

CAOS 2 – syscall

Manakov Danila

MIPT

19 сентября 2022 г.

Этапы сборки проекта

- 1 препроцессор(Исходный код на C) → код на C
- 2 компилятор(код на C) → код на языке ассемблера
- 3 ассемблер(код на ассемблере) → машинный код
- 4 линковщик(машинный код) → ELF

Исходный код на C

```
// foo.h
#ifdef VAR
int a = 10;
#else
int a = VAR;
#endif
```

```
// main.c
#include <stdio.h>
#include "foo.h"
```

```
int main()
{
    printf("%d\n", a);
}
```

Код после препроцессора

```
729
730     # 3 "foo.h"
731     int a = 10;
732     # 3 "main.c" 2
733
734     int main()
735     {
736     |     printf("%d\n", a);
737     }
738
```

Автосгенерированный код на ассемблере

```
...
main:
.LFB0:
.cfi_startproc
endbr64
push     rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
mov      rbp, rsp
.cfi_def_cfa_register 6
mov      eax, DWORD PTR a[rip]
mov      esi, eax
lea      rdi, .LC0[rip]
mov      eax, 0
call     printf@PLT
mov      eax, 0
pop      rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
```

Объектный файл (целиком)

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

```
main.o:      file format elf64-x86-64
```

Disassembly of section .text:

0000000000000000 <main>:

0: f3 0f 1e fa

endbr64

4: 55

push %rbp

5: 48 89 e5

mov %rsp,%rbp

8: 8b 05 00 00 00 00

mov 0x0(%rip),%eax

e: 89 c6

mov %eax,%esi

10: 48 8d 3d 00 00 00 00

lea 0x0(%rip),%rdi

17: b8 00 00 00 00

mov \$0x0,%eax

1c: e8 00 00 00 00

callq 21 <main+0x21>

21: b8 00 00 00 00

mov \$0x0,%eax

26: 5d

pop %rbp

27: c3

retq

Кусочек ELF файла

```
// objdump -d test
```

```
...
```

```
00000000000001149: <main>:
```

```
1149: f3 0f 1e fa
```

```
114d: 55
```

```
114e: 48 89 e5
```

```
1151: 8b 05 b9 2e 00 00
```

```
1157: 89 c6
```

```
1159: 48 8d 3d a4 0e 00 00
```

```
1160: b8 00 00 00 00
```

```
1165: e8 e6 fe ff ff
```

```
116a: b8 00 00 00 00
```

```
116f: 5d
```

```
1170: c3
```

```
1171: 66 2e 0f 1f 84 00 00
```

```
1178: 00 00 00
```

```
117b: 0f 1f 44 00 00
```

```
...
```

```
endbr64
```

```
push    %rbp
```

```
mov     %rsp,%rbp
```

```
mov     0x2eb9(%rip),%eax
```

```
mov     %eax,%esi
```

```
lea     0xea4(%rip),%rdi
```

```
mov     $0x0,%eax
```

```
callq   1050 <printf@plt>
```

```
mov     $0x0,%eax
```

```
pop     %rbp
```

```
retq
```

```
nopw    %cs:0x0(%rax,%rax,
```

```
nopl    0x0(%rax,%rax,1)
```

(again) Этапы сборки проекта

- 1 препроцессор(Исходный код на C) \rightarrow код на C
- 2 компилятор(код на C) \rightarrow код на языке ассемблера
- 3 ассемблер(код на ассемблере) \rightarrow машинный код
- 4 линковщик(машинный код) \rightarrow ELF

Вопросики?)

Сборка при помощи скриптов

Как вообще можно собрать код?

```
1  $> gcc main.c
2  $> ls
3  main.c a.out
```

Что делать, если файлов несколько?

```
1 // a.h
2 #include <stdio.h>
3
4 void foo() {
5     printf("hello from foo\n");
6 }
```

```
1 // main.c
2 #include "a.h"
3
4 int main() {
5     foo();
6 }
```

И как собрать такой проект?)

Что делать, если файлов несколько?

```
1 // a.h
2 #include <stdio.h>
3
4 void foo() {
5     printf("hello from foo\n");
6 }
```

```
1 // main.c
2 #include "a.h"
3
4 int main() {
5     foo();
6 }
```

И как собрать такой проект?)

```
1 $> gcc main.c
2 $> ls
3 main.c a.h a.out
```

А если так?)

```
1 // main.c
2 #include "a.h"
3 #include "b.h"
4
5 int main() {
6     say_a();
7     say_b();
8 }
```

А если так?)

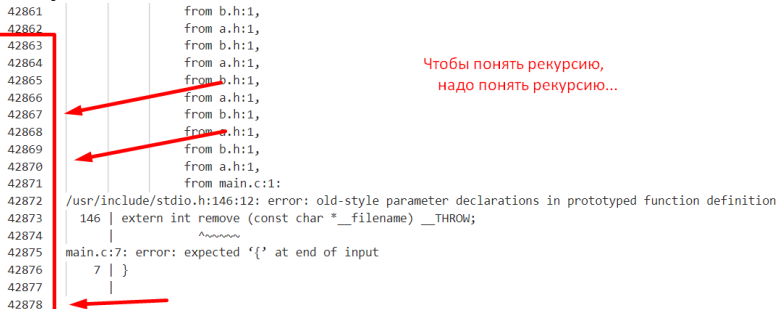
```
1 // a.h
2 #include "b.h"
3 #include <stdio.h>
4
5 void say_a() {
6     hidden_say_a();
7 }
8
9 void hidden_say_b() {
10     printf("hello, B!\n");
11 }
```

```
1 // b.h
2 #include "a.h"
3 #include <stdio.h>
4
5 void say_b() {
6     hidden_say_b();
7 }
8
9 void hidden_say_a() {
10     printf("hello, A!\n");
11 }
```

А если так?)

То будет ошибка

```
42861      from b.h:1,  
42862      from a.h:1,  
42863      from b.h:1,  
42864      from a.h:1,  
42865      from b.h:1,  
42866      from a.h:1,  
42867      from b.h:1,  
42868      from a.h:1,  
42869      from b.h:1,  
42870      from a.h:1,  
42871      from main.c:1:  
42872 /usr/include/stdio.h:146:12: error: old-style parameter declarations in prototyped function definition  
42873 146 | extern int remove (const char *__filename) __THROW;  
42874     |                   ~~~~~  
42875 main.c:7: error: expected '{' at end of input  
42876     7 | }  
42877     |  
42878
```



Чтобы понять рекурсию,
надо понять рекурсию...

Давайте подумаем, почему так

```
1 // a.h  
2 #include "b.h"  
3  
4 /* some code */
```

```
1 // b.h  
2 #include "a.h"  
3  
4 /* some code */
```


Давайте подумаем, почему так

```
1 // a.h
2 /* -- #include "b.h" --- */
3     #include "a.h"
4
5     /* BBB */
6 /* ----- */
7
8 /* AAA */
```

```
1 // b.h
2 /* -- #include "a.h" --- */
3     #include "b.h"
4
5     /* AAA */
6 /* ----- */
7
8 /* BBB */
```

Давайте попробуем это решить

Решение в том, чтоб собрать файлы ОТДЕЛЬНО
Но при этом нужно что-то подключать в `main.c`

Переделаем файлики (A)

```
1 // a.h
2 void say_a();
3
4 void hidden_say_b();
```

```
1 // a.c
2 #include "a.h"
3 #include "b.h"
4 #include <stdio.h>
5
6 void say_a() {
7     hidden_say_a();
8 }
9
10 void hidden_say_b() {
11     printf("hello, B!\n");
12 }
```

Переделаем файлики (В)

```
1 // b.h
2 void say_b();
3
4 void hidden_say_a();
```

```
1 // b.c
2 #include "b.h"
3 #include "a.h"
4 #include <stdio.h>
5
6 void say_b() {
7     hidden_say_b();
8 }
9
10 void hidden_say_a() {
11     printf("hello, A!\n");
12 }
```

Команды для сборки

```
1 $> gcc -c a.c
2 $> gcc -c b.c
3 $> gcc -c main.c
4 $> gcc a.o b.o main.o -o test
5 $> ./test
6 hello, A!
7 hello, B!
```

Круто, но есть проблемы

- Нужно по отдельности компилировать каждый файл
- Нужно помнить, какие файлы были изменены, чтоб не пересобирать все разом
- При изменении header-файлов необходимо пересобирать все файлы, где они используются
- ...

Можно не надо пожалуйста...

Эти проблемы превращаются в БОЛЬШИЕ ПРОБЛЕМЫ, когда проект становится нетривиальным.

Чтоб это упростить есть системы сборки

Makefile (bad version)

```
1 all:
2     gcc -c a.c
3     gcc -c b.c
4     gcc -c main.c
5     gcc *.o -o test
```

```
1 $> make
2 $> ./test
3 hello, A!
4 hello, B!
```

Makefile (better version)

```
1 all: test
2
3 test: main.o a.o b.o
4     gcc main.o a.o b.o -o test
5
6 a.o: a.h a.c
7     gcc -std=c11 -c a.c
8
9 b.o: b.h b.c
10    gcc -std=c11 -c b.c
11
12 main.o: a.h b.h main.c
13    gcc -std=c11 -c main.c
14
15 clean:
16    rm -rf *.o test
17
```

```
1 $> make
2 rm -rf *.o test
3
4 $> make
5 gcc -std=c11 -c main.c
6 gcc -std=c11 -c a.c
7 gcc -std=c11 -c b.c
8 gcc main.o a.o b.o -o test
9
10 $> ./test
11 hello, A!
12 hello, B!
```

Makefile (preultimate version)

```
1  SOME_VAR=123
2  COMPILER=gcc
3  CFLAGS=-std=c11
4  OUTPUT=test
5
6  all: $(OUTPUT)
7
8  $(OUTPUT): main.o a.o b.o
9          $(COMPILER) main.o a.o b.o -o $(OUTPUT)
10
11 a.o: a.h a.c
12     $(COMPILER) -std=c11 -c a.c
13
14 b.o: b.h b.c
15     $(COMPILER) -std=c11 -c b.c
16
17 main.o: a.h b.h main.c
18     $(COMPILER) -std=c11 -c main.c
19
20 clean:
21     rm -rf *.o $(OUTPUT)
```

Вопросики?)

А с чего начинается выполнение программы?)

```
1 // main.c
2 #include <stdio.h>
3
4 int main() {
5     printf("Hello, CAOS!");
6 }
```

Если так, то зачем так много строчек?

```
113  
114 0000000000001149 <main>:  
115     1149: f3 0f 1e fa      endbr64  
116     114d: 55              push   %rbp  
117     114e: 48 89 e5        mov    %rsp,%rbp  
118     1151: 8b 05 b9 2e 00 00 mov    0x2eb9(%rip),%eax    # 4010 <a>  
119     1157: 89 c6           mov    %eax,%esi  
120     1159: 48 8d 3d a4 0e 00 00 lea     0xea4(%rip),%rdi    # 2004 <_IO_stdin_used+0x4>  
121     1160: b8 00 00 00 00  mov    $0x0,%eax  
122     1165: e8 e6 fe ff ff   callq  1050 <printf@plt>  
123     116a: b8 00 00 00 00  mov    $0x0,%eax  
124     116f: 5d              pop    %rbp  
125     1170: c3              retq  
126     1171: 66 2e 0f 1f 84 00 00 nopw   %cs:0x0(%rax,%rax,1)  
127     1178: 00 00 00        nopl  
128     117b: 0f 1f 44 00 00  nopl   0x0(%rax,%rax,1)  
129
```

Объектный файл (целиком)

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

```
main.o:      file format elf64-x86-64
```

Disassembly of section .text:

0000000000000000 <main>:

0: f3 0f 1e fa

endbr64

4: 55

push %rbp

5: 48 89 e5

mov %rsp,%rbp

8: 8b 05 00 00 00 00

mov 0x0(%rip),%eax

e: 89 c6

mov %eax,%esi

10: 48 8d 3d 00 00 00 00

lea 0x0(%rip),%rdi

17: b8 00 00 00 00

mov \$0x0,%eax

1c: e8 00 00 00 00

callq 21 <main+0x21>

21: b8 00 00 00 00

mov \$0x0,%eax

26: 5d

pop %rbp

27: c3

retq

А вот функция, которая вызывает main

```
43
44 0000000000001060 <_start>:
45     1060: f3 0f 1e fa          endbr64
46     1064: 31 ed               xor    %ebp,%ebp
47     1066: 49 89 d1            mov    %rdx,%r9
48     1069: 5e                 pop    %rsi
49     106a: 48 89 e2            mov    %rsp,%rdx
50     106d: 48 83 e4 f0         and    $0xfffffffffffffff0,%rsp
51     1071: 50                 push   %rax
52     1072: 54                 push   %rsp
53     1073: 4c 8d 05 76 01 00 00 lea     0x176(%rip),%r8      # 11f0 <__libc_csu_fini>
54     107a: 48 8d 0d ff 00 00 00 lea     0xff(%rip),%rcx     # 1180 <__libc_csu_init>
55     1081: 48 8d 3d c1 00 00 00 lea     0xc1(%rip),%rdi     # 1149 <main>
56     1088: ff 15 52 2f 00 00   callq  *0x2f52(%rip)       # 3fe0 <__libc_start_main@GLIBC_2.2.5>
57     108e: f4                 hlt
58     108f: 90                 nop
59
```

А вот функция, которая вызывает main

```
43
44 0000000000001060 <_start>:
45     1060: f3 0f 1e fa          endbr64
46     1064: 31 ed               xor    %ebp,%ebp
47     1066: 49 89 d1            mov    %rdx,%r9
48     1069: 5e                 pop    %rsi
49     106a: 48 89 e2            mov    %rsp,%rdx
50     106d: 48 83 e4 f0         and    $0xfffffffffffffff0,%rsp
51     1071: 50                 push   %rax
52     1072: 54                 push   %rsp
53     1073: 4c 8d 05 76 01 00 00 lea     0x176(%rip),%r8      # 11f0 <__libc_csu_fini>
54     107a: 48 8d 0d ff 00 00 00 lea     0xff(%rip),%rcx     # 1180 <__libc_csu_init>
55     1081: 48 8d 3d c1 00 00 00 lea     0xc1(%rip),%rdi     # 1149 <main>
56     1088: ff 15 52 2f 00 00   callq  *0x2f52(%rip)       # 3fe0 <__libc_start_main@GLIBC_2.2.5>
57     108e: f4                 hlt
58     108f: 90                 nop
59
```

Внимание вопрос: А кто написал функцию `_start`? (И написана ли она вообще?)

А что если...

```
1 // main.c
2
3 void _start() {
4     // nop
5 }
```

Если это скомпилировать (попытаться), то будет это:

```
1 $> gcc main.c -o test
2 /usr/bin/ld: /tmp/cc35l1gr.o: in function `_start':
3 main.c:(.text+0x0): multiple definition of `_start'; /usr/lib/gcc/x86_64-
4 /usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../../x86_64-linux-gnu/
5 (.text+0x24): undefined reference to `main'
6 collect2: error: ld returned 1 exit status
```

Теперь, когда мы знаем правду, давайте попробуем

```
1 void _start() {  
2     int a = 10;  
3     int b = 20;  
4     int c = a + b;  
5 }
```

```
1 $> gcc -nostdlib main.c -o test  
2 $> objdump -d test  
3 ...  
4 0000000000001000 <_start>:  
5      1000:      f3 0f 1e fa                endbr64  
6      1004:      55                          push    %rbp  
7      1005:      48 89 e5                    mov     %rsp,%rbp  
8      1008:      c7 45 fc 0a 00 00 00        movl    $0xa,-0x4(%rbp)  
9      100f:      c7 45 f8 14 00 00 00        movl    $0x14,-0x8(%rbp)  
10     1016:      8b 55 fc                    mov     -0x4(%rbp),%edx  
11     1019:      8b 45 f8                    mov     -0x8(%rbp),%eax  
12     101c:      01 d0                      add     %edx,%eax  
13     101e:      89 45 f4                    mov     %eax,-0xc(%rbp)  
14     1021:      90                          nop  
15     1022:      5d                          pop     %rbp  
16     1023:      c3                          retq
```

Можно (без stdlib) написать hello world

```
1  .global _start
2
3  .text
4  _start:
5      mov     $1, %rax          # system call 1 is write
6      mov     $1, %rdi          # file handle 1 is stdout
7      mov     $message, %rsi    # address of string to output
8      mov     $13, %rdx         # number of bytes
9      syscall                  # invoke operating system to do the write
10
11     mov     $60, %rax          # system call 60 is exit
12     xor     %rdi, %rdi        # we want return code 0
13     syscall                  # invoke operating system to exit
14
15 message:
16     .ascii  "Hello, world\n"
```

А можно и на С (ну почти)

```
1  .global say_hello
2  .global my_exit
3
4  .text
5  say_hello:
6      mov     $1, %rax           # system call 1 is write
7      mov     $1, %rdi           # file handle 1 is stdout
8      mov     $message, %rsi     # address of string to output
9      mov     $13, %rdx          # number of bytes
10     syscall                    # invoke operating system to do the write
11     ret
12
13 message:
14     .ascii  "Hello, world\n"
15
16 my_exit:
17     mov     $60, %rax           # system call 60 is exit
18     xor     %rdi, %rdi          # we want return code 0
19     syscall                    # invoke operating system to exit
```

А можно и на С (ну почти)

```
1 extern void say_hello(void);
2 extern void my_exit(void);
3
4 int _start()
5 {
6     say_hello();
7     my_exit();
8 }
```

```
1 all: main.o test.o
2     ld *.o
3
4 main.o: main.c
5     gcc -c main.c
6
7 test.o: test.s
8     gcc -c test.s
9
10 clean:
11     rm -rf *.o a.out
```

ПРИМЕЧАНИЕ! НЕ СКОМПИЛИРУЕТСЯ!

```
1  #include <unistd.h>
2  #include <sys/syscall.h>
3
4  extern long syscall(long number, ...);
5
6  void _start() {
7      const char Hello[] = "Hello, World!";
8      syscall(SYS_write, 1, Hello, sizeof(Hello) - 1);
9      syscall(SYS_exit, 0);
10 }
```

Вопросы?)

Thx

Спасибо! На этом всё)