**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Операционные системы и сети

**Отчет по лабораторной работе № 3**

«Разработка сетевых приложений для работы с текстовым протоколом»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-357 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Акмуризин М.Э. |  |  |  |
| Принял | Ямилева А .М, |  |  |  |

**Уфа 2023**

**Цель работы:** ознакомиться c принципами разработки и функционирования сетевых приложений, использующих сокеты.

Для достижения цели необходимо выполнить следующие задачи:

1. Изучить необходимый теоретический материал.
2. Выполнить задания согласно 2 варианту.

**Задание:** Создайте серверное и клиентское приложения, выполняющие следующие действия:

* Серверное приложение садится на определенный порт и ждет соединений по этому порту. Порт должен задаваться при запуске серверной части в командной строке приложения. В случае, если порт не указан – сервер должен садиться на определенный порт  
  по умолчанию;
* Клиентское приложение, которое соединяется с серверным приложением по заданному в командной строке порту, передает вводимые пользователем команды (в соответствии с вариантом задания)  
  и выводит на экран полученные ответы;
* При удачном подключении клиентское приложение соединяется с сервером, передает вводимые пользователем команды и выводит на экран полученные ответы. Обмен данными между серверным и клиентским приложением должен осуществляться в соответствии с текстовым протоком и вариантом задания.
* Необходимо предусмотреть корректную обработку ошибок работы  
  с протоколом UDP и корректную работу при неправильно введенных данных.

*Описание текстового протокола:*

При запуске сервера выводится сообщение на каком порте он слушает сообщения. Сервер воспринимает следующие команды в любой последовательности:  
0,1,2,3,4.

Описание команд:

0 — LOGIN

Эта команда авторизует пользователя. Ему присваивается уникальный uuid ключ, по которому прописываются логи в отдельный файл «log.txt». В случае повторной авторизации программа предупредит пользователя и в файл с логами запишется ошибка. Если авторизации увенчалась не успехом, пользователь может повторно ввести пароль при вызове команды.

1- LIST

Эта команда выводит список доступный команд. В случае если пользователь не авторизован на сервере придет сообщение о ограничение прав, и в файл с логами запишется ошибка.

2- GET\_TASK1

Данная команда вызывает логику первого задания. Ввод координаты и получение соответствующих данных после введения координаты. В случае если пользователь не авторизован на сервере придет сообщение о ограничение прав, и в файл с логами запишется ошибка.

3 -GET\_TASK2

Данная команда вызывает логику второго задания. А именно получения соответствующих данных после вызова команды. В случае если пользователь не авторизован на сервере придет сообщение о ограничение прав, и в файл с логами запишется ошибка.

4- LOGOUT

Данная команда завершает работу пользователя на сервере. В случае если пользователь не авторизован придет сообщения о попытке , завершить не начавшийся сеанс.

**Теоретическая часть**

***Разработка сервера***

Основные действия, которые должна выполнить программа-сервер:

* создать сокет (программное гнездо)
* дать сокету имя (адрес)
* объявить сокет могущим принимать соединения
* принять соединение
* по окончании работы закрыть сокет

***Создание сокета***

Эта операция производится посредством вызова функции *socket()*, имеющей следующий прототип:

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_002.png

Функция возвращает целое положительное число, называемое дескриптором сокета, вполне аналогичное дескриптору файла. При ошибке: -1. Дескриптор нужно запомнить для последующих операций над сокетом. Первый параметр задает семейство протоколов, по которым будет происходить сетевое общение. Он может принимать, в частности, следующие значения:

* AF\_UNIX, AF\_LOCAL - протокол Unix для локального взаимодействия
* AF\_INET - IP версии 4
* AF\_INET6 - IP версии 6
* AF\_IPX - протоколы IPX, используемые в сетях Novell

Сокет имеет тип, указанный вторым параметром. Тип сокета определяет семантику взаимодействия и может принимать следующие значения:

* SOCK\_STREAM - обеспечивает надежный двусторонний обмен потоками байтов, основанный на установлении соединения. При этом гарантируется правильный порядок байтов (не в смысле сетевого порядка байтов), то есть байты будут приняты в том порядке, в каком они были посланы. Например, для семейства AF\_INET это фактически означает использование протокола транспортного уровня TCP.
* SOCK\_DGRAM - ненадежный обмен на основе передачи датаграмм без установления соединения. Например, для семейства AF\_INET это означает использование протокола транспортного уровня UDP.
* SOCK\_SEQPACKET - обеспечивает основанный на установлении соединения надежный упорядоченный двусторонний обмен датаграммами фиксированного максимального размера. От получателя требуется, чтобы от читал весь пакет целиком за один системный вызов.
* SOCK\_RAW - непонятно
* SOCK\_RDM - надежная передача датаграмм, но без гарантии упорядочения.

Следует заметить, что может оказаться так, что в рамках данного семейства протоколов реализованы не все типы сокетов.

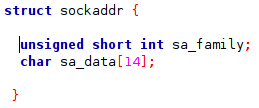
Третий параметр указывает номер конкретного протокола в рамках указанного семейства для указанного типа сокета. Как правило, существует единственный протокол для каждого типа сокета внутри каждого семейства, однако, их может быть и больше. В таких случаях для получения информации о протоколах можно воспользоваться функциями getprotobyname(), getprotobynumber() или группой setprotoent(), getprotoent(), endprotoend().

***Именование сокета***

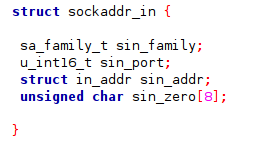
После создания сокет еще не способен принимать и посылать данные, так как он, хотя уже и существует в определенном пространстве имен, но имени пока не имеет. Для именования сокета используется функция bind() со следующим прототипом:

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_004.png

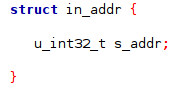
При успешном завершении функция возвращает 0, при ошибке - -1. Первый параметр представляет собой дескриптор сокета, возвращенный функцией socket(). Второй - это то имя (локальный адрес), который мы хотим дать сокету. Третий - длина второго параметра в байтах. Форматы адресов различаются для различных семейств протоколов и различных семейств адресов. Структура sockaddr- это 'родовая' (generic) структура, которая выглядит вот так



Здесь sa\_family - семейство адресов (не путать с семейством протоколов), а массив sa\_data - данные об адресе сокета, специфичные для конкретного семейства адресов. Поле sa\_family - общее для всех семейств протоколов и адресов (правда, оно может по разному называться). Рассмотрим в качестве примера формат структуры sock\_addr для семейства протоколов AF\_INET:

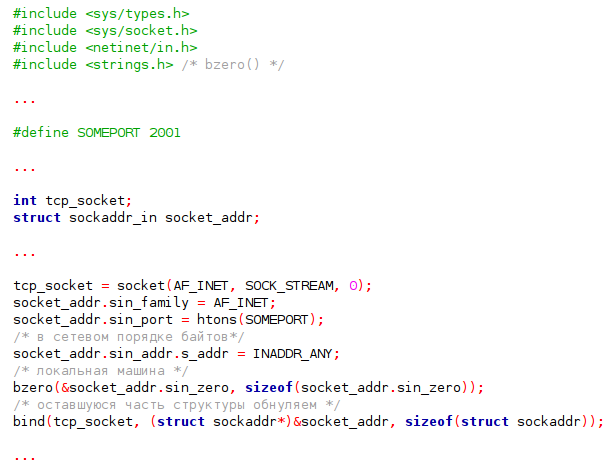


Тип sa\_family\_t эквивалентен типу unsigned short int. Структура in\_addr состоит всего из одного элемента и имеет следующий формат



Поле sin\_family может принимать только одно значение, а именно AF\_INET. Следующее поле в структуре sockaddr\_in - это TCP-порт. Поле s\_addr, входящее в состав структуры in\_addr - IP-адрес. Наконец, массив sin\_zero дополняет структуру sockaddr\_in до размера структуры sock\_addr. Отметим, что sin\_port и s\_addr хранятся в сетевом порядке байтов.

Таким образом, для именования созданного TCP-сокета программа должна заполнить структуру sockaddr\_in и вызвать функцию bind(), передав ей указатель на эту структуру вторым параметром:



***Переключение сокета в режим прослушивания***

Как правило, программы-серверы исполняются в фоновом режиме, ожидая соединений (слушая сеть) от программ-клиентов. После создания и именования сокета он еще не готов принимать соединения от клиентов. Чтобы программа-сервер могла это делать, сокет нужно перевести в прослушивающий режим, что осуществляется посредством вызова функции listen(), имеющей следующий прототип:

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_009.png

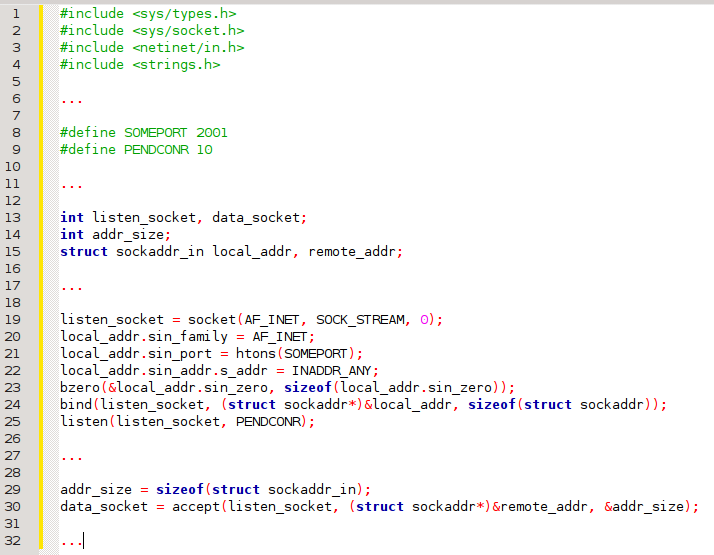
При успешном завершении эта функция возвращает 0, при ошибке -1. Первый параметр - дескриптор сокета, второй - максимальное число соединений в очереди соединений, ожидающих обработки.

***Принятие запросов на соединение от клиентов***

В ответ на попытку установления соединения со стороны клиента сервер должен принять это соединение. Это делается с помощью функции accept(), прототип которой выглядит так:

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_011.png

Первый параметр - это дескриптор сокета, который должен быть связан с именем посредством функции bind() и переведен в прослушивающий режим вызовом функции listen() до вызова accept(). Функция accept() выполняет следующие действия: извлекает из очереди соединений, ожидающих обработки, первый запрос и создает новый сокет с такими же свойствами, как и sockfd/ Если в момент вызова accept() в очереди не было запросов на соединение, то поведение функции зависит от того, в каком режиме находится сокет, блокирующем или неблокирующем. В первом случае программа блокируется до прихода запроса на соединение, во втором функция accept()возвращается с ошибкой ( errno будет EWOULDBLOCK или EAGAIN ). Функция возвращает дескриптор вновь созданного сокета, который нужно использовать для обмена данными, но нельзя для приема соединений. Первоначальный сокет sockfd остается открытым и служит для принятия последующих соединений на этом порту. Второй параметр заполняется самой функцией и по завершении вызова будет содержать информацию об адресе того, кто присоединился (имя удаленного сокета). Третий параметр одновременно является как входным, так и выходным. При вызове он должен содержать размер объекта, на который показывает указатель remote\_addr, а по завершении он будет содержать фактическую длину адреса.



***Чтение из сокета***

Прием данных из сети можно осуществлять посредством функций recvfrom(), recvmsg(), recv() и read(). Первые две функции можно использовать для чтения данных вне зависимости от того, является ли сокет ориентированным на соединение или нет. Вторые две используются для приема данных из сокета, ориентированного на соединение. Функция read() - это обычная функция чтения, с помощью которой мы читаем из файлов и т.п. По сравнению с ней функция recv() ориентирована на работу исключительно с сокетами и обладает более богатыми возможностями. Рассмотрим подробнее функцию recv(). Она имеет следующий прототип

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_013.png

и возвращает при успешном завершении число прочитанных байт, а при ошибке - -1, при этом, как обычно, переменная errno принимает соответствующее значение. Первый параметр функции - сокет, из которого нужно прочитать данные, второй - указатель на область памяти, в которую нужно записать принятые данные, третий - сколько байт читать. С помощью четвертого параметра можно управлять поведением функции. Например, указав в качестве флага MSG\_PEEK, мы прочитаем данные из начала очереди, но после чтения они останутся в очереди. Разные флаги можно комбинировать, объединяя соответствующие константы посредством операции побитного ИЛИ. Отметим, что по умолчанию только что созданный сокет является блокирующим. В отношении функции recv() это означает, что если в момент ее вызова данных нет, она блокируется до тех пор, пока они не придут из сети.

***Запись в сокет***

Посылку данных в сеть можно осуществлять посредством функций sendto(), sendmsg(), send() и write(). Первые две функции можно использовать для записи данных вне зависимости от того, является ли сокет ориентированным на соединение или нет. Вторые две используются для записи данных в сокет, ориентированный на соединение. Функция write() - это обычная функция записи, с помощью которой мы пишем в файлы и т.п. По сравнению с ней функция send() ориентирована на работу исключительно с сокетами и обладает более богатыми возможностями. Рассмотрим подробнее функцию send(). Она имеет следующий прототип

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_014.png

и возвращает при успешном завершении число записанных байт, а при ошибке - -1, при этом, как обычно, переменная errno принимает соответствующее значение. Первый параметр функции - сокет, в который нужно записать данные, второй - указатель на область памяти, из которой нужно взять данные, третий - сколько байт записать. С помощью четвертого параметра можно управлять поведением функции. Например, указав в качестве флага MSG\_DONTROUTE, мы заставим TCP/IP посылать данные в обход обычных средств маршрутизации непосредственно на сетевой интерфейс получателя, что используется, например, различными диагностическими программами и маршрутизаторами. Разные флаги можно комбинировать, объединяя соответствующие константы посредством операции побитного ИЛИ.

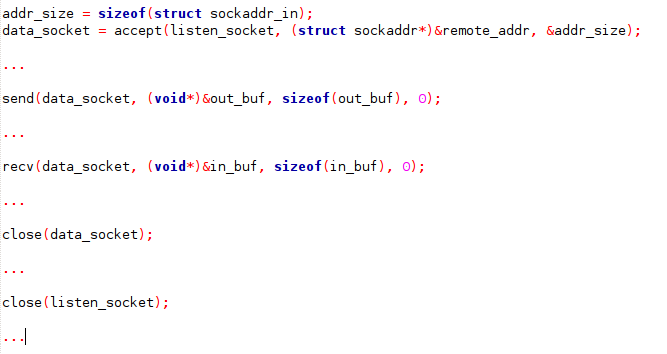
***Закрытие сокета***

После окончания обмена данными программа должна закрыть сокет(ы), вызвав функцию close(). Прототип этой функции описан в файле unistd.h и имеет вид

http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_015.png

Функции нужно передать дескриптор сокета, который нужно закрыть. При успешном завершении функция возвращает 0, при ошибке - -1.

В нижеследующем примере демонстрируется использование всех описанных выше функций работы с сокетом.



***Разработка клиента***

Минимальный набор действий для клиента:

* создать сокет
* установить соединение
* по окончании работы закрыть сокет

Создание сокета описано выше. Чтобы обмениваться данными с сервером, клиент должен установить с ним соединение.

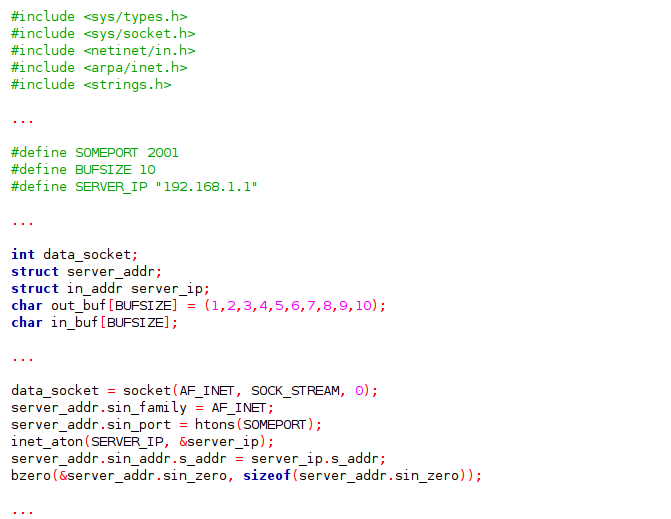
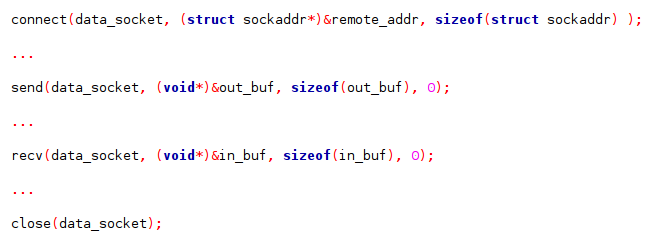
**Установление соединения**

Данная операция осуществляется посредством вызова функции connect(), имеющей следующий прототип

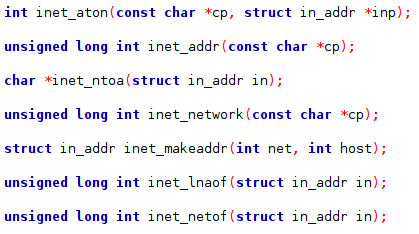
http://bikmeyev-at.ugatu.su/students/OSiS/labSocketBase/Pics/lab09_018.png

При успешном завершении функция возвращает 0, при ошибке - -1. Первый параметр функции - дескриптор сокета, возвращенный вызовом socket(), второй - указатель на структуру, содержащую адрес удаленного сокета, которую нужно заполнить перед вызовом connect() и третий - длина структуры, на которую указывает remote\_addr в байтах. Отметим, что вызов bind() не является необходимым для клиента, так как назначение порта сокету функция connect() сделает сама, а клиенту, вообще говоря, все равно, какой у него порт.

Ниже приведены фрагменты кода, содержащие основные шаги, которые должна выполнить программа-клиент для того, чтобы обменяться данными с программой-сервером.

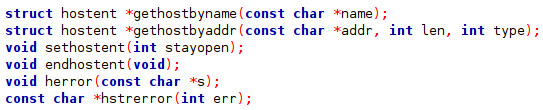
В этом отрывке используется не описанная ранее функция inet\_aton(). Эта функция наряду с другими применяется для манипуляций с IP-адресами. Прототипы всех таких функций определены в заголовочном файле arpa/inet.h следующим образом:



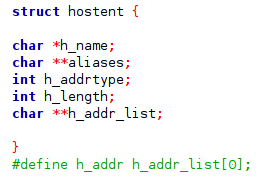
* Функция inet\_aton() преобразует IP-адрес, задаваемый первым аргументом из стандартной формы в виде десятичных чисел, разделенными точками в бинарную в сетевом порядке байтов. При успешном завершении возвращается ненулевое значение, а если адрес неправильный, то 0. Результат преобразования помещается в структуру, на которую указывает второй параметр.
* Функция inet\_ntoa() выполняет обратное преобразование, то есть преобразует IP-адрес, заданный в двоичном виде в сетевом порядке байтов в стандартную форму числа-точки. Результат хранится в статическом буфере (возвращается указатель), поэтому при последующих вызовах он будет переписан.
* Функция inet\_addr() преобразует IP-адрес, задаваемый первым аргументом из стандартной формы в виде десятичных чисел, разделенными точками в бинарную в сетевом порядке байтов. При успешном завершении возвращается результат преобразования, в противном случае INADDR\_NONE (-1). Эта функция устаревшая, поскольку -1=255.255.255.255 представляет собой корректный IP-адрес. Поэтому пользуйтесь inet\_aton().
* Функция inet\_network() извлекает из IP-адреса в стандартной текстовой записи числа-точки номер сети в двоичном виде в локальном порядке байтов. При некорректном адресе возвращается -1.
* Функция inet\_makeaddr() составляет из номера сети net и номера узла host, заданных в локальном порядке байтов, IP-адрес в сетевом порядке байтов.
* Функция inet\_lnaof() извлекает из IP-адреса, заданного ее аргументом, часть, соответствующую узлу (в локальном порядке байтов).
* Функция inet\_netof() извлекает из IP-адреса, заданного ее аргументом, часть, соответствующую сети (в локальном порядке байтов).

***Работа со службой доменных имен***

В приведенном выше отрывке IP-адрес узла, с которым мы хотим установить соединение, был жестко задан в тексте программы. Понятно, что программа, которая умеет соединяться только с одним узлом в сети, большой ценности не представляет. Как правило, адрес узла задает пользователь, причем он может задать его как в виде IP-адреса в стандартной записи, так и в виде символического имени. Для получения информации об узлах сети имеется группа функций, прототипы которых определены в файле netdb.h и имеют следующий вид:



Структура hostent, определенная также в заголовочном файле netdb.h, имеет следующий формат:



* Первое поле структуры - официальное имя узла;
* Второе - массив альтернативных имен (псевдонимов) узла;
* Третье - тип адреса узла (в настоящее время всегда AF\_INET);
* Поле h\_length - длина адреса в байтах;
* Поле h\_addr\_list - массив адресов узла в сетевом порядке байтов, заканчивающийся нулем.

Функция gethostbyname() возвращает указатель на структуру hostent для узла, указанного ее единственным параметром, который может быть доменным именем, IPv4-адресом в стандартной записи или IPv6-адресом. Память под структуру выделять не надо, достаточно объявить указатель. Сама структура хранится в памяти ядра операционной системы. Отметим, что если параметр nameпредставляет собой IPv4 или IPv6 адрес, то процедура разрешения адреса (обращение к DNS и т.п.) не выполняется, а name просто копируется в поле h\_name структуры hostent. Если нужно получить доменное имя по IP-адресу, используйте функцию gethostbyaddr().

Функция gethostbyaddr() возвращает указатель на структуру hostent для узла, адрес которого (в двоичной форме в сетевом порядке байтов) указан первым параметром функции. Второй параметр задает длину адреса в байтах, а третий - тип адреса. Единственно допустимый тип адреса в настоящее время - AF\_INET.

Для получения информации об узлах сети функции gethostbyname() и gethostbyaddr() используют комбинации следующих методов: обращение к службе доменных имен (DNS), поиск по файлу /etc/hosts и обращение к службе сетевой информации (NIS) в порядке, определяемом содержимым строки order в файле /etc/host.conf.

Функция sethostent() (если stayopen равно 1) указывает, что для обращений к DNS нужно использовать соединение и что это соединение не должно закрываться между последовательными запросами. Если же stayopen равно 0 (FALSE), то запросы к DNS будут выполняться с использованием UDP датаграмм.

Функция endhostent() заканчивает использование TCP соединения для выполнения запросов к DNS.

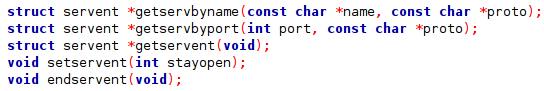
Функция herror() выводит сообщение об ошибке, соответствующее текущему значению переменной h\_errno на стандартное устройство вывода для сообщений об ошибках. Тут надо пояснить, что описываемые функции вместо переменной errno для хранения текущего значения номера ошибки используют переменную h\_errno.

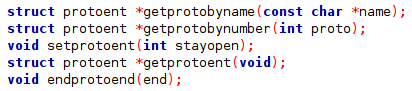
Функция hstrerror() принимает в качестве параметра номер ошибки (обычно h\_errno) и возвращает строку, содержащую соответствующее сообщение.

***Получение информации о стандартных сетевых службах и протоколах***

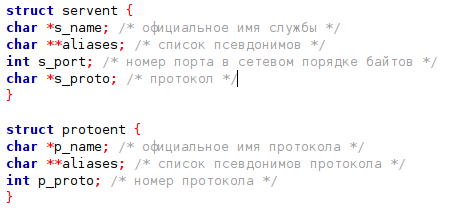
В комплект программной системы, составляющей стек TCP/IP, входят базы данных по стандартным сетевым службам и протоколам. Базы представляют собой текстовые файлы, содержащие информацию об именах, псевдонимах и портах различных сетевых служб и номерах и именах протоколов, соответственно. В операционных системах Unix эти файлы, как правило, называются /etc/services и /etc/protocols. Для получения информации из них имеется ряд функций, прототипы которых определены в заголовочном файле netdb.h следующим образом:

(функции для работы со службами)



(функции для работы с протоколами)

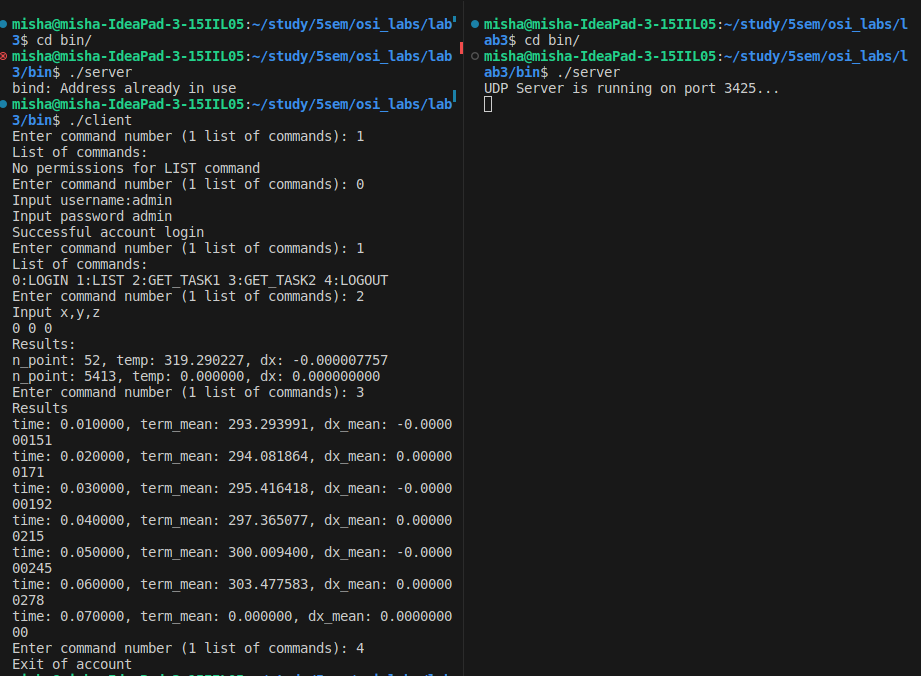
Структуры servent и protoent определены в том же файле следующим образом:

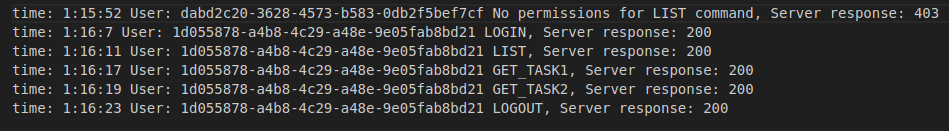


* Функция getservbyname() возвращает указатель на структуру servent, содержащую информацию из файла /etc/services о службе, имя которой совпадает с именем, указанным первым параметром функции, а протокол - с именем протокола, указанным ее вторым параметром. Если заданная служба не существует, возвращается константа NULL.
* Функция getservbyport() делает то же самое, но в первым параметром ей нужно передать номер порта в сетевом порядке байтов.
* Функция setservent() открывает файл /etc/services и устанавливает указатель файла на начало. Если при этом параметр stayopen имеет значение 1, файл не будет закрываться вызовами getservbyname()и getservbyport().
* Функция getservent() читает очередную строку из файла /etc/services, заполняет структуру serventсоответствующей информацией и возвращает указатель на эту структуру (NULL в случае, если произошла ошибка или достигнут конец файла).
* Функция endservent() закрывает файл /etc/services.
* Функции для работы с протоколами работают вполне аналогичным образом.

**Практическая часть**

Работа программы:

Логи соответствующее данному примеру работы программы:



**Заключение**

В результате проделанной лабораторной работы были изучены основы работы в сетевых приложениях, написан текстовый протокол, модифицированы серверное и клиентское приложения из лабораторной работы №3.

**Список литературы**

1. Жиганов Е.Д. "Разработка сетевых приложений на основе протокола TCP/IP в среде Unix-подобных операционных систем"

**Приложение**

https://github.com/Mis-prog/\_osi\_5sem/tree/lab3