ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
ФАКУЛЬТЕТ АЭРОКОСМИЧЕСКИХ ТЕХНОЛОГИЙ

Лабораторная 4

Статические и динамические библиотеки

Рогозин Владимир
Группа Б03-106

Пункт 1: Статическая библиотека

Статическая библиотека, по сути, содержит объединённые объектные файлы с инструкциями, которые потом вставляются в бинарный код. Команды подключаются и вставляются в код на этапе линковки. В итоге получаем один исполняемый файл со всеми инструкциями. Очевидным минусом статических библиотек является то, что при обновлении библиотеки придётся перекомпилировать каждый файл, использующий её, а также размер исполняемого файла может сильно увеличиваться при использовании статических библиотек. Создадим простейшую статическую библиотеку, состоящую из одной функции, которая суммирует два целых числа. На первой картинке представлен код программы, использующей эту функцию.

```c
#include <stdio.h>

int sum(int x, int y);

int main() {
        int x = 5;
        int y = 4;
        printf("Sum of x and y is %d\n", sum(x, y));
        return 0;
}
```

Рис. 1: Код программы на C

```
.text
.globl sum
sum:
        movl    4(%esp), %eax
        addl    8(%esp), %eax
        ret
```

(a) Листинг функции 32-х битной системы

```
.text
.globl sum
sum:
        movl    %esi, %eax
        addl    %edi, %eax
        ret
```

(b) Листинг функции 64-х битной системы

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc -c sum.s -o sum.o
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ar rc libSUM.a sum.o
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ranlib libSUM.a
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc base.c libSUM.a -o base.out
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./base.out
Sum of x and y is 9
```

Рис. 3: Создание статической библиотеки

Теперь посмотрим на листинг первой программы для обеих систем. Видим, что наша функция *sum* вызывается через *call* с припиской *@PLT* (Procedure Linkage Table), чего бы не было при вызове функции,

если бы она была описана в этом же файле. Заметить, что также вызывается и *printf*, то есть можно предположить, что все стандартные функции (*printf, scanf etc.*) реализованы примерно также.

```
main:
.LFB0:
        .cfi_startproc
        endbr32
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ebx
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x78,0x6
        .cfi_escape 0x10,0x3,0x2,0x75,0x7c
        subl    $16, %esp
        call    __x86.get_pc_thunk.bx
        addl    $_GLOBAL_OFFSET_TABLE_, %ebx
        movl    $5, -16(%ebp)
        movl    $4, -12(%ebp)
        subl    $8, %esp
        pushl   -12(%ebp)
        pushl   -16(%ebp)
        call    sum@PLT
        addl    $16, %esp
        subl    $8, %esp
        pushl   %eax
        leal    .LC0@GOTOFF(%ebx), %eax
        pushl   %eax
        call    printf@PLT
        addl    $16, %esp
        movl    $0, %eax
        leal    -8(%ebp), %esp
        popl    %ecx
        .cfi_restore 1
        .cfi_def_cfa 1, 0
        popl    %ebx
        .cfi_restore 3
        popl    %ebp
        .cfi_restore 5
        leal    -4(%ecx), %esp
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

(a) Листинг программы, использующей функцию sum, 32 бита

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -8(%rbp)
        movl    $4, -4(%rbp)
        movl    -4(%rbp), %edx
        movl    -8(%rbp), %eax
        movl    %edx, %esi
        movl    %eax, %edi
        call    sum@PLT
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Листинг программы, использующей функцию sum, 64 бита

Пункт 2: Динамическая библиотека

Динамическая библиотека работает немного иначе. Она состоит из подпрограмм, которые подключаются уже во время выполнения. Они не становятся частью бинарника , а так и остаются отдельными модулями, поэтому несколько программ могут использовать одну и ту же копию библиотеки, что может сэкономить много места. Ещё одним преимуществом таких библиотек является возможность легко обновлять файлы(не надо будет перекомпилировать все программы, использовавшие библиотеку до обновления). Минусом использования динамических библиотек может являться сложность в осуществлении связи между библиотекой и программой(система должна знать где и как найти нужную библиотеку). Теперь создадим динамическую библиотеку, посмотрим на листинг программы, использующей функцию из неё.

```
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc -shared -o libSUM.so sum.o -m32
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ gcc base.c -L. -lSUM -o base -m32
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ LD_LIBRARY_PATH=../base
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ sudo cp libSUM.so /usr/lib
clear@DESKTOP-FOMMSSB:~/assembler_3sem$ sudo ldconfig
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link

clear@DESKTOP-FOMMSSB:~/assembler_3sem$ ./base
Sum of x and y is 9
```

Рис. 5: Создание динамической библиотеки

Команда *ldconfig* создаёт необходимые привязки для динамических библиотек, которые используются компановщиками, которые выполняют связывание во время выполнения. *LD_LIBRARY_PATH* указывает где следует искать библиотеку.

Ассемблерные листинги абсолютно идентичны тем, которые получились в первом пункте. Функция *sum* вызывается точно также.

```
main:
.LFB0:
        .cfi_startproc
        endbr32
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ebx
        pushl   %ecx
        .cfi_escape 0xf,0x3,0x75,0x78,0x6
        .cfi_escape 0x10,0x3,0x2,0x75,0x7c
        subl    $16, %esp
        call    __x86.get_pc_thunk.bx
        addl    $_GLOBAL_OFFSET_TABLE_, %ebx
        movl    $5, -16(%ebp)
        movl    $4, -12(%ebp)
        subl    $8, %esp
        pushl   -12(%ebp)
        pushl   -16(%ebp)
        call    sum@PLT
        addl    $16, %esp
        subl    $8, %esp
        pushl   %eax
        leal    .LC0@GOTOFF(%ebx), %eax
        pushl   %eax
        call    printf@PLT
        addl    $16, %esp
        movl    $0, %eax
        leal    -8(%ebp), %esp
        popl    %ecx
        .cfi_restore 1
        .cfi_def_cfa 1, 0
        popl    %ebx
        .cfi_restore 3
        popl    %ebp
        .cfi_restore 5
        leal    -4(%ecx), %esp
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
```

(a) Листинг программы, использующей функцию sum, 32 бита

```
main:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $5, -8(%rbp)
        movl    $4, -4(%rbp)
        movl    -4(%rbp), %edx
        movl    -8(%rbp), %eax
        movl    %edx, %esi
        movl    %eax, %edi
        call    sum@PLT
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

(b) Листинг программы, использующей функцию sum, 64 бита

Пункт 3: Смотрим на бинарники

С помощью *objdump* посмотрим на бинарники в обоих случаях. Видим различия в местах вызова функции: в случае статической библиотеки идёт переход на строчку, где описана функция, в случае динамической библиотеки программа переходит на строчку, где лежат инструкции о том, где искать сам исполняемый файл. Заметны различия и в номерах строк куда переходит программа. В первом случае функция описана под *main'*ом(номер 117c, *main* заканчивается на 117b), во втором случае гораздо дальше, но примерно там же и лежит *printf* (номера 1070 и 1060 соответственно).

```
1159:       89 d6                   mov    %edx,%esi
115b:       89 c7                   mov    %eax,%edi
115d:       e8 1a 00 00 00          callq  117c <sum>
1162:       89 c6                   mov    %eax,%esi
1164:       48 8d 3d 99 0e 00 00    lea    0xe99(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
116b:       b8 00 00 00 00          mov    $0x0,%eax
1170:       e8 bb fe ff ff          callq  1030 <printf@plt>
1175:       b8 00 00 00 00          mov    $0x0,%eax
117a:       c9                      leaveq
117b:       c3                      retq
```

Рис. 7: Бинарник программы, использующей статическую библиотеку

```
116d:       55                      push   %rbp
116e:       48 89 e5                mov    %rsp,%rbp
1171:       48 83 ec 10             sub    $0x10,%rsp
1175:       c7 45 f8 05 00 00 00    movl   $0x5,-0x8(%rbp)
117c:       c7 45 fc 04 00 00 00    movl   $0x4,-0x4(%rbp)
1183:       8b 55 fc                mov    -0x4(%rbp),%edx
1186:       8b 45 f8                mov    -0x8(%rbp),%eax
1189:       89 d6                   mov    %edx,%esi
118b:       89 c7                   mov    %eax,%edi
118d:       e8 de fe ff ff          callq  1070 <sum@plt>
1192:       89 c6                   mov    %eax,%esi
1194:       48 8d 3d 69 0e 00 00    lea    0xe69(%rip),%rdi        # 2004 <_IO_stdin_used+0x4>
119b:       b8 00 00 00 00          mov    $0x0,%eax
11a0:       e8 bb fe ff ff          callq  1060 <printf@plt>
11a5:       b8 00 00 00 00          mov    $0x0,%eax
11aa:       c9                      leaveq
11ab:       c3                      retq
11ac:       0f 1f 40 00             nopl   0x0(%rax)
```

Рис. 8: Бинарник программы, использующей динамическую библиотеку