Описание протокола взаимодействия клиентской и серверной части

Задание: "Система терминального доступа"

Функциональные требования

Задание: разработать клиент-серверную систему терминального досту- па, позволяющую клиентам подсоединяться к серверу и выполнять эле- ментарные команды операционной системы. Основные возможности. Серверное приложение должно реализовы- вать следующие функции:

1) Прослушивание определенного порта

2) Обработка запросов на подключение по этому порту от клиентов

3) Поддержка одновременной работы нескольких терминальных кли- ентов через механизм нитей

4) Проведение аутентификации клиента на основе полученных имени пользователя и пароля

5) Выполнение команд пользователя: · ls – выдача содержимого каталога · cd – смена текущего каталога · who – выдача списка зарегистрированных пользователей с указани- ем их текущего каталога · kill – Привилегированная команда. Завершение сеанса другого поль- зователя. · logout – выход из системы

6) Принудительное отключение клиента Клиентское приложение должно реализовывать следующие функции: 1) Установление соединения с сервером 2) Посылка аутентификационных данных клиента (имя и пароль) 3) Посылка одной из команд (ls, cd, who, kill, logout) серверу 4) Получение ответа от сервера 5) Разрыв соединения 6) Обработка ситуации отключения клиента сервером или другим кли- ентом

Настройки приложений. Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени удалённого терминального сервера и номера порта, используемого сервером. Разработанное серверное приложение должно хранить аутентификаци- онные данные для вех пользователей, а также списки разрешенных каждо- му пользователю команд. Методика тестирования. Для тестирования приложений запускается терминальный сервер и несколько клиентов. В процессе тестирования про- веряются основные возможности сервера по параллельной работе несколь- ких клиентов, имеющих различные привилегии (списки разрешенных ко- манд). Проверяется корректность выполнения всех команда в различных ситуациях.

Нефункциональные требования

1.Требования к реализации

Соединение будет инициировать клиент, в случае протокола TCP клиент будет осуществлять обычное соединение с получением просьбы от сервера прислать имя и пароль. В случае использования протокола UDP клиент инициирует соединение и осуществляет отправку любой последовательности символов, после чего приходит запрос от сервера на получение имени пользователя и пароля. Дальнейшие действия сводятся к процессу аутентификации, и в случае удачной аутентификации происходит переход в режим ввода команд.

2.Требования к надежности

Длина посылаемых сообщений фиксирована и не требует проверки. Проверка выполняется на вводимые данные, пользователь не может выполнить команды, не имеющейся в списке. При неудачной аутентификации происходит разрыв соединения.

3.Накладываемые ограничения

Основное ограничение идет на длину посылаемых клиентом команд - 256 байт.

Ответы от сервера также имеют размер 256 байт, кроме результатов выполненяи команд в терминале, длина которых ограничена 1024 символами.

Анализ задачи

Система терминального доступа представляет сервис клиенту для осуществления команд в терминале на удаленном компьютере. Система состоит из двух частей:

1) **Серверная часть** - находится на галвном компьютере, предоставляющем свой терминал для пользования другим компьютерам и осуществления операций согласно правам доступа клиента к данному компьютеру.

2) **Клиентская часть** - является внешним компьютером, получающим возможность обращения к компьютеру на сервере для осуществления необходимых операций.

Сервер предоставляет право доступа к терминалу для нескольких подключенных пользователей. В его основные задачи входит проверка пользователя и пароля, предоставление списка доступных команд и проверка их ввода клиентом, обнаружение и исправление ошибок.

Клиент имеет право запускать только те команды, которые указаны в списке допустимых, а также осуществляет мониторинг ошибок при соединении с сервером.

Протокол взаимодействия опишем отдельно для клиента и для сервера и для разных реализаций TCP и UDP.

Описание взаимодействий по протоколу TCP

Описание протокола взаимодействия серверной части

Прикладной протокол

Имя	Формат	Действие	Длина
Connect	null	Инициализация соединения	0
Запрос от сервера	"Please, give me your username and password"	Сервер запрашивает имя пользователя и пароля	256
usernamepasswd	<username>_<password>	Клиент отправляет свои имя и пароль	256
Sequence	<sequence>	Сервер отправляет случайную последовательность	256
HASHCLIENT	int	Клиент отправляет подсчитанный хэш серверу	256
HASH_CHECKED	"hash ok"	Сообщение от сервера при удачной аутентификации	256
HASH_NOT_CHECKED	"hash not ok"	Сообщение от сервера при неудачной аутентификации	256
Аутентификация	"HELLO!"	Сервер отправляет сообщение об удачной аутентификаии	256
Command	"command"	Клиент отправляет серверу запрос на выполнение команды	256
CD()	cdok	Сервер отвечает клиенту на команду cd	1024
Logout()	"GOOD buy!"	Сервер обрывает соединение	1024
Ls()	"list of files"	Сервер высылает клиенту список файлов текущей директории	1024
Who()	"list of username, directories"	Сервер отправляет список пользователей и папок	1024
Kill()	null	Сервер отвечает клиенту на команду kill	1024

Формат строки null не имеет своего сообщения

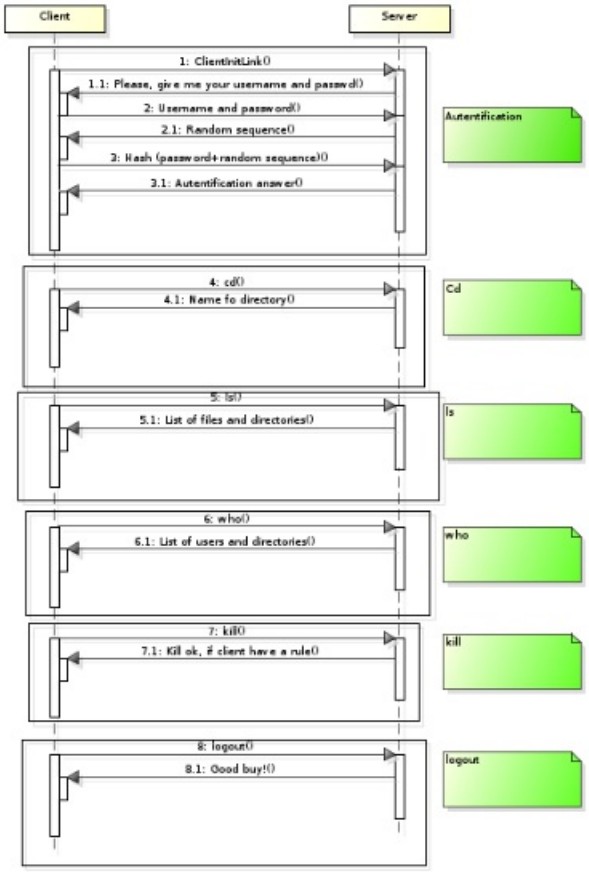
Формат строки <username>_<password> имеет следующий формат, как пример, Anton_12345

Формат строки "command" может принимать все различные значения из списка : cd,ls,kill,logout,who

Формат строки может иметь любое значение, например, "1a2b3c4d" - случайная последовательность. В программе она принимает постоянное значение

Формат строки "list of files" представлен в виде списка имеющихся файлов

В остальных случаях посылается текст, который виден по таблице.



Архитектура приложения

Дизайн протокола

Серверная часть инициализируется запуском с указанием используемого TCP-порта. После чего сервер переводится в режим прослушивания порта. Опишем как работает протокол для взаимодействия с один клиентом. Сервер осуществляет многопоточное соединение с различными подключающимися клиентами. Сервер ожидает подключения клиента и получения первого сообщения с указанием имени пользователя и пароля. Имя пользователя и пароль присылаются в следующем формате:

<username>_<password>

Синтаксический обработчик прочтет имя пользователя и пароль и запишет их в переменные. Далее сервер начнет проводить аутентификацию имени пользователя и пароля, аутентификация будет происходить как на серверной, так и на клиентской части. Сначала сервер обратиться к файлу, содержащему имена и пароли всех компьютеров, у которых есть доступ к удаленному терминалу. Данный файл имеет формат .txt и содержит данные о пользователе, пароле и используемых командах в следующем формате:

Идентификатор	Значение
Имя пользователя	Антон
Пароль	1234
Доступные команды терминала	cd ls mkdir
Текущий каталог	/home/anton
Имя пользователя	User
Пароль	5678
Доступные команды терминала	cd ls
Текущий каталог	/home/anton
Имя пользователя	Root

Пароль	0912
Доступные команды терминала	cd ls rm
Текущий каталог	/home/anton

Данная таблица представлет структуру файла для трёх пользователей.

Сервер произведет сравнение данных, полученных от пользователя с данными в файле: при положительном результате клиент будет переведен на второй этап аутентификации пользователя и пароля. Клиенту будет отправлено число для пересчета хэш-функции имени пользователя и пароля. Сервер также будет осуществлять подсчет хэш-фугкций имени пользователя и пароля. Клиент, имеющий доступ к терминалу владеет специальной функцией подсчета хэш-кода, как и сервер. Для подсчета кода взята функция Ly.

Ly- функция

```
unsigned int HashLy(const char * str)
{
    unsigned int hash = 0;

    for(; *str; str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;

    return hash;
}
```

После подсчет хэш-функций клиентом и сервером, сервер осуществит сравнение полученных данных и в случае полодительной проверки подлинности предоставит терминал клиенту на право пользования, в противном случае "обрубит соединение". В случае положительной проверки пользовательских данных клиент получит сообщение: **"Hello!"** Далее клиенту будет отпрален список всех возможных команд для использования.

Сервер будет осуществлять запуск команд от клиента, извлекая их из сообщений. Каждое сообщение для удобства будет помещаться в буфер типа char размером до 256 байт.

Каждую полученную команду сервер будет проверять по файлу всех команд для данного пользователя.

Команды будут вызываться в терминале с помощью функции **popen** из библиотеки **stdio.h**. Результат выполнения будет записываться в бубчер размером 1024 байта и отправляться клиенту.

Обработка сервером ошибок

В случае если на сервере появилась ошибка при запуске команды из терминала, то клиенту будет отправлено соответствующее сообщение об ошибке, а сервер приостановит процесс выполнения команд и отключит клиента.

Ошибки соединения контролируются пользователем на компьютере клиента, в случае неудачного присоединения пользователь сам совершит переприсоединение к серверу.

Описание протокола взаимодействия клиентской части

Клиентская часть осуществляет запрос на главный компьютер с целью получить терминал компьютера в пользование. Все сообщения посылаемые клиентом помещаются в символьный буфер размером 256 байт, за исключением сообщения от команды who (1024 байта). Запрос на подключение к терминалу сервера осуществляется автоматически: клиент инициирует соединение, сервер запрашивает у него имя пользователя и пароль, которые далее отправляются. После сервер отправляет клиенту случаную последовательность символов. Полученное сообщение конкатенируется с паролем и для них просчитывается хэш-код. Клиент отправляет свой вычисленный хэш и пользователю останется только дождаться сообщения о готовности использования терминала: "Добро пожаловать!".

Пользователю достаточно вводить используемые команды в командной строке при наличии подключения. Все команды отправляются в виде 256-байтного сообщения, введенного с консоли. Клиент принимает сообщения от сервера в буфер размером 256 либо 1024 байта.

На различные исключительные ситуации расставлены обработчики, осуществляющие обработку и исправление ошибок.

Список вводимых клиентом команд:

- ls: получение списка файлов и директорий в текущей директории;
- cd: произвести переход к другой папке;
- who: запрос на получение списка всех пользователей и их текущих директорий;
- logout: завершение сеанса;

Многопоточное взаимодействие для связи с несколькими клиентами.

Для осуществления многоклиентского взаимодействия используются потоки, это удобно посколько с каждым клиентом устанавливается логическое соединение, что удобно, потому что каждый поток будет защищен от другого клиента. Сервер при постоянном режиме прослушивает необходимый порт и при подключении клиента, выделяет ему отдельный поток и переводит себя в стандартный режим общения с клиентом.

Каждый поток запускается на сервере и для каждого клиента, запущенного в разных приложениях. При появлении нового клиента происходит создание для него потока. Важной задачей является задача синхронизации ресурсов между потоками. Все потоки разделют между собой ресурс конфигурационного файла, каждый поток производит считывание из файла и запись в него. Для того, чтобы обеспечить безопасный доступ к файлу с целью неповреждения его используют мьютексы. В данном приложении можно использовать мьютекс для синхронизации доступа к ресурсу файла. Мьютекс находящийся в состоянии 0, не захвачен потоком. При изменении идентификатор мьютекса на идентификатор потока происходит захват потоком мьютекса. Это решение позволит обеспечить безопасный доступ к файлу и сделать каждому потоку монопольное обладание файлом.

Создание мьютекса можно продемонстрировать на небольшом примере:

```
int main(int argc, char* argv[])
{
    char filename[30]; //имя файла ко��фигурации
    if(argc<2)
    {
     … queue.data=new char*[config.sizeOfQueue]; //инициализируем средство синхронизации
     mutex=CreateMutex(NULL,FALSE,""); //запускаем потоки на исполнение

     for(int i=0;i<config.numOfReaders+config.numOfWriters+1;i++) ResumeThread(allhandlers[i]); //ожидаем
завершения всех потоков
     WaitForMultipleObjects(config.numOfReaders+config.numOfWriters+1,allhandlers,TRUE,INFINITE
     );
     for(int i=0;i<config.numOfReaders+config.numOfWriters+1;i++) //закрываем handle потоков
     CloseHandle(allhandlers[i]); //удаляем объект синхронизации CloseHandle(mutex);
     printf("all is done\n"); return 0;
}
```

Тестирование TCP-приложения

Для тестирования была проведена необходимая отладка отдельных частей и компонентов. Были проверены вводимые данные и производился поиск основных ошибок, связанных с число передаваемых байт, аутентификация чтение из файла и выполнение команд в терминале.

Протокол TCP тестировался на создание логического соединения, на осуществления правильной аутентификации и на передачу команд, предусмотрены некоторые исключения, касающиеся неправильно введенных

предусмотрены некоторые исключения, касающиеся неправильно введенных данных и неправильно подсчитанных хэш-функций.

Тест 1. Проведение теста на ввод других команд. В результате выдается обычная ошибка на сервере и система переходит в режим ввода новой команды.

Тест 2. Проверка аутентификации, в случае если аутентификация была пройдена неуспешно, то произойдет обрыв соединения.

Тест 3. Проверка на многопоточноть произошла успешно после подключения нескольких клиентов было включено несколько потоков на каждого, смешивание пространств каждого клиента не происходит.

Из нереализованных функций остались введение пользователем некорректных данных, а именно неправильно введенных имени пользователя и пароля, в данном случае необходимо было бы переключить систему снова на этап запроса на аутентиикацию, чего не происходит. Осталось нереализованным завершение соединения с клиентом от сервера. В данном случае необходимо было бы наделить сервера такой функцией и реализовать её в цикле набора терминальных команд.

Описание взаимодействий по протоколу UDP

Описание клиентской части

Клиент UDP является управляющим и инициирующем всю систему, порядок работы настраивает именно он, поскольку сервер выполняет функцию обработчика команд, то есть в зависимости от присланного клиентом сообщения сервер будет передавать одно или иное сообщение.

Клиент в данном варианте инициализирует соединение. Пользователь вводит с консоли любую команду и отправляет её серверу и подсоединяется к нему. После чего в обычном порядке происходит стандартное взаимодействие. Единственным отличием от TCP-взаимодействия является то, что каждая команда, которую собирается набрать пользователь предварительно передается с ключевым значением. Всего имеются следующие ключевые слова:

1. My data - говорит серверу, что далее будет осуществляться аутентификация клиента.

2. commands - говорит серверу, что далее будет осуществляться переслыка команд терминала.

3. иное - говорит серверу инициировать общение с клиентом.

Имя	Формат	Действие	Длина
MY_DATA_REQUEST	"My data"	Запрос на отправку данных	256
COMMANDS_REQUEST	"commands"	Запрос от клиента на сервер для ввода команд	
256			
OTHER_REQUEST	other	Установление соединения клиента с сервером	256

После передачи каждой такой команды осуществляется соответствующее действие по аутентификации, или передачи команд аналогично протоколу TCP.

Описание серверной части

Сервер UDP содержит необходимую логику для взаимодействия с несколькими клиентами в отличии от протокола TCP, так как данный протокол не устанавливает логического соединения. Сервер находится в постоянной работе по принятию дейтаграммных сообщений и в зависимости от присланного сообщения осуществляет те или иные действия по взаимодействию.

Сервер дожидается специальных команд, после чего каждую принятую он обрабатывает на соответствие и переводит себя в необходимое состояние для работы. Каждый блок для специальных команд полностью аналогичен взаимодействию по протоколу TCP.

Многоклиентское взаимодействие

В данной задаче, так как не предусматрвиается установления логического соединения, то удобна организация асинхронного сервера, реагирующего на сообщения. Поэтому реализация такой задачи была осуществлена с помощью логики, а не потоков.

UDP дейтаграмма

Длина посылки

Поле, задающее длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка — 8 байт. Теоретически, максимальный размер поля — 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 — 65507 (помимо 8 байт на UDP-заголовок требуется ещё 20 на IP-заголовок).

На практике также следует учитывать, что если длина IPv4 пакета с UDP будет превышать MTU (для Ethernet по умолчанию 1500 байт), то отправка такого пакета может вызвать его фрагментацию, что может привести к тому, что он вообще не сможет быть доставлен, если промежуточные маршрутизаторы или конечный хост не будут поддерживать фрагментированные IP пакеты. Также в RFC791 указывается минимальная длина IP пакета не менее 576 байт и рекомендуется отправлять IP пакеты большего размера только в том случае если вы уверены, что принимающая сторона может принять пакеты такого размера. Следовательно, чтобы избежать фрагментации UDP пакетов (и возможной их потери), размер данных в UDP не должен превышать: MTU — (Max IP Header Size) — (UDP Header Size) = 1500 — 60 — 8 = 1432 байт. Для того чтобы быть уверенным, что пакет будет принят любым хостом, размер данных в UDP не должен превышать: (минимальная длина IP пакета) — (Max IP Header Size) — (UDP Header Size) = 576 — 60 — 8 = 508 байт.

Из данных теоретических сведений можно сделать вывод о том, как нужно отправлять большие сообщения по протоколу UDP. В данной задании осуществляется передача больших данных, а для обеспечения нажености их доставки, можно организовать обмен, поделив исходную большую посылку на несколько частей, каждая из которых точно вложится в дейтаграмму. Для этого необходимо организовать цикл на стороне сервера и стороне приема таким образом, чтобы оба вели контроль за принимаемыми сообщениями. Сервер первым делом отправит сообщение о колчестве дейтаграмм, которые он хочет отправить, клиент получив данное сообщение инициализирует цикл приема. Передача от сервера клиенту осуществляется подейтаграммно. В конце приема всех дейтаграмм клиент осуществляет сравнение количества принятых посылок и контрольного числа, отправленного сервером. В случае удачного приема осуществляется уведомление серверу о приеме. В случае ошибки, цикл приема длинного сообщения перезапускается и начинается вновь.

Еще одним аспектом служит контроль за порядком приема, чтобы посылки приходили в нужно порядке или их можно было распознать и составить из них целостное сообщение. Для осуществления такого необходимо предусмотреть усправляющую логику. Перед каждой отправкой одной дейтаграммы, сервер может отправлять её номер отдельной посылкой, а клиент будет их собирать поочередно и составлять из них единое сообщение. Данный способ будет весьма надежен, так как вероятность, не получения клиентом номера дейтаграммы нисока в связи с небольшим размером самого сообщения.

Тестирование UDP-приложения

Тестирование UDP аналогично TCP, особым случаем является проверка многоклиентского режима работы. В данном случае не происходит наложение пространств данных, так как оба клиента пользуются только своими данными. Логическая реализация многоклиентского режима была эффективна и сбоев не вывела.

Выводы

Анализ выполненных заданий, сравнение удобства/эффективности/ко- личества проблем при программировании TCP/UDP

Реализация для TCP

Теоритические сведения TCP — ориентированный на соединение протокол, что означает необхо- димость «рукопожатия» для установки соединения между двумя хоста- ми. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях. Особенности протокола TCP • Надёжность — TCP управляет подтверждением, повторной пере- дачей и тайм-аутом сообщений. Производятся многочисленные по- пытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших дан- ных, ни (в случае многочисленных тайм-аутов) разорванных соеди- нений. • Упорядоченность — если два сообщения последовательно отправле- ны, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправ- ляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению. • Тяжеловесность — TCP необходимо три пакета для установки сокет- соединения перед тем, как отправить данные. TCP следит за на- дёжностью и перегрузками. • Потоковость — данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов. Анализ TCP протокол является надежным протоколом с установлением соедине- ния, в связи с чем для TCP клиента помимо создания сокета, необходимо организовать соединение с помощью функции connect. TCP сервер должен содержать, как минимум, 2 сокета. Один сокет необ- ходим для фиксирования прихода запроса на соединение.После чего для каждого подключившегося клиента создается отдельный сокета. Это послужило затруднением при реализации многопоточной работы приложения, так как первоначально необходимо создать главный по- ток, слушающий определенный порт. Как только к серверу подключа- ется клиент, создается новый сокет для этого клиента, после чего он передается в новую нить, содержащую необходимые действия клиента. Главная нить имеет возможность закрыть сокет любого подключенного клиента.

Реализация для UDP

Теоритические сведения UDP — более простой, основанный на сообщениях протокол без установ- ления соединения. Протоколы такого типа не устанавливают выделенно- го соединения между двумя хостами. Связь достигается путем передачи информации в одном направлении от источника к получателю без про- верки готовности или состояния получателя. Особенности протокола UDP • Ненадёжный — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут. • Неупорядоченность — если два сообщения отправлены одному по- лучателю, то порядок их достижения цели не может быть предуга- дан. • Легковесность — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уро- вень, разработанный на IP. • Датаграммы — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют опреде- ленные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано. Анализ Заметим, что структура протокол UDP более простая, чем TCP протокола.Во-первых, нет необходимости использования функции connect, то есть установления адреса и порта по умолчанию для протокола UDP. Однако тогда параметры удалённой стороны бу- дем указывать или получать при каждом вызове операций записи или чтения с помощью функций sendto и recvfrom. Однако это создаёт проблемы при попытки реализации многопо- точной работы сервера. Так как на каждого клиента не создается отдельного сокета и все клиенты используют 1 сокет, не получи- лось организовать параллллельной работы клиентов. Я использо- вала мьютексы для распределения доступа к сокету, это не дает возможности параллельной работы клиентов, и новый подключив- шийся клиент может общаться с сервером только после завершения работы предыдущего клиента.

TCP Windows Server

```
    /*
    Bind socket to port 8888 on localhost
*/
#include<io.h>
#include<stdio.h>
#include<winsock2.h>
#include <stdlib.h>
#include <string.h>
#include <direct.h>
#include <locale.h>


#include <sys/types.h>
#include <limits.h>


#pragma comment(lib,"ws2_32.lib") //Winsock Library

#define N 80

//Структура для хранения имен пользователей, паролей, команд, текущих директорий
struct UserFields{
    char username[256];
    char password[256];
    char commands[256];
    char currentDir[256];
};
//Массив структур пользователей
struct ArrFields{
    int size;
    struct UserFields* arr;
};
//Возвращает текущий каталог
char* retCurrentDirectory()
{
    char *PathName = NULL;
    size_t t = FILENAME_MAX;
    PathName = _getcwd (PathName,t);
    if (PathName == NULL)
        printf ("Ошибка определения пути");
    else
        printf ("Текущая директория: %s\n",PathName);
    return PathName;
}

//Сменить текущюю директорию
void chDirectory(struct ArrFields fields, char newdir[256], char username[256])
{
    int i;
    for(i = 0; i < fields.size; i++)
    {
        printf("%s\n",username);
        printf("%s\n",fields.arr[i].username);
        printf("\n");
        if(strcmp(fields.arr[i].username,username) == 0)
        {
            memset(fields.arr[i].currentDir,0,256);
               strncpy_s(fields.arr[i].currentDir,newdir,strlen(newdir));
        }
    }
    FILE *file;
    file = fopen("usersandpasswords.txt", "w");
    for(i = 0; i < fields.size; i++)
    {
        fprintf(file, "%s", fields.arr[i].username);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].password);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].commands);
        fprintf(file, "%s", "\n");
```

```
        fprintf(file, "%s", fields.arr[i].currentDir);
        fprintf(file, "%s", "\n");
    }
    fclose(file);
}

//Создать сообщение из списка имен пользователей и директорий
char* createWhoMessage(struct ArrFields fields)
{
    char whomessage[1024];
    memset(whomessage,0,1024);
    int i;
    printf("%i\n",fields.size);
    for(i = 0; i < fields.size; i++)
    {
        strcat_s(whomessage,fields.arr[i].username);
        strcat_s(whomessage,"\n");
        strcat_s(whomessage,fields.arr[i].currentDir);
        strcat_s(whomessage,"\n");
    }
    return whomessage;
}
//Хеш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for( ; *str; str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод сообщения об ошибке
void error(const char *msg)
{
    perror(msg);
    exit(1);
}
//Отсоединение клиента
void killclient()
{
    system("killall client");
    exit(1);
}

//Добавить нового пользователя
void addNewUser(char* username, char* password)
{
    char command[1024];
    memset(command,0,1024);
    strcat_s(command,"echo '");
    strcat_s(command,username);
    strcat_s(command,"\n");
    strcat_s(command,password);
    strcat_s(command,"\n");
    strcat_s(command,"cd ls who kill logout\n");
    strcat_s(command,retCurrentDirectory());
    strcat_s(command,"' >> usersandpasswords.txt");
       FILE   *pPipe;

        /* Run DIR so that it writes its output to a pipe. Open this
         * pipe with read text attribute so that we can read it
         * like a text file.
         */
    if( (pPipe = _popen( command, "rt" )) == NULL )
        exit( 1 );

    /* Close pipe and print return value of pPipe. */
    if (feof( pPipe))
    {
        printf( "\nProcess returned %d\n", _pclose( pPipe ) );
```

```
    }
    else
    {
        printf( "Error: Failed to read the pipe to the end.\n");
    }
}
//Считывание содержимого файла пользователей и паролей
struct ArrFields readFile()
{
    struct UserFields usr;
    struct UserFields arr[10];
    struct ArrFields arrfields;
    // Переменная, в которую будет помещен указатель на созданный
    // поток данных
    FILE *mf;
    // Переменная, в которую поочередно будут помещаться считываемые строки
    char str[50];
    //Указатель, в который будет помещен адрес массива, в который считана
    // строка, или NULL если достигнут коней файла или произошла ошибка
    char *estr;

    // Открытие файла с режимом доступа «только чтение» и привязка к нему
    // потока данных
    mf = fopen("usersandpasswords.txt","r");

    // Проверка открытия файла
    if (mf == NULL)
    {
        printf ("ошибка\n");
    }
    int strcount = 0;
    int size = 0;
    //Чтение (построчно) данных из файла в бесконечном цикле
    while (1)
    {
       // Чтение одной строки  из файла
       estr = fgets (str,sizeof(str),mf);
       //Проверка на конец файла или ошибку чтения
       if (estr == NULL)
       {
          // Проверяем, что именно произошло: кончился файл
          // или это ошибка чтения
          if ( feof (mf) != 0)
          {
             //Если файл закончился, выводим сообщение о завершении
             //чтения и выходим из бесконечного цикла
             //printf ("\nЧтение файла закончено\n");
             break;
          }
          else
          {
             //Если при чтении произошла ошибка, выводим сообщение
             //об ошибке и выходим из бесконечного цикла
            // printf ("\nОшибка чтения из файла\n");
             break;
          }
       }
       if(strcount == 0){
                 strncpy_s(usr.username,str,strlen(str));
                 usr.username[strlen(str)-1] = '\0';
            }
       else if(strcount == 1){
                 strncpy_s(usr.password,str,strlen(str));
                 usr.password[strlen(str)-1] = '\0';
            }
       else if(strcount == 2){
                 strncpy_s(usr.commands,str,strlen(str));
                 usr.commands[strlen(str)-1] = '\0';
            }
       else if(strcount == 3){
```

```
                strncpy_s(usr.currentDir,str,strlen(str));
                usr.currentDir[strlen(str)-1] = '\0';
        }
        strcount = strcount+1;
        if(strcount == 4)
        {
            strcount = 0;
            arr[size] = usr;
            size = size+1;
        }
    }
    arrfields.size = size;
    arrfields.arr = arr;
    // Закрываем файл
    if ( fclose (mf) == EOF)
        printf ("ошибка\n");
    else printf ("выполнено\n");
    return arrfields;
}

DWORD WINAPI myThread1(LPVOID lpParameter)
{
    SOCKET new_socket = *((SOCKET*)lpParameter);
    char hashsend[256] = "1q2w3e4r";//Случайная последовательность

    char answer[1024];
    char buffer[256];
    int n;
    //Аутентификация пользователя
    int is_auntentity_ok = 0;
    char savebuf[256];
     //do
    // {
         n = send(new_socket,"Please, give me your username and password via blank:",255,0);
       //  if (n < 0)
       //      error("ERROR writing to socket");

         memset(buffer,0,256);//Взятие данных от пользователя
         n = recv(new_socket,buffer,256,0);
         Sleep(5);
        // if (n < 0)
        //     error("ERROR reading from socket");
         printf("Here is the message: %s\n",buffer);
         char temp[256];
         memset(temp,0,256);
         memset(savebuf,0,256);
         strncpy_s(temp,buffer,strlen(buffer));
         struct ArrFields arrfields = readFile();
         int i = 0;
         int yes = 0;
         for(i = 0; i < arrfields.size; i++)
         {
             char usrpassw [256];
             memset(usrpassw,0,256);
             strcat_s(usrpassw,arrfields.arr[i].username);
             strcat_s(usrpassw," ");
             strcat_s(usrpassw,arrfields.arr[i].password);
             if(strcmp(temp,usrpassw) == 0)
             {
                 yes = 1;
                 strncpy_s(savebuf,temp,strlen(temp));
             }
         }
         printf("%i\n",yes);
         if(yes == 1)
         {
             is_auntentity_ok = 1;
             printf("%s\n",hashsend);
             printf("%i\n",strlen(hashsend));
             n = send(new_socket,hashsend,strlen(hashsend),0);
```

```
    //      if (n < 0)
    //          error("ERROR writing to socket");
      }
      else
      {
          n = send(new_socket,"Polzovatel' and Parol' neverny!",255,0);
   //       if (n < 0)
   //           error("ERROR writing to socket");
          //close(newsockfd);
          //killclient();
      }
//  } while(is_auntentity_ok != 1);
  is_auntentity_ok = 0;
 // do
 // {
       char hashbuf[256];
       memset(hashbuf,0,256);
       strcat_s(hashbuf,savebuf);
       strcat_s(hashbuf,hashsend);
      printf("%s\n",hashbuf);
      printf("%i\n",strlen(hashbuf));
      unsigned int res = HashH37(hashbuf);//Вычисление хэш-функции
      printf("%i\n",res);
      char ans [256];
      memset(ans,0,256);
      memset(buffer,0,256);
      sprintf_s(buffer, "%d", res);
      strcpy_s(ans,buffer);
      printf("Подсчитанный хеш: %s\n",ans);
      printf("Подсчитанный хеш длина: %i\n",strlen(ans));
      memset(buffer,0,256);
      n = recv(new_socket,buffer,256,0);
      printf("Полученный хеш: %s\n",buffer);
      printf("Полученный хеш длина: %i\n",strlen(buffer));
   //   if (n < 0)
   //       error("ERROR reading from socket");
      if (strcmp(ans,buffer) == 0)
      {
          printf("Hash ok\n");
          n = send(new_socket,"Hello!",255,0);
   //       if (n < 0)
   //           error("ERROR writing to socket");
          is_auntentity_ok = 1;
         }
      else
      {
          printf("Hash not ok\n");
          n = send(new_socket,"Hash neveren",255,0);
    //      if (n < 0)
     //         error("ERROR writing to socket");
          killclient();
         }
 // } while(is_auntentity_ok != 1);
  while(1)
  {
      memset(buffer,0,256);
      memset(answer,0,256);
      n = recv(new_socket,buffer,256,0);
    //  if (n < 0)
    //      error("ERROR reading from socket");
      printf("Here is the message: %s\n",buffer);
      char tempbuf[256];
      memset(tempbuf,0,256);
      strncpy_s(tempbuf,buffer,strlen(buffer)-1);
      printf("%s\n",tempbuf);
      if(strcmp(tempbuf,"logout") == 0)
      {
          printf("logout yes\n");
          n = send(new_socket,"Good Buy!",255,0);
```

```
    //         if (n < 0)
    //             error("ERROR writing to socket");
            char path[256];
            memset(path,0,256);
            closesocket(new_socket);
        }
        else if(strcmp(tempbuf,"who") == 0)
        {
            printf("who yes\n");
            char whomessage[1024];
            memset(whomessage,0,1024);
            chDirectory(arrfields, "/home/anton/workspace/tcpproj", "Anton");
            strcpy_s(whomessage,createWhoMessage(arrfields));
            n = send(new_socket,whomessage,255,0);
    //         if (n < 0)
     //            error("ERROR writing to socket");
        }

        else
        {
            printf("%s\n",buffer);


            FILE *fp;
            int status;
            char path[256];
            memset(path,0,256);
            fp = _popen(buffer, "r");
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            char prov[256];
            memset(prov,0,256);
            strncpy_s(prov,buffer,strlen(buffer)-1);
            if((strcmp(prov,"cd") == 0) || (strcmp(prov,"cd ..") == 0))
            {
                printf("cdok");
                strcpy_s(answer,"cdok");
            }
            else {
                while (fgets(path, 256, fp) != NULL)
                {
                    printf("%s", path);
                    memset(buffer,0,256);
                    strncpy_s(buffer,path,strlen(path));
                    strcat_s(answer,buffer);
                }
            }
            n = send(new_socket,answer,strlen(answer),0);
    //         if (n < 0)
    //             error("ERROR writing to socket");
            status = _pclose(fp);
            if (status == -1) {
                error("Error with executing of command\n");
            }
        }
     }
     closesocket(new_socket);
    return 0;
}


int main(int argc , char *argv[])
{
    setlocale(LC_ALL,"Russian");
    HANDLE myHandle1;
    DWORD myThreadID;
    WSADATA wsa;
    SOCKET sockfd, new_socket;
    int portno;
    char hashsend[256] = "1q2w3e4r";
```

```
    int clilen;
    struct sockaddr_in serv_addr, cli_addr;

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");

    //Create a socket
    if((sockfd = socket(AF_INET , SOCK_STREAM , 0 )) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d" , WSAGetLastError());
    }

    printf("Socket created.\n");

    memset((char *) &serv_addr,0, sizeof(serv_addr));
    portno = 50000;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(portno);
     //Bind
    if( bind(sockfd ,(struct sockaddr *)&serv_addr , sizeof(serv_addr)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
    }

    puts("Bind done");

    //Listen to incoming connections
    listen(sockfd , 5);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    while(1)
    {
    //Listen to incoming connections
    listen(sockfd , 5);

        clilen = sizeof(struct sockaddr_in);
        printf("%i\n",clilen);
        new_socket = accept(sockfd , (struct sockaddr *)&cli_addr, &clilen);
        if (new_socket == INVALID_SOCKET)
        {
            printf("accept failed with error code : %d" , WSAGetLastError());
        }
        printf("Est' soedineniye\n");

        myHandle1 = CreateThread(NULL, 0, myThread1,&new_socket, 0, NULL);
    //  CloseHandle(myHandle1);
    }
    closesocket(sockfd);
    CloseHandle(myHandle1);
    WSACleanup();
    return 0;
}
```

TCP Windows Client

```
    /**
    Simple TCP client to fetch a web page
    Silver Moon (m00n.silv3r@gmail.com)
*/

/*
```

```
    Create a TCP socket
*/

#include<stdio.h>
#include<winsock2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#pragma comment(lib,"ws2_32.lib") //Winsock Library

//Хэш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for(;*str;str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод ошибок клиента
void error(const char *msg)
{
    perror(msg);//Открыть сообщение с выводом информации на консоль
    exit(0);
}
int main(int argc, char *argv[])
{
     WSADATA wsa;
     SOCKET sockfd;
     int portno, n;//Дескриптор сокета
     struct sockaddr_in serv_addr;//Адрес сервера
     struct hostent *server;//IP-address

    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        return 1;
    }

    printf("Initialised.\n");
     char buffer[256];//Полученное сообщение
     portno = 50000;//Получение номера порта сервера
     sockfd = socket(AF_INET, SOCK_STREAM, 0);//Получение дескриптора сокета
     if (sockfd < 0)//Если получить дескриптор не удалось
         error("ERROR opening socket");

    memset((char*) &serv_addr,0,sizeof(serv_addr));
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(50000);

     /*if (*/connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr));/* < 0);*///Подсоединение к серверу
          //error("ERROR connecting");

     char passbuf[256];
     int is_auntentity_ok = 0;
         memset(buffer,0,256);
         n = recv(sockfd,buffer,255,0);//Чтение полученного сообщения
         printf("%s\n",buffer);
         printf("Please enter the message: ");//Введите сообщение
         memset(buffer,0,256);
         memset(passbuf,0,256);
         fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
         n = send(sockfd,buffer,strlen(buffer)-1,0);//Отправка сообщения на сервер
         strncpy(passbuf,buffer,strlen(buffer)-1);

         memset(buffer,0,256);
         n = recv(sockfd,buffer,255,0);//Чтение полученного сообщения
         printf("%s\n",buffer);
```

```
        printf("%i\n",strlen(buffer));
        if(strcmp(buffer,"Polzovatel' and Parol' neverny!") == 0)
        {
            is_auntentity_ok = 1;
        }
    is_auntentity_ok = 0;
    char hashbuf[256];
    memset(hashbuf,0,256);
    strcat_s(hashbuf,passbuf);
    strcat_s(hashbuf,buffer);
    printf("%s\n",hashbuf);
    printf("%i\n",strlen(hashbuf));
        unsigned int res = HashH37(hashbuf);
        printf("%i\n",res);
        memset(buffer,0,256);
        sprintf_s(buffer, "%d", res);
        n = send(sockfd,buffer,256,0);//Отправка сообщения на сервер

        memset(buffer,0,256);
        n = recv(sockfd,buffer,255,0);//Чтение полученного сообщения

        printf("%s\n",buffer);//Выведение сообщения на экран
        if(strcmp(buffer,"Hello!") == 0)
            is_auntentity_ok = 1;

    int i = 0;
    for(i = 0; i < 10; i++)
    {
        printf("Please enter the message: ");//Введите сообщение
        memset(buffer,0,256);
        fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
        n = send(sockfd,buffer,strlen(buffer),0);//Отправка сообщения на сервер

        char tmp[256];
        memset(tmp,0,256);
        strncpy_s(tmp,buffer,strlen(buffer)-1);
        if(strcmp(tmp,"logout") == 0)
        {
            printf("sovpalo\n");
           closesocket(sockfd);
        }
        memset(buffer,0,256);
        n = recv(sockfd,buffer,255,0);//Чтение полученного сообщения

        printf("%s\n",buffer);//Выведение сообщения на экран
     }
     closesocket(sockfd);//Закрытие сокета
     WSACleanup();
     return 0;
}
```

UDP Windows Client

```
    Simple udp client
    Silver Moon (m00n.silv3r@gmail.com)
*/

/* Sample UDP client */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library
//Хэш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
```

```
{
    unsigned int hash = 0;
    for(;*str;str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод ошибок клиента
void error(const char *msg)
{
    perror(msg);//Открыть сообщение с выводом информации на консоль
    exit(0);
}

int main(int argc, char**argv)
{
   int sockfd,n;
   struct sockaddr_in servaddr,cliaddr;
   char sendline[256];
    WSADATA wsa;

    //Initialise winsock
    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    printf("Initialised.\n");

   if (argc != 2)
   {
      printf("usage:  udpcli <IP address>\n");
      exit(1);
   }
   sockfd=socket(AF_INET,SOCK_DGRAM,0);

   memset(&servaddr,0,sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[1]);
   servaddr.sin_port=htons(32000);

   //Инициализация соединения от клиента
   //Передача адреса, считывается с консоли
    fgets(sendline, 256,stdin);
    memset(sendline,0,256);
    strcpy(sendline,"new");
    sendto(sockfd, sendline, 256,0,(struct sockaddr*) &servaddr,sizeof(servaddr));
    char buffer[256];
    char passbuf[256];
    int is_auntentity_ok = 0;
     memset(buffer,0,256);
     memset(buffer,0,256);
     memset(passbuf,0,256);
     //Отправка кода операций
     sendto(sockfd,"My data",256,0,
                 (struct sockaddr *)&servaddr,sizeof(servaddr));

     n = recvfrom(sockfd,buffer,256,0,NULL,NULL);
     printf("%s\n",buffer);

     //Далее происходит этап аутентификации
     printf("Please enter the message: \n");//Введите сообщение
     memset(buffer,0,256);
     memset(passbuf,0,256);
     fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
     buffer[strlen(buffer)] = '\0';
     sendto(sockfd,buffer,strlen(buffer)-1,0,
                 (struct sockaddr *)&servaddr,sizeof(servaddr));

     strncpy_s(passbuf,buffer,strlen(buffer)-1);
```

```
memset(buffer,0,256);
n = recvfrom(sockfd,buffer,255,0,NULL,NULL);

printf("%s\n",buffer);
printf("%i\n",strlen(buffer));
if(strcmp(buffer,"Polzovatel' and Parol' neverny!") == 0)
{
    is_auntentity_ok = 1;
}

is_auntentity_ok = 0;

char hashbuf[256];
memset(hashbuf,0,256);
strcat_s(hashbuf,passbuf);
strcat_s(hashbuf,buffer);
printf("%s\n",hashbuf);
printf("%i\n",strlen(hashbuf));
unsigned int res = HashH37(hashbuf);
printf("%i\n",res);
memset(buffer,0,256);
sprintf(buffer, "%d", res);
sendto(sockfd,buffer,256,0,
                (struct sockaddr *)&servaddr,sizeof(servaddr));
memset(buffer,0,256);
n = recvfrom(sockfd,buffer,256,0,NULL,NULL);

if (n < 0)
    error("ERROR reading from socket");
printf("%s\n",buffer);//Выведение сообщения на экран
if(strcmp(buffer,"Hello!") == 0)
    is_auntentity_ok = 1;

int i = 0;
for(i = 0; i < 10; i++)
{
    printf("Please enter the message: ");//Введите сообщение

    memset(buffer,0,256);
    fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив

    sendto(sockfd,"commands",strlen("commands"),0,
                        (struct sockaddr *)&servaddr,sizeof(servaddr));
    Sleep(2);
    sendto(sockfd,buffer,strlen(buffer),0,
                    (struct sockaddr *)&servaddr,sizeof(servaddr));
    char tmp[256];
    memset(tmp,0,256);
    strncpy_s(tmp,buffer,strlen(buffer)-1);
    if(strcmp(tmp,"logout") == 0)
    {
        printf("sovpalo\n");
        FILE *fp;
        int status;
       /* char path[256];
        memset(path,0,256);
        memset(buffer,0,256);
        char command[256];
        memset(command,0,256);
        strcat_s(command,"kill -s 9 ");
        char n_str[10];
        memset(n_str,0,10);
        sprintf(n_str, "%d", pid);
        strcat_s(command,n_str);
        fp = _popen(command, "r");
        if (fp == NULL)
            error("Failed to execute a command in the terminal\n");
        status = _pclose(fp);
        if (status == -1) {
            error("Error with executing of command\n");
```

```
            }*/
        }
        memset(buffer,0,256);
        n = recvfrom(sockfd,buffer,256,0,NULL,NULL);
        if (n < 0)
            error("ERROR reading from socket");
        printf("%s\n",buffer);//Выведение сообщения на экран
        //Запись сообщения от пользователоя в массив
      }
      closesocket(sockfd);
      WSACleanup();
      return 0;
}
```

UDP Windows Server

```
#include<io.h>
#include<stdio.h>
#include<winsock2.h>
#include <stdlib.h>
#include <string.h>
#include <direct.h>
#include <locale.h>
#include<stdio.h>
#include<winsock2.h>

#pragma comment(lib,"ws2_32.lib") //Winsock Library


#include <sys/types.h>
#include <limits.h>

//Структура для хранения имен пользователей, паролей, команд, текущих директорий
struct UserFields{
    char username[256];
    char password[256];
    char IP[256];
    char commands[256];
    char currentDir[256];
};
//Массив структур пользователей
struct ArrFields{
    int size;
    struct UserFields* arr;
};
//Возвращает текущий каталог
char* retCurrentDirectory()
{
    char *PathName = NULL;
    size_t t = FILENAME_MAX;
    PathName = _getcwd (PathName,t);
    if (PathName == NULL)
        printf ("Ошибка определения пути");
    else
        printf ("Текущая директория: %s\n",PathName);
    return PathName;
}
//Сменить текущюю директорию
void chDirectory(struct ArrFields fields, char newdir[256], char username[256])
{
    int i;
    for(i = 0; i < fields.size; i++)
    {
        printf("%s\n",username);
        printf("%s\n",fields.arr[i].username);
        printf("\n");
        if(strcmp(fields.arr[i].username,username) == 0)
        {
            memset(fields.arr[i].currentDir,0,256);
```

```
            strncpy_s(fields.arr[i].currentDir,newdir,strlen(newdir));
        }
    }
    FILE *file;
    file = fopen("udpusersandpasswords.txt", "w");
    for(i = 0; i < fields.size; i++)
    {
        fprintf(file, "%s", fields.arr[i].username);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].password);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].commands);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].currentDir);
        fprintf(file, "%s", "\n");
    }
    fclose(file);
}
//Создать сообщение из списка имен пользователей и директорий
char* createWhoMessage(struct ArrFields fields)
{
    char whomessage[1024];
    memset(whomessage,0,1024);
    int i;
    printf("%i\n",fields.size);
    for(i = 0; i < fields.size; i++)
    {
        strcat_s(whomessage,fields.arr[i].username);
        strcat_s(whomessage,"\n");
        strcat_s(whomessage,fields.arr[i].currentDir);
        strcat_s(whomessage,"\n");
    }
    return whomessage;
}
//Хеш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for( ; *str; str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод сообщения об ошибке
void error(const char *msg)
{
    perror(msg);
    exit(1);
}
//Отсоединение клиента
void killclient()
{
    system("killall client");
    exit(1);
}
//Добавить нового пользователя
void addNewUser(char* username, char* password, char* IP)
{
    FILE *fp;
    int status;
    char command[1024];
    memset(command,0,1024);
    strcat_s(command,"echo '");
    strcat_s(command,username);
    strcat_s(command,"\n");
    strcat_s(command,password);
    strcat_s(command,"\n");
    strcat_s(command,IP);
    strcat_s(command,"\n");
    strcat_s(command,"cd ls who kill logout\n");
    strcat_s(command,retCurrentDirectory());
```

```
    strcat_s(command,"' >> udpusersandpasswords.txt");
    fp = _popen(command, "r");
    status = _pclose(fp);
}
//Считывание содержимого файла пользователей и паролей
struct ArrFields readFile()
{
    struct UserFields usr;
    struct UserFields arr[10];
    struct ArrFields arrfields;
    // Переменная, в которую будет помещен указатель на созданный
    // поток данных
    FILE *mf;
    // Переменная, в которую поочередно будут помещаться считываемые строки
    char str[50];
    //Указатель, в кот��рый будет помещен адрес массива, в который считана
    // строка, или NULL если достигнут коней файла или произошла ошибка
    char *estr;

    // Открытие файла с режимом доступа «только чтение» и привязка к нему
    // потока данных
    mf = fopen ("udpusersandpasswords.txt","r");

    // Проверка открытия файла
    if (mf == NULL)
    {
        printf ("ошибка\n");
    }
    int strcount = 0;
    int size = 0;
    //Чтение (построчно) данных из файла в бесконечном цикле
    while (1)
    {
        // Чтение одной строки  из файла
        estr = fgets (str,sizeof(str),mf);
        //Проверка на конец файла или ошибку чтения
        if (estr == NULL)
        {
            // Проверяем, что именно произошло: кончился файл
            // или это ошибка чтения
            if ( feof (mf) != 0)
            {
                //Если файл закончился, выводим сообщение о завершении
                //чтения и выходим из бесконечного цикла
                //printf ("\nЧтение файла закончено\n");
                break;
            }
            else
            {
                //Если при чтении произошла ошибка, выводим сообщение
                //об ошибке и выходим из бесконечного цикла
                // printf ("\nОшибка чтения из файла\n");
                break;
            }
        }
        if(strcount == 0) {
            strncpy_s(usr.username,str,strlen(str));
            usr.username[strlen(str)-1] = '\0';
        }
        else if(strcount == 1){
            strncpy_s(usr.password,str,strlen(str));
            usr.password[strlen(str)-1] = '\0';
        }
        else if(strcount == 2){
            strncpy_s(usr.commands,str,strlen(str));
            usr.commands[strlen(str)-1] = '\0';
        }
        else if(strcount == 3){
            strncpy_s(usr.currentDir,str,strlen(str));
            usr.currentDir[strlen(str)-1] = '\0';
```

```
            usr.currentDir[strlen(str)-1] = '\0';
        }
        strcount = strcount+1;
        if(strcount == 4)
        {
            strcount = 0;
            arr[size] = usr;
            size = size+1;
        }
    }
    arrfields.size = size;
    arrfields.arr = arr;
    // Закрываем файл
    if ( fclose (mf) == EOF)
        printf ("ошибка\n");
    else printf ("выполнено\n");
    return arrfields;
}

int main(int argc, char**argv)
{
    setlocale(LC_ALL,"Russian");
    struct ArrFields arrfields = readFile();
    char answer[1024];
    char buffer[256];
    char IP [256];
    char hashsend[256] = "1q2w3e4r";//Случайная последовательность
    int n;
    SOCKET sockfd;
    struct sockaddr_in servaddr,cliaddr;
    int len;

    WSADATA wsa;
     //Initialise winsock
    printf("\nInitialising Winsock...");
    if (WSAStartup(MAKEWORD(2,2),&wsa) != 0)
    {
        printf("Failed. Error Code : %d",WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    printf("Initialised.\n");

    if ((sockfd = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)) == INVALID_SOCKET)
    {
        printf("Socket not create");
    }
    memset(&servaddr,0,sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(32000);

    //Bind
    if( bind(sockfd ,(struct sockaddr *)&servaddr , sizeof(servaddr)) == SOCKET_ERROR)
    {
        printf("Bind failed with error code : %d" , WSAGetLastError());
        exit(EXIT_FAILURE);
    }
    puts("Bind done");
        char mesg[256];
        memset(mesg,0,256);
    while(1) {
        len = sizeof(cliaddr);
        n = recvfrom(sockfd,mesg,256,0,(struct sockaddr *)&cliaddr,&len);
        printf("%s\n",mesg);

        if(strcmp(mesg,"My data") == 0)
        {
            //Если получено сообщение на принятие данных о пользователе и пароле
            //Аутентификация пользователя
            int is_auntentity_ok = 0;
```

```
            char savebuf[256];
            sendto(sockfd,"Please, give me your username and password via blank:",255,0,(struct sockaddr
*)&cliaddr,sizeof(cliaddr));
            memset(buffer,0,256);//Взятие данных от пользователя
            n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
            Sleep(5);
            if (n < 0)
                error("ERROR reading from socket");
            printf("Here is the message: %s\n",buffer);
            char temp[256];
            memset(temp,0,256);
            memset(savebuf,0,256);
            strncpy_s(temp,buffer,strlen(buffer));
            int i = 0;
            int yes = 0;
            for(i = 0; i < arrfields.size; i++)
            {
                 char usrpassw [256];
                 memset(usrpassw,0,256);
                 strcat_s(usrpassw,arrfields.arr[i].username);
                 strcat_s(usrpassw," ");
                 strcat_s(usrpassw,arrfields.arr[i].password);
                 printf("%i\n",strlen(usrpassw));
                 printf("%i\n",strlen(temp));
                 if(strcmp(temp,usrpassw) == 0)
                 {
                     yes = 1;
                     strncpy_s(savebuf,temp,strlen(temp));
                 }
             }
             printf("%i\n",yes);
             if(yes == 1)
             {
                 is_auntentity_ok = 1;
                 printf("%s\n",hashsend);
                 printf("%i\n",strlen(hashsend));
                 sendto(sockfd,hashsend,strlen(hashsend),0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
                 if (n < 0)
                     error("ERROR writing to socket");
             }
             else
             {
                 sendto(sockfd,"Polzovatel' and Parol' neverny!",255,0,(struct sockaddr
*)&cliaddr,sizeof(cliaddr));
                 if (n < 0)
                     error("ERROR writing to socket");
             }

             is_auntentity_ok = 0;
             char hashbuf[256];
             memset(hashbuf,0,256);
             strcat_s(hashbuf,savebuf);
             strcat_s(hashbuf,hashsend);
             printf("%s\n",hashbuf);
             printf("%i\n",strlen(hashbuf));
             unsigned int res = HashH37(hashbuf);//Вычисление хэш-функции
             printf("%i\n",res);
             char ans [256];
             memset(ans,0,256);
             memset(buffer,0,256);
             sprintf(buffer, "%d", res);
             strcpy_s(ans,buffer);
             printf("Подсчитанный хеш: %s\n",ans);
             printf("Подсчитанный хеш длина: %i\n",strlen(ans));
             memset(buffer,0,256);
             n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
             printf("Полученный хеш: %s\n",buffer);
             printf("Полученный хеш длина: %i\n",strlen(buffer));
             if (n < 0)
                 error("ERROR reading from socket");
```

```
        if (strcmp(ans,buffer) == 0)
        {
            printf("Hash ok\n");
            sendto(sockfd,"Hello!",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
            is_auntentity_ok = 1;
        }
        else
        {
            printf("Hash not ok\n");
            sendto(sockfd,"Hash neveren",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
            killclient();
        }
    }
    else if(strcmp(mesg,"commands") == 0)
    {
        printf("The commands are used!\n\n");
        //Если получено сообщение в режиме ввода комманд

        memset(buffer,0,256);
        memset(answer,0,256);
        n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
        Sleep(5);
        if (n < 0)
            error("ERROR reading from socket");
        printf("Here is the message: %s\n",buffer);
        char tempbuf[256];
        memset(tempbuf,0,256);
        strncpy_s(tempbuf,buffer,256);
        printf("%s\n",tempbuf);
        if(strcmp(tempbuf,"logout") == 0)
        {
            printf("logout yes\n");
            sendto(sockfd,"Good Buy!",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            FILE *fp;
            int status;
            char path[256];
            memset(path,0,256);
        }
        else if(strcmp(tempbuf,"who") == 0)
        {
            printf("who yes\n");
            char whomessage[1024];
            memset(whomessage,0,1024);
            chDirectory(arrfields, "/home/anton/workspace/tcpproj", "Anton");
            strcpy_s(whomessage,createWhoMessage(arrfields));
            sendto(sockfd,whomessage,255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
        }

        else
        {
            printf("%s\n",buffer);

            FILE *fp;
            int status;
            char path[256];
            memset(path,0,256);
            buffer[strlen(buffer)] = '\0';
            fp = _popen(buffer, "r");
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            char prov[256];
            memset(prov,0,256);
            strncpy_s(prov,buffer,256);
            if((strcmp(prov,"cd") == 0) || (strcmp(prov,"cd ..") == 0))
            {
```

```
                    printf("cdok\n");
                    strcpy_s(answer,"cdok");
                }
                else {
                    while (fgets(path, 256, fp) != NULL)
                    {
                        printf("%s", path);
                        memset(buffer,0,256);
                        strncpy_s(buffer,path,strlen(path));
                        strcat_s(answer,buffer);
                    }
                }
                sendto(sockfd,answer,strlen(answer),0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
                status = _pclose(fp);
                if (status == -1) {
                    error("Error with executing of command\n");
                }
            }
            memset(mesg,0,10000);

        }
        else
        {
            //Если происходит запрос на получение от клиента IP-адреса
            //Здесь сервер получает адрес и сохраняет его в файле, чтобы дальше проводить общение
            memset(IP,0,256);
            strcpy(IP,mesg);
            //Далее необходимо обновить конфигурационный файл
            strcpy(arrfields.arr->IP,IP);
        }
    }
    closesocket(sockfd);
    WSACleanup();
}
```

TCP Linux Server

```
/*
 * server.c
 *
 *  Created on: 05.11.2014
 *      Author: anton
 */



/* Sample TCP server */

/*Create by Anton Kiselyov*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <limits.h>

#define N 80

//Структура для хранения имен пользователей, паролей, команд, текущих директорий
struct UserFields{
    char username[256];
    char password[256];
    char commands[256];
    char currentDir[256];
};
//Массив структур пользователей
```

```
struct ArrFields{
    int size;
    struct UserFields* arr;
};
//Возвращает текущий каталог
char* retCurrentDirectory()
{
    char PathName[PATH_MAX];
    char PN;
    PN = getwd (PathName);
    if (PN == NULL)
        printf ("Ошибка определения пути");
    else
        printf ("Текущая директория: %s\n",PathName);
    return PathName;
}
//Сменить текущюю директорию
void chDirectory(struct ArrFields fields, char newdir[256], char username[256])
{
    int i;
    for(i = 0; i < fields.size; i++)
    {
        printf("%s\n",username);
        printf("%s\n",fields.arr[i].username);
        printf("\n");
        if(strcmp(fields.arr[i].username,username) == 0)
        {
            bzero(fields.arr[i].currentDir,256);
            strncpy(fields.arr[i].currentDir,newdir,strlen(newdir));
        }
    }
    FILE *file;
    file = fopen("usersandpasswords.txt", "w");
    for(i = 0; i < fields.size; i++)
    {
        fprintf(file, "%s", fields.arr[i].username);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].password);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].commands);
        fprintf(file, "%s", "\n");
        fprintf(file, "%s", fields.arr[i].currentDir);
        fprintf(file, "%s", "\n");
    }
    fclose(file);
}
//Создать сообщение из списка имен пользователей и директорий
char* createWhoMessage(struct ArrFields fields)
{
    char whomessage[1024];
    bzero(whomessage,1024);
    int i;
    printf("%i\n",fields.size);
    for(i = 0; i < fields.size; i++)
    {
        strcat(whomessage,fields.arr[i].username);
        strcat(whomessage,"\n");
        strcat(whomessage,fields.arr[i].currentDir);
        strcat(whomessage,"\n");
    }
    return whomessage;
}
//Хеш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for( ; *str; str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
```

```
}
//Вывод сообщения об ошибке
void error(const char *msg)
{
    perror(msg);
    exit(1);
}
//Отсоединение клиента
void killclient()
{
    system("killall client");
    exit(1);
}
//Добавить нового пользователя
void addNewUser(char* username, char* password)
{
    FILE *fp;
    int status;
    char command[1024];
    bzero(command,1024);
    strcat(command,"echo '");
    strcat(command,username);
    strcat(command,"\n");
    strcat(command,password);
    strcat(command,"\n");
    strcat(command,"cd ls who kill logout\n");
    strcat(command,retCurrentDirectory());
    strcat(command,"' >> usersandpasswords.txt");
    fp = popen(command, "r");
    status = pclose(fp);
}
//Считывание содержимого файла пользователей и паролей
struct ArrFields readFile()
{
    struct UserFields usr;
    struct UserFields arr[10];
    struct ArrFields arrfields;
    // Переменная, в которую будет помещен указатель на созданный
    // поток данных
    FILE *mf;
    // Переменная, в которую поочередно будут помещаться считываемые строки
    char str[50];
    //Указатель, в который будет помещен адрес массива, в который считана
    // строка, или NULL если достигнут коней файла или произошла ошибка
    char *estr;

    // Открытие файла с режимом доступа «только чтение» и привязка к нему
    // потока данных
    mf = fopen ("usersandpasswords.txt","r");

    // Проверка открытия файла
    if (mf == NULL)
    {
        printf ("ошибка\n");
    }
    int strcount = 0;
    int size = 0;
    //Чтение (построчно) данных из файла в бесконечном цикле
    while (1)
    {
       // Чтение одной строки  из файла
       estr = fgets (str,sizeof(str),mf);
       //Проверка на конец файла или ошибку чтения
       if (estr == NULL)
       {
          // Проверяем, что именно произошло: кончился файл
          // или это ошибка чтения
          if ( feof (mf) != 0)
          {
             //Если файл закончился, выводим сообщение о завершении
```

```
            //чтения и выходим из бесконечного цикла
            //printf ("\nЧтение файла закончено\n");
            break;
         }
         else
         {
            //Если при чтении произошла ошибка, выводим сообщение
            //об ошибке и выходим из бесконечного цикла
           // printf ("\nОшибка чтения из файла\n");
            break;
         }
      }
      if(strcount == 0){
             strncpy(usr.username,str,strlen(str));
             usr.username[strlen(str)-1] = '\0';
         }
      else if(strcount == 1){
             strncpy(usr.password,str,strlen(str));
             usr.password[strlen(str)-1] = '\0';
         }
      else if(strcount == 2){
             strncpy(usr.commands,str,strlen(str));
             usr.commands[strlen(str)-1] = '\0';
         }
      else if(strcount == 3){
             strncpy(usr.currentDir,str,strlen(str));
             usr.currentDir[strlen(str)-1] = '\0';
      }
      strcount = strcount+1;
      if(strcount == 4)
      {
          strcount = 0;
          arr[size] = usr;
          size = size+1;
      }
   }
   arrfields.size = size;
   arrfields.arr = arr;
   // Закрываем файл
   if ( fclose (mf) == EOF)
       printf ("ошибка\n");
   else printf ("выполнено\n");
   return arrfields;
}
//Запуск сервера
void * thread_func(int newsockfd)
{
   char hashsend[256] = "1q2w3e4r";//Случайная последовательность
   socklen_t clilen;
   pid_t pid;
   char answer[1024];
   char buffer[256];
   int n;
   //Аутентификация пользователя
   int is_auntentity_ok = 0;
   char savebuf[256];
    //do
   // {
        n = write(newsockfd,"Please, give me your username and password via blank:",255);
        if (n < 0)
            error("ERROR writing to socket");

        bzero(buffer,256);//Взятие данных от пользователя
        n = read(newsockfd,buffer,256);
        sleep(5);
        if (n < 0)
            error("ERROR reading from socket");
        printf("Here is the message: %s\n",buffer);
        char temp[256];
        bzero(temp,256);
```

```
        bzero(savebuf,256);
        strncpy(temp,buffer,strlen(buffer));
        struct ArrFields arrfields = readFile();
        int i = 0;
        int yes = 0;
        for(i = 0; i < arrfields.size; i++)
        {
            char usrpassw [256];
            bzero(usrpassw,256);
            strcat(usrpassw,arrfields.arr[i].username);
            strcat(usrpassw," ");
            strcat(usrpassw,arrfields.arr[i].password);
            if(strcmp(temp,usrpassw) == 0)
            {
                yes = 1;
                strncpy(savebuf,temp,strlen(temp));
            }
        }
        printf("%i\n",yes);
        if(yes == 1)
        {
            is_auntentity_ok = 1;
            printf("%s\n",hashsend);
            printf("%i\n",strlen(hashsend));
            n = write(newsockfd,hashsend,strlen(hashsend));
            if (n < 0)
                error("ERROR writing to socket");
        }
        else
        {
            n = write(newsockfd,"Polzovatel' and Parol' neverny!",255);
            if (n < 0)
                error("ERROR writing to socket");
            //close(newsockfd);
            //killclient();
        }
//  } while(is_auntentity_ok != 1);
  is_auntentity_ok = 0;
 // do
 // {
        char hashbuf[256];
        bzero(hashbuf,256);
        strcat(hashbuf,savebuf);
        strcat(hashbuf,hashsend);
        printf("%s\n",hashbuf);
        printf("%i\n",strlen(hashbuf));
        unsigned int res = HashH37(hashbuf);//Вычисление хэш-функции
        printf("%i\n",res);
        char ans [256];
        bzero(ans,256);
        bzero(buffer,256);
        sprintf(buffer, "%d", res);
        strcpy(ans,buffer);
        printf("Подсчитанный хеш: %s\n",ans);
        printf("Подсчитанный хеш длина: %i\n",strlen(ans));
        bzero(buffer,256);
        n = read(newsockfd,buffer,256);
        printf("Полученный хеш: %s\n",buffer);
        printf("Полученный хеш длина: %i\n",strlen(buffer));
        if (n < 0)
            error("ERROR reading from socket");
        if (strcmp(ans,buffer) == 0)
        {
            printf("Hash ok\n");
            n = write(newsockfd,"Hello!",255);
            if (n < 0)
                error("ERROR writing to socket");
            is_auntentity_ok = 1;
        }
```

```
        else
        {
            printf("Hash not ok\n");
            n = write(newsockfd,"Hash neveren",255);
            if (n < 0)
                error("ERROR writing to socket");
            killclient();
        }
    // } while(is_auntentity_ok != 1);
     while(1)
     {
        bzero(buffer,256);
        bzero(answer,256);
        n = read(newsockfd,buffer,256);
        if (n < 0)
            error("ERROR reading from socket");
        printf("Here is the message: %s\n",buffer);
        char tempbuf[256];
        bzero(tempbuf,256);
        strncpy(tempbuf,buffer,strlen(buffer)-1);
        printf("%s\n",tempbuf);
        if(strcmp(tempbuf,"logout") == 0)
        {
            printf("logout yes\n");
            n = write(newsockfd,"Good Buy!",255);
            if (n < 0)
                error("ERROR writing to socket");
            FILE *fp;
            int status;
            char path[256];
            bzero(path,256);
            close(newsockfd);
        }
        else if(strcmp(tempbuf,"who") == 0)
        {
            printf("who yes\n");
            char whomessage[1024];
            bzero(whomessage,1024);
            chDirectory(arrfields, "/home/anton/workspace/tcpproj", "Anton");
            strcpy(whomessage,createWhoMessage(arrfields));
            n = write(newsockfd,whomessage,255);
            if (n < 0)
                error("ERROR writing to socket");
        }

        else
        {
            printf("%s\n",buffer);

            FILE *fp;
            int status;
            char path[256];
            bzero(path,256);
            fp = popen(buffer, "r");
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            char prov[256];
            bzero(prov,256);
            strncpy(prov,buffer,strlen(buffer)-1);
            if((strcmp(prov,"cd") == 0) || (strcmp(prov,"cd ..") == 0))
            {
                printf("cdok");
                strcpy(answer,"cdok");
            }
            else {
                while (fgets(path, 256, fp) != NULL)
                {
                    printf("%s", path);
                    bzero(buffer,256);
                    strncpy(buffer,path,strlen(path));
```

```
                    strcat(answer,buffer);
                }
            }
            n = write(newsockfd,answer,strlen(answer));
            if (n < 0)
                error("ERROR writing to socket");
            status = pclose(fp);
            if (status == -1) {
                error("Error with executing of command\n");
            }
        }
    }
    close(newsockfd);
}

int main(int argc, char * argv[])
{
    int id1, result;
    pthread_t thread;

    int sockfd, newsockfd, portno;
    char hashsend[256] = "1q2w3e4r";
    socklen_t clilen;
    pid_t pid;
    char answer[1024];
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
    if (argc < 2) {//указать порт
        fprintf(stderr,"ERROR, no port provided\n");
        exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr,//привязка сервера к ресурсам
         sizeof(serv_addr)) < 0)
         error("ERROR on binding");
    listen(sockfd,5);
    clilen = sizeof(cli_addr);

    while(1)
    {
        newsockfd = accept(sockfd,//Открытие нового сокета
                (struct sockaddr *) &cli_addr,
            &clilen);
        if (newsockfd < 0)
            error("ERROR on accept");

        result = pthread_create(&thread, NULL, thread_func, newsockfd);
        if (result != 0)
        {
            perror("Creating the first thread");
            return EXIT_FAILURE;
        }
    }
    //close(sockfd);
    return 0;
}
```

TCP Linux Client

```
/*
 * client.c
```

```
 * client.c
 *
 *  Created on: 05.11.2014
 *      Author: anton
 */


/* Sample TCP client */
/*Created by Anton*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
//Хэш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for(;*str;str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод ошибок клиента
void error(const char *msg)
{
    perror(msg);//Открыть сообщение с выводом информации на консоль
    exit(0);
}

int main(int argc, char *argv[])
{
    int pid = getpid();
     int sockfd, portno, n;//Дескриптор сокета
     struct sockaddr_in serv_addr;//Адрес сервера
     struct hostent *server;//IP-address

     char buffer[256];//Полученное сообщение
     if (argc < 3) {//Ввести имя хоста, если не введено
         fprintf(stderr,"usage %s hostname port\n", argv[0]);
         exit(0);
     }
     portno = atoi(argv[2]);//Получение номера порта сервера
     sockfd = socket(AF_INET, SOCK_STREAM, 0);//Получение дескриптора сокета
     if (sockfd < 0)//Если получить дескриптор не удалось
         error("ERROR opening socket");
     server = gethostbyname(argv[1]);//Взятие IP-адреса сервера
     if (server == NULL) {//Если такого имени не существует
         fprintf(stderr,"ERROR, no such host\n");
         exit(0);
     }
     bzero((char *) &serv_addr, sizeof(serv_addr));//инициализация сервера
     serv_addr.sin_family = AF_INET;//Тип используемого транспортного протокола
     bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,//Адрес сервера
             server->h_length);
     serv_addr.sin_port = htons(portno);//Порт сервера
     //printf("h_addr: %s\n", inet_ntoa(serv_addr.sin_addr));
     if (connect(sockfd,(struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)//Подсоединение к серверу
          error("ERROR connecting");

     char passbuf[256];
     int is_auntentity_ok = 0;
     //do
     //{
         bzero(buffer,256);
         n = read(sockfd,buffer,255);//Чтение полученного сообщения
         printf("%s\n",buffer);
         printf("Please enter the message: ");//Введите сообщение
```

```
        bzero(buffer,256);
        bzero(passbuf,256);
        fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
        n = write(sockfd,buffer,strlen(buffer)-1);//Отправка сообщения на сервер
        strncpy(passbuf,buffer,strlen(buffer)-1);
        if (n < 0)
            error("ERROR writing to socket");
        bzero(buffer,256);
        n = read(sockfd,buffer,255);//Чтение полученного сообщения
        printf("%s\n",buffer);
        printf("%i\n",strlen(buffer));
        if(strcmp(buffer,"Polzovatel' and Parol' neverny!") == 0)
        {
            is_auntentity_ok = 1;
        }
    // }while(is_auntentity_ok != 1);
     is_auntentity_ok = 0;
     //do
     //{
     char hashbuf[256];
     bzero(hashbuf,256);
     strcat(hashbuf,passbuf);
     strcat(hashbuf,buffer);
     printf("%s\n",hashbuf);
     printf("%i\n",strlen(hashbuf));
        unsigned int res = HashH37(hashbuf);
        printf("%i\n",res);
        bzero(buffer,256);
        sprintf(buffer, "%d", res);
        n = write(sockfd,buffer,256);//Отправка сообщения на сервер
        if (n < 0)
            error("ERROR writing to socket");
        bzero(buffer,256);
        n = read(sockfd,buffer,255);//Чтение полученного сообщения
        if (n < 0)
            error("ERROR reading from socket");
        printf("%s\n",buffer);//Выведение сообщения на экран
        if(strcmp(buffer,"Hello!") == 0)
            is_auntentity_ok = 1;

    // }
    // while(is_auntentity_ok != 1);
     int i = 0;
     for(i = 0; i < 10; i++)
     {
        printf("Please enter the message: ");//Введите сообщение
        bzero(buffer,256);
        fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
        n = write(sockfd,buffer,strlen(buffer));//Отправка сообщения на сервер
        if (n < 0)
            error("ERROR writing to socket");
        char tmp[256];
        bzero(tmp,256);
        strncpy(tmp,buffer,strlen(buffer)-1);
        if(strcmp(tmp,"logout") == 0)
        {
            printf("sovpalo\n");
            FILE *fp;
            int status;
            char path[256];
            bzero(path,256);
            bzero(buffer,256);
            char command[256];
            bzero(command,256);
            strcat(command,"kill -s 9 ");
            char n_str[10];
            bzero(n_str,10);
            sprintf(n_str, "%d", pid);
            strcat(command,n_str);
            fp = popen(command, "r");
```

```
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            status = pclose(fp);
            if (status == -1) {
                error("Error with executing of command\n");
            }
        }
        bzero(buffer,256);
        n = read(sockfd,buffer,255);//Чтение полученного сообщения
        if (n < 0)
            error("ERROR reading from socket");

        printf("%s\n",buffer);//Выведение сообщения на экран
     }
     close(sockfd);//Закрытие сокета
     return 0;
}
```

UDP Linux Server

```c/* Sample UDP server */

# include

# include

# include

# include

# include

# include

# include

# include

# include

//Структура для хранения имен пользователей, паролей, команд, текущих директорий struct UserFields{ char username[256]; char password[256]; char IP[256]; char commands[256]; char currentDir[256]; }; //Массив структур пользователей struct ArrFields{ int size; struct UserFields* arr; }; //Возвращает текущий каталог char* retCurrentDirectory() { char PathName[PATH_MAX]; char PN; PN = getwd (PathName); if (PN == NULL) printf ("Ошибка определения пути"); else printf ("Текущая директория: %s\n",PathName); return PathName; } //Сменить текущюю директорию void chDirectory(struct ArrFields fields, char newdir[256], char username[256]) { int i; for(i = 0; i < fields.size; i++) { printf("%s\n",username); printf("%s\n",fields.arr[i].username); printf("\n"); if(strcmp(fields.arr[i].username,username) == 0) { bzero(fields.arr[i].currentDir,256); strncpy(fields.arr[i].currentDir,newdir,strlen(newdir)); } } FILE *file; file = fopen("udpusersandpasswords.txt", "w"); for(i = 0; i < fields.size; i++) { fprintf(file, "%s", fields.arr[i].username); fprintf(file, "%s", "\n"); fprintf(file, "%s", fields.arr[i].password); fprintf(file, "%s", "\n"); fprintf(file, "%s", fields.arr[i].commands); fprintf(file, "%s", "\n"); fprintf(file, "%s", fields.arr[i].currentDir); fprintf(file, "%s", "\n"); } fclose(file); } //Создать сообщение из списка имен пользователей и директорий char createWhoMessage(struct ArrFields fields) { char whomessage[1024]; bzero(whomessage,1024); int i; printf("%i\n",fields.size); for(i = 0; i < fields.size; i++) { strcat(whomessage,fields.arr[i].username); strcat(whomessage,"\n"); strcat(whomessage,fields.arr[i].currentDir); strcat(whomessage,"\n"); } return whomessage; } //Хеш-функция, используемая для проверки данных пользователя и пароля unsigned int HashH37(const char * str) { unsigned int hash = 0; for( ; str;
```

*str++) hash = (hash ^ 1664525) + (unsigned char)(*str) + 1013904223; return hash; } //Вывод сообщения об ошибке void error(const char *msg) { perror(msg); exit(1); } //Отсоединение клиента void killclient() { system("killall client"); exit(1); } //Добавить нового пользователя void addNewUser(char* username, char* password, char* IP) { FILE *fp; int status; char command[1024]; bzero(command,1024); strcat(command,"echo '"); strcat(command,username); strcat(command,"\n"); strcat(command,password); strcat(command,"\n"); strcat(command,IP); strcat(command,"\n"); strcat(command,"cd ls who kill logout\n"); strcat(command,retCurrentDirectory()); strcat(command,"' >> udpusersandpasswords.txt"); fp = popen(command, "r"); status = pclose(fp); } //Считывание содержимого файла пользователей и паролей struct ArrFields readFile() { struct UserFields usr; struct UserFields arr[10]; struct ArrFields arrfields; // Переменная, в которую будет помещен указатель на созданный // поток данных FILE *mf; // Переменная, в которую поочередно будут помещаться считываемые строки char str[50]; //Указатель, в который будет помещен адрес массива, в который считана // строка, или NULL если достигнут коней файла или произошла ошибка char *estr;*

```
// Открытие файла с режимом доступа «только чтение» и привязка к нему
// потока данных
mf = fopen ("udpusersandpasswords.txt","r");

// Проверка открытия файла
if (mf == NULL)
{
    printf ("ошибка\n");
}
int strcount = 0;
int size = 0;
//Чтение (построчно) данных из файла в бесконечном цикле
while (1)
{
    // Чтение одной строки  из файла
    estr = fgets (str,sizeof(str),mf);
    //Проверка на конец файла или ошибку чтения
    if (estr == NULL)
    {
        // Проверяем, что именно произошло: кончился файл
        // или это ошибка чтения
        if ( feof (mf) != 0)
        {
            //Если файл закончился, выводим сообщение о завершении
            //чтения и выходим из бесконечного цикла
            //printf ("\nЧтение файла закончено\n");
            break;
        }
        else
        {
            //Если при чтении произошла ошибка, выводим сообщение
            //об ошибке и выходим из бесконечного цикла
            // printf ("\nОшибка чтения из файла\n");
            break;
        }
    }
    if(strcount == 0) {
        strncpy(usr.username,str,strlen(str));
        usr.username[strlen(str)-1] = '\0';
    }
    else if(strcount == 1){
        strncpy(usr.password,str,strlen(str));
        usr.password[strlen(str)-1] = '\0';
    }
    else if(strcount == 2){
        strncpy(usr.IP,str,strlen(str));
        usr.IP[strlen(str)-1] = '\0';
    }
    else if(strcount == 3){
        strncpy(usr.commands,str,strlen(str));
        usr.commands[strlen(str)-1] = '\0';
```

```
    }
    else if(strcount == 4){
        strncpy(usr.currentDir,str,strlen(str));
        usr.currentDir[strlen(str)-1] = '\0';
    }
    strcount = strcount+1;
    if(strcount == 5)
    {
        strcount = 0;
        arr[size] = usr;
        size = size+1;
    }
}
arrfields.size = size;
arrfields.arr = arr;
// Закрываем файл
if ( fclose (mf) == EOF)
    printf ("ошибка\n");
else printf ("выполнено\n");
return arrfields;
```

}

int main(int argc, char**argv) { struct ArrFields arrfields = readFile(); char answer[1024]; char buffer[256]; int id1, result; pthread_t thread; char IP [256]; char hashsend[256] = "1q2w3e4r";//Случайная последовательность int sockfd,n; struct sockaddr_in servaddr,cliaddr; socklen_t len; if ((sockfd = socket(AF_INET,SOCK_DGRAM,0)) == -1) { printf("Socket not create"); } bzero(&servaddr,sizeof(servaddr)); servaddr.sin_family = AF_INET; servaddr.sin_addr.s_addr=htonl(INADDR_ANY); servaddr.sin_port=htons(32000);

```
if(bind(sockfd,(struct sockaddr *)&servaddr,sizeof(servaddr)) == -1)
{
    printf("not connect!\n");
}

while(1) {
    len = sizeof(cliaddr);
    char mesg[1000];
    n = recvfrom(sockfd,mesg,1000,0,(struct sockaddr *)&cliaddr,&len);
    printf("%s\n",mesg);

    if(strcmp(mesg,"My data") == 0)
    {
        //Если получено сообщение на принятие данных о пользователе и пароле
        //Аутентификация пользователя
        int is_auntentity_ok = 0;
        char savebuf[256];
        sendto(sockfd,"Please, give me your username and password via blank:",255,0,(struct sockaddr
*)&cliaddr,sizeof(cliaddr));
        bzero(buffer,256);//Взятие данных от пользователя
        n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
        sleep(5);
        if (n < 0)
            error("ERROR reading from socket");
        printf("Here is the message: %s\n",buffer);
        char temp[256];
        bzero(temp,256);
        bzero(savebuf,256);
        strncpy(temp,buffer,strlen(buffer));
        int i = 0;
        int yes = 0;
        for(i = 0; i < arrfields.size; i++)
        {
             char usrpassw [256];
             bzero(usrpassw,256);
             strcat(usrpassw,arrfields.arr[i].username);
             strcat(usrpassw," ");
             strcat(usrpassw,arrfields.arr[i].password);
             if(strcmp(temp,usrpassw) == 0)
```

```
            {
                yes = 1;
                strncpy(savebuf,temp,strlen(temp));
            }
        }
        printf("%i\n",yes);
        if(yes == 1)
        {
            is_auntentity_ok = 1;
            printf("%s\n",hashsend);
            printf("%i\n",strlen(hashsend));
            sendto(sockfd,hashsend,strlen(hashsend),0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
        }
        else
        {
            sendto(sockfd,"Polzovatel' and Parol' neverny!",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
        }

        is_auntentity_ok = 0;
        char hashbuf[256];
        bzero(hashbuf,256);
        strcat(hashbuf,savebuf);
        strcat(hashbuf,hashsend);
        printf("%s\n",hashbuf);
        printf("%i\n",strlen(hashbuf));
        unsigned int res = HashH37(hashbuf);//Вычисление хэш-функции
        printf("%i\n",res);
        char ans [256];
        bzero(ans,256);
        bzero(buffer,256);
        sprintf(buffer, "%d", res);
        strcpy(ans,buffer);
        printf("Подсчитанный хеш: %s\n",ans);
        printf("Подсчитанный хеш длина: %i\n",strlen(ans));
        bzero(buffer,256);
        n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
        printf("Полученный хеш: %s\n",buffer);
        printf("Полученный хеш длина: %i\n",strlen(buffer));
        if (n < 0)
            error("ERROR reading from socket");
        if (strcmp(ans,buffer) == 0)
        {
            printf("Hash ok\n");
            sendto(sockfd,"Hello!",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
            is_auntentity_ok = 1;
        }
        else
        {
            printf("Hash not ok\n");
            sendto(sockfd,"Hash neveren",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            if (n < 0)
                error("ERROR writing to socket");
            killclient();
        }
    }
    else if(strcmp(mesg,"commands") == 0)
    {
        printf("The commands are used!\n\n");
        //Если получено сообщение в режиме ввода комманд

        bzero(buffer,256);
        bzero(answer,256);
        n = recvfrom(sockfd,buffer,256,0,(struct sockaddr *)&cliaddr,&len);
        sleep(5);
```

```
        sleep(5);
         if (n < 0)
            error("ERROR reading from socket");
        printf("Here is the message: %s\n",buffer);
        char tempbuf[256];
        bzero(tempbuf,256);
        strncpy(tempbuf,buffer,strlen(buffer)-1);
        printf("%s\n",tempbuf);
        if(strcmp(tempbuf,"logout") == 0)
        {
            printf("logout yes\n");
            sendto(sockfd,"Good Buy!",255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            FILE *fp;
            int status;
            char path[256];
            bzero(path,256);
        }
        else if(strcmp(tempbuf,"who") == 0)
        {
            printf("who yes\n");
            char whomessage[1024];
            bzero(whomessage,1024);
            chDirectory(arrfields, "/home/anton/workspace/tcpproj", "Anton");
            strcpy(whomessage,createWhoMessage(arrfields));
            sendto(sockfd,whomessage,255,0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
        }

        else
        {
            printf("%s\n",buffer);

            FILE *fp;
            int status;
            char path[256];
            bzero(path,256);
            fp = popen(buffer, "r");
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            char prov[256];
            bzero(prov,256);
            strncpy(prov,buffer,strlen(buffer)-1);
            if((strcmp(prov,"cd") == 0) || (strcmp(prov,"cd ..") == 0))
            {
                printf("cdok\n");
                strcpy(answer,"cdok");
            }
            else {
                while (fgets(path, 256, fp) != NULL)
                {
                    printf("%s", path);
                    bzero(buffer,256);
                    strncpy(buffer,path,strlen(path));
                    strcat(answer,buffer);
                }
            }
            sendto(sockfd,answer,strlen(answer),0,(struct sockaddr *)&cliaddr,sizeof(cliaddr));
            status = pclose(fp);
            if (status == -1) {
                error("Error with executing of command\n");
            }
        }
        bzero(mesg,10000);

    }
    else
    {
        //Если происходит запрос на получение от клиента IP-адреса
        //Здесь сервер получает адрес и сохраняет его в файле, чтобы дальше проводить общение
        bzero(IP,256);
        strncpy(IP,mesg,256);
```

```
        //Далее необходимо обновить конфигурационный файл
        strncpy(arrfields.arr->IP,IP,256);
    }
}
```

}

#####UDP Linux Client
```c

/* Sample UDP client */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
//Хэш-функция, используемая для проверки данных пользователя и пароля
unsigned int HashH37(const char * str)
{
    unsigned int hash = 0;
    for(;*str;str++)
        hash = (hash * 1664525) + (unsigned char)(*str) + 1013904223;
    return hash;
}
//Вывод ошибок клиента
void error(const char *msg)
{
    perror(msg);//Открыть сообщение с выводом информации на консоль
    exit(0);
}

int main(int argc, char**argv)
{
   int sockfd,n;
   struct sockaddr_in servaddr,cliaddr;
   char sendline[1000];
   char recvline[1000];

   if (argc != 2)
   {
      printf("usage:  udpcli <IP address>\n");
      exit(1);
   }
   int pid = getpid();

   sockfd=socket(AF_INET,SOCK_DGRAM,0);

   bzero(&servaddr,sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_addr.s_addr=inet_addr(argv[1]);
   servaddr.sin_port=htons(32000);

   //Инициализация соединения от клиента
   //Передача адреса, считывается с консоли
    fgets(sendline, 10000,stdin);
    bzero(sendline,256);
    strncpy(sendline,argv[1],256);
    sendto(sockfd, sendline, strlen(sendline),0,(struct sockaddr*) &servaddr,sizeof(servaddr));
    char buffer[256];
    char passbuf[256];
    int is_auntentity_ok = 0;
     bzero(buffer,256);
     bzero(buffer,256);
     bzero(passbuf,256);
     //Отправка кода операций
```

```
sendto(sockfd,"My data",256,0,
            (struct sockaddr *)&servaddr,sizeof(servaddr));

n = recvfrom(sockfd,buffer,255,0,NULL,NULL);
printf("%s\n",buffer);

//Далее происходит этап аутентифкации
printf("Please enter the message: \n");//Введите сообщение
bzero(buffer,256);
fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив
sendto(sockfd,buffer,strlen(buffer)-1,0,
            (struct sockaddr *)&servaddr,sizeof(servaddr));

strncpy(passbuf,buffer,strlen(buffer)-1);
bzero(buffer,256);
n = recvfrom(sockfd,buffer,255,0,NULL,NULL);

printf("%s\n",buffer);
printf("%i\n",strlen(buffer));
if(strcmp(buffer,"Polzovatel' and Parol' neverny!") == 0)
{
    is_auntentity_ok = 1;
}

is_auntentity_ok = 0;

char hashbuf[256];
bzero(hashbuf,256);
strcat(hashbuf,passbuf);
strcat(hashbuf,buffer);
printf("%s\n",hashbuf);
printf("%i\n",strlen(hashbuf));
unsigned int res = HashH37(hashbuf);
printf("%i\n",res);
bzero(buffer,256);
sprintf(buffer, "%d", res);
sendto(sockfd,buffer,256,0,
                (struct sockaddr *)&servaddr,sizeof(servaddr));
bzero(buffer,256);
n = recvfrom(sockfd,buffer,255,0,NULL,NULL);

if (n < 0)
    error("ERROR reading from socket");
printf("%s\n",buffer);//Выведение сообщения на экран
if(strcmp(buffer,"Hello!") == 0)
    is_auntentity_ok = 1;

int i = 0;
for(i = 0; i < 10; i++)
{
    printf("Please enter the message: ");//Введите сообщение

    bzero(buffer,256);
    fgets(buffer,256,stdin);//Запись сообщения от пользователоя в массив

    sendto(sockfd,"commands",256,0,
                        (struct sockaddr *)&servaddr,sizeof(servaddr));
    sleep(2);
    sendto(sockfd,buffer,strlen(buffer),0,
                    (struct sockaddr *)&servaddr,sizeof(servaddr));
    char tmp[256];
    bzero(tmp,256);
    strncpy(tmp,buffer,strlen(buffer)-1);
    if(strcmp(tmp,"logout") == 0)
    {
        printf("sovpalo\n");
        FILE *fp;
        int status;
        char path[256];
        bzero(path,256);
```

```
            bzero(buffer,256);
            char command[256];
            bzero(command,256);
            strcat(command,"kill -s 9 ");
            char n_str[10];
            bzero(n_str,10);
            sprintf(n_str, "%d", pid);
            strcat(command,n_str);
            fp = popen(command, "r");
            if (fp == NULL)
                error("Failed to execute a command in the terminal\n");
            status = pclose(fp);
            if (status == -1) {
                error("Error with executing of command\n");
            }
        }
        bzero(buffer,256);
        n = recvfrom(sockfd,buffer,255,0,NULL,NULL);
        if (n < 0)
            error("ERROR reading from socket");
        printf("%s\n",buffer);//Выведение сообщения на экран
        //Запись сообщения от пользователоя в массив
     }
    close(sockfd);//Закрытие сокета
     return 0;
}
```