Seminar 9: Compiling and loading programs

Calling assembly code from C (25 minutes)

The `string.asm` file contains the `print_string` and `world` functions:

```
global world
global print_string

section .rodata
message: db ", world!", 10, 0

section .text

string_length:
  mov rax, 0
    .loop:
      xor rax, rax
    .count:
      cmp byte [rdi+rax], 0
      je .end
      inc rax
      jmp .count
    .end:
      ret

world:
    mov rdi, message

print_string:
    push rax
    push rdi
    call string_length
    mov rsi, [rsp]
    mov rdx, rax
    mov rax, 1
    mov rdi, 1
    push rcx
    syscall
    pop rcx
    pop rdi
    pop rax
    ret
```

Question 1. What does the `world` function do? Explain how it works (i.e. explain why after calling `world` the “world” word was printed).

The `hello.c` file contains the `main` function, which calls `print_string` and prints “hello” with it in this way:

```
int main() {
    print_string("hello");
    world();
    return 0;
}
```

Function `main` calls `world` as well.

Question These files are missing a few lines so they can't interact with each other's code. Add missing lines to the files so that the `print_string` and `hello` functions are called and check the result. Hint: remember what it takes to call another file code from one C code file.

Question 2. What are the sections `.rodata` and `.bss`? What are they used for?

Question 3. What do the `resb`, `resq` directives do in nasm (use Baidu)?

Question 4. In the `hello.c` file, in which section will the line = "hello" = be placed (use objdump -D)?

ELF-files (25 minutes)

In the previous step, during the compilation process, we received the files:

- `string.o` from assembler file `string.asm`;
- `hello.o` from a file with C code `hello.c`;
- `hello`, executable file.

Let's examine these files in more detail using `readelf`. To facilitate the research process, record all results.

The ELF file has three headers:

File header
: main header, contains general information about the file and links to the other two headers.

Section header
: a list of sections (the same ones that are in assembler, and many others for service purposes)

Program header
: list of *segments*. Each segment describes a memory region into which one or more sections will be loaded.

Question Run `readelf` with no arguments. Read the output and determine what keys are needed to display the three file headers. Insert them to the form.

Question Print the file header for the `hello.o` file. What is Entry point address and why is it 0?

Question Output the program header for the `hello.o` file. Explain the result.

Question Define the addresses for the `.text` and `.rodata` sections.

Question 5. Output the program header for `hello`. What segments do `.text` and `.rodata` fall into? What are the addresses of these segments?

Loading the file (20 minutes)

Modify your C program `hello.c` so that it goes into an infinite loop.

Question 6. Display the memory regions map (see seminar 7) for the running program we are working with today. Map the segments containing the `.rodata` and `.text` sections to memory regions. Is it true that only one segment corresponds to one region of memory?

Question Which regions correspond to the parts of the dynamic link libraries? Dynamic link libraries in Linux have the `*.so` extension.

Question Run `ldd hello`. Explain the result.

Question 7. Why do you think the C standard library is implemented as a dynamic library rather than being included statically in the executable file like other `.o` files?