### 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.

Ранее вариантов реализации было много, и 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

…(возможно, нужно продолжение

<https://lecturesnet.readthedocs.io/net/low-level/ipc/socket/int> )

… (нужны ли описания TCP/UDP???)

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

Windows NT определяет два вида системных служб. Один из них - это так называемые службы режима ядра (kernel-mode services), которые есть не что иное, как драйверы устройств. Другой вид - службы Win32 - это обычные Win32-процессы, использующие специальный набор функций для взаимодействия с менеджером системных служб.

Подробнее о службах Windows вы можете найти в [?] и [?].

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”.

Как бы то ни было, автор решил отказаться от реализации программы для ОС Linux, так как не уверен, что везде можно найти “Systemd”.

# 1.2 Постановка задачи проектирования

Клиент курсового проекта должен получать данные о службах. Автором было решено, что это будут:

* 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 TCP;
* Предоставление пользователю список служб сервера, их состояние и конфигурацию (тип запуска);
* Изменение состояния служб и типа запуска;
* Возможность сортировки полученного списка по любому из полей.

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 некоторых раздумий автор решил, что наилучшим вариантом будет объединение клиента и сервера в одно приложение.

Алгоритм работы представлен набором последовательных действий, осуществление которых обеспечивает корректную и стабильную работу программы. В зависимости от организации последовательности действий, алгоритмы подразделяются на линейные, разветвленные и циклические. По мере необходимости, алгоритмы могут одновременно объединять в себе циклическую и разветвленную форму, сохраняя участки линейной последовательности действий.

|  |
| --- |
| 3 алгоритма?  Сервер.Поток обработки:  Цикл(пока можно):  Если(клиент подключился):  Проверить заголовок  Распознать команду:  Список:  Подготовить и отправить список  Установка:  Изменить состояние/конфигурацию службы  Клиент.Получить список:  Запросить список  Проверить заголовок  Разобрать полученные данные  Клиент.Изменить состояние службы:  Отправить просьбу серверу |

Алгоритм функционирования разрабатываемой программы приведен на рисунке 2.1. Обобщенный алгоритм работы программы содержит следующие этапы:

1. Запуск программы
2. Выбор действия
   1. Создание новой базы данных
   2. Открытие файла базы данных
      1. Выбор файла
      2. Загрузка данных из файла
   3. Сохранение файла базы данных
      1. Выбор файла (для «Сохранить как…»)
      2. Сохранение данных в файл
   4. Добавление новой записи или редактирование существующей
      1. Ввод данных записи
      2. Добавление (изменение) записи в таблицу
   5. Удаление одной или нескольких записей
      1. Запрос на подтверждение
      2. Удаление соответствующей записи
   6. Поиск записей
      1. Выбор метода: по шаблону, в диапазоне
      2. Ввод необходимых параметров
      3. Переход к первой найденной записи
   7. Фильтрация записей
      1. Выбор исходных данных: база данных, отображаемый список
      2. Выбор метода: по шаблону, в диапазоне
      3. Ввод необходимых параметров
      4. Фильтрация исходных данных
   8. Сортировка записей
   9. Вывод информации о базе данных
3. Обновление главного окна



## **Рисунок 2.1** – общий алгоритм работы программы

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 2.1 и 2.2 соответственно.

|  |  |
| --- | --- |
| Рисунок 2.1 – интерфейс главного окна (режим сервера) | Рисунок 2.2 – 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 2.3.

## 

## Рисунок 2.3 – 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:

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.

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:

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 ? – содержимое поля State

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

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 2.2.



## **Рисунок 2.2** – компонентная схема системы

На рисунке 2.2. 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. tcpsock и sock – модули для работы с сокетами (TCP);

Таким образом, приложение позволяет осуществлять хранение информации и обеспечивать её целостность, проводить такие операции, как просмотр, редактирование, поиск, фильтрацию, добавление, изменение и удаление данных.

# 2.6 Разработка интерфейса

…

### 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 ?.

Engine???...

## 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 "tcpsock.h"
9. #define SocketType TcpSocket
10. //
11. // СТРУКТУРА: ListItem
12. //
13. // СОДЕРЖИМОЕ: элемент распакованного списка клиента
14. //
15. struct ListItem {
16. BYTE state;
17. char \*name, \*viewname;
18. };
19. //
20. // СТРУКТУРА: ServerParams
21. //
22. // СОДЕРЖИМОЕ: общие параметры для потоков сервера
23. //
24. struct ServerParams {
25. SocketType sock;
26. bool active;
27. pthread\_mutex\_t mutex;
28. };
29. //
30. // КЛАСС: Server
31. //
32. // НАЗНАЧЕНИЕ: управление основным потоком сервера
33. //
34. class Server {
35. private:
36. pthread\_t thread;
37. ServerParams param;
38. public:
39. bool Start(unsigned short port = 0);
40. bool IsWorking();
41. void Stop();
42. ~Server();
43. };
44. //
45. // КЛАСС: Client
46. //
47. // НАЗНАЧЕНИЕ: реализация клиента
48. //
49. class Client {
50. private:
51. ClientParams param;
52. public:
53. Client();
54. bool GetList(Address addr);
55. void SetSvc(Address addr, unsigned int idx, BYTE state);
56. unsigned int ListSize();
57. ListItem\* GetItem(unsigned int idx);
58. };
59. // Инициализировать работу с сокетами
60. bool InitializeSockets();
61. // Завершить работу с сокетами
62. void ShutdownSockets();
63. #endif

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

## Листинг 3.2 – 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

## Листинг 3.2 – продолжение

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

## Листинг 3.2 – продолжение

1. bool isserver;
2. public:
3. Socket();
4. ~Socket();
5. void Wait();
6. bool SetNonBlocking();
7. virtual bool Open() = 0;
8. bool Bind(Address addr);
9. void Close();
10. bool IsOpen();
11. bool IsServer();
12. virtual bool Accept() = 0;
13. virtual bool Connect(Address addr) = 0;
14. virtual bool IsConnected() = 0;
15. virtual void Disconnect() = 0;
16. virtual bool Send(void \*data, int size) = 0;
17. virtual int Receive(void \*data, int size) = 0;
18. };
19. // Приостановить программу на некоторое время
20. void wait( float seconds );
21. #endif

## Листинг 3.3 – заголовочный файл «tcpsock.h»

1. //
2. // ЗАГОЛОВОК: TCPSOCK.H
3. //
4. // ОПИСАНИЕ: класс TcpSocket
5. //
6. #ifndef tcpsock\_h\_
7. #define tcpsock\_h\_
8. #include "sock.h"
9. class TcpSocket : public Socket {
10. public:
11. bool Open();
12. bool Bind(Address addr);
13. bool Accept();
14. bool Connect(Address addr);
15. bool IsConnected();
16. bool Send(void \*data, int size);
17. int Receive(void \*data, int size);
18. void Disconnect();
19. };
20. #endif

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

## Листинг 3.4 – 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

…

### 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. Электронная библиотека книг Александра Фролова и Григория Фролова [сайт]. – Режим доступа: www.frolov-lib.ru
2. MSDN – сеть разработчиков Microsoft [сайт]. – Режим доступа: msdn.microsoft.com
3. winapi listview check item c++ - Stack Overflow [электронный ресурс] // Stack Overflow - Where Developers Learn, Share,& Build Careers [сайт]. – Режим доступа: stackoverflow.com/questions/15787078/winapi-listview-check-item-c
4. SQLite Home Page [сайт]. – Режим доступа: sqlite.org
5. Автоматизированные информационные системы [электронный ресур] // Кафедра "Информационные системы" МФПА [сайт]. – Режим доступа: <http://inftis.narod.ru/is/is-n3.htm>
6. «Управление системными службами Windows NT» // RSDN [сайт]. – Режим доступа: https://rsdn.org/article/baseserv/svcadmin-1.xml
7. «Управление системными службами Windows NT2» // RSDN [сайт]. – Режим доступа: https://rsdn.org/article/baseserv/svcadmin-2.xml

### ПРИЛОЖЕНИЕ А

# Общий алгоритм работы системы



### ПРИЛОЖЕНИЕ К

# Диаграмма классов и структур системы

