### СОДЕРЖАНИЕ

|  |  |
| --- | --- |
| ВВЕДЕНИЕ | 4 |
| 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ | 5 |
| * 1. Описание предметной области | 5 |
| * 1. Постановка задачи проектирования | 8 |
| 1. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ | 10 |
| * 1. Алгоритм работы программы   2. Макет пользовательского интерфейса   3. Выбор и обоснование средств разработки      1. Обоснование выбора языка программирования      2. Обоснование выбора среды разработки      3. Обоснование выбора средств разработки пользовательского интерфейса      4. Выбор средств разработки программы | 10  12  12  12  13 |
| * 1. Прикладной протокол взаимодействия клиента и сервера | 13 |
| * 1. Разработка архитектуры приложения | 15 |
| * 1. Разработка интерфейса | 16 |
| 1. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ | 20 |
| * 1. Реализация программы | 20 |
| * 1. Тестовый пример работы системы | 34 |
| ЗАКЛЮЧЕНИЕ | 38 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 39 |
| ПРИЛОЖЕНИЯ | 40 |

### ВВЕДЕНИЕ

Архитектура «Клиент-сервер» – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер – это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Сервер – компьютер, обычно более мощный, чем компьютер-клиент, ожидает от клиентских программ запросы и предоставляет им свои ресурсы в виде данных или в виде сервисных функций. Клиент – компьютер, осуществляющий запрос к серверу на выполнение каких-либо действий или предоставление какой-либо информации. Модель функционирования такой системы заключается в следующем: клиент делает запрос серверу, который выполняет его и возвращает результат.

### 1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ И ПОСТАНОВКА ЗАДАЧ ПРОЕКТИРОВАНИЯ ПРОГРАММЫ

# 1.1 Описание предметной области

Целью курсового проекта является создание клиент-серверного программного обеспечения, позволяющего на компьютере клиента управлять сервисами (службами) операционной системы сервера.

Клиент-сервер – это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.

Для сетевого взаимодействия на компьютере сервера/клиента сначала создаётся сокет (с указанием протокола передачи данных). После, в случае сервера, он привязывается к определённому порту и ожидает подключения клиентов (TCP) или просто поступления новых данных от клиентов (UDP). Клиенту же, для взаимодействия с сервером, нужно подключиться (TCP) или послать какие-то данные (UDP).

Сокет – абстрактный объект, представляющий конечную точку соединения. Проще говоря, в данном случае это виртуальный интерфейс обмена данными между различными устройствами. Первых вариантов реализации сокетов было достаточно много, поэтому в Калифорнийском университете в Беркли был придуман общий стандарт «сокетов Беркли». Вот список основных функций для работы с ними:

|  |  |
| --- | --- |
| **Общие** |  |
| Socket | Создать новый сокет и вернуть файловый дескриптор |
| Send | Отправить данные по сети |
| Receive | Получить данные из сети |
| Close | Закрыть соединение |
|  |  |
| **Серверные** |  |
| Bind | Связать сокет с IP-адресом и портом |
| Listen | Объявить о желании принимать соединения. Слушает порт и ждет когда будет установлено соединение |
| Accept | Принять запрос на установку соединения |
|  |  |
| **Клиентские** |  |
| Connect | Установить соединение |

## **Таблица 1.1** – список функций для работы с сокетами Беркли

Подробную информацию о сетевых протоколах TCP/UDP вы можете найти здесь [1-3].

Сервисы ОС – приложения, автоматически (если настроено) запускаемые операционной системой при запуске, обычно не взаимодействующие с пользователем. В Unix-подобных системах называются «демонами», а в ОС Windows – «службами».

Для управления службами в Windows имеется специальный «Менеджер управления службами» (“Service control manager“, SCM).

## https://rsdn.org/article/baseserv/svcadmin-1/svcarch.gif

## **Рис. 1.1** – Архитектура системных служб Windows NT

Подробнее о службах Windows вы можете найти в [4-5].

В \*nix-системах существует различные варианты реализации служб. Одна из первых – программа инициализации “init”. Её скрипты нельзя назвать полноценными службами, поэтому во многиз современных ОС Linux существует подсистема управления “Systemd”, а в ОС Solaris “Service Management Facility”.

В процессе разработки было решено отказаться от реализации программы для ОС Linux, так как автор не уверен, что везде можно найти “Systemd”.

# 1.2 Постановка задачи проектирования

Клиент курсового проекта должен получать данные о службах:

* Внутреннее имя службы
* Отображаемое имя службы
* Состояние (статус) службы
* Тип запуска

Основные функции приложения клиента:

* Подключение к серверу с последующим обменом информацией с сервером посредством транспортного протокола UDP;
* Предоставление пользователю список служб сервера, их состояние и конфигурацию (тип запуска);
* Изменение состояния служб и типа запуска;

Основные функции приложения сервера:

* Предоставление возможности клиентскому приложению получения запрашиваемых данных;
* Обеспечение надежности, согласованности и защищенности предоставляемых данных;
* Выполнение указаний клиента по отношению к получаемым данным.

Кроме этого, каждое приложение должно вести логи событий.

В связи с тем, что не каждый Linux содержит инструментарий для управления сервисами (демонами) в реальном времени, было решено, что программа будет написана лишь для Windows.

### 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

# 2.1 Алгоритм работы программы

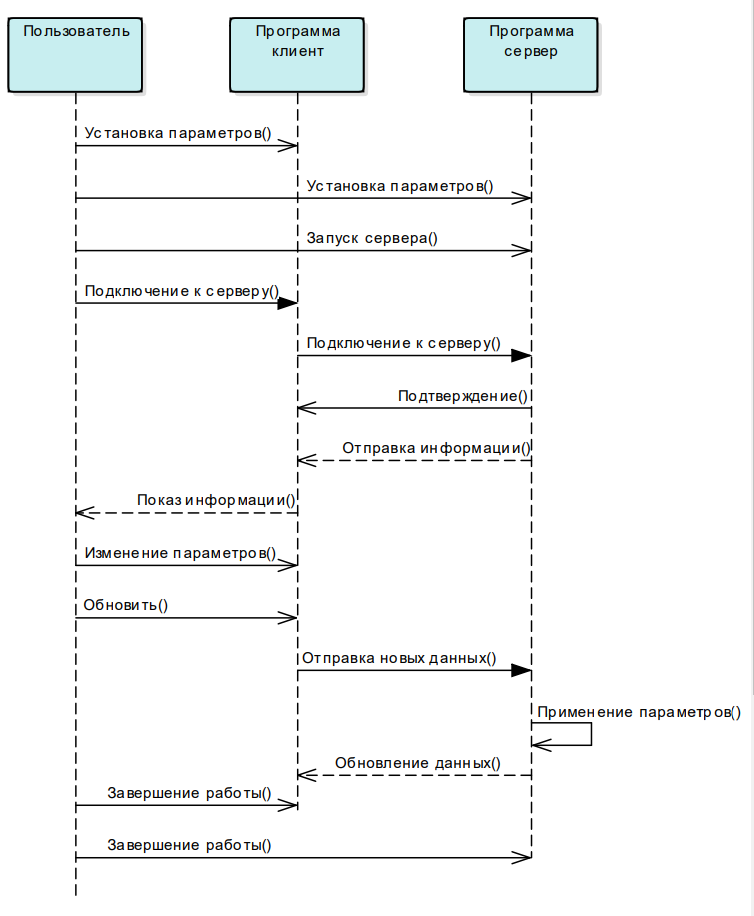
В ходе проектирования было принято решение, что наилучшим вариантом будет объединение клиента и сервера в одно приложение.

Алгоритм работы клиента и сервера может быть представлен в двух формах: как список действий, предоставляемых пользователю, и как программная реализация этого списка. Первая форма определяется условием курсового проекта, а вторая форма – используемыми средствами разработки и отчасти используемым протоколом, который также указан в условии.

В конечном счете, разрабатываемая программа будет позволять пользователю осуществлять следующие действия:

* Запуск сервера;
* Запуск клиента;
* Подключение к серверу;
* Получение от сервера информации о параметрах служб;
* Просмотр полученной информации;
* Сортировка полученной информации;
* Изменение параметров служб сервера.

Каждое действие в данном списке является последовательностью программных операций и/или сообщений между приложением клиента и приложением сервера. Таким образом, изобразив на диаграмме действия из списка с соответствующими сообщениями, их реализующими, мы получим алгоритм работы программы (рисунок 2.1).



## **Рисунок 2.1** – алгоритм

Вышеперечисленные действия иллюстрируют общую последовательность работы программы. Очевидно, что при добавлении функциональных модулей или повышении детализации этапов указанных процессов в схеме алгоритма произойдут соответствующие изменения.

# 2.2 Макет пользовательского интерфейса

Основным окном является диалог выбора режима работы приложения: как сервера или как клиента. Для выбора используются две кнопки combobox. Для настройки запуска/подключения существуют поля редактирования хоста и порта сервера. Хост представляет собой поле ввода IP-адреса, а порт – поле для ввода 16-битного числа. При активации кнопки режима сервера поле хоста становится недоступным для ввода.

Программа является оконным приложением, способным работать в соответствии с выбранным режимом. То есть, и приложение клиента, и приложение сервера является одной и той же программой, запущенной в соответствующем режиме.

При работе в любом из режимов диалог приложения содержит элементы интуитивно понятные в использовании и позволяющие выполнять следующие функции:

* Переключение между режимами работы.
* Осуществлять ввод необходимых данных для запуска приложения в соответствии с выбранным режимом.
* Осуществлять непосредственно запуск работы приложения.
* Макеты главного окна приложения, запущенного в режиме сервера и клиента изображены на рисунках 2.2 и 2.3 соответственно.

|  |  |
| --- | --- |
| **Рисунок 2.2** – интерфейс главного окна (режим сервера) | **Рисунок 2.3** – интерфейс главного окна (режим клиента) |

В случае успешного подключения клиента к серверу, вместо главного окна клиента появляется окно для работы с полученными данными. Данное окно должно содержать такие элементы управления, которые позволят предельно рационально использовать пространство рабочего стола, и при этом предоставлять пользователю возможность просматривать и изменять полученные данные наиболее удобным образом. Макет окна-обозревателя данных представлен на рисунке 2.4.

## 

## **Рисунок 2.4** – макет окна-обозревателя

# 2.3 Выбор и обоснование средств разработки

# 2.3.1 Обоснование выбора языка программирования

В качестве языка программирования для разработки программы выбран С++. Выбор данного языка обусловлен следующими причинами:

* Самыми используемыми языками программирования являются языки, чей синтаксис происходит от синтаксиса языка C. Выбор   
  C-подобного языка улучшает шанс быстрого развития проекта при дальнейшей модернизации его сторонними программистами;
* C++ – расширение языка C – имеет поддержку объектно-ориентированного программирования (ООП), что позволяет создавать и использовать классы и объекты;
* Наличие библиотек для создания клиент-серверного приложения.

# 2.3.2 Обоснование выбора среды разработки

Для разработки программы с использованием вышеупомянутого языка изначально была выбрана такая среда разработки как Bloodshed Dev-C++. Причинами такого выбора послужили:

* Компилятор GCC (после компиляции программе не требуются дополнительные библиотеки времени выполнения – используется функции из стандартной библиотеки «msvcrt.dll»);
* Оптимальное для автора количество настроек;
* Удобное средство автоматического форматирования кода – AStyle;
* Наличие используемого в проекте, портированного posix-совместимого, модуля Pthread (в Microsoft Visual Studio его нет).

# 2.3.3 Обоснование выбора средств разработки пользовательского интерфейса

Наиболее гибкой, совместимой и достаточно универсальной средой для взаимодействия с OS Windows является её API – интерфейс прикладного программирования. Кроме этого, необходимый интерфейс SCM, для управления службами, также является частью WinAPI.

# 2.3.4 Выбор средств разработки программы

В итоге, имеем:

* Язык разработки – C++;
* Среда разработки – Bloodshed Dev-C++ (Code Blocks);
* Средства разработки пользовательского интерфейса – WinAPI.

# 2.4 Прикладной протокол взаимодействия клиента и сервера

Для взаимодействия клиента и сервера используются некоторые данные, которые описывает следующая структура:

## **Листинг 2.1** – структура Datagram

1. //
2. // СТРУКТУРА: Datagram
3. //
4. // СОДЕРЖИМОЕ: передаваемая комманда и данные
5. //
6. struct Datagram {
7. DWORD id;
8. DWORD cmd\_cou;
9. char data[];
10. };

Примечание: старший байт поля cmd\_cou является командой серверу, а младшие – количество служб в передаваемом клиенту списке.

## **Листинг 2.2** – идентификатор протокола и доступные команды

1. #define PROTOCOLID \*((DWORD \*)"\17VS\2")
2. #define CMD\_ANY 0
3. #define CMD\_LIST 0x0C000000
4. #define CMD\_SET 0x18000000

По команде CMD\_LIST сервер возвращает массив структур:

## **Листинг 2.3** – структура возвращаемого элемента сервером

1. ; Синтаксис ассемблера
2. **State** db ?
3. **Name** db ?
4. db 0
5. **ViewName** db ?
6. db 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Биты | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Описание | Ошибка при получении типа запуска | Ошибка при получении состояния | Тип запуска | | | Состояние службы | | |

## **Таблица 2.1** – содержимое поля State

Для установки состояния службы сервера, клиент отправляет c CMD\_SET часть структуры CMD\_LIST:

## **Листинг 2.4 –** структура передаваемых данных серверу

1. ; Синтаксис ассемблера
2. **New\_State** db ?
3. **Name** db ?
4. db 0

# 2.5 Разработка архитектуры приложения

Клиент-серверное приложение для управления сервисами функционирует на основе модульной архитектуры, имеющей иерархическую структуру, схема которой изображена на рисунке 2.5.



## **Рисунок 2.5** – компонентная схема системы

На рисунке 2.5. изображены следующие модули:

1. mysvcctl – основной модуль программы (отвечает в основном за пользовательский интерфейс);
2. engine – модуль основных функций;
   1. services – модуль для работы со службами;
   2. log – модуль для ведения лога;
   3. udpsock и sock – модули для работы с сокетами (UDP);

# 2.6 Разработка интерфейса

Согласно разработанным макетам, программный интерфейс состоит из двух окон: главного и обозревателя. Для реализации всех необходимых функций, главное окно содержит следующие элементы управления:

* Два combobox с соответствующими подписями «Сервер» и «Клиент», которые используются для выбора работы программы. По умолчанию выбран режим сервера.
* Два поля с подписями «Хост» и «Шлюз», необходимые для получения от пользователя данных для запуска сервера либо подключения к нему. Поле ввода хоста доступно только в режиме работы «Клиент» и имеет соответствующий шаблон для ввода.
* Две кнопки, выполняющие функции, соответствующие выбранному режиму работы. Во время работы программы в режиме «Сервера» они используются для запуска и остановки сервера соответственно. В случае работы в режиме «Клиент» первая кнопка используется для подключения к серверу. Подписи на кнопках также соответствуют режиму работы программы.

Изображения работы программы в режимах «Сервер» и «Клиент» изображены на рисунках 2.6 и 2.7, соответственно.

## 

## **Рисунок 2.6** – главное окно в режиме работы «Сервер»

## 

## **Рисунок 2.7** – главное окно в режиме работы «Клиент»

В целях реализации функционала окна-обозревателя данное окно содержит в себе следующие элементы управления:

* Кнопка «Обновить», которая используется для обновления информации (применения изменений для сервера и отображения информации о службах в соответствующем виде). Также данная кнопка устанавливает размеры остальных элементов управления данного окна в стандартные значения, если они были изменены.
* Так как одной из лучших форм представления информации о системных службах является таблица, окно-обозреватель использует элемент listview для её реализации. Таблица содержит колонки: «Внутр. имя» (внутреннее имя службы), «Имя», «Статус» и «Тип запуска».
* Для изменения параметров конкретных служб используется контекстное меню, вызов которого осуществляется посредством нажатия правой кнопки мыши по соответствующей службе. Данное меню позволяет изменить статус службы и её тип запуска, для чего имеет соответствующие пункты.

Внешний вид окна-обозревателя изображен на рисунке 2.8. На рисунке 2.9 изображено контекстное меню в момент его использования.

## 

## **Рисунок 2.8** – окно-обозреватель служб

## 

## **Рисунок 2.9** – использование контекстного меню

### 3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

# 3.1 Реализация программы

Поскольку интерфейс программы курсового проекта разработан с использованием средств WinAPI, точкой входа в программу является функция WinMain. Код данной функции представлен в листинге 3.1.

## **Листинг 3.1** – функция WinMain()

1. //
2. // ФУНКЦИЯ: int WinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
3. //
4. // НАЗНАЧЕНИЕ: точка входа для приложения
5. //
6. int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
7. LPTSTR lpCmdLine, int nCmdShow) {
8. UNREFERENCED\_PARAMETER(hPrevInstance);
9. UNREFERENCED\_PARAMETER(lpCmdLine);
10. hInst = hInstance;
11. InitCommonControls();
12. hClientWnd = 0;
13. LoadString(hInstance, IDS\_WNDCLASS, szWndClass, MAX\_LOADSTRING);
14. if (!RegisterWndClass()) {
15. MessageBox(0, "Cannot register windows class!",
16. szError, MB\_ICONERROR | MB\_OK);
17. return EXIT\_FAILURE;
18. }
19. if (!InitializeSockets()) {
20. MessageBox(0, "Cannot initialize sockets!",
21. szError, MB\_ICONERROR | MB\_OK);
22. return EXIT\_FAILURE;
23. }
24. LoadString(hInstance, IDS\_START, szStart, MAX\_LOADSTRING);
25. LoadString(hInstance, IDS\_CONNECT, szConnect, MAX\_LOADSTRING);
26. LoadString(hInstance, IDS\_STOP, szStop, MAX\_LOADSTRING);
27. LoadString(hInstance, IDS\_ERROR, szError, MAX\_LOADSTRING);
28. LoadString(hInstance, IDS\_ERRSTART, szErrStart, MAX\_LOADSTRING);
29. LoadString(hInstance, IDS\_ERRLDLIST, szErrLdList, MAX\_LOADSTRING);
30. LoadSettings();
31. int r;
32. do {
33. r = DialogBox(hInst, MAKEINTRESOURCE(IDD\_CHOOSE), 0, ChooseDlgProc);
34. if (r) {
35. client = new Client();
36. if ( !(hClientWnd = InitClientWnd()) )
37. break;
38. ShowWindow(hClientWnd, SW\_SHOW);
39. UpdateWindow(hClientWnd);
40. MSG msg;
41. while (GetMessage(&msg, NULL, 0, 0)) {
42. TranslateMessage(&msg);
43. DispatchMessage(&msg);
44. }
45. client->~Client();
46. delete client;
47. }

## **Листинг 3.1** – продолжение

1. } while (r);
2. ShutdownSockets();
3. SaveSettings();
4. return EXIT\_SUCCESS;
5. }

Функция WinMain осуществляет следующие действия (в порядке выполнения кода):

* Регистрирует класс окна обозревателя и инициализирует систему для работы с сокетами;
* Загружает ключевые строки из файла ресурсов;
* Вызывает диалог выбора и ожидает результата после его закрытия;
* Если была нажата кнопка «Соединиться», начинает инициализироваться объект клиента, создаётся окно обозревателя служб и обрабатываются сообщения вплоть до его закрытия;
* Если окно диалога было закрыто, программа сохраняет текущие настройки в файл конфигурации и завершает свою работу

В связи с тем, что основной темой исследования данного курсового проекта не является работа с WinAPI, исходный текст основного модуля «mysvcctl.cpp» вы можете найти в ПРИЛОЖЕНИИ A.

В модуле «engine.cpp» реализованы основные служебные функции программы.

## **Листинг 3.2** – заголовочный файл «engine.h»

1. //
2. // ЗАГОЛОВОК: ENGINE.H
3. //
4. // ОПИСАНИЕ: реализация работы сервера и клиента
5. //
6. #ifndef engine\_h\_
7. #define engine\_h\_
8. #include "udpsock.h"
9. #define SocketType UdpSocket
10. //
11. // СТРУКТУРА: ListItem
12. //
13. // СОДЕРЖИМОЕ: элемент распакованного списка клиента
14. //
15. struct ListItem {
16. BYTE state;
17. char \*name, \*viewname;
18. };

## **Листинг 3.2** – продолжение

1. //
2. // СТРУКТУРА: ServerParams
3. //
4. // СОДЕРЖИМОЕ: общие параметры для потоков сервера
5. //
6. struct ServerParams {
7. SocketType sock;
8. bool active;
9. pthread\_mutex\_t mutex;
10. };
11. //
12. // КЛАСС: Server
13. //
14. // НАЗНАЧЕНИЕ: управление основным потоком сервера
15. //
16. class Server {
17. private:
18. pthread\_t thread;
19. ServerParams param;
20. public:
21. bool Start(unsigned short port = 0);
22. bool IsWorking();
23. void Stop();
24. ~Server();
25. };
26. //
27. // КЛАСС: Client
28. //
29. // НАЗНАЧЕНИЕ: реализация клиента
30. //
31. class Client {
32. private:
33. ClientParams param;
34. public:
35. Client();
36. bool GetList(Address addr);
37. void SetSvc(Address addr, unsigned int idx, BYTE state);
38. unsigned int ListSize();
39. ListItem\* GetItem(unsigned int idx);
40. };
41. // Инициализировать работу с сокетами
42. bool InitializeSockets();
43. // Завершить работу с сокетами
44. void ShutdownSockets();
45. #endif

Для работы с сокетами были разработаны специальные классы и модули. Это «Sock.cpp» и «UdpSock.cpp».

## **Листинг 3.3** – заголовочный файл «sock.h»

1. //
2. // ЗАГОЛОВОК: SOCK.H
3. //
4. // ОПИСАНИЕ: абстрактный класс Socket
5. //
6. #ifndef sock\_h\_
7. #define sock\_h\_
8. #define PLATFORM\_WINDOWS 1
9. #define PLATFORM\_MAC 2
10. #define PLATFORM\_UNIX 3
11. #if defined(\_WIN32)
12. #define PLATFORM PLATFORM\_WINDOWS
13. #elif defined(\_\_APPLE\_\_)
14. #define PLATFORM PLATFORM\_MAC
15. #else
16. #define PLATFORM PLATFORM\_UNIX
17. #endif
18. #if PLATFORM == PLATFORM\_WINDOWS
19. #include <winsock2.h>
20. #pragma comment( lib, "wsock32.lib" )
21. #elif PLATFORM == PLATFORM\_MAC || PLATFORM == PLATFORM\_UNIX
22. #include <sys/socket.h>
23. #include <netinet/in.h>
24. #include <fcntl.h>
25. #define closesocket(socket) close(socket)
26. #else
27. #error unknown platform!
28. #endif
29. //
30. // КЛАСС-СТРУКТУРА: Address
31. //
32. // СОДЕРЖИМОЕ: адрес и порт
33. //
34. struct Address {
35. union {
36. struct {
37. unsigned char a, b, c, d;
38. };
39. unsigned int addr;
40. };
41. unsigned short port;
42. Address() {
43. addr = port = 0;
44. }
45. Address(unsigned int addr, unsigned short port) {
46. this->addr = addr;
47. this->port = port;
48. }
49. bool operator == (Address &x) {
50. return (x.addr == addr) && (x.port == port);
51. }

## **Листинг 3.3** – продолжение

1. bool operator != (Address &x) {
2. return !(\*this == x);
3. }
4. };
5. //
6. // КЛАСС: Socket
7. //
8. // НАЗНАЧЕНИЕ: реализация сокета
9. //
10. class Socket {
11. protected:
12. int sock;
13. int consock; // сокет соединения
14. bool isserver;
15. public:
16. Socket();
17. ~Socket();
18. void Wait();
19. bool SetNonBlocking();
20. virtual bool Open() = 0;
21. bool Bind(Address addr);
22. void Close();
23. bool IsOpen();
24. bool IsServer();
25. virtual bool Accept() = 0;
26. virtual bool Connect(Address addr) = 0;
27. virtual bool IsConnected() = 0;
28. virtual void Disconnect() = 0;
29. virtual bool Send(void \*data, int size) = 0;
30. virtual int Receive(void \*data, int size) = 0;
31. };
32. // Приостановить программу на некоторое время
33. void wait( float seconds );
34. #endif

## **Листинг 3.4** – заголовочный файл «udpsock.h»

1. //
2. // ЗАГОЛОВОК: UDPSOCK.H
3. //
4. // ОПИСАНИЕ: класс UdpSocket
5. //
6. #ifndef udpsock\_h\_
7. #define udpsock\_h\_
8. #include "sock.h"
9. class UdpSocket : public Socket {
10. public:
11. bool Open();
12. bool Bind(Address addr);

## **Листинг 3.4** – продолжение

1. bool Accept();
2. bool Connect(Address addr);
3. bool IsConnected();
4. bool Send(void \*data, int size);
5. int Receive(void \*data, int size);
6. void Disconnect();
7. };
8. #endif

Для работы со службами был также создан отдельный модуль. Вот содержимое файла-заголовка модуля:

## **Листинг 3.5** – заголовочный файл «services.h»

1. //
2. // ЗАГОЛОВОК: SERVICES.H
3. //
4. // ОПИСАНИЕ: интерфейс управления службами Windows
5. //
6. #ifndef services\_h\_
7. #define services\_h\_
8. #include "include.h"
9. #define SVC\_TIMEOUT 10000
10. */\**
11. *#define SERVICE\_STOPPED 0x00000001*
12. *#define SERVICE\_START\_PENDING 0x00000002*
13. *#define SERVICE\_STOP\_PENDING 0x00000003*
14. *#define SERVICE\_RUNNING 0x00000004*
15. *#define SERVICE\_CONTINUE\_PENDING 0x00000005*
16. *#define SERVICE\_PAUSE\_PENDING 0x00000006*
17. *#define SERVICE\_PAUSED 0x00000007*
18. *#define SERVICE\_CONTROL\_STOP 0x00000001*
19. *#define SERVICE\_CONTROL\_PAUSE 0x00000002*
20. *#define SERVICE\_CONTROL\_CONTINUE 0x00000003*
21. *#define SERVICE\_BOOT\_START 0x00000000*
22. *#define SERVICE\_SYSTEM\_START 0x00000001*
23. *#define SERVICE\_AUTO\_START 0x00000002*
24. *#define SERVICE\_DEMAND\_START 0x00000003*
25. *#define SERVICE\_DISABLED 0x00000004*
26. *\*/*
27. #define SERVICE\_CONTROL\_START 0x00000004
28. void \*SVC\_getEnum(DWORD &sz, DWORD &num);
29. int SVC\_GetStatus(char \*name);
30. int SVC\_SetStatus(char \*name, int flags);
31. #endif

# 3.2 Методика и результаты тестирования

Тестирование программы курсового проекта проводилось методом чёрного ящика. Сути данного метода заключается в том, что программное обеспечение рассматривается как «чёрный ящик», то есть анализируются только основные аспекты системы. Преимущества данного метода:

* эффективен для большого сегмента кода;
* более быстрое создание теста;
* простота восприятия.

Основным недостатком данного подхода является то, что в действительности выполняется выборочное число сценариев, что ограничивает тестируемую область. Также, в данном методе отсутствует чёткая спецификация, что затрудняет разработку тестовых сценариев.

Ниже представлены результат тестирования программы:

|  |
| --- |
| 1. Действие: запуск программ.  Ожидаемый результат: появление диалогов для выбора режима работы;  Полученный результат: после запуска двух экземпляров программы появились два окна с элементами выбора режима работы;  Метка: Выполнено. |
| 2. Действие: настройка и запуск сервера.  Ожидаемый результат: после ввода порта и нажатия на кнопку «Запустить», сервер станет активным. Кнопка «Запустить» станет не активна, а кнопка «Остановить» – активна;  Полученный результат: как и ожидалось;  Метка: Выполнено. |
| 3. Действие: настройка и запуск клиента, получение списка.  Ожидаемый результат: после ввода хоста (в данном случае это localhost – 127.0.0.1) и порта сервера и нажатия на кнопку «Соединиться», клиент подключится к серверу и получит список служб. Далее отобразится окно обозревателя служб и автоматически отобразит полученный список. После нажатия кнопки «Обновить» список, соответственно, обновился;  Полученный результат: всё так и было;  Метка: Выполнено. |
| 4. Действие: изменение состояния службы.  Ожидаемый результат: изменение отображаемой информации о параметрах изменяемой службы;  Полученный результат: после обновления списка состояние службы изменилось соответствующим образом;  Метка: Выполнено. |
| 5. Действие: изменение типа запуска службы.  Ожидаемый результат: изменение типа запуска службы;  Полученный результат: после обновления списка конфигурация службы изменилась соответствующим образом;  Метка: Выполнено. |
| 6. Завершение работы приложений.  Ожидаемый результат: после закрытия всех окон приложений не появится не одной ошибки;  Полученный результат: ни одной ошибки не появилось;  Метка: Выполнено. |

Согласно результатам тестирования программы, проверяемые действия программы оправдали все ожидаемые результаты. Это означает, что основные аспекты программы работают правильно как визуально, так и в плане реализации. Тем не менее, проверка методом «Черного ящика» не гарантирует полного отсутствия изъянов в коде и достижения максимальной оптимизации работы программы.

### ЗАКЛЮЧЕНИЕ

В процессе выполнения данного курсового проекта было разработано клиент-серверное приложение «Удаленный менеджер сервисов». В результате выполнения были приобретены знания и умения разработки клиент-серверных приложений, программирования сокетов и сетевых соединений.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. «Сокеты — Сетевое программирование» [электронный ресурс] // «Сетевое программирование» [сайт]. – Режим доступа: https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/intro.html
2. «www.ПЕРВЫЕ ШАГИ.ru :: Шаг 1 - Что такое Windows Sockets» [электронный ресурс] // «Первые шаги» [сайт]. – Режим доступа: http://www.firststeps.ru/mfc/net/socket/r.php?1
3. «Сетевое программирование для разработчиков игр. Часть 1\_ UDP vs. TCP \_ Хабр» [электронный ресурс] // «Habr» [сайт]. – Режим доступа: https://habr.com/ru/post/209144/
4. «Управление системными службами Windows NT» [электронный ресурс] // RSDN [сайт]. – Режим доступа: https://rsdn.org/article/baseserv/svcadmin-1.xml
5. «Управление системными службами Windows NT2» [электронный ресурс] // RSDN [сайт]. – Режим доступа: https://rsdn.org/article/baseserv/svcadmin-2.xml

### ПРИЛОЖЕНИЕ A

# Исходный код модуля «mysvcctl.cpp»

1. //
2. // ОСНОВНОЙ МОДУЛЬ: MYSVCCTL.CPP
3. //
4. // ОПИСАНИЕ ПРОГРАММЫ:
5. // Программа позволяет управлять службами на удалённом компьютере (в режиме
6. // клиента) или предоставлять доступ другим программам с поддержкой специального
7. // протокола (в режиме сервера).
8. //
9. // ЦЕЛЕВАЯ ПЛАТФОРМА: WIN32
10. //
11. #include "include.h"
12. #include "resource.h"
13. #include "engine.h"
14. #include "services.h"
15. pthread\_mutex\_t mutex;
16. //
17. // Прототипы функций модуля
18. //
19. INT\_PTR CALLBACK ChooseDlgProc(HWND, UINT, WPARAM, LPARAM);
20. ATOM RegisterWndClass();
21. HWND InitClientWnd();
22. bool InitListView(HWND);
23. LRESULT CALLBACK ClientWndProc(HWND, UINT, WPARAM, LPARAM);
24. void RefreshWindow(HWND);
25. void UpdateClientMenu(HMENU hMenu, BYTE state);
26. UINT LV\_Selection();
27. //
28. // Глобальные константы
29. //
30. const TCHAR \*szNull = TEXT("");
31. const TCHAR \*szConfigFile = TEXT("mysvcctl.cfg");
32. //
33. // Глобальные переменные
34. //
35. HINSTANCE hInst;
36. HWND hClientWnd;
37. HWND hListView;
38. Address addr;
39. TCHAR szWndClass[MAX\_LOADSTRING];
40. TCHAR szStart[MAX\_LOADSTRING];
41. TCHAR szConnect[MAX\_LOADSTRING];
42. TCHAR szStop[MAX\_LOADSTRING];
43. TCHAR szError[MAX\_LOADSTRING];
44. TCHAR szErrStart[MAX\_LOADSTRING];
45. TCHAR szErrLdList[MAX\_LOADSTRING];
46. Server \*server;
47. Client \*client;
48. //
49. // ФУНКЦИЯ: void LoadSettings()
50. //
51. // НАЗНАЧЕНИЕ: загружает настройки из файла конфигурации
52. //
53. void LoadSettings() {
54. FILE \*f = fopen(szConfigFile, "rb");
55. if (f) {
56. struct {
57. DWORD a;
58. WORD p;
59. } adr;
60. fread(&adr, sizeof(adr), 1, f);
61. fclose(f);
62. addr = Address(adr.a, adr.p);
63. }
64. }
65. //
66. // ФУНКЦИЯ: void SaveSettings()
67. //
68. // НАЗНАЧЕНИЕ: сохраняет настройки в файл конфигурации
69. //
70. void SaveSettings() {
71. FILE \*f = fopen(szConfigFile, "wb");
72. if (f) {
73. struct {
74. DWORD a = addr.addr;
75. WORD p = addr.port;
76. } adr;
77. fwrite(&addr, sizeof(addr), 1, f);
78. fclose(f);
79. }
80. }
81. //
82. // ФУНКЦИЯ: int WinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
83. //
84. // НАЗНАЧЕНИЕ: точка входа для приложения
85. //
86. int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
87. LPTSTR lpCmdLine, int nCmdShow) {
88. UNREFERENCED\_PARAMETER(hPrevInstance);
89. UNREFERENCED\_PARAMETER(lpCmdLine);
90. hInst = hInstance;
91. InitCommonControls();
92. hClientWnd = 0;
93. LoadString(hInstance, IDS\_WNDCLASS, szWndClass, MAX\_LOADSTRING);
94. if (!RegisterWndClass()) {
95. MessageBox(0, "Cannot register windows class!",
96. szError, MB\_ICONERROR | MB\_OK);
97. return EXIT\_FAILURE;
98. }
99. if (!InitializeSockets()) {
100. MessageBox(0, "Cannot initialize sockets!",
101. szError, MB\_ICONERROR | MB\_OK);
102. return EXIT\_FAILURE;
103. }
104. LoadString(hInstance, IDS\_START, szStart, MAX\_LOADSTRING);
105. LoadString(hInstance, IDS\_CONNECT, szConnect, MAX\_LOADSTRING);
106. LoadString(hInstance, IDS\_STOP, szStop, MAX\_LOADSTRING);
107. LoadString(hInstance, IDS\_ERROR, szError, MAX\_LOADSTRING);
108. LoadString(hInstance, IDS\_ERRSTART, szErrStart, MAX\_LOADSTRING);
109. LoadString(hInstance, IDS\_ERRLDLIST, szErrLdList, MAX\_LOADSTRING);
110. LoadSettings();
111. int r;
112. do {
113. r = DialogBox(hInst, MAKEINTRESOURCE(IDD\_CHOOSE), 0, ChooseDlgProc);
114. if (r) {
115. client = new Client();
116. if ( !(hClientWnd = InitClientWnd()) )
117. break;
118. ShowWindow(hClientWnd, SW\_SHOW);
119. UpdateWindow(hClientWnd);
120. MSG msg;
121. while (GetMessage(&msg, NULL, 0, 0)) {
122. TranslateMessage(&msg);
123. DispatchMessage(&msg);
124. }
125. client->~Client();
126. delete client;
127. }
128. } while (r);
129. ShutdownSockets();
130. SaveSettings();
131. return EXIT\_SUCCESS;
132. }
133. //
134. // ФУНКЦИЯ: ATOM RegisterWndClass()
135. //
136. // НАЗНАЧЕНИЕ: регистрирует класс окна
137. //
138. // ВОЗВРАТИТЬ: флаг успеха выполнения операции
139. //
140. ATOM RegisterWndClass() {
141. WNDCLASSEX wcex;
142. ATOM aReturn;
143. wcex.cbSize = sizeof(WNDCLASSEX);
144. wcex.style = CS\_HREDRAW | CS\_VREDRAW;
145. wcex.lpfnWndProc = (WNDPROC)ClientWndProc;
146. wcex.cbClsExtra = 0;
147. wcex.cbWndExtra = 0;
148. wcex.hInstance = hInst;
149. wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);
150. wcex.hbrBackground = (HBRUSH) (COLOR\_WINDOW + 1);
151. wcex.lpszMenuName = MAKEINTRESOURCE(IDM\_MAINMENU);
152. wcex.lpszClassName = szWndClass;
153. wcex.hIcon = LoadIcon(hInst, IDI\_APPLICATION);
154. wcex.hIconSm = (HICON) LoadImage(hInst,
155. IDI\_APPLICATION, IMAGE\_ICON,
156. 16, 16, 0);
157. aReturn = RegisterClassEx(&wcex);
158. // Поддержка для предыдущих версий Windows(NT 4.5? =)
159. if (!aReturn) {
160. WNDCLASS wc;
161. wc.style = 0;
162. wc.lpfnWndProc = (WNDPROC)ClientWndProc;
163. wc.cbClsExtra = 0;
164. wc.cbWndExtra = 0;
165. wc.hInstance = hInst;
166. wc.hIcon = LoadIcon(hInst, IDI\_APPLICATION);
167. wc.hCursor = LoadCursor(NULL, IDC\_ARROW);
168. wc.hbrBackground = (HBRUSH) (COLOR\_WINDOW + 1);
169. wc.lpszMenuName = MAKEINTRESOURCE(IDM\_MAINMENU);
170. wc.lpszClassName = szWndClass;
171. aReturn = RegisterClass(&wc);
172. }
173. return aReturn;
174. }
175. //
176. // ФУНКЦИЯ: INT\_PTR CALLBACK ChooseDlgProc(HWND, UINT, WPARAM, LPARAM)
177. //
178. // ВОЗВРАТИТЬ: 0 - закрытие диалога,
179. // -1 - переключение в режим клиента (при успешном соединении)
180. //
181. INT\_PTR CALLBACK ChooseDlgProc(HWND hDlg, UINT msg, WPARAM wParam,
182. LPARAM lParam) {
183. BOOL fError;
184. switch (msg) {
185. case WM\_INITDIALOG:
186. server = new Server();
187. DWORD adr;
188. WORD port;
189. SendMessage(GetDlgItem(hDlg, IDD1\_IP),
190. IPM\_SETADDRESS, 0, (LPARAM) addr.addr);
191. SetDlgItemInt(hDlg, IDD1\_PORT, addr.port, false);
192. SendMessage(GetDlgItem(hDlg, IDD1\_SERVER), BM\_SETCHECK, TRUE, 0);
193. SendMessage(GetDlgItem(hDlg, IDD1\_CLIENT), BM\_SETCHECK, FALSE, 0);
194. EnableWindow(GetDlgItem(hDlg, IDD1\_STOP), FALSE);
195. goto idd1\_server;
196. case WM\_COMMAND:
197. switch (LOWORD(wParam)) {
198. case IDD1\_START:
199. SendMessage(GetDlgItem(hDlg, IDD1\_IP),
200. IPM\_GETADDRESS, 0, (LPARAM) &adr);
201. port = GetDlgItemInt(hDlg, IDD1\_PORT, &fError, false);
202. addr = Address(adr, port);
203. SaveSettings();
204. if (SendMessage(GetDlgItem(hDlg, IDD1\_CLIENT),
205. BM\_GETCHECK, 0, 0))
206. EndDialog(hDlg, -1);
207. if (SendMessage(GetDlgItem(hDlg, IDD1\_SERVER),
208. BM\_GETCHECK, 0, 0)) {
209. if (server->Start(addr.port)) {
210. EnableWindow(GetDlgItem(hDlg, IDD1\_START), FALSE);
211. EnableWindow(GetDlgItem(hDlg, IDD1\_STOP), TRUE);
212. EnableWindow(GetDlgItem(hDlg, IDD1\_CLIENT), FALSE);
213. EnableWindow(GetDlgItem(hDlg, IDD1\_PORT), FALSE);
214. } else {
215. MessageBox(hDlg, szErrStart, szError,
216. MB\_ICONERROR | MB\_OK);
217. }
218. }
219. break;
220. case IDD1\_STOP:
221. server->Stop();
222. EnableWindow(GetDlgItem(hDlg, IDD1\_START), TRUE);
223. EnableWindow(GetDlgItem(hDlg, IDD1\_STOP), FALSE);
224. EnableWindow(GetDlgItem(hDlg, IDD1\_CLIENT), TRUE);
225. EnableWindow(GetDlgItem(hDlg, IDD1\_PORT), TRUE);
226. break;
227. case IDD1\_SERVER:
228. idd1\_server:
229. EnableWindow(GetDlgItem(hDlg, IDD1\_IP), FALSE);
230. SetWindowText(GetDlgItem(hDlg, IDD1\_START), szStart);
231. SetWindowText(GetDlgItem(hDlg, IDD1\_STOP), szStop);
232. EnableWindow(GetDlgItem(hDlg, IDD1\_START),
233. !server->IsWorking());
234. EnableWindow(GetDlgItem(hDlg, IDD1\_CLIENT),
235. !server->IsWorking());
236. break;
237. case IDD1\_CLIENT:
238. EnableWindow(GetDlgItem(hDlg, IDD1\_IP), TRUE);
239. SetWindowText(GetDlgItem(hDlg, IDD1\_START), szConnect);
240. SetWindowText(GetDlgItem(hDlg, IDD1\_STOP), szNull);
241. EnableWindow(GetDlgItem(hDlg, IDD1\_STOP), FALSE);
242. break;
243. }
244. return true;
245. case WM\_CLOSE:
246. server->~Server();
247. delete server;
248. SendMessage(GetDlgItem(hDlg, IDD1\_IP),
249. IPM\_GETADDRESS, 0, (LPARAM) &adr);
250. port = GetDlgItemInt(hDlg, IDD1\_PORT, &fError, false);
251. addr = Address(adr, port);
252. EndDialog(hDlg, 0);
253. break;
254. }
255. return false;
256. }
257. //
258. // ФУНКЦИЯ: WHND InitClientWnd()
259. //
260. // НАЗНАЧЕНИЕ: создаёт окно и элементы управление, отображает окно
261. //
262. // ВОЗВРАТИТЬ: идетификатор окна / NULL
263. //
264. HWND InitClientWnd() {
265. bool err = false;
266. HWND hWnd = CreateWindow(szWndClass, szWndClass,
267. WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, 0,
268. 640, 480, NULL, NULL, hInst, NULL);
269. if (hWnd) {
270. if (!InitListView(hWnd))
271. err = true;
272. if (err) {
273. DestroyWindow(hWnd);
274. hWnd = NULL;
275. }
276. }
277. return hWnd;
278. }
279. //
280. // ФУНКЦИЯ: bool InitListView(HWND)
281. //
282. // НАЗНАЧЕНИЕ: создаёт элемент управления ListView
283. //
284. // ВОЗВРАТИТЬ: флаг успеха выполнения операции
285. //
286. bool InitListView(HWND hWnd) {
287. INT dwStyle = WS\_TABSTOP
288. | WS\_CHILD
289. | WS\_VISIBLE
290. | LVS\_SHOWSELALWAYS
291. | LVS\_SINGLESEL
292. | LVS\_REPORT
293. | LVS\_OWNERDATA
294. ;
295. hListView = CreateWindowEx(WS\_EX\_CLIENTEDGE,
296. WC\_LISTVIEW,
297. (LPCTSTR) NULL,
298. dwStyle,
299. 0, 0, 10, 10,
300. hWnd,
301. (HMENU) IDC\_LISTVIEW,
302. hInst,
303. NULL);
304. if (hListView) {
305. ListView\_SetExtendedListViewStyle (
306. hListView, LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES
307. );
308. LV\_COLUMN lvColumn;
309. TCHAR buf[MAX\_LOADSTRING];
310. lvColumn.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;
311. lvColumn.fmt = LVCFMT\_LEFT;
312. lvColumn.cx = 100;
313. lvColumn.pszText = buf;
314. for (UINT i = 0; i < IDS\_COL\_num; i++) {
315. LoadString(hInst, IDS\_COL+i, buf, MAX\_LOADSTRING);
316. ListView\_InsertColumn(hListView, i, &lvColumn);
317. }
318. }
319. return (hListView != NULL);
320. }
321. //
322. // ФУНКЦИЯ: LRESULT CALLBACK ClientWndProc(HWND, UINT, WPARAM, LPARAM)
323. //
324. // НАЗНАЧЕНИЕ: обрабатывает сообщения от окна клиента
325. //
326. LRESULT CALLBACK ClientWndProc(HWND hWnd, UINT message, WPARAM wParam,
327. LPARAM lParam) {
328. TCHAR buf[MAX\_LOADSTRING];
329. ListItem \*item;
330. switch (message) {
331. case WM\_CREATE:
332. pthread\_mutex\_lock(&mutex);
333. if (!client->GetList(addr))
334. MessageBox(hWnd, szErrLdList,
335. szError, MB\_ICONERROR | MB\_OK);
336. RefreshWindow(hWnd);
337. pthread\_mutex\_unlock(&mutex);
338. break;
339. case WM\_COMMAND:
340. switch (LOWORD(wParam)) {
341. case IDM\_REFRESH:
342. pthread\_mutex\_lock(&mutex);
343. if (!client->GetList(addr))
344. MessageBox(hWnd, szErrLdList,
345. szError, MB\_ICONERROR | MB\_OK);
346. RefreshWindow(hWnd);
347. pthread\_mutex\_unlock(&mutex);
348. break;
349. case IDM\_START:
350. pthread\_mutex\_lock(&mutex);
351. client->SetSvc( addr, LV\_Selection(),
352. SERVICE\_CONTROL\_START );
353. pthread\_mutex\_unlock(&mutex);
354. break;
355. case IDM\_PAUSE:
356. pthread\_mutex\_lock(&mutex);
357. client->SetSvc( addr, LV\_Selection(),
358. SERVICE\_CONTROL\_PAUSE );
359. pthread\_mutex\_unlock(&mutex);
360. break;
361. case IDM\_RESUME:
362. pthread\_mutex\_lock(&mutex);
363. client->SetSvc( addr, LV\_Selection(),
364. SERVICE\_CONTROL\_CONTINUE );
365. pthread\_mutex\_unlock(&mutex);
366. break;
367. case IDM\_STOP:
368. pthread\_mutex\_lock(&mutex);
369. client->SetSvc( addr, LV\_Selection(),
370. SERVICE\_CONTROL\_STOP );
371. pthread\_mutex\_unlock(&mutex);
372. break;
373. case IDM\_AUTO:
374. pthread\_mutex\_lock(&mutex);
375. client->SetSvc( addr, LV\_Selection(),
376. SERVICE\_AUTO\_START << 3 );
377. pthread\_mutex\_unlock(&mutex);
378. break;
379. case IDM\_DEMAND:
380. pthread\_mutex\_lock(&mutex);
381. client->SetSvc( addr, LV\_Selection(),
382. SERVICE\_DEMAND\_START << 3 );
383. pthread\_mutex\_unlock(&mutex);
384. break;
385. case IDM\_DISABLED:
386. pthread\_mutex\_lock(&mutex);
387. client->SetSvc( addr, LV\_Selection(),
388. SERVICE\_DISABLED << 3 );
389. pthread\_mutex\_unlock(&mutex);
390. break;
391. }
392. break;
393. case WM\_NOTIFY:
394. LV\_DISPINFO \*lpdi;
395. lpdi = (LV\_DISPINFO\*) lParam;
396. NMITEMACTIVATE \*lpNMITEMACTIVATE;
397. lpNMITEMACTIVATE = (NMITEMACTIVATE\*) lParam;
398. UINT itemid;
399. if ((((LPNMHDR)lParam)->hwndFrom) == hListView) {
400. switch (lpdi->hdr.code) {
401. case LVN\_GETDISPINFO:
402. itemid = lpdi->item.iItem;
403. if (lpdi->item.mask & LVIF\_TEXT) {
404. if ( client->ListSize() == 0 ) {
405. break;
406. }
407. pthread\_mutex\_lock(&mutex);
408. item = client->GetItem( itemid );
409. pthread\_mutex\_unlock(&mutex);
410. switch(lpdi->item.iSubItem) {
411. case 0:
412. lpdi->item.pszText = item->name;
413. break;
414. case 1:
415. lpdi->item.pszText = item->viewname;
416. break;
417. case 2:
418. if (item->state & 0x40)
419. LoadString(hInst, IDS\_UNKNOWN,
420. buf, MAX\_LOADSTRING);
421. else
422. LoadString(hInst,
423. IDS\_STATE+(item->state & 7),
424. buf, MAX\_LOADSTRING);
425. lpdi->item.pszText = buf;
426. break;
427. case 3:
428. if (item->state & 0x80)
429. LoadString(hInst, IDS\_UNKNOWN,
430. buf, MAX\_LOADSTRING);
431. else
432. LoadString(hInst,
433. IDS\_RUNTYPE +
434. ((item->state>>3) & 7), buf,
435. MAX\_LOADSTRING);
436. lpdi->item.pszText = buf;
437. break;
438. }
439. }
440. break;
441. case NM\_RCLICK:
442. pthread\_mutex\_lock(&mutex);
443. HMENU hmenuPopup;
444. hmenuPopup = GetSubMenu(
445. LoadMenu(hInst, "IDM\_POPUPMENU"), 0);
446. if (!hmenuPopup)
447. break;
448. ListItem \*item = client->GetItem( LV\_Selection() );
449. UpdateClientMenu( hmenuPopup, item->state );
450. pthread\_mutex\_unlock(&mutex);
451. POINT cursor;
452. GetCursorPos(&cursor);
453. TrackPopupMenu(hmenuPopup,
454. TPM\_LEFTALIGN | TPM\_RIGHTBUTTON,
455. cursor.x, cursor.y, 0, hWnd, NULL);
456. DestroyMenu(hmenuPopup);
457. break;
458. }
459. }
460. break;
461. case WM\_SIZE:
462. pthread\_mutex\_lock(&mutex);
463. RefreshWindow(hWnd);
464. pthread\_mutex\_unlock(&mutex);
465. break;
466. case WM\_DESTROY:
467. PostQuitMessage(0);
468. break;
469. }
470. return DefWindowProc(hWnd, message, wParam, lParam);
471. }
472. //
473. // ФУНКЦИЯ: void RefreshWindow(HWND)
474. //
475. // НАЗНАЧЕНИЕ: обновляет элементы окна клиента
476. //
477. void RefreshWindow(HWND hWnd) {
478. // 1. Устанавливаем количество записей ListView
479. ListView\_SetItemCount( hListView, client->ListSize() );
480. // 2. Изменяем размеры ListView
481. RECT rc;
482. GetClientRect(hWnd, &rc);
483. MoveWindow(hListView,
484. rc.left,
485. rc.top,
486. rc.right - rc.left,
487. rc.bottom - rc.top,
488. true);
489. for (int i=0; i<IDS\_COL\_num; i++)
490. ListView\_SetColumnWidth(hListView, i, LVSCW\_AUTOSIZE\_USEHEADER);
491. }
492. //
493. // ФУНКЦИЯ: void UpdateClientMenu(HMENU hMenu, BYTE state)
494. //
495. // НАЗНАЧЕНИЕ: обновляет доступность элементов меню
496. //
497. void UpdateClientMenu(HMENU hMenu, BYTE state) {
498. const UINT mf\_state[] = { MF\_DISABLED, MF\_ENABLED };
499. BYTE st;
500. st = ( state & 0x40 )? 0: state & 7;
501. EnableMenuItem( hMenu, IDM\_START,
502. mf\_state[ (int)(st == SERVICE\_STOPPED) ] );
503. EnableMenuItem( hMenu, IDM\_PAUSE,
504. mf\_state[ (int)(st == SERVICE\_RUNNING) ] );
505. EnableMenuItem( hMenu, IDM\_RESUME,
506. mf\_state[ (int)(st == SERVICE\_PAUSED) ] );
507. EnableMenuItem( hMenu, IDM\_STOP,
508. mf\_state[ (int)( st == SERVICE\_RUNNING
509. | st == SERVICE\_PAUSED) ] );
510. st = ( state & 0x80 )? 7: (state >> 3) & 7;
511. EnableMenuItem( hMenu, IDM\_AUTO,
512. mf\_state[ (int)(st != SERVICE\_AUTO\_START
513. && st > 1) ] );
514. EnableMenuItem( hMenu, IDM\_DEMAND,
515. mf\_state[ (int)(st != SERVICE\_DEMAND\_START
516. && st > 1) ] );
517. EnableMenuItem( hMenu, IDM\_DISABLED,
518. mf\_state[ (int)(st != SERVICE\_DISABLED
519. && st > 1) ] );
520. }
521. //
522. // ФУНКЦИЯ: UINT LV\_Selection()
523. //
524. // НАЗНАЧЕНИЕ: получить номер выделенного элемента
525. //
526. UINT LV\_Selection() {
527. INT index;
528. index = ListView\_GetNextItem(hListView,
529. -1, LVNI\_ALL | LVNI\_SELECTED);
530. if (index == -1)
531. index = client->ListSize();
532. return (UINT) index;
533. }