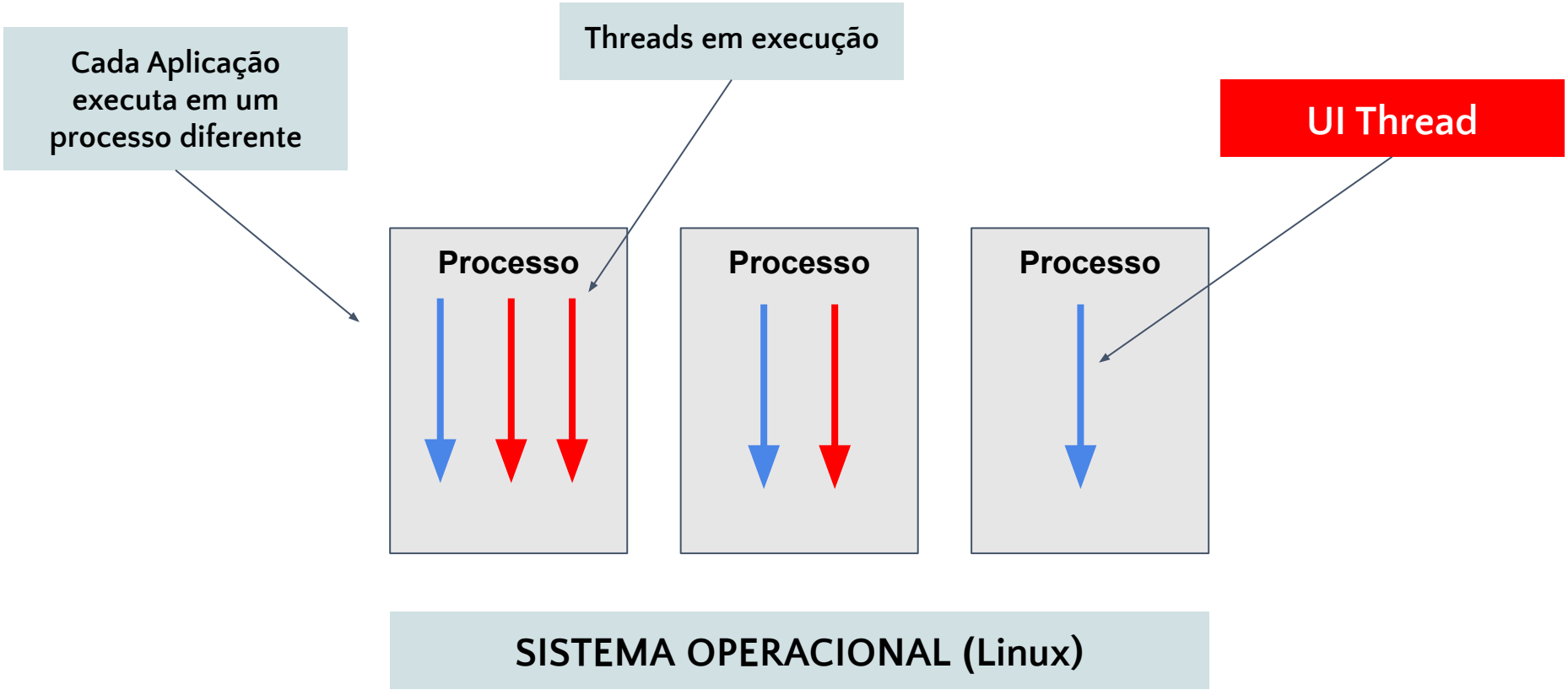
Threads

Professor Emerson Alencar
emerson@imd.ufrn.br

EXECUÇÃO DE APLICAÇÕES NO ANDROID



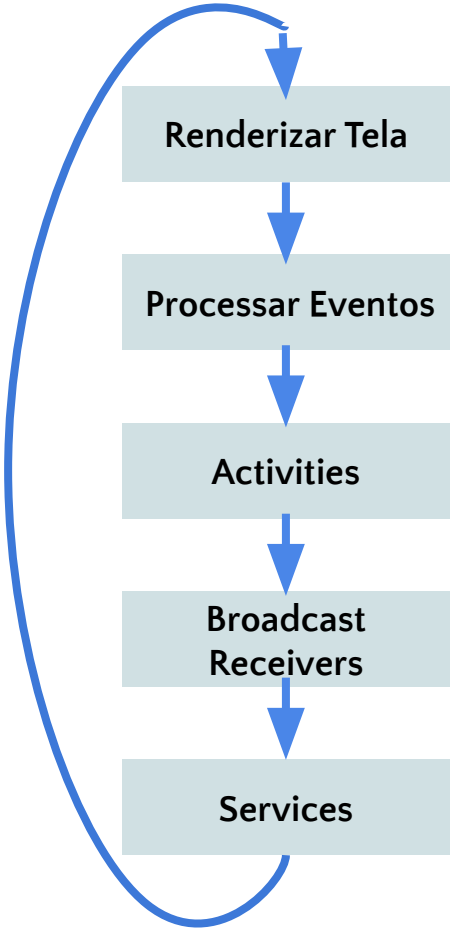
UI THREAD

- 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

THREADS

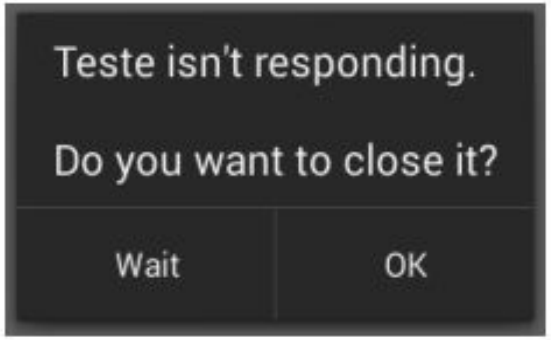
- Nas versões mais novas do Android, se o código fizer uma operação de I/O na thread principal, o sistema vai lançar a exceção ***NetworkOnMainThreadException***
- A thread principal do aplicativo deve responder aos eventos do usuário em, no máximo, cinco segundos. Se esse tempo for ultrapassado, o erro ANR (Application Not Responding) será lançado.

UI THREAD



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

A aplicação toda vai parecer lenta



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

THREAD

```
17
18      object: Thread() {
19          override fun run() {
20              txtTexto.text = "Texto Alterado na Thread"
21              Log.d( tag: "AppExemplo",  msg: "Texto da Thread")
22
23          }
24      }.start()
25
```

```
26
27      Thread{
28          txtTexto.text = "Texto Alterado na Thread 3"
29          Log.d( tag: "AppExemplo",  msg: "Texto da Thread")
30      }.start()
31
```

Handler

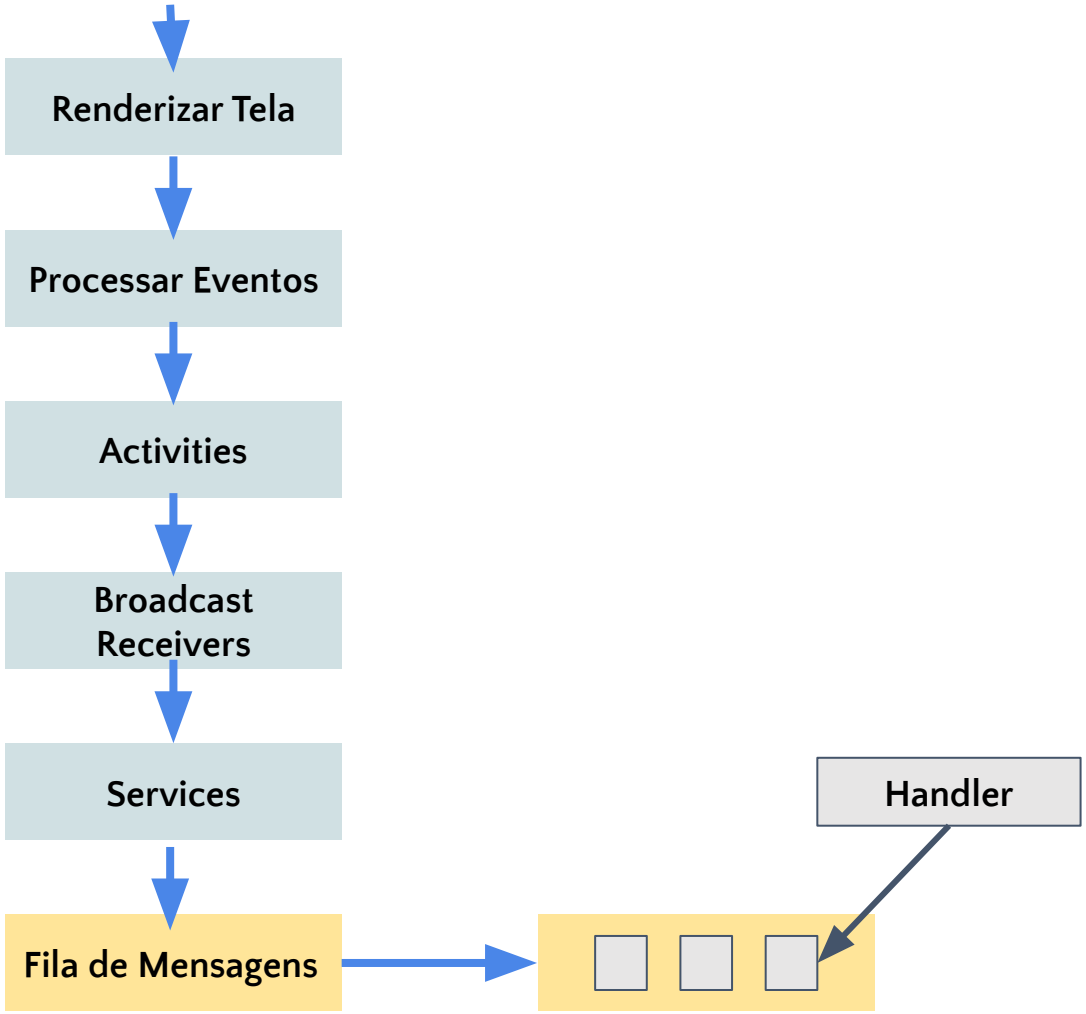
- Sempre que uma Thread é iniciada, temos um problema, pois, por questões de segurança e concorrência, não é permitido que uma thread diferente da principal atualize a interface gráfica
- Por isso, a classe android.os.Handler foi criada com o objetivo de enviar uma mensagem para a thread principal, para que, em algum momento apropriado, essa mensagem possa ser processada de forma segura e atualize a interface gráfica.

Handler

```
33
34          val handler = Handler()
35
36          Thread{
37              handler.post {
38                  //Código que atualiza a interface gráfica
39              }
40          }.start()
41
42
```

```
33
34          val handler = Handler()
35
36          Thread{
37              runOnUiThread { txtTexto.text = "Texto Alterado na Thread" }
38          }.start()
39
40
41      }
```

HANDLERS



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

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
post(Runnable)	Enfileira um *Runnable* imediatamente
postDelayed(Runnable, long)	Enfileira um *Runnable* com atraso
postAtTime(Runnable, long)	Enfileira um *Runnable* 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()**

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        setContentView(R.layout.activity_main)


        runOnUiThread { txtTexto.text = "Texto Alterado na Thread" }

    }
}
```

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

```
1   package com.example.exemplothread
2
3   import android.os.AsyncTask
4
5   class MyTask : AsyncTask<Int, Int, Int>(){
6
7       override fun onPreExecute() {
8           super.onPreExecute()
9       }
10
11      override fun onPostExecute(result: Int?) {
12          super.onPostExecute(result)
13      }
14
15      override fun onProgressUpdate(vararg values: Int?) {
16          super.onProgressUpdate(*values)
17      }
18
19      override fun doInBackground(vararg params: Int?): Int {
20          TODO( reason: "not implemented") //To change body of create
21      }
22  }
```

Val task = MyTask(this)
task.execute(10)

TAREFAS ASSÍNCRONAS

- Os métodos executam em threads diferentes
- **UI Thread:**
  - onPreExecute()
  - onProgressUpdate()
  - onPostExecute()
- **Background Thread:**
  - doInBackground()

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

Obrigado!

Dúvidas?

Professor Emerson Alencar
emerson@imd.ufrn.br