



INSTITUTO FEDERAL
PARANÁ

DESENVOLVIMENTO PARA DISPOSITIVOS MÓVEIS

PROF^a. M.Sc. JULIANA H Q BENACCHIO



Threads, Handler e AsyncTask

- No Android, cada aplicação é executada em um único processo. Cada processo por sua vez tem uma **thread** dedicada.
- A classe **Handler** é utilizada para enviar ou agendar mensagens para serem executadas na **thread** principal da aplicação.
- A classe **AsyncTask** representa uma pequena biblioteca de **threads**.

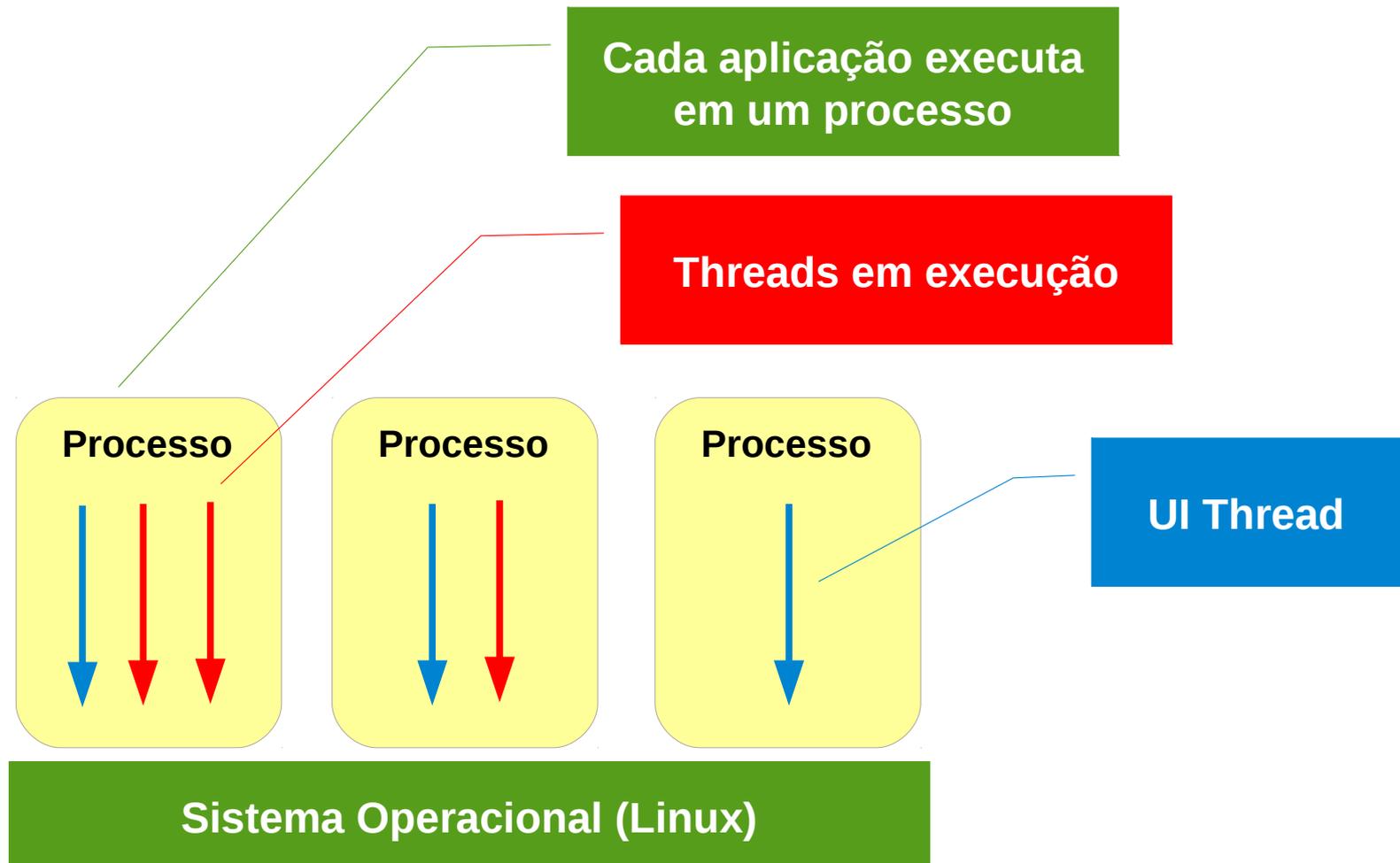


Threads

- Quando um aplicativo é aberto no Android, um processo dedicado no sistema operacional é criado para executá-lo
- Cada processo tem uma única **thread**, conhecida como **Main Thread** ou **UI Thread**, responsável por gerenciar todos os eventos da tela, assim como atualizar a interface gráfica.



Threads

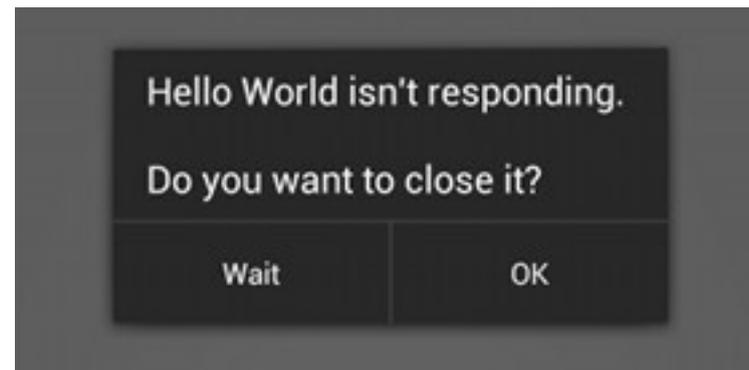
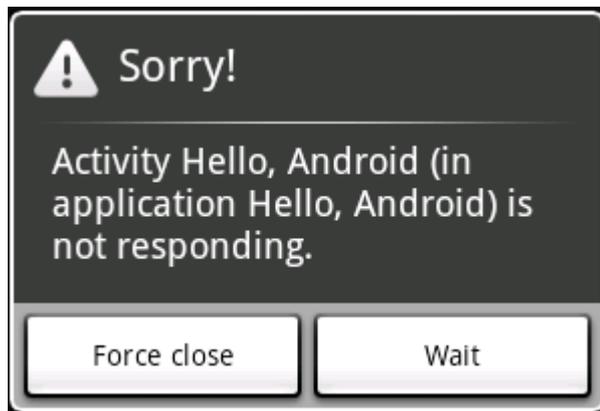


- É a **thread** principal da aplicação
- Cada aplicação tem uma **UI thread**
- Responsabilidades da **UI thread**
 - Desenhar a tela
 - Tratar eventos
 - Executar componentes
 - **Activities**
 - **Broadcast receivers**
 - **Services**



Responsividade

- A **thread** principal da aplicação deve responder aos eventos do usuário, em no máximo **cinco segundos**.
- Se esse tempo for ultrapassado, o erro **ANR** (***Application Not Responding***) será lançado.



- Como regra, toda operação de I/O, seja consultar um *web service*, ler um arquivo ou acessar o banco de dados, deve executar em uma **thread** separada para desvincular esse processamento da **thread principal**.

Se o código fizer uma operação de I/O na thread principal, o sistema vai lançar a exceção `NetworkOnMainThreadException`



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



Threads no Android

- Trecho de código em Java para executar um código em uma nova **thread**.

```
new Thread(new Runnable) {  
    public void run(){  
        //código que deve executar em segundo plano  
    };  
}.start();
```

- Uma **thread** deve ser filha da classe **Thread** e deve implementar o método **run()**.
- Ao chamar o método **start()**, a **thread** é iniciada, e o método **run()** vai executar em segundo plano.

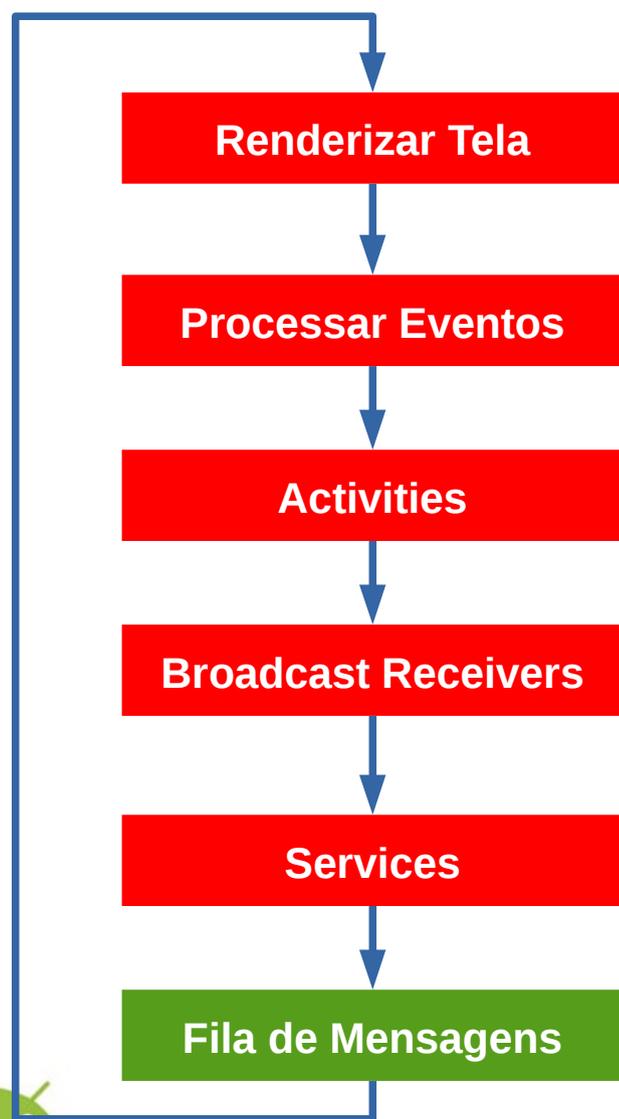


- Outra forma é utilizar o método **runOnUiThread(runnable)**, que é um atalho para utilizar um handler que está dentro da activity

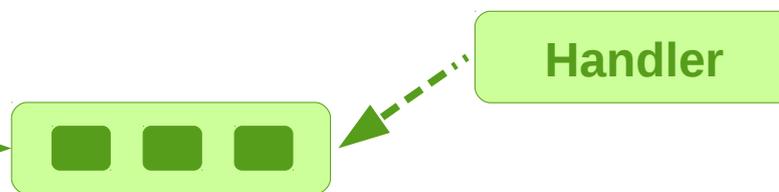
```
new Thread(new Runnable) {  
    public void run(){  
        //código que deve executar em segundo plano  
        runOnUiThread(new Runnable(){  
            public void run(){  
                //código que atualiza a interface  
            }  
        });  
    };  
}.start();
```



Handlers



- As **threads** no Android possuem uma fila de mensagens e um **handler** associado.
- O **handler** permite enfileirar mensagens para serem processadas pela **thread**.



- A classe **android.os.Handler** foi criada com o objetivo de enviar uma mensagem para a thread principal, para que, em algum momento apropriado, essa mensagem possa ser processada de forma segura e conseqüentemente atualizar a interface gráfica da tela (view).



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



Handler e a UI Thread

- Exemplo de utilização de threads e atualização de interface gráfica com um Handler

```
final Handler handler = new Handler();
new Thread() {
    public void run(){
        //código que deve executar em segundo plano
        handler.post(new Runnable(){
            public void run(){
                //código que atualiza a interface
            }
        });
    };
}.start();
```



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
<code>sendMessage (Message)</code>	Enfileira a mensagem imediatamente
<code>sendMessageDelayed (Message, long)</code>	Enfileira a mensagem com atraso
<code>sendMessageAtTime (Message, long)</code>	Enfileira a mensagem num determinado horário

Handler e Messages

- A classe **android.os.Message** tem o atributo **what**, que pode ser usado para identificar a mensagem

```
Message msg = new Message();  
msg.what = MENSAGEM_TESTE;  
handler.sendMessageDelayed(msg, 3000);
```

