**Описание решения**.

Документ содержит описание сетевого варианта решения, подробное описание которого можно найти в [1] и [2]. Используется архитектура “клиент-сервер”. Приложение chat\_server реализует функции сервера, приложение chat\_client реализует функции клиента. Данные хранятся в БД на стороне сервера. Для работы приложений требуется пакет MS Visual C++ redistributable.

Первая часть документа содержит общее описание, вторая часть содержит описание архитектуры решения и списки исходных файлов, третья часть содержит подробное описание протокола обмена между клиентом и сервером.

**Общее описание**.

Общая организация решения показана на рис. 1. Белым цветом показаны файлы, используемые приложениями. Пунктирной линией показан показан обмен сообщениями через сеть.

Client side

Server side

chat\_client

chat\_client\_conf

chat\_server\_log

chat\_server\_conf

chat\_server

chat\_data

Рис. 1. Общая организация решения.

Транспортное соединение между клиентом и сервером создается только на время выполнения операции клиента, на это же время на стороне сервера создается сессия пользователя. В процессе выполнения операции происходит диалог между клиентом и сервером, по окончании которого разрывается транспортное соединение и закрывается сессия пользователя на стороне сервера.

На клиентской стороне имитируется протяженная сессия пользователя, после выполнения успешной аутентификации пользователя (или после успешного создания нового пользователя) запоминаются учетные данные. Аутентификация пользователя выполняется при каждой операции клиента. Такой подход позволяет организовать “одновременную” работу нескольких клиентов с сервером.

***a) Клиентская часть.***

Клиентская часть представлена приложением chat\_client. Человеко-машинный интерфейс chat\_client не изменился и полностью идентичен интерфейсу, описанному в [1].

Приложение использует конфигурационный файл chat\_client\_conf, содержащий следующие параметры, каждый из которых представлен строкой.

Server\_ip IP-адрес сервера.

Server\_port Порт сервера

Изменение этих параметров требует перезапуска приложения.

***б) Серверная часть.***

Серверная часть представлена приложением chat\_servert. Сервер обрабатывает запросы, поступающие от клиентов, информация о запросах и возможных ошибках может выводиться на консоль и записываться в лог-файл. Приложение использует следующие файлы.

chat\_data файл с базой данных

chat\_server\_log лог-файл сервера

chat\_server\_conf конфигурационный файл сервера.

База данных, как и прежде, содержит данные трех пользователей и данные переписки между ними.

Конфигурационный файл сервера содержит следующие параметры, каждый из которых представлен отдельной стороной.

Server\_ip IP-адрес сервера.

Server\_port Порт сервера

Loglevel Параметр, определяющий подробности записи в лог файл.

Параметр Loglevel может принимать следующие значения.

ERROR выводится только информация об ошибках сервера

DETAIL выводится информация об ошибках сервера, обо всех запросах и о всех ответах

Изменение параметров в chat\_server\_config требует перезапуска приложения. Для остановки сервера следует просто закрыть приложение..

**Архитектура решения**.

По сравнению с вариантом, описанным в [2] была несколько изменена архитектура решения, были добавлены промежуточные слои Access и Transport. В связи с этим незначительно изменился модуль chat\_interface.cpp, который теперь использует интерфейс, описанный в chat\_interface.h, и реализуемый модулем chat\_client.cpp. Модули chat\_client.cpp и chat\_server.cpp реализуют протокол обмена между клиентом и сервером.

Чтобы основная часть решения была независима от операционого окружения, все сетевые функции реализованы в модуле network.cpp, интерфейс которого описывается файлом network.h

Chat\_Client

Interface

chat\_interface.cpp

buffer.cpp

chat\_interface.h

Access

buffer.h

chat\_client.cpp

network.cpp

network.h

chat\_client.h

Transport

client\_net.cpp

Network

Chat\_Server

Access

Transport

Logic

DBCS

server\_net.cpp

network.h

chat\_server.h

network.cpp

buffer.h

chat\_server.cpp

chat\_access.h

buffer.cpp

chat\_class.h

Hash\_tabler.cpp

hash\_table.h

chat\_logic.cpp

chat\_object.h

sha1.cpp

sha1.h

chat\_object.cpp

storage\_access.h

file\_storage.cpp

Storage

ram\_storage.cpp

Рис. 2. Схема зависимостей модулей.

Модули client\_net.cpp и server\_net реализуют цикл обмена сообщениями в процессе диалога. Организация диалога сама по себе не зависит от протокола обмена, реализуемого chat\_server и chat\_client и не записит от реализации модуля network.cpp.

Логика работы клиента показана на рис. 3 (пунктиром показана передача сообщений через сеть). Интерфейс вызывает функции, описанные в chat\_interface.h и реализованные в chat\_client.cpp. Клиент формирует данные запроса на сервер и вызывает функцию send\_Request, которая после установления соединения с сервером отправляет запрос.

server

client\_net

chat\_client

chat\_interface

send\_Request

operation (data)

request

reply

client\_Processing

result

request

…

reply

client\_Processing

request (end)

success, finish

Завершение send\_Request, success

Завершение operation

Рис. 3. Логика работы клиента.

После получения ответа send\_Request вызывает функцию client\_Processing, которая обрабатывает ответ и формирует очередной запрос. Обмен продолжается до тех пор, пока client\_Processing не устанавливает флаг окончания. В этом случае send\_Request отправляет на сервер запрос окончания диалога и завершает работу. Функция client\_Processing может также вызывать функцию show\_Message, по окончании диалога может быть вызвана функция show\_Result, которая выводит информацию об ошибке. Функция client\_Processing также может устанавливать флаг success, если операция выполнена успешно.

На стороне сервера функция wait\_Request прослушивает порт, при установлении транспортного соединения начинает прием запросов и формирование ответов. Если запрос принят, вызывается функция server\_Processing, анализирующая запрос и формирующая ответ, при этом может вызываться функция chat\_logic. После приема запроса на окончание диалога server\_Processing устанавливает флаг finish, и wait\_Request снова переходит к прослушиванию порта.

chat\_logic

chat\_server

server\_net

client

server\_Processing

request

operation (data)

reply

…

…

server\_Processing

request (end)

finish

Рис. 4. Логика работы сервера.

Модуль network.cpp реализует выполнение сетевых функций и делает реализацию основной части решения независимой от операционного окружения. Интерфейс модуля определяется заголовочным файлом network.h.

Время ожидания поступления очередного запроса от клиента ограничено величиной TIMEOUT (в мсек). По истечении таймера считается, что время ожидания пакета истекло, сервер закрывает транспортное соединение и вновь переходит к ппрослушиванию порта. Это позволяет сделать работу сервера независисимой от возможных сетевых сбоев или некоррекной работы клиента. Если запрос от клиента не поступает вовремя, сервер не “вешается”, а продолжает работу.

Время ожидания поступления очередного ответа от сервера также ограничено величиной TIMEOUT, по истечении таймера клиент фиксирует ошибку обмена. Это позволяет сделать работу клиента назависимой от сетевых сбоев и некорретной работы сервера.

**Описание протокола**.

Все сообщения представлены в текстовой форме. Сообщение содержит код запроса или ответа, и, возможно, параметры. Коды запросов и параметры представлены строками. (см. таблицу 1).

Каждый диалог между клиентом и сервером, за исключением регистрации нового пользователя, начинается с процедуры аутентификации пользователя, в случае успешной аутентификации на сервере создается сессия пользователя, далее выполняется операция пользователя. Каждый диалог завершается передачей серверу сообщения R\_END, после этого завершается сессия пользователя на стороне сервера, разрывается транспортное соединение.

Создание сессии на стороне клиента выполняется при регистрации пользователя или при входе в сессию. В первом случае выполняется создание учетных данных пользователя на стороне сервера, во втором случае – проверка учетных данных, при этом в случае успешного выполнения на стороне клиента учетные данные запоминаются, на стороне сервера сессия пользователя сразу завершается.

**Таблица 1. Сообщения**.

|  |  |  |  |
| --- | --- | --- | --- |
| Запрос | Параметры | Возможное значение параметра | Возможный ответ |
| R\_SIGN\_UP | login  password | логин  пароль | R\_AUTHENTICATED,  R\_ALREADY\_EXISTS |
| R\_SIGN\_IN | login  password | логин  пароль | R\_AUTHENTICATED,  R\_USER\_NOT\_FOUND,  WRONG\_PASSWORD |
| R\_CHANGE | password | пароль | R\_OK |
| R\_SEND | peer  text | получатель или отправитель  текст | R\_OK |
| R\_BROADCAST | text | текст | R\_OK |
| R\_SELECT | sent  rcvd  user  number  order | SENT\_YES, SENT\_NO  RCVD\_YES, RCVD\_NO  кому или от кого  число сообщений  DIRECT, REVERSE | R\_MESSAGE, R\_END, |
| R\_NEXT |  |  | R\_MESSAGE, R\_END |
| R\_END |  |  |  |

Ответ R\_MESSAGE содержит следующие параметры, каждый из который также представлен текстовой строкой.

timestamp текстовое представление значения time\_t.

sent SENT\_YES или SENT\_NO

received RCVD\_YES или RCVD\_NO

sender отправитель

recipient получатель

text текст сообщения

Возможными ответами на каждый запрос являются также следующие ответы.

R\_DATABASE\_ERROR ошибка доступа к БД

R\_PROTOCOL\_ERROR ошибка протокола

Сценарий регистрации пользователя показан на рис. 5. Регистрация нового пользователя выполняется аналогичным образом.

client

server

R\_SIGN\_IN( login, password )

R\_AUTHENTICATED

Создание сессии пользователя

Создание сессии пользователя

Завершение сессии пользователя

R\_END

Рис. 5. Вход в сессию.

В случае неуспешной аутентификации от сервера передается сообщение об ошибке, в ответ на которое передается R\_END (см. рис. 6).

client

server

R\_SIGN\_IN( login, password )

R\_WRONG\_PASSWORD

R\_END

Рис. 6. Ошибка аутентификации.

Если сессия пользователя на стороне клиента уже создана, то при выполнении операции клиент сначала передает серверу учетные данные пользователя текущей сессии клиента. После успешной аутентификации сервер создает сессию пользователя и ждет запросов.

server

client

Отправка учетных данных

R\_SIGN\_IN( login, password )

Создание сессии пользователя

R\_AUTHENTICATED

R\_SEND ( message data)

Отправка данных сообщения

Выполнение операции отправки

R\_OK

R\_END

Завершение сессии пользователя

Рис. 7. Выполнение простой операции.

Выполнение простой операции сводится к отправке одного запроса и получению ответа, после чего клиент передает R\_END. Пример выполнения такой операции (отправка сообщения пользователем) показан на рис. 7.

При выполнении операции выборки сообщений (см. рис. 8) после выполнения аутентификации клиент отправляет серверу запрос R\_SELECT, содержащий параметры выборки. Если сообщения, соответствующие параметрам выборки отсутствуют, то сервер сразу отправляет клиенту R\_END.

server

client

Отправка учетных данных

R\_SIGN\_IN( login, password )

Создание сессии пользователя

R\_AUTHENTICATED

Отправка условия выборки

Выборка сообщений

R\_SELECT (condition, order)

R\_MESSAGE( data )

Представление результата

R\_NEXT

Представление результата

R\_MESSAGE( data )

…

R\_NEXT

R\_END

R\_END

Завершение сессии пользователя

Рис. 8. Выборка сообщений.

Если имеются сообщения, соответствующие параметрам выборки имеются, то сервер отправляет клиенту сообщение МЕSSAGE, содержащее данные первого сообщения в последовательности. В ответ на это клиент отправляет серверу R\_NEXT, в ответ на которое сервер передает данные очередного сообщения. После передачи последнего сообщения в последовательности сервер передает R\_END. При получении R\_END клиент также передает серверу R\_END, и диалог завершается, сервер завершает сессию клиента.

**Заключение. Обсуждение и вопросы**.

В заключение хотелось бы обсудить некоторые аспекты задачи и ее решения и также задать некоторые вопросы.

Поскольку работа сервера не должна зависеть от сетевых сбоев и некорректной работы клиента, необходимо ограничить время ожидания постепления пакета.

Если используется библиотека Winsock, то едитнственная возможность, которую я нашел, заключается в том, чтобы вызывать функцию WSAWaitForMultipleEvents. Время ожидания можно ограничить, но эта функция переводит весь процесс приложения в состояние ожидания. Вероятно, можно было бы в цикле периодически проверять битовые флаги событий. Но, откровенно говоря, я ожидал найти какую-то возможность передать библиотеке указатель на некоторую функцию обратного вызова (callback), которая вызывалась бы при наступлении события. Такой возможности и пока в Winsock не нашел. Возможно, плохо искал? Возможно, есть еще какие-то библиотеки для работы с сетью, обладающие подобной функциональностью?

Есть и еще один вопрос. Сейчас все сообщения передаются незашифрованными. Поскольку существует проблема передачи заранее неизвестного пароля при регистрации пользователя и при смене пароля, то единственным разумным путем защиты информации я считаю использование какого-то защищенного транспортного протокола. Лучше всего использовать некоторое стандартное решение (например, протокол TLS), но для этого необходима какая-то стандартная библиотека. Возможно, что даже и в этом случае будет разумным минимизировать число передач пароля и использовать стандартную четырехфазную схему аутентификации, позволяющую не передавать пароль. Можно ли порекомендовать какие-то стандартные решения этой задачи?

Ссылки.

1. <https://github.com/dmitryfedotov-56/chat>

2. <https://github.com/dmitryfedotov-56/hash>