



Android

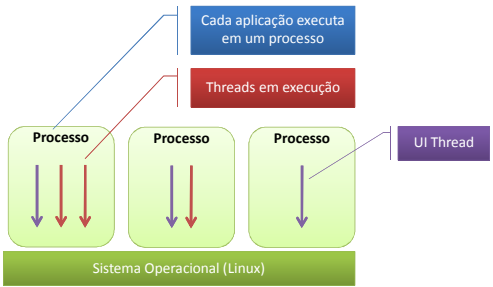
Threads e o Android

Softblue
cursos online

Tópicos Abordados

- Execução de aplicações no Android
- UI thread
- Handlers
 - Handlers e UI thread
 - Handlers e messages
- Tarefas assíncronas

Aplicações Executando no Android




Cada aplicação executa em um processo


Threads em execução

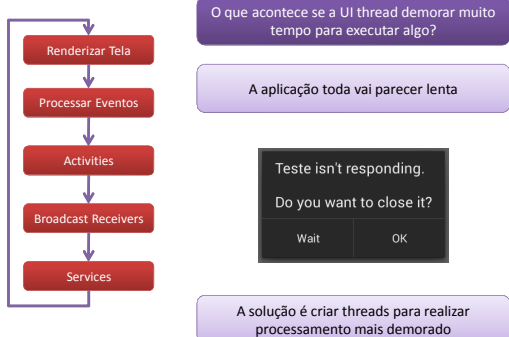
UI Thread

Sistema Operacional (Linux)

UI Thread


- É a thread principal da aplicação
- Cada aplicação tem uma UI thread
- Responsabilidades da UI thread
 - Desenhar a tela
 - Tratar eventos
 - Executar componentes
 - Activities
 - Broadcast receivers
 - Services

UI Thread





O que acontece se a UI thread demorar muito tempo para executar algo?

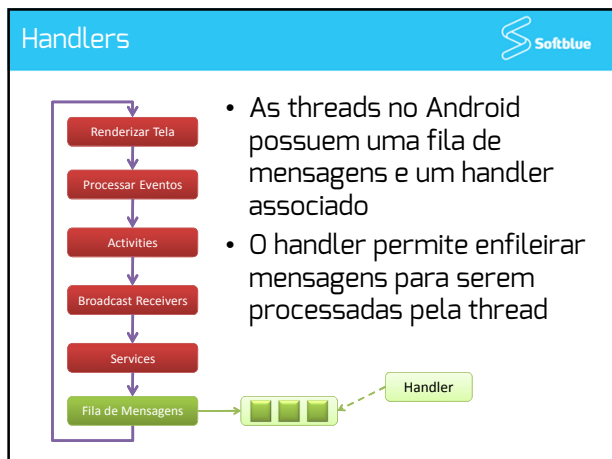
A aplicação toda vai parecer lenta

Teste isn't responding.
Do you want to close it?
Wait OK

A solução é criar threads para realizar processamento mais demorado

Threads no Android


- O Android é uma plataforma multithread
- É possível criar threads no Android da mesma forma que é feito no Java SE
 - Classe *Thread*
 - Interface *Runnable*
- A criação de threads que executam de forma independente da UI thread para atividades demoradas melhora a resposta da aplicação às ações do usuário



Handler e a UI Thread

- É comum o uso do handler da UI thread para alterar elementos da interface gráfica em threads que não sejam a UI thread
 - O Android só permite que a própria UI thread altere elementos da interface gráfica

Método	Descrição
<code>post(Runnable)</code>	Enfileira um <i>Runnable</i> imediatamente
<code>postDelayed(Runnable, long)</code>	Enfileira um <i>Runnable</i> com atraso
<code>postAtTime(Runnable, long)</code>	Enfileira um <i>Runnable</i> num determinado horário

Handler e a UI Thread

- Outra forma fácil de submeter uma alteração em componentes da interface à UI thread é usar o método **`runOnUiThread()`**

```
runOnUiThread(new Runnable() {
    public void run() {
        view.setText("msg");
    }
});
```

Handler e Messages



- Além de agendar um *Runnable* para ser executado, o handler permite enviar objetos do tipo **Message** para serem processados
- Neste caso é possível criar seu próprio handler, estendendo a classe *Handler*
 - Implementar o método **handleMessage()**

Método	Descrição
sendMessage(Message)	Enfileira a mensagem imediatamente
sendMessageDelayed(Message, long)	Enfileira a mensagem com atraso
sendMessageAtTime(Message, long)	Enfileira a mensagem num determinado horário

Handler e Messages



- A classe **Message**
 - Tem o atributo **what**, que pode ser usado para identificar a mensagem
 - Possui os métodos **setData()** e **getData()**, que permitem recuperar e associar um objeto *Bundle* à mensagem

Tarefas Assíncronas



- O Android tem a classe **AsyncTask**, que facilita a comunicação de uma thread arbitrária com a UI thread

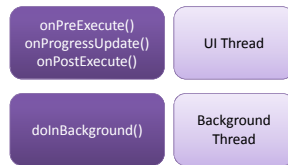
```
public class MyTask extends AsyncTask<Integer, Integer, Integer> {  
    protected Integer doInBackground(Integer... args) {  
    }  
    protected void onPostExecute() {  
    }  
    protected void onProgressUpdate(Integer... values) {  
    }  
    protected void onPostExecute(Integer result) {  
    }  
}
```

```
new MyTask().execute(100);
```

Tarefas Assíncronas



- Os métodos executam em threads diferentes



Regras das Tarefas Assíncronas



- Regras para que as *AsyncTask* funcionem
 - O método *doInBackground()* não pode interagir com elementos da interface gráfica
 - A instância de *AsyncTask* deve ser criada na UI thread
 - O método *execute()* deve ser invocado na UI thread
 - Não se deve executar diretamente os métodos herdados de *AsyncTask*
 - A tarefa pode ser executada apenas uma vez

Qual Técnica Usar?



- O método *runOnUiThread()* usa um handler internamente
 - Este método é apenas um atalho para o uso de handlers
- Tarefas assíncronas (*AsyncTask*) gerenciam o uso de múltiplas threads
 - *AsyncTask* é apenas um atalho para a criação de threads manualmente