1. Программа на Си

```c
#include <stdio.h>

#define Hi "Hi, world\n"
#define OK 0

int main(void)
{
    printf(Hi);
    return OK;
}
```

2. Этапы получения исходного файла

Препроцессирование

На этом этапе препроцессор принимает исходный файл **main.c** и генерирует промежуточный файл **main.i** (единица трансляции) , содержащий код программы после выполнения действий препроцессора.

Препроцессор выполняет следующие действия:

- удаление комментариев;
- вставку файлов (директива include);
- текстовые замены (по-другому говорят - раскрытие макросов, директива define);
- условную компиляцию (директива if)

Команда:

```
cpp main.c -o main.i
```

Результат этапа:

```
# 0 "main.c"
# 0 "<built-in>"
# 0 "<command-line>"

…
extern int printf (const char *__restrict __format, ...);
```

```
…

# 6 "main.c"
int main(void)
{
 printf("Hi, world\n");
 return 0;
}
```

Транслирование на язык ассемблера

Файл **main.i** , полученный препроцессором, передается на вход транслятору c99, который переводит его с языка Си на язык ассемблера. В итоге получаем программу на языке ассемблера в файле **main.s**.

Команды языка ассемблера практически соответствуют командам процессора.

Транслирование на язык ассемблера позволяет:

- упростить реализацию и отладку транслятора в машинный код
- повысить переносимость с одного устройства на другое

Команда:

```
c99 -S -fverbose-asm main.i
```

Результат этапа:

```
    .file   "main.c"
# GNU C99 (Ubuntu 11.3.0-1ubuntu1~22.04) version 11.3.0 (x86_64-linux-gnu)
#   compiled by GNU C version 11.3.0, GMP version 6.2.1, MPFR version 4.1.0, MPC
version 1.2.1, isl version isl-0.24-GMP

# GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
# options passed: -mtune=generic -march=x86-64 -std=c99 -fasynchronous-unwind-
tables -fstack-protector-strong -fstack-clash-protection -fcf-protection
    .text
    .section    .rodata
```

```
.LC0:
    .string "Hi, world"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    endbr64
    pushq   %rbp    #
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq    %rsp, %rbp  #,
    .cfi_def_cfa_register 6
# main.c:8:     printf(Hi);
    leaq    .LC0(%rip), %rax    #, tmp84
    movq    %rax, %rdi  # tmp84,
    call    puts@PLT    #
# main.c:9:     return OK;
    movl    $0, %eax    #, _3
# main.c:10: }
    popq    %rbp    #
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
    .section    .note.GNU-stack,"",@progbits
    .section    .note.gnu.property,"a"
    .align 8
    .long   1f - 0f
    .long   4f - 1f
    .long   5
0:
    .string "GNU"
1:
    .align 8
    .long   0xc0000002
    .long   3f - 2f
2:
    .long   0x3
3:
    .align 8
4:
```

Ассемблирование

С языка ассемблера программа переводится в машинный код. На выходе этого транслятора получается двоичный файл (объектный файл).

Объектный файл – содержащий скомпилированный объектный код

Команда:

```
as main.s -o main.o
```

Результат этапа:

С помощью утилиты **objdump**

```
objdump -drw main.o
```

Результат:

```
main.o:     формат файла elf64-x86-64

Дизассемблирование раздела .text:

0000000000000000 <main>:

   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax    # f <main+0xf>    b: R_X86_64_PC32    .rodata-0x4
   f:   48 89 c7                mov    %rax,%rdi
  12:   e8 00 00 00 00          call   17 <main+0x17>    13: R_X86_64_PLT32   puts-0x4
  17:   b8 00 00 00 00          mov    $0x0,%eax
  1c:   5d                      pop    %rbp
  1d:   c3                      ret
```

С помощью утилиты **nm**

```
nm --format=sysv main.o
```

Ключ –format=sysv выводит символы в формате таблицы

Результат:

```
Символы из main.o:

Имя                  Знач.           Класс        Тип          Размер            Строка Раздел

main                |0000000000000000|   T  |              FUNC|000000000000001e|      |.text
puts                |                |   U  |            NOTYPE|                |      |*UND*
```

Компоновка

Вызываем компоновщик чтобы получить исполняемый файл **main.exe**.

Исполняемый файл - файл, содержащий программу в виде, в котором она может быть исполнена компьютером.

Компоновщик решает несколько задач:

- объединяет несколько объектных файлов в единый исполняемый файл
- выполняет связывание переменных и функций, которые требуются очередному объектному файлу, но находятся где-то в другом месте
- добавляет специальный код, который подготавливает окружение для вызова функции main, а после ее завершения выполняет обратные действия

Команда:

```
ld main.o -dynamic-linker /lib64/ld-linux-x86-64.so.2
/usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o
/usr/lib/x86_64-linux-gnu/crtn.o -lc -o main.exe
```

Ключ **-lc** указывает линкеру на связь со стандартными библиотеками C с именем **libc.so**

Ключ **-dynamic-linker** используется для задания пути к динамической библиотеке, которая будет использоваться в качестве динамического линкера при загрузке исполняемого файла

Динамические библиотеки:

/lib64/ld-linux-x86-64.so.2 - путь к разделяемой библиотеке динамического компоновщика

/usr/lib/x86_64-linux-gnu/crt1.o;/usr/lib/x86_64-linux-gnu/crti.o; /usr/lib/x86_64-linux-gnu/crtn.o - это объектные файлы, содержащие код запуска.

Результат компановки:

Исполняемый файл **main.exe**

Результат работы исполняемого файла:

Hi, world

3. Программы драйверы

gcc и **clang** называют программами драйверами, т.к. они соединяют весь процесс компиляции автоматизированным образом(т.е. вызывает каждый инструмент в компиляторе с соответствующими параметрами и порядком, и в итоге производит исполняемый файл).

4. Ключи -v и -save-temps

Ключ -v

Используется для вывода более подробной информации о процессе компиляции и линковки. Он позволяет увидеть все вызовы компонентов компилятора и линковщика, а также параметры, переданные им при выполнении.

```
gcc -v main.c -o main.exe
```

Результат:

```
Using built-in specs.

COLLECT_GCC=gcc

COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper

OFFLOAD_TARGET_NAMES=nvptx-none:amdgcn-amdhsa
```

```
OFFLOAD_TARGET_DEFAULT=1

Target: x86_64-linux-gnu

Configured with: ../src/configure -v --with-pkgversion='Ubuntu 11.3.0-1ubuntu1~22.04' --
with-bugurl=file:///usr/share/doc/gcc-11/README.Bugs --enable-
languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++,m2 --prefix=/usr --with-gcc-major-
version-only --program-suffix=-11 --program-prefix=x86_64-linux-gnu- --enable-shared --
enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-
threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --
enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --
enable-gnu-unique-object --disable-vtable-verify --enable-plugin --enable-default-pie --
with-system-zlib --enable-libphobos-checking=release --with-target-system-zlib=auto --
enable-objc-gc=auto --enable-multiarch --disable-werror --enable-cet --with-arch-32=i686 --
with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --
enable-offload-targets=nvptx-none=/build/gcc-11-xKiWfi/gcc-11-11.3.0/debian/tmp-
nvptx/usr,amdgcn-amdhsa=/build/gcc-11-xKiWfi/gcc-11-11.3.0/debian/tmp-gcn/usr --without-
cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --
target=x86_64-linux-gnu --with-build-config=bootstrap-lto-lean --enable-link-serialization=2

Thread model: posix

Supported LTO compression algorithms: zlib zstd

gcc version 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04)

COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64' '-dumpdir' 'a-'

 /usr/lib/gcc/x86_64-linux-gnu/11/cc1 -quiet -v -imultiarch x86_64-linux-gnu main.c -quiet -
dumpdir a- -dumpbase main.c -dumpbase-ext .c -mtune=generic -march=x86-64 -version -
fasynchronous-unwind-tables -fstack-protector-strong -Wformat -Wformat-security -fstack-
clash-protection -fcf-protection -o /tmp/cc1zXA7a.s

GNU C17 (Ubuntu 11.3.0-1ubuntu1~22.04) version 11.3.0 (x86_64-linux-gnu)

      compiled by GNU C version 11.3.0, GMP version 6.2.1, MPFR version 4.1.0, MPC version
1.2.1, isl version isl-0.24-GMP



GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072

ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu"

ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/11/include-fixed"

ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/11/../../../../x86_64-linux-
gnu/include"

#include "..." search starts here:

#include <...> search starts here:

 /usr/lib/gcc/x86_64-linux-gnu/11/include

 /usr/local/include

 /usr/include/x86_64-linux-gnu

 /usr/include

End of search list.

GNU C17 (Ubuntu 11.3.0-1ubuntu1~22.04) version 11.3.0 (x86_64-linux-gnu)

      compiled by GNU C version 11.3.0, GMP version 6.2.1, MPFR version 4.1.0, MPC version
1.2.1, isl version isl-0.24-GMP



GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
```

```
Compiler executable checksum: 3f6cb05d963ad324b8f9442822c95179

COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64' '-dumpdir' 'a-'

 as -v --64 -o /tmp/ccvA9p3C.o /tmp/cc1zXA7a.s

GNU ассемблер, версия 2.38 (x86_64-linux-gnu); используется BFD версии (GNU Binutils for
Ubuntu) 2.38

COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/11/:/usr/lib/gcc/x86_64-linux-
gnu/11/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
gnu/11/:/usr/lib/gcc/x86_64-linux-gnu/

LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/11/:/usr/lib/gcc/x86_64-linux-
gnu/11/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-
gnu/11/../../../../lib/:/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-
gnu/:/usr/lib/../lib/:/usr/lib/gcc/x86_64-linux-gnu/11/../../../:/lib/:/usr/lib/

COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64' '-dumpdir' 'a.'

 /usr/lib/gcc/x86_64-linux-gnu/11/collect2 -plugin /usr/lib/gcc/x86_64-linux-
gnu/11/liblto_plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/11/lto-wrapper -plugin-
opt=-fresolution=/tmp/cc58tEXI.res -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-
through=-lgcc_s -plugin-opt=-pass-through=-lc -plugin-opt=-pass-through=-lgcc -plugin-opt=-
pass-through=-lgcc_s --build-id --eh-frame-hdr -m elf_x86_64 --hash-style=gnu --as-needed -
dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie -z now -z relro /usr/lib/gcc/x86_64-linux-
gnu/11/../../../x86_64-linux-gnu/Scrt1.o /usr/lib/gcc/x86_64-linux-gnu/11/../../../x86_64-
linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/11/crtbeginS.o -L/usr/lib/gcc/x86_64-linux-
gnu/11 -L/usr/lib/gcc/x86_64-linux-gnu/11/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-
linux-gnu/11/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-
gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/11/../../.. /tmp/ccvA9p3C.o -lgcc --
push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state --as-needed -lgcc_s --pop-
state /usr/lib/gcc/x86_64-linux-gnu/11/crtendS.o /usr/lib/gcc/x86_64-linux-
gnu/11/../../../x86_64-linux-gnu/crtn.o

COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=x86-64' '-dumpdir' 'a.'
```

Ключ **--save-temps** сохраняет временные файлы, созданные на этапах компиляции

```
gcc –save-temps main.c -o main.exe
```

Результат(получаем с помощью **ls**):

```
a-main.i  a-main.o  a-main.s  a.out
```

Ответы на вопросы:

a) Отличие компиляции gcc от ручной заключается в том, что gcc автоматически выполняет несколько этапов и использует больше ключей оптимизации и доп. библиотек

b) Содержимое временных файлов:
   - *.i – исходный код предпроцессора
   - *.s – код на языке ассемблера
   - *.o – объектный файл
   - *.exe – исполняемый файл

c) Временные файлы отличаются названиями и расширениями

d) Компоновка происходит с объектными файлами **crt1.o; crti.o; crtn.o**

e) Описание объектных файлов:
   - Стандартные библиотеки:
     i) **crt1.o** –выполняется перед функцией **main()**

ii) **crti.o** – выполняется при инициализации глобальных переменных

iii) **crtn.o** – выполняется при завершении работы программы

- Системные библиотеки
- Динамические библиотеки
- Другие библиотеки

5. Clang

Этапы получения исполняемого файла с помощью **clang** идентично получению исполняемого файла с помощью **gcc**. Различие в лицензии. **clang** использует лицензию LLVM и создает дополнительные файлы с расширениями **bc** и **tmp**

6. main_asm.s

В компилятор **gcc** параметры с языка ассемблера можно передавать с помощью ключа **-Wa,<option>**

```
gcc main.c -Wa,-al -o main_asm.s
```

Параметр **-al** выдает листинг с адресами, ассемблерным кодом и исходными строками

Результат:

```
GAS LISTING /tmp/ccNfALOz.s                     page 1


   1                            .file   "main.c"
   2                            .text
   3                            .section    .rodata
   4                        .LC0:
   5 0000 48692C20              .string "Hi, world"
   5      776F726C
   5      6400
   6                            .text
   7                            .globl  main
   8                            .type   main, @function
   9                        main:
  10                        .LFB0:
  11                            .cfi_startproc
  12 0000 F30F1EFA              endbr64
  13 0004 55                    pushq   %rbp
  14                            .cfi_def_cfa_offset 16
  15                            .cfi_offset 6, -16
  16 0005 4889E5                movq    %rsp, %rbp
  17                            .cfi_def_cfa_register 6
  18 0008 488D0500              leaq    .LC0(%rip), %rax
  18      000000
  19 000f 4889C7                movq    %rax, %rdi
  20 0012 E8000000              call    puts@PLT
  20      00
```

```
  21 0017 B8000000      movl    $0, %eax
  21      00
  22 001c 5D            popq    %rbp
  23                    .cfi_def_cfa 7, 8
  24 001d C3            ret
  25                    .cfi_endproc
  26                .LFE0:
  27                    .size   main, .-main
  28                    .ident  "GCC: (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0"
  29                    .section    .note.GNU-stack,"",@progbits
  30                    .section    .note.gnu.property,"a"
  31                    .align 8
  32 0000 04000000      .long   1f - 0f
  33 0004 10000000      .long   4f - 1f
  34 0008 05000000      .long   5
  35                0:
  36 000c 474E5500      .string "GNU"
  37                1:
  38                    .align 8
  39 0010 020000C0      .long   0xc0000002
  40 0014 04000000      .long   3f - 2f
  41                2:
  42 0018 03000000      .long   0x3
  43                3:
  44 001c 00000000      .align 8
  45                4:
```

7. Получение map-файла

В компилятор **gcc** параметры можно передавать с помощью ключа **-Wl, <option>**

```
gcc main.c -o main.exe -Wl,-Map=main.map
```

Map file - текстовый файл, который содержит информацию о том, какие объектные файлы были связаны, какие символы были определены и где они находятся.

Параметр **-Map** задает имя файла(main.map) карты символов

main.map содержит карту символов программы main.exe

8. Дизассемблирование

Дизассемблирование можно выполнить с помощью утилиты **objdump** и ключа **-d**

```
objdump -d main.o
```

Результат:

```
main.o:     формат файла elf64-x86-64


Дизассемблирование раздела .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <main+0xf>
   f:   48 89 c7                mov    %rax,%rdi
```

```
  12:   e8 00 00 00 00          call   17 <main+0x17>
  17:   b8 00 00 00 00          mov    $0x0,%eax
  1c:   5d                      pop    %rbp
  1d:   c3                      ret
```

Различие с транслятором

Дизассемблирование демонстрирует нам только секцию **.text**, в отличии от транслятора, так же демонстрирует все более структурированно

Секции объектного файла:

.text – содержит машинный код (или инструкции) программы

.data – содержит инициализированные статические переменные и глобальные переменные, которые будут располагаться в памяти при запуске программы.

.rodata – содержит только для чтения данные, такие как константы и строковые литералы.

.bss – содержит неинициализированные статические переменные и глобальные переменные.

9. Глобальные переменные

global_init – инициализированная глобальная переменная

global_uninit – неинициализированная глобальная переменная

Вывод символов объектного файла

```
Символы из main.o:

Имя                  Знач.           Класс        Тип          Размер            Строка Раздел

global_init         |0000000000000000|   D  |            OBJECT|0000000000000004|      |.data
global_uninit       |0000000000000000|   B  |            OBJECT|0000000000000004|      |.bss
main                |0000000000000000|   T  |              FUNC|000000000000001e|      |.text
puts                |                |   U  |            NOTYPE|                |      |*UND*
```

Локальные переменные находятся в секции стека

global_uninit находится в секции .bss

global_init находится в секции .data

10. Отладочная информация в объектном файле

Команда:

```
gcc -c main.c -g -o main.o
```

Объектный файл с отладочной информацией отличается.

В объектном файле содержатся секции **.debug_info; .debug_abbrev; .debug_line;**

.debug_str; .debug_line_str

```
main.o:     формат файла elf64-x86-64
main.o
архитектура: i386:x86-64, флаги 0x00000011:
HAS_RELOC, HAS_SYMS
начальный адрес 0x0000000000000000

Разделы:
Idx Name          Разм      VMA               LMA               Фа  смещ.  Выр.
  0 .text         0000001e  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  0000005e  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  0000005e  2**0
                  ALLOC
  3 .rodata       0000000a  0000000000000000  0000000000000000  0000005e  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .debug_info   0000008c  0000000000000000  0000000000000000  00000068  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  5 .debug_abbrev 00000045  0000000000000000  0000000000000000  000000f4  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_aranges 00000030  0000000000000000  0000000000000000  00000139  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  7 .debug_line   00000052  0000000000000000  0000000000000000  00000169  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  8 .debug_str    000000f7  0000000000000000  0000000000000000  000001bb  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_line_str 0000009f  0000000000000000  0000000000000000  000002b2  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 10 .comment      0000002c  0000000000000000  0000000000000000  00000351  2**0
                  CONTENTS, READONLY
 11 .note.GNU-stack 00000000  0000000000000000  0000000000000000  0000037d  2**0
                  CONTENTS, READONLY
 12 .note.gnu.property 00000020  0000000000000000  0000000000000000  00000380  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 13 .eh_frame     00000038  0000000000000000  0000000000000000  000003a0  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 main.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .rodata    0000000000000000 .rodata
0000000000000000 l    d  .debug_info    0000000000000000 .debug_info
0000000000000000 l    d  .debug_abbrev  0000000000000000 .debug_abbrev
0000000000000000 l    d  .debug_line    0000000000000000 .debug_line
0000000000000000 l    d  .debug_str 0000000000000000 .debug_str
0000000000000000 l    d  .debug_line_str    0000000000000000 .debug_line_str
0000000000000000 g     F .text  000000000000001e main
0000000000000000         *UND*  0000000000000000 puts



Дизассемблирование раздела .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <main+0xf>
            b: R_X86_64_PC32    .rodata-0x4
   f:   48 89 c7                mov    %rax,%rdi
  12:   e8 00 00 00 00          call   17 <main+0x17>
            13: R_X86_64_PLT32  puts-0x4
```

```
  17:   b8 00 00 00 00          mov    $0x0,%eax
  1c:   5d                      pop    %rbp
  1d:   c3                      ret
```

11. Исполняемый файл

Команда получения:

```
gcc main.o -o main.exe
```

12. Различия файлов с и без отладочной информации

a) Различия в размере

- Объектные файлы – файл с отладочной информацией больше

```
3560 debug_main.o
1496 main.o
```

- Исполняемые файлы – файл с отладочной информацией больше

```
17056 debug_main.exe
15960 main.exe
```

b) Различия в количестве секций(количество секций находится с помощью утилиты objdump и ключа -h)

- Объектные файлы – файл с отладочной информацией содержит больше секций

```
Разделы:
Idx Name          Разм      VMA               LMA               Фа  смещ.  Выр.
  0 .text         0000001e  0000000000000000  0000000000000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  0000000000000000  0000000000000000  0000005e  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  0000000000000000  0000000000000000  0000005e  2**0
                  ALLOC
  3 .rodata       0000000a  0000000000000000  0000000000000000  0000005e  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .debug_info   0000008c  0000000000000000  0000000000000000  00000068  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  5 .debug_abbrev 00000045  0000000000000000  0000000000000000  000000f4  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_aranges 00000030  0000000000000000  0000000000000000  00000139  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  7 .debug_line   00000052  0000000000000000  0000000000000000  00000169  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
  8 .debug_str    000000f7  0000000000000000  0000000000000000  000001bb  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_line_str 0000009f  0000000000000000  0000000000000000  000002b2  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 10 .comment      0000002c  0000000000000000  0000000000000000  00000351  2**0
                  CONTENTS, READONLY
 11 .note.GNU-stack 00000000  0000000000000000  0000000000000000  0000037d  2**0
                  CONTENTS, READONLY
 12 .note.gnu.property 00000020  0000000000000000  0000000000000000  00000380  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 13 .eh_frame     00000038  0000000000000000  0000000000000000  000003a0  2**3
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, DATA
```

- Исполняемые файлы – файл с отладочной информацией содержит больше секций

```
Разделы:
```

```
Idx Name          Разм      VMA               LMA               Фа  смещ.  Выр.
  0 .interp       0000001c  0000000000000318  0000000000000318  00000318  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.gnu.property 00000030  0000000000000338  0000000000000338  00000338  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000000368  0000000000000368  00000368  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .note.ABI-tag 00000020  000000000000038c  000000000000038c  0000038c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     00000024  00000000000003b0  00000000000003b0  000003b0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .dynsym       000000a8  00000000000003d8  00000000000003d8  000003d8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .dynstr       0000008d  0000000000000480  0000000000000480  00000480  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .gnu.version  0000000e  000000000000050e  000000000000050e  0000050e  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .gnu.version_r 00000030  0000000000000520  0000000000000520  00000520  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .rela.dyn     000000c0  0000000000000550  0000000000000550  00000550  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .rela.plt     00000018  0000000000000610  0000000000000610  00000610  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .init         0000001b  0000000000001000  0000000000001000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .plt          00000020  0000000000001020  0000000000001020  00001020  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .plt.got      00000010  0000000000001040  0000000000001040  00001040  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .plt.sec      00000010  0000000000001050  0000000000001050  00001050  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .text         00000107  0000000000001060  0000000000001060  00001060  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 16 .fini         0000000d  0000000000001168  0000000000001168  00001168  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 17 .rodata       0000000e  0000000000002000  0000000000002000  00002000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 18 .eh_frame_hdr 00000034  0000000000002010  0000000000002010  00002010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 19 .eh_frame     000000ac  0000000000002048  0000000000002048  00002048  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 20 .init_array   00000008  0000000000003db8  0000000000003db8  00002db8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .fini_array   00000008  0000000000003dc0  0000000000003dc0  00002dc0  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .dynamic      000001f0  0000000000003dc8  0000000000003dc8  00002dc8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 23 .got          00000048  0000000000003fb8  0000000000003fb8  00002fb8  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 24 .data         00000010  0000000000004000  0000000000004000  00003000  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 25 .bss          00000008  0000000000004010  0000000000004010  00003010  2**0
                  ALLOC
 26 .comment      0000002b  0000000000000000  0000000000000000  00003010  2**0
                  CONTENTS, READONLY
 27 .debug_aranges 00000030  0000000000000000  0000000000000000  0000303b  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 28 .debug_info   0000008c  0000000000000000  0000000000000000  0000306b  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 29 .debug_abbrev 00000045  0000000000000000  0000000000000000  000030f7  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 30 .debug_line   00000052  0000000000000000  0000000000000000  0000313c  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 31 .debug_str    000000d9  0000000000000000  0000000000000000  0000318e  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 32 .debug_line_str 0000004c  0000000000000000  0000000000000000  00003267  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

c) Расположение функций, глобальных и локальных переменных не изменилось

```
Символы из main.o:

Имя                  Знач.           Класс        Тип          Размер            Строка Раздел

main                |0000000000000000|   T  |              FUNC|000000000000001e|      |.text
puts                |                |   U  |            NOTYPE|                |      |*UND*
```

```
Символы из debug_main.o:

Имя                     Знач.           Класс        Тип           Размер            Строка Раздел

main                |0000000000000000|   T  |              FUNC|000000000000001e|      |.text
puts                |                |   U  |            NOTYPE|                |      |*UND*
```

13. Динамические библиотеки

Исполняемый файл использует динамическую библиотеку **libc.so.6**

libc.so.6 – содержит реализации стандартных функций языка Си.