Лабораторная работа 4
Статические и динамические библиотеки

Задание 1

Напишем статические библиотеки с одной функцией для сложения двух целых чисел для 64- и 32- битных архитектур. Для начала напишем код на C (См. Рис. 1), в котором объявим две переменных типа **int** и функцию **sum(int x, int y)**, возвращающую значение типа **int**. Внутри функции **main()** просто выведем возвращаемое значение функции **sum** на экран.

```c
#include <stdio.h>

int sum(int x, int y);

int x = 5, y = 3;

int main()
{
    printf("Sum is %d", sum(x, y));
    return 0;
}
```

Рисунок 1: program1.c

```c
int sum(int x, int y)
{
    return x + y;
}
```

Рисунок 2: sum.c

Напишем реализацию функции **sum** (См. Рис. 2). Начнём с 64-х битной архитектуры. Создадим из файла **sum.c** объектный файл **sum_64_bit.o**, используя команду «**gcc -c sum.c -o sum_64_bit.o**». Сгенерируем файл **libSUM_64_bit.a** из полученного объектного файла с помощью команды «**ar rc libSUM_64_bit.a sum_64_bit.o**». Теперь создадим требуемую статическую библиотеку, используя «**ranlib libSUM_64_bit.a**». Подключим её при компиляции файла **program1.c** с помощью команды **«gcc program1.c libSUM_64_bit.a -o program1_64_bit**». Если мы сейчас запустим полученный исполняемый файл **program1_64_bit**, то увидим в выводе «**Sum is 8**», что является верным ответом. На Рис. 3 приведён перечень используемых команд.

```
gcc -c sum.c -o sum_64_bit.o
ar rc libSUM_64_bit.a sum_64_bit.o
ranlib libSUM_64_bit.a
gcc program1.c libSUM_64_bit.a -o program1_64_bit
./program1_64_bit
```

*Рисунок 3: Создание **libSUM_64_bit.a***

Поговорим немного подробнее о вышеупомянутых командах. Программа **ar** (сокр. от **archiver** — архиватор) используется для создания, модификации и просмотра объектных файлов в статических библиотеках. В результате её использования мы получили файл **libSUM_64_bit.a**, в котором лежит копия объектного файла **sum_64_bit.o**. Если файл библиотеки уже существует, то архиватор будет анализировать содержимое архива: он добавит новые объектные файлы и заменит старые обновленными версиями. Опция **c** (от **create**) заставляет создавать библиотеку, если ее нет, а опция **r** (от **replace**) заменяет старые объектные файлы новыми версиями.

Архивный файл **libSUM.a** не является полноценной статической библиотекой объектных файлов. Для того чтобы это исправить, нам надо добавить к этому архиву индекс символов, т.е. список вложенных в библиотеку функций и переменных, чтобы линковка происходила быстрее. Делается это с помощью команды **ranlib**.

Программа **ranlib** добавит индекс к архиву и получится полноценная статическая библиотека объектных файлов. Стоит отметить, что на некоторых системах программа **ar** автоматически создает индекс, и использование **ranlib** не имеет никакого эффекта. Но тут надо быть осторожным при автоматической компиляции библиотеки с помощью файлов

makefile, если вы не будете использовать утилиту **ranlib**, то возможно на каких-то системах библиотеки будут создаваться не верно и потеряется независимость от платформы. Также стоит заметить, что компилятору нужны библиотеки лишь на этапе создания конечного файла, т.е. линковки. На этапе компиляции будет достаточно написанной нами заглушки для функции **sum**.

Теперь создадим и запустим статическую библиотеку **libSUM_32_bit.a** на 32-х битной архитектуре, используя флаг -m32. Не будем останавливаться на командах настолько же подробно, как раньше, так как их смысл остаётся неизменным. Приведём их полный перечень на Рис. 4.

```
gcc -c sum.c -o sum_32_bit.o -m32
ar rc libSUM_32_bit.a sum_32_bit.o
ranlib libSUM_32_bit.a
gcc program1.c libSUM_32_bit.a -o program1_32_bit -m32
./program1_32_bit
```

Рисунок 4: Создание ***libSUM_32_bit.a***

Задание 2

Теперь сгенерируем динамические библиотеки для 64- и 32- битных архитектур. Начнём с первой из них. Динамическая библиотека это уже не архивный файл, а настоящая загружаемая программа, поэтому созданием динамических библиотек занимается сам компилятор **gcc**. Как и ранее для начала создадим из файла **sum.c** объектный файл **sum_64_bit.o**, используя команду «**gcc -c sum.c -o sum_64_bit.o**». Для того, чтобы создать динамическую библиотеку надо использовать ключ **-shared**. Сама команда будет иметь следующий вид: «**gcc -shared -o libSUMdyn_64_bit.so sum_64_bit.o**». В результате мы получим динамическую библиотеку **libSUMdyn_64_bit.so**, которая имеет тот же функционал, что и её статический вариант, рассмотренный ранее. Теперь, для того чтобы компилировать результирующий файл с использованием динамической библиотеки, нам нужно собрать его следующей командой: «g**cc program1.c -L. libSUMdyn_64_bit.so -o program1_64_bit./**» (См. Рис. 5).

```
gcc -shared -o libSUMdyn_64_bit.so sum_64_bit.o
gcc program1.c -L. libSUMdyn_64_bit.so -o program1_64_bit
```

Рисунок 5: Создание динамической библиотеки и сборка исполняемого файла

Если мы сейчас попробуем запустить файл **program1_64_bit** с помощью команды «**./program1_64_bit**», то получим такую ошибку:

```
./program1_64_bit: error while loading shared libraries: libSUMdyn_64_bit.so: cannot open shared object file: No such file or directory
```

Рисунок 6: Вывод терминала

Это сообщение выдает динамический линковщик. Он просто не может найти файл нашей динамической библиотеки. Дело в том, что загрузчик ищет файлы динамических библиотек в известных ему стандартных директориях, а наша директория такой не является. Исправить эту проблему можно, используя различные команды.

Первый вариант. Это использование специальной переменной среды **LD_LIBRARY_PATH**, в которой перечисляются все каталоги содержащие пользовательские динамические библиотеки. Сначала нам нужно изменить её значение, написав следующее: «**LD_LIBRARY_PATH=.**». Теперь данная переменная указывает на текущую дерикторию, а значит линковщик сможет найти нашу динамическую библиотеку при запуске. Если мы хотим единоразово запустить нашу программу, изменив данную переменную, то можно написать так: «**LD_LIBRARY_PATH=. ./program1_64_bit**». Если же нет, то можно

экспортировать её значение с помощью «**export LD_LIBRARY_PATH**». И тогда мы можем запустить нашу программу как обычно. Естественно в обоих случаях результат будет ожидаемым: «**Sum is 8**». Команды для обоих случаев можно увидеть на Рис. 7 и 8.

```
LD_LIBRARY_PATH=.
export LD_LIBRARY_PATH
./program1_64_bit
```

Рисунок 7: Пример 1 запуска программы

```
LD_LIBRARY_PATH=. ./program1_64_bit
```

Рисунок 8: Пример 2 запуска программы

Теперь если мы обнулим значение данной переменной, написав «**LD_LIBRARY_PATH=""**», то программа снова перестанет работать.

Второй вариант. В отличие от первого, он, возможно более удобен, но также и более опасен. Так как его неосторожное и бездумное использование может закончится переустановкой операционной системы и утраченными данными. Сам способ заключается в том, что теперь мы не будем менять системные пути, по которым линковщик ищет библиотеки, а просто поместим нашу библиотеку в стандартную системную папку. Перед этим необходимо убедиться, что библиотеки с таким же именем, как и ваша, нет в этой папке. Посмотреть на её содержимое можно с помощью команды «**ls /usr/lib**». Теперь мы скопируем нашу библиотеку в эту папку, используя команду «**sudo cp libSUMdyn_64_bit.so /usr/lib**». Если мы сейчас запустим нашу программу то увидим, что она корректно выполнилась. После этого не лишним будет удалить нашу библиотеку из системной папки. Сделать это можно написав «**sudo rm /usr/lib/libSUMdyn_64_bit.so**». Теперь при запуске программы мы увидим уже знакомую нам ошибку (См. Рис. 9).

```
vladimir@vladimir-GF65-Thin-10UE:~/code/LAB4/ex2_dynamic_library$ sudo cp libSUMdyn_64_bit.so /usr/lib
vladimir@vladimir-GF65-Thin-10UE:~/code/LAB4/ex2_dynamic_library$ ./program1_64_bit
Sum is 8vladimir@vladimir-GF65-Thin-10UE:~/code/LAB4/ex2_dynamic_library$ sudo rm /usr/lib/libSUMdyn_64_bit.so
vladimir@vladimir-GF65-Thin-10UE:~/code/LAB4/ex2_dynamic_library$ ./program1_64_bit
./program1_64_bit: error while loading shared libraries: libSUMdyn_64_bit.so: cannot open shared object file: No such file or directory
```

Рисунок 9: Фрагмент терминала

Теперь создадим и запустим динамическую библиотеку **libSUMdyn_32_bit.so** на 32-х битной архитектуре, используя флаг -m32. Не будем останавливаться на командах настолько же подробно, как раньше, так как их смысл остаётся неизменным. Приведём их полный перечень на Рис. 10.

```
gcc -c sum.c -o sum_32_bit.o -m32
gcc -shared -o libSUMdyn_32_bit.so sum_32_bit.o -m32
gcc program1.c -L. libSUMdyn_32_bit.so -o program1_32_bit -m32
LD_LIBRARY_PATH=. ./program1_32_bit
```

Рисунок 10: Создание **libSUMdyn_32_bit.so**

Задание 3

Посмотрим как выглядят полученные файлы и сравним их друг с другом. Если мы попробуем сделать это, используя ,например, **vim**, то всё, что мы увидим, так это малопонятную кашу из символов. Это происходит потому, что vim пытается трактовать машинный код как текст, и ,очевидно, получается так себе. Для того, чтобы хотя бы немного приблизиться к пониманию внутренней структуры файла мы можем внутри **vim** перейти в режим HEX-редактирования. Делается это с помощью команды «**%!xxd**». Выход из этого режима — команда «**%!xxd -r**». Здесь мы получим уже что-то более осмысленное. Перед нами предстанут колонки из цифр — это шестнадцатеричный код. Левый столбец — номера команд, 8 центральных столбцов — код самого файла, правый столбец — попытки vim интерпретировать машинный код. Как можно видеть, из данного представления файла, трудно что то сказать о его структуре.

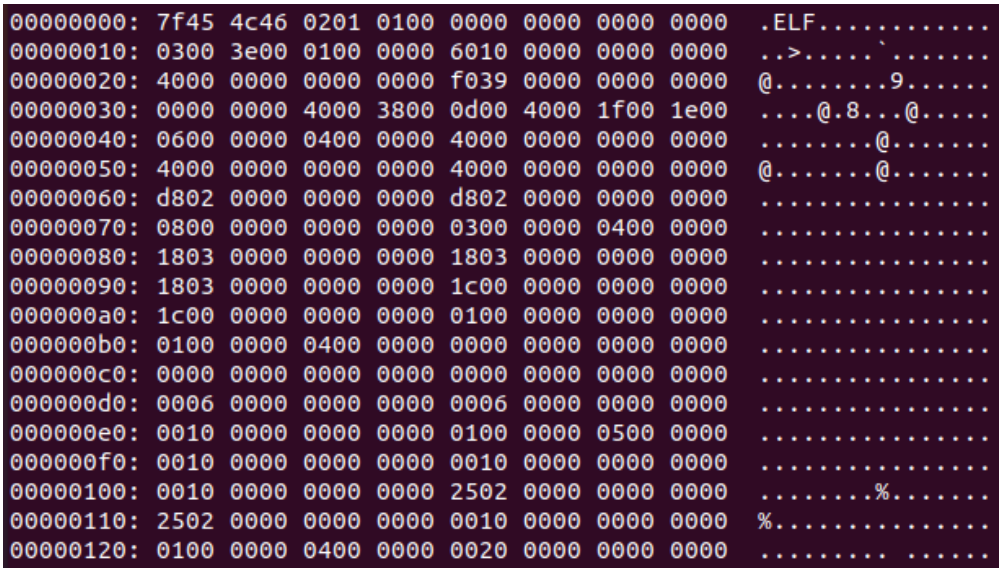

*Рисунок 11: Фрагмент машинного кода в **vim** в режиме HEX-редактирования*

Тогда используем команду «**objdump -d <name_of_file> > <name_of_file>**». Команда **objdump** покажет нам ассемблерный код, который она попыталась воспроизвести из машинных инструкций в исполняемом файле. Опция **-d** говорит нам о том, что дизассемблированы будут лишь те секции, которые ожидаемо должны содержать инструкции. Данная часть «…. > **<name_of_file>**» означает, что полученный код мы будем записывать в файл, а не выводить напрямую в терминал. Мы делаем так, потому что это более удобно, а иногда другого выхода, у нас просто может и не быть, так как терминал имеет конченый размер буфера и не сможет вместить действительно большой файл.

Попытаемся дизассемблировать файл **program1stat_64_bit**, используя «**objdump -d program1stat_64_bit > disas_stat_64_bit.txt**».

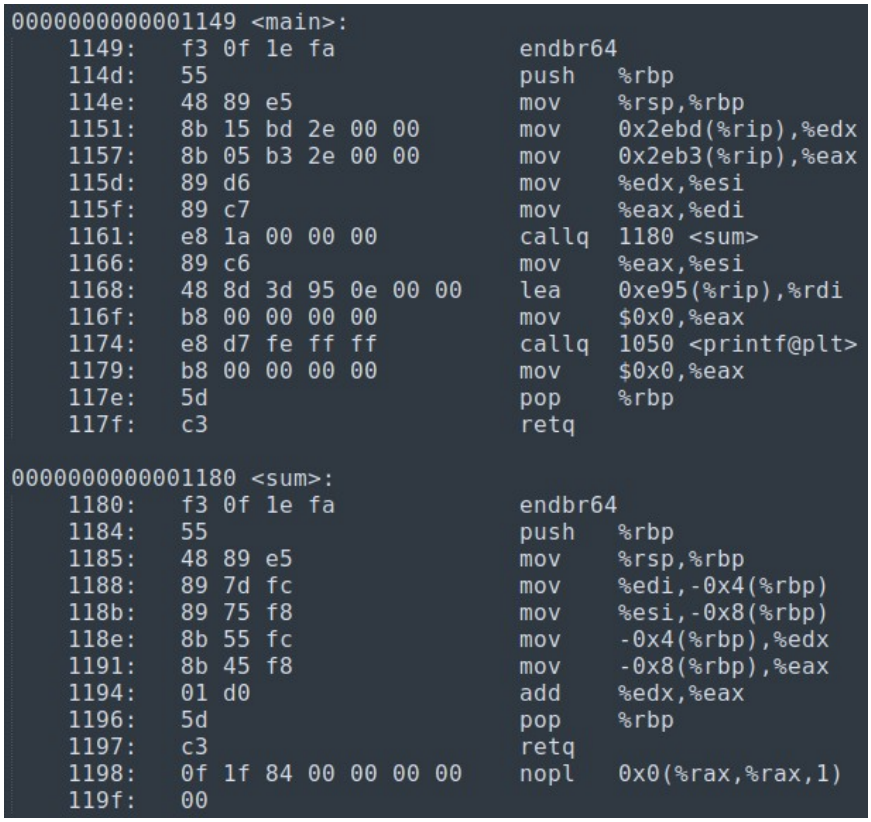

*Рисунок 12: Фрагмент файла **disas_stat_64_bit.txt***

```
Дизассемблирование раздела .plt.sec:

0000000000001050 <printf@plt>:
    1050:   f3 0f 1e fa             endbr64
    1054:   f2 ff 25 75 2f 00 00    bnd jmpq *0x2f75(%rip)
    105b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

Рисунок 13: Фрагмент файла ***disas_stat_64_bit.txt***

Теперь мы получим пригодный для анализа файл. Как можно видеть, функции **<main>** и **<sum>** расположены в исполняемом файле друг за другом. А также дизассемблированы они из одного и того же раздела **.text**. Когда в строке **1161** внутри функции **<main>** происходит вызов функции **<sum>**, то мы переходим на строку **1180** — на первую строчку функции **<sum>**. В ней написаны все необходимые инструкции и логика, а в конце — возврат в вызывающую функцию. Напротив, если мы посмотри на функцию **<printf@plt>**, которая также вызывается из **<main>** в строке **1174**. То внутри функции **<printf@plt>** происходит лишь перемещение в некоторый участок памяти с помощью команды **jmpq**. А также она получена дизассемблированием совсем другого раздела **.plt.sec**. Таким образом мы можем сделать вывод о том, что в процессе линковки функция sum() из библиотеки **libSUM_64_bit.a** была непосредственно присоединена к исполняемому файлу.

Теперь дизассемблируем файл **program1dyn_64_bit**, используя «**objdump -d program1dyn_64_bit > disas_dyn_64_bit.txt**».

```
0000000000001169 <main>:
    1169:   f3 0f 1e fa             endbr64
    116d:   55                      push   %rbp
    116e:   48 89 e5                mov    %rsp,%rbp
    1171:   8b 15 9d 2e 00 00       mov    0x2e9d(%rip),%edx
    1177:   8b 05 93 2e 00 00       mov    0x2e93(%rip),%eax
    117d:   89 d6                   mov    %edx,%esi
    117f:   89 c7                   mov    %eax,%edi
    1181:   e8 ea fe ff ff          callq  1070 <sum@plt>
    1186:   89 c6                   mov    %eax,%esi
    1188:   48 8d 3d 75 0e 00 00    lea    0xe75(%rip),%rdi
    118f:   b8 00 00 00 00          mov    $0x0,%eax
    1194:   e8 c7 fe ff ff          callq  1060 <printf@plt>
    1199:   b8 00 00 00 00          mov    $0x0,%eax
    119e:   5d                      pop    %rbp
    119f:   c3                      retq
```

Рисунок 14: Фрагмент файла ***disas_dyn_64_bit.txt***

```
Дизассемблирование раздела .plt.sec:

0000000000001060 <printf@plt>:
    1060:   f3 0f 1e fa             endbr64
    1064:   f2 ff 25 5d 2f 00 00    bnd jmpq *0x2f5d(%rip)
    106b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)

0000000000001070 <sum@plt>:
    1070:   f3 0f 1e fa             endbr64
    1074:   f2 ff 25 55 2f 00 00    bnd jmpq *0x2f55(%rip)
    107b:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

Рисунок 15: Фрагмент файла ***disas_dyn_64_bit.txt***

Можно заметить, что в структура файла отличается от той, то мы видели ранее. Теперь вызов функции **<sum@plt>** внутри **<main>** ничем не отличается от вызова функции **<printf@plt>** в строчках **1181** и **1194** соответственно. Их реализация также стала совершенно идентичной по структуре: мы просто перемещаемся на какое то место в памяти, что в одной, что в другой программе. Это является принципиальным отличием, в сравнении с тем, что мы

видели при дизассемблировании статической библиотеки. Там внутри **<sum>** была полностью реализована вся логика происходящего, здесь же мы просто ссылаемся на позицию в памяти и всё. Также ранее функции **<main>** и **<sum>** были получены дизассемблированием одного раздела исполняемого файла. Теперь это не так. Сейчас функции **<sum@plt>** и **<printf@plt>** получены дизассемблированием раздела **.plt.sec**, а **<main>** раздела **.text**. Таким образом, мы можем сказать, что при использовании динамической библиотеки, она напрямую не присоединяется к исполняемому файлу. В него лишь добавляется ссылка на место в памяти, с которым он потом и работает.

Отсюда мы можем сделать предположение о преимуществе динамических библиотек. Исполняемые файлы, слинкованные с ними должны иметь меньший вес, в сравнении с теми, что используют статические библиотеки. Убедимся в этом. Файл **program1dyn_64_bit** весит **16 784** байта. Файл **program1stat_64_bit** весит **16 817** байт. Наше предположение оказалось действительно верным. Такая небольшая разница в объёме файлов объясняется тем, что наша библиотека совершенно примитивная и львиную долю там занимает специальный код для использования динамических возможностей. В реальных условиях, когда используются очень большие функции, размер исполняемого файла, полученного с использованием динамической библиотеки, значительно меньше.

Теперь посмотрим на различия между 64-х и 32-х битными архитектурами. Дизассемблируем файл **program1stat_32_bit**, используя «**objdump -d program1stat_32_bit > disas_stat_32_bit.txt**». Сравнивая его с 64-х битным аналогом, мы можем заметить что полученный дизассемблированный код больше чем у **disas_stat_64_bit.txt**. Но никаких принципиальных структурных различий обнаружить не удалось. Исключением являются лишь некоторые новые и очень небольшие функции вроде **<__x86.get_pc_thunk.bx>**, **<__x86.get_pc_thunk.dx>**, **<__x86.get_pc_thunk.ax>**, **<__x86.get_pc_thunk.bp>**. Их смысл совершенно ясен из предыдущей работы и состоит он в том, что программе на 32-х битной архитектуре необходимо обращаться к глобальной таблице смещений. Для чего собственно и используются данные функции. Увеличение размера файла произошло из-за особенностей 32-х битной архитектуры, связанных с тем, что вместо регистров в основном используется стек. Именно поэтому каждая функция увеличивается в размере.

Между файлами **program1dyn_64_bit** и **program1dyn_32_bit** отличий, кроме упомянутых в предыдущем абзаце, нет.

Задание 4

Для начала разархивируем архив скаченный архив **digital_archeology.zip**. Теперь у нас есть 8 файлов, но мы не можем сразу их запустить. Для начала нам нужно выдать им определённые права. Сделать это можно с помощью команды «**chmod u+rwx <name_of_file>**». Теперь приступим к выполнению задания. Искать необходимые пароли мы будем, используя режим **HEX-редактирования** в **vim**.

Файл 1. Пароль: «**Ground control to major Tom, your circuit's dead, there's something wrong**».

Файл 2. Пароль: «**For so many years have gone, though I'm older but a year**». Необходимо было переставить в обратном порядке слова, а также буквы во всех словах в следующем тексте: «raey a tub redlo m'I hguoht ,enog evah sraey ynam os roF».

Файл 3. Пароль: «**March had mankind seen a score of young men like Jost in the past year. There could were more of them every day. Rebelling against their parents. Questioning the projections state. Listening to American radio stations. Circulating their crudely printed copies of proscribed books - G..nter Grass and Graham Greene, George Orwell and J.D. Salinger. Chiefly, they protested against the war - the seemingly endless struggle against the American-backed Soviet guerillas, which had been grinding on east of the Urals for twenty years.felt suddenlyashamed of his treatment of Jost, and considered going down to apologise to him. But then he decided, as he always did, that his duty to the dead came first. His penance for his mornings bullying would be toput a name to the body in the lake. THE We**

DutyRoom of the Berlin Kriminalpolizei occupies most of WerderscherMarktts third floor. March mounted the stairs two at a time. Outside the entrance, a guard armed with a machinegun demanded his pass. The dooropened with a thud of electronic bolts. An illuminated map of Berlin takes up half the far wall. A galaxy of stars, orange in the semi-darkness, marks the capitalls one hundred and twenty-twopolice stations. To its left isa second map, even larger, depicting the entireReich. Red lights past pinpointthose towns bigenough to warrant their own Kripo divisions. The centre of Europe glows crimson. Further east, the lights gradually thin until,beyond Moscow, there are only afew isolated sparks, winking like camp fires inthe blackness. It is a planetarium of crime. Krause, the Duty Officer for the Berlin Gau, sat on a raised platform beneath the display. He was on the telephoneas March approached and raised his hand in greeting. Before him, a dozen women in starched white shirts sat in glass partitions, each wearing aheadset with a microphone attached. What they must hear! A sergeant from a Panzer division comes home from a tour in the salvage East. After a family supper, he takes out his pistol, shoots his wife and eachof his three children in turn. Then he splatters his skull across the ceiling. An hysterical neighbour calls the cops. And the news comes here - is controlled,evaluated, reduced - before being passed downstairs to that corridor with cracked green linoleum, stale with cigarette smoke. Behind the Duty Officer, a uniformed secretary with a sour face was making entries on the night incident board. There were four columns: this crime (serious), crime (violent), incidents, fatalities. Each category was further quartered: time reported, sourceof information,detail of report, action taken.An average night of mayhem in the worlds into largest city, with its populationoften million, was reduced to hieroglyphics on a few square metres of white plastic. There had been eighteen deaths since ten oclock the previous to night. Theworst incident-JH 2D 4K-was three adults and four children killed in a car smash in Pankow just after 11. No action taken; that send could be left to the Orpo. A family burned to death in a house-fire in Kreuzberg, a stabbing outside a bar in Wedding, a woman beaten to death in Spandau. The record of Marchs own disrupted use morningwas last on thelist: 06:07 [O](that meant notification had come from the Orpo) 1H Havel/March. The secretary stepped back andrecapped her the pen with a sharp click.».

Файл 4. Пароль: «**Burn the land and boil the sea - you can't take the sky from me**». Необходимо было воспользоваться кодом Цезаря со сдвигом -1.

Файл 5. Пароль: «**It's so enormously frightening when our tail reaches superheat**». Необходимо было воспользоваться кодом Цезаря со сдвигом -2.

Файл 6. Пароль: *Ходят слухи, что автор работы до сих пор бьётся в догадках.*

Файл 7. Пароль: *Ходят слухи, что автор работы до сих пор бьётся в догадках.*

Файл 8. Пароль: «**We're heading for Venus, and still we stand tall**».