Университет ИТМО
Кафедра ВТ

Лабораторная работа №2
Низкоуровневое программирование

Выполнил: Федоров Сергей
Группа: P33113

Санкт-Петербург
2020 г.

Задание лабораторной работы:

Задача стояла в том чтобы реализовать простенькую I/O библиотеку с следующими функциями:

Выполнение:

Files structure:

```
lab2/
├── colon.inc
├── dict.asm
├── lib.asm
├── main.asm
├── Makefile
└── words.inc
```

colon.inc:

```
%define last_address 0
%macro colon 2
%2:
  dq last_address
  db %1, 0
%define last_address %2
%endmacro
```

dict.asm:

```
global find_word
extern string_equals
section .text
; args: rdi = key string pointer, rsi = list first elem → returns: rax = address
(found) or rax = 0 (not found)
find_word:
    mov r8, rdi
    .loop:
        mov r9, rsi
        test rsi, rsi                   ; Check if dictionary is empty (rsi = 0)
        je .exit_not_found
        mov rdi, r8
        add rsi, 8
        call string_equals          ; Compare strings
        mov rsi, r9
        test rax, rax               ; If equals then exit
        jnz .exit_found
        mov rsi, [rsi]                  ; If not then next pair
        jmp .loop
    .exit_found:
        mov rax, rsi
        ret
    .exit_not_found:
        mov rax, 0
        ret
```

lib.asm: Во имя сохранения места и не повторения уже показанного: https://is.gd/8FWKQr

words.inc:

```
%include "colon.inc"
section .data
colon "Limbo", limbo
db "First circle of HELL", 0
colon "Lust", lust
db "Second circle of HELL", 0
colon "Gluttony", gluttony
db "Third circle of HELL", 0
colon "Greed", greed
db "Fourth circle of HELL", 0
colon "Wraith", wraith
db "Fifth circle of HELL", 0
colon "Heresy", heresy
db "Sixth circle of HELL", 0
colon "Violence", violence
db "Seventh circle of HELL", 0
colon "Fraud", fraud
db "Eighth circle of HELL", 0
colon "Treachery", treachery
db "Ninth circle of HELL", 0
```

main.asm:

```
%include "words.inc"

%define DICT_START last_address
%define KEY_MAX_SIZE 255
%define STDOUT 1
%define STDERR 2

global _start
extern read_word
extern find_word
extern string_length
extern print_string
extern print_newline
extern exit

section .data
    reading_error_msg: db "Error: Reading problem", 0
   not_found_msg: db "Error: Non-Existent key", 0

section .text
_start:
   mov rsi, KEY_MAX_SIZE
   sub rsp, KEY_MAX_SIZE
   mov rdi, rsp
   call readc_word          ; Read key into rax
   test rax, rax        ; Check if word is null or error?
   je .reading_error_exit
   mov rdi, rax
   mov rsi, DICT_START
   call find_word       ; Check dictionary for key
   test rax, rax
```

```
    je .not_found_exit      ; Check if no such key
    add rax, 8              ; Set accumulator as a key pointer
    mov r10, rax            ; Save rax in r10
    mov rdi, rax            ; Provide string address for string_length
    call string_length      ; calc key length
    inc r10                 ; Skipping key string
    add r10, rax          ; To access value string
    mov rdi, r10            ; Printing value string
    mov r8, STDOUT
    call print_string
    jmp .exit
.reading_error_exit:
    mov rdi, reading_error_msg
    call string_length
    mov rsi, rax
     jmp .error_exit
.not_found_exit:
    mov rdi, not_found_msg
    call string_length
    mov rsi, rax
    jmp .error_exit
.error_exit:
    mov r8, STDERR
    call print_string
    jmp .exit
.exit:
     call print_newline
     mov rax, 60
     mov rdi, rax
     syscall
     ret
```

Вывод:

Что мы в данной работе сделали такого, чего не делали до этого:
1. Познакомились с/Использовали макросы для выполнения части вычислений еще в **compile-time**.
2. Познакомились с/Поняли **compilation-pipeline**. Поняли зачем существует препроцессинг, транслирование, линковка. И почему нельзя было объеденить это все в один процесс (точнее не почему нельзя, а скорее почему не стоит).
3. Познакомились с/Использовали утилиту сборки **make**.

Пару тезисов которые приходят на ум:
1. Использование макросов усложняет программу, заставляет больше думать. А если начинать еще думать о том как происходит само «**макрорасширение**» и в каком порядко подставляются значения препроцессором, то думать приходится еще больше.
2. Область применения макросов достаточно сильно ограничена, за что того что мы можем оперировать только данными, которые у нас доступны на момент **compile-time**, следовательно только константы и прочие не задаваемые пользователем значения.
3. Из-за разделения компилятора **nasm** и линкера **ld**, приходится писать **Makefile**, и учитывая что это еще один синтаксис который нужно понять, становится еще сложнее. Однако в конце возможность получения исполняемого файла путем «**make all clean**», весьма оправдывает старания.
4. Даже не смотря на то что это уже вторая лаба и была она меньше по обьем кода, писать на **asm**, все равно достаточно сложная и неприятная задача.