Задание №2 в рамках вычислительного практикума

Этапы получения исполняемого файла

Оглавление

Цель работы

Целью данной работы является изучение процесса получения исполняемого файла и организации объектных и исполняемых файлов.

Задание

1. Текст программы

```
#include <stdio.h>
#define N "HI"


int main(void)
{
    printf("%s", N);
    return 0;
}
```

2. Изучение этапов получения исполняемого файла

1. Обработка предпроцессором

```
cpp main.c > main.i
```

Создается файл main.i

```
# 0 "main.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "main.c"
# 1 "/usr/include/stdio.h" 1 3 4
# 27 "/usr/include/stdio.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/libc-header-start.h" 1 3 4
# 33 "/usr/include/aarch64-linux-gnu/bits/libc-header-start.h" 3 4
# 1 "/usr/include/features.h" 1 3 4
# 472 "/usr/include/features.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 1 3 4
# 462 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/wordsize.h" 1 3 4
# 463 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 2 3 4
# 1 "/usr/include/aarch64-linux-gnu/bits/long-double.h" 1 3 4
# 464 "/usr/include/aarch64-linux-gnu/sys/cdefs.h" 2 3 4
# 473 "/usr/include/features.h" 2 3 4
# 496 "/usr/include/features.h" 3 4
# 1 "/usr/include/aarch64-linux-gnu/gnu/stubs.h" 1 3 4
```

2. Трансляция на языке ассемблера

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

Создается файл main.s

```
.arch armv8-a
        .file   "main.c"
// GNU C99 (Debian 11.2.0-13) version 11.2.0 (aarch64-linux-gnu)
//      compiled by GNU C version 11.2.0, 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: -mlittle-endian -mabi=lp64 -std=c99 -fasynchronous-unwind-
tables
        .text
        .section .rodata
        .align  3
```

```
.LC0:
        .string "HI"
        .align  3
.LC1:
        .string "%s"
        .text
        .align  2
        .global main
        .type   main, %function
main:
.LFB0:
        .cfi_startproc
        stp     x29, x30, [sp, -16]!    //,,,
        .cfi_def_cfa_offset 16
        .cfi_offset 29, -16
        .cfi_offset 30, -8
        mov     x29, sp //,
// main.c:7:     printf("%s", N);
        adrp    x0, .LC0// tmp94,
        add     x1, x0, :lo12:.LC0      //, tmp94,
        adrp    x0, .LC1// tmp95,
        add     x0, x0, :lo12:.LC1      //, tmp95,
        bl      printf          //
// main.c:8:     return 0;
        mov     w0, 0   // _3,
// main.c:9: }
        ldp     x29, x30, [sp], 16      //,,,
        .cfi_restore 30
        .cfi_restore 29
        .cfi_def_cfa_offset 0
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 11.2.0-13) 11.2.0"
        .section .note.GNU-stack,"",@progbits
```

3. Ассемблирование в объектный файл

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

Создается объектный файл (машинный код)

4. Компановка

```
ld -o main.exe main.o /usr/lib/aarch64-linux-gnu/crt1.o /usr/lib/aarch64-
linux-gnu/crti.o /usr/lib/aarch64-linux-gnu/crtn.o /usr/lib/gcc/aarch64-
linux-gnu/11/crtbegin.o /usr/lib/gcc/aarch64-linux-gnu/11/crtend.o
/usr/lib/aarch64-linux-gnu/libc.so -I /lib64/ld-linux-x86-64.so.2
```

Создается исполняемый файл программы (машинный код)

3. Gcc и Clang – программы драйверы

Мы называем Gcc и Clang программами драйверами, так как они вызывают другие команды последовательно

4. Описание ключей v и save-temps

ключ -v

Показывает информацию о программе - версию и конфигурацию

ключ --save-temps - оставляет файлы этапов сборки:

результат работы предроцессора

файл, получившийся в результате трансляции на языке ассемблера

объектный файл
исполняемый файл

a) Видимых отличий нет
b) Содержимое верменных файлов на этапах сборки
    - Файл единици трансляции - исходная программа без комментариев и с библиотеками.
    - Файл ассемблера - файл, который содержит текст исходной программы на языке ассемблер.
    - Объектный файл состоит из секций которые содержат данные в широком смысле этого слова - заголовки код данные таблицу символов
c) Файлы идентичны
d) Компановка программы происходит с этими файлами и библиотеками

```
/usr/lib/gcc/aarch64-linux-gnu/11/collect2 -plugin /usr/lib/gcc/aarch64-linux-gnu/11/liblto_plugin.so -plugin-opt=/usr/lib/gcc/aarch64-linux-gnu/11/lto-wrapper -plugin-opt=-fresolution=/tmp/ccnPbqjt.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 --hash-style=gnu --as-needed -dynamic-linker /lib/ld-linux-aarch64.so.1 -X -EL -maarch64linux --fix-cortex-a53-843419 -pie /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/Scrt1.o /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/crti.o /usr/lib/gcc/aarch64-linux-gnu/11/crtbeginS.o -L/usr/lib/gcc/aarch64-linux-gnu/11 -L/usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu -L/usr/lib/gcc/aarch64-linux-gnu/11/../../../../lib -L/lib/aarch64-linux-gnu -L/lib/../lib -L/usr/lib/aarch64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/aarch64-linux-gnu/11/../../../.. /tmp/ccxcVsWX.o -lgcc --push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state --as-needed -lgcc_s --pop-state /usr/lib/gcc/aarch64-linux-gnu/11/crtendS.o /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/crtn.o
```

e) Объектные файлы используются для компановки

5. Этапы получения исполняемого файла, которые выполняет clang

1. Препроцессинг
2. Парсинг и семантический анализ
3. Генерация и оптимизация кода
4. Ассемблирование
5. Линовка

6. Получение ассемблерного листинга программы с помощью gcc

```
gcc -S -o main_asm.s main.c
```

```
.arch armv8-a
        .file   "main.c"
        .text
        .section .rodata
        .align  3
.LC0:
        .string "HI"
        .align  3
.LC1:
        .string "%s"
        .text
        .align  2
        .global main
        .type   main, %function
```

```
main:
.LFB0:
        .cfi_startproc
        stp     x29, x30, [sp, -16]!
        .cfi_def_cfa_offset 16
        .cfi_offset 29, -16
        .cfi_offset 30, -8
        mov     x29, sp
        adrp    x0, .LC0
        add     x1, x0, :lo12:.LC0
        adrp    x0, .LC1
        add     x0, x0, :lo12:.LC1
        bl      printf
        mov     w0, 0
        ldp     x29, x30, [sp], 16
        .cfi_restore 30
        .cfi_restore 29
        .cfi_def_cfa_offset 0
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 11.3.0-1) 11.3.0"
        .section .note.GNU-stack,"",@progbits
```

7. Получение map файла

```
gcc main.c -Xlinker -Map=main.map
```

```
Для удовлетворения ссылок на файл (символ) включён член архива

/usr/lib/aarch64-linux-gnu/libc_nonshared.a(elf-init.oS)
                              /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-
linux-gnu/Scrt1.o (__libc_csu_init)

Для удовлетворения ссылок на файл (символ) включена библиотека по необходимости

libc.so.6                     /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-
linux-gnu/Scrt1.o (abort@@GLIBC_2.17)

Отброшенные входные разделы

 .note.GNU-stack
                0x0000000000000000        0x0 /usr/lib/gcc/aarch64-linux-
gnu/11/../../../aarch64-linux-gnu/Scrt1.o
 .....
Настройки памяти

Имя           Происхождение Длина         Атрибуты
*default*        0x0000000000000000 0xffffffffffffffff

Сценарий компоновщика и карта памяти

LOAD /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/Scrt1.o
......
START GROUP
LOAD /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/libgcc_s.so.1
LOAD /usr/lib/gcc/aarch64-linux-gnu/11/libgcc.a
END GROUP
.....
LOAD /usr/lib/gcc/aarch64-linux-gnu/11/../../../aarch64-linux-gnu/crtn.o
                [!provide]                      PROVIDE (__executable_start =
SEGMENT_START ("text-segment", 0x0))
```

```
                0x0000000000000238                . = (SEGMENT_START ("text-
segment", 0x0) + SIZEOF_HEADERS)

.interp         0x0000000000000238       0x1b
 *(.interp)
 .interp        0x0000000000000238       0x1b /usr/lib/gcc/aarch64-linux-
gnu/11/../../../aarch64-linux-gnu/Scrt1.o
.....

.gcc_except_table
 *(.gcc_except_table .gcc_except_table.*)

.gnu_extab
 *(.gnu_extab*)

.exception_ranges
 *(.exception_ranges*)
                0x0000000000010db8                . = DATA_SEGMENT_ALIGN (CONSTANT
(MAXPAGESIZE), CONSTANT (COMMONPAGESIZE))

.eh_frame
 *(.eh_frame)
 *(.eh_frame.*)

.....
 .data          0x0000000000011038        0x0 /tmp/ccnCaZak.o
 ......
.tm_clone_table
 ......
.data1
 *(.data1)
                0x0000000000011038                _edata = .
                [!provide]                        PROVIDE (edata = .)

.....

.comment        0x0000000000000000       0x1e
 *(.comment)
 .comment       0x0000000000000000       0x1e /usr/lib/gcc/aarch64-linux-
gnu/11/crtbeginS.o
                                         0x1f (размер перед ослаблением)
 .comment       0x000000000000001e       0x1f /tmp/ccnCaZak.o
 .comment       0x000000000000001e       0x1f /usr/lib/gcc/aarch64-linux-
gnu/11/crtendS.o

.....

/DISCARD/
 *(.note.GNU-stack)
 *(.gnu_debuglink)
 *(.gnu.lto_*)
OUTPUT(a.out elf64-littleaarch64)
LOAD linker stubs
```

8. Дизасемблирование объектного файла

```
objdump -D main.o > main_dis.s
```

Разное количество секций, дизасемблированный файл разнесен по разделам, результат исполнения одинаковый

9. Таблица символов и секций объектного файла

```
objdump -t -h main_glob.o > table_symb.s
```

```
main_glob.o:     формат файла elf64-littleaarch64

Разделы:
Idx Name          Разм      VMA               LMA               Фа  смещ.  Выр.
0 .text         00000028  0000000000000000  0000000000000000  00000040  2**2
CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
1 .data         00000004  0000000000000000  0000000000000000  00000068  2**2
CONTENTS, ALLOC, LOAD, DATA
2 .bss          00000004  0000000000000000  0000000000000000  0000006c  2**2
ALLOC
3 .rodata       0000000b  0000000000000000  0000000000000000  00000070  2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA
4 .comment      0000001f  0000000000000000  0000000000000000  0000007b  2**0
CONTENTS, READONLY
5 .note.GNU-stack 00000000  0000000000000000  0000000000000000  0000009a  2**0
CONTENTS, READONLY
6 .eh_frame     00000038  0000000000000000  0000000000000000  000000a0  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  .data    0000000000000000 .data
0000000000000000 l    d  .bss     0000000000000000 .bss
0000000000000000 l    d  .rodata  0000000000000000 .rodata
0000000000000000 l    d  .note.GNU-stack  0000000000000000 .note.GNU-stack
0000000000000000 l    d  .eh_frame        0000000000000000 .eh_frame
0000000000000000 l    d  .comment 0000000000000000 .comment
0000000000000000 g     O .data    0000000000000004 global_a
0000000000000000 g     O .bss     0000000000000004 global_b
0000000000000000 g     F .text    0000000000000028 main
0000000000000000         *UND*    0000000000000000 printf
```

В 6-ю таблицу попали символы и глобальные переменные

10.Добавляем отладочную информацию к объектному файлу

```
gcc -g main.c -c -o otlad_main.o
objdump -t -h otlad_main.o > table_symb_otlad.s
```

Добавилась секция debug

11.Получаем исполняемый файл (без отладочной информации)

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

12.Вопросы

1. Как отличаются объектный файл и исполняемый файлы с отладочной информацией и без по размеру

```
ls -lh
```

Объектный файл без отладочной информации - 1.8 кб
Объектный файл с отладочной информацией - 4 кб
Исполняемый файл без отладочной информации - 9.2 кб
Исполняемый файл с отладочной информацией - 11 кб

2. Как отличаются объектный файл и исполняемый файлы с отладочной информацией и без по количеству секций

Объектный файл без отладочной информации - 6 секций
Объектный файл с отладочной информацией - 12 секций
Исполняемый файл без отладочной информации - 24 секций
Исполняемый файл с отладочной информацией - 29 секций

3. Изменилось ли расположение функций, глобальных и локальных переменных

Нет, не изменилось

13.Динамические библиотеки, которые использует исполняемый файл

```
ldd main.exe
```

```
linux-vdso.so.1 (0x0000ffff87cec000)
libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6 (0x0000ffff87b15000)
/lib/ld-linux-aarch64.so.1 (0x0000ffff87cb9000)
```