Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Task-manager для ОС Android

БГУИР КП 1-40 04 01 004 ПЗ

Выполнил: студент группы 753505  
Осипик И.Ф.

Руководитель: ассистент кафедры

информатики

Леченко А.В.

Минск, 2019

Содержание

[ВВЕДЕНИЕ 5](#_Toc532825599)

[1. Программная среда реализации 8](#_Toc532825600)

[1.1. Android Studio 8](#_Toc532825601)

[1.2. Kotlin 9](#_Toc532825602)

[2. Анализ предметной области 11](#_Toc532825603)

[2.1. Анализ существующих аналогов 11](#_Toc532825604)

[2.1.1. Advanced Task Killer 11](#_Toc532825605)

[2.1.2. Advanced Task Manager 12](#_Toc532825606)

[2.1.3. ES диспетчер задач 12](#_Toc532825607)

[2.1.4. Super Task Killer 13](#_Toc532825608)

[2.1.5. Ccleaner 14](#_Toc532825609)

[2.2. Возникшие проблемы 16](#_Toc532825610)

[2.3. Постановка задачи 19](#_Toc532825611)

[2.4. Технические требования 19](#_Toc532825612)

[3. Программная реализация 20](#_Toc532825613)

[3.1. Main Activity 20](#_Toc532825614)

[3.2. IgnorListActivity 25](#_Toc532825615)

[3.3. SettingsActivity 26](#_Toc532825616)

[4. Описание применения 28](#_Toc532825617)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc532825618)

[И ЛИТЕРАТУРЫ 29](#_Toc532825619)

[ПРИЛОЖЕНИЕ 1. Текст программы 30](#_Toc532825620)

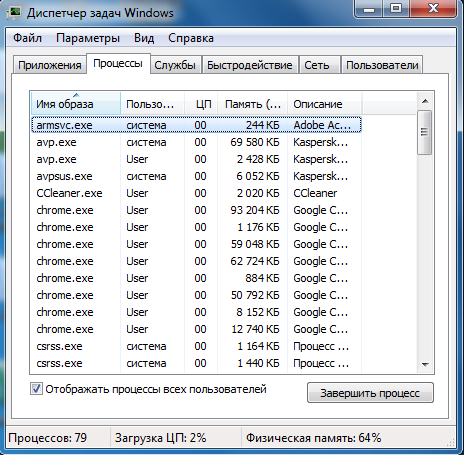
# ВВЕДЕНИЕ

Следует ли вообще использовать task-manager на Андроид устройстве? Какой выбрать? Чем грозит отключение и сброс системных приложений?

Когда платформа Андроид еще не была настолько популярной, многие процессы системы были далеко не оптимизированы. Если быть честными, платформа андроид начала свое восхождение со сплошных неудач. Особенно эти фоновые неотключаемые процессы, поедающие всю оперативную память устройства и не дающие нормально работать, а иногда и отвечать на звонки. В наше время, конечно, эта проблема не настолько существенна для мощных устройств, так как производители телефонов устанавливают в свои девайсы по 4-8 Гб оперативной памяти. Но для бюджетных телефонов этот вопрос все еще актуален.

Разберем несколько тонкостей и нюансов таск-менеджеров.

По принципам работы «менеджеры задач» похожи на диспетчер задач платформы Windows (см. рисунок 1). На персональном компьютере мы заходим во вкладку процессы, видим полный их перечень, включая системные задачи, затем кликаем правой кнопкой мыши и завершаем работу ненужной, поедающей память программы.

  
Рисунок 1: Диспетчер задач Windows

Но Windows не даст вам выключить системные утилиты и приложения, в отличие от таск-менеджеров, которые смогут "вырубить" процессы операционной системы. Удаление процессов должно приводить к экономии заряда и увеличению производительности смартфона. Но, к сожалению, это не всегда так.

* Эти суждения подводят нас к выводу о том, что вообще такое   
  Task-manager. Это приложения, позволяющие закрывать и управлять программами, работающими на фоне, с надеждой на улучшение производительности андроид-устройства и соответственно продлевающие жизнь заряда аккумулятора.
* Так все-таки устанавливать или нет? Посоветуем устанавливать «менеджеры» только в следующих случаях:
* Если вам хочется больше контроля над процессами вашего телефона
* Если ваш андроид тормозит и размер оперативной памяти Вашего гаджета не превышает 1 Гб
* Если зарядка аккумулятора садится крайне быстро, даже при условии выключенного экрана и полном отсутствии работы смартфона (спящий режим).
* Если вы установили некачественный софт и после установки начались постоянные зависания, но это приложение необходимо для выполнения определенных задач.
* Если вы хотите узнать, какое приложение кушает больше всего оперативной памяти.

Даже при этих условиях стоит быть аккуратным, при прерывании работы определенных процессов, могут не сработать другие, абсолютно независимые на первый взгляд утилиты.

Например, мы столкнулись со следующим: поставили функцию autokill (автокил) для всех много потребляющих память процессов Google. В итоге, стандартный будильник просто перестал работать. При отключении функции автокила, будильник пришел в норму. Оказалось, что «менеджер» выключает не только сам процесс, но и некоторые мелкие зависящие процессы устройства, которые могут затронуть другие программы.

Так что следует подойти разумно к выбору таск-менеджера для своего телефона. Нужно помнить правило Супермена: «С большой силой приходит большая ответственность».

# Программная среда реализации

## 1.1. Android Studio

Android Studio **—**это интегрированная среда разработки (IDE) для работы с платформой Android, анонсированная 16 мая 2013 года на конференции Google I/O.

IDE находилась в свободном доступе начиная с версии 0.1, опубликованной в мае 2013, а затем перешла в стадию бета-тестирования, начиная с версии 0.8, которая была выпущена в июне 2014 года. Первая стабильная версия 1.0 была выпущена в декабре 2014 года, тогда же прекратилась поддержка плагина Android Development Tools (ADT) для Eclipse.

Android Studio, основанная на программном обеспечении IntelliJ IDEA от компании JetBrains, - официальное средство разработки Android приложений. Данная среда разработки доступна для Windows, OS X и Linux. 17 мая 2017, на ежегодной конференции Google I/O, Google анонсировал поддержку языка Kotlin, используемого в Android Studio, как официального языка программирования для платформы Android в добавление к Java и С++.

Новые функции появляются с каждой новой версией Android Studio. На данный момент доступны следующие функции:

* Расширенный редактор макетов: WYSIWYG, способность работать с UI компонентами при помощи Drag-and-Drop, функция предпросмотра макета на нескольких конфигурациях экрана.
* Сборка приложений, основанная на Gradle.
* Различные виды сборок и генерация нескольких .apk файлов
* Рефакторинг кода
* Статический анализатор кода (Lint), позволяющий находить проблемы производительности, несовместимости версий и другое.
* Встроенный ProGuard и утилита для подписывания приложений.
* Шаблоны основных макетов и компонентов Android.
* Поддержка разработки приложений для Android Wear и Android TV.
* Встроенная поддержка Google Cloud Platform, которая включает в себя интеграцию с сервисами Google Cloud Messaging и App Engine.
* Android Studio 2.1 поддерживает Android N Preview SDK, а это значит, что разработчики смогут начать работу по созданию приложения для новой программной платформы.
* Новая версия Android Studio 2.1 способна работать с обновленным компилятором Jack, а также получила улучшенную поддержку Java 8 и усовершенствованную функцию Instant Run [7].
* Platform-tools 23.1.0 для Linux без объявления стала исключительно 64-разрядной, даже при попытке установить 32-разрядную версию. Иными словами Android Studio больше НЕ работает (выдаёт неустранимые ошибки) в   
  32-разрядных версиях Linux.  
  (https://code.google.com/p/android/issues/detail?id=196866)
* В Android Studio 3.0 будут по стандарту включены инструменты языка Kotlin основанные на JetBrains IDE.

Эта среда разработки была выбрана мной, т. к. является основной средой разработки под Android и имеет широкий спектр функциональности. [1]

### 1.2. Kotlin

Kotlin (Ко́тлин) — это статически типизированный язык программирования, работающий поверх JVM и разрабатываемый компанией JetBrains. Компилируется в JavaScript, а также на другие платформы через инфраструктуру LLVM. Язык назван в честь острова Котлин в Финском заливе, на котором расположен город Кронштадт.

Авторы ставили целью создать язык более лаконичный и типобезопасный, чем Java, и более простой, чем Scala. Следствием упрощения по сравнению со Scala стали также более быстрая компиляция и лучшая поддержка языка в IDE.

Язык разрабатывается с 2010 года, представлен общественности в июле 2011. Исходный код реализации языка открыт в феврале 2012. В феврале выпущен milestone 1, включающий плагин для IDEA. В июне — milestone 2 с поддержкой Android. В декабре 2012 года вышел milestone 4, включающий, в частности, поддержку Java 7.

В феврале 2016 года вышел официальный релиз-кандидат версии 1.0, а 15 февраля 2016 года — релиз 1.0. 1 марта 2017 вышел релиз 1.1.

В мае 2017 года компания Google сообщила, что инструменты языка Kotlin, основанные на JetBrains IDE, будут по стандарту включены в Android Studio 3.0 — официальный инструмент разработки для ОС Android. Что стало одной из причин, почему мной был выбран именно этот язык программирования.

Позиционируется разработчиками как объектно-ориентированный язык промышленного уровня, а также как язык, который сможет заменить Java. При этом язык полностью совместим с Java, что позволяет разработчикам постепенно перейти с Java на Kotlin. В частности, в Android язык встраивается с помощью Gradle, что позволяет для существующего Android-приложения внедрять новые функции на Kotlin без переписывания приложения целиком.

Синтаксис языка похож на Pascal, TypeScript, Haxe, PL/SQL, F#, Go и Scala, C++, Java, C#, Rust и D. При объявлении переменных и параметров типы данных указываются после названия (разделитель — двоеточие). Точка с запятой как разделитель операторов так же необязательна, как в Scala и Groovy; в большинстве случаев перевода строки достаточно, чтобы компилятор понял, что выражение закончилось. Кроме объектно-ориентированного подхода, Kotlin также поддерживает процедурный стиль с использованием функций. Как и в языках C/C++/D, точка входа в программу — функция "main", принимающая массив параметров командной строки. Программы на Kotlin также поддерживают Perl- и Unix/Linux shell-стиль интерполяции строк (переменные, включенные в строку заменяются на их содержимое). Kotlin также поддерживает вывод типов. [2]

По моему мнению Котлин сейчас является одним из самых перспиктивных языков программирования, сочетающим в себе преимущества статической типизации, мощность функционального программирования, приятный синтаксис, наличие большого количества «синтаксического сахара», что позволяет разработчику меньше думать о том как написать код, чем о том, что он хочет написать.

# Анализ предметной области

## 2.1. Анализ существующих аналогов

### 2.1.1. Advanced Task Killer

10 миллионов скачиваний говорят сами за себя. Для таких программ это очень много. Интерфейс удобный, но слишком темный, как-то неприятно. Задачи свои выполняет отлично. Много расширений и удобств для пользователя, однако, установить автоматическое «убийство» для определенных программ нельзя. Зато есть возможность уничтожать процессы в crazy режиме (каждые 5 минут). Не советуем выбирать эту функцию, поскольку этот менаджер задач имеет приоритет и будет закрывать иногда даже работающие программы. Кстати, приложение на английском.

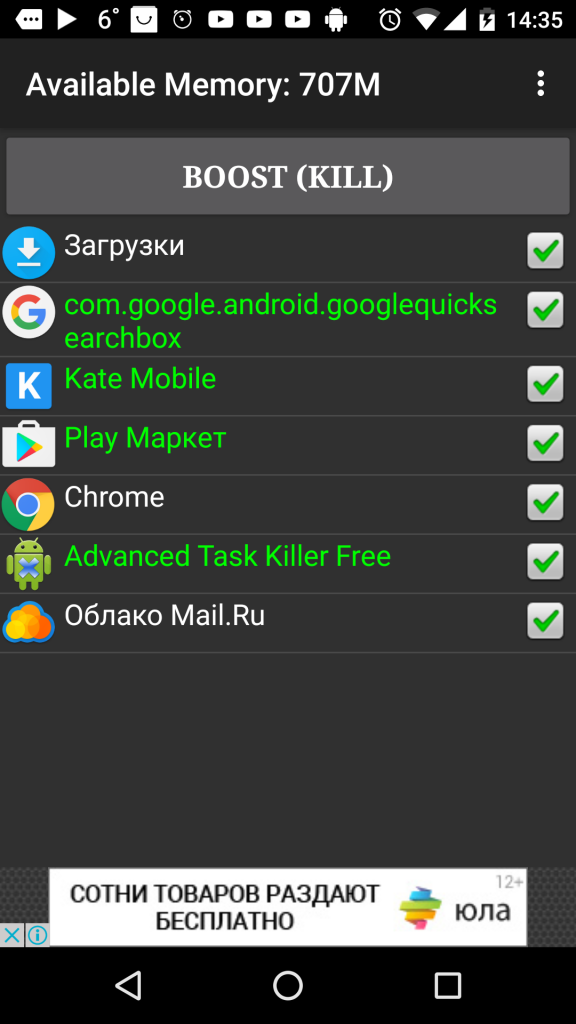


Рисунок 2: Advanced Task Killer

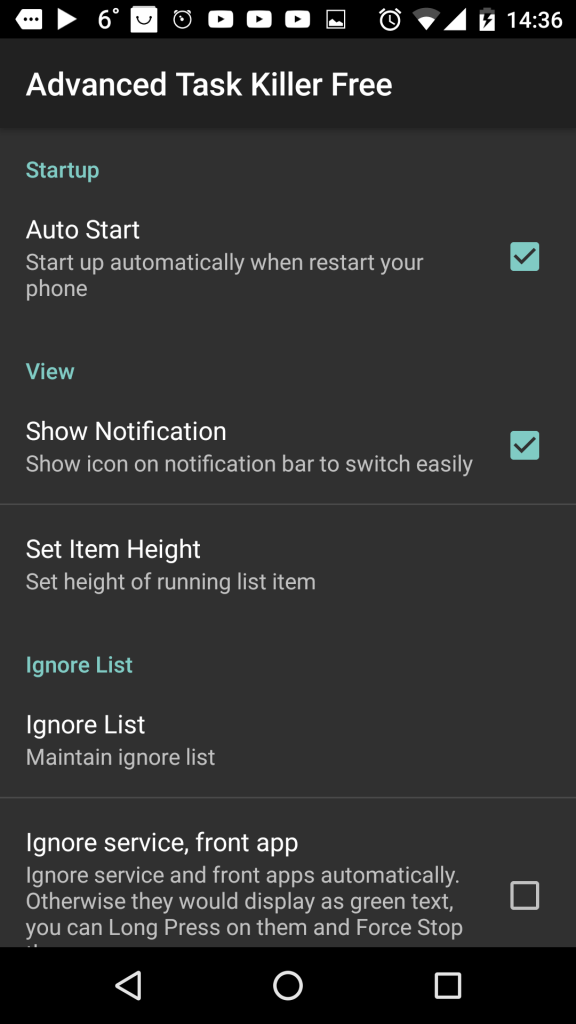
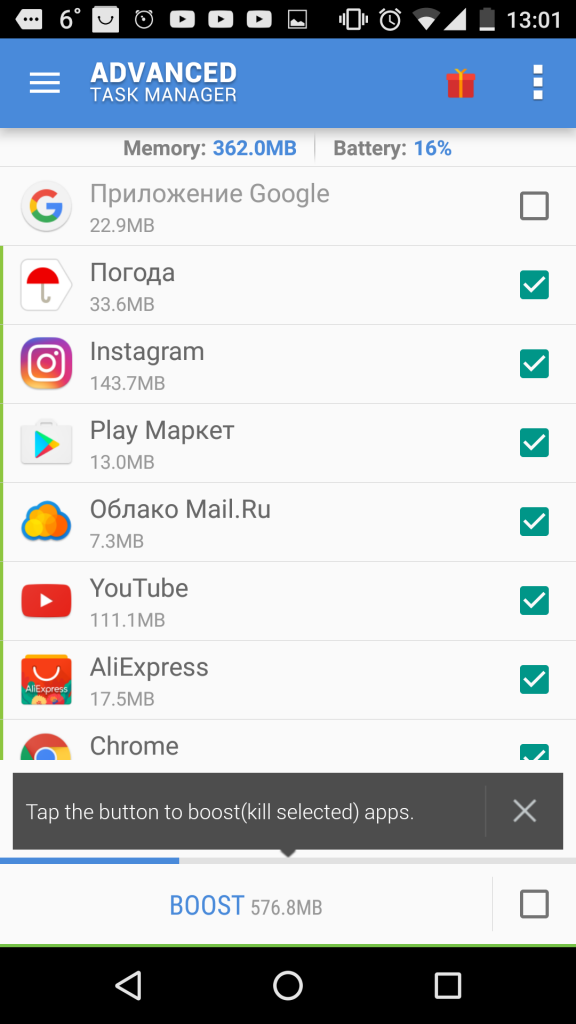
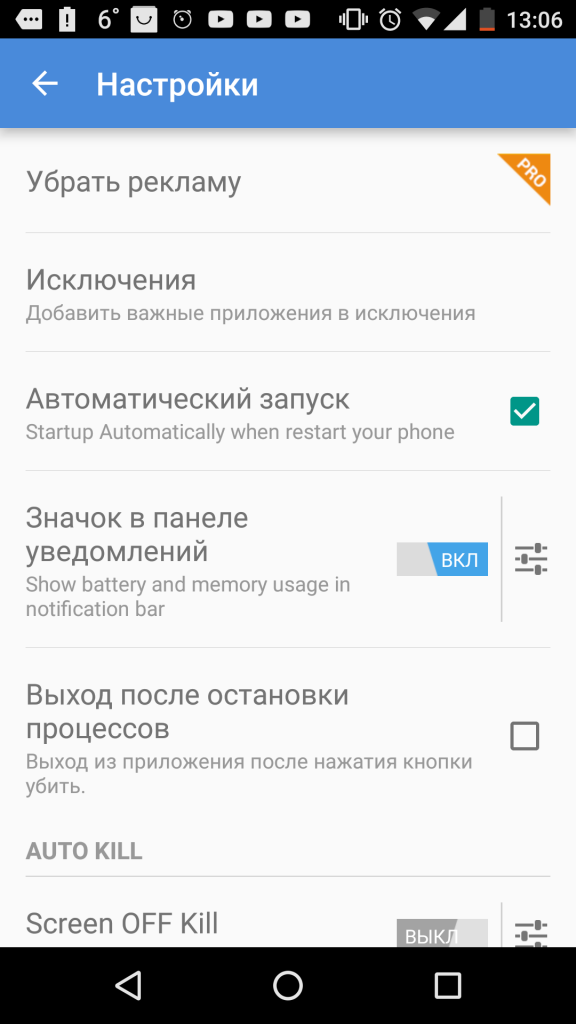


Рисунок 3: Advanced Task Killer

### 2.1.2. Advanced Task Manager

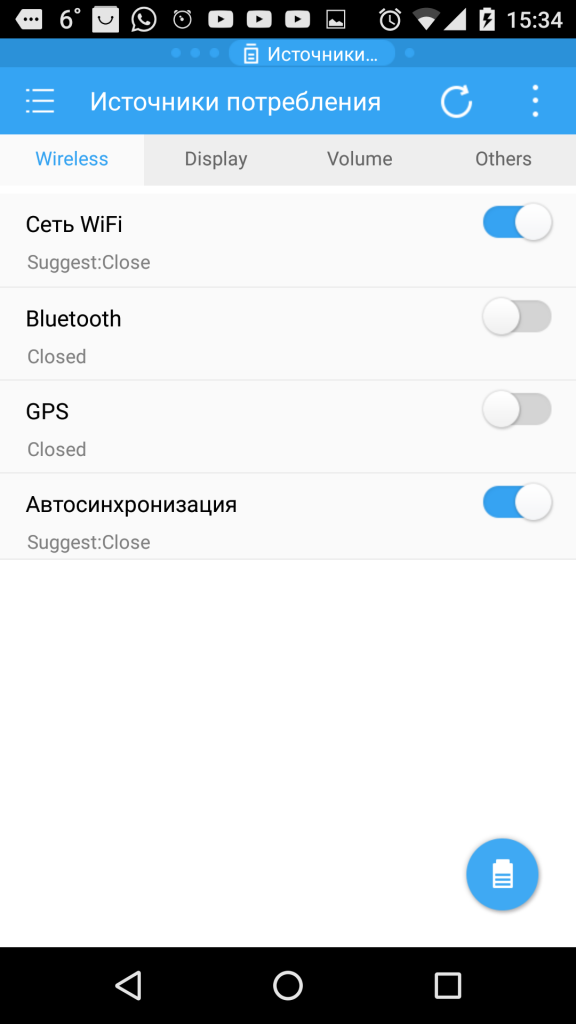
Один из первых появившихся менаджеров задач на просторах Play Market. Опознает абсолютно все работающие процессы, как фоновые, так и необходимые для работы в настоящий момент. Закрыть может любые из перечисленных, в том числе и системные. Обладает функцией «автокил», присущей практически всем task killer’ам. Исключение важных процессов и добавление в общий список исключений так же доступно. Несколько режимов: закрытие всех утилит после выключения экрана, регулярные «убийства», «автокил» при включении телефона.

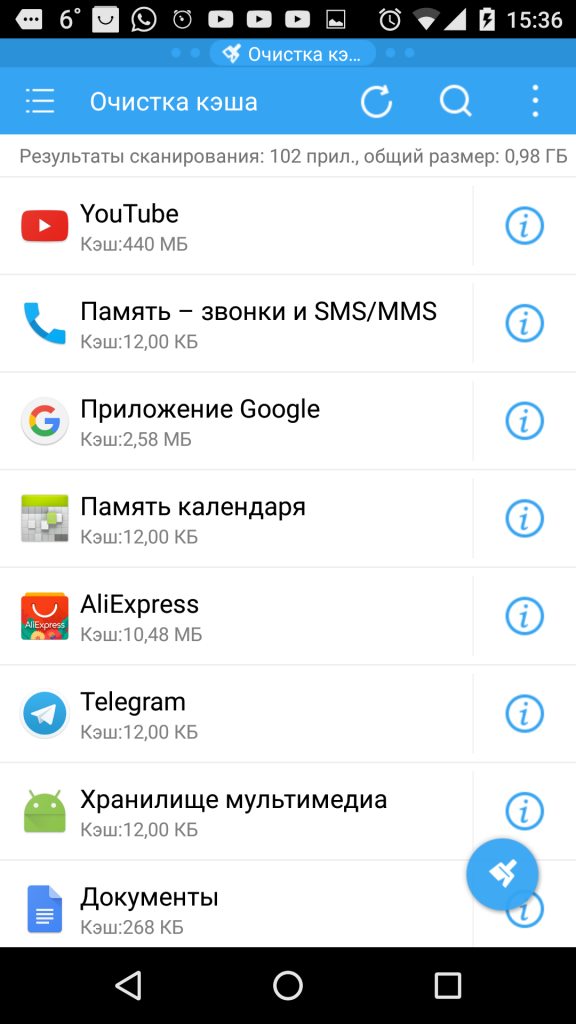
  
Рисунок 4: Advanced Task Manager

  
Рисунок 5: Advanced Task Manager

### 2.1.3. ES диспетчер задач

Отличный менеджер задач, с множеством настроек. Представляет собой не только программу для закрытия утилит, но и позволяет редактировать их запуск, автозагрузку приложений, мониторинг расхода батареи, настраивать звуковые оповещения и другие. В целом хорошо справляется с поставленными задачами: гибкий и емкий менеджер, функция «автокила» так же имеется, совместно со списком нужных исключений. Отслеживает самые требовательные программы и оповещает вас об этом. Есть платный контент, а для полнофункциональной реализации, предлагается установить «ES проводник».

  
Рисунок 6: ES диспетчер задач

  
Рисунок 7: ES диспетчер задач

### 2.1.4. Super Task Killer

Пожалуй единственный минус этой программы – наличие надоедливой рекламы. Очень приятен внешний вид программы, хорошо подобранные цвета и оформление. Обладает способностями автоматического ускорения (автокил) по заданному времени. Так же как и остальные, приложение обладает белым списком. Для экономии заряда, имеет несколько режимов: пользовательский   
(с собственными настройками), обычный и суперэкономичный. Последний,   
отключит Вам все возможности устройства: WIFI, мобильные сети, Bluetooth, режимы звонков и снизит яркость до минимально возможной.

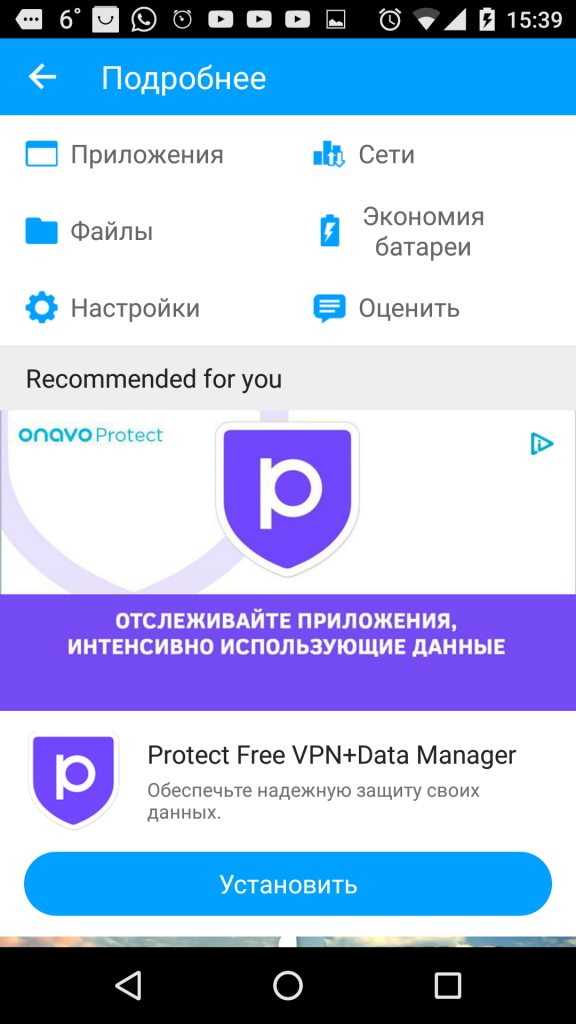


Рисунок 8: Super Task Killer

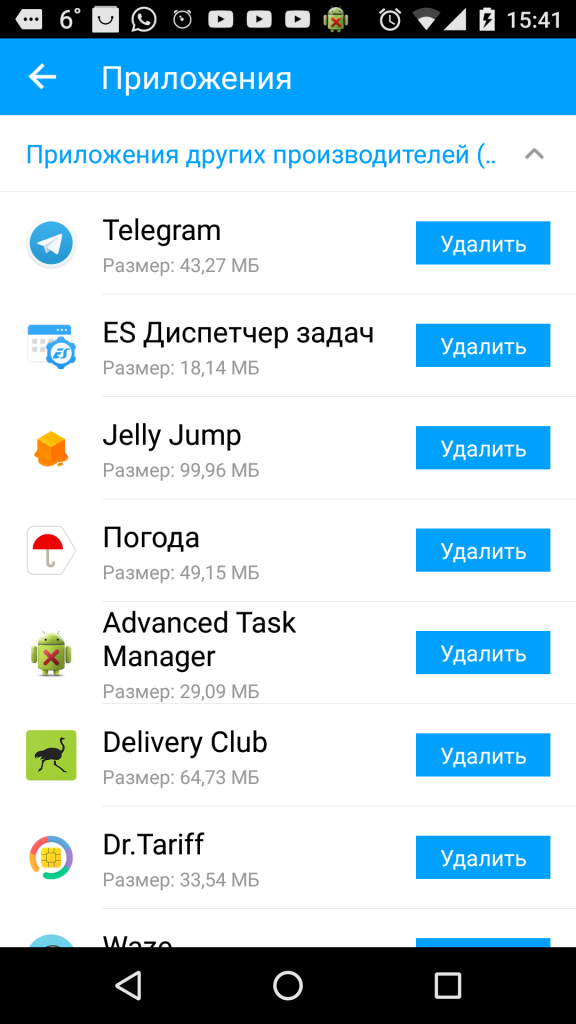
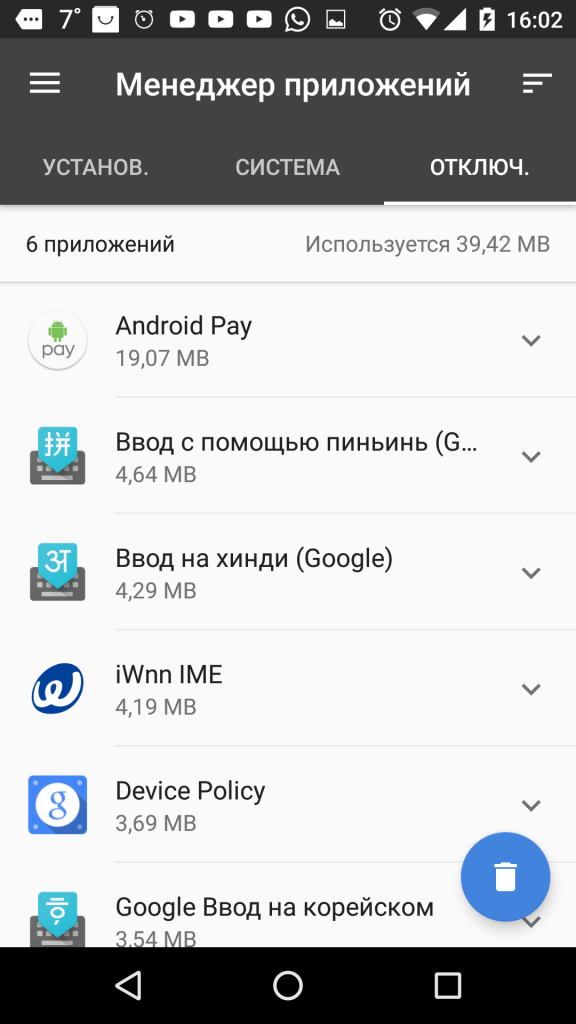
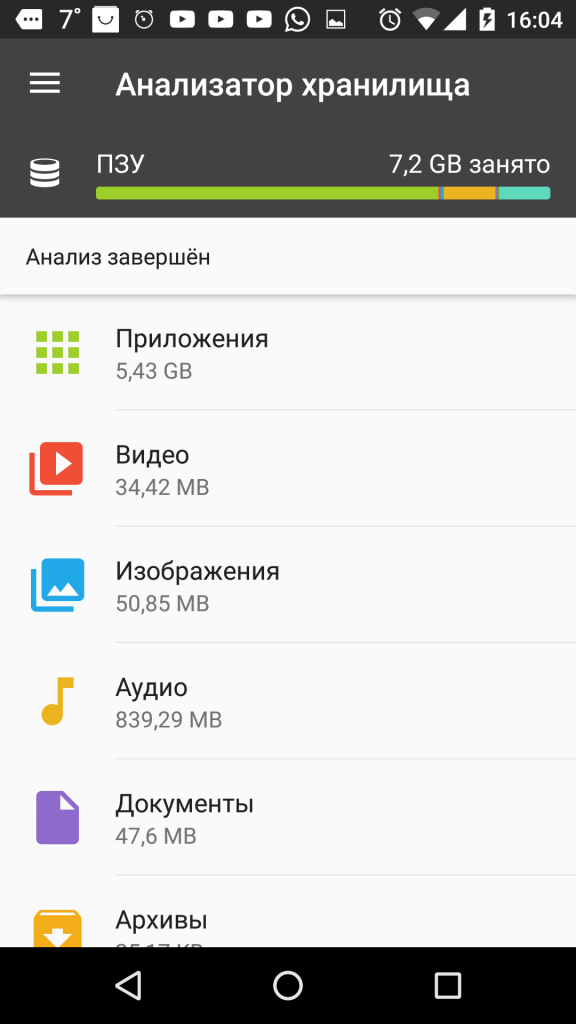


Рисунок 9: Super Task Killer

### 2.1.5. Ccleaner

Многие узнают это название. Аналог существует на персональных компьютерах, и, признаться, является почти что лучшей программой для очистки кэша и реестра. Самый простой и удобный, на наш взгляд. Отлично делает свою работу, качественно и без выключения системных фоновых программ, так как они вообще разделены на несколько подгрупп: установленные собственноручно, система и отключенные – великолепное решение. Внешний вид позволяет найти нужную кнопку в считанные секунды. Посмотреть автозагрузку и что работает на фоне — так же не составит особого труда. Проводит анализ как оперативного, так и постоянного запоминающего устройства. Планировщик очистки, к сожалению, доступен только пользователям РФ, но любые ограничения можно обойти. Это не исключение.

  
Рисунок 10: Ccleaner

  
Рисунок 11: Ccleaner

Все вышеуказанные диспетчеры задач имеют удобные и интерактивные виджеты, предоставляющие пользователям быстро очищать свой телефон от ненужного мусора.

## 2.2. Возникшие проблемы

После исследования аналогов и попыток установить их на свой телефон (модели Xiaomi 4X с версией Android 7.1.2) я обнаружил следующие проблемы. Многие (если не все) проверенные таск-менеджеры делают лишь видимость прекращения процессов, но не прекращают их в действительности, показывая пользователю лишь красивую анимацию «очищения оперативной памяти телефона и ускорения его». Что же действительно мешает разработчиком работать напрямую с процессами телефона из какого-либо приложения? Почему они не могут в действительности дать пользователю полный контроль над памятью и оптимизацией работы своего телефона?

Исследовав данный вопрос я наткнулся на замечательную статью [3] Джареда Руммлера — разработчика одной из библиотек, позволяющих управлять процессами телефона. Отрывки из этой статьи я сейчас и приведу:

«В 2015 году я работал над приложением для Android, которое должно было получить список запущенных имен процессов для отображения пользователю. После обновления до Android 5.0 (Lollipop) я заметил, что мое приложение больше не работает по назначению. Список процессов вернул только мой процесс, запрашивавший информацию. Я обнаружил, что Google „убил“ getRunningTasks(int) и getRunningAppProcesses() в Android 5.0. Оба эти методы являются устаревшими и процесса подачи заявки абонента.

Android 5.0 представил UsageStatsManager, который обеспечивает доступ к истории использования устройства и статистике. Этот API требует разрешения android.permission.PACKAGE\_USAGE\_STATS - это разрешение системного уровня, которое не предоставляется сторонним приложениям. Однако объявление разрешения подразумевает намерение использовать API, и пользователь устройства может предоставить разрешение через приложение "Параметры". Несколько пользователей и разработчиков сообщили, что некоторые изготовители оборудования удалили этот параметр из настроек телефона. Тем не менее этот параметр в любом случае был бы не слишком полезен и надёжен для пользователей и разработчиков

Я думаю, что каждая операционная система должна позволить пользователю легко обнаружить, какие приложения в настоящее время работают на устройстве, и предоставить API для этого. Я был полон решимости найти способ, чтобы получить список запущенных процессов на Android 5.0. Сначала я запустил команду ps в оболочке и проанализировал выходные данные. Это работало, но это не было эффективно. Увидев много коллег-разработчиков об этой проблеме на StackOverflow, я решил, что разработаю библиотеку с открытым исходным кодом, чтобы обойти изменения Google.

В октябре 2015 года я открыл исходный код библиотеки AndroidProcesses. К моему удивлению, библиотека начала набирать популярность на GitHub и была быстро скопирована некоторыми разработчиками в Китае, где она получила еще больше внимания. Недавно я декомпилировал многие из 150 лучших бесплатных приложений в Google Play и был удивлен тем, сколько приложений используют библиотеку. С помощью библиотеки включают Cheetan Mobile и Baidu. Приложения, которые я нашел, которые в настоящее время используют библиотеку, включают ES File Explorer, Clean Master, Security Master, CM Launcher 3D, Virus Cleaner by Hi Security и Super Cleaner. В сочетании, эти приложения имеют не менее 1,2 миллиарда загрузок!

Я лично не одобряю эти приложения вроде “диспетчер задач", которые ложно обещают повысить производительность вашего устройства.   
По-видимому, Google тоже не любит приложения этих крупных китайских фирм. Можно задуматься, если Google и эти крупные китайские фирмы оба   
собирают статистику использования приложений для улучшения таргетинга объявлений, то ограничения Google на получение запущенных процессов   
действительно направлено на эти фирмы, а не по соображениям безопасности.

В Android 7.0 (Nougat) Google значительно ограничил доступ к файловой системе, что делает невозможным получение списка запущенных процессов с помощью моей библиотеки. Ведущий разработчик CopperheadOS любезно изложил изменения, уменьшающие доступ к /proc:

* + - * Файловая система procfs теперь смонтирована с hidepid=2, что исключает доступ к каталогам /proc/PID других пользователей. Это изменение было реализовано в CopperheadOS и затем было принято вверх по иерархии на его основе. Существует группа для создания исключений, но она не предоставляется каким-либо разрешением. Она используется только для создания исключений для некоторых процессов в базовой системе. Разрешение могло бы быть получаться как "опасное", и это то, что и ожидалось, что Google в конечном итоге сделает, но они решили, что пользователи не поймут последствий этого.
      * Политика SELinux также стала намного строже. Для приложений больше нет базового доступа к /proc, хотя это относится только к файлам, отличным от каталогов /proc/PID. Там все еще есть доступ к нескольким файлам с ярлыками, не подпадающими под общую политику proc, но в основном она исчезла. Это происходит постепенно, и есть много соответствующих обязательств.
      * Это не только скрывает много важной информации, но и закрывает некоторые более вопиющие дыры в безопасности, связанные с сторонними каналами, запрещая такие вещи, как логирование сторонними приложениями вводимой с клавиатуры информации. [4]

В Android 7.0+ только предустановленные приложения Google будут иметь доступ к информации о процессах приложений, исключая эту информацию для сторонних приложений.

Однако, Тод Liebeck (основатель NextApp, Inc.в который заработал значок лучшего разработчика по версии Google) оперативно ответил, отметив, что такая атака все еще возможна в последней версии Android.

Изменения, которые были внесены в Android Nougat, которые калечат SystemPanel (и другие системные утилиты, безопасность и антивирусные приложения), мало что сделают, чтобы остановить этот вид атаки. Опять же, единственную реальную выгоду будут получать авторы вредоносных программ, которые лучше умеют скрываться.

Похоже, Google не планирует предоставлять доступ к proc сторонним приложениям и с тех пор закрыл проблему. Соответственно я планирую закрыть библиотеку AndroidProcesses. Надеюсь, Google сделает то, что CopperheadOS просит от них, и предоставит "опасное" разрешение, позволяющее пользователям предоставлять сторонним приложениям доступ к информации о запущенном процессе приложения».

Здесь мы приходим к выводу, что создать приложение, предоставляющее полный контроль над процессами телефона (т.е. диспетчера задач) для Android 7+ не предоставляется возможным, если вы не разработчик Google.

Но раз уж я взялся за эту тему, то постараюсь написать приложение для Android младше 7 версии, где ещё предоставляются все необходимые разрешения.

## 2.3. Постановка задачи

Разработать менаджер задач (процессов) для ОС Android. С учётом описанного в разделе «Возникшие проблемы» менаджер задач будет предназначаться для телефонов с версией Android 5 и меньше.

### 2.4. Технические требования

Разработанное приложение должно удовлетворять следующим требованиям:

* Показывать все текущие процессы
* Предоставлять возможность останавливать произвольный процесс
* Предоставлять возможность остановить все текущие процессы
* Возможность посмотреть информацию о выбранном процессе
* Иметь игнор-лист — список приложений, которые можно будет самому выбрать и которые после не будут показываться в общем списке процессов и которые пользователь не сможет случайно остановить (имеются ввиду какие-либо системные приложения или приложения, работа которых важна пользователю)
* Предоставлять возможность просто и удобно заносить и удалять приложения из игнор-листа
* Показывать используемую приложениями оперативную память (RAM), чтобы пользователь понимал, когда её не хватает и предпринимал действия, чтобы ускорить свой телефон (т. е. останавливал не нужные в данный момент процессы)
* Иметь пользовательские настройки, например возможность выбрать действие на клик по приложению в списке (посмотреть информацию о приложении, остановить, занести в игнор-лист и т. д.)

# 3. Программная реализация

Далее описываются основные компоненты приложения, заостряется внимание на важных деталях реализации управления процессами с примерами кода.

## 3.1. Main Activity

Main activity — главная страница приложения, здесь пользователю показывается оперативная память, потребляемая процессами и список запущенных процессов.

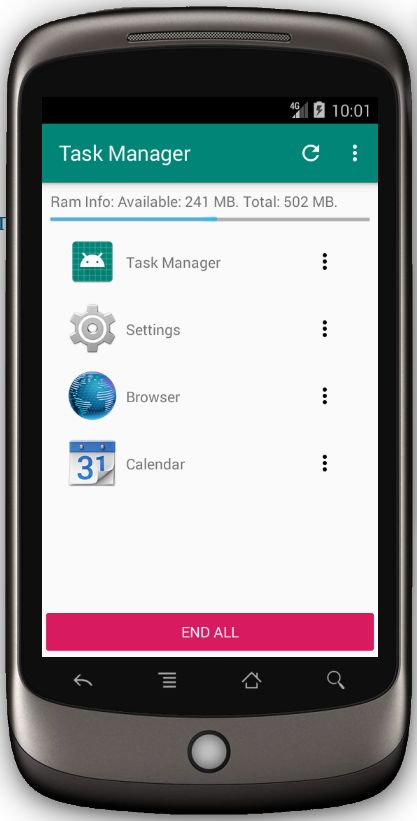


Рисунок 12: Главный экран приложения

Используя Android API [5] мы узнаем сколько оперативной памяти доступно пользователю и сколько свободно для использования.

Для этого мы обращаемся к системному файлу meminfo в папке proc (processes). Парсим данные в нём и получаем общий объем оперативной памяти:

val reader = RandomAccessFile("/proc/meminfo", "r")

val load = reader.readLine()

val memInfo = load.split(" ")

totalMemory = memInfo[9].toFloat() / 1024

Чтобы определить сколько памяти доступно пользователю, мы обращаемся к системному сервису:

val activityManager = (Activity.ACTIVITY\_SERVICE) as ActivityManager

val memoryInfo = ActivityManager.MemoryInfo()

activityManager.getMemoryInfo(memoryInfo)

val availableMemory = memoryInfo.availMem / 1048576L

Чтобы узнать какие процессы запущены мы также используем этот сервис. Для каждого из запущенным приложений мы извлекаем из сервиса информацию о процессе, имя процесса и иконку:

val activityManager = this.getSystemService(Context.ACTIVITY\_SERVICE) as ActivityManager

val listOfProcesses = activityManager.runningAppProcesses

val appsList = ArrayList<App>()

for (process in listOfProcesses) {

if (packageManager.getLaunchIntentForPackage(process.processName) != null && !checkIfIgnore(process.processName)) {

try {

val info = packageManager.getApplicationInfo(process.processName, PackageManager.GET\_META\_DATA)

val appIcon = packageManager.getApplicationIcon(info)

val appName = packageManager.getApplicationLabel(info)

val app = App(appName.toString(), process.processName, appIcon)

appsList.add(app)

} catch (e: PackageManager.NameNotFoundException) {

e.printStackTrace()

}

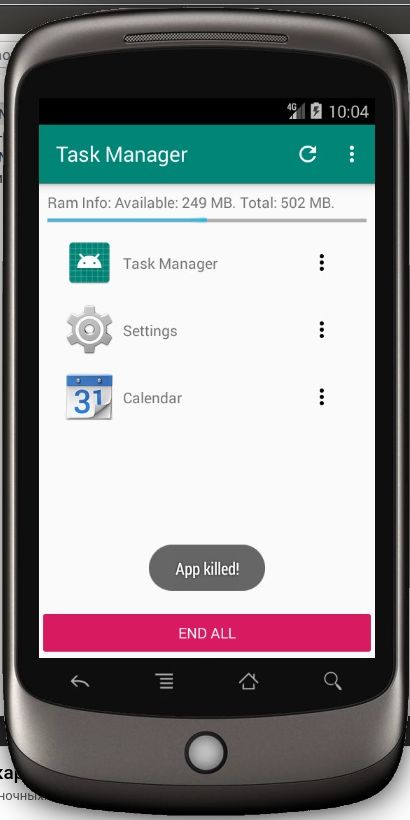
}

}

Остановка выбранного процесса происходит следующими методами:

val activityManager = this.getSystemService(Context.ACTIVITY\_SERVICE) as ActivityManager

activityManager.restartPackage(pkgName)

  
Рисунок 13: Остановка процесса

Переключиться на выбранный процесс (открыть приложение, которое его запустило) нам позволяет packageManager:

val intent = packageManager.getLaunchIntentForPackage(pkgName)

applicationContext.startActivity(intent)

Android API также позволяет открыть системную информацию о выбранном приложении, где показывается занимаемая им память, где можно удалить его или очистить данные о приложении:

val intent = Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)

intent.addCategory(Intent.CATEGORY\_DEFAULT)

intent.data = Uri.parse("package:$pkgName")

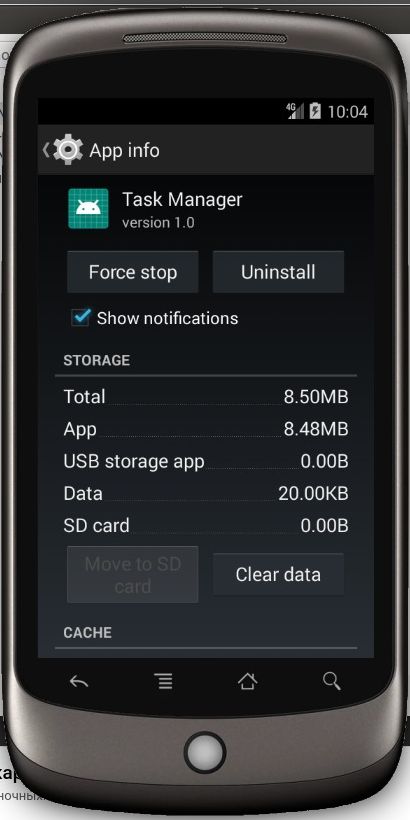
startActivity(intent)

Рисунок 14: Информация о процессе

Добавление в игнор-лист происходит с использованием SharedPreferences. Это специальный класс, который позволяет сохранять во внутреннее хранилище приложения пары ключ-значение, при этом это файл с данными доступен только этому приложению:

ignoreArray = getSharedPreferences(IGNORE\_ARRAY, 0)

ignoreArray.edit(). apply {

putString(pkgName, pkgName)

apply()

}

ignoreList.add(pkgName)

Для выбора из этих опций для каждого процесса в списке доступно всплывающее меню, где пользователь может решить, что собирается делать с выбранным процессом:

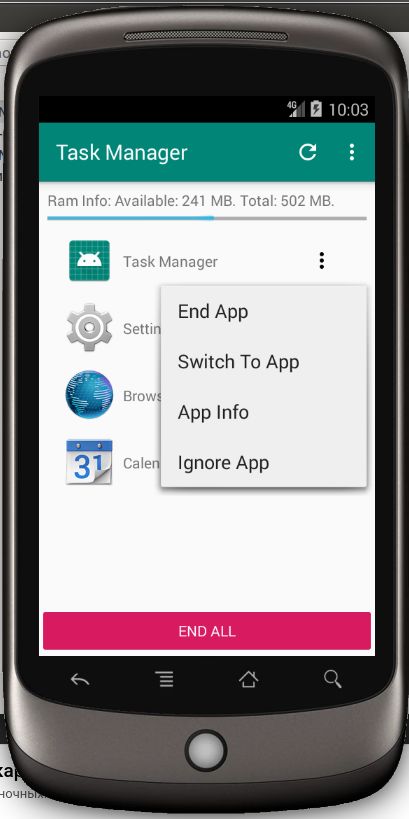
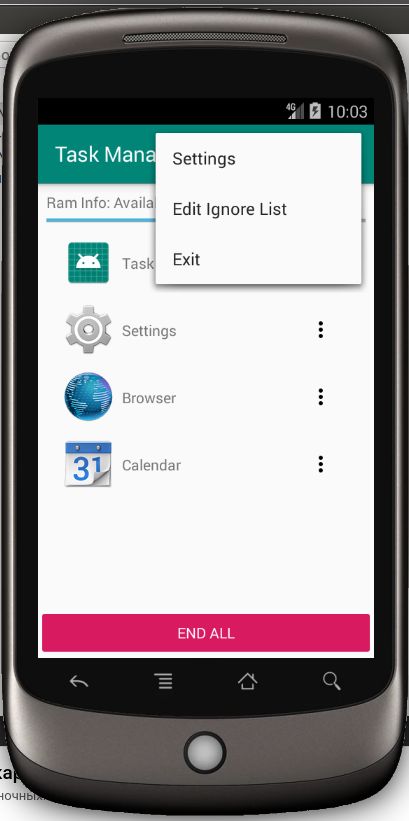


Рисунок 15: Всплывающее меню при нажатии на процесс

## 3.2. IgnorListActivity

Открыть игнор-лист можно или при помощи контекстного меню на главном экране приложения, или при помощи ссылки на экране настроек:

  
Рисунок 16: Контекстное меню на главном экране приложения

Здесь, также используя SharedPreferences мы можем удалить процесс из игнор-листа:

ignoreArray.edit().apply{

remove(item.packageName)

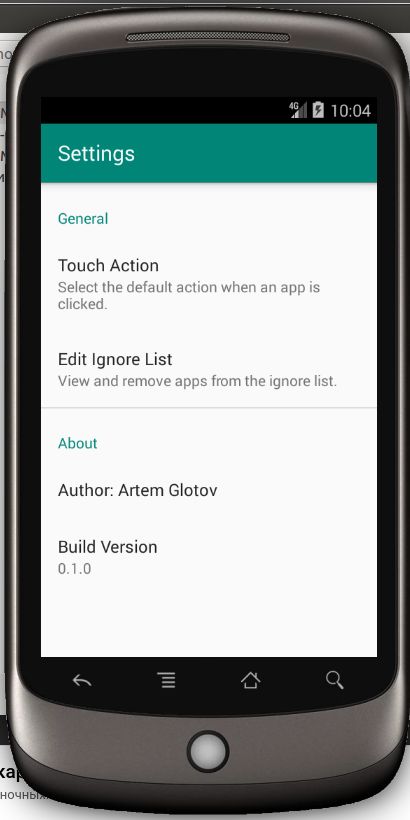
apply()

}

refreshIgnoreList()

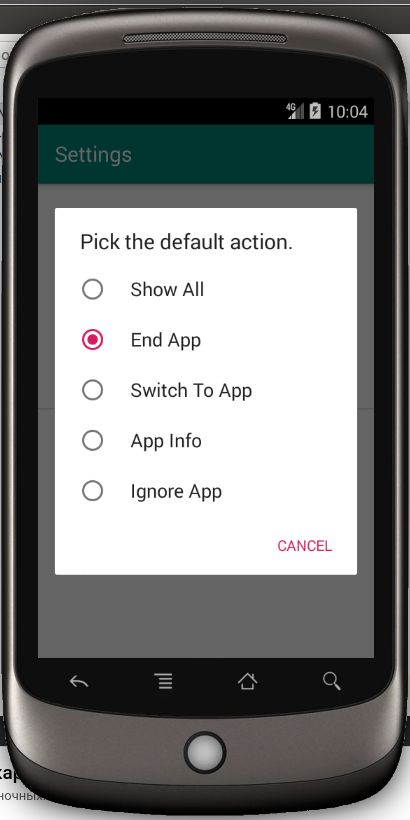
## 3.3. SettingsActivity

Экран настроек также открывается через контекстное меню на главном экране приложения.

  
Рисунок 17: Экран настроек

Он предоставляет возможность открыть игнор-лист, посмотреть версию приложения, информацию о разработчике и выбрать действие, которое будет происходить при нажатии на процесс на главном экране приложения:

Используя реализованные разработчиком возможности, пользователь может интуитивно пользоваться приложением, управлять запущенными процессами и следить за потребляемой оперативной памятью.

  
Рисунок 18: Выбор функции для нажатия на процесс

# 4. Описание применения

Используя главный экран приложения и всплывающие меню, пользователь может управлять процессами на своём телефоне, заносить их в игнор-лист, смотреть подробную информацию о них и подключаться к ним.

Также в приложении имеются настройки, где он может задат действие на клик по процессу в списке, посмотреть информацию о разработчике и редактировать игнор-лист.

Это приложение позволит пользователю иметь детальный контроль над процессами своего телефона и следить за объёмом используемой оперативной памяти.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

# И ЛИТЕРАТУРЫ

1. Википедия. Android Studio [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/Android_Studio>

2. Википедия. Koltin [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/wiki/Kotlin>

3. Блог Джареда Руммлера [Электронный ресурс]. – Электронные данные –Режим доступа: <https://jaredrummler.com/2017/09/13/android-processes/>

4. Комментарии разработчиков ОС Android [Электронный ресурс]. – Электронные данные. – Режим доступа:

[*https://android-review.googlesource.com/#/c/181345/*](https://android-review.googlesource.com/#/c/181345/)

[*https://android-review.googlesource.com/#/c/105337/*](https://android-review.googlesource.com/#/c/105337/)

5. Официальный сайт с документацией по Android [Электронный ресурс] – Электронные данные. – Режим доступа: <https://developer.android.com/>

# ПРИЛОЖЕНИЕ 1. Текст программы

**class** MainActivity : AppCompatActivity(), AppListFragment.AppListener {  
 **override fun** switchToApp(packageName: String) {  
 **val** intent = packageManager.getLaunchIntentForPackage(packageName)  
 applicationContext.startActivity(intent)  
 }  
  
 **override fun** endApp(packageName: String) {  
 **if** (packageName == **"com.artemglotov.taskmanager"**) {  
 **this**.finish()  
 }  
 killApp(packageName)  
 refreshAppList()  
 Toast.makeText(applicationContext, **"App killed!"**, Toast.LENGTH\_SHORT).show()  
 }  
  
 **override fun** appInfo(packageName: String) {  
 getAppInfo(packageName)  
 }  
  
 **override fun** ignoreApp(packageName: String) {  
 addToIgnore(packageName)  
 refreshAppList()  
 }  
  
 **private lateinit var** appListFragment: AppListFragment  
 **private var** ignoreList = ArrayList<String>()  
  
 **lateinit var** ignoreArray: SharedPreferences  
 **lateinit var** preferences: SharedPreferences  
 **private var** totalMemory: Float = 0.0f  
  
 **private lateinit var** touchAction: String  
  
  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_main)  
  
 ignoreArray = getSharedPreferences(IGNORE\_ARRAY, 0)  
 ignoreList.addAll(ignoreArray.all.keys)  
  
 preferences = PreferenceManager.getDefaultSharedPreferences(**this**)  
 touchAction = preferences.getString(**"touch\_action"**, **"Show All"**) ?: **"Show All"** kill\_all\_button.setOnClickListener **{** endAllTask(getAppsList())  
 **}** appListFragment = AppListFragment()  
 appListFragment.arguments = Bundle().apply **{** putBoolean(**"IS\_SHOW\_MORE"**, **true**)  
 **}** supportFragmentManager  
 .beginTransaction()  
 .replace(R.id.app\_list\_container, appListFragment)  
 .commit()  
  
 refreshAppList()  
 refreshMemoryInfo()  
 }  
  
  
 **override fun** onStart() {  
 **super**.onStart()  
 refreshAppList()  
 }  
  
 **override fun** onResume() {  
 touchAction = preferences.getString(**"touch\_action"**, **"Show All"**) ?: **"Show All"  
  
 super**.onResume()  
 }  
  
 **override fun** onCreateOptionsMenu(menu: Menu): Boolean {  
 menuInflater.inflate(R.menu.options\_menu, menu)  
 **return true** }  
  
  
 **override fun** onOptionsItemSelected(item: MenuItem): Boolean {  
  
 **when** (item.itemId) {  
 R.id.menu\_refresh -> {  
 refreshAppList()  
 **return true** }  
 R.id.settings -> {  
 **val** t = Intent(**this**, SettingsActivity::**class**.java)  
 startActivity(t)  
 **return true** }  
 R.id.edit\_ignore -> {  
 **val** intent = Intent(**this**, IgnoreListActivity::**class**.java)  
 startActivity(intent)  
 **return true** }  
 R.id.exit -> {  
 **this**.finish()  
 **return true** }  
 **else** -> **return super**.onOptionsItemSelected(item)  
 }  
 }  
  
 **private fun** getAppsList(): ArrayList<App> {  
 **val** activityManager = **this**.getSystemService(Context.ACTIVITY\_SERVICE) **as** ActivityManager  
 **val** listOfProcesses = activityManager.runningAppProcesses  
 **val** appsList = ArrayList<App>()  
 **for** (process **in** listOfProcesses) {  
 **if** (packageManager.getLaunchIntentForPackage(process.processName) != **null** && !checkIfIgnore(process.processName)) {  
 **try** {  
 **val** info = packageManager.getApplicationInfo(process.processName, PackageManager.GET\_META\_DATA)  
 **val** appIcon = packageManager.getApplicationIcon(info)  
 **val** appName = packageManager.getApplicationLabel(info)  
 **val** app = App(appName.toString(), process.processName, appIcon)  
 appsList.add(app)  
 } **catch** (e: PackageManager.NameNotFoundException) {  
 e.printStackTrace()  
 }  
 }  
 }  
 **return** appsList  
 }  
  
 **private fun** refreshAppList() {  
 **val** appList = getAppsList()  
 appListFragment.refreshAppList(appList)  
 checkNoAppsRunning(appList)  
 refreshMemoryInfo()  
 }  
  
 **private fun** checkNoAppsRunning(appsList: ArrayList<App>) {  
 **if** (appsList.count() > 0) {  
 app\_list\_container.visibility = View.VISIBLE  
 kill\_all\_button.isEnabled = **true** no\_apps\_text.visibility = View.GONE  
 } **else** {  
 no\_apps\_text.visibility = View.VISIBLE  
 app\_list\_container.visibility = View.INVISIBLE  
 kill\_all\_button.isEnabled = **false** }  
 }  
  
  
 **override fun** onAppClick(item: App) {  
 **val** packageName = item.packageName  
 **when** (touchAction) {  
 **"End App"** -> {  
 **if** (packageName == **"com.artemglotov.taskmanager"**) {  
 **this**.finish()  
 }  
 killApp(packageName)  
 refreshAppList()  
 Toast.makeText(applicationContext, **"App killed!"**, Toast.LENGTH\_SHORT).show()  
 }  
 **"Switch To App"** -> {  
 **val** intent = packageManager.getLaunchIntentForPackage(packageName)  
 applicationContext.startActivity(intent)  
 }  
 **"App Info"** -> getAppInfo(packageName)  
 **"Ignore App"** -> addToIgnore(packageName)  
 *// TODO check it* **else** -> {}*//parent.showContextMenuForChild(view)* }  
 refreshAppList()  
 }  
  
 **private fun** getAppInfo(pkgName: String) {  
 **val** intent = Intent(Settings.ACTION\_APPLICATION\_DETAILS\_SETTINGS)  
 intent.addCategory(Intent.CATEGORY\_DEFAULT)  
 intent.data = Uri.parse(**"package:$**pkgName**"**)  
 startActivity(intent)  
 }  
  
 **private fun** addToIgnore(apps: List<App>) {  
 **for** (app **in** apps) {  
 **val** pkgName = app.packageName  
 addToIgnore(pkgName)  
 }  
 refreshAppList()  
 Toast.makeText(applicationContext, **"${**apps.count()**} apps added to ignore list."**, Toast.LENGTH\_SHORT).show()  
 }  
  
 **private fun** addToIgnore(pkgName: String) {  
 ignoreArray.edit(). apply **{** putString(pkgName, pkgName)  
 apply()  
 **}** ignoreList.add(pkgName)  
 }  
  
 **private fun** killApp(pkgName: String) {  
 **val** activityManager = **this**.getSystemService(Context.ACTIVITY\_SERVICE) **as** ActivityManager  
 activityManager.restartPackage(pkgName)  
 }  
  
 **override fun** onContextItemSelected(item: MenuItem): Boolean {  
 **val** info = item.menuInfo **as** AdapterView.AdapterContextMenuInfo  
 **val** pkgName = info.targetView.tag **as** String  
  
 **when** (item.itemId) {  
 R.id.ccmenu\_endApp -> {  
 **if** (pkgName == **"com.artemglotov.taskmanager"**) {  
 **this**.finish()  
 }  
 killApp(pkgName)  
 refreshAppList()  
 Toast.makeText(applicationContext, **"App killed!"**, Toast.LENGTH\_SHORT).show()  
 **return true** }  
 R.id.ccmenu\_switchToApp -> {  
 **val** intent = packageManager.getLaunchIntentForPackage(pkgName)  
 applicationContext.startActivity(intent)  
 **return true** }  
 R.id.ccmenu\_appInfo -> {  
 getAppInfo(pkgName)  
 addToIgnore(pkgName)  
 refreshAppList()  
 **return super**.onContextItemSelected(item)  
 }  
 R.id.ccmenu\_ignoreApp -> {  
 addToIgnore(pkgName)  
 refreshAppList()  
 **return super**.onContextItemSelected(item)  
 }  
 **else** -> **return super**.onContextItemSelected(item)  
 }  
 }  
  
 **private fun** endAllTask(**vararg** names: List<App>) {  
 GlobalScope.launch(Dispatchers.IO) **{  
 val** apps = names[names.size - 1]  
 **for** (app **in** apps) {  
 killApp(app.packageName)  
 }  
 withContext(Dispatchers.Main) **{** Toast.makeText(applicationContext, **"All apps killed!"**, Toast.LENGTH\_SHORT).show()  
 refreshAppList()  
 refreshMemoryInfo()  
 **}  
 }** }  
  
 **private fun** refreshMemoryInfo() {  
 **if** (totalMemory.equals(0.0f)) {  
 **try** {  
 **val** reader = RandomAccessFile(**"/proc/meminfo"**, **"r"**)  
 **val** load = reader.readLine()  
 **val** memInfo = load.split(**" "**)  
 totalMemory = memInfo[9].toFloat() / 1024  
 } **catch** (ex: IOException) {  
 ex.printStackTrace()  
 }  
 }  
  
 **val** activityManager = getSystemService(Activity.ACTIVITY\_SERVICE) **as** ActivityManager  
 **val** memoryInfo = ActivityManager.MemoryInfo()  
 activityManager.getMemoryInfo(memoryInfo)  
 **val** availableMemory = memoryInfo.availMem / 1048576L  
 memory\_info\_text.text = getString(R.string.memory\_template, availableMemory.toInt(), totalMemory.toInt())  
 **val** progress = ((totalMemory - availableMemory) / totalMemory \* 100).toInt()  
 available\_memory\_progress.progress = progress  
 }  
  
 **private fun** checkIfIgnore(pkgName: String): Boolean {  
 **return** ignoreList.contains(pkgName)  
 }  
  
 **companion object** {  
 **const val** IGNORE\_ARRAY = **"ignore\_array"** }  
}

**class** IgnoreListActivity : AppCompatActivity(), AppListFragment.AppListener {  
 **override fun** switchToApp(packageName: String) {}  
  
 **override fun** endApp(packageName: String) {}  
  
 **override fun** appInfo(packageName: String) {}  
  
 **override fun** ignoreApp(packageName: String) {}  
  
 **private lateinit var** appListFragment: AppListFragment  
 **var** ignoreList = arrayListOf<App>()  
 **lateinit var** ignoreArray: SharedPreferences  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_ignore\_list)  
  
 actionBar?.setDisplayHomeAsUpEnabled(**true**)  
  
  
 appListFragment = AppListFragment()  
 supportFragmentManager  
 .beginTransaction()  
 .replace(R.id.ignore\_list\_container, appListFragment)  
 .commit()  
  
 refreshIgnoreList()  
 }  
  
*// @Override  
// fun onOptionsItemSelected(item: MenuItem): Boolean {  
// when (item.getItemId()) {  
// android.R.id.home -> {  
// // app icon in action bar clicked; go home  
// val intent = Intent(this, TaskManagerActivity::class.java)  
// intent.addFlags(Intent.FLAG\_ACTIVITY\_CLEAR\_TOP)  
// startActivity(intent)  
// return true  
// }  
// else -> return super.onOptionsItemSelected(item)  
// }  
// }* **private fun** refreshIgnoreList() {  
 ignoreArray = getSharedPreferences(IGNORE\_ARRAY, 0)  
 ignoreList.clear()  
  
 **for** (pkgName **in** ignoreArray.all.keys) {  
 **try** {  
 **val** icon = packageManager.getApplicationIcon(packageManager.getApplicationInfo(pkgName, PackageManager.GET\_META\_DATA))  
 **val** title = packageManager.getApplicationLabel(packageManager.getApplicationInfo(pkgName, PackageManager.GET\_META\_DATA))  
 **if** (title != **null**) {  
 **val** app = App(title.toString(), pkgName, icon!!)  
 ignoreList.add(app)  
 }  
 } **catch** (e: Exception) {}  
 }  
 appListFragment.refreshAppList(ignoreList)  
 }  
  
 **override fun** onAppClick(item: App) {  
 ignoreArray.edit().apply**{** remove(item.packageName)  
 apply()  
 **}** refreshIgnoreList()  
 Toast.makeText(**this**, **"App removed from ignore list!"**, Toast.LENGTH\_SHORT).show()  
 }  
  
 **companion object** {  
 **const val** IGNORE\_ARRAY = **"ignore\_array"** }  
}

**class** SettingsActivity : AppCompatActivity() {  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 setContentView(R.layout.activity\_settings)  
  
 actionBar?.setDisplayHomeAsUpEnabled(**true**)  
   
 supportFragmentManager  
 .beginTransaction()  
 .replace(R.id.settings\_fragment\_container, SettingsFragment())  
 .commit()  
 }  
}

**class** SettingsFragment : PreferenceFragmentCompat() {  
 **override fun** onCreatePreferences(savedInstanceState: Bundle?, rootKey: String?) {  
 setPreferencesFromResource(R.xml.preference\_screen, rootKey)  
 }  
}

**class** AppRecyclerViewAdapter(  
 **private val** listener: AppListener?,  
 **private val** context: Context,  
 **private val** isShowMoreButton: Boolean = **false**) : androidx.recyclerview.widget.RecyclerView.Adapter<AppRecyclerViewAdapter.ViewHolder>() {  
  
 **private var** apps: ArrayList<App> = arrayListOf()  
  
 **override fun** onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 **val** view = LayoutInflater.from(parent.context)  
 .inflate(R.layout.fragment\_app, parent, **false**)  
 **return** ViewHolder(view)  
 }  
  
 **override fun** onBindViewHolder(holder: ViewHolder, position: Int) {  
 **val** app = apps[position]  
 holder.icon.setImageDrawable(app.icon)  
 holder.name.text = app.name  
 **if** (isShowMoreButton) {  
 holder.moreButton.setOnClickListener **{  
 val** popupMenu = PopupMenu(context, holder.moreButton)  
 popupMenu.inflate(R.menu.click\_context\_menu)  
 popupMenu.setOnMenuItemClickListener **{** item **->  
 val** packageName = (holder.view.tag **as** App).packageName  
 **when** (item.itemId) {  
 R.id.ccmenu\_endApp -> {  
 listener?.endApp(packageName)  
 **true** }  
 R.id.ccmenu\_switchToApp -> {  
 listener?.switchToApp(packageName)  
 **true** }  
 R.id.ccmenu\_appInfo -> {  
 listener?.appInfo(packageName)  
 **true** }  
 R.id.ccmenu\_ignoreApp -> {  
 listener?.ignoreApp(packageName)  
 **true** }  
 **else** -> **false** }  
 **}** popupMenu.show()  
 **}** } **else** {  
 holder.moreButton.visibility = View.GONE  
 }  
  
  
 with(holder.view) **{** tag = app  
 setOnClickListener **{  
 val** item = it.tag **as** App  
 listener?.onAppClick(item)  
 **}  
 }** }  
  
 **override fun** getItemCount(): Int = apps.size  
  
 **fun** refreshAppList(appList: ArrayList<App>) {  
 apps = appList  
 notifyDataSetChanged()  
 }  
  
 **inner class** ViewHolder(**val** view: View) : RecyclerView.ViewHolder(view) {  
 **val** icon: ImageView = view.app\_icon  
 **val** name: TextView = view.app\_name  
 **val** moreButton: ImageButton = view.more\_button  
 }  
}

**data class** App (  
 **var** name: String,  
 **var** packageName: String,  
 **var** icon: Drawable  
) : Serializable

**class** AppListFragment : androidx.fragment.app.Fragment() {  
 **private var** listener: AppListener? = **null  
 private var** appsAdapter: AppRecyclerViewAdapter? = **null  
  
 override fun** onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 **val** view = inflater.inflate(R.layout.fragment\_app\_list, container, **false**)  
  
 **val** isShowMoreButton = arguments?.getBoolean(**"IS\_SHOW\_MORE"**, **false**) ?: **false** appsAdapter = AppRecyclerViewAdapter(listener, context!!, isShowMoreButton)  
 *// Set the appsAdapter* **if** (view **is** androidx.recyclerview.widget.RecyclerView) {  
 with(view) **{** layoutManager = androidx.recyclerview.widget.LinearLayoutManager(context)  
 **this**.adapter = appsAdapter  
 **}** }  
 **return** view  
 }  
  
 **fun** refreshAppList(appList: ArrayList<App>) {  
 appsAdapter?.refreshAppList(appList)  
 }  
  
 **override fun** onAttach(context: Context) {  
 **super**.onAttach(context)  
 **if** (context **is** AppListener) {  
 listener = context  
 } **else** {  
 **throw** RuntimeException(context.toString() + **" must implement AppListener"**)  
 }  
 }  
  
 **override fun** onDetach() {  
 **super**.onDetach()  
 listener = **null** }  
  
 **interface** AppListener {  
 **fun** onAppClick(item: App)  
 **fun** switchToApp(packageName: String)  
 **fun** endApp(packageName: String)  
 **fun** appInfo(packageName: String)  
 **fun** ignoreApp(packageName: String)  
 }  
}

Ведомость курсового проекта

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные  сведения | | |
|  | | | | Текстовые документы | | | |  | | |
|  | | | |  | | | |  | | |
| БГУИР КР 1–40 04 01 004 ПЗ | | | | Пояснительная записка | | | | 39 с. | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  | | | |  | | | |  | | |
|  |  |  |  |  | БГУИР КП 1-40 04 01 004 Д1 | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Task-manager для ОС Android |  | | | Лист | Листов |
| Разраб. | | Студент | Глотов А.А. | 19.12.18 | Т |  |  | 39 | 39 |
| Пров. | | Руководи- тель | Шиманский В.В. | 19.12.18 | Кафедра информатики гр. 653501 | | | | |
|  | |  |  |  |
|  | |  |  |  |