Оглавление

Введение

1 Аналитическая часть

В данном разделе проводится сравнительный анализ методов решения поставленной задачи, делается обоснованный выбор метода.

1.1 Постановка задачи

В соответствии техническим заданием необходимо разработать программное обеспечение для удаленного подключения к терминалу (сервер) нескольких пользователей одновременно (клиенты). Сервер поддерживает связь с множеством клиентов одновременно, поддерживая обработку запросов от каждого и рассылая изменения всем подключенным в определенный момент клиентам. По необходимости запускает команду терминала, высылая результат всем подключенным клиентам.

1.2 Анализ предметной области

1.3 Модель клиент-сервер

В модели клиент-сервер роли определены: сервер предоставляет ресурсы и службы одному или нескольким клиентам, которые обращаются к серверу за обслуживанием. В качестве примеров серверов можно привести веб-серверы, почтовые серверы и файловые серверы . Каждый из этих серверов предоставляет ресурсы для клиентских устройств, таких как настольные компьютеры, ноутбуки, планшеты и смартфоны . Большинство серверов могут устанавливать отношение «один ко многим» с клиентами, что означает, что один сервер может предоставлять ресурсы нескольким клиентам одновременно. Когда клиент запрашивает соединение с сервером, сервер может либо принять, либо отклонить это соединение. Если соединение принято, сервер устанавливает и поддерживает соединение с клиентом по определенному протоколу. Например, почтовый клиент может запросить SMTP-соединение с почтовым сервером для отправки сообщения.

Затем приложение SMTP на почтовом сервере запросит проверку подлинности у клиента, например адрес электронной почты и пароль. Если эти учетные данные совпадают с учетной записью на почтовом сервере, сервер отправит электронное письмо целевому получателю. Часто клиенты и серверы взаимодействуют через компьютерную сеть на разных аппаратных средствах, но и клиент и сервер могут находиться в одной и той же системе. Хост сервера запускает одну или несколько серверных программ, которые совместно используют свои ресурсы с клиентами. Клиент не предоставляет общий доступ ни к одному из своих ресурсов, но запрашивает данные или службу у сервера. Поэтому клиенты инициируют сеансы связи с серверами, которые ожидают входящих запросов. Клиенту не знает о том, как работает сервер при выполнении запроса и доставке ответа. Клиент должен только понимать ответ, основанный на хорошо известном прикладном протоколе, т.е. содержание и форматирование данных для запрашиваемой службы. Клиенты и серверы обмениваются сообщениями в шаблоне обмена сообщениями запрос-ответ. Клиент отправляет запрос, а сервер возвращает ответ.

1.4 Сокеты

Сокет (англ. socket [**?**])— конечный пункт передачи данных, абстракция, обозначающая одно из окончаний сетевого соединения. Каждая из программ, устанавливаю щих соединение, должна иметь собственный сокет. Сокеты – это интерфейс прикладного программирования для сетевых приложений TCP/IP. Интерфейс сокетов был создан в восьмидесятых годах для операционной системы UNIX. Позднее интерфейс сокетов был перенесен в Microsoft Windows. Сокеты до сих пор используются в приложениях для сетей TCP/IP. Сетевые приложения используют сокеты, как виртуальные разъемы для обмена данными между собой. Сокеты бывают трех видов: клиентские, слушающие и серверные.

1.4.1 Принципы сокетов

Каждый процесс может создать слушающий сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024). Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить таймаут для операции и т.д. Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него. Сокеты типа INET доступны из сети и требуют выделения номера порта [1]. Обычно клиент явно подсоединяется к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером.

1.4.2 Основные функции сокетов

В таблицах 1.1 - 1.3 представлены общие функции, функции сервера и клиента соотвественно.

Функция	Описание
`socket`	Создать новый сокет и вернуть файловый дескриптор
`send`	Отправить данные по сети
`receive`	Получить данные по сети
`close`	Закрыть соединение

Таблица 1.1: Общие функции

1.4.3 Типы сокетов

Различают два типа сокетов:

Функция	Описание
`bind`	Связать сокет с IP-адресом и портом
`listen`	Слушает порт и ждет когда будет установлено соединение
`accept`	Принять запрос на установку соединения

Таблица 1.2: Функции сервера

Функция	Описание
`connect`	Установить соединение

Таблица 1.3: Функции клиента

— сокеты с предварительным установлением соединения, когда до начала передачи данных устанавливаются адреса сокетов отправителя и получателя данных – такие сокеты соединяются друг с другом и остаются соединенными до окончания обмена данными;

— сокеты без установления соединения, когда соединение до начала передачи данных не устанавливается, а адреса сокетов отправителя и получателя передаются с каждым сообщением.

Если тип сокета – виртуальный канал, то сокет должен устанавливать соединение, если же тип сокета – датаграмма, то, как правило, это сокет без установления соединения, хотя последнее не является требованием.

1.5 Обработка запросов клиентов

При приходе клиентского запроса у сервера имеется несколько вариан тов действий:

— обрабатывать все запросы в одном потоке;

— обрабатывать каждый запрос в отдельном потоке;

— организовать пул потоков.

1.5.1 Обработка всех запросов в одном потоке

Это решение подходит только для ограниченного числа случаев, в которых количество клиентов невелико, и обращаются они к серверу не часто. Эта самая простая схема работы: минимум потоков, минимум ресурсов, нет синхронизации. Необходимо построить очередь входящих запросов, чтобы они не терялись при последовательной обработке. Но при большом количестве клиентов, клиенты будут долго ждать ответа от сервера.

1.5.2 Обработка каждого запроса в отдельном потоке

Для каждого клиентского запроса создается отдельный поток. Такая схема работает следующим образом: первичный поток приложения прослушивает клиентские запросы и при поступлении каждого создает новый поток, передавая ему клиентский пакет (данные или команду). Созданный поток выполняет соответствующую обработку, передает результаты обратно клиенту, или же помещает их в БД, и завершает свое существование.

Основные недостатки такой модели:

— частое создание и завершение потоков;

— малое время работы потока;

— нерегулируемое количество потоков;

— в большинстве случаев отсутствие очереди клиентских запросов;

— большое количество переключений контекстов рабочих потоков.

Для решения этих проблем и предназначен пул потоков.

1.5.3 Организация пула потоков

Имеется главный поток приложения, прослушивающий клиентские запросы. Пул потоков создается заранее или при поступлении первого за-

проса. При поступлении запроса главный поток выбирает поток из пула и передает ему запрос. Если все потоки занимаются обработкой, то есть активны, пакет ставится в очередь и ждет освобождения одного из потоков. Алгоритмы добавления потоков в пул и определения оптимального размера пула сильно зависят от решаемой задачи [2].

1.6 Протколы

Протокол – набор правил, определяющих взаимодействие двух одноименных уровней модели взаимодействия открытых систем в различных абонентских ЭВМ. Стек протоколов — набор взаимодействующих сетевых протоколов.

Наиболее популярные стеки протоколов: TCP/IP, IPX/SPX, NetBIOS/SMB, DECnet, SNA и OSI. Большинство протоколов (все из перечисленных, кроме SNA) одинаковы на физическом и канальном уровне, но на других уровнях как правило используют разные протоколы.

1.6.1 Протоколы транспортного уровня

Протоколы транспортного уровня предназначены для обеспечения непосредственного информационного обмена между двумя пользовательскими процессами. Существует два типа протоколов транспортного уровня – сегментирующие протоколы и не сегментирующие протоколы доставки дейтаграмм [3].

Протоколы доставки дейтаграмм просты для реализации, однако, не обеспечивают гарантированной и достоверной доставки сообщений.

В качестве протоколов транспортного уровня в сети Internet могут быть использованы два протокола:

— UDP – User Datagram Protocol;

— TCP – Transmission Control Protocol.

Описание принципов построения протокола UDP приведено в RFC 768. Для передачи сообщений UDP используются пакеты IP. Сообщения UDP в данном случае размещаются в поле данных переносящего их пакета.

Дейтаграммы UDP имеют переменную длину и состоят из заголовка сообщения UDP header и собственно сообщения UDP Data. В таблице 1.4 приведена структура заголовка сообщения UDP.

UPD SOURCE PORT	UPD DESTINATION PORT
UDP MESSAGE LENGTH	UPD CHECKSUM
DATA	DATA

Таблица 1.4: Структура заголовка UDP

Протокол TCP используется для обеспечения надежного информаци онного обмена на транспортном уровне в сетях Internet.

Существует достаточно много причин, которые могут помешать пакету, который передается в сети, успешно достичь станции назначения. Таким образом, если не будут использованы специальные методы для обеспечения гарантированной доставки, принятое сообщение может существенным образом отличаться от того сообщения, которое было передано.

Надежный информационный обмен предполагает следующие возможности:

— потоковый обмен;

— использование виртуальных соединений;

— буферизированная передача данных;

— неструктурированный поток;

— обмен в режиме полного дуплекса.

Структура заголовка TCP показана на риснуке 1.1.

Рис. 1.1: Структура заголовка TCP

1.6.2 Протоколы прикладного уровня

Протоколы прикладного уровня описывают взаимодействие между кли ентской и серверной частями программы. Прикладные протоколы работают на верхнем уровне модели OSI. Они обеспечивают взаимодействие приложений и обмен данными между ними **TODO**.

Популярные протоколы прикладного уровня:

- HTTP
- HTTPS
- FTP
- STMP, IMAP-4, POP3
- TELNET
- SNMP

Вывод

2 Конструкторская часть

В данном разделе будет рассмотрена <...>

2.1 Состав программного обеспечения

2.2 Функциональная модель

2.3 Сценарий использования

2.4 Проектирование протокола прикладного уровня

Вывод

В данном разделе была рассмотрена <...>

3 Технологическая часть

В данном разделе рассматривается выбор языка программирования для реализации поставленной задачи, листинги реализации разработанного программного обеспечения и приведены результаты работы ПО.

3.1 Выбор языка программирования

3.2 Детали реализации

3.2.1 Реализация сервера

3.2.2 Реализация клиента

3.2.3 Перадача пакетов

3.3 Примеры работы разработанного ПО

Вывод

В данном разделе был обоснован выбор языка программирования, рассмотрены листинги реализованного программного обеспечения и приведены результаты работы ПО.

Заключение

В ходе проделанной работы был разработан загружаемый модуль ядра, предоставляющий информацию о загруженности системы: количество системных вызовов за выбранный промежуток времени, количество свободной и доступной оперативной памяти, статистика по процессам и в каких состояниях они находятся.

Изучены структуры и функции ядра, которые предоставляют информацию о процессах и памяти. Проанализированы существующие подходы к перехвату системных вызовов.

На основе полученных знаний и проанализированных технологий реализован загружаемый модуль ядра.

Литература

[1] Сокеты - сетевое программирование [Электронный ресурс]. Режим доступа: https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html (дата обращения: 08.12.2021).

[2] Многопоточность [Электронный ресурс]. Режим доступа: https://se.ifmo.ru/documents/10180/1422934/prog-concurrency.pdf/da25f4c8-02fc-506d-79da-7ea9a2186dac (дата обращения: 08.12.2021).

[3]