Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт лабораторной работе №6**

Дисциплина: Проектирование мобильных приложений

Тема: Многопоточные Android приложения

Выполнил студент гр. 3530901/90201 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Швецов Д.С.

(подпись)

Принял \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузнецов А. Н.

(подпись)

“ ” 2022 г.

Санкт-Петербург

2022

**Цели работы:**

Получить практические навыки разработки многопоточных приложений:

1. Организация обработки длительных операций в background (worker) thread:
   * Запуск фоновой операции (Coroutine/ExecutionService/Thread)
   * Остановка фоновой операции (Coroutine/ExecutionService/Thread)
2. Публикация данных из background (worker) thread в main (ui) thread.

Освоить 3 основные группы API для разработки многопоточных приложений:

1. Kotlin Coroutines
2. ExecutionService
3. Java Threads

**Задание №1.**

В данном задание нам необходимо решить задачу несекундомер разными способами(На обычных потоках, executorService и Kotlin Coroutines)

Способ №1. Java Threads.

Фактически, в лаб 2. Я уже привел правильную реализацию, стоит только убедиться, что когда приложения нет на экране никаких потоков не существует и при пересоздании активити в системе не работаю несколько потоков.

Напомним вариант решения с Thread

Изображение выглядит как текст

Автоматически созданное описание

Попробуем с помощью команд в adb shell посмотреть, посмотреть состояния потоков.

Обычная работа:

Изображение выглядит как стол

Автоматически созданное описание

Видим, что работает Thread811

Переведем в Background.

Изображение выглядит как текст

Автоматически созданное описание

Thread 811 исчез

Вернем из Background.

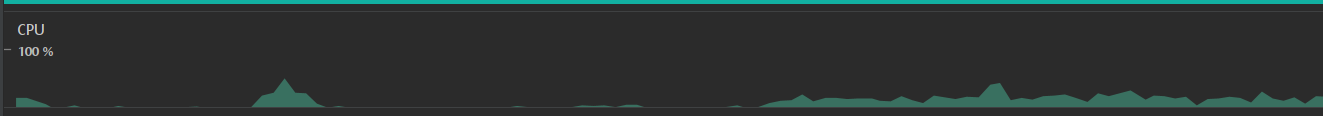
Изображение выглядит как текст

Автоматически созданное описание

Видим, что создался новый Thread 814, Thread 811 уже в системе нет.

Изображение выглядит как текст, снимок экрана, монитор

Автоматически созданное описание

Также если посмотреть по профилировщику, то нагрузки после поворота дополнительной нет. Значит, приложение работает верно.

**Решение №2.**

Решение с помощью ExecutorService.

Изображение выглядит как текст

Автоматически созданное описание

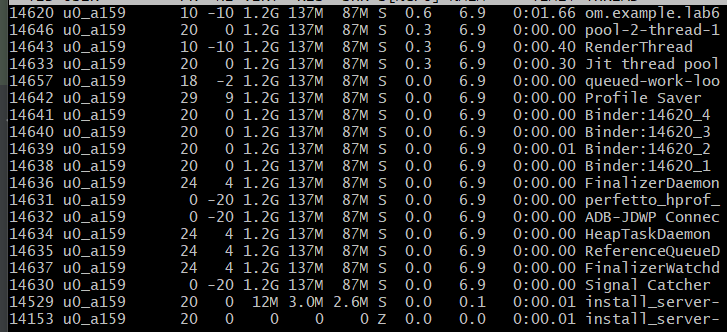
Мы просто отдаем Future в executor и он выполняет ее, считая наши секунды, когда мы хотим остановить подсчет мы отменяем наше Future, но чтобы executorService не создавался при каждом пересоздание Activity мы переносим его в класс, который наследуется от application, то есть ExecutorService будет уничтожен только при закрытие приложения.

Изображение выглядит как текст

Автоматически созданное описание

Для выполнения подсчета нам хватит одного потока.

Посмотрим процессы.



Видим, что работает pool-2-thread-1, после поворота экрана он же и остается.

**Решение №3.**

Решение с помощью kotlin coroutines.

Изображение выглядит как текст

Автоматически созданное описание

В данном случае мы в resume запускаем корутину с возвращаем Job( аналог future), как мы видим, корутина запускается в Dispatchers.Main, поэтому мы можем напрямую выставлять значение text. В onPause мы отменяем выполнение Job и счет приостанавливается, isActive становится false, когда Job не активна.

**Задание №2.**

Скачивание картинки в background потоке и вывод ее на экран.

Решим эту задачу с помощью ExecutorService, не забываем, что здесь нам тоже нужно использовать Application.

Изображение выглядит как текст

Автоматически созданное описание

Вид решения

Изображение выглядит как текст, цветок, растение, снимок экрана

Автоматически созданное описание

**Задание №3.**

Здесь мы выполняем то же самое, что и в задание 2, но только с помощью корутин.

Изображение выглядит как текст

Автоматически созданное описание

Как мы видим, скачиваем картинку мы в отдельном потоке, потом переходим в поток UI и выставляем картинку на экране.

При завершении активити downloading job будет завершена, следовательно, будут завершены и внутренние корутины.

**Задание №4.**

Используем библиотеку Picasso для выполнения тех же самых действий.

Изображение выглядит как текст

Автоматически созданное описание

Все также работает правильно.

**Вывод:**

В данной лабораторной работе мы научились скачивать картинку в потоке отдельном от потока UI, реализовали правильный подсчет секунд, сделав это несколькими способами, обобщим полученные знания.

1. Java Threads.

Самый примитивный вид API, создается с помощью инициализации объекта Thread и метода start(), внутри потока исполняется объект Runnable, чтобы завершить Thread мы должны любыми способами выйти из Runnable.

1. ExecutorService

Это обертка над потоками, которая поддерживает удобное управление ими, мы можем создать пул из нескольких потоков, одного, ставить им задачи и по мере освобождения поток будет принимать задачу и выполнять, отменить задачу мы можем методом Future.cancel(), а остановить весь ExecutorService, методами shutdown() и shutdownNow().

1. Kotlin Coroutines

Так называемые легкие потоки, у них есть понятие CoroutineScope, он нужен для вложенности Coroutine, также есть объект Dispatcher, который определяет, где будет выполнятся корутина, запустить задачу на исполнение можно методом coroutine. launch, который возвращает объект Job, мы можем отменить Job методом cancel (), корутина же завершается, когда у нее нет активных Job.