МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»
(БГТУ им. В.Г.Шухова)

Курсовая работа

дисциплина «Сети ЭВМ и телекомуникации»

по теме «Разработка многопоточного сканера портов с графическим интерфейсом.»

Выполнил: студент группы ВТ-31 Макаров Д.С.

Проверил: Федотов Е.А.

Белгород 2020

Содержание

Введение

Постановка задачи: Разработать многопоточный сканер портов с графическим интерфейсом.

Сканер портов - программное средство, используемое для обнаружение открытых портов хостов сети. В зависимоти от реализаций может использовтся как для проверки всех портов 1 хоста, так и для проверки конкретных портов во всей сети.

В данной работе будут освещенно сканирование *TCP*/*UDP* портов и реализован многопоточный сканер портов *TCP*.

Теоретические сведения

2.1 Основные термины

2.1.1 Порт

Порт - целое неотрицательное 16 битное число, используемое в заголовках транспортного уровня по модели OSI, для многоканальной передачи данных различного программного обеспечения в пределах 1 хоста. [**?**] Количество портов ограниченно 16 битами (**0-65535**), и все порты разделены на 3 диапазона:

- общеизвестные "*well-known*" порты (**0-1023**).
- зарегистрированные или пользовательские порты (**1024-49151**).
- динамические порты (**49152-65535**).

2.1.2 Протокол TCP

TCP - протокол передачи данных, транспортного уровня по модели OSI. TCP предоставляет поток данных с установкой соединения до передачи данных, при потере пакета осуществляется повторный запрос потерянного пакета, дубликаты пакетов удаляются. Тем самым гарантируется целостность передаваемых данных. [**?**]

Структура заголовка

Бит	0 — 3	4 — 9	10 — 15	16 — 31
0	Порт источника, **Source Port**			Порт назначения, **Destination Port**
32	Порядковый номер, **Sequence Number (SN)**			
64	Номер подтверждения, **Acknowledgment Number (ACK SN)**			
96	Длина заголовка, (**Data offset**)	Зарезервировано	Флаги	Размер Окна, **Window size**
128	Контрольная сумма, **Checksum**			Указатель важности, **Urgent Point**
160	Опции (необязательное, но используется практически всегда)			
160/192+	Данные			

Рис. 1: Заголовок сегмента TCP

Описание содержимого TCP заголовка:

- **Порт источника** - номер порта из которого был отправлен пакет, ответ формируется по порту источника.
- **Порт назначение** - номер порта на который был отправлен пакет.
- **Порядковый номер** (**SYN**) - число полезные переданных данных в байтах. При установленном флаге SYN поле будет содержать начальный порядковый номер (ISN) - случайно сгенерированное число, а первый байт полезных данных в сессии будет иметь номер ISN+1.
- **Номер подтверждения** (**ACK SN**) - при установленном флаге ACK, поле содержит номер октета, который отправитель хочет получить. Так же это значит что октеты от ISN+1 до ACK-1 были успешно получены.
- **Длина заголовка** - смещение полезных данных относительно начала пакета.
- Поле **зарезервировано** и **флаги** - 12 бит выделенные под различные флаги (ACK,SYN и др.), а так же зарезервированное пространство для будущих флагов.
- **Размер окна** - количество байт полезны данных после передачи которых отправитель будет ожидать подтверждение получения.
- **Контрольная сумма** - сумма всех 16 битных слов заголовка и данных. Если сегмент не кратен 16 битам то он дополняется нулями. Само поле в момент расчета суммы принимается равным нулю.
- **Указатель важности** - используется для передачи внеполосных данных.

2.1.3 Протокол UDP

UDP - протокол транспортного для работы с датаграммами, расширяет IP добавлением в заголовок полей: порт отправителя, порт получателя, длина датаграммы и контрольной суммы. [?]

Биты	0 - 15	16 - 31
0-31	Порт отправителя (Source port)	Порт получателя (Destination port)
32-63	Длина датаграммы (Length)	Контрольная сумма (Checksum)
64-...	Данные (Data)	

Рис. 2: Заголовок сегмента UDP

Не требует установки соединения для передачи данных и тем самым не гарантирует доставку пакетов и правильных их порядок. Используется в ситуациях когда нужно принимать множество пакетов от разных пользователей или когда потеря нескольких пакетов не критична.

2.1.4 Сканирование портов (сканер портов)

Сканирование портов - сканирование хоста или сети на наличие открытых портов. ПО для сканирование портов называется **сканером портов**. Так как программное обеспечение работающее по тому или иному порту может иметь уязвимости, такой открытый порт может являться серьезной угрозой информационной безопасности. В связи с этим сканерами портов пользуются или системные администраторы для предотвращения атак или злоумышленники для поиска точек входа в систему. [?] Результаты сканирования сети классифицируются следующим образом:

- открытый порт - сканером получен ответ, хост принимает соединения на данный порт.
- закрытый порт - сканером получен ответ, хост не принимает соединения на данный порт.
- заблокированный порт - сканер не получил ответ.

2.2 Типы сканирования

Перед сканированием любого типа обычно проводится проверка на наличие указанного хоста в сети. При помощи протокола ICMP отправляются echo сообщения на все сканируемые адреса, но отсутствие ответа на echo запрос не всегда означает отсутствие хоста в сети, так как системные администраторы зачастую запрещают работу ICMP в целях безопасности.

Существуют следующие алгоритмы сканирования портов:

2.2.1 Сканирование TCP портов

- **SYN сканирование** - самый распространенный тип сканирования. Сканер портов генерирует IP пакеты напрямую без использования сетевых функций ОС предназначенных для установки TCP соединения. Это все необходимо для того чтобы напрямую управлять содержимым заголовка TCP и позволяет не создавать полностью открытое соединение. Принцип работы SYN сканирования таков.

  1. Сканер создает пакет с установленным флагом SYN и отправляет его на указанный адрес (диапазон адресов).
  2. Если порт на целевом хосте открыт то в ответ сканеру придет пакет SYN-ACK и это означает что порт открыт. Если хост не отвечает значит SYN сканирование скорее всего заблокировано на уровне правил межсетевого экрана.
  3. Сканер отвечает пакетом RST и закрывает соединение до завершение его установки.

  Этот способ сканирования позволяет одновременно сканировать большое количество адресов и портов не создавая большой нагрузки на хост и сеть созданием и закрытием множества TCP соединений. Но для работы такого сканера потребуются повышенные привилегии и стороннее ПО для генерации IP пакетов в обход TCP стека операционной системы.

- **TCP сканирование** - самый простой способ сканирования портов. Используя сетевые функции операционной системы осуществляется попытка создать TCP соединение с хостом. При условии что порт открыт соединение будет установлено, иначе порт закрыт. Такой тип сканирования не требует специальный драйверов сетевых устройств и повышенных привилегий для сканирования сети, но сильно нагружает сканируемый хост.

- **ACK сканирование** - этот тип сканирования используется для проверки наличия межсетевого экрана и определения сложности его правил фильтрации. На хост отправляются пакеты с установленным флагом ACK, который устанавливается только при установленном соединении. Если в межсетевом экране используются простые правила фильтрации то экран пропустит этот пакет, более сложные правила учитывают и блокируют такой тип сканирования.

- **FIN сканирование** - данный тип сканирования использует особенность спецификации TCP (RFC 793), в которой описано что на пакет FIN отправленный на закрытый порт, сервер должен ответить пакетом с флагом RST, если порт открыт то сервер игнорирует такой пакет. Такое сканирование применяется если сервер умеет распознавать другие типы сканирования, но

не все разработчики ПО придерживаются спецификаций RFC, поэтому FIN сканирование может не дать результатов.

2.2.2 Сканирование UDP портов

В протоколе UDP отсутствует понятие соединение, поэтому мы не можем определить силами протокола был получен отправленный пакет или нет. Сканирование UDP порта имеет ряд особенностей. При отправке UDP пакета на закрытый порт при включенном протоколе ICMP, сканер получит ответ "порт закрыт", если же системный администратор отключил ICMP то компьютерам извне будет казаться что все порты открыты.

Для того чтобы обойти отключение ICMP или межсетевой экран можно формировать UDP пакет специфичный для ПО работающего с данным портом. Таким образом при открытом порте мы получим ответ уровня приложения. Но подготавливать тестовые пакеты для всего сетевого ПО, работающего по UDP - невозможная задача. Поэтому может использоваться комбинированное сканирование - сначала сканируются все порты UDP отправкой пустого пакета, а далее используются специализированные пакеты для выбранных портов, если они поддерживаются ПО для сканирования.

2.3 Сканирования портов в законодательстве РФ и правилах обслуживания интернет-провайдеров

Уголовный правонарушения связанные с ЭВМ и сетями ЭВМ описаны в 28 главе Уголовного кодекса РФ под названием "*Преступления в сфере компьютерной информации*". [?] Эта глава содержит 4 статьи:

- **Статья 272. Неправомерный доступ к компьютерной информации**
- **Статья 273. Создание, использование и распространение вредоносных компьютерных программ**
- **Статья 274. Нарушение правил эксплуатации средств хранения, обработки или передачи компьютерной информации и информационно-телекоммуникационных сетей**
- **Статья 274^1. Неправомерное воздействие на критическую информационную инфраструктуру Российской Федерации**

Ни в одной из вышеперечисленных статей не описано наказание за сканирование сегментов сети, но информацию полученную при сканировании сети такую как: ОС и ее версию, открытые порты для приложений прикладного уровня и их версии, можно использовать для совершения преступлений в информационной сфере.

Так же провайдеры интернет услуг зачастую запрещают использовать сканеры для сканирования портов в сети Интернет.

Разработка программы

3.1 Выбор инструментов разработки

В рамках данной курсовой работы был реализован TCP сканер портов, SYN сканирование не было реализовано так как ОС Windows не разрешает отправлять пакеты используя “сырые” сокеты (“сырые” сокеты позволяют напрямую изменять и просматривать содержимое IP пакета). Для реализации SYN сканирования в ОС Windows потребовались бы привилегии администратора в системе, а так же специальный сетевой драйвер позволяющий изменять IP пакеты напрямую.

Сканер был реализован на языке C++ с вставками на языке C, а так же использовании кроссплатформенного фреймворка Qt для создания графического интерфейса и многопоточности.

3.2 Проектирование графического интерфейса

Графический интерфейс, был спроектирован используя возможности библиотеки элементов **Qt**. Эта библиотека позволяет создавать программы с графическим интерфейсом, которые имеют одинаковое поведение в разных ОС. А механизм слотов и сигналов позволяет быстро и эффективно писать приложения с графическим интерфейсом. [?] Интерфейс сканера содержит:

- адресную строку
- шкалу прогресса
- список событий сканера

Корректность введенного адреса проверяется при помощи регулярного выражения.

3.3 Реализация многопоточного сканирования

Многопоточное сканирование использует, объединение потоков в пул потоков для эффективного использование ресурсов компьютера. **QThreadPool** [?]

- реализация пула потоков в фреймворке Qt, при инициализации пула выделяется количество потоков равное количеству логических процессоров. При старте сканирования каждый поток сканирует 1 порт из диапазона общеизвестных портов, при завершении работы поток не закрывается, а используется для сканирование следующего необходимого порта.

3.4 Описание классов используемых в приложении

Для отображения графического интерфейса, и обработки взаимодействий с пользованием используется класс **SimplePortScan**.

В конструкторе класса создаются и настраиваются все элементы интерфейса и их валидация.

```
QRegExp *re = new
↪  QRegExp("^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0
QRegExpValidator *validator = new QRegExpValidator(*re);
ui->setupUi(this);
ui->progressBar->hide();
ui->progressBar->setMinimum(0);
ui->progressBar->setMaximum(PORT_EDGE);
ui->modeButton1->hide();
ui->modeButton2->hide();
ui->addressLine->setValidator(validator);
```

и выбирается кол-во потоков в пуле.

```
this->threads.setMaxThreadCount(400);
```

В слоте кнопки старта сканирования `on_pushButton_clicked()` запускается пул потоков и в цикле добавляются задачи **SimplePortScan** на сканирования конкретного порта (от 0 до PORT_EDGE). Так же запускается шкала процесса сканирования и она отображается на экран.

Каждая задача **SimplePortScan** вызывает сигнал `exit_code(int,int)` указывающий на завершение сканирование порта. он содержит номер сканируемого порта, и его состояние.

Обработчик этих сигналов `on_result_thread(int,int)` записывает результаты в **QMap** [?]

- словарь основанный на хеш таблице, где ключ - это номер порта, а значение - состояние полученное при сканировании. Для того чтобы не возникло состояние гонки потоков в качестве объекта синхронизации используется класс **QMutex**

[?] реализующий обьект синхронизации мьютекс. Он блокируется потоком во время инкрементирования счетчика полосы прогресса и записи в **QMap** [?] .

Архитектура программы спроектирована так что, можно легко и быстро реализовывать различные способы сканирования.

Используя наследования от класса **QRunnable** [?] , мы можем создать новый тип задачи в котором будет реализовано то или иное сканирование.

В данной программе в связи с особенностями сетевой архитектуры Windows, реализовано простейшее TCP сканирование в классе **SimplePortScan**

В качестве конструктора используется конструктор родительского класса **QRunnable** [?] , а так же переопределена виртуальная функция **run()** запускающая поток с задачей.

Внутри этого потока создается TCP сокет и происходит попытка открыть TCP соединение, если спустя 500 мс соединение не было установлено, то порт считается закрытым и задача завершает с кодом 1, иначе при установленном соединении сокет закрывает соединение и задача завершается с кодом 0.

Руководство пользователя

При запуске сканера выводится предупреждение о запрете сканирования в глобальных сетях.

В поле адрес необходимо ввести сканируемый адрес сети в формате IPv4/

Далее необходимо нажать кнопку “Сканировать” для начала процесса сканирования.

Программа сканирует порты из списка общеизвестных портов (**0-1023**) и зарегистрированных портов (**1024-49151**).

В процессе работы сканера появится шкала прогресса и в логе отобразится сообщение о старте сканирования.

После завершения работы в логе сканера отобразится список открытых портов вышеуказанного адреса.

Тестирование

5.1 Исходные данные

Для проверки работоспособности программы было совершено сканирование 3 хостов в локальной сети.

- Маршрутизатор.
- ПК, на котором был запущен сканер.
- Смартфон, подключенный к маршрутизатору.

5.2 Тест 1. Маршрутизатор

Маршрутизатор с ОС Open-Wrt с вручную открытым портом для SSH (20).

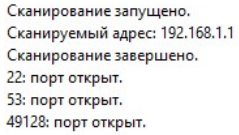

Рис. 3: Результат работы программы

Программа показала что роутере открыты порты:

- 22 - порт SSH
- 53 - DNS порт
- 1009 - неиспользуемый порт

5.3 Тест 2. ПК

Компьютер работает под управление ОС Windows.

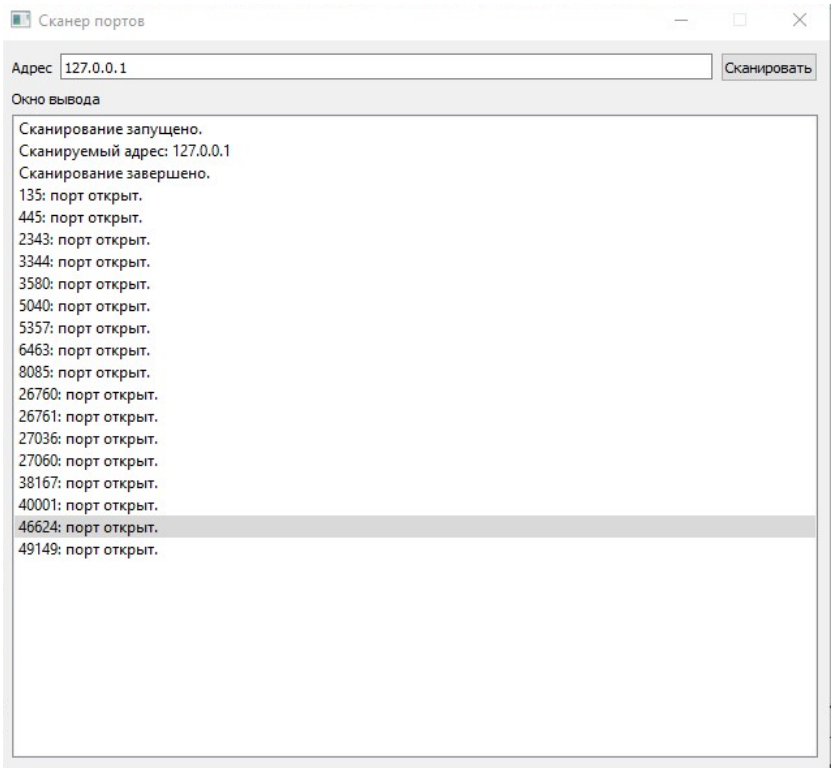

Рис. 4: Результат работы программы

Программа показала что на компьютере открыты порты:

- 135 - порт EPMAP, Microsoft RPC, Locator service
- 445 - порт MICROSOFT-DS
- 2343, 3344, 3580, 5040, 5357, 6463, 8085, 26760, 26761, 27036, 27060, 38167, 40001, 46624, 49139 - неиспользуемые открытые порты.

5.4 Тест 3. Смартфон

Смартфон под управлением ОС Android с отключенным ABD на порту 5555.

Рис. 5: Результат работы программы

Программа показала что на смартфоне открыты порты:

- 1009 - неиспользуемый порт

Через 20 секунд после работы сканера, маршрутизатор отключил ПК с сканером от локальной сети. Сработало правило сетевого экрана.

Заключение

В ходе выполнения данной курсовой работы был реализован многопоточный сканер портов с графическим интерфейсом, а так же проверена его работоспособность в локальной сети. В процессе написания работы были изучены различные типы сканирования портов, законность выполнения сканирования портов, основные диапазоны портов. В ходе написания программы были изучены принципы построения графических интерфейсов в библиотеке Qt, а так же архитектурные решения предоставляемые этой библиотекой для создания многопоточных приложений. При написании сканера оптимизации для многопоточной работы было применено архитектурный шаблон “пул” потоков, из-за технических особенностей ОС Windows был реализован только 1 тип сканирования портов - TCP, для реализации других типов сканирования необходимо использовать специализированные драйвера сетевых устройств, позволяющие работать на уровне IP пакетов.

Список литературы

[1] Интернет-ресурс *Википедия. Протокол TCP.* URL: https://ru.wikipedia.org/wiki/TCP

[2] Интернет-ресурс *Википедия. Протокол UDP.* URL: https://ru.wikipedia.org/wiki/UDP

[3] Интернет-ресурс *Википедия. Порт.* URL: https://ru.wikipedia.org/wiki/Порт_(

[4] Интернет-ресурс *Википедия. Сканер портов.* URL: https://ru.wikipedia.org/wiki/Сканирование_портов

[5] Интернет-ресурс *Викитека. Уголовный кодекс Российской Федерации/Глава 28.* URL: https://ru.wikisource.org/wiki/Уголовный_кодекс_Российской_Федерации/Гла

[6] Интернет-ресурс *Qt Core 5.14 Documentation.* URL: https://doc.qt.io/qt-5

[7] Интернет-ресурс *QRunnable Class | Qt Core 5.14 Documentation.* URL: https://doc.qt.io/qt-5/qrunnable.html

[8] Интернет-ресурс *QMap Class | Qt Core 5.14 Documentation.* URL: https://doc.qt.io/qt-5/qmap.html

[9] Интернет-ресурс *QMutex Class | Qt Core 5.14 Documentation.* URL: https://doc.qt.io/qt-5/qmutex.html

[10] Интернет-ресурс *QThreadPool Class | Qt Core 5.14 Documentation.* URL: https://doc.qt.io/qt-5/qthreadpool.html

Приложение

Содержимое файла course-work-v2.pro

```
QT       += core gui network

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before
↪  Qt 6.0.0

SOURCES += \
    crossSocket.cpp \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    crossSocket.h \
    cross_sockets.h \
    mainwindow.h \
    scan_functions.h

FORMS += \
    mainwindow.ui

TRANSLATIONS += \
    course-work-v2_ru_RU.ts

win32:LIBS += -lws2_32

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

Содержимое файла crossSocket.cpp

Содержимое файла crossSocket.h

```
#ifndef CROSSSOCKET_H
#define CROSSSOCKET_H

#endif // CROSSSOCKET_H
```

Содержимое файла crosssockets.h

```
#ifndef SCAN_FUNCTIONS_H
#define SCAN_FUNCTIONS_H
//stdout messages
//#define LOGS_ENABLE

//OS independed includes
#include <stdio.h>
//OS depended includes
#ifdef _WIN32
    #define WIN32_LEAN_AND_MEAN
    #include <windows.h>
    #include <winsock2.h>
#else
    #include <unistd.h>
    #include <socket.h>
    #include <errno.h>
    #pragma message "UNIX sockets API isn\'t fully implemented ...\n"
#endif

int crs_init();
int crs_get_last_error();
int crs_socket(int domain,int type,int protocol);
int crs_cleanup();
int crs_close_socket(int socket);
int crs_in_connect(int sock,const struct sockaddr_in *adr,int adr_len);

int last_error_code = 0;
//sockets API functions wrappers
int crs_init(){
    #ifdef _WIN32
        #ifdef LOGS_ENABLE
            printf("Winsock init: ");
        #endif
        WSADATA wsaData = {0};
        WORD versionRequested = MAKEWORD(2,2);
        int startupResult = WSAStartup(versionRequested,&wsaData);
        if(startupResult != 0){
            #ifdef LOGS_ENABLE
                printf("Winsock startup failed with error: %d\n",startupResult);
            #endif
            exit(1);
        };
    #else
        #ifdef LOGS_ENABLE
            printf("UNIX sockets init: ");
        #endif
    #endif
    #ifdef LOGS_ENABLE
        printf("OK!\n");
    #endif
    last_error_code = 0;
    return 0;
};

int crs_get_last_error(){
    int last_error = 0;
    #ifdef _WIN32
        last_error = WSAGetLastError();
        #ifdef LOGS_ENABLE
            printf("Program failed with error: %d (Winsock Error)\n",last_error);
```

```
        #endif
    #else
        last_error = errno;
        #ifdef LOGS_ENABLE
            printf("Program failed with error: %d (Unix ErrNo)\n",last_error);
        #endif
    #endif
    last_error_code = last_error;
    return last_error;
};

int crs_socket(int domain,int type,int protocol){
    int result_socket = (int)socket(domain,type,protocol);
    #ifdef _WIN32
        if(result_socket == INVALID_SOCKET){
    #else
        if(result_socket == -1){
    #endif
            crs_get_last_error();
            return(-1);
        }
    return(result_socket);
};



int crs_cleanup(){
    #ifdef LOGS_ENABLE
        printf("Cleanup: ");
    #endif
    #ifdef _WIN32
        WSACleanup();
    #else
    #endif
    #ifdef LOGS_ENABLE
        printf("OK!\n");
    #endif
    return 0;
};

int crs_close_socket(int socket){
    int return_value = 0;
    #ifdef _WIN32
        return_value = closesocket((SOCKET)socket);
    #else
        return_value = close(socket);
    #endif
    if(return_value == -1){
        crs_get_last_error();
        return(2);
    }
    return(0);
};

int crs_in_connect(int sock,const struct sockaddr_in *adr,int adr_len){
    int return_value = 0;
    #ifdef _WIN32
        return_value = connect((SOCKET)sock,(struct sockaddr*)adr,adr_len);
    #else
        return_value = connect(sock,(struct sockaddr*)adr,adr_len);
```

```
    #endif
    if(return_value == -1){
        crs_get_last_error();
        return(3);
    }
    #ifdef LOGS_ENABLE
        printf("OK!\n");
    #endif
    return 0;
};
#endif
```

Содержимое файла main.cpp

```
#include "mainwindow.h"

#include <QApplication>
#include <QMessageBox>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.setWindowTitle("Сканер портов");
    QMessageBox msgBox;
    msgBox.setIcon(QMessageBox::Critical);
    msgBox.setWindowTitle("ВНИМАНИЕ");
    msgBox.setText("Программа создана исключительно в образовательных целях.\nБольшинство
    ↪  интернет-провайдеров прямо запрещают сканирование портов. Используйте программу
    ↪  исключительно в локальных сетях.");
    msgBox.exec();
    w.show();
    return a.exec();
}
```

Содержимое файла mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "cross_sockets.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    QRegExp *re = new
    ↪  QRegExp("^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0
    QRegExpValidator *validator = new QRegExpValidator(*re);
    ui->setupUi(this);
    ui->progressBar->hide();
    ui->progressBar->setMinimum(0);
    ui->progressBar->setMaximum(PORT_EDGE);
    ui->modeButton1->hide();
    ui->modeButton2->hide();
    ui->addressLine->setValidator(validator);
    this->steps_completed = 0;
    this->threads.setMaxThreadCount(400);
    connect(this,SIGNAL(last_task()),SLOT(last_task_completed()));

}
```

```
MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_pushButton_clicked()
{
   if(scan_running){
        this->scan_running= false;
       this->ui->pushButton->setText("Сканировать");
       this->ui->modeButton1->setEnabled(true);
       this->ui->modeButton2->setEnabled(true);
       this->ui->addressLine->setEnabled(true);
       this->ui->progressBar->hide();
       this->threads.clear();
       qDebug()<<"Сканирование остановлено\n";
   }else{
       this->scan_running = true;

       this->ui->pushButton->setText("Отменить");
       this->ui->modeButton1->setEnabled(false);
       this->ui->modeButton2->setEnabled(false);
       this->ui->addressLine->setEnabled(false);

       this->ui->progressBar->show();
       this->ui->progressBar->reset();

       this->steps_completed = 0;
       this->ports.clear();
       this->ui->listWidget->addItem("Сканирование запущено.");
       this->ui->listWidget->addItem(QString("Сканируемый адрес:
       ↪  ")+this->ui->addressLine->text());

       qDebug()<<"Сканирование запущено\n";
       qDebug()<<"Режим: "<<(int)this->scan_mode<<"\n";

       for(int i = 0;i<PORT_EDGE;i++){
           if(scan_running){
            SimplePortScan *scan_thread = new SimplePortScan(i,ui->addressLine->text());
            connect(scan_thread,SIGNAL(exit_code(int,int)),SLOT(on_result_thread(int,int)));
            this->threads.start(scan_thread);
           }
       };

   }
}


void MainWindow::on_modeButton1_clicked()
{
    this->scan_mode=0;
}

void MainWindow::on_modeButton2_clicked()
{
    this->scan_mode=1;
}

void MainWindow::on_result_thread(int result_code,int port)
```

```
{
    this->step_mutex.lock();
    this->steps_completed++;
    this->ui->progressBar->setValue(this->steps_completed);
    if(steps_completed==this->ui->progressBar->maximum()){
        emit last_task();
    }
    this->step_mutex.unlock();

    this->map_mutex.lock();
    this->ports.insert(port,(int)result_code);
    this->map_mutex.unlock();

};

void MainWindow::last_task_completed(){
    this->ui->listWidget->addItem("Сканирование завершено.");
    for(int i = 0;i<PORT_EDGE;i++){
        if(this->ports[i]==0) {
            this->ui->listWidget->addItem(QString::number(i)+QString(": порт открыт."));
        }
    }
    this->scan_running= false;
    this->ui->pushButton->setText("Сканировать");
    this->ui->modeButton1->setEnabled(true);
    this->ui->modeButton2->setEnabled(true);
    this->ui->addressLine->setEnabled(true);
    this->ui->progressBar->hide();
    this->ui->listWidget->addItem("");
};
```

Содержимое файла mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QMap>
#include <QDebug>
#include <QThreadPool>
#include <QMutex>
#include <QRegExp>
#include <QRegExpValidator>

#include "scan_functions.h"

#define PORT_EDGE 49151
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    //Список портов
    QMap<int,char> ports;
```

```
    bool scan_running = false;
    //0 - сканирование с соединением
    //1 - сканирование с без соединения
    char scan_mode = 0;
    int steps_completed = 0;
    QThreadPool threads;
    QMutex step_mutex,map_mutex;
private slots:
    void on_pushButton_clicked();

    void on_modeButton1_clicked();

    void on_modeButton2_clicked();

    void on_result_thread(int result_code,int port);

    void last_task_completed();
private:
    Ui::MainWindow *ui;
signals:
    void last_task();
};
#endif // MAINWINDOW_H
```

Содержимое файла scanfunctions.h

```
#ifndef SCAN_FUNCTIONS_H
#define SCAN_FUNCTIONS_H

#include <QObject>
#include <QTcpSocket>
#include <QHostAddress>
#include <QRunnable>
#include <QTimer>
#include <QEventLoop>
#include <QDebug>
#include "cross_sockets.h"

//1 - закрыт
//-1 - ошибка
//0 - открыт

class SimplePortScan : public QObject,public QRunnable{
    Q_OBJECT
    signals:
        void exit_code(int return_code,int port);
    public:
        int code = 0;
        explicit SimplePortScan(int port, QString address, QObject *parent =
        ↪  0):QObject(parent),port(port),address(address){};
        void run(){
            //переписать под событийный таймер
            QTcpSocket socket;
            QTimer timer;
            timer.setSingleShot(true);
            QEventLoop loop;
            connect(&timer,&QTimer::timeout,&loop,&QEventLoop::quit);
            connect(&socket,&QAbstractSocket::connected,&loop,&QEventLoop::quit);
            socket.connectToHost(address,(qint16)port);
            timer.start(1500);
```

```
            loop.exec();
            if(timer.isActive()){
                    qDebug()<<port<<": open";
                    code = 0;
                    emit exit_code(code,port);
            }else{
                    qDebug()<<port<<": timeout";
                    code = 1;
                    emit exit_code(code,port);
            }

        };
    protected:
        int port;
        QString address;
};

class NonConnectPortScan : public QObject,public QRunnable{
    Q_OBJECT
    signals:
        void exit_code(int return_code,int port);
    public:
        explicit NonConnectPortScan(int port, QString address, QObject *parent =
        ↪  0):QObject(parent),port(port),address(address){};
        void run(){
            emit exit_code(0,port);
        };
    protected:
        int port;
        QString address;
};
#endif // SCAN_FUNCTIONS_H
```