МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №7

по дисциплине: Компьютерные сети
тема: «Протоколы POP3 и SMTP»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
Рубцов Константин Анатольевич

Белгород 2025 г.

Лабораторная работа №6
Протоколы POP3 и SMTP

Цель работы: изучить принципы и характеристику протоколов POP3 и SMTP и составить программу для приема/отправки электронной почты.

Краткие теоретические сведения

Протокол POP3

POP3 (англ. Post Office Protocol Version 3) - стандартный Интернет-протокол прикладного уровня, используемый клиентами электронной почты для извлечения электронного сообщения с удаленного сервера по TCP/IP-соединению.

В некоторых небольших узлах Интернет бывает непрактично поддерживать систему передачи сообщений (MTS - Message Transport System). Рабочая станция может не иметь достаточных ресурсов для обеспечения непрерывной работы SMTP-сервера [RFC-821]. Для “домашних ЭВМ” слишком дорого поддерживать связь с Интернет круглые сутки.

Но доступ к электронной почте необходим как для таких малых узлов, так и индивидуальных ЭВМ. Для решения этой проблемы разработан протокол POP3 (Post Office Protocol - Version 3, STD- 53. M. Rose, RFC-1939). Этот протокол обеспечивает доступ узла к базовому почтовому серверу.

POP3 не ставит целью предоставление широкого списка манипуляций с почтой. Почтовые сообщения принимаются почтовым сервером и сохраняются там, пока на рабочей станции клиента не будет запущено приложение POP3. Это приложение устанавливает соединение с сервером и забирает сообщения оттуда. Почтовые сообщения на сервере стираются. POP3 поддерживает простые требования «загрузи-и-удали» для доступа к удаленным почтовым ящикам. Хотя большая часть POP-клиентов предоставляют возможность оставить почту на сервере после загрузки, использующие POP клиенты обычно соединяются, извлекают все письма, сохраняют их на пользовательском компьютере как новые сообщения, удаляют их с сервера, после чего разъединяются.

Другие протоколы, в частности IMAP, предоставляют более полный и комплексный удаленный доступ к типичным операциям с почтовым ящиком. Многие клиенты электронной почты поддерживают как POP, так и IMAP; однако, гораздо меньше интернет-провайдеров поддерживают IMAP.

POP3-сервер прослушивает порт 110. Шифрование связи для POP3 запрашивается после запуска протокола, с помощью либо команды STLS (если она поддерживается), либо POP3S, которая соединяется с сервером используя TLS или SSL по TCP-порту 995.

Доступные сообщения клиента фиксируются при открытии почтового ящика POP-сессией и определяются количеством сообщений для сессии, или, по желанию, с помощью уникального идентификатора, присваиваемого сообщению POP-сервером. Этот уникальный идентификатор является постоянным и уникальным для почтового ящика и позволяет клиенту получить доступ к одному и тому же сообщению в разных POP-сессиях. Почта извлекается и помечается для удаления с помощью номера сообщения. При выходе клиента из сессии помеченные сообщения удаляются из почтового ящика.

Обычно POP3-сервис устанавливается на 110-й TCP -порт сервера, который будет находиться в режиме ожидания входящего соединения. Когда клиент хочет воспользоваться POP3-сервисом, он просто устанавливает TCP-соединение с портом 110 этого хоста. После установления соединения сервис POP3 отправляет подсоединившемуся клиенту приветственное сообщение. После этого клиент и сервер начинают обмен командами и данными. По окончании обмена POP3-канал закрывается.

Команды POP3 состоят из ключевых слов, состоящих из ASCIIсимволов, и одним или несколькими параметрами, отделяемыми друг от друга символом "пробела" - . Все команды заканчиваются символами "возврата каретки" и "перевода строки" - . Длина ключевых слов не превышает четырех символов, а каждого из аргументов может быть до 40 символов.

Ответы POP3-сервера на команды состоят из строки статусиндикатора, ключевого слова, строки дополнительной информации и символов завершения строки. Длина строки ответа может достигать 512 символов. Строка статус-индикатора принимает два значения: положительное ("+OK") и отрицательное ("-ERR"). Любой сервер POP3 обязан отправлять строки статус-индикатора в верхнем регистре, тогда как другие команды и данные могут приниматься или отправляться как в нижнем, так и в верхнем регистрах. Ответы POP3-сервера на отдельные команды могут составлять несколько строк. В этом случае строки разделены символами . Последнюю строку информационной группы завершает строка, состоящая из символа "." (код — 046) и , т. е. последовательность "CRLF.CRLF".

POP3-сессия состоит из нескольких частей. Как только открывается TCP-соединение и POP3-сервер отправляет приветствие, сессия должна быть зарегистрирована - состояние аутентификации (AUTHORIZATION state). Клиент должен зарегистрироваться в POP3сервере, т. е. ввести свой идентификатор и пароль.

После этого сервер предоставляет клиенту его почтовый ящик и открывает для данного клиента транзакцию - состояние начала транзакции обмена (TRANSACTION state). На этой стадии клиент может считать и удалить почту своего почтового ящика. После того как клиент заканчивает работу (передает команду QUIT), сессия переходит в состояние UPDATE - завершение транзакции. В этом состоянии POP3-сервер закрывает транзакцию данного клиента (на языке баз данных - операция COMMIT) и закрывает TCP-соединение. В случае получения неизвестной, неиспользуемой или неправильной команды, POP3-сервер должен ответить отрицательным состоянием индикатора.

POP3-сервер может использовать в своей работе таймер контроля времени соединения. Этот таймер отсчитывает время "бездействия" ("idle") клиента в сессии от последней переданной команды. Если время сессии истекло, сервер закрывает TCP-соединение, не переходя в состояние UPDATE (иными словами, откатывает транзакцию или на языке баз данных — выполняет ROLLBACK). POP3-сервер может 64 обслуживать группу клиентов, которые, возможно, присоединяются по коммутируемой линии, и, следовательно, необходимо иметь средство автоматического регулирования времени соединения. По спецификации POP3-таймер контроля состояния "idle" должен быть установлен на промежуток времени не менее 10 минут.

Команды протокола POP3:

- USER - идентифицирует пользователя с указанным именем;
- PASS - указывает пароль для пары клиент-сервер;
- QUIT - закрывает TCP-соединение;
- STAT - сервер возвращает количество сообщений в почтовом ящике плюс размер почтового ящика;
- LIST - сервер возвращает идентификаторы сообщений вместе с размерами сообщений;
- RETR - извлекает сообщение из почтового ящика;
- DELE - отмечает сообщение для удаления;
- NOOP - Сервер возвращает положительный ответ, но не совершает никаких действий;

- LAST - Сервер возвращает наибольший номер сообщения из тех, к которым ранее уже обращались;
- RSET - Отменяет удаление сообщения, отмеченного ранее командой DELE.

Протокол SMTP

SMTP (Simple Mail Transfer Protocol) - широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP. SMTP впервые был описан в RFC 821 (1982 год); последнее обновление в RFC 5321 (2008) включает масштабируемое расширение - ESMTP (Extended SMTP). В настоящее время под «протоколом SMTP», как правило, подразумевают и его расширения. Протокол SMTP предназначен для передачи исходящей почты, используя для этого порт TCP 25.

Упрощенно схема взаимодействия представлена на рис. 7.1 (объемными стрелками показано направление движения почтовых сообщений).

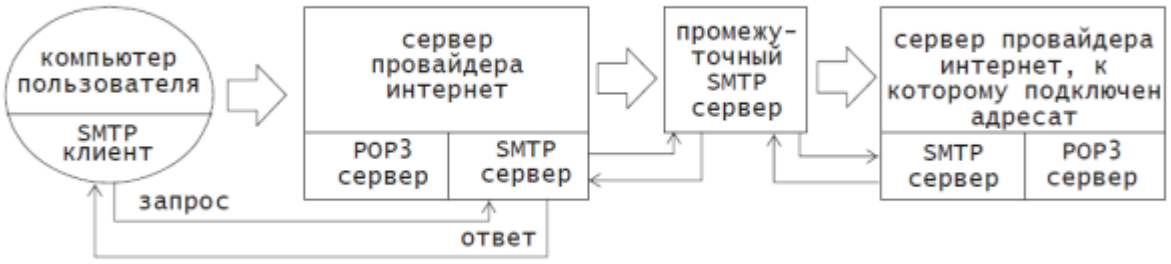


Рис. 7.1. Отправка почты по протоколу SMTP

Со стороны пользователя обычно одна и та же программа выступает в роли и POP3 клиента, и SMTP клиента отправителя. Наиболее распространенными на данный момент являются MS Outlook, The Bat, Netscape Messenger, Eudora, Pegasus mail, Mutt, Pine и др. При нажатии в них на кнопочку "отправить" происходит формирование очереди сообщений, и установление двустороннего сеанса общения с SMTP сервером провайдера. На схеме у пользователя есть клиентское ПО, а у провайдера – серверная часть приложения. На самом деле это немного не так. Протокол SMTP делает возможным смену сторон даже в ходе одного сеанса. Условно принято считать клиентом ту сторону, которая начинает взаимодействие и хочет отослать почту, а сервером ту, что принимает запросы. После того, как клиент посылает серверу несколько служебных команд и получает положительные ответы на них, он отправляет SMTP серверу собственно тело сообщения. SMTP сервер получает сообщение, вносит в него дополнительные заголовки, указывающие на то, что он обработал данное послание, устанавливает связь со следующим SMTP сервером по пути следования письма. Общение между любыми SMTP серверами происходит по той же схеме. Инициирует переговоры клиент, сервер на них отвечает, а затем получает корреспонденцию и "ставит штампик" в теле письма (в его заголовочной части). Все это очень напоминает обычную бумажную почту, где работу по сортировке и отправке почты выполняют люди.

Если на каком-нибудь этапе передачи SMTP клиент обнаружит невозможность подключиться к следующему серверу (например, компьютер отправили на профилактику или аппаратура связи вышла из строя), он будет пытаться отправить сообщение через некоторое время – 1 час, 4 часа, день и т.д. до 4 суток в общем случае. Причем временные отрезки между попытками, как правило, зависят от настроек программы-пересыльщика почты. Одновременно, такой сервер должен уведомить отправителя сообщения о невозможности доставить почту,

послав ему стандартное письмо "Failed delivery" (доставка невозможна) и рассказав о графике дальнейших попыток по продвижению исходного сообщения. Если канал связи не восстановится за указанный большой промежуток времени (например, 4 дня), посланная информация будет считаться утерянной.

Как только почта достигнет конечного пункта (SMTP сервера адресата сообщения), она будет сложена в почтовый ящик абонента, который всегда сможет в удобное для него время изъять ее по протоколам POP3 или IMAP, в зависимости от того, какой из них поддерживается провайдером.

Анализируя заголовок письма, можно узнать какими путями оно путешествовало, как долго длился сам путь, как называлась почтовая программа отправителя и многое другое. Получить эту информацию можно в "Свойствах письма", кликнув правой кнопкой мыши на самом письме в MS Outlook, нажав Ctrl+Shift+H в The Bat или совершить нечто подобное в других почтовых клиентах.

Рассмотрим клиент-серверное взаимодействие по протоколу SMTP. Программа пользователя, выбрав для связи соответствующий почтовый сервер, устанавливает с ним контакт на транспортном и сеансовом уровнях эталонной модели взаимодействия открытых систем OSI/RM (в терминах TCP/IP (transmission control protocol / internet protocol) это – TCP уровень). Взаимодействие на более низких уровнях (канальном, сетевом) происходит прозрачно для обеих сторон. Протокол SMTP – протокол прикладного уровня и базируется поверх TCP. В его рамках не оговаривается ни размер сегментов данных, ни правила квитирования, ни отслеживание ошибок, возникающих при передаче информации.

По уже установленному соединению клиентское ПО передает команды SMTP серверу, ожидая тут же получить ответы. В арсенал SMTP клиента, равно как и сервера, входит около 10 команд, но, воспользовавшись только пятью из них, уже можно легально послать почтовое сообщение. Это HELO, MAIL, RCPT, DATA, QUIT. Их использование подразумевается именно в такой последовательности. HELO предназначена для идентификации отправителя, MAIL указывает адрес отправителя, RCPT – адрес назначения. После команды DATA и ответа на нее, клиент посылает серверу тело сообщения, которое должно заканчиваться строкой, содержащей лишь одну точку.

Непосредственно после установления соединения сервер выдает строчку с кодом ответа 220. В ответ на нее клиент может инициировать сеанс связи по протоколу SMTP, послав команду HELO и указав у нее в аргументах имя своего компьютера. По принятии команды HELO сервер обязан сделать запрос в DNS и, если это возможно, по IP адресу определить доменное имя компьютера клиента. (IP адрес уже известен на момент установления соединения по TCP протоколу).

Далее в команде "MAIL FROM:" клиент сообщает обратный адрес отправителя, который проверяется обычно только на корректность. После слов "RCPT TO:" следует набрать адрес электронной почты абонента на данном сервере. Клиент отсылает команду DATA и ждет приглашения начать пересылку тела письма (код 354).

Сообщение может быть достаточно длинным, но обязательно должно заканчиваться строкой, в которой есть одна-единственная точка. Это служит сигналом SMTP серверу о том, что тело письма закончилось. Он присваивает этому письму определенный идентификатор, и ждет команды QUIT, после чего сеанс считается завершенным.

Если клиент посылает сообщение, у которого в заголовочной части в поле CC указаны несколько e-mail адресов, первый по пути следования SMTP сервер должен будет в общем случае установить сеанс продвижения почты с каждым из серверов данного списка и отослать точную копию письма каждому. В случае использования поля BCC клиент, формирующий сообщение, уничтожит запись BCC в теле сообщения и по количеству адресатов отошлет первому SMTP серверу команду "RCPT TO:" каждый раз с новым адресом в качестве аргумента. Таким образом, сервер получит указание разослать почту по многим адресатам. Причем, в этом случае получатели писем ничего не будут знать друг о друге, т.к. рассылка осуществляется посредством команд SMTP протокола.

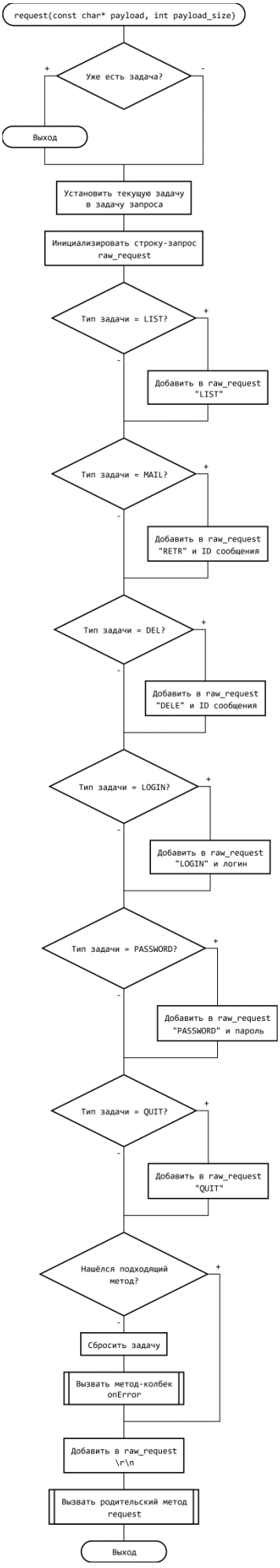
Используемые функции

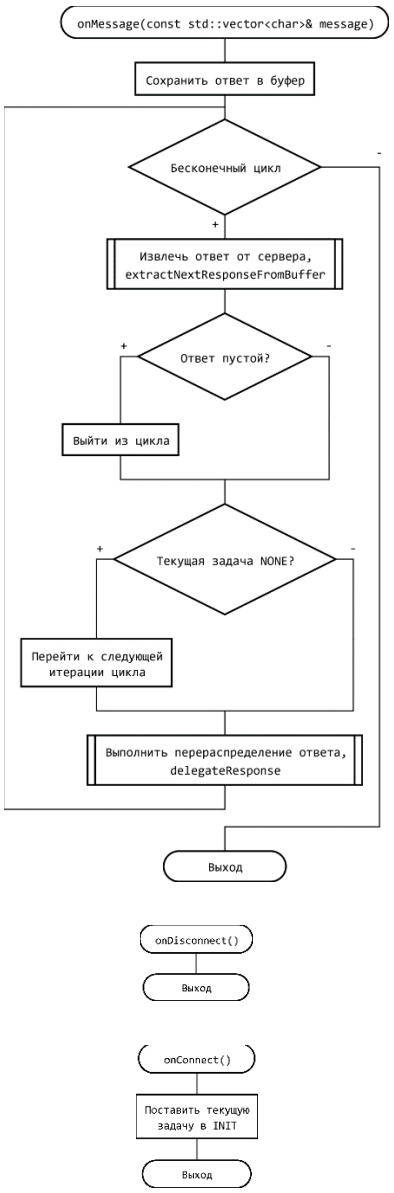
- Перевод сокета в состояние “прослушивания” (для TCP) осуществляется функцией listen (SOCKET s, int backlog), где s – дескриптор сокета; backlog – максимальный размер очереди входящих сообщений на соединение. Используется сервером, чтобы информировать ОС, что он ожидает (“слушает”) запросы связи на данном сокете. Без этой функции всякое требование связи с сокетом будет отвергнуто.
- Функция connect (SOCKET s, const struct sockaddr FAR* name, int namelen) нужна для соединения с сокетом, находящимся в состоянии “прослушивания” (для TCP). Она ипользуется процессомклиентом для установления связи с сервером. В случае успешного установления соединения connect возвращает 0, иначе SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Функция accept (SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen) служит для подтверждения запроса на соединение (для TCP). Функция используется для принятия связи на сокет. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется.
- В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции getsockname (SOCKET s, struct sockaddr FAR* name, int FAR* namelen). Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для передачи данных по протоколу UDP используется функция sendto (SOCKET s, const char FAR * buf, int len, int flags, const struct sockaddr FAR * to, int tolen). Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для передачи данных по протоколу TCP используется функция send (SOCKET s, const char FAR * buf, int len, int flags), где s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена

успешно, возвращает количество переданных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

- Для приема данных по протоколу UDP используется функция recvfrom (SOCKET s, char FAR* buf, int len, int flags, struct sockaddr FAR* from, int FAR* fromlen). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
- Для приема данных по протоколу TCP используется функция recv (SOCKET s, char FAR* buf, int len, int flags). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError

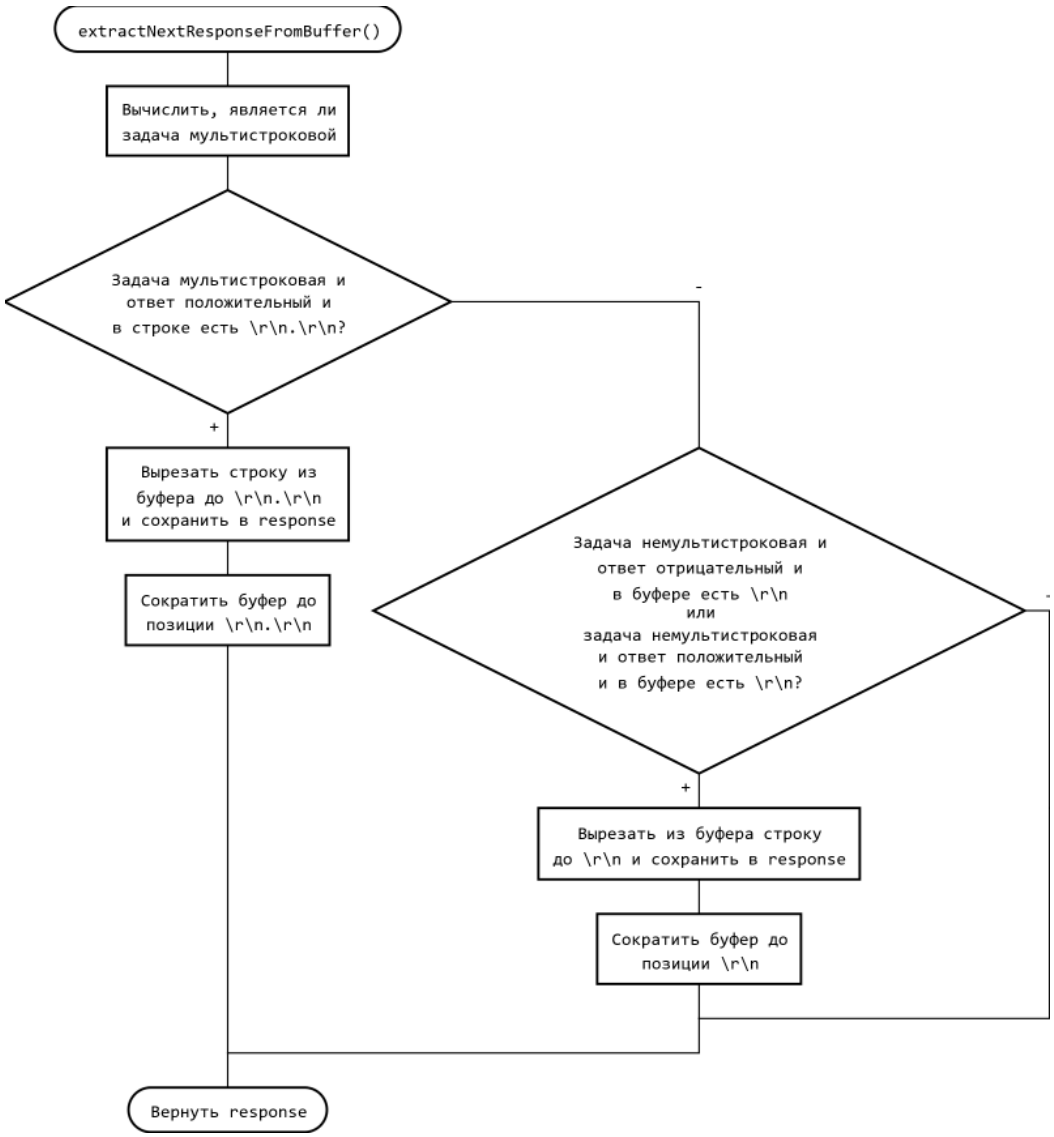
Разработка программы. Блок-схемы программы. POP3Client

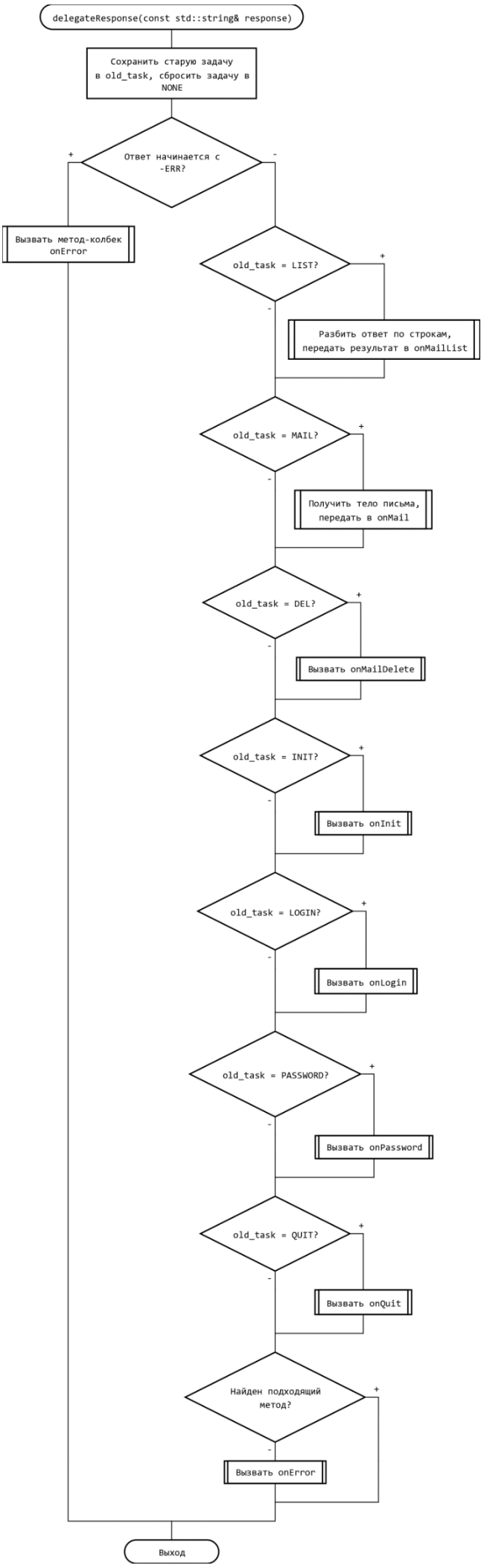



onMessage(const std::vector<char>& message)
Сохранить ответ в буфер
Бесконечный цикл
-
+
Извлечь ответ от сервера, extractNextResponseFromBuffer
Ответ пустой?
+
-
Выйти из цикла
Текущая задача NONE?
+
-
Перейти к следующей итерации цикла
Выполнить перераспределение ответа, delegateResponse
Выход

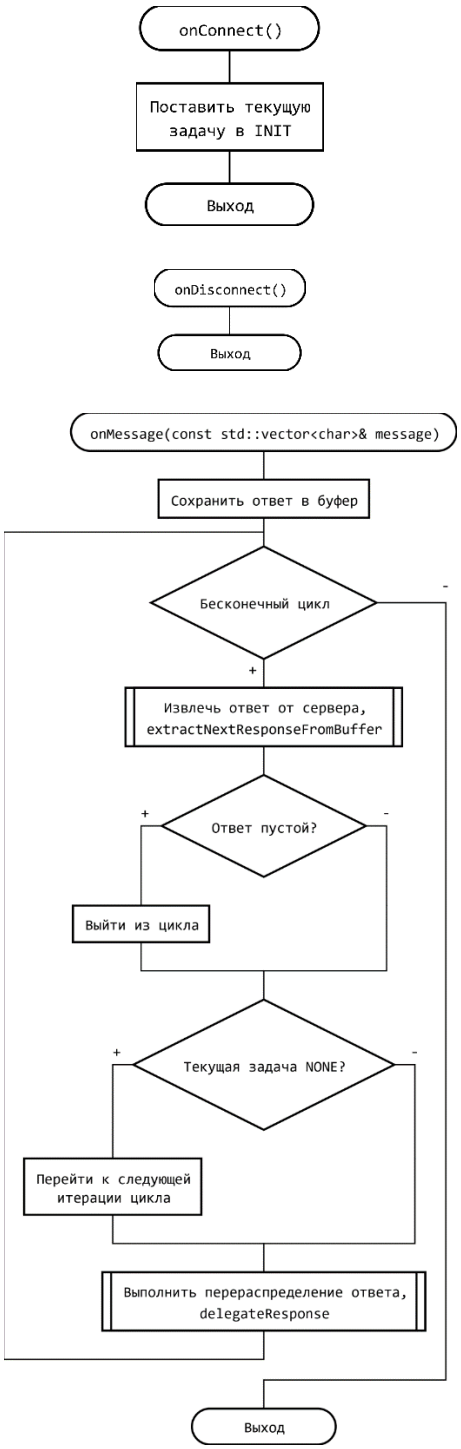
onDisconnect()
Выход

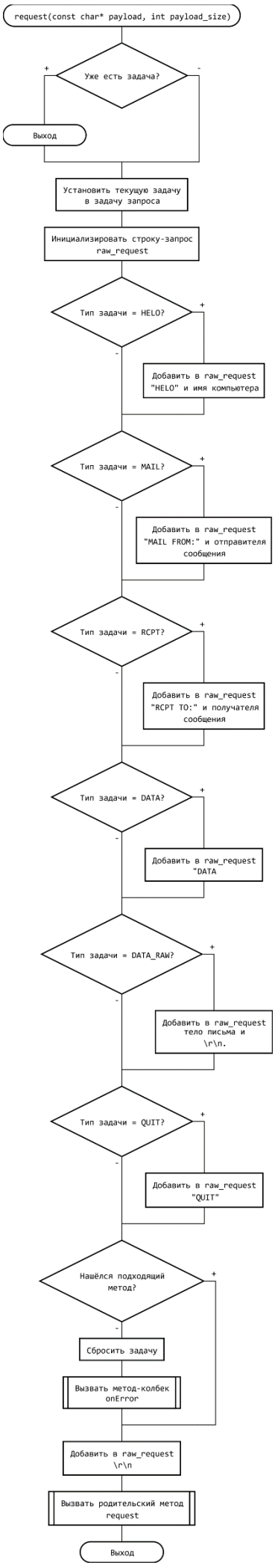
onConnect()
Поставить текущую задачу в INIT
Выход

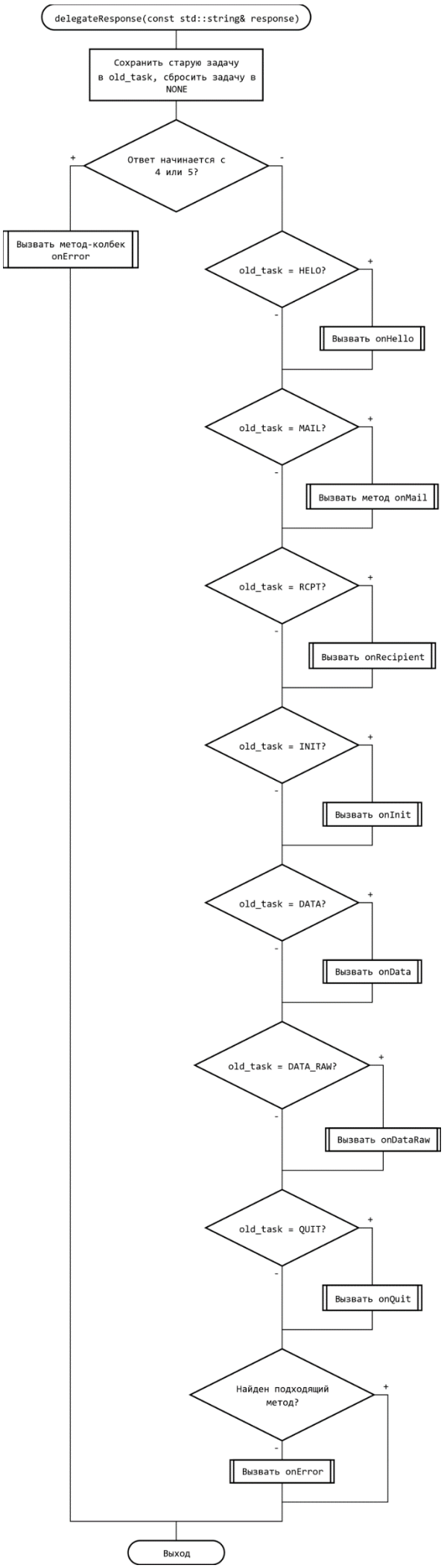

extractNextResponseFromBuffer()
Вычислить, является ли задача мультистроковой
Задача мультистроковая и ответ положительный и в строке есть \r\n.\r\n?
-
+
Вырезать строку из буфера до \r\n.\r\n и сохранить в response
Сократить буфер до позиции \r\n.\r\n
Задача немультистроковая и ответ отрицательный и в буфере есть \r\n или задача немультистроковая и ответ положительный и в буфере есть \r\n?
-
+
Вырезать из буфера строку до \r\n и сохранить в response
Сократить буфер до позиции \r\n
Вернуть response

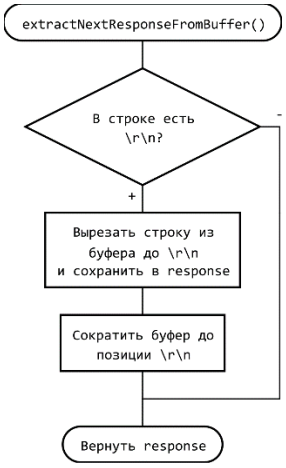

delegateResponse(const std::string& response)
Сохранить старую задачу в old_task, сбросить задачу в NONE
Ответ начинается с -ERR?
+
-
Вызвать метод-колбек onError
old_task = LIST?
+
-
Разбить ответ по строкам, передать результат в onMailList
old_task = MAIL?
+
-
Получить тело письма, передать в onMail
old_task = DEL?
+
-
Вызвать onMailDelete
old_task = INIT?
+
-
Вызвать onInit
old_task = LOGIN?
+
-
Вызвать onLogin
old_task = PASSWORD?
+
-
Вызвать onPassword
old_task = QUIT?
+
-
Вызвать onQuit
Найден подходящий метод?
+
-
Вызвать onError
Выход

SMTPClient

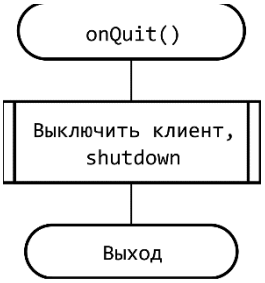

onConnect()
Поставить текущую задачу в INIT
Выход
onDisconnect()
Выход
onMessage(const std::vector<char>& message)
Сохранить ответ в буфер
Бесконечный цикл
+
-
Извлечь ответ от сервера, extractNextResponseFromBuffer
Ответ пустой?
+
-
Выйти из цикла
Текущая задача NONE?
+
-
Перейти к следующей итерации цикла
Выполнить перераспределение ответа, delegateResponse
Выход

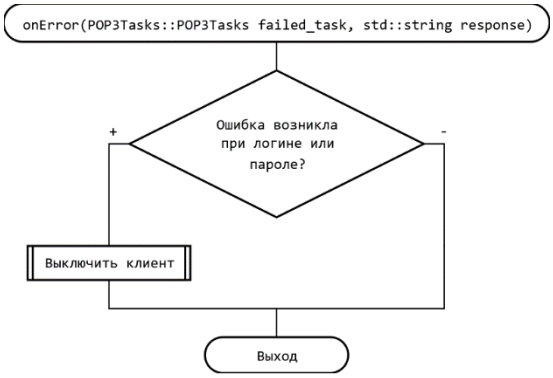

request(const char* payload, int payload_size)
Уже есть задача?
+
-
Выход
Установить текущую задачу в задачу запроса
Инициализировать строку-запрос raw_request
Тип задачи = HELO?
+
-
Добавить в raw_request "HELO" и имя компьютера
Тип задачи = MAIL?
+
-
Добавить в raw_request "MAIL FROM:" и отправителя сообщения
Тип задачи = RCPT?
+
-
Добавить в raw_request "RCPT TO:" и получателя сообщения
Тип задачи = DATA?
+
-
Добавить в raw_request "DATA
Тип задачи = DATA_RAW?
+
-
Добавить в raw_request тело письма и \r\n.
Тип задачи = QUIT?
+
-
Добавить в raw_request "QUIT"
Нашёлся подходящий метод?
+
-
Сбросить задачу
Вызвать метод-колбек onError
Добавить в raw_request \r\n
Вызвать родительский метод request
Выход

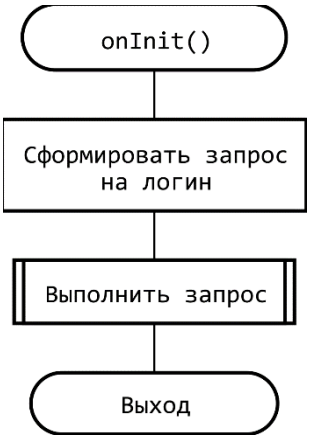

delegateResponse(const std::string& response)
Сохранить старую задачу в old_task, сбросить задачу в NONE
Ответ начинается с 4 или 5?
+
-
Вызвать метод-колбек onError
old_task = HELO?
+
-
Вызвать onHello
old_task = MAIL?
+
-
Вызвать метод onMail
old_task = RCPT?
+
-
Вызвать onRecipient
old_task = INIT?
+
-
Вызвать onInit
old_task = DATA?
+
-
Вызвать onData
old_task = DATA_RAW?
+
-
Вызвать onDataRaw
old_task = QUIT?
+
-
Вызвать onQuit
Найден подходящий метод?
+
-
Вызвать onError
Выход

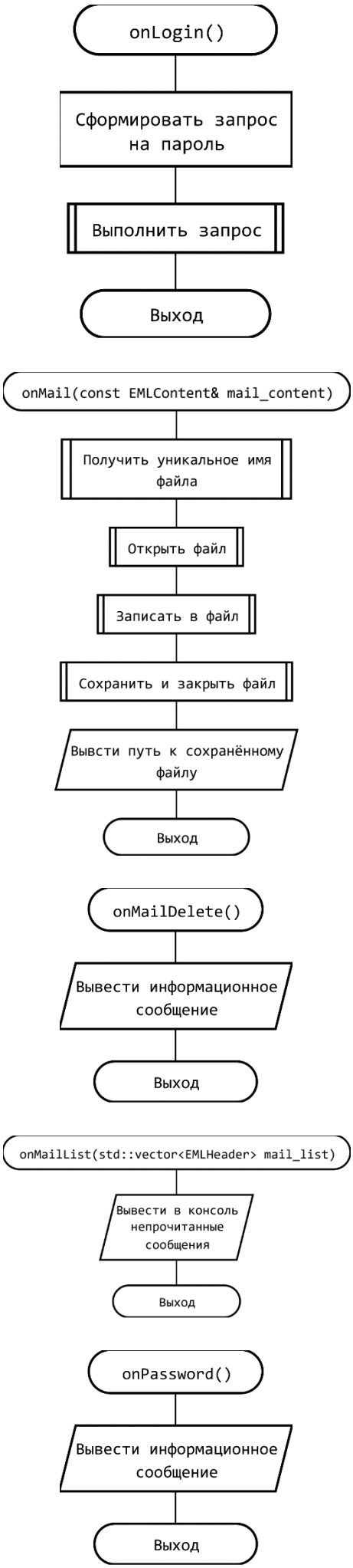


MyPOP3Client

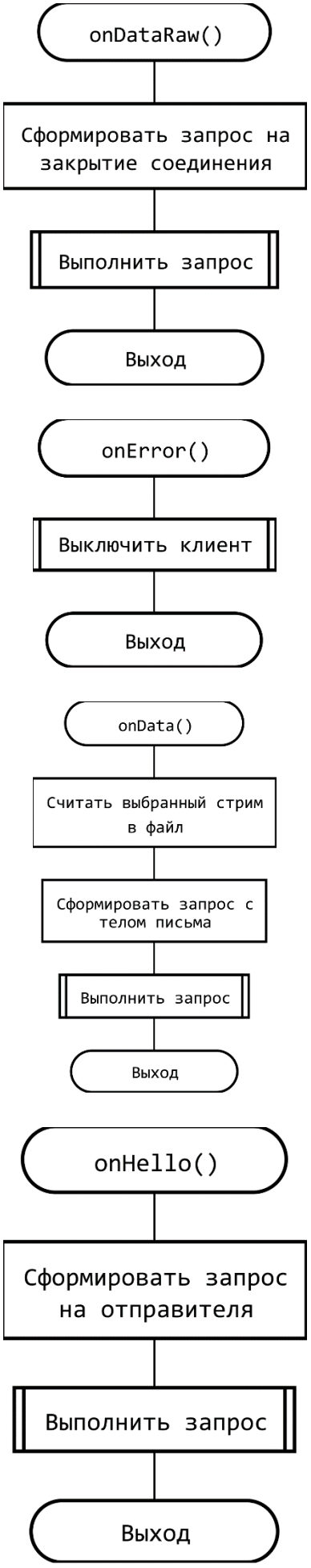


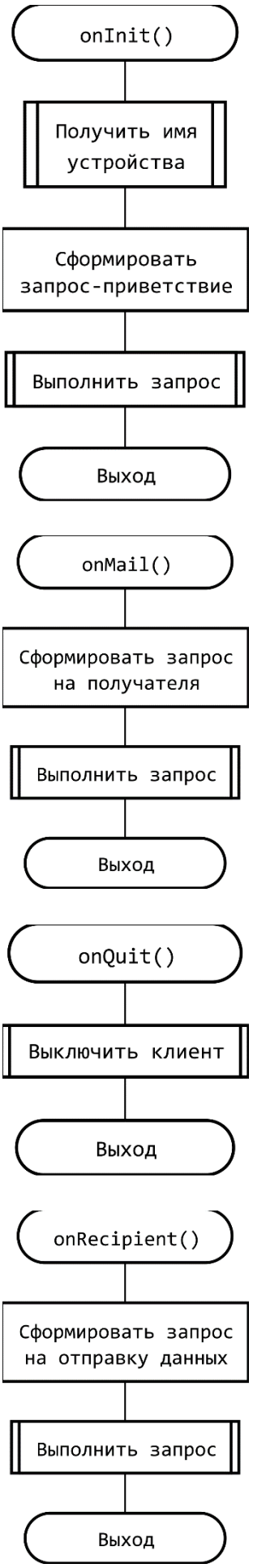




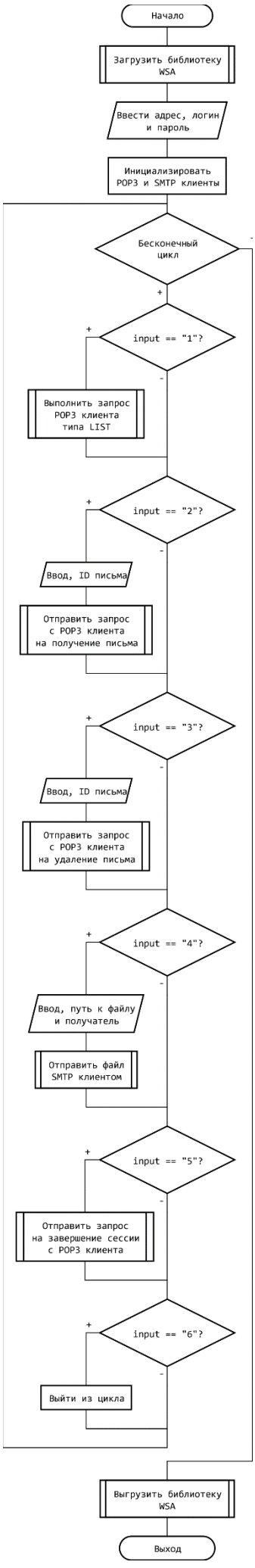

onLogin()
Сформировать запрос на пароль
Выполнить запрос
Выход
onMail(const EMLContent& mail_content)
Получить уникальное имя файла
Открыть файл
Записать в файл
Сохранить и закрыть файл
Вывсти путь к сохранённому файлу
Выход
onMailDelete()
Вывести информационное сообщение
Выход
onMailList(std::vector<EMLHeader> mail_list)
Вывести в консоль непрочитанные сообщения
Выход
onPassword()
Вывести информационное сообщение
Выход

MySMTPClient




onInit()
Получить имя устройства
Сформировать запрос-приветствие
Выполнить запрос
Выход
onMail()
Сформировать запрос на получателя
Выполнить запрос
Выход
onQuit()
Выключить клиент
Выход
onRecipient()
Сформировать запрос на отправку данных
Выполнить запрос
Выход

Основная программа


Начало
Загрузить библиотеку WSA
Ввести адрес, логин и пароль
Инициализировать POP3 и SMTP клиенты
Бесконечный цикл
input == "1"?
Выполнить запрос POP3 клиента типа LIST
input == "2"?
Ввод, ID письма
Отправить запрос с POP3 клиента на получение письма
input == "3"?
Ввод, ID письма
Отправить запрос с POP3 клиента на удаление письма
input == "4"?
Ввод, путь к файлу и получатель
Отправить файл SMTP клиентом
input == "5"?
Отправить запрос на завершение сессии с POP3 клиента
input == "6"?
Выйти из цикла
Выгрузить библиотеку WSA
Выход

Анализ функционирования программ

Программа представляет из себя почтовый клиент, работающий по протоколам POP3/SMTP, что позволяет принимать и отправлять сообщения клиенту одновременно. Программа фактически состоит из двух независимых клиентов: POP3 и SMTP. Первый занимается получением сообщений, второй занимается отправкой сообщений. Приложение располагает следующими возможностями:

- Получить список непрочитанных сообщений

  После получения списка сообщений выводится список сообщений, состоящих из ID письма и его размера. Пример:

```
1 mails are not read
  ID: 1 Size: 196 bytes
```

- Получить сообщение

  Сохраняет письмо в файл .rmf формата. Также можно отметить, что к оригинальному отправленному файлу добавляется заголовок с отправителем. Пример полученного сообщения:

```
Return-Path: test2@local.ru
Received: from SHELEZYAKA (Shelezyaka [192.168.1.175])
        by SHELEZYAKA with ESMTP
        ; Mon, 12 May 2025 23:34:31 +0300
Message-ID: <E90685C1-E58B-435C-B680-43ABB1206278@SHELEZYAKA>
Hello World!
```

- Удалить сообщение

  Удаление осуществляется при помощи POP3 клиента с указанием ID письма

- Отправить сообщение

  Сообщение должно быть в файле, необходимо указать путь к нему, а также почтовый адрес получателя. Пример взаимодействия:

```
Input recipient: test@local.ru
Input file path: C:\Users\vladi\Downloads\mail.txt
Creating socket
Creating socket bind addr
Binding socket
Starting client
Enter 1 to get mail list
Enter 2 to retrieve mail
Enter 3 to delete mail
Enter 4 to send mail
Enter 5 to update POP3-server state
Enter 6 to exit program

Starting TCP client
My SMTP client: An initial connection established. Sending hello request
My SMTP client: Hello request succeeded. Sending mail from request
My SMTP client: Mail from request succeeded. Sending recipient request
My SMTP client: Recipient to request succeeded. Sending data request
My SMTP client: Data request succeeded. Sending a data
My SMTP client: Data sent. Now we can quit
My SMTP client: SMTP client quitted and says BB!
Stopping client
Closing client socket
```

- Сохранить состояние POP3 сессии

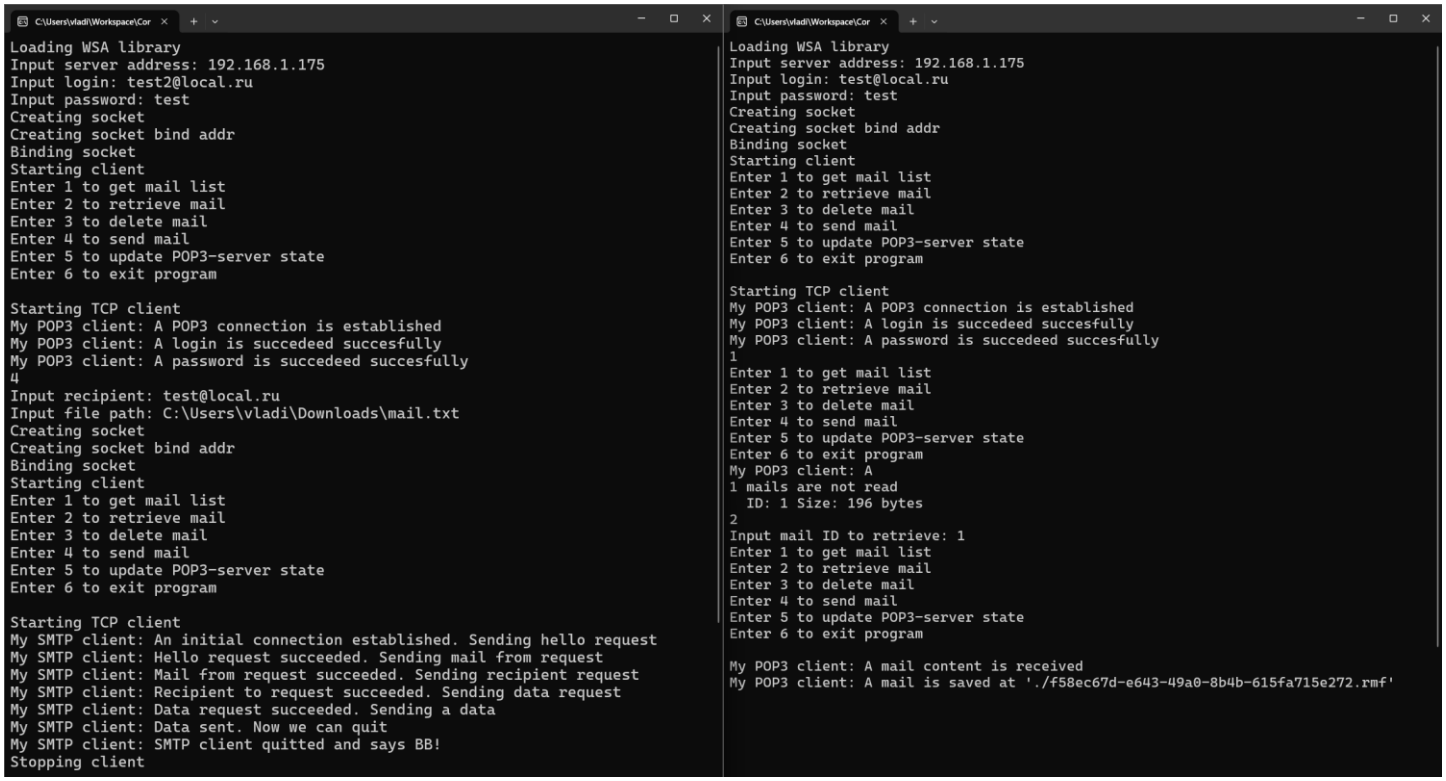
  После сохранения состояния из программы можно безопасно выходить, сессия сохранилась.

Данный функционал достаточен для выполнения функции почтового клиента.

Вывод: в ходе лабораторной изучили принципы и характеристику протоколов POP3 и SMTP и составить программу для приема/отправки электронной почты, которая способна выполнять операции: отправка, удаление, получение и так далее. Данного функционала достаточно для реализации почтового клиента.

Текст программ. Скриншоты программ.

Ссылка на репозиторий с кодом: https://github.com/IAmProgrammist/comp_net/tree/lab7




```cpp
// Содержит базовые классы для создания POP3-клиента

#pragma once

#include <queue>
#include <map>
#include <webstur/ip/tcp/tcpclient.h>
#include <sstream>

// Структура, описывающая идентификатор и размер письма
struct EMLHeader {
    std::size_t id;
    std::size_t size = 0;
};

// Структура, содержащая данные полученного письма
struct EMLContent {
    std::string data;
};

namespace POP3Tasks {
    // Структура, описывающая типы задач, выполняемые POP3-клиентом
    enum POP3Tasks
    {
        // У клиента нет задачи, все входящие сообщения будут проигнорированы
        NONE,
        // Список непрочитанных сообщений
```

```
        LIST,
        // Получение конкретного сообщения
        MAIL,
        // Удалить сообщение
        DEL,
        // Первое сообщение от сервера +OK
        INIT,
        // Ввод логина от почтового ящика
        LOGIN,
        // Ввод пароля от почтового ящика
        PASSWORD,
        // Выход из POP3-сессии
        QUIT
    };
}

// Описывает задачу к выполнению и её аргументы
struct POP3Request {
    POP3Request() {};
    ~POP3Request() {};
    // Тип задачи
    POP3Tasks::POP3Tasks task;
    union Arguments {
        Arguments() {};
        ~Arguments() {};

        // Идентификатор сообщения, используется в задачах DEL, MAIL
        std::size_t id;
        // Логин, используется в задаче LOGIN
        const char* login;
        // Пароль, используется в задаче PASSWORD
        const char* password;
    } arguments;
};

// Абстрактный POP3-клиент, исользующий незащищённое соединение.
// Отправляет команды и принимает ответы от сервера.
// Вызывает соответствующий метод для каждого ответа.
class DLLEXPORT POP3Client : public TCPClient {
private:
    // Текущая активная задача
    POP3Tasks::POP3Tasks current_task;
    // Буфер накопления ответов с сервера
    std::string buffer;
    // Извлекает из буфера следующий ответ от сервера
    std::string extractNextResponseFromBuffer();
    // Распарсивает ответ от сервера, распределяет в соответствующий
    // метод в зависимости от текущей задачи, сбрасывает текущую задачу.
    void delegateResponse(const std::string& response);
protected:
    // Определяет, является ли команда мультистроковой.
    static std::map<POP3Tasks::POP3Tasks, bool> is_multi_line;
    // Метод, вызываемый при установлении соединения с сервером
    void onConnect();
    // Метод, вызываемый при разрыве соединения с сервером
    // Переданный сокет уже не является действительным,
    // однако передаётся для отладочной информации
    void onDisconnect();
    // Метод, вызываемый при отправке получении сообщения от сервера
    void onMessage(const std::vector<char>& message);
public:
    // Конструирует POP3-клиент, порт по умолчанию - 110 (незащищённое соединение)
    POP3Client(std::string address, int port = 110);
    // Метод для отправки команды серверу.
    // В параметр payload передаётся указатель на структуру POP3Request,
    // в payload_size - размер структуры
    void request(const char* payload, int payload_size);
```

```
    // Метод, вызываемый при получении списка доступных писем
    virtual void onMailList(std::vector<EMLHeader> mail_list) = 0;
    // Метод, вызываемый при получении письма
    virtual void onMail(const EMLContent &mail_content) = 0;
    // Метод, вызываемый после удаления письма
    virtual void onMailDelete() = 0;
    // Метод, вызываемый после успешного подтверждения
    // соединения с сервером
    virtual void onInit() = 0;
    // Метод, вызываемый после успешного ввода логина
    virtual void onLogin() = 0;
    // Метод, вызываемый после успешного ввода пароля
    virtual void onPassword() = 0;
    // Метод, вызываемый после получения ответа на выход из POP3-сессии
    virtual void onQuit() = 0;
    // Метод, вызываемый если не удалось обработать ответ от сервера
    virtual void onError(POP3Tasks::POP3Tasks failed_task, std::string response) = 0;
};
```

```
#include "pch.h"
#include <assert.h>
#include <webstur/ip/tcp/pop3client.h>
#include <webstur/utils.h>

std::map<POP3Tasks::POP3Tasks, bool> POP3Client::is_multi_line = {
    { POP3Tasks::LIST, true },
    { POP3Tasks::MAIL, true },
    { POP3Tasks::DEL, false },
    { POP3Tasks::INIT, false },
    { POP3Tasks::NONE, false },
    { POP3Tasks::LOGIN, false },
    { POP3Tasks::PASSWORD, false },
    { POP3Tasks::QUIT, false },
};

void POP3Client::onConnect() {
    // Ожидаем, что первое полученное сообщение будет сообщение инициализации
    this->current_task = POP3Tasks::INIT;
}

void POP3Client::onDisconnect() {
    // Ничего не делаем
}

std::string POP3Client::extractNextResponseFromBuffer() {
    // Определить, является ли текущий запрос мультистроковым
    auto is_current_task_multi_line_it = this->is_multi_line.find(this->current_task);
    assert(is_current_task_multi_line_it != this->is_multi_line.end());
    auto is_current_task_multi_line = is_current_task_multi_line_it->second;

    std::size_t break_line_pos;
    std::string response;
    // Задача мультистроковая и ответ положительный
    if ((break_line_pos = this->buffer.find("\r\n.\r\n")) != std::string::npos &&
        is_current_task_multi_line &&
        this->buffer.starts_with("+OK")) {
        response = this->buffer.substr(0, break_line_pos);
        this->buffer = this->buffer.substr(break_line_pos + 5);
        // Задача немультистроковая или мультистроковая, но ответ отрицательный
    }
    else {
        break_line_pos = this->buffer.find("\r\n");
        if (!is_current_task_multi_line && break_line_pos != std::string::npos ||
            is_current_task_multi_line && !this->buffer.starts_with("+OK") &&
            break_line_pos != std::string::npos) {
```

```
            response = this->buffer.substr(0, break_line_pos);
            this->buffer = this->buffer.substr(break_line_pos + 2);
        }
    }

    return response;
}

void POP3Client::delegateResponse(const std::string& response) {
    // Сбросить текущую задачу
    auto old_task = this->current_task;
    this->current_task = POP3Tasks::NONE;

    // Если ответ начинается с ошибки, то вызвать метод-обработчик ошибки
    if (response.starts_with("-ERR")) {
        this->onError(old_task, response.substr(5));
        return;
    }
    else {
        switch (old_task) {
        case POP3Tasks::LIST:
        {
            auto splitted_data = split(response, "\r\n");
            std::vector<EMLHeader> headers;
            for (int i = 1; i < splitted_data.size(); i++) {
                auto splitted_header = split(splitted_data[i], " ");
                EMLHeader header = {
                    std::atoi(splitted_header[0].c_str()),
                    std::atoi(splitted_header[1].c_str())
                };
                headers.push_back(header);
            }
            this->onMailList(headers);
            break;
        }
        case POP3Tasks::MAIL:
        {
            auto break_line_pos = response.find("\r\n");
            EMLContent dat = {
                response.substr(break_line_pos + 2)
            };
            this->onMail(dat);
            break;
        }
        case POP3Tasks::DEL:
            this->onMailDelete();
            break;
        case POP3Tasks::INIT:
            this->onInit();
            break;
        case POP3Tasks::LOGIN:
            this->onLogin();
            break;
        case POP3Tasks::PASSWORD:
            this->onPassword();
            break;
        case POP3Tasks::QUIT:
            this->onQuit();
            break;
        default:
            this->onError(old_task, "Task not implemented");
        }
    }
}

void POP3Client::onMessage(const std::vector<char>& message) {
    // Сохранить ответ в буфер
```

```
    this->buffer += std::string(message.begin(), message.end());

    while (true) {
        std::string response = extractNextResponseFromBuffer();

        // Если ответ пустой, выходим из метода
        if (response.empty())
            return;

        // Если текущая задача не задана, также выходим
        if (this->current_task == POP3Tasks::NONE)
            continue;

        this->delegateResponse(response);
    }
}

void POP3Client::request(const char* payload, int payload_size) {
    if (this->current_task != POP3Tasks::NONE)
        return;
    assert(payload_size >= sizeof(POP3Request));
    auto request = ((POP3Request*) payload);
    std::stringstream raw_request;
    this->current_task = request->task;

    switch (request->task) {
    case POP3Tasks::LIST:
    {
        raw_request << "LIST";
        break;
    }
    case POP3Tasks::MAIL:
    {
        raw_request << "RETR " << request->arguments.id;
        break;
    }
    case POP3Tasks::DEL:
        raw_request << "DELE " << request->arguments.id;
        break;
    case POP3Tasks::LOGIN:
        raw_request << "USER " << std::string(request->arguments.login,
            request->arguments.login + strlen(request->arguments.login));
        break;
    case POP3Tasks::PASSWORD:
        raw_request << "PASS " << std::string(request->arguments.password,
            request->arguments.password + strlen(request->arguments.password));;
        break;
    case POP3Tasks::QUIT:
        raw_request << "QUIT";
        break;
    default:
        this->current_task = POP3Tasks::NONE;
        this->onError(request->task, "Task not implemented");
    }
    raw_request << "\r\n";

    TCPClient::request(raw_request.str().c_str(), raw_request.str().size());
}

POP3Client::POP3Client(std::string address, int port) :
    TCPClient(address, port) {
    this->current_task = POP3Tasks::NONE;
}
```

```
// Содержит базовые классы для создания SMTP-клиента
```

```cpp
#pragma once

#include <queue>
#include <map>
#include <webstur/ip/tcp/tcpclient.h>
#include <sstream>

namespace SMTPTasks {
    // Структура, описывающая типы задач, выполняемые SMTP-клиентом
    enum SMTPTasks
    {
        // У клиента нет задачи, все входящие сообщения будут проигнорированы
        NONE,
        // Первое сообщение от сервера, подтверждающее успешное соединение
        INIT,
        // Отправка информации о комьютере
        HELO,
        // Отправка информации об отправителе
        MAIL,
        // Отправка информации о получателе
        RCPT,
        // Отправка запроса на приём почты
        DATA,
        // Отправка тела письма
        DATA_RAW,
        // Выход из SMTP-сессии
        QUIT
    };
}

// Описывает задачу к выполнению и её аргументы
struct SMTPRequest {
    SMTPRequest() {};
    ~SMTPRequest() {};

    SMTPTasks::SMTPTasks task = SMTPTasks::NONE;
    union Arguments {
        Arguments() {};
        ~Arguments() {};

        // Имя компьютера, используется в задаче INIT
        const char* helo;
        // Отправитель, используется в задаче MAIL
        const char* from;
        // Отправитель, используется в задаче RCPT
        const char* to;
        // Письмо, используется в задаче DATA_RAW
        struct MailContents {
            MailContents() {};
            ~MailContents() {};

            // Тело письма
            const char* body;
            // Размер письма
            std::size_t size;
        } mail;
    } arguments;
};

class DLLEXPORT SMTPClient : public TCPClient {
private:
    SMTPTasks::SMTPTasks current_task;
    std::string buffer;

    std::string extractNextResponseFromBuffer();
    void delegateResponse(const std::string& response);
```

```
protected:
    void onConnect();
    void onDisconnect();
    void onMessage(const std::vector<char>& message);
public:
    SMTPClient(std::string address, int port = 25);

    void request(const char* payload, int payload_size);

    virtual void onHello() = 0;

    virtual void onMail() = 0;

    virtual void onRecipient() = 0;

    virtual void onData() = 0;

    virtual void onDataRaw() = 0;

    virtual void onInit() = 0;

    virtual void onQuit() = 0;

    virtual void onError(SMTPTasks::SMTPTasks failed_task, std::string response) = 0;
};
```

```
#include "pch.h"
#include <assert.h>
#include <webstur/ip/tcp/smtpclient.h>
#include <webstur/utils.h>

void SMTPClient::onConnect() {
    // Ожидаем, что первое полученное сообщение будет сообщение инициализации
    this->current_task = SMTPTasks::INIT;
}

void SMTPClient::onDisconnect() {
    // Ничего не делаем
}

std::string SMTPClient::extractNextResponseFromBuffer() {
    std::size_t break_line_pos;
    std::string response;
    break_line_pos = this->buffer.find("\r\n");
    if (break_line_pos != std::string::npos) {
        response = this->buffer.substr(0, break_line_pos);
        this->buffer = this->buffer.substr(break_line_pos + 2);
    }

    return response;
}

void SMTPClient::delegateResponse(const std::string& response) {
    // Сбросить текущую задачу
    auto old_task = this->current_task;
    this->current_task = SMTPTasks::NONE;

    // Если ответ начинается с ошибки, то вызвать метод-обработчик ошибки
    if (response.starts_with("4") || response.starts_with("5")) {
        this->onError(old_task, response);
        return;
    }

    else {
```

```
        switch (old_task) {
        case SMTPTasks::INIT:
            this->onInit();
            break;
        case SMTPTasks::HELO:
            this->onHello();
            break;
        case SMTPTasks::MAIL:
            this->onMail();
            break;
        case SMTPTasks::RCPT:
            this->onRecipient();
            break;
        case SMTPTasks::DATA:
            this->onData();
            break;
        case SMTPTasks::DATA_RAW:
            this->onDataRaw();
            break;
        case SMTPTasks::QUIT:
            this->onQuit();
            break;
        default:
            this->onError(old_task, "Task not implemented");
        }
    }
}

void SMTPClient::onMessage(const std::vector<char>& message) {
    // Сохранить ответ в буфер
    this->buffer += std::string(message.begin(), message.end());

    while (true) {
        std::string response = extractNextResponseFromBuffer();

        // Если ответ пустой, выходим из метода
        if (response.empty())
            return;

        // Если текущая задача не задана, также выходим
        if (this->current_task == SMTPTasks::NONE)
            continue;

        this->delegateResponse(response);
    }
}

void SMTPClient::request(const char* payload, int payload_size) {
    if (this->current_task != SMTPTasks::NONE)
        return;
    assert(payload_size >= sizeof(SMTPRequest));
    auto request = ((SMTPRequest*)payload);
    std::stringstream raw_request;
    this->current_task = request->task;

    switch (request->task) {
    case SMTPTasks::HELO:
        raw_request << "HELO " << std::string(request->arguments.helo,
            request->arguments.helo + strlen(request->arguments.helo));
        break;
    case SMTPTasks::MAIL:
        raw_request << "MAIL FROM:<" << std::string(request->arguments.from,
            request->arguments.from + strlen(request->arguments.from)) << ">";
        break;
    case SMTPTasks::RCPT:
        raw_request << "RCPT TO:<" << std::string(request->arguments.to,
            request->arguments.to + strlen(request->arguments.to)) << ">";
```

```
        break;
    case SMTPTasks::DATA:
        raw_request << "DATA";
        break;
    case SMTPTasks::DATA_RAW:
        raw_request << std::string(request->arguments.mail.body,
            request->arguments.mail.body + request->arguments.mail.size) << "\r\n.";
        break;
    case SMTPTasks::QUIT:
        raw_request << "QUIT";
        break;
    default:
        this->current_task = SMTPTasks::NONE;
        this->onError(request->task, "Task not implemented");
    }
    raw_request << "\r\n";

    TCPClient::request(raw_request.str().c_str(), raw_request.str().size());
}

SMTPClient::SMTPClient(std::string address, int port) :
    TCPClient(address, port) {
    this->current_task = SMTPTasks::NONE;
}
```

```
// POP3-клиент для консольного приложения

#pragma once

#include <webstur/ip/tcp/pop3client.h>

class MyPOP3Client : public POP3Client {
    std::string login;
    std::string password;
public:
    MyPOP3Client(std::string address, std::string login, std::string password);

    void onMailList(std::vector<EMLHeader> mail_list);

    void onMail(const EMLContent& mail_content);

    void onMailDelete();

    void onInit();

    void onLogin();

    void onPassword();

    void onQuit();

    void onError(POP3Tasks::POP3Tasks failed_task, std::string response);
};
```

```
#include <iostream>
#include <fstream>
#include "mypop3client.h"

#define LOG_PREFIX "My POP3 client: "

MyPOP3Client::MyPOP3Client(std::string address, std::string login, std::string password) :
POP3Client(address), login(login), password(password) {}

void MyPOP3Client::onMailList(std::vector<EMLHeader> mail_list) {
```

```
    std::cout << LOG_PREFIX << "A " << mail_list.size() << " mails are not read\n";
    for (auto& mail : mail_list)
        std::cout << "  ID: " << mail.id << " Size: " << mail.size << " bytes\n";

    std::cout << std::flush;
}

void MyPOP3Client::onMail(const EMLContent& mail_content) {
    std::cout << LOG_PREFIX << "A mail content is received" << std::endl;

    // Получить уникальное имя файла
    auto unique_filename = getUniqueFilepath();
    unique_filename = unique_filename.substr(0, unique_filename.size() - 4) + ".rmf";

    // Открыть файл
    std::ofstream out(unique_filename, std::ios::binary);

    if (!out.is_open()) {
        std::cout << "Unable to save a file" << std::endl;
        return;
    }

    // Запись в файл
    out.write(mail_content.data.c_str(), mail_content.data.size());

    // Сохранить и закрыть файл
    out.flush();
    out.close();

    std::cout << LOG_PREFIX << "A mail is saved at '" << unique_filename << "'" << std::endl;
}

void MyPOP3Client::onMailDelete() {
    std::cout << LOG_PREFIX << "A mail is deleted successfully" << std::endl;
}

void MyPOP3Client::onInit() {
    std::cout << LOG_PREFIX << "A POP3 connection is established" << std::endl;
    // Отправить запрос на логин
    POP3Request login_request;
    login_request.task = POP3Tasks::LOGIN;
    login_request.arguments.login = this->login.c_str();
    this->request((const char*)(&login_request), sizeof(login_request));
}

void MyPOP3Client::onLogin() {
    std::cout << LOG_PREFIX << "A login is succedeed succesfully" << std::endl;
    // Отправить запрос на пароль
    POP3Request password_request;
    password_request.task = POP3Tasks::PASSWORD;
    password_request.arguments.password = this->password.c_str();
    this->request((const char*)(&password_request),
        sizeof(password_request));
}

void MyPOP3Client::onPassword() {
    std::cout << LOG_PREFIX << "A password is succedeed succesfully" << std::endl;
}

void MyPOP3Client::onQuit() {
    std::cout << LOG_PREFIX << "A client quitted, you can exit now" << std::endl;

    // Выключить клиент
    this->shutdown();
}

void MyPOP3Client::onError(POP3Tasks::POP3Tasks failed_task, std::string response) {
```

```
    std::cout << LOG_PREFIX << "A POP3 failed with error '" << response <<
        "' for task " << failed_task << std::endl;

    switch (failed_task) {
    case POP3Tasks::LOGIN:
        std::cout << LOG_PREFIX << "Invalid login" << std::endl;
        this->shutdown();
        break;
    case POP3Tasks::PASSWORD:
        std::cout << LOG_PREFIX << "Invalid password" << std::endl;
        this->shutdown();
        break;
    }
}
```

```
// SMTP-клиент для консольного приложения

#pragma once

#include <istream>
#include <webstur/ip/tcp/smtpclient.h>

class MySMTPClient : public SMTPClient {
    std::string login;
    std::string receiver;
    std::istream* data;
public:
    MySMTPClient(std::string address);

    void setLogin(const std::string& login);

    void setReceiver(const std::string& receiver);

    void setDataStream(std::istream& istream);

    void onHello();

    void onMail();

    void onRecipient();

    void onData();

    void onDataRaw();

    void onInit();

    virtual void onQuit();

    void onError(SMTPTasks::SMTPTasks failed_task, std::string response);
};
```

```
#include <iostream>
#include <iterator>
#include "mysmtpclient.h"

#define LOG_PREFIX "My SMTP client: "

MySMTPClient::MySMTPClient(std::string address) : SMTPClient(address) {

}

void MySMTPClient::setLogin(const std::string& login) {
    this->login = login;
```

```
}

void MySMTPClient::setReceiver(const std::string& receiver) {
    this->receiver = receiver;
}

void MySMTPClient::setDataStream(std::istream& istream) {
    this->data = &istream;
}

void MySMTPClient::onHello() {
    std::cout << LOG_PREFIX << "Hello request succeeded. Sending mail from request" << std::endl;

    // Отправить запрос с отправителем
    SMTPRequest request;
    request.task = SMTPTasks::MAIL;
    request.arguments.from = this->login.c_str();
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onMail() {
    std::cout << LOG_PREFIX << "Mail from request succeeded. Sending recipient request" <<
std::endl;

    // Отправить запрос с отправителем
    SMTPRequest request;
    request.task = SMTPTasks::RCPT;
    request.arguments.to = this->receiver.c_str();
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onRecipient() {
    std::cout << LOG_PREFIX << "Recipient to request succeeded. Sending data request" <<
std::endl;

    // Отправить запрос на отправку данных
    SMTPRequest request;
    request.task = SMTPTasks::DATA;
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onData() {
    std::cout << LOG_PREFIX << "Data request succeeded. Sending a data" << std::endl;

    // Считать стрим в массив
    std::vector<char> data_raw(std::istreambuf_iterator<char>(*this->data),
        std::istreambuf_iterator<char>());

    // Отправить запрос на отправку данных
    SMTPRequest request;
    request.task = SMTPTasks::DATA_RAW;
    request.arguments.mail.body = &data_raw[0];
    request.arguments.mail.size = data_raw.size();
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onDataRaw() {
    std::cout << LOG_PREFIX << "Data sent. Now we can quit" << std::endl;

    // Отправить запрос на отправку данных
    SMTPRequest request;
    request.task = SMTPTasks::QUIT;
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onInit() {
```

```
    std::cout << LOG_PREFIX << "An initial connection established. Sending hello request" <<
std::endl;

    // Получить имя устройства
    char deviceName[1024];
    unsigned long deviceNameSize = sizeof(deviceName);
    if (!GetComputerNameA(deviceName, &deviceNameSize)) {
        std::cout << LOG_PREFIX << "Couldn't get computer name" << std::endl;
        this->shutdown();
    }

    // Отправить HELLO-запрос
    SMTPRequest request;
    request.task = SMTPTasks::HELO;
    request.arguments.helo = deviceName;
    this->request((const char*)&request, sizeof(request));
}

void MySMTPClient::onQuit() {
    std::cout << LOG_PREFIX << "SMTP client quitted and says BB!" << std::endl;

    // Выключить клиент
    this->shutdown();
}

void MySMTPClient::onError(SMTPTasks::SMTPTasks failed_task, std::string response) {
    std::cout << LOG_PREFIX << "An error occured while send. Error: " << response << std::endl;

    // Выключить клиент
    this->shutdown();
}
```

```
#include <iostream>
#include <algorithm>
#include <ws2tcpip.h>
#include <fstream>
#include "mypop3client.h"
#include "mysmtpclient.h"

int main() {
    try {
        IClient::init();
        std::string address, login, password;

        std::cout << "Input server address: " << std::flush;
        std::cin >> address;
        std::cout << "Input login: " << std::flush;
        std::cin >> login;
        std::cout << "Input password: " << std::flush;
        std::cin >> password;

        MyPOP3Client client(address, login, password);
        client.start();
        MySMTPClient smtp_client(address);
        smtp_client.setLogin(login);
        std::ifstream file_to_send;
        smtp_client.setDataStream(file_to_send);

        // Считываем команду пользователя
        while (true)
        {
            std::cout
                << "Enter 1 to get mail list\n"
                << "Enter 2 to retrieve mail\n"
                << "Enter 3 to delete mail\n"
```

```
                << "Enter 4 to send mail\n"
                << "Enter 5 to update POP3-server state\n"
                << "Enter 6 to exit program\n"
                << std::endl;

            std::string input;
            std::cin >> input;
            std::cin.get();
            std::transform(input.begin(), input.end(), input.begin(), toupper);
            if (input == "1") {
                // Отправить запрос на список писем
                POP3Request mail_list_request;
                mail_list_request.task = POP3Tasks::LIST;
                client.request((const char*)&mail_list_request, sizeof(mail_list_request));
            }
            else if (input == "2") {
                // Отправить запрос на письмо
                POP3Request mail_request;
                mail_request.task = POP3Tasks::MAIL;
                // Ввести ID письма
                std::cout << "Input mail ID to retrieve: " << std::flush;
                std::cin >> mail_request.arguments.id;
                client.request((const char*)&mail_request, sizeof(mail_request));
            }
            else if (input == "3") {
                // Отправить запрос на удаление письма
                POP3Request mail_request;
                mail_request.task = POP3Tasks::DEL;
                // Ввести ID письма
                std::cout << "Input mail ID to delete: " << std::flush;
                std::cin >> mail_request.arguments.id;
                client.request((const char*)&mail_request, sizeof(mail_request));
            }
            else if (input == "4") {
                // Ввести получателя и файл для отправки
                std::string recipient;
                std::cout << "Input recipient: " << std::flush;
                std::cin >> recipient;
                std::string file_path;
                std::cout << "Input file path: " << std::flush;
                std::cin >> file_path;

                smtp_client.setReceiver(recipient);
                if (file_to_send.is_open())
                    file_to_send.close();
                file_to_send.open(file_path);

                smtp_client.start();
            }
            else if (input == "5") {
                // Отправить запрос на закрытие клиента
                POP3Request quit_request;
                quit_request.task = POP3Tasks::QUIT;
                client.request((const char*)&quit_request, sizeof(quit_request));
            }
            else if (input == "6") {
                break;
            }
        }
    }
    catch (const std::runtime_error& error) {
        std::cerr << "Failed while running server. Caused by: '" << error.what() << "'" <<
std::endl;

        return -1;
    }
    IClient::detach();
```

```
    return 0;
}
```