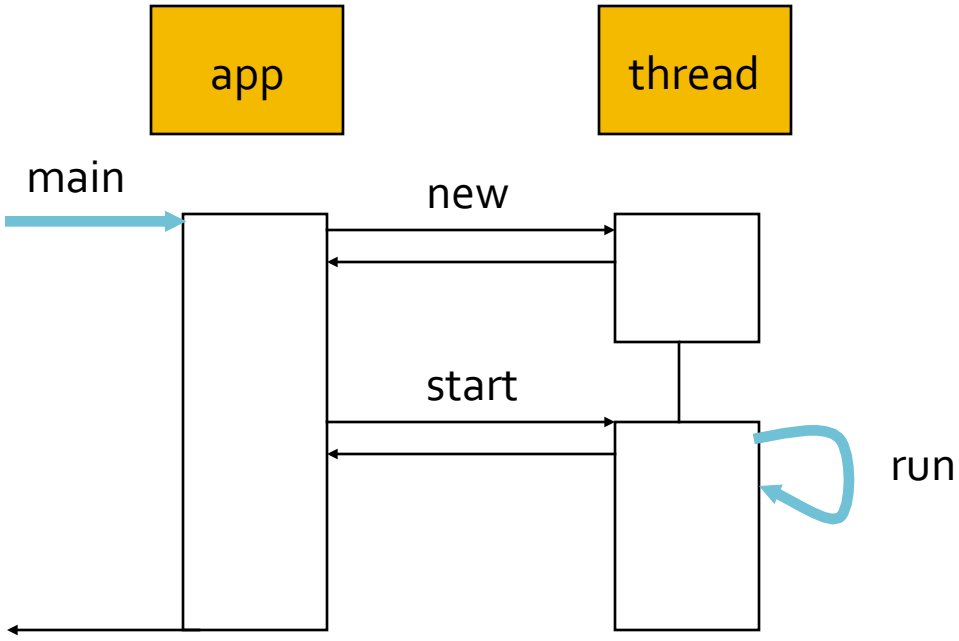
Some Static Thread Methods

- void sleep(long time)
  - Sleeps for the given period
- Thread currentThread()
  - Thread object for currently executing thread
- Boolean holdsLock(Object object)
  - Returns true if calling Thread holds an intrinsic lock on object

Basic Thread Use Case

- Instantiate a Thread object
- Invoke the Thread's start() method
  - Thread will invoke its own run()
- Thread terminates when run() returns

Basic Thread Use Case (cont.)



Thread Example

```
public class SimpleThreadingExample extends Activity {
  private Bitmap bitmap;
  public void onCreate(Bundle savedInstanceState) {
  …
  final ImageView iview = …
  new Thread(new Runnable() {
     public void run() {
       synchronized (iview) {
         bitmap = BitmapFactory
              .decodeResource(getResources(),R.drawable.icon);
         iview.notify();
       }
     }
}).start();
…
```

Thread Example (cont.)

```
final Button button = …
button.setOnClickListener(new OnClickListener() {
  public void onClick(View v) {
    synchronized (iview) {
      while (null == bitmap) {
        try {
          iview.wait();
        } catch (InterruptedException e) {…}
      }
      iview.setImageBitmap(bitmap);
    }
  }
});
…
```

The UIThread

- Applications have a main thread (the UI thread)
- Application components in the same process use the same main thread
- User interaction, system callbacks & lifecycle methods handled in the UI thread
- UI toolkit is not thread safe

Implications

- Blocking the UI thread hurts responsiveness
  - Long-running ops should run in background thread
- Don't access the UI toolkit from non-UI thread
  - UI & background threads will need to communicate

Posting Runnables on UI thread

```
public class SimpleThreadingExample extends Activity {
  private Bitmap bitmap;
  public void onCreate(Bundle savedInstanceState) {
  …
    final ImageView iview = …
    final Button button = …
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new Thread(new Runnable() {
          public void run() {
            Bitmap  = …
            iview.post(new Runnable() {
              public void run() { iview.setImageBitmap(bitmap);}
            });
          }
        }).start();
…
```

Posting Runnables on UI thread

```
public class SimpleThreadingExample extends Activity {
  private Bitmap bitmap;
  public void onCreate(Bundle savedInstanceState) {
  …
    final ImageView iview = …
    final Button button = …
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new Thread(new Runnable() {
          public void run() {
            Bitmap  = …
            SimpleThreadingExample.this
                    .runOnUiThread( new Runnable() {
                            public void run() { iview.setImageBitmap(bitmap);}
            });
          }
        }).start();
…
```

AsyncTask

- Structured way to manage work involving background &UI threads
- In background thread
  - Perform work
- In UI Thread
  - Setup
  - Indicate progress
  - Publish results

AsyncTask (cont.)

- Generic class

```
class AsyncTask<Params, Progress, Result> {
  …
}
```

- Generic type parameters
  - Params – Types used in background work
  - Progress – Types used when indicating progress
  - Result – Types of result

AsyncTask (cont.)

- void onPreExecute()
  - Runs before doInBackground()
- Result doInBackground (Params... params)
  - Performs work
  - Can call void publishProgress(Progress... values)
- void onProgressUpdate (Progress... values)
  - Invoked in response to publishProgress()
- void onPostExecute (Result result)
  - Runs after doInBackground()

AsyncTask (cont.)

```
public class SimpleThreadingExample extends Activity {
  ImageView iview;
  ProgressBar progress;
  public void onCreate(Bundle savedInstanceState) {
    …
    iview = …
    progress = …
    final Button button = …
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new LoadIconTask().execute(R.drawable.icon);
      }
    });
  }
…
```

AsyncTask (cont.)

```
class LoadIconTask extends AsyncTask<Integer, Integer, Bitmap> {
  protected Bitmap doInBackground(Integer... resId) {
    Bitmap tmp = BitmapFactory.decodeResource(
                                              getResources(), resId[0]);
   // simulated long-running operation
    for (int i = 1; i < 11; i++) {
      ...
      publishProgress(i * 10);
    }
    return tmp;
  }
...
```

AsyncTask (cont.)

```
...
protected void onProgressUpdate(Integer... values) {
   progress.setProgress(values[0]);
}
protected void onPostExecute(Bitmap result) {
   iview.setImageBitmap(result);
}
...
```

Handler

- Threads can also communicate by posting Messages & Runnables to a Handler
- Message
  - Can contain a code, data object & args
  - Recipient (Handler) implements response
- Runnable
  - Contains an instance of the Runnable interface
  - Sender implements response

Handler

- sendMessage()
  - Puts Message on MessageQueue
- post()
  - Puts Runnable on MessageQueue
- Looper
  - One per Thread
  - Dispatches MessageQueue entries
    - Calls handleMessage() for Messages
    - Calls run() for Runnables

Handler (cont.)

- Two main uses for a Handler
  - Schedule Message/Runnable for future execution
  - Enqueue action to be performed on a different thread

Runnables & Handlers

- boolean post(Runnable r)
  - Add Runnable to the MessageQueue
- boolean postAtTime(Runnable r, long uptimeMillis)
  - Add Runnable to the MessageQueue. Run at a specific time (based on SystemClock.upTimeMillis())
- boolean postDelayed(Runnable r, long delayMillis)
  - Add Runnable to the message queue. Run after the specified amount of time elapses

Runnables & Handlers (cont.)

```
public class SimpleThreadingExample extends Activity {
  private ImageView iview;
  private Handler handler = new Handler();
  public void onCreate(Bundle savedInstanceState) {
    …
    iview = …
    final Button = …
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        new Thread(new LoadIconTask(R.drawable.icon)).start();
      }
    });
  }
…
```

Runnables & Handlers (cont.)

```
private class LoadIconTask implements Runnable {
  int resId;

  LoadIconTask(int resId) {
    this.resId = resId;
  }

  public void run() {
    final Bitmap tmp =
                    BitmapFactory.decodeResource(getResources(),resId);
    handler.post(new Runnable() {
        public void run() {
           iview.setImageBitmap(tmp);
        }
     });
  }
...
```

Messages & Handlers

- Create Message & set Message content
  - Handler.obtainMessage()
  - Message.obtain()
  - Many variants. See documentation
- Message parameters include
  - int arg1, arg2
  - int what
  - Object obj
  - Bundle data

Messages & Handlers (cont.)

- sendMessage()
  - Queue Message immediately
- sendMessageAtFrontOfQueue()
  - Insert Message immediately at front of queue
- sendMessageAtTime()
  - Queue Message at the stated time
- sendMessageDelayed()
  - Queue Message after stated delay

Messages & Handlers (cont.)

```
public class SimpleThreadingExample extends Activity {
  …
  Handler handler = new Handler() {
    public void handleMessage(Message msg) {
      switch (msg.what) {
        case SET_PROGRESS_BAR_VISIBILITY: {
              progress.setVisibility((Integer) msg.obj);  break; }
        case PROGRESS_UPDATE: {
              progress.setProgress((Integer) msg.obj); break; }
        case SET_BITMAP: {
              iview.setImageBitmap((Bitmap) msg.obj);  break; }
      }
…
```

Messages & Handlers (cont.)

```
public void onCreate(Bundle savedInstanceState) {
  ...
  iview = ...
  progress = ...
  final Button button = ...
  button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      new Thread(
                new LoadIconTask(R.drawable.icon, handler)).start();
    }
  });
 }
...
```

Messages & Handlers (cont.)

```
private class LoadIconTask implements Runnable {
...
public void run() {
  Message msg = handler.obtainMessage (
              SET_PROGRESS_BAR_VISIBILITY, ProgressBar.VISIBLE);
  handler.sendMessage(msg);
  final Bitmap tmp =
              BitmapFactory.decodeResource(getResources(),resId);
  for (int i = 1; i < 11; i++) {
    msg = handler.obtainMessage(PROGRESS_UPDATE, i * 10);
    handler.sendMessageDelayed(msg, i * 200);
  }
...
```

Messages & Handlers (cont.)

```
...
  msg = handler.obtainMessage(SET_BITMAP, tmp);
  handler.sendMessageAtTime(msg, 11 * 200);
  msg = handler.obtainMessage(
              SET_PROGRESS_BAR_VISIBILITY, ProgressBar.INVISIBLE);
  handler.sendMessageAtTime(msg, 11 * 200);
  }
 }
}
```

Source Code Examples

- ThreadingSimple
- ThreadingRunOnUiThread
- ThreadingViewPost
- ThreadingAsyncTask
- ThreadingHandlerRunnable
- ThreadingHandlerMessages