САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра компьютерного моделирования и многопроцессорных систем

Сахаровский Иван Дмитриевич

Выпускная квалификационная работа бакалавра

Удаленное управление вычислительной системой

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,
доктор физико-математических наук,
профессор
Андрианов С. Н.

Санкт-Петербург

2016

Содержание

Введение

В современном обществе на смену персональным компьютерам приходят такие устройства, как смартфоны, планшетные компьютеры и т.д. Это характеризуется тем, что мобильные устройства компактнее и зачастую мощнее стандартных ПК, которые каждый привык видеть у себя дома, либо на рабочем месте. Так же практически все функции, которые можно выполнить с помощью ПК постепенно становятся доступны и на мобильных устройствах. В связи с развитием этих технологий, стоит задумываться об удобстве любого приложения для конечного пользователя.

Современные приложения, чаще всего работают как приложения типа клиент-сервер, нацелены на простое «общение» с пользователем, они не предоставляют никаких возможностей контролировать процесс взаимодействия. Например, любое приложение, требующее подключение к интернету, передает данные, заданные пользователем, начинает производить над ними операции на сервере, и получает результат своего взаимодействия. Всё то, что происходит с данными за пределами своего устройства, пользователю недоступно.

Такая реализация имеет ряд недостатков[3], например, излишнее время ожидания (при отправлении неправильных, некорректных запросов необходимо дождаться результата выполнение предыдущего запроса), отсутствие информации об использовании данных (при передаче паролей, пользователь не знает будет ли использоваться данный пароль только для авторизации в этом приложении). В связи с этим необходимо иметь возможность управлять использованием данных на другой стороне приложения (сервере).

Актуальность выбранной темы состоит в том, что во всех приложениях, использующих данные для решения математических задач, не используется

возможность управления процессом. Методы, использующиеся в задачах численных методов и подобных им, выполняются не мгновенно, соответственно у пользователя должна быть возможность остановить задачу, либо приостановить её выполнение для проверки.

Постановка задачи

Цель дипломной работы: реализовать вычислительную систему, способную реагировать на запросы клиента во время выполнения вычислительной задачи.

Для достижения данной цели, необходимо решить следующие задачи:

- провести исследование на возможность управления вычислительной системой со стороны клиента
- исследовать инструменты, позволяющие передавать указания вычислительной системе
- выбрать алгоритм, реализующий какой-либо численный метод
- разработать схему приложения для клиентской и серверных частей
- анализ разработанной системы

Для решения этих задач необходимо использование инструментов, способных реализовать обе части системы. В качестве мобильного клиента будем использовать систему Android. Данная система имеет ряд преимуществ, таких как:

1. Язык, на котором пишутся приложения, возможен для реализации серверной части.
2. Система позволяет устанавливать приложения без приобретения аккаунта разработчика, либо взлома системы
3. Преобладание устройств данной системы на рынке

В качестве языка, используемого для реализации серверной части, будет выбран язык программирования Java. Главное преимущество данного языка заключается в том, что данный язык является кроссплатформенным, то есть нет проблем с интеграцией серверной части с одной системы на другую.

Обзор готовых решений

Бенчмаркинг конкурентоспособности – это измерение характеристик предприятия, исследования специфических продуктов, возможностей процесса или административных методов и сопоставление их с характеристиками конкурентов. В терминах приложений для мобильных телефонов, бенчмаркинг представляет собой оценку по десятибалльной шкале, полученную при тестировании функциональности приложения на заявленные требования. В качестве критериев эффективности будут выступать использованы следующие параметры:

- Реализация алгоритмов интегрирования
- Возможность управления процессом интегрирования
- Получение результата во время процесса выполнения программы

По результатам поиска приложений в магазине приложений для Android устройств были также найдены приложения, позволяющие выполнять вычислительные процессы. В приведенной ниже таблице слева перечислены основные аналоги, а в качестве критериев эффективности приведены основные критерии оценки этих приложений.

Таблица 1. Бенчмаркинг конкурентоспособности

№	Критерий эффективности / Приложение	Решение задачи интегрирования	Управление процессом	Промежуточные результаты	Оценка пользователей	Итог
	WolframAlpha	10	4	4	9	256
	MalMath	9	4	2	8	240
	Мое приложение	7	9	8	9	272

Основным выводом, полученным после проведения анализа, заключается в том, что большинство приложений удаленного управления вычислительной системой не предоставляет достаточных возможностей для контролирования процесса вычисления. В моей системе предполагается устранить большинство этих недостатков, используя современные технологии работы с серверами, обработки запросов, работы с многопоточностью.

Глава 1. Основные технологии

В этой главе даются все основные определения и инструменты, которые будут использоваться в данной дипломной работе.

Анализ используемых приложений

С ростом количества мобильных устройств увеличивается и количество приложений, разрабатываемых для каждой мобильной ОС. Каждый день создаются сотни таких приложений. Однако их можно разделить на 3 большие группы[4]:

1. Нативные приложения
2. Web-приложения или мобильная версия сайта
3. Гибридные приложения

Нативные приложения – вид приложений, созданный на «родных» языках ОС (например, для Windows Phone – C#, Android – Java, iOS – Objective-C, Swift). Главное преимущество таких приложений состоит в том, что они оптимизированы под свою систему, а значит работают быстро. Так же такие приложения имеют доступ к аппаратным функциям, то есть могут использовать камеру, микрофон, адресную книгу, геолокацию и т.д.

Нативные приложения являются самыми распространёнными приложениями из-за того, что они написаны под конкретную платформу, учитывая слабые и сильные стороны этой платформы.

Большинство нативных приложений использует архитектуру «клиент-сервер». Доля таких приложений около 70% (остальные 30% распределены между играми, повседневными приложениями типа дневник и т.п.).

Web-приложения, по сути являются просто мобильными версиями сайтов только с расширенными возможностями. Разница между web-верстkой и адаптивной версией сайта не велика, поэтому в обоих случаях применяются стандартные технологии, а скорость работы таких приложений ограничена качеством интернет соединения. При этом web-приложения не выкладываются в магазинах приложений, для них требуется лишь браузер. Но

для таких приложений соответственно возникает проблема совместимости для браузеров (как раньше возникали большие проблемы при верстке сайта для IE9). Иногда же бывает, что web-приложение находится в магазине, но по факту, оно представляет из себя лишь браузер, который может работать с одним сайтом.

Срок жизни таких приложений невелик, так как при изменении функционала приложение придется переписывать заново. Так же такие приложения не безопасны, т.к. не имеют доступа к шифрованию системы.

Основные плюсы таких приложений – кроссплатформенность, низкая стоимость и быстрые сроки реализации.

Гибридные приложения – соединенные нативные и веб-приложения. Главными преимуществом гибридных приложений является кроссплатформенность на web-технологиях и возможность доступа к функциям смартфона, таким как камера, микрофон.

Гибридные и web-приложения дешевле и быстрее выполняются, но качество таких приложений пока оставляет желать лучшего. Так же такие приложения не предоставляют пользователю никаких возможностей, кроме как просмотр своего ресурса. Нативные приложения позволяют в полном объеме контролировать весь процесс работы приложения. Поэтому далее, под мобильным приложением будет иметься ввиду нативное приложение.

Структура архитектуры «клиент-сервер»

В основном мобильные приложения делятся на 3 большие группы:

1. Приложения для ежедневных потребностей
2. Развлекательные
3. Приложения, работающие с удаленными серверами

Первая группа приложений предоставляет собой приложения, которые используются пользователями каждый день, но используют его 1 – 2 раза в сутки. Это приложения типа будильник, список покупок и т.д.

Вторая группа приложений представляет собой приложения, в которые люди заходят, чтобы отвлечься от каких-то дел. К таким относятся игры.

Третья, самая большая группа, это приложения, основной задачей которых является отправка, и прием данных с каких-либо других программ, называемыми серверами. Согласно статистике, полученной из Google Play, на данный момент около 70% приложений относятся к 3 группе. Поэтому необходимо понять структуру этих приложений, определить её основные плюсы и минусы.

Клиент-сервер — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами[8]. В физическом плане и клиент и сервер являются программными обеспечениями, то есть сервер и клиент могут располагаться на одном вычислительном устройстве. Конечно в большинстве случаев сервер — это отдельная машина, на которой стоит программа, ожидающая запросы клиентских программ и предоставляющая этим программам свои ресурсы (например, вычислительные ресурсы, доступ к базе данных, загрузка данных и т.д.). Клиент – это прикладная программа, предоставляющая возможность пользователям «общаться» с сервером посредством запросов, генерируемых самим клиентом. Общение в приложениях с такой архитектурой осуществляется с помощью сетевых протоколов (например, HTTPS). В хороших клиентских программах

пользователь не должен знать о том, какие запросы и с какими параметрами он отправляет.

Данная архитектура имеет ряд достоинств и недостатков.

Преимущества:

- Отсутствие дублирование кода программы-сервера на программах-клиентах
- Снижение требований вычислительной мощности на клиентах (т.к. большинство вычислений происходит на сервере)
- Надежное хранение данных. По причине того, что все данные обычно хранятся на сервере, то для него проще организовать систему контроля полномочий и правил, которые запрещают редактирование либо удаление данных клиентам без особых полномочий.

Недостатки:

- Неработоспособность сервера влечет за собой неработоспособность всей системы.
- Высокая стоимость оборудования.

Архитектура клиент-серверных приложений, реализующая архитектуру «клиент-сервер» в большинстве приложений, состоит из 2 компонентов (клиент и сервер соответственно). Такая реализация называется **двухзвенная архитектура**. Основной принцип заключается в том, что сервер отвечает на запросы в полном объеме, используя только собственные ресурсы. То есть сервер не вызывает сторонние сетевые приложения и не обращается к сторонним сетевым ресурсам для выполнения запроса. Схема такой архитектуры проиллюстрирована на *рисунке 1*.

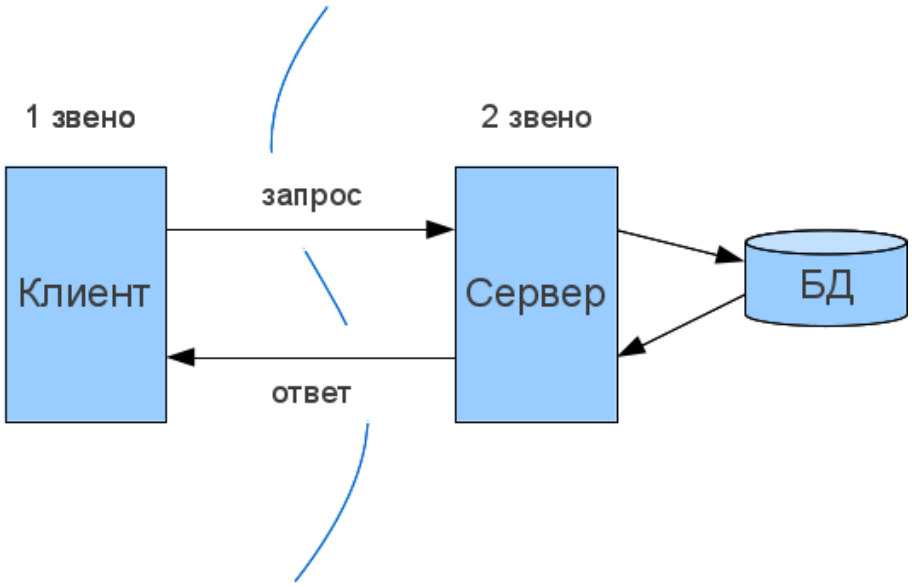

Рисунок 1. Двухзвенная архитектура

Итак, основная задача архитектуры «клиент-сервер» состоит в разделении сетевого приложения на компоненты, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом.

Обоснование выбора OS Android

В XXI веке информационные технологии настолько прочно вошли в нашу жизнь, что большинство людей перестают думать о каких-то бытовых вещах, за которые теперь отвечают технологии. Начиная от управления температурой в комнате и заканчивая датчиками в самолётах для стабилизации полета. Но технологии постоянно развиваются и поэтому появилась необходимость в использовании некого «пульта управления» всеми эти механизма. Первоначально этим занялась компания Apple, создав свою систему iOS для мобильных устройств, которая позволяет разрабатывать сторонние приложения, начиная от возможности смотреть время и заканчивая управлением новых автомобилей.

После разработки системы iOS появилась компания с более амбициозными планами – создать открытую архитектуру, чтобы любой производитель мог сам ее изменять по своему усмотрению, не влияя на общую производительность и актуальность системы. Такой компанией стала Android Inc., которую в июле 2005 года купила компания Google.

Android (Андроид) – это операционная система для смартфонов, планшетов, электронных книг, наручных часов и т.д. (в будущем планируется использовать как основное ПО для роботов и автомобилей). В основе системы лежит ядро Linux и собственная реализация виртуальной Java – машины, разработанная Google.

Начиная с 2011 года Android имеет колоссальную долю на рынке. Практически 4 из 5 смартфонов, находящихся в пользовании, являются смартфонами с операционной системой от Google. А это значит, что разработка приложений именно для этой системы является наиболее популярной и востребованной.

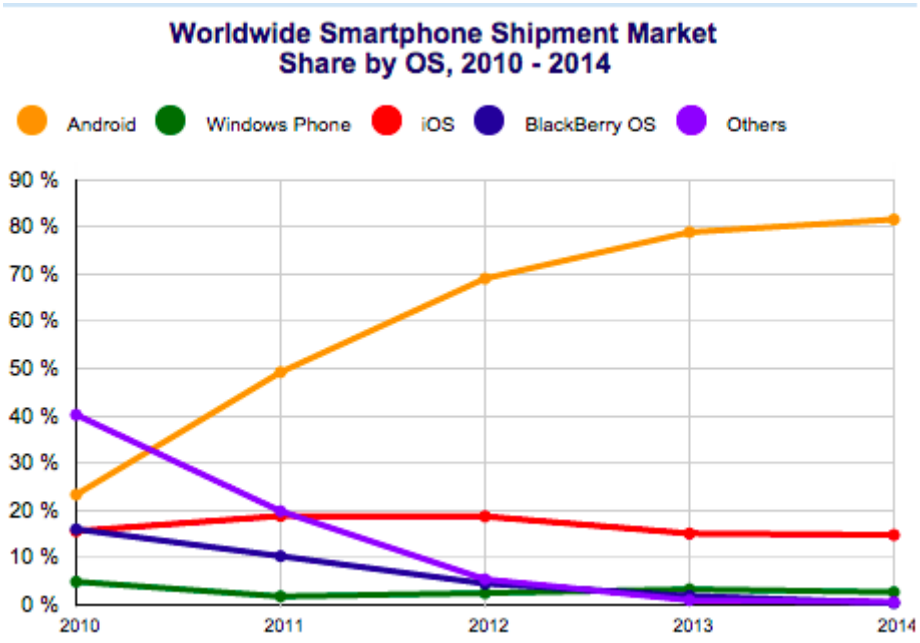

Рисунок 2.Доля смартфонов на рынке по OS

Популярность системы обеспечивается возможностью легкой установки на мобильные устройства любых приложений. Система позволяет подстроить свой телефон полностью под себя. Пользователь может установить приложение, которое написал сам, либо написал его друг, не затрачивая на это лишних действий (к примеру, чтобы установить приложение на iOS, которого нет в AppStore необходимо иметь либо аккаунт разработчика (тоже не бесплатно), либо взламывать систему (так называемый jailbreak)). В Android смартфонах функция разработчика доступна каждому пользователю. Даже эти небольшие причины позволяют говорить, что эта система в будущем будет развивать еще больше и спрос на нее так же будет расти, что приведет к тому, что и потребность в приложениях именно на эту систему так же будет оцениваться еще выше[13].

Так же хочется отметить постоянную изменение системы. Google совершенствует свое детище, выпуская новые версии, в которых улучшает работоспособность приложений. В недавнем времени была выпущена версия Android 6.0 Marshmallow, до которой начинают обновляться все устройства. Начиная с версии 1.0 (первая выпущенная версия) Google улучшает качество своей продукции. Первой успешной версией Android разработчики называют

версию 2.3, которая используется до сих пор на 3% устройств. В версии 4.0 разработчики полностью поменяли оболочку и API для интеграции с разнообразным оборудованием, что позволило поднять систему на новый уровень в плане производительности.

Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%

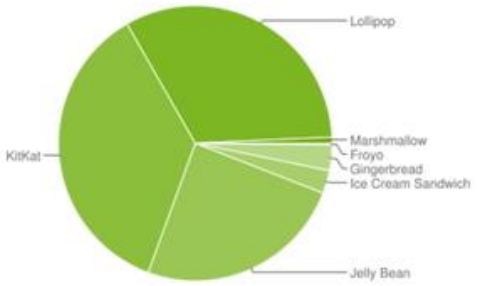

Рисунок 3.Версионность системы за 2016 год

На рис.3 можно заметить, что 96.8% устройств уже обновились до версии 4.0.3. Это значит, что в ближайшее время версии 2.2 и 2.3 будут считаться устаревшими, и писать приложение под эти устройства уже не имеет реальной пользы (устройства на этих версиях android были выпущены до 2013 года, по стандартам производства и статистике смены мобильных устройств, такие устройства либо выйдут из строя, либо будут обменены на более новые). Поэтому сейчас для разработки приложений, минимальной поддерживаемой версией устанавливается версия 4.0.3 (API 15). Именно для этой версии и будет написано приложение.

Инструменты для разработки под Android

Как говорилось ранее, Android – операционная система, основанная на ядре Linux с интерфейсом программирования Java. Поэтому необходимо выбрать тот инструмент, который позволяет программировать на языке Java и интегрируется с системой Android. Для начала надо установить все необходимые компоненты работы с JVM (Java Virtual Machine). Самое последнее ПО для разработки на Java можно загрузить на официальном сайте разработчика, Oracle Corporation. К этому программному комплексу относятся такие инструменты как Java Runtime Environment (в дальнейшем – JRE) и Java Development Kit (в дальнейшем – JDK). Первый инструмент представляет собой среду выполнения – минимальную реализацию виртуальной машины, в которой запускается и выполняется программный код на Java. Второй инструмент – это в свою очередь целый набор инструментов, комплект разработчика приложений на языке Java. JDK включает в себя JRE, различные стандартные библиотеки классов Java, компилятор javac, документации и пример кода. Распространяется Java бесплатно для различных ОС (Mac OS, Linux, Windows).

Сейчас имеется большое количество интегрированных средств разработки на Java, но большинство из них не подходят для разработки на OS Android. Существует 2 инструмента разработки под эту ОС:

1. Eclipse
2. Android Studio

Первоначально компания Google предлагала использовать среду Eclipse для разработки приложений. Для этого они выпустили Android Developer Tools (в дальнейшем – ADT) для интеграции с данной средой. ADT включала в себя Android Software Development Kit (в дальнейшем – SDK), эмулятор устройства, debugger для отлаживания приложений. Но в мае 2013 Google отказалась от поддержки ADT и представила новую среду разработки, названную Android Studio. Данная среда основана на программном обеспечении IntelliJ IDEA от

компании JetBrains. Причиной перехода на новую среду разработки являются функции, которые не поддерживала система Eclipse. Поэтому, после анонсирования новой среды разработки у разработчиков появились новые инструменты такие как:

1. Сборка приложений, основанная на Gradle (автоматический сборщик, построенный на принципах Apache Ant и Apache maven).
2. Статический анализатор кода (Lint), позволяющий находить проблемы производительности, несовместимости версий и другое.
3. Встроенный ProGuard и утилита для подписки приложений.
4. Шаблоны основных макетов и компонентов Android.
5. Встроенная поддержка Google Cloud Platform.

Скачать Android Studio можно с официального сайта разработчиков android[9].

Вывод

Таким образом, невозможно отрицать, что технологии развиваются в очень быстром темпе. Чтобы идти «в ногу со временем» необходимо использовать концепцию постоянного обучения, то есть осваивать новые технологии и улучшать старые. Технологии клиент-серверных приложений приобретают новые возможности, поэтому необходимо придумывать новые инструменты, реализации для улучшение этой архитектуры.

Android технологии так же относятся к быстроразвивающимся и перспективным технологиям. Исходя из описания, предоставленного выше, стоит констатировать факт, что система Android предоставляет больше всего возможностей для реализации новых технологий, связанных с мобильными устройствами.

Исходя из всего вышеперечисленного, я считаю целесообразным разработать программное обеспечение для управления вычислительной системой именно на платформе Android, а так как языком, на котором следует писать мобильные приложения под данную платформу является Java, то использовать этот язык так же для реализации серверной части системы.

Глава II. Описание работы приложения

В этой главе описываются структура работы системы и технологические инструменты, используемые для реализации данной структуры.

Структура Android приложения

При создании проекта система Android Studio генерирует .gradle файл, который является полным описанием приложения. [14] В этом файле указывается минимальная версия системы, для которой создано приложение, все подключаемые сторонние библиотеки. Так же создается xml файл, который имеет константное имя – AndroidManifest.xml. В этом файле описываются все разрешения, которые необходимы приложению для работы (сеть, звонки, смс, доступ к камере и т.д.). Также в манифест-файле записаны все экраны, используемые приложением, все сервисы (процессы, происходящие вне зависимости от состояния приложения и не влияющие на работу графического интерфейса).

Программное приложение для системы Android включает в себя набор активностей (Activity), каждая из которых реализует как минимум один экран приложения. Каждая из таких Activity описывается своим классом на языке Java, причем для этого класса необходимо наследовать класс Activity, расположенный в пакете android.app Android SDK. Весь интерфейс описывается в прикрепленном xml файле. Чтобы установить определенный xml файл к конкретной Activity необходимо передать имя этого файла, используя функцию *setContentView(long layoutId)* внутри переопределённой функции *onCreate()*. При подключении файла система автоматически распознает экран мобильного устройства и форматирует контент в соответствии с заданной разметкой. Таким образом, для различных размеров экранов не обязательно создавать разные xml файлы.

При запуске приложения система находит Activity в манифест-файле, у которой стоит параметр launcher. В этот момент у Activity начинается

жизненный цикл, вызываемый функцией *onCreate()*. Во время работы жизненный цикл может оказаться в 3 состояниях:

1. Экран создан, и находится на переднем плане приложения
2. Экран создан, но находится не на переднем плане приложения
3. Экран удален

Во время состояния 1 пользователь может взаимодействовать со всем графическим интерфейсом приложения, передавать команды и т.д. Во время состояния 2 пользователь видит экран приложения, на экране могут происходить некоторые изменения, пользователь не может влиять на интерфейс и взаимодействовать с ним. При переходе в состояние 3 все данные, которые хранит приложение на этом экране будут удалены, соответственно перед переходом в это состояние необходимо сохранять все важные данные, которые могут понадобится во время следующего запуска этого экрана. Обычно для хранения используют реляционную базу данных SQLite, которая встроена в систему. Работа с базами данных организована с помощью библиотеки, встроенной в систему. Основные классы для работы с данной библиотекой лежат в пакете android.database.sqlite[7].

Так как в приложениях, реализующих архитектуру клиент-сервер, существует методы для работы с сетью, то возникает вопрос: Возможно ли передавать запросы на сервер, не находясь постоянно на экране какого-либо Activity? Разработчики Android OS решили этот вопрос внедрением класса Service, который может работать вне зависимости от конкретного экрана приложения и его состояние. Это значит, что сервис возможно запустить даже если приложение находится в свернутом состоянии, либо во время перехода с одного экрана на другой. Сервис предоставляет возможность выполнять нетрудоёмкие задачи, которые укладываются в небольшой промежуток времени. То есть, во время работы приложения мы можем передавать данные на сервер, при этом работаю с другими функциями приложения. Такой подход позволяет не задумываться о возможной потере данных при передаче[2].

Структура серверного приложения на Java

Во время создания Java приложения необходимо создать класс, который будет выполняться первым при запуске. Чтобы определить такой класс необходимо определить у него метод *public static void main(String[] args)*, где аргументы этого метода – параметры, переданные из командной строки. Основой для реализации архитектуры клиент-сервер является технология *socket,* входящая в состав пакета java.net.Socket. Socket – технология, позволяющая уникально задать адрес приложения для компьютера в сети. Уникальность достигается с помощью двух компонентов:

1. IP-адрес
2. Порт-число

С помощью IP-адреса можно уникально определить каждый компьютер, находящийся в сети. Порт позволяет уникально определить программу, которая будет исполняться на данном компьютере. Сокеты основаны на базе сетевого протокола TCP/IP. Так же они позволяют постоянно поддерживать соединение, тем самым облегчая задачу постоянной проверки с помощью обычных запросов.

Для создания сервера необходимо создать экземпляр класса *ServerSocket,* указав ему в параметрах порт, на котором будет работать приложение. После этого необходимо перейти в ожидание нового подключения, а при создании подключения установить потоки ввода-вывода для обмена запросами между двумя сокетами. Переход в ожидание осуществляется с помощью функции *ServerSocket.accept(),* которая возвращает сокет, соединяющий сервер с клиентом. После получения сокета создаются потоки вводы-вывода (*InputStream* и *OutputStream* соответственно). *InputStream* используется для получения запросов от клиента с помощью функций *readLine(), readInt()* и т.д. Далее следует обработка запроса. Чаще всего запросы выглядят как json файл со списком параметров. Использование Json имеет ряд преимуществ таких как:

- Компактность
- Удобность в чтении
- Удобное преобразование в структуру данных

Для создания json файла в java используют библиотеку org.json, в которой располагаются такие классы для работы с json объектами как: JSONObject, JSONArray и т.д. Параметры получают с помощью функций *getDouble(String paramName)* и т.п. [10]

Для создания подключения используют многопоточное программирование в Java, то есть для каждого нового подключения генерируют новый поток, в котором происходит обработка запросов, выполнение операций, составление результатов и отправка ответа. Чтобы создать новый поток необходимо наследовать класс от базового класса *Thread* и переопределить метод *run(),* в котором происходят все алгоритмы работы с данными.

При закрытии соединения необходимо вызвать функцию *socket.close(),* которая передает серверу, что необходимо остановить поток обработки запросов и также закрыть соединение со стороны сервера. При неожиданном разрыве соединения на сервере сокет «выбрасывает» ошибку IOException, которую можно обработать блоком try-catch. При перехвате такой ошибки необходимо со стороны сервера закрыть потоки ввода-вывода и закрыть данный сокет.

При закрытии серверного сокета необходимо отключить все подключенные к нему клиентские сокеты, после чего вызвать функцию *ServerSocket.close()* и остановить серверное приложение. [6]

Описание разработанной системы

При решении задачи удаленного управления вычислительной системы необходимо было создать систему, способную решать некоторую математическую задачу, используя вычислительные методы. В качестве такой задачи была выбрана задача интегрирования, то есть вычислительная система

позволяет находить определённые интегралы от заданных функций, причем во время любой итерации пользователь может приостановить программу, либо остановить вычисление полностью. Такая возможность предоставляет полный контроль над сервером.

При запуске приложения на мобильном устройстве создается сокет, который подключается к серверу. Создание подключения происходит асинхронно, то есть система не дожидается подключения, а предоставляет функции ввода данных при отсутствии подключения. При попытке передать данные при недоступном соединении приложение должно выводить сообщение об ошибке подключения.

Как было описано выше, все запросы передаются в формате json, который позволяет удобно сериализовать данные. После приема все данные десериализуются с помощью класса JsonObject и запускается процесс подсчета интеграла, используя параметры, полученные в json. К таким параметрам относятся:

1. Функция, которую необходимо интегрировать
2. Начальная и конечная точка
3. Количество участков разбиения

Во время выполнения процесса, пользователь может послать новый запрос, например, для паузы интегрирования. Получив такую команду, сервер передает в поток выполнения вычисления информацию о том, что процесс необходимо приостановить.

Выполняя численное интегрирование, на каждой итерации необходимо передавать промежуточный результат. Для этого после каждого прохода цикла поток интегрирования передает в основной поток соединения с клиентом информацию о текущем состоянии вычисления. После этого составляется json файл, в котором в качестве параметра указывается состояние процесса (номер итерации, либо информация о том, что все итерации пройдены), и результат, соответствующий этому состоянию.

При выходе из клиентского приложения передается запрос к серверу на завершение сессии, после чего происходит завершение потока, отвечающего за соединение с данным клиентом на сервере.

Глава III. Реализация

В этой главе описывается реализация системы удаленного управления. При создании проекта используются технологии, описанные в первых двух главах.

Реализация клиентской части

В клиентском приложении для корректной работы приложения необходимо создать 3 потока: 1 поток отвечает за взаимодействие с пользователем (его также называют UI-поток, либо главный поток), 2 поток отвечает за соединение с сервером и передачу данных. Соответственно вся информация, отправленная на сервер будет обрабатываться в этом потоке.[11] Последний поток отвечает за прием сообщений с сервера и передачу этих сообщений в главный поток, чтобы пользователь мог увидеть информацию, полученную от сервера.

Для создания графического интерфейса будут использоваться стандартные компоненты Android, такие как[12]:

1. Кнопки
2. Поля для ввода значений
3. Текстовые поля

Графический интерфейс содержит в себе кнопки для начала/паузы/остановки вычислений. Также создаются поля для ввода значений, которые будут переданы на сервер для дальнейшего вычисления (см рис. 4). [1]

После ввода всех значений и нажатия на кнопку начала вычислений на генерируется json файл, имеющий следующую структуру:

1. method: String
2. params: json

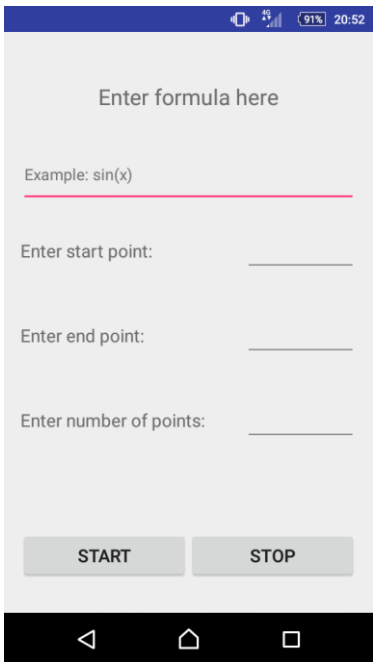

Рисунок 4.Экран клиентского приложения

Метод включает себя команду для сервера, которую ему необходимо выполнить. В моей системе содержится 5 методов, которые может обрабатывать сервер:

1. Start – начинает операцию численного интегрирования
2. Pause – приостанавливает операцию интегрирования
3. Continue – возобновляет операцию интегрирования
4. Stop – завершает операцию вне зависимости от её текущего состояния
5. Disconnect – получает при отключении клиента от сервера, останавливает поток, отвечающий за обмен запросами с клиентом

Параметры содержат в себе необходимые параметры для выполнения этого метода. Например, для метода Start, который начинает вычисление задачи численного интегрирования, параметрами являются функция интегрирования, начальная и конечная точка интегрирования и количество разбиений.

После установления соединения запускается поток, отвечающий за прием запросов. Сервер также передает запросы в формате json, которые имеют структуру, схожую с запросами клиента. После приема запроса, клиент

десериализует его и получает параметры, отправленные сервером. Далее необходимо передать эту информацию пользователю. Так как поток, в котором происходит обработка входящих запросов не может взаимодействовать с интерфейсом пользователя, то необходимо реализовать межпоточное «общение». Для этого в Android OS используют класс *EventBus*, который является частью библиотеки Google Guava. EventBus позволяет «кинуть» событие из любого потока и передать другому потоку, что он должен отреагировать на это событие. В качестве событий можно использовать любой класс java приложения. Для взаимодействия между потоками в моем приложении используется класс *ServerRequestParams(String jsonParams)*, который принимает параметры в качестве входных данных. UI-поток, который подписан на принятие такого типа данных из других потоков, выводит информацию пользователю на экран.

При закрытии приложения (метод *onStop()*, см. описание выше) на сервер отправляется запрос о закрытии сокета. После этого закрывается клиентский сокет и завершается приложение.

Реализация серверной части

Как было описано выше сервер создает новый поток для каждого нового подключения. В случае моего приложения, на сервере запущено ожидание клиентов с помощью функции *ServerSocket.accept()*, которая возвращает сокет подключения к клиенту. Далее этот сокет передается в новый поток, который реализован с помощью класса *Connection* наследуемого от класса *Thread*. При инициализации нового соединения класс Connection переходит в режим ожидания, в котором он принимает команды от клиента. Все запросы, приходящие от клиента, описаны ранее.

При получении команды Start происходит разбор параметров, переданных от клиента. После получения параметров, начинается разбор функции, используя метод обратной польской записи. В этой записи

автоматизация основывается на использовании стека. При получении символа он перемещается в вершину стека, если это операнд, в ином случае (символ это операция) из стека извлекается необходимое для этой операции данных и результат этот операции кладется в вершину стека. Цикл повторяется до тех пор, пока в стеке не останется одно значение, которое и будет являться результатом операции. [5]

После разбора выражения запускается новый поток, в котором происходит разбивка заданного интервала на определенное количество отрезков, указанное в параметрах. Далее начинается процесс интегрирования. После каждой итерации сервер передает клиенту команду с промежуточным либо конечным результатом. Все команды также записываются в формате json, который имеет структуру клиентского запроса, но с другими методами. Сервер может передавать одну из трех команд:

1. StepResult – промежуточный результат
2. FinalResult – ответ
3. Error – ошибка во время разбора функции

При получении запроса Pause сервер приостанавливает поток интегрирования, ожидая новой команды от клиента.

Запрос Continue/Stop продолжает выполнение потока/ останавливает поток.

При получении запроса Disconnect сервер останавливает поток, отвечающий за соединение с клиентом, соответственно все дочерние потоки так же будут остановлены.

Заключение

Краткие выводы. В современном мире, где технологии развиваются каждый день, нельзя не сказать об актуальности разработки системы, соответствующей тематике поставленной проблеме. Данную систему необходимо реализовывать во всех приложениях, использующих «тяжелые» вычисления, либо затрачивающие много времени на обработку какого-либо запроса.

В рамках дипломной работы были достигнуты следующие задачи:

1. Произведен анализ архитектуры клиент-сервер и приложений, реализующих эту архитектуру.
2. Исследованы новые технологии для разработки мобильных приложений, инструменты для создания клиент-серверных приложений

Перспективы развития. Улучшение программы включает в себя добавление новых решений задач численных методов, различные улучшения интерфейса.

Положение, выносимое на защиту. Создана система удаленного управления на базе архитектуры клиент-сервер с использованием технологии Socket, решена задача численного интегрирования в этой системе, создана возможность управления вычислением с клиентского приложения.

Список литературы

1. Амелин К. С., Граничин О. Н., Кияев В. И., Корявко А. В.. Введение в разработку приложений для мобильных платформ. Издательство ВВМ, 2011.
2. Варакин М.В. Разработка мобильных приложений под Android. УЦ «Специалист» при МГТУ им. Н. Э. Баумана, 2012.
3. Коржов В. Многоуровневые системы клиент-сервер // Сети. 1997. N 6. С.72-75
4. Медникс З., Дорнин Л. Программирование под Android. Издательство Питер, 2012.
5. Никлаус Вирт Алгоритмы и структуры данных. Новая версия для Оберона ДМК Пресс, 2010
6. Herbert Schildt Java: The Complete Reference, Ninth Edition, 2015
7. Mark Murphy The Busy Coder's Guide to Android Development // CommonsWare, 2016
8. Лекция на тему «Классификация клиент – серверных приложений» http://www.4stud.info/networking/lecture5.html
9. Основы разработки под Android http://developer.android.com/intl/ru/index.html
10. Основы формата json, реализция json в языке программирования Java https://learn.javascript.ru/json
11. Простые клиент-серверные взаимодействия на Android https://habrahabr.ru/post/269135/
12. Material Design в Android https://habrahabr.ru/company/redmadrobot/blog/252773/
13. Статья: причины популярности андроид http://www.androidtalk.ru/articles/operatsionnaya-sistema-android/
14. Android Gradle for automatic library install https://docs.gradle.org/current/dsl/org.gradle.api.tasks.javadoc.Javadoc.html