

Федеральное государственное бюджетное образовательное учреждение высшего образования «Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Операционные системы

Курсовая работа

по дисциплине: «Операционные системы»

Тема: «Разработка сетевой службы – просмотр РЕ-файлов»

Выполнила: студентка третьего курса группы ИП-911 Белоусова Е. Е.

Проверил: Малков Е. А.

Новосибирск 2021

Оглавление

| | |
|---------------------------------|----|
| Задача..... | 3 |
| Описание работы программы | 3 |
| Заключение..... | 9 |
| Листинг | 10 |

Задача

Разработка сетевой службы, реализующей функциональность приложения – просмотр PE-файлов.

Описание работы программы

Службы Microsoft Windows, ранее известные как службы NT, позволяют создавать долговременные исполняемые приложения, которые запускаются в собственных сеансах Windows. Для этих служб не предусмотрен пользовательский интерфейс. Они могут запускаться автоматически при загрузке компьютера, их также можно приостанавливать и перезапускать. Благодаря этому службы идеально подходят для использования на сервере, а также в ситуациях, когда необходимы долго выполняемые процессы, которые не мешают работе пользователей на том же компьютере.

В программе служба реализовывает функциональность приложения – просмотр PE-файлов.

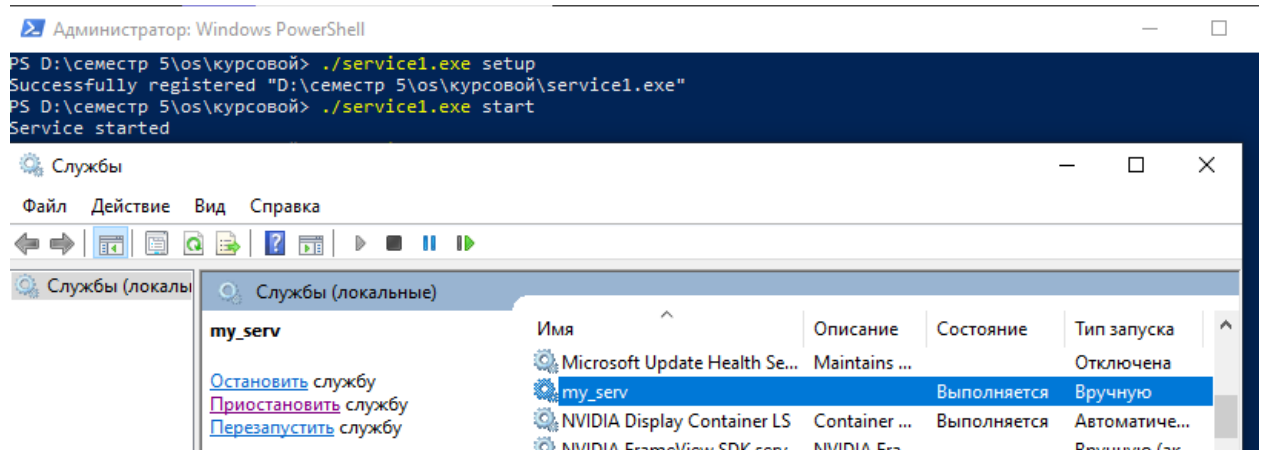


Рисунок 1 Служба

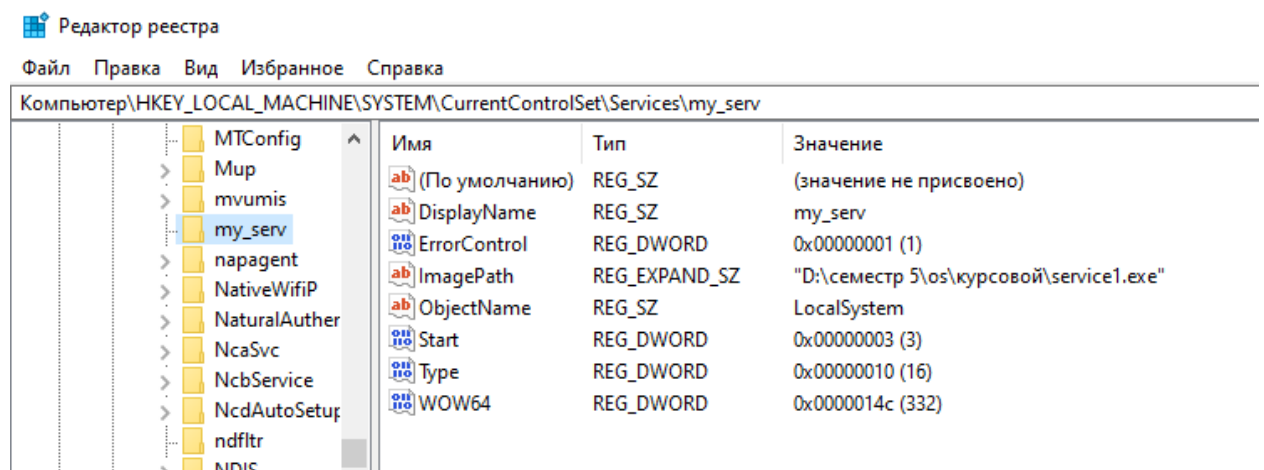


Рисунок 2

Объявим глобальные переменные для текущего состояния службы и дескриптора статуса службы.

В функции main создаем таблицу точек входа служба, указывая имя службы и главную функцию службы.

Функция StartServiceCtrlDispatcher() связывает главный поток службы с менеджером служб (service control manager или scm). Диспетчер управления службами (SCM) обслуживает базу данных установленных служб в системном реестре. База данных используется SCM и программами, которые добавляют, изменяют или конфигурируют службы. Запускается бесконечный цикл. SCM использует это соединение, чтобы посылать сервису управляющие запросы .

Функция wserv_testStart определяется процессом как точка входа для службы. Заполним структуру, определяющую состояние службы: тип службы, текущее состояние, управляющие коды.

dwServiceType

Тип службы. Этот член структуры может быть одним из следующих значений.

| Значение | Предназначение |
|-----------------------------|---|
| SERVICE_FILE_SYSTEM_DRIVER | Служба - драйвер файловой системы. |
| SERVICE_KERNEL_DRIVER | Служба - драйвер устройства. |
| SERVICE_WIN32_OWN_PROCESS | Служба запускается в своем собственном процессе. |
| SERVICE_WIN32_SHARE_PROCESS | Служба совместно использует процесс с другими службами. |

dwCurrentState

Текущее состояние службы. Этот член структуры может быть одним из следующих значений.

| Значение | Предназначение |
|--------------------------|---------------------------------------|
| SERVICE_CONTINUE_PENDING | Продолжение работы службы ожидается. |
| SERVICE_PAUSE_PENDING | Приостановка работы службы ожидается. |
| SERVICE_PAUSED | Служба приостановлена. |
| SERVICE_RUNNING | Служба в рабочем состоянии. |
| SERVICE_START_PENDING | Служба запускается. |
| SERVICE_STOP_PENDING | Служба останавливается. |
| SERVICE_STOPPED | Служба не в рабочем состоянии. |

Функция RegisterServiceCtrlHandler() регистрирует функцию, которая будет обрабатывать управляющие запросы сервиса – CtrlHandler. В случае успешного выполнения функция вернет дескриптор статуса службы.

Когда инициализация выполнена, запустим службу, установив состояние SERVICE_RUNNING. После изменения состояния, выполняется основной код программы. Открывается файл для записи данных, полученных при работе службы, в него записывается текущее время системы. Запускается функция clientApp(). Данная функция пробует установить соединение с сервером. В случае успешного соединения, служба получает файл конфигурации,

содержащий путь к РЕ-файлу и время в минутах, через которое будет обновлен файл с данными.

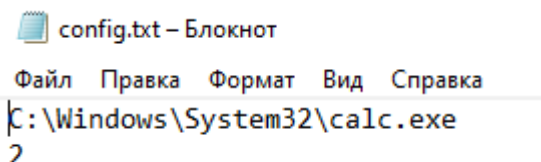


Рисунок 3 Файл конфигурации

Во время работы основной программы также поддерживается файл с логами. В него выписывается время и необходимая информация для оценки работы программы.

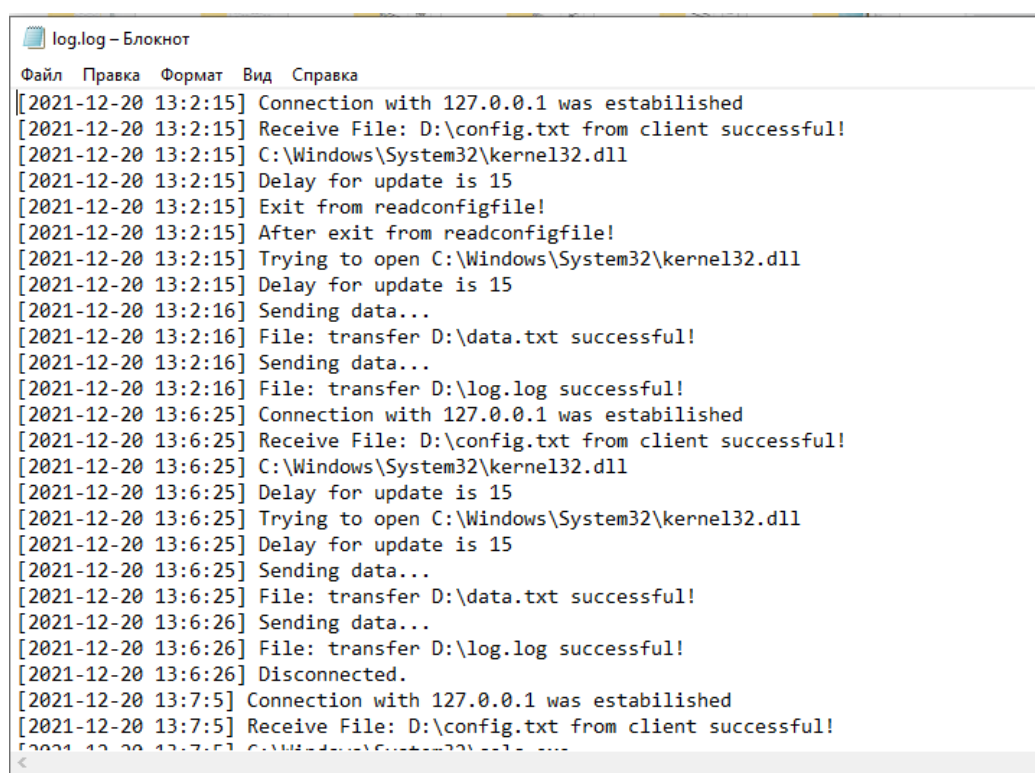
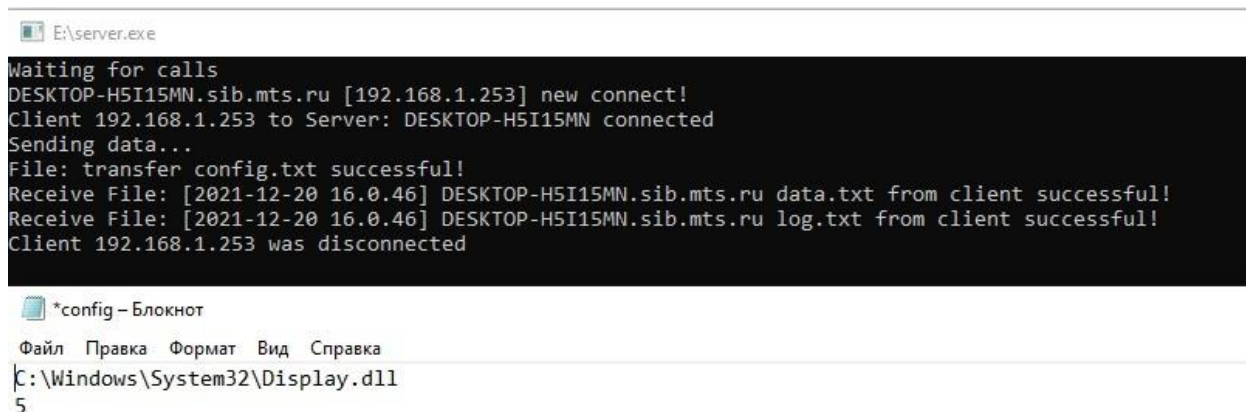


Рисунок 4 Файл с логами

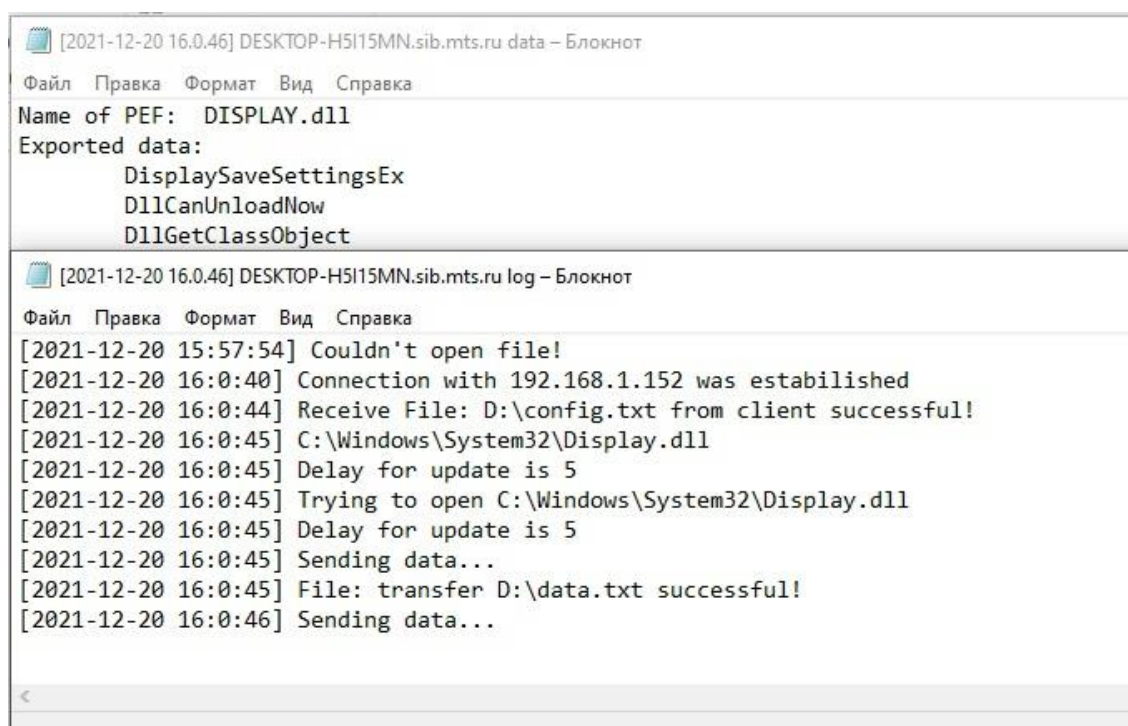
После просмотра РЕ-файлов (через отображение на память), файлы с данными и с логами отправляются на сервер, служба автоматически отключается и выжидает время для обновления информации.



```
E:\server.exe
Waiting for calls
DESKTOP-H5I15MN.sib.mts.ru [192.168.1.253] new connect!
Client 192.168.1.253 to Server: DESKTOP-H5I15MN connected
Sending data...
File: transfer config.txt successful!
Receive File: [2021-12-20 16:0.46] DESKTOP-H5I15MN.sib.mts.ru data.txt from client successful!
Receive File: [2021-12-20 16:0.46] DESKTOP-H5I15MN.sib.mts.ru log.txt from client successful!
Client 192.168.1.253 was disconnected

*config - Блокнот
Файл Правка Формат Вид Справка
C:\Windows\System32\Display.dll
5
```

Рисунок 5 Подключение к серверу



```
[2021-12-20 16:0.46] DESKTOP-H5I15MN.sib.mts.ru data - Блокнот
Файл Правка Формат Вид Справка
Name of PEF: DISPLAY.dll
Exported data:
    DisplaySaveSettingsEx
    DllCanUnloadNow
    DllGetClassObject

[2021-12-20 16:0.46] DESKTOP-H5I15MN.sib.mts.ru log - Блокнот
Файл Правка Формат Вид Справка
[2021-12-20 15:57:54] Couldn't open file!
[2021-12-20 16:0:40] Connection with 192.168.1.152 was established
[2021-12-20 16:0:44] Receive File: D:\config.txt from client successful!
[2021-12-20 16:0:45] C:\Windows\System32\Display.dll
[2021-12-20 16:0:45] Delay for update is 5
[2021-12-20 16:0:45] Trying to open C:\Windows\System32\Display.dll
[2021-12-20 16:0:45] Delay for update is 5
[2021-12-20 16:0:45] Sending data...
[2021-12-20 16:0:45] File: transfer D:\data.txt successful!
[2021-12-20 16:0:46] Sending data...
```

Рисунок 6 Файлы, полученные с клиента

В случае, если служба не смогла установить соединение с сервером, она выполняет все те же действия, но локально.

Рассмотрим функцию CtrlHandler. У каждой службы есть обработчик для обработки запросов от scm. В этой функции мы меняем состояние службы и устанавливаем его с помощью SetServiceStatus().

Рассмотрим программное управление службой. Для этого используются функции OpenSCManager, OpenService, CreateService, ControlService. Функция OpenSCManager устанавливает связь с диспетчером управления службами и открывает указанную базу данных управления службами. Требуется доступ SC_MANAGER_ALL_ACCESS, функция OpenSCManager завершается

ошибкой, если вызывающий процесс не имеет привилегий Администратора. Функция `OpenService` открывает существующую службу. Функция `CreateService` создает объект службы и добавляет его в указанную базу данных диспетчера управления службами. Функция `ControlService` отправляет управляющий код службе.

| Управляющий код | Предназначение |
|---|--|
| <code>SERVICE_CONTROL_CONTINUE</code> | Уведомляет, что временно остановленный сервис должен возобновить работу. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . |
| <code>SERVICE_CONTROL_INTERROGATE</code> | Уведомляет сервис, что он должен сообщить информацию о его текущем состоянии. Диспетчеру управления сервисами (SCM). Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_INTERROGATE</code> . |
| <code>SERVICE_CONTROL_NETBINDADD</code> | Уведомляет сетевой сервис, что есть новый компонент для соединения. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . Однако, этот управляющий код не рекомендуется применять, вместо него используйте функциональные возможности технологии <code>Plug and Play</code> . |
| <code>SERVICE_CONTROL_NETBINDDISABLE</code> | Windows NT: Это значение не поддерживается. Уведомляет сетевой сервис, что одна из его связей заблокирована. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . Однако, этот управляющий код не рекомендуется применять, вместо него используйте функциональные возможности технологии <code>Plug and Play</code> . |
| <code>SERVICE_CONTROL_NETBINDENABLE</code> | Windows NT: Это значение не поддерживается. Уведомляет сетевой сервис, что одна из его заблокированных связей включена. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . Однако, этот управляющий код не рекомендуется применять, вместо него используйте функциональные возможности технологии <code>Plug and Play</code> . |
| <code>SERVICE_CONTROL_NETBINDREMOVE</code> | Windows NT: Это значение не поддерживается. Уведомляет сетевую службу о том, что компонент для связывания был удален. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . Однако, этот управляющий код не рекомендуется применять, используйте вместо него функциональные возможности технологии <code>Plug and Play</code> . |
| <code>SERVICE_CONTROL_PARAMCHANGE</code> | Windows NT: Это значение не поддерживается. Уведомляет службу, что ее параметры запуска изменились. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . |
| <code>SERVICE_CONTROL_PAUSE</code> | Windows NT: Это значение не поддерживается. Уведомляет службу, что она должна сделать паузу. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_PAUSE_CONTINUE</code> . |
| <code>SERVICE_CONTROL_STOP</code> | Уведомляет службу, что она должна остановиться. Дескриптор <code>hService</code> должен иметь право доступа <code>SERVICE_STOP</code> . |

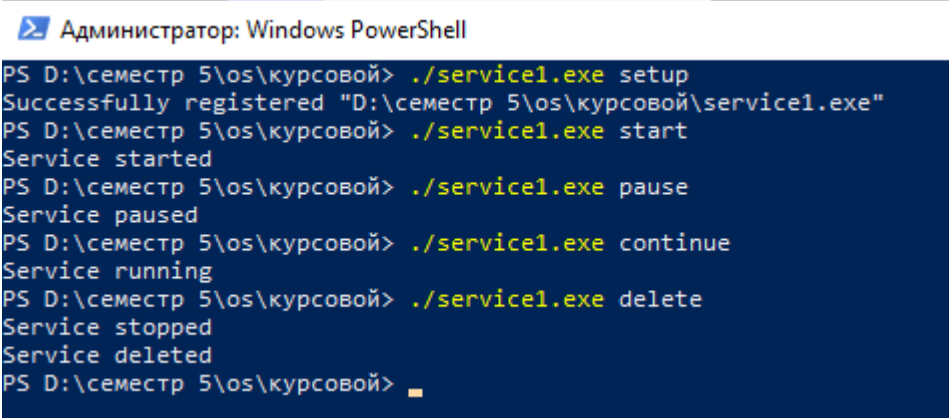


Рисунок 7 Программное управление службой

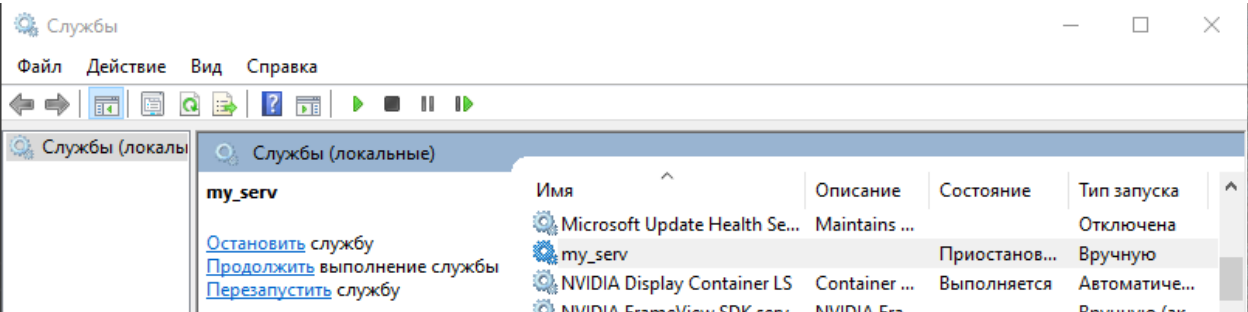


Рисунок 8 Приостановка службы

Рассмотрим сетевую часть службы. Для ее работы необходимо:

1. Инициализация сокетных интерфейсов Win32API. Запуск программного интерфейса сокетов в Win32API (WSAStartup()), где первый параметр – запрашиваемая версия сокетов, второй параметр - указатель на структуру WSADATA*, хранящую текущую версию реализации сокетов).

2. Инициализация сокета, т.е. создание специальной структуры данных и её инициализация вызовом функции.

SOCKET socket(int <семейство используемых адресов, IPv4 AF_INET>, int <тип сокета>, int <тип протокола>)

Функция socket() возвращает дескриптор с номером сокета, под которым он зарегистрирован в ОС. Если же инициализировать сокет по каким-то причинам не удалось – возвращается значение INVALID_SOCKET.

3. «Привязка» созданного сокета к конкретной паре IP-адрес/Порт – с этого момента данный сокет (его имя) будет ассоциироваться с конкретным процессом, который «висит» по указанному адресу и порту.

int bind(SOCKET <имя сокета, к которому необходимо привязать адрес и порт>, sockaddr* <указатель на структуру, содержащую детальную информацию по адресу и порту, к которому надо привязать сокет>, int <размер структуры, содержащей адрес и порт>)

Заполнение полей структуры sockaddr_in: servInfo.sin_family = AF_INET (семейство адресов); servInfo.sin_port = htons(<указать номер порта как unsigned short>); порт всегда указывается через вызов функции htons(), которая переупаковывает привычное цифровое значение порта типа unsigned short в побайтовый порядок понятный для протокола TCP/IP (протоколом установлен порядок указания портов от старшего к младшему байту или «big-endian»).

4. Для серверной части приложения: запуск процедуры прослушивания подключений на привязанный сокет.

int listen(SOCKET <«слушающий» сокет, который мы создавали на предыдущих этапах>, int <максимальное количество процессов, разрешенных к подключению>)

Для клиентской части приложения: запуск процедуры подключения к серверному сокету (должны знать его IP-адрес/Порт). Привязка сокета к конкретному процессу (bind()) не требуется, т.к. сокет будет привязан к серверному Адресу и Порту через вызов функции connect()(по сути аналог bind() для Клиента). Собственно, после создания и инициализации сокета на

клиентской стороне, нужно вызвать указанную функцию `connect()`. Её прототип:

```
int connect(SOCKET <инициализированный сокет>, sockaddr* <указатель на
структуру, содержащую IP-адрес и Порт сервера>, int <размер структуры
sockaddr>)
```

5. Подтверждение подключения (обычно на стороне сервера).

```
SOCKET accept(SOCKET <"слушающий" сокет на стороне Сервера>, sockaddr*
<указатель на пустую структуру sockaddr, в которую будет записана
информация по подключившемуся Клиенту>, int* <указатель на размер
структуры типа sockaddr>)
```

Если подключение подтверждено, то вся информация по текущему соединению передаётся на новый сокет, который будет отвечать со стороны Сервера за конкретное соединение с конкретным Клиентом. Перед вызовом `accept()` нам надо создать пустую структуру типа `sockaddr_in`, куда запишутся данные подключившегося Клиента после вызова `accept()`.

6. Обмен данными между процессами через установленное сокетное соединение.

Принимать информацию на любой стороне можно с помощью функции `recv()`, которая при своём вызове блокирует исполнение кода программы до того момента, пока она не получит информацию от другой стороны, либо пока не произойдет ошибка в передаче или соединении.

Отправлять информацию с любой стороны можно с помощью функции `send()`. При вызове данной функции обычно никакого ожидания и блокировки не происходит, а переданные в неё данные сразу же отправляются другой стороне.

```
int recv(SOCKET <сокет акцептованного соединения>, char[] <буфер для
приёма информации с другой стороны>, int <размер буфера>, <флаги>)
```

```
int send(SOCKET <сокет акцептованного соединения>, char[] <буфер хранящий
отсылаемую информацию>, int <размер буфера>, <флаги>)
```

7. Закрытие сокетного соединения.

Заключение

Благодаря курсу “Операционные системы” удалось создать сетевую службу, реализующую функциональность программы просмотр PE-файлов.

Листинг

```
//service.cpp

#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <string.h>
#include <time.h>
#include <process.h>
#include <stdio.h>
#include <imagehlp.h>
#include <locale.h>
#include <tchar.h>

#pragma comment(lib, "advapi32.lib")
#pragma comment(lib, "imagehlp.lib")
#define LOGPATH "D:\\log.log"
#define DATAPATH "D:\\data.txt"
#define CONFIGPATH "D:\\config.txt"
#pragma comment(lib, "Ws2_32.lib")
#define PORT 1952
#define SERVERADDR "192.168.1.152"
// #define SERVERADDR "192.168.1.253"
// #define SERVERADDR "127.0.0.1"

//cmd.exe /k "C:\Program Files (x86)\Microsoft Visual
Studio\2019\Enterprise\VC\Auxiliary\Build\vcvars32.bat" & powershell

SERVICE_STATUS wserv_testStatus;
SERVICE_STATUS_HANDLE wserv_testStatusHandle;
char pathToFile[CHAR_MAX];
int delayForUpdate = 1;
char serviceName[] = "my_serv";
using namespace std;

void writeToLogfile(char* error)
{
    FILE* flog = fopen(LOGPATH, "a");
```

```

SYSTEMTIME stSystemTime;

GetSystemTime(&stSystemTime);

fprintf(flog, "[%d-%d-%d  %d:%d:%d]  %s\n", stSystemTime.wYear,  stSystemTime.wMonth,
stSystemTime.wDay, stSystemTime.wHour, stSystemTime.wMinute,

        stSystemTime.wSecond, error);

fclose(flog);
}

```

```

void writeToLogfile(int error)
{
    FILE* flog = fopen(LOGPATH, "a");

    SYSTEMTIME stSystemTime;

    GetSystemTime(&stSystemTime);

    fprintf(flog, "[%d-%d-%d  %d:%d:%d]  Delay for update is %d\n", stSystemTime.wYear,
stSystemTime.wMonth, stSystemTime.wDay, stSystemTime.wHour, stSystemTime.wMinute,

        stSystemTime.wSecond, error);

    fclose(flog);
}

```

```

void Export(IMAGE_NT_HEADERS *pNtHdr, LPVOID pSrcFile){
    char buff[CHAR_MAX] = {'\0'};

    IMAGE_EXPORT_DIRECTORY* ExpTable;

    char *pName, *sName, **pNames;

    DWORD nNames;

    DWORD RVAExpDir, VAExpAddress;

    FILE *fp = fopen(DATAPATH, "a");

    int i;

    RVAExpDir = >OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;
    VAExpAddress = (DWORD)ImageRvaToVa(pNtHdr, pSrcFile, RVAExpDir, NULL);
    ExpTable=(IMAGE_EXPORT_DIRECTORY*)VAExpAddress;
    if(ExpTable == 0)

```

```

{
    writeToLogfile("There is no exported data.");
    return;
}

sName=(char*)ImageRvaToVa(pNtHdr,pSrcFile,ExpTable->Name,NULL);
fprintf(fp, "Name of PEF: %s\n",sName);
pNames=(char**)ImageRvaToVa(pNtHdr,pSrcFile,ExpTable->AddressOfNames,NULL);
nNames=ExpTable->NumberOfNames;
fprintf(fp, "Exported data: \n");
for(i = 0; i < nNames; i++){
    if (wserv_testStatus.dwCurrentState == SERVICE_STOPPED)
    {
        fclose(fp);
        return;
    }
    while (wserv_testStatus.dwCurrentState == SERVICE_PAUSED)
    {
        if (wserv_testStatus.dwCurrentState == SERVICE_STOPPED)
        {
            fclose(fp);
            return;
        }
    }
    pName = (char*)ImageRvaToVa(pNtHdr,pSrcFile,(DWORD)*pNames,NULL);
    fprintf(fp, "    %s\n",pName);
    *pNames++;
}
fprintf(fp, "\n");
fclose(fp);
}

```

```

void Import(IMAGE_NT_HEADERS *pNtHdr, LPVOID pSrcFile){
    char *pName = nullptr, *sName, **pNames;
    DWORD nNames;

```

```

DWORD RVAExpDir, VAExpAddress;

IMAGE_IMPORT_DESCRIPTOR* ImportTable;

RVAExpDir = pNtHdr->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;
VAExpAddress = (DWORD)ImageRvaToVa(pNtHdr,pSrcFile,RVAExpDir,NULL);
ImportTable=(IMAGE_IMPORT_DESCRIPTOR*)VAExpAddress;
if(ImportTable == NULL)
{
    writeToLogfile("There is no imported data.");
    return;
}
FILE *fp = fopen(DATAPATH, "a");
fprintf(fp, "Imported data: \n");
while(ImportTable->Name != NULL){
    pNames=(char**)ImageRvaToVa(pNtHdr,pSrcFile,ImportTable->FirstThunk,NULL);
    sName=(char*)ImageRvaToVa(pNtHdr,pSrcFile,ImportTable->Name,NULL);
    char buff[CHAR_MAX] = {'\0'};
    fprintf(fp, "Name of PEF: %s\n",sName);
    while(pName != 0)
    {
        if (wserv_testStatus.dwCurrentState == SERVICE_STOPPED)
        {
            fclose(fp);
            return;
        }
        while (wserv_testStatus.dwCurrentState == SERVICE_PAUSED)
        {
            if (wserv_testStatus.dwCurrentState == SERVICE_STOPPED)
            {
                fclose(fp);
                return;
            }
        }
    }
}

```

```

        pName = (char*)ImageRvaToVa(pNtHdr,pSrcFile,(DWORD)*pNames+2,NULL);
        if(pName != 0){
            fprintf(fp, "    %s\n",pName);
        }
        *pNames++;
    }
    *pName++;
    ImportTable++;
}
fprintf(fp, "\n\n");
fclose(fp);
}

```

```

void readConfigFile()
{
    FILE* fp = fopen(CONFIGPATH, "r");
    char delayForUpdate_[CHAR_MAX];
    char* r;
    char buff[CHAR_MAX];
    if (fp == NULL) {
        writeToLogfile("Error: can't open configuration file.");
        return;
    }
    r = fgets(pathToFile, sizeof(pathToFile), fp);
    if (r == NULL)
    {
        if (feof(fp) != 0)
        {
            writeToLogfile(&pathToFile[0]);
            writeToLogfile("Error: wrong configuration file!");
            return;
        }
        else
        {

```

```

        writeToLogfile(&pathToFile[0]);
        writeToLogfile("Error: can't read configuration file!");
        return;
    }
}
pathToFile[strlen(pathToFile) - 1] = '\0';
writeToLogfile(&pathToFile[0]);
r = fgets(delayForUpdate_, sizeof(delayForUpdate_), fp);
if (r == NULL)
{
    if (feof(fp) != 0)
    {
        writeToLogfile("Error: wrong configuration file! Set default value.");
    }
    else
    {
        writeToLogfile("Error: can't read configuration file!");
        return;
    }
}
const char* tmp = delayForUpdate_;
delayForUpdate = atoi(tmp);
writeToLogfile(delayForUpdate);
fclose(fp);
}

```

```

int checkPEFile()
{
    HANDLE hFileMap, hFile;
    LPVOID pSrcFile;
    IMAGE_DOS_HEADER *pDosHeader;
    IMAGE_NT_HEADERS *pNtHdr;
    IMAGE_SECTION_HEADER *pFirstSectionHeader, *pSectionHeader;
    readConfigFile();
}

```



```

char* log = new char[255];
sprintf(log, "Trying to open %s", pathToFile);
writeToLogfile(log);
writeToLogfile(delayForUpdate);

hFile = CreateFileA(pathToFile, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0,
NULL);
if (hFile == INVALID_HANDLE_VALUE)
{
    writeToLogfile("Couldn't open file!");
    return -1;
}

hFileMap = CreateFileMapping (hFile, NULL, PAGE_READONLY, 0, 0, NULL);
if(hFileMap == NULL)
{
    writeToLogfile("Could not create mapping file.");
    return -1;
}

pSrcFile = (PBYTE) MapViewOfFile(hFileMap,FILE_MAP_READ,0,0,0);
if(pSrcFile == NULL)
{
    writeToLogfile("Could not map file.");
    return -1;
}

pDosHeader = (IMAGE_DOS_HEADER *)pSrcFile;
pNtHdr = (IMAGE_NT_HEADERS *)((DWORD)pDosHeader + pDosHeader->e_lfanew);
Export(pNtHdr,pSrcFile);
Import(pNtHdr,pSrcFile);
return 0;
}

```

```

int sendFile(char* filename, SOCKET my_sock)

```

```

{
    char buff[1024];
    //printf("Sending data...\n");
    writeToLogfile("Sending data...");
    FILE* fp = fopen(filename, "r");
    if (fp == NULL) {
        writeToLogfile("Error: can't open configuration file");
        //printf("Error: can't open configuration file.\n");
        return -1;
    }
    memset(buff, 0, sizeof(buff));
    char* r;
    while ((r = fgets(buff, sizeof(buff), fp)) != 0)
    {
        if (send(my_sock, buff, strlen(buff), 0) < 0)
        {
            writeToLogfile("Send File: failed");
            //printf("Send File: failed\n");
            break;
        }
        memset(buff, 0, sizeof(buff));
    }
    fclose(fp);
    char* log = new char[255];
    sprintf(log, "File: transfer %s successful!", filename);
    writeToLogfile(log);
    //printf("File: transfer %s successful!\n", filename);
    return 0;
}

int receiveFile(SOCKET sock)
{
    FILE* fp = fopen(CONFIGPATH, "w");
    int length = 0;

```

```

char buff[1024];
int bytes_rcv;
memset(buff, 0, sizeof(buff));
//printf("file opened\n");
while ((length = recv(sock, buff, sizeof(buff), 0)) > 0)
{
    //printf("%s\n", buff);
    int breakAfter = 0;
    char* p;
    if((p = strstr(buff, "<end of file>\n")) != NULL)
    {
        breakAfter = p - buff;
        memcpy(buff, buff, breakAfter);
        length = breakAfter;
        breakAfter=1;
    }
    else if ((p = strstr(buff, "Error\n")) != NULL)
    {
        fclose(fp);
        return -1;
    }
    if (fwrite(buff, sizeof(char), length, fp) < length)
    {
        writeToLogfile("File: write failed");
        //printf("File: Write Failed\n");
        break;
    }
    if (breakAfter)
    {
        break;
    }
    // fprintf(fp, "%s", buff);
    memset(buff, 0, sizeof(buff));
}

```

```

//printf("Recv error %d\n", WSAGetLastError());
fclose(fp);

    char* log = new char[255];
    sprintf(log, "Receive File: %s from client successful!", CONFIGPATH);
    writeToLogfile(log);
//printf("Receive File: %s from client successful!\n", CONFIGPATH);
    return 0;
}

int clientApp()
{
    char buff[1024];
    struct sockaddr_in dest_addr;
    SOCKET my_sock;
    struct hostent *hst;
    int nsize;
    char* log;
    if(WSAStartup(0x202, (WSADATA *)&buff[0]))
    {
        //printf("WSAstart error %d\n", WSAGetLastError());
        log = new char[255];
        sprintf(log, "WSAstart error %d", WSAGetLastError());
        writeToLogfile(log);
        return -1;
    }
    my_sock = socket(AF_INET, SOCK_STREAM, 0);
    if(my_sock < 0)
    {
        //printf("Socket() error %d\n", WSAGetLastError());
        log = new char[255];
        sprintf(log, "Socket() error %d", WSAGetLastError());
        writeToLogfile(log);
        return -1;
    }
}

```

```

dest_addr.sin_family=AF_INET;
dest_addr.sin_port=htons(PORT);
if(inet_addr(SERVERADDR)!=INADDR_NONE)
    dest_addr.sin_addr.s_addr=inet_addr(SERVERADDR);
else if(gethostbyname(SERVERADDR))
    ((unsigned long*)&dest_addr.sin_addr)[0]=((unsigned long**)hst->h_addr_list)[0][0];
else {
    //printf("Invalid address %s\n", SERVERADDR);
    log = new char[255];
    sprintf(log, "Invalid address %s", SERVERADDR);
    writeToLogfile(log);

    closesocket(my_sock);
    WSACleanup();
    return -1;
}

if(connect(my_sock, (struct sockaddr*)&dest_addr, sizeof(dest_addr)))
{
    //printf("Connect error %d\n", WSAGetLastError());
    log = new char[255];
    sprintf(log, "Connect error %d", WSAGetLastError());
    writeToLogfile(log);

    return -1;
}

//printf("Connection with %s was established", SERVERADDR);
log = new char[255];
sprintf(log, "Connection with %s was established", SERVERADDR);
writeToLogfile(log);

char localName[100], toSend[100];
if(gethostname(localName, sizeof(localName)) == SOCKET_ERROR)
{
    //printf("Error: wrong local name %d\n", WSAGetLastError());
    log = new char[255];
    sprintf(log, "Error: wrong local name %d", WSAGetLastError());
    writeToLogfile(log);
}

```

```

}

sprintf(toSend, "%s connected\n", localName);
while((nsize=recv(my_sock, &buff[0], sizeof(buff)-1, 0)) != SOCKET_ERROR)
{
    buff[nsize] = 0;
    //printf("Server To Client: %s", buff);
    if(!strcmp(buff, "Connected!\n"))
    {
        send(my_sock, toSend, sizeof(toSend), 0);

        int res = receiveFile(my_sock);
        if (res == -1)
        {
            writeToLogfile("Error from server!");

            break;
        }

        res = 0;
        res = checkPEFile();
        if (res == -1)
        {
            writeToLogfile("Error when reading PE file!");

            break;
        }

        //printf("_____ \n");
    } else if(!strcmp(buff, "Waiting for data...\n"))
    {
        sendFile(DATAPATH, my_sock);
        send(my_sock, "<end of file>\n", sizeof("<end of file>\n"), 0);
        Sleep(500);
        sendFile(LOGPATH, my_sock);
        send(my_sock, "<end of file>\n", sizeof("<end of file>\n"), 0);
        //printf("Exit...");
        writeToLogfile("Disconnected.");

        closesocket(my_sock);
        WSACleanup();
    }
}

```

```

        return 0;
    }
}

    log = new char[255];
    sprintf(log, "Recv error %d", WSAGetLastError());
    writeToLogfile(log);
    //printf("Recv error %d\n", WSAGetLastError());
    closesocket(my_sock);
    WSACleanup();
    return -1;
}

```

```

VOID __stdcall CtrlHandler (DWORD Opcode)
{
    DWORD status;
    switch(Opcode)
    {
        case SERVICE_CONTROL_PAUSE:
            wserv_testStatus.dwCurrentState = SERVICE_PAUSED;
            break;

        case SERVICE_CONTROL_CONTINUE:
            wserv_testStatus.dwCurrentState = SERVICE_RUNNING;
            break;

            case SERVICE_CONTROL_STOP:
                wserv_testStatus.dwWin32ExitCode = 0;
                wserv_testStatus.dwCurrentState = SERVICE_STOPPED;
                wserv_testStatus.dwCheckPoint = 0;
                wserv_testStatus.dwWaitHint = 0;
                if (!SetServiceStatus (wserv_testStatusHandle,
                    &wserv_testStatus))
                {
                    status = GetLastError();
                }
            }
    }
}

```



```

        return;

        default:

            break;

    }

    if (!SetServiceStatus (wserv_testStatusHandle, &wserv_testStatus))
    {
        status = GetLastError();
    }

    return;
}

void __stdcall wserv_testStart (DWORD argc, LPTSTR *argv)
{
    DWORD status;

    wserv_testStatus.dwServiceType    = SERVICE_WIN32;
    wserv_testStatus.dwCurrentState   = SERVICE_START_PENDING;
    wserv_testStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP |
        SERVICE_ACCEPT_PAUSE_CONTINUE;
    wserv_testStatus.dwWin32ExitCode  = 0;
    wserv_testStatus.dwServiceSpecificExitCode = 0;
    wserv_testStatus.dwCheckPoint     = 0;
    wserv_testStatus.dwWaitHint       = 0;
    wserv_testStatusHandle = RegisterServiceCtrlHandler(
        TEXT("my_serv"),    CtrlHandler);

    if (wserv_testStatusHandle == (SERVICE_STATUS_HANDLE)0)
        return;

    wserv_testStatus.dwCurrentState    = SERVICE_RUNNING;
    wserv_testStatus.dwCheckPoint      = 0;
    wserv_testStatus.dwWaitHint        = 0;

```

```

if (!SetServiceStatus (wserv_testStatusHandle, &wserv_testStatus))
{
    status = GetLastError();
}

FILE* fp;
SYSTEMTIME stSystemTime;

while (wserv_testStatus.dwCurrentState!=SERVICE_STOPPED)
{
    if (wserv_testStatus.dwCurrentState!=SERVICE_PAUSED){
        GetSystemTime(&stSystemTime);
        fp = fopen(DATAPATH, "a");
        fprintf(fp, "[%d-%d-%d                %d:%d:%d]\n", stSystemTime.wYear,
stSystemTime.wMonth, stSystemTime.wDay, stSystemTime.wHour, stSystemTime.wMinute,

                stSystemTime.wSecond);
        fclose(fp);
        int res = clientApp();
        if(res != 0)
        {
            checkPEFile();
        }
    }
    Sleep(60000 * delayForUpdate);
}
return;
}

```

```

void main(int argc, char *argv[])
{
    setlocale(LC_ALL, "Russian");
    SERVICE_TABLE_ENTRY DispatchTable[] =
    {

```

```

{ TEXT("my_serv"), wserv_testStart    }
    ,{ NULL,        NULL        }
};

if (argc>1 && (!strcmp(argv[1],"stop") || !strcmp(argv[1],"delete")))
{
    SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_ALL_ACCESS);
    if (!scm)
    {
        cout<<"Can't open SCM\n";
        exit(1);
    }
    SC_HANDLE svc=OpenService(scm,"my_serv",SERVICE_STOP | SERVICE_QUERY_STATUS
);
    if (!svc)
    {
        cout<<"Can't open service\n";
        exit(2);
    }
    DWORD dwBytesNeeded{0};
    SERVICE_STATUS_PROCESS ssp {0};
    QueryServiceStatusEx(    svc,    SC_STATUS_PROCESS_INFO,    (LPBYTE)(    &ssp),
sizeof(SERVICE_STATUS_PROCESS), &dwBytesNeeded );
    if (ssp.dwCurrentState != SERVICE_STOPPED)
    {
        if (ControlService(svc, SERVICE_CONTROL_STOP, (LPSERVICE_STATUS)&ssp))
        {
            if    (!(ssp.dwCurrentState    ==    SERVICE_STOP_PENDING    ||
ssp.dwCurrentState == SERVICE_STOPPED))
            {
                cout << "Can't stop service\n";
                exit(3);
            }
        }
        cout<<"Service stopped\n";
        CloseServiceHandle(svc);

```

```

        CloseServiceHandle(scm);

        if (strcmp(argv[1],"delete"))
        {
            exit(0);
        }
    }

}

if (argc>1 && !strcmp(argv[1],"pause"))
{
    SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_ALL_ACCESS);
    if (!scm)
    {
        cout<<"Can't open SCM\n";
        exit(1);
    }

    SC_HANDLE svc=OpenService(scm,"my_serv", SERVICE_ALL_ACCESS);
    if (!svc)
    {
        cout<<"Can't open service\n";
        exit(2);
    }

    SERVICE_STATUS_PROCESS ssp {0};
    if (ControlService(svc, SERVICE_CONTROL_PAUSE, (LPSERVICE_STATUS)&ssp)
    {
        if (!(ssp.dwCurrentState == SERVICE_PAUSE_PENDING || ssp.dwCurrentState ==
SERVICE_PAUSED))
        {
            cout << "Can't pause service\n";
            exit(3);
        }
    }
}

```

```

        cout<<"Service paused\n";
        CloseServiceHandle(svc);
        CloseServiceHandle(scm);
        exit(0);
    }
    if (argc>1 && !strcmp(argv[1],"continue"))
    {
        SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_ALL_ACCESS);
        if (!scm)
        {
            cout<<"Can't open SCM\n";
            exit(1);
        }
        SC_HANDLE svc=OpenService(scm,"my_serv", SERVICE_ALL_ACCESS);
        if (!svc)
        {
            cout<<"Can't open service\n";
            exit(2);
        }

        SERVICE_STATUS_PROCESS ssp {0};
        if (ControlService(svc, SERVICE_CONTROL_CONTINUE, (LPSERVICE_STATUS)&ssp))
        {
            if (!(ssp.dwCurrentState == SERVICE_CONTINUE_PENDING || ssp.dwCurrentState
== SERVICE_RUNNING))
            {
                cout << "Can't continue service\n";
                exit(3);
            }
        }
        cout<<"Service running\n";
        CloseServiceHandle(svc);
        CloseServiceHandle(scm);
        exit(0);
    }

```

```

}

if (argc>1 && !strcmp(argv[1],"delete"))
{
    SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_CREATE_SERVICE);
    if (!scm)
    {
        cout<<"Can't open SCM\n";
        exit(1);
    }
    SC_HANDLE svc=OpenService(scm,"my_serv",DELETE);
    if (!svc)
    {
        cout<<"Can't open service\n";
        exit(2);
    }
    if (!DeleteService(svc))
    {
        cout<<"Can't delete service\n";
        exit(3);
    }
    cout<<"Service deleted\n";
    CloseServiceHandle(svc);
    CloseServiceHandle(scm);
    exit(0);
}

if (argc>1 && !strcmp(argv[1],"start"))
{
    SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_ALL_ACCESS);
    if (!scm)
    {
        cout<<"Can't open SCM\n";
        exit(1);
    }
    SC_HANDLE svc=OpenService(scm,"my_serv",SERVICE_ALL_ACCESS);

```

```

        if (!svc)
        {
            cout<<"Can't open service\n";
            exit(2);
        }
        if (!StartService(svc, 0, NULL))
        {
            cout<<"Can't start service\n";
            exit(3);
        }
        cout<<"Service started\n";
        CloseServiceHandle(svc);
        CloseServiceHandle(scm);

        exit(0);
    }

    if (argc>1 && !strcmp(argv[1],"setup"))
    {
        char pname[1024];
        pname[0]="";
        GetModuleFileName(NULL, pname+1, 1023);
        strcat(pname,"\\");
        SC_HANDLE scm=OpenSCManager(NULL,NULL,SC_MANAGER_CREATE_SERVICE),svc;
        if (!scm)
        {
            cout<<"Can't open SCM\n";
            exit(1);
        }
        if (!(svc=CreateService(scm,"my_serv","my_serv",SERVICE_ALL_ACCESS,
            SERVICE_WIN32_OWN_PROCESS,SERVICE_DEMAND_START,
            SERVICE_ERROR_NORMAL,pname,NULL,NULL,NULL,NULL)))
        {
            cout<<"Registration error!\n";

```



```

        exit(2);
    }
    cout<<"Successfully registered "<<pname<<"\n";
    CloseServiceHandle(svc);
    CloseServiceHandle(scm);
    exit(0);
}

if (!StartServiceCtrlDispatcher(DispatchTable))
{

}

}
//server.c
#include <stdio.h>
#include <winsock2.h>
#include <string.h>
#include <windows.h>
#pragma comment(lib, "Ws2_32.lib")

#define MY_PORT 1952
#define CONFIG_FILE "config.txt"

struct CLIENT_INFO
{
    SOCKET hClientSocket ;
    struct sockaddr_in clientAddr ;
};

DWORD WINAPI ClientThread( LPVOID lpData );
SOCKET mysocket;

int main(int argc, char* argv[])
{

```

```

char buff[1024];
struct sockaddr_in local_addr;
if(WSAStartup(0x0202, (WSADATA *) &buff[0]))
{
    printf("Error WSAStartup %d\n", WSAGetLastError());
    return -1;
}

if((mysocket=socket(AF_INET, SOCK_STREAM, 0))<0)
{
    printf("Error socket %d\n", WSAGetLastError());
    WSACleanup();
    return -1;
}
local_addr.sin_family=AF_INET;
local_addr.sin_port=htons(MY_PORT);
local_addr.sin_addr.s_addr=0;
if(bind(mysocket, (struct sockaddr *) &local_addr, sizeof(local_addr)))
{
    printf("Error bind %d\n", WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}
if(listen(mysocket, 0x100))
{
    printf("Error listen %d\n", WSAGetLastError());
    closesocket(mysocket);
    WSACleanup();
    return -1;
}
printf("Waiting for calls\n");
while (1)
{

```

```

SOCKET hClientSocket ;
struct sockaddr_in clientAddr ;
int nSize = sizeof(clientAddr) ;
hClientSocket = accept(mysocket, ( struct sockaddr *) &clientAddr, &nSize);
if (hClientSocket != INVALID_SOCKET)
{
    HANDLE hClientThread ;
    struct CLIENT_INFO clientInfo ;
    DWORD dwThreadId ;
    clientInfo.clientAddr = clientAddr ;
    clientInfo.hClientSocket = hClientSocket ;
    struct hostent *hst;
    int bytes_recv;
    hst=gethostbyaddr((char*)&clientAddr.sin_addr.s_addr, 4, AF_INET);
    printf("%s [%s] new connect!\n", (hst)?hst->h_name:"", inet_ntoa(clientAddr.sin_addr));
    hClientThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) ClientThread, (LPVOID)
    &clientInfo, 0, &dwThreadId);
        if ( hClientThread == NULL )
        {
            printf("can't create thread.\n");
        }
        else
        {
            CloseHandle( hClientThread ) ;
        }
    }
    else printf("Error accept %d\n", WSAGetLastError());;
}
printf("By myself!\n");
closesocket(mysocket);
WSACleanup();
return 0;
}

```

```

void receiveFile(struct CLIENT_INFO *pCl, char* suffix)
{
    SYSTEMTIME stSystemTime;

    GetSystemTime(&stSystemTime);

    char date[255];
    char filename[CHAR_MAX];
    struct hostent *hst;

    hst=gethostbyaddr((char*)&pCl->clientAddr.sin_addr.s_addr, 4, AF_INET);

    sprintf(date, "[%d-%d-%d %d.%d.%d]\0", stSystemTime.wYear, stSystemTime.wMonth,
stSystemTime.wDay, stSystemTime.wHour, stSystemTime.wMinute,

        stSystemTime.wSecond);

    sprintf(filename, "%s %s %s.txt", date, (hst)?hst->h_name:"", suffix);
    FILE* fp = fopen(filename, "a");
    int length = 0;
    char buff[1024];
    int bytes_rcv;
    memset(buff, 0, sizeof(buff));
    while ((length = recv(pCl->hClientSocket, buff, sizeof(buff), 0)) > 0)
    {
        //printf("%s\n", buff);

        int breakAfter = 0;
        char* p;

        if((p = strstr(buff, "<end of file>\n")) != NULL)
        {
            breakAfter = p - buff;

            memcpy(buff,buff,breakAfter);

            length = breakAfter;

            breakAfter=1;
        }
        if (fwrite(buff, sizeof(char), length, fp) < length)
        {
            printf("File: Write Failed\n");

            break;

```

```

    }

    if (breakAfter)
    {
        break;
    }

    // fprintf(fp, "%s", buff);
    memset(buff, 0, sizeof(buff));
}
fclose(fp);
printf("Receive File: %s from client successful!\n", filename);
}

```

```

int sendFile(char* filename, SOCKET my_sock)
{
    char buff[1024];
    printf("Sending data...\n");
    FILE* fp = fopen(filename, "r");
    if (fp == NULL) {
        printf("Error: can't open configuration file.\n");
        send(my_sock, "Error\n", sizeof("Error\n"), 0);
        return -1;
    }
    memset(buff, 0, sizeof(buff));
    char* r;
    while ((r = fgets(buff, sizeof(buff), fp)) != 0)
    {
        if (send(my_sock, buff, strlen(buff), 0) < 0)
        {
            printf("Send File: failed\n");
            break;
        }
        memset(buff, 0, sizeof(buff));
    }
    fclose(fp);
}

```

```

    printf("File: transfer %s successful!\n", filename);
}

DWORD WINAPI ClientThread(LPVOID lpData)
{
    struct CLIENT_INFO *pCI = (struct CLIENT_INFO*) lpData;
    char buff[1024];
    int bytes_rcv;
    send(pCI->hClientSocket, "Connected!\n", sizeof("Connected!\n"), 0);

    while((bytes_rcv=recv(pCI->hClientSocket, &buff[0], sizeof(buff), 0)) && bytes_rcv!=SOCKET_ERROR)
    {
        buff[bytes_rcv] = 0;
        printf("Client %s to Server: %s", inet_ntoa(pCI->clientAddr.sin_addr), buff);
        //printf("Enter name of config file: ");
        //fgets(&buff[0], sizeof(buff)-1, stdin);
        //buff[strlen(buff) - 1] = '\0';
        int res = sendFile(CONFIG_FILE, pCI->hClientSocket);
        if (res == -1)
        {
            break;
        }
        memset(buff, 0, sizeof(buff));
        send(pCI->hClientSocket, "\n<end of file>\n", sizeof("\n<end of file>\n"), 0);
        Sleep(500);
        sprintf(buff, "Waiting for data...\n");
        send(pCI->hClientSocket, &buff[0], bytes_rcv, 0);
        receiveFile(pCI, "data");
        receiveFile(pCI, "log");
    }
    printf("Client %s was disconnected\n", inet_ntoa(pCI->clientAddr.sin_addr));
    closesocket(pCI->hClientSocket);
    return 0;
}

```