Лабораторная работа №4

Реализация собственного протокола поверх TCP/IP

Цель работы: разработать и реализовать протокол для решения конкретной задачи.

Набор протоколов *TCP/IP* так включает много прикладных протоколов, но буквально ежедневно появляются все новые и новые протоколы. По существу, создание двух не встречавшихся ранее программ, в которых для связи применяются протоколы *TCP/IP,* равносильно изобретению нового прикладного протокола. Безусловно, некоторые прикладные протоколы описаны, стандартизированы и включены в состав специально утвержденного набора протоколов *TCP/IP.* Такие протоколы принято называть *стандартными прикладными протоколами*. Другие протоколы, разработанные прикладными программистами для частного использования, называются *нестандартными прикладными протоколами*.

В основе сервиса *TCP* лежат так называемые сокеты (гнезда или конечные точки), создаваемые как отправителем, так и получателем. Они обсуждались в предыдущих лабораторных работах. У каждого сокета есть номер (адрес), состоящий из *IP*-адреса хоста и 16-битного номера, локального по отношению к хосту, называемого портом. Портом в *TCP* называют *TSAP*-адрес. Для обращения к сервису *TCP* между сокетом одной машины и сокетом другой машины должно быть явно установлено соединение.

Большинство сетевых администраторов стремятся по мере возможности использовать стандартные прикладные протоколы; не имеет смысла изобретать новый прикладной протокол, если все необходимое предоставляет уже существующий протокол. Например, набор протоколов TCP/IP включает стандартные прикладные протоколы для таких служб, как передача файлов, дистанционный доступ к данным и электронная почта. Поэтому программисту следуют использовать для таких служб стандартный протокол.

Методические указания

Разработаем протокол отправки SMS-сообщений на SMS-центр с временным ограничением. Это означает, что пользователь услуги имеет право удалить отправленное на центр сообщение из очереди, если не прошла минута со времени его отправления.

***СЕРВЕРНАЯ часть**:*

Для разработки данного протокола сначала необходимо решить, какая структура данных будет играть роль SMS-центра. Очевидно, это будет файловая структура данных. Для представления файла удобно использовать класс *CFile* (предоставляет интерфейс для универсальных операций двоичного файла) из библиотеки классов *MFC*. Для этого необходимо

подключить модуль “*afx.h*”, а также выбрать опцию *Using MFC in a shared DLL* (Alt+F7, вкладка *General*).

Для управления временем используем тип данных *clock_t* (для хранения значения времени) и функцию *clock_t clock (void);* для запуска и остановки таймера. Функция *clock()* возвращает количество временных тактов, прошедших с начала запуска программы (в случае ошибки, функция возвращает значение -1). С помощью макроса *CLOCKS_PER_SEC* функция получает количество пройденных тактов за 1 секунду. Таким образом, зная сколько выполняется тактов в секунду и время запуска программы, можно посчитать время работы всей программы или отдельного её фрагмента. Эти возможности реализованы в библиотеке “*time.h*”.

```
#include <iostream>
#include "afx.h"
#include <winsock2.h>
#include <process.h>    /* _beginthread, _endthread */
#include <string.h>
#include <time.h>
Using namespace std;

CFile f;
CFileException ex;
clock_t start, finish;

// удалить сообщение с номером
void del(char* p,int n){
     char tel[200];
     int j=0;
     for (int i=n;p[i]!=' ';i++){
          tel[j]=p[i];
          j++;
     }
     tel[j]='\0';
     char fName[200];
     fName[0]='\0';
     strcat(fName,tel);
     DeleteFile((LPCSTR)fName);
     cout<<"SMS is removed"<<endl;}

//Отослать сообщение на SMS-центр для номера
void sms(char* p,int n){
     char tel[200];
     char str[200]; int j=0;
     for (int i=n;p[i]!=' ';i++){
          tel[j]=p[i];
          j++;
     }
     tel[j]='\0';
     n=j+1;j=0;
     for(i=n;p[i];i++){
          str[j]=p[i];
```

```
            j++;
        }
        str[j]='\0';
        char fName[200];
        fName[0]='\0';
        strcat(fName,tel);
        cout<<"sms processing...";
        if(!f.Open(fName,CFile::modeWrite | CFile::modeCreate,&ex)){
            cerr<<"SMS storage error. Try again\n";
            exit(EXIT_FAILURE);
        }
        f.Write(str,strlen(str));
        cout<<"sent successfully"<<endl;}

    void SMSworking(void* newS){
        int c;
        char p[200], com[200];
        com[0]='\0';p[0]='\0';
        strcat(p,"SMS center connected...\n");
        send((SOCKET)newS,p,sizeof(p),0);
        while ((c=recv((SOCKET)newS,p,sizeof(p),0)!=0)) {
            int i=0;
            while (p[i]!=' '){
                com[i]=p[i];
                i++;
            };
            com[i]='\0';i++;
            if (!strcmp(com,"sms")) {
                start = clock();
                sms(p,i);
                com[0]='\0';
            }
            if (!strcmp(com,"del")) {
                finish = clock();
        //если с отправки сообщения прошло больше минуты
            if((double)(finish - start) / CLOCKS_PER_SEC > 60 )
                        cout<<"Cannot be canceled"<<endl;
                else
                del(p,i);
                com[0]='\0';
            }
            if (!strcmp(com,"quit")) {
                closesocket((SOCKET)newS);
                exit(EXIT_SUCCESS);
                com[0]='\0';}}}

    int main(){

        WORD wVersionRequested;
        WSADATA wsaData;
        int err;
        wVersionRequested = MAKEWORD(2,2);
        err = WSAStartup(wVersionRequested, &wsaData);
```

```
     if(err != 0) return -1;
     sockaddr_in local;
     local.sin_family = AF_INET;
     local.sin_port = htons(1280);
     local.sin_addr.s_addr = htonl(INADDR_ANY);
     SOCKET s = socket(AF_INET, SOCK_STREAM,0);
     int c=bind(s,(struct sockaddr*)&local,sizeof(local));
     int r=listen(s,5);
while(true){

     sockaddr_in remote;
     int j = sizeof(remote);
     SOCKET newS = accept(s, (struct sockaddr*) &remote, &j);

     _beginthread(SMSworking,0,(void*)newS);
     }
     WSACleanup();

     return 0;
}
```

В ***КЛИЕНТСКОЙ*** части необходимо позаботиться о возможности считывания и отправки команд на сервер:

```
#include <winsock2.h>
#include <iostream.h>

int main(){
     WORD wVersionRequested;
     WSADATA wsaData;
     int err;
     wVersionRequested = MAKEWORD(2,2);
     err = WSAStartup(wVersionRequested, &wsaData);
     if(err != 0) return -1;
     struct sockaddr_in peer;
     peer.sin_family=AF_INET;
     peer.sin_port=htons(1280);
     peer.sin_addr.s_addr=inet_addr("127.0.0.1");
     SOCKET s=socket(AF_INET,SOCK_STREAM,0);
     connect(s,(struct sockaddr*) &peer,sizeof(peer));

     char b[200], buf[200];
     recv(s,b,sizeof(b),0);
     cout<<b;
     b[0]='\0';
while (strcmp(b,"quit")) {
          cin.getline(b,200,'\n');
          send(s,b,sizeof(b),0);
     }

     WSACleanup();
     return 0;
}
```

Сначала запустим серверную часть, потом клиентскую. Если соединение произойдет успешно, появится строка "*SMS center connected...*". Для отправки сообщения необходимо набрать

sms <номер телефона> <сообщение>

Для удаления сообщения необходимо набрать

del <номер телефона>

Для выхода из программы наберите *quit* .

Контрольные вопросы

1. Какие протоколы называются стандартными прикладными протоколами; нестандартными?
2. Всегда ли оправдано использование собственных протоколов?

Варианты индивидуального задания

1. Реализовать простейший почтовый сервер и протокол взаимодействия с ним. Предусмотреть команды авторизации пользователя и отправки писем.
2. Реализовать простейший почтовый сервер и протокол взаимодействия с ним. Предусмотреть команды авторизации пользователя и удаления отправленных писем.
3. Реализовать простейший почтовый сервер и протокол взаимодействия с ним. Предусмотреть команды авторизации пользователя и удаления входящих писем.
4. Реализовать простейший почтовый сервер и протокол взаимодействия с ним. Предусмотреть команды отправки и чтения входящих писем.
5. Реализовать простейший почтовый сервер и протокол взаимодействия с ним. Предусмотреть команды чтения входящих писем и удаления отправленных.
6. Доработать протокол отправки SMS-сообщений таким образом, чтобы сообщения, предназначавшиеся для одного номера, записывать в один и тот же файл. В этом случае удалять только последнее сообщение для конкретного номера.
7. В протоколе отправки SMS-сообщений ограничить частоту отправки сообщений с одного IP адреса по времени. Добавить команду sms (без параметров), которая выводила б правила пользования услугой.
8. В протоколе отправки SMS-сообщений ограничить частоту отправки сообщений для одного номера по времени. Добавить команду del (без параметров), которая удаляла б все отправленные сообщения.
9. Доработать протокол отправки SMS-сообщений таким образом, чтобы отправленные сообщения отсылались на телефон (удалялись с центра) ежеминутно. Обеспечить возможность получения клиентом отчетов об отправке.

10. Доработать протокол отправки SMS-сообщений таким образом, чтобы удалять можно было только разрешенные для удаления сообщения. Разрешения присваивает отправитель дополнительным параметром в команде sms.

ПРИЛОЖЕНИЕ

Стек TCP/IP. Протоколы прикладного уровня.

TCP/IP – Transmission Control Protocol / Internet Protocol (Протокол Управления Передачей Данных / Межсетевой Протокол). Стек *TCP/IP* – совокупность протоколов организации взаимодействия между структурами и программными компонентами сети; представляет собой программно реализованный набор протоколов межсетевого взаимодействия.

<table>
<tr><th>OSI</th><th>TCP/IP</th><th></th></tr>
<tr><td>7 Прикладной уровень</td><td rowspan="3">Прикладной уровень</td><td rowspan="3">IV</td></tr>
<tr><td>6 Уровень программирования</td></tr>
<tr><td>5 Сеансовый уровень</td></tr>
<tr><td>4 Транспортный уровень</td><td>Транспортный уровень</td><td>III - TCP/IP, UDP</td></tr>
<tr><td>3 Сетевой уровень</td><td>Межсетевое взаимодействие</td><td>II - IP</td></tr>
<tr><td>2 Канальный уровень</td><td rowspan="2">Уровень сетевого интерфейса (физический, физического интерфейса)</td><td rowspan="2">I</td></tr>
<tr><td>1 Физический уровень</td></tr>
</table>

I-й уровень должен обеспечить интеграцию в составную сеть любой другой сети, независимо от технологии передачи данных этой сети.

II-й уровень должен обеспечить возможность передачи пакетов через составную сеть, используя разумный (оптимальный) на данный момент маршрут.

III-й уровень решает задачу обеспечения надежной передачи данных между источником и адресатом.

IV-й уровень объединяет все сетевые службы и услуги, предоставляемые сетью пользователю.

В TCP/IP достаточно хорошо развит **первый уровень**, соответствующий 1 и 2 уровням OSI.

Второй уровень *TCP/IP* представляет протокол *IP*. Также присутствует *ICMP (Internet Control Message Protocol)* – протокол управляющих сообщений сети *Internet*. Уровень *IP* не гарантирует правильную доставку дейтаграмм и не накладывает ограничений на скорость их отправки. Основная задача уровня *IP* – выбор наилучшего маршрута. Решение этой задачи *IP* перекладывает на *RIP* и *OSPF* протоколы.

Третий уровень – *TCP*, основная функция – надежность и правильность доставки данных. Именно уровню *TCP* приходится выбирать правильную скорость отправки (следуя целям эффективного использования пропускной способности и предотвращения перегрузок), следить за истекшими интервалами ожидания и в случае необходимости заниматься повторной передачей дейтаграмм, не дошедших до места назначения. Бывает, что дейтаграммы прибывают в неправильном порядке. Восстанавливать сообщения из таких дейтаграмм обязан также уровень *TCP*. Таким образом, протокол *TCP* призван обеспечить хорошую производительность и

надежность, о которой мечтают многие приложения и которая не предоставляется протоколом *IP*.

На третьем уровне также используется протокол *UDP*, в нем каждый пакет передается независимо. Надежность доставки данных не гарантируется, т.к. не устанавливается связь заранее. Обычно по *UDP* передаются данные, не критичные к надежности.

Четвёртый уровень (прикладной) – набор служб и услуг, предлагаемых пользователю.

ПРОТОКОЛЫ ПРИКЛАДНОГО УРОВНЯ:

1)*Telnet* – протокол удаленного доступа (эмуляция терминала). Обеспечивает подключение пользователя за неинтеллектуальным терминалом (используется крайне редко)

2)*FTP* – протокол передачи данных

3)*SMTP* – протокол передачи электронной почты

4)*POP3* – почтовый протокол

5)*DNS* – протокол доменных имен. Устанавливает соответствие между символьным адресом компьютера и его *IP* адресом.

6)*HTTP* – протокол передачи гипертекста

7)*Kerberos* – протокол защиты информации в сетях. Отвечает за пароли и ключи.

TELNET

Telnet – это прикладной протокол стека *TCP/IP*, обеспечивающий эмуляцию терминалов. Терминал – это устройство, состоящее из монитора и клавиатуры и используемое для взаимодействия с хост-компьютерами (обычно мэйнфреймами или мини-компьютерами), на которых выполняются программы. Программы запускаются на хосте, поскольку терминалы, как правило, не имеют собственного процессора.

Протокол *Telnet* функционирует поверх *TCP/IP* и имеет две важные особенности, отсутствующие в других эмуляторах. Протокол *Telnet* присутствует практически в каждой реализации стека TCP/IP, а также является открытым стандартом (т. е. каждый производитель или разработчик легко может, реализовать его). Для некоторых реализаций Telnet нужно, чтобы хост был сконфигурирован как Telnet-сервер. Протокол Telnet поддерживается многими рабочими станциями, работающими под управлением MS-DOS, UNIX и любых версий Windows.

File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP), Network File System (NFS)

Стек TCP/IP содержит три протокола для передачи файлов: File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP) и Network File System (NFS). Самым распространенным протоколом является FTP, поскольку именно его чаще всего выбирают для передачи файлов пользователи Интернета. С помощью FTP можно, работая на компьютере в одном городе, подключиться к хост-компьютеру, расположенному в другом городе, и скачать один или несколько файлов. (При этом, конечно, нужно знать имя учетной записи и пароль для удаленного хоста.) Пользователи

Интернета нередко с помощью FTP скачивают различные файлы (например, сетевые драйверы или обновления системы).

FTP – это приложение, позволяющее с помощью протокола TCP передать данные от одного удаленного устройства к другому. Как и в протоколе Telnet, заголовок FTP и соответствующие данные инкапсулируются в поле полезной нагрузки пакета TCP. Преимущество FTP по сравнению с протоколами TFTP и NFS заключается в том, что FTP использует два TCP-порта: 20 и 21. Порт 21 – это управляющий порт для команд FTP, которые определяют способ передачи данных. Например, команда get служит для получения файла, а команда put используется для пересылки файла некоторому хосту. FTP поддерживает передачу двоичных или текстовых (ASCII) файлов, Для чего применяются команды binary и ascii. Порт 20 служит только для Передачи данных, задаваемых командами FTP.

FTP предназначен для передачи файлов целиком, что делает его удобным средством для пересылки через глобальную сеть файлов большого размера. FTP не позволяет передать часть файла или некоторые записи внутри файла. Поскольку данные инкапсулированы в пакеты TCP, коммуникации с использованием FTP являются надежными и обеспечиваются механизмом служб с установлением соединения (что подразумевает отправку подтверждения после приема пакета). При FTP-коммуникациях выполняется передача одного потока данных, в конце которого следует признак конца файла (EOF).

TFTP – это файловый протокол стека TCP/IP, предназначенный для таких задач, как передача с некоторого сервера файлов, обеспечивающих загрузку бездисковой рабочей станции. Протокол TFTP не устанавливает соединений и ориентирован на пересылку небольших файлов в тех случаях, когда появление коммуникационных ошибок не является критичным и нет особых требований к безопасности. Отсутствие соединений при работе TFTP объясняется тем, что он функционирует поверх протокола UDP (через UDP-порт 69), а не с использованием TCP. Это означает, что в процессе передачи данных отсутствуют подтверждения пакетов или не задействованы службы с установлением соединений, гарантирующие успешную доставку пакетов в пункт назначения.

Simple Mail Transfer Protocol (SMTP)

Протокол Simple Mail Transfer Protocol (SMTP) предназначен для передачи сообщений электронной почты между сетевыми системами. С помощью этого протокола системы UNIX, OpenVMS, Windows и Novell NetWare могут пересылать электронную почту поверх протокола TCP. SMTP можно рассматривать как альтернативу протоколу FTP при передаче файла от одного компьютера к другому. При работе с SMTP не нужно знать имя учетной записи и пароль для удаленной системы. Все, что нужно, – это адрес электронной почты принимающего узла. SMTP может пересылать только текстовые файлы, поэтому файлы в других форматах должны быть конвертированы в текстовый вид, только после этого их можно поместить в SMTP-сообщение.

Domain Name System (DNS) (служба имен доменов) представляет собой службу стека TCP/IP, преобразующую имя компьютера или домена в IP-адрес или, наоборот, конвертирующую IP-адрес в компьютерное или доменное имя. Этот процесс называется разрешением (имен или адресов). Пользователям легче запоминать имена, а не IP-адреса в десятичном представлении с разделительными точками, однако поскольку компьютерам все равно нужны IP-адреса, то должен быть способ преобразования одного способа адресации в другой. Для этого служба DNS использует таблицы просмотра, в которых хранятся пары соответствующих значений.

Dynamic Host Configuration Protocol (DHCP)

Протокол Dynamic Host Configuration Protocol (DHCP) (Протокол динамически конфигурации хоста) позволяет автоматически назначать в сети IP-адреса с помощью DHCP-сервера. Когда новый компьютер, настроенный на работу с DHCP, подключается к сети, он обращается к DHCP-серверу, который выделяет (сдает в аренду) компьютеру IP-адрес, передавая его посредством протокола DHCP. Длительность аренды устанавливается на DHCP-сервере сетевым администратором.

Address Resolution Protocol (ARP)

В большинстве случаев для отправки пакета принимающему узлу отправитель должен знать как IP-адрес, так и MAC-адрес. Например, при групповых передачах используются оба адреса (IP и MAC). Эти адреса не могут совпадать и имеют разные форматы (десятичный с разделительными точками и шестнадцатеричный соответственно).

Address Resolution Protocol (ARP) (Протокол разрешения адресов) позволяет передающему узлу получить MAC-адрес выбранного принимающего узла перед отправкой пакетов. Если исходному узлу нужен некоторый MAC-адрес, то он посылает широковещательный ARP-фрейм, содержащий свой собственный MAC-адрес и IP-адрес требуемого принимающего узла. Принимающий узел отправляет обратно пакет ARP-ответа, содержащий свой MAC-адрес. Вспомогательным протоколом является Reverse Address Resolution Protocol (RARP) (Протокол обратного разрешения имен), с помощью которого сетевой узел может определить свой собственный IP-адрес. Например, RARP используется бездисковыми рабочими станциями, которые не могут узнать свои адреса иначе как выполнив RARP-запрос к своему хост-серверу. Кроме того, RARP используется некоторыми приложениями для определения IP-адреса того компьютера, на котором он выполняются.

Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) (Простой протокол сетевого управления) позволяет администраторам сети непрерывно следить за активностью сети. Протокол SNMP был разработан в 1980-х годах для того, чтобы снабдить стек TCP/IP механизмом, альтернативным стандарту OSI на управление сетями – протоколу Common Management Interface Protocol (CMIP) (Протокол общей управляющей информации). Хотя протокол SNMP был создан для стека TCP/IP, он соответствует эталонной

модели OSI. Большинство производителей предпочли использовать SNMP, а не CMIP, что объясняется большой популярностью протоколов TCP/IP, а также простотой SNMP. Протокол SNMP поддерживают многие сотни сетевых устройств, включая файловые серверы, карты сетевых адаптеров, маршрутизаторы, повторители, мосты, коммутаторы и концентраторы. В сравнении с этим, протокол CMIP применяется компанией IBM в некоторых сетях с маркерным кольцом, однако во многих других сетях он не встречается.