



Android Developer

Android Multi-Threading



Проверить, идет ли запись

```
if (видно && слышно) {  
    chat.print("+")  
}
```



Ставим "+", если все хорошо
"-", если есть проблемы



Тема вебинара

Android Multi-Threading



Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

24 года в IT, 8 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:

LinkedIn: <https://www.linkedin.com/in/motorro/>

GitHub: <https://github.com/motorro/>

Medium: <https://medium.com/@motorro>



Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в Slack **#канал группы**
или #general



Задаем вопрос
в чат или голосом



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или
задайте вопрос

Маршрут вебинара

Вспоминаем процессы и потоки

Необходимость многопоточности

Диспетчеризация задач

Исполнение кода

Диспетчеризация Android



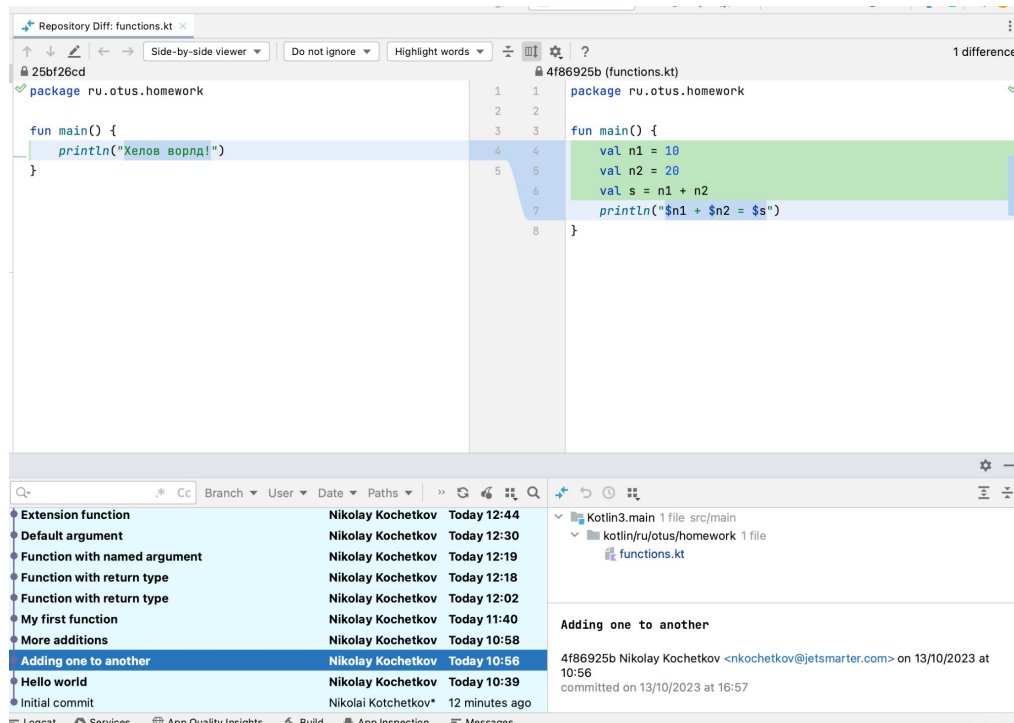
Репозиторий к занятию



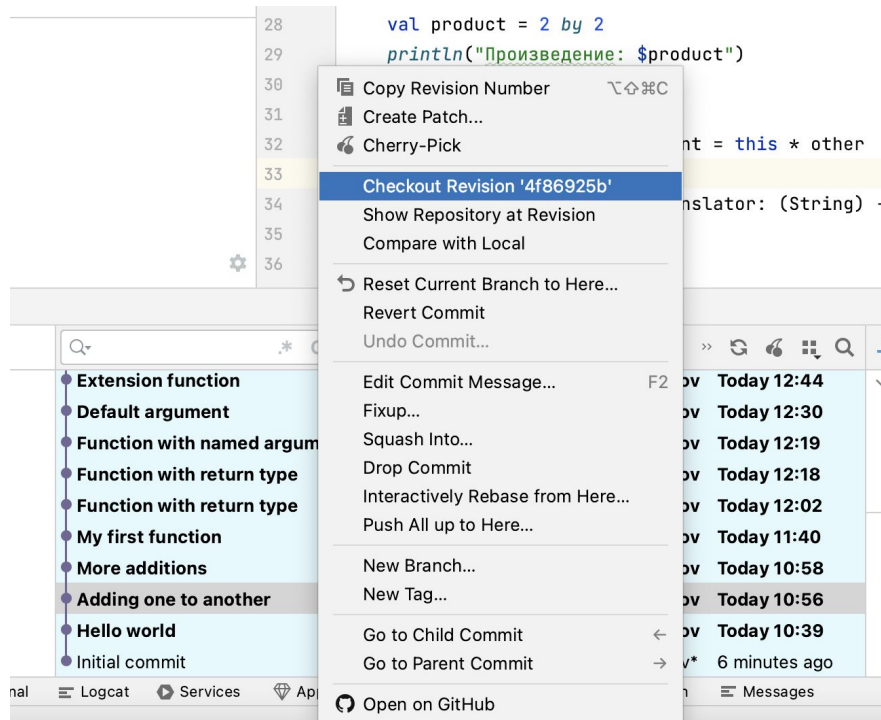
<https://github.com/Android-Developer-Basic/Multithreading>



Репозиторий к занятию - по шагам



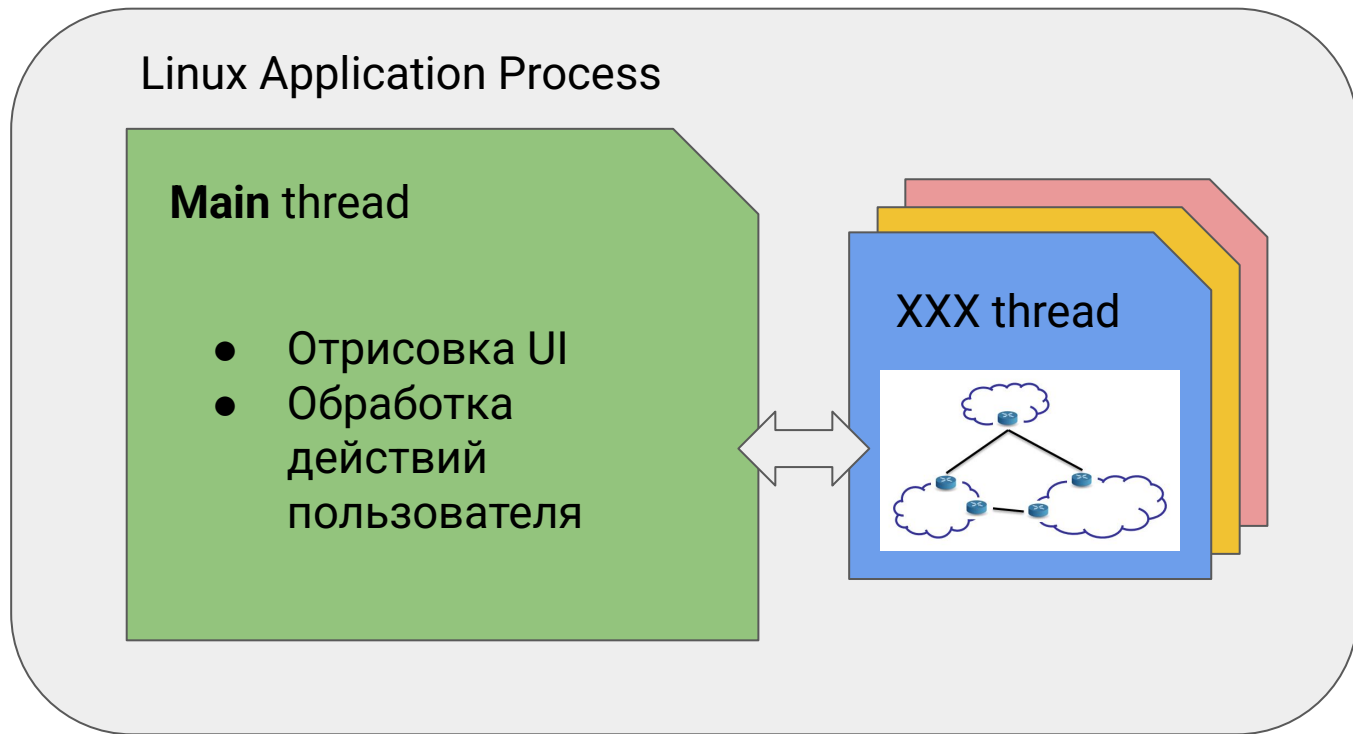
Репозиторий к занятию - по шагам



Процессы и потоки

Processes and Threads

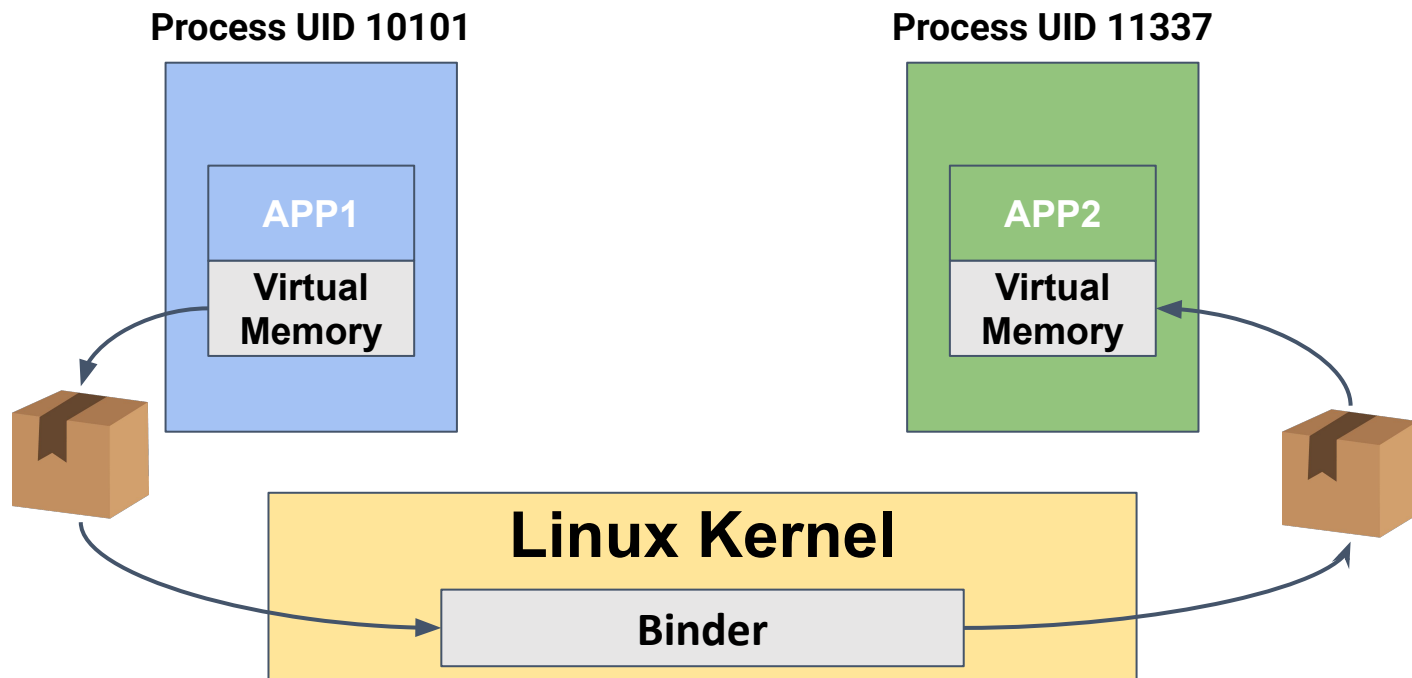
Процесс и потоки Android приложения



Процесс (process)

- Запущенное приложение
- Собственное пространство адресов
- Переключение процессов - длительная процедура.
Загружаются память и стеки.
- Взаимодействие между процессами требует дополнительных механизмов (IPC)
- ОС управляет памятью процессов (может вытеснить низкоприоритетный процесс на дисковый своп)
- Запуск и переключение процесса - длительный с т.з. операционной системы

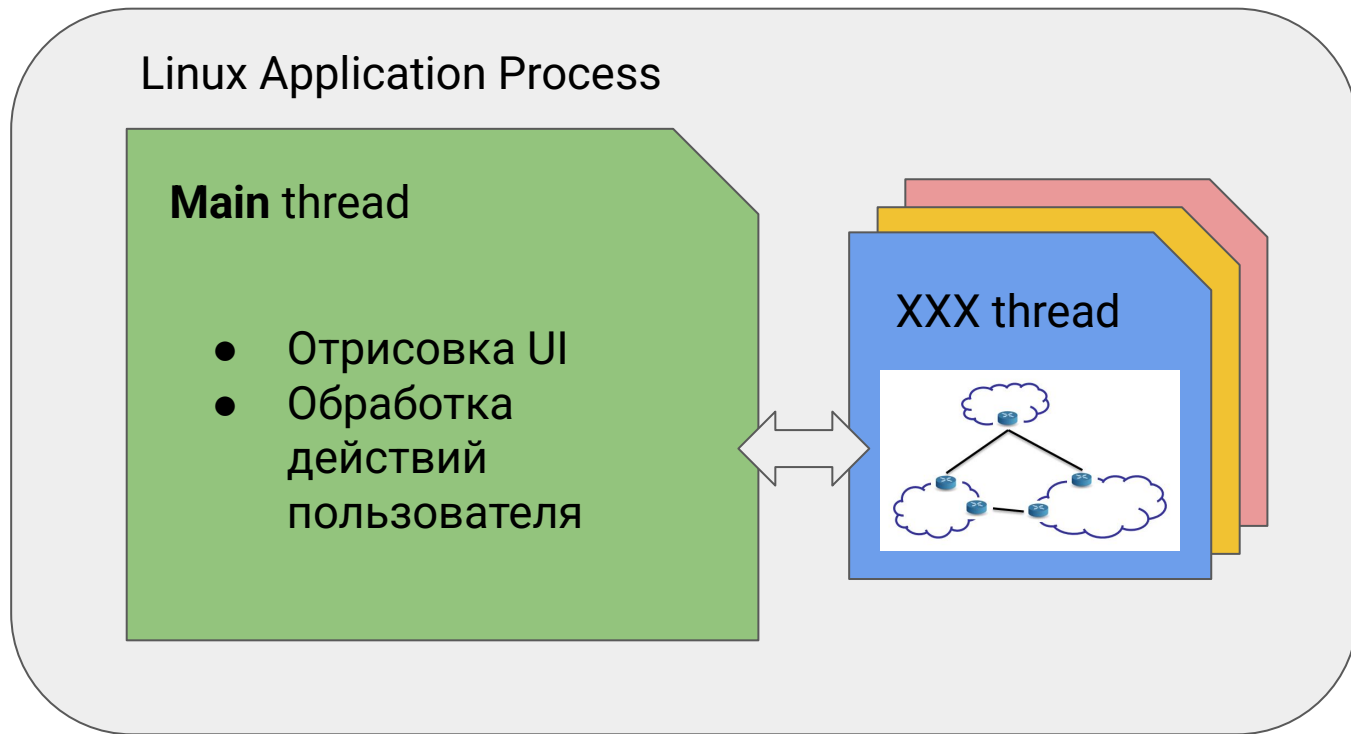
Процесс (process)



Поток (thread)

- Позволяет процессу запускать задачи параллельно
- Общее пространство адресов внутри процесса (heap)
- Запуск потока занимает меньше времени
- Переключение потока занимает меньше времени - загрузка только стека.
- Потоки легко* взаимодействуют между собой (общая память)
- ОС не управляет памятью потоков*

Процесс и потоки Android приложения



Операции ввода-вывода

```
3 ▶ fun main() {
```

```
4     print("Enter your name: ")
```

```
5     val name = readln() ←
```

Поток будет ждать в этой точке, пока пользователь не введет строку

```
6     println("Hello, $name!")
```

```
7 }
```

Операции ввода-вывода блокирующие!

Длительные вычисления

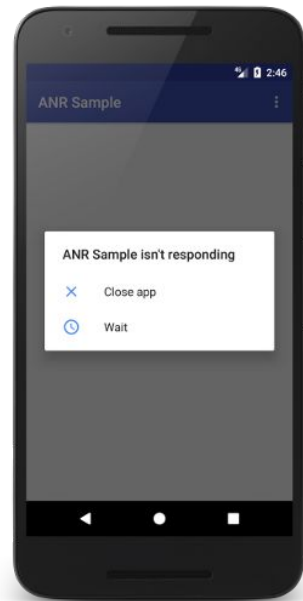
```
5  ▶ fun main() {  
6    val timeElapsed = measureTimeMillis {  
7      calculate()  
8    }  
9    println("Time elapsed: $timeElapsed")  
10 }  
11  
12 private fun calculate() {  
13   println("Starting calculations...")  
14   Thread.sleep(200L)  
15   println("Done!")  
16 }
```

```
Starting calculations...  
Done!  
Time elapsed: 204
```

↑
Время выполнения

Длительные вычисления

- Частота обновления экрана на телефоне: 60Hz
- Время на обработку одного кадра:
 $1 / 60 = 0.017\text{c} = 17\text{ms}$




Вывод

Нельзя блокировать Main поток

Вопросы?

Создаем поток

```
5 ▶ fun main() {  
6     log { "Starting thread..." }  
7     val thread = thread(start = true, name = "Worker") {  
8         log { "Thread task" }  
9     }  
10    thread.join()  
11    log { "Task complete" }  
12 }
```



```
val thread = object : Thread() {  
    public override fun run() {  
        block()  
    }  
}
```

Накладные расходы

```
12 private fun doInMain() {  
13     val counter = AtomicInteger(0)  
14     println("Doing in main thread...")  
15     (0 ≤ .. ≤ 10_000).forEach {  
16         counter.incrementAndGet()  
17     }  
18     println("Counter: ${counter.get()}")  
19 }
```

```
21 private fun doInThread() {  
22     val counter = AtomicInteger(0)  
23     println("Doing in other thread...")  
24     (0 ≤ .. ≤ 10_000).forEach {  
25         thread { counter.incrementAndGet() }.join()  
26     }  
27     println("Counter: ${counter.get()}")  
28 }
```

Doing in main thread...
Counter: 10001
Main thread: 7
Doing in other thread...
Counter: 10001
Other thread: 729

Thread Pool

```
26 private class ThreadPool(threads: Int) {
27     @Volatile
28     private var running = true
29     private val queue = ConcurrentLinkedQueue<Runnable>()
30     private val threads = (1 ≤ .. ≤ threads).map {
31         thread(start = false, name = "Thread-$it") {
32             while (running) {
33                 queue.poll()?.run()
34             }
35         }
36     }
37
38     fun enqueue(runnable: Runnable) {...}
41
42     fun start() {...}
50 }
```

← Очередь задач

1. Достаем задачу из очереди
2. Выполняем задачу

Thread Pool

```
9  ▶ fun main() {  
10      log { "Starting thread pool..." }  
11      val pool = ThreadPool( threads: 10)  
12      val counter = AtomicInteger(0)  
13      val time = measureTimeMillis {  
14          (0 ≤ .. ≤ 10_000).forEach {  
15              pool.enqueue {  
16                  counter.incrementAndGet()  
17              }  
18          }  
19          pool.start()  
20          println("Counter: ${counter.get()}")  
21      }  
22      println("Total time: $time")  
23  }
```

```
Thread: main: Starting thread pool...  
Counter: 10001  
Total time: 48
```

Вопросы?

Проблемы параллельного доступа

Проблемы параллельного доступа

```
5 ▶ fun main() {  
6     log { "Starting thread pool..." }  
7     val pool = ThreadPool( threads: 10)  
8     var counter = 0  
9     val time = measureTimeMillis {  
10         (0 ≤ .. ≤ 10_000).forEach {  
11             pool.enqueue {  
12                 ++counter  
13             }  
14         }  
15         pool.start()  
16         println("Counter: $counter")  
17     }  
18 }
```

Обычная переменная

Изменяем из нескольких
ПОТОКОВ

Проблемы параллельного доступа

```
Thread: main: Starting thread pool...
```

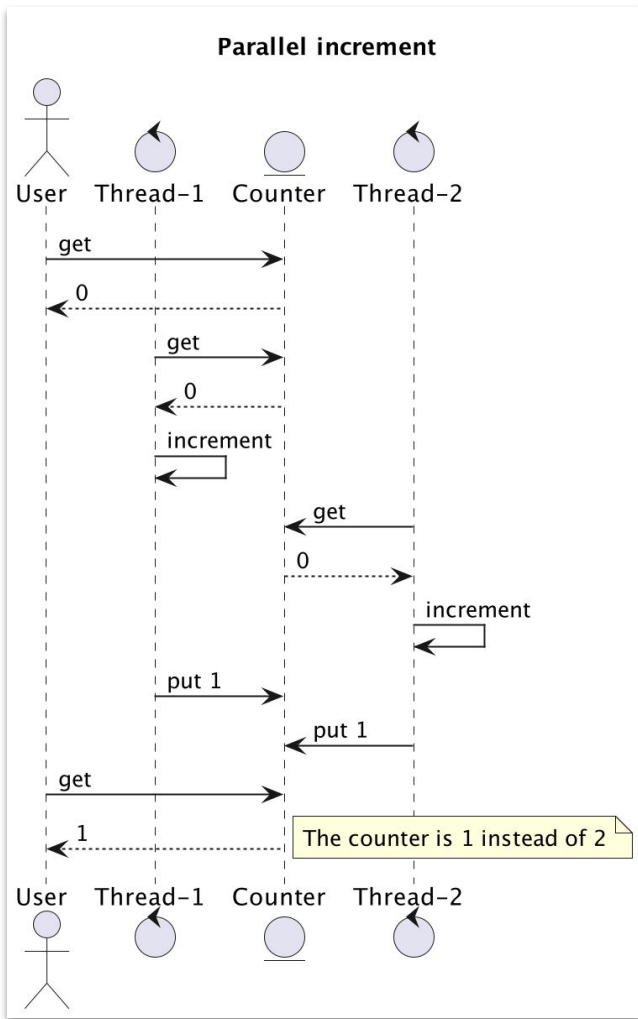
```
Counter: 9581
```

```
Thread: main: Starting thread pool...
```

```
Counter: 9796
```

```
Thread: main: Starting thread pool...
```

```
Counter: 9827
```



Проблемы параллельного доступа

- Неатомарные операции
- Кэширование памяти
- Синхронизация блокирует потоки
- Сложность реализации потокобезопасного кода

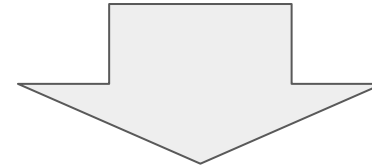
Thread Pool

```
9  ▶ fun main() {  
10      log { "Starting thread pool..." }  
11      val pool = ThreadPool( threads: 10)  
12      val counter = AtomicInteger(0)  
13      val time = measureTimeMillis {  
14          (0 ≤ .. ≤ 10_000).forEach {  
15              pool.enqueue {  
16                  counter.incrementAndGet()  
17              }  
18          }  
19          pool.start()  
20          println("Counter: ${counter.get()}")  
21      }  
22      println("Total time: $time")  
23  }
```

```
Thread: main: Starting thread pool...  
Counter: 10001  
Total time: 48
```

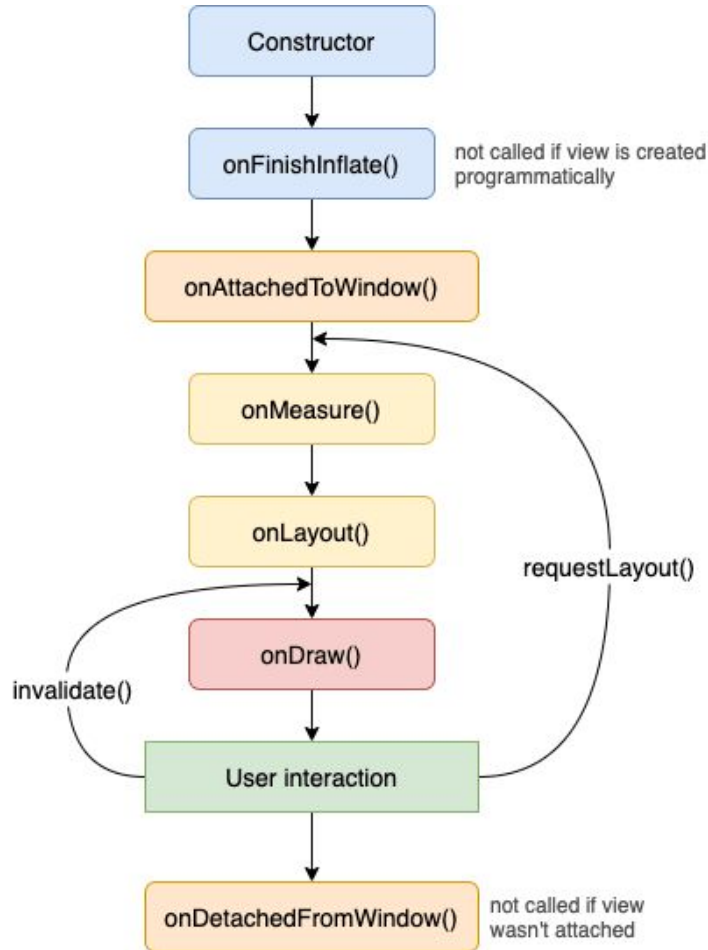
View Lifecycle

- Критично малое время на обработку кадра
- Сложность реализации
- Множество компонентов



View - не thread-safe подсистема*

* Compose - немного другая история



Вывод

Не изменять View из других
ПОТОКОВ

Вопросы?

Thread Pools

Cached thread pool

```
5 ▶ fun main() {  
6     val executor = Executors.newCachedThreadPool()  
7     repeat( times: 10) {  
8         executor.execute {  
9             log { "Running a task..." }  
10        }  
11    }  
12    executor.shutdown()  
13 }
```

- Если в пуле нет свободного потока - он создается
- Если поток неактивен какое-то время - он удаляется
- Применяем для операций, где время переключения не критично (IO)

Cached thread pool

```
Thread: pool-1-thread-6: Running a task...  
Thread: pool-1-thread-2: Running a task...  
Thread: pool-1-thread-5: Running a task...  
Thread: pool-1-thread-7: Running a task...  
Thread: pool-1-thread-4: Running a task...  
Thread: pool-1-thread-10: Running a task...  
Thread: pool-1-thread-3: Running a task...  
Thread: pool-1-thread-8: Running a task...  
Thread: pool-1-thread-1: Running a task...  
Thread: pool-1-thread-9: Running a task...
```

Fixed thread pool

```
5 ▶ fun main() {  
6     val executor = Executors.newFixedThreadPool(2)  
7     repeat( times: 10) {  
8         executor.execute {  
9             log { "Running a task..." }  
10        }  
11    }  
12    executor.shutdown()  
13 }
```

- Если в пуле нет свободного потока - задача ожидает в очереди
- Малое кол-во потоков обуславливает малое кол-во переключений
- Используем для интенсивных вычислений



Fixed thread pool

```
Thread: pool-1-thread-2: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-2: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-2: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-2: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-2: Running a task...
Thread: pool-1-thread-1: Running a task...
```

Single thread pool

```
5 ▶ fun main() {  
6     val executor = Executors.newSingleThreadExecutor()  
7     repeat( times: 10) {  
8         executor.execute {  
9             log { "Running a task..." }  
10        }  
11    }  
12    executor.shutdown()  
13 }
```

- Задачи ожидают в очереди в порядке поступления
- Используем для задач, требующих выполнения по порядку (UI)

Single thread pool

```
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
Thread: pool-1-thread-1: Running a task...
```



Библиотечные диспетчеры

Описание	Coroutines	RxJava
Основной (View) поток*	MAIN	UI
Для расчетов	DEFAULT	COMPUTATION
Для IO	IO	IO

* зависит от платформы и предоставляется отдельной библиотекой

Хорошая статья про диспетчеры Coroutines



<https://kt.academy/article/cc-dispatchers#definition-2>

Вопросы?

Выполнение кода...

Функция

```
18 suspend fun perform(takesTime: Int = 1000, block: () -> String) {  
19     log { "Starting task: ${block()}" }  
20     repeat( times: takesTime / 100) {  
21         Thread.sleep(100) ← Блокирует поток  
22     }  
23     log { "Finished task: ${block()}" }  
24 }
```

Последовательное выполнение

```
6 ► fun main() = runBlocking {  
7     log { "Делаем бургер..." }  
8     val time = measureTimeMillis {  
9         kneadDough()  
10        grindMeat()  
11        bakeBuns()  
12        roastPatty()  
13        assemble()  
14    }  
15    log { "Готово! Время: $time" }  
16 }
```

Thread: main: Делаем бургер...

Thread: main: Starting task: Месим тесто

Thread: main: Finished task: Месим тесто

Thread: main: Starting task: Рубим мясо

Thread: main: Finished task: Рубим мясо

Thread: main: Starting task: Печем булку

Thread: main: Finished task: Печем булку

Thread: main: Starting task: Жарим котлету

Thread: main: Finished task: Жарим котлету

Thread: main: Starting task: Собираем

Thread: main: Finished task: Собираем

Thread: main: Готово! Время: 5497

Пытаемся оптимизировать...

```
8  ▶ fun main() = runBlocking {  
9      log { "Делаем бургер..." }  
10     val time = measureTimeMillis {  
11         val buns = launch {  
12             kneadDough()  
13             bakeBuns()  
14         }  
15         val patties = launch {  
16             grindMeat()  
17             roastPatty()  
18         }  
19         joinAll(buns, patties)  
20         assemble()  
21     }  
22     log { "Готово! Время: $time" }  
23 }
```

```
Thread: main: Делаем бургер...  
Thread: main: Starting task: Месим тесто  
Thread: main: Finished task: Месим тесто  
Thread: main: Starting task: Печем булку  
Thread: main: Finished task: Печем булку  
Thread: main: Starting task: Рубим мясо  
Thread: main: Finished task: Рубим мясо  
Thread: main: Starting task: Жарим котлету  
Thread: main: Finished task: Жарим котлету  
Thread: main: Starting task: Собираем  
Thread: main: Finished task: Собираем  
Thread: main: Готово! Время: 5508
```

???

Пытаемся сократить время...

```
25 suspend fun perform(takesTime: Int = 1000, block: () -> String) {  
26     log { "Starting task: ${block()}" }  
27     repeat( times: takesTime / 100) {  
28         Thread.sleep(100) ←  
29     }  
30     log { "Finished task: ${block()}" }  
31 }
```

Было

```
26 suspend fun perform(takesTime: Int = 1000, block: () -> String) {  
27     log { "Starting task: ${block()}" }  
28     repeat( times: takesTime / 100) {  
29         Thread.sleep(100)  
30         yield() ←  
31     }  
32     log { "Finished task: ${block()}" }  
33 }
```

Стало

Конкурентное выполнение

Поток один - main

```
Thread: main: Делаем бургер...
Thread: main: Starting task: Месим тесто
Thread: main: Starting task: Рубим мясо
Thread: main: Finished task: Рубим мясо
Thread: main: Starting task: Жарим котлету
Thread: main: Finished task: Месим тесто
Thread: main: Starting task: Печем булку
Thread: main: Finished task: Жарим котлету
Thread: main: Finished task: Печем булку
Thread: main: Starting task: Собираем
Thread: main: Finished task: Собираем
Thread: main: Готово! Время: 5488
```



Конкурентное выполнение

- При конкурентном выполнении, поток разделяется между корутинами диспетчером.
- Разделение должно быть добровольным!

Параллельное выполнение

```
26 suspend fun perform(takesTime: Int = 1000, block: () -> String) {  
27     log { "Starting task: ${block()}" }  
28     repeat( times: takesTime / 100) {  
29         Thread.sleep(100)  
30     }  
31     log { "Finished task: ${block()}" }  
32 }
```

Убираем yield

Параллельное выполнение

```
10 ▶ fun main() = runBlocking {  
11     log { "Делаем бургер..." }  
12     val time = measureTimeMillis {  
13         val bun = launch(Dispatchers.Default) {  
14             kneadDough()  
15             bakeBuns()  
16         }  
17         val patty = launch(Dispatchers.Default) {  
18             grindMeat()  
19             roastPatty()  
20         }  
21         joinAll(bun, patty)  
22         assemble()  
23     }  
24     log { "Готово! Время: $time" }  
25 }
```

Переходим на другой пул потоков

Параллельное выполнение

Несколько
ПОТОКОВ

Thread: main: Делаем бургер...
Thread: DefaultDispatcher-worker-1: Starting task: Месим тесто
Thread: DefaultDispatcher-worker-2: Starting task: Рубим мясо
Thread: DefaultDispatcher-worker-2: Finished task: Рубим мясо
Thread: DefaultDispatcher-worker-2: Starting task: Жарим котлету
Thread: DefaultDispatcher-worker-1: Finished task: Месим тесто
Thread: DefaultDispatcher-worker-1: Starting task: Печем булку
Thread: DefaultDispatcher-worker-2: Finished task: Жарим котлету
Thread: DefaultDispatcher-worker-1: Finished task: Печем булку
Thread: main: Starting task: Собираем
Thread: main: Finished task: Собираем
Thread: main: Готово! Время: 4151

← Время

Параллельное выполнение

- При параллельном выполнении, разные пути программы выполняются в разных потоках.
- Блокирующие операции не блокируют основной поток!

Delay в нормальной жизни

```
26 suspend fun perform(takesTime: Long = 1000, block: () -> String) {  
27     log { "Starting task: ${block()}" }  
28     delay(takesTime) ← Suspending...  
29     log { "Finished task: ${block()}" }  
30 }
```

Thread: main: Делаем бургер...

Thread: main: Starting task: Месим тесто

Thread: main: Starting task: Рубим мясо

Thread: main: Finished task: Рубим мясо

Thread: main: Starting task: Жарим котлету

Thread: main: Finished task: Месим тесто

Thread: main: Starting task: Печем булку

Thread: main: Finished task: Жарим котлету

Thread: main: Finished task: Печем булку

Thread: main: Starting task: Собираем

Thread: main: Finished task: Собираем

Thread: main: Готово! Время: 4028

← Время

Вопросы?

Конкуренция в Android Handler, Looper, MessageQueue

Компоненты Android

- В Android-приложении всегда есть Main-поток
- Действия с View должны происходить на Main-потоке
- Блокировать Main-поток нельзя. Это приводит к ANR

Компоненты Android

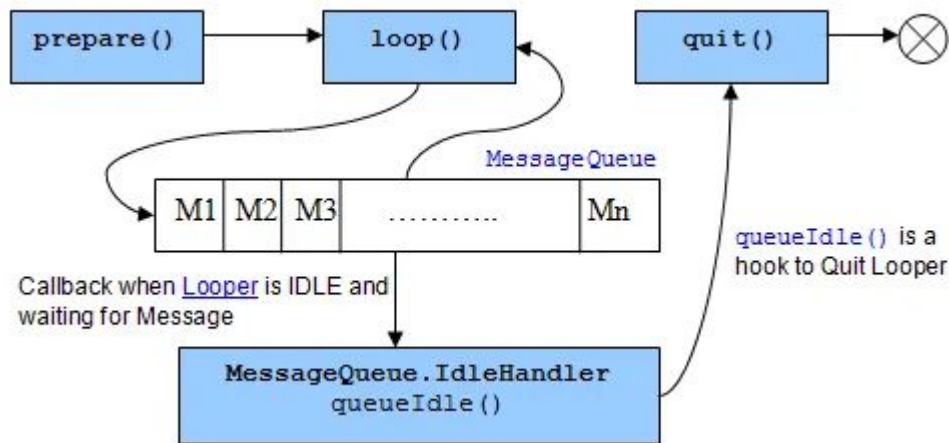


Иллюстрация <https://i.stack.imgur.com/UHY9C.png>

- **Thread** - запущенный поток
- **Looper** - диспетчер очереди задач
- **MessageQueue** - очередь сообщений
- **Handler** - посылает сообщения в очередь

Компоненты Android

```
56 private class BgThread : Thread() {
57     private lateinit var handler: Handler
58
59     init {
60         this.name = "BgThread"
61     }
62
63     override fun run() {
64         super.run()
65         log { "Running thread..." }
66         Looper.prepare()
67         handler = Handler(requireNotNull(Looper.myLooper())) { "Should have a looper" }
68         Looper.loop()
69         log { "Thread complete" }
70     }
71
72     fun post(runnable: Runnable) {...}
73
74
75
76     fun terminate() {...}
77 }
78 }
```

1. Запускаем поток
2. Создаем looper и очередь
3. Запускаем цикл

Посылаем сообщения через Handler

```
56 private class BgThread : Thread() {  
57     private lateinit var handler: Handler  
58  
59     init {  
60         this.name = "BgThread"  
61     }  
62  
63     override fun run() {...}  
71  
72     fun post(runnable: Runnable) {  
73         handler.post(runnable)  
74     }  
75  
76     fun terminate() {...}  
81 }  
82 }
```

Thread

```
44 private fun sendMessage() {  
45     th?.post {  
46         log {...}  
49     }  
50 }
```

Activity

```
I Thread: BgThread: Running thread...  
D Installing profile for otus.gpb.multithreading  
D app_time_stats: avg=98.83ms min=4.55ms max=4321.30ms count=51  
I Thread: BgThread: Current time: 15:20:53.518164  
D app_time_stats: avg=23.28ms min=9.95ms max=352.28ms count=52  
I Thread: BgThread: Current time: 15:20:54.782391
```

Console

Попытка изменить View из другого потока

```
45     private fun sendMessage() {  
46         th?.post {  
47             val currentTime = Clock.System.now().toLocalDateTime(TimeZone.currentSystemDefault()).time  
48             log { "Current time: $currentTime" }  
49             log { "Logging current time..." }  
50             binding.message.text = getString(R.string.updated_at, currentTime)  
51         }  
52     }
```

Activity

android.view.ViewRootImpl\$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.

Console

Возврат на основной поток

```
45 private fun sendMessage() {  
46     th?.post {  
47         val currentTime = Clock.System.now().toLocalDateTime(TimeZone.currentSystemDefault()).time  
48         log { "Current time: $currentTime" }  
49         Handler(Looper.getMainLooper()).post { ← Looper главного потока  
50             log { "Logging current time..." }  
51             binding.message.text = "Updated at: {currentTime}"  
52         }  
53     }  
54 }
```

Activity

```
I Thread: BgThread: Current time: 15:29:46.165365  
I Thread: main: Logging current time...
```

Console

Выводы

1. Изменять View можно только из основного потока
2. Все обращения к основному потоку из других потоков проходят через диспетчер Handler/MessageQueue



1. Переключение потоков может занять время
2. Не стоит переходить на другой поток, если у вас нет блокировок или вы помещаетесь в кадр

Диспетчеры главного потока*

Coroutines: Dispatchers.Main(.immediate)

RxJava: Schedulers.UI

* Оба диспетчера устанавливаются отдельной библиотекой

Рефлексия

Маршрут вебинара

Вспоминаем процессы и потоки

Необходимость многопоточности

Диспетчеризация задач

Исполнение кода

Диспетчеризация Android

Вопросы для проверки

По пройденному материалу всего вебинара

1. Почему нам приходится пользоваться другими потоками?
2. Чем нам поможет Thread Pool?
3. Почему View должен работать в основном потоке?
4. Как организована диспетчеризация в Android?
5. Почему недолгие операции стоит делать прямо во View-потоке?



Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то, что узнали на вебинаре?

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**

Спасибо за внимание!

Приходите на следующие вебинары



Николай Кочетков

Руководитель Андроид разработки FlyXO

Об опыте (например):

24 года в IT, 8 лет - играющий тренер Андроид

Телефон / эл. почта / соц. сети:

LinkedIn: <https://www.linkedin.com/in/motorro/>

GitHub: <https://github.com/motorro/>

Medium: <https://medium.com/@motorro>

