Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Николич С.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 26.11.25

Москва, 2025

Постановка задачи

Вариант 32.

Цель работы:

- Создание динамических библиотек
- Создание программ, которые используют функции динамических библиотек

Задание:

- Требуется создать динамические библиотеки, которые реализуют заданный вариантом функционал. Далее использовать данные библиотеки 2-мя способами:

1. Во время компиляции (на этапе линковки/linking)
2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками

В конечном итоге, в лабораторной работе необходимо получить следующие части:

- Динамические библиотеки, реализующие контракты, которые заданы вариантом;
- Тестовая программа (программа №1), которая используют одну из библиотек, используя информацию полученные на этапе компиляции;
- Тестовая программа (программа №2), которая загружает библиотеки, используя только их относительные пути и контракты.

Провести анализ двух типов использования библиотек.

Пользовательский ввод для обоих программ должен быть организован следующим образом:

- Если пользователь вводит команду «0», то программа переключает одну реализацию контрактов на другую (необходимо только для программы №2). Можно реализовать лабораторную работу без данной функции, но максимальная оценка в этом случае будет «хорошо»;
- “1 arg1 arg2 … argN”, где после “1” идут аргументы для первой функции, предусмотренной контрактами. После ввода команды происходит вызов первой функции, и на экране появляется результат ее выполнения;
- “2 arg1 arg2 … argM”, где после “2” идут аргументы для второй функции, предусмотренной контрактами. После ввода команды происходит вызов второй функции, и на экране появляется результат ее выполнения.

Контракты и реализации функций

Функция 1. 6. Расчет значения числа e (основание натурального логарифма):
Сигнатура функции: float e(int x);

- Реализация №1: (1 + 1 / x)^x
- Реализация №2: Сумма ряда по n от 0 до x, где элементы ряда равны: (1 / (n!))

Функция 2. 8. Перевод числа x из десятичной системы счисления в другую:
Сигнатура функции: char *convert(int x);

- Реализация №1: Перевод в двоичную
- Реализация №2: Перевод в троичную

Общий метод и алгоритм решения

Использованные системные вызовы:

- void *dlopen(const char *filename, int flags) – открывает динамическую библиотеку. Флаги RTLD_LAZY | RTLD_LOCAL обеспечивают ленивую загрузку символов и локальную видимость.
- void *dlsym(void *handle, const char *symbol) – получает адрес функции из загруженной библиотеки по её имени.
- int dlclose(void *handle) – закрывает загруженную динамическую библиотеку и освобождает ресурсы.
- char *dlerror(void) – возвращает строку с описанием последней ошибки при работе с динамическими библиотеками.
- ssize_t read(int fd, void *buf, size_t count) – читает данные из файлового дескриптора. Используется для чтения пользовательского ввода из STDIN_FILENO.
- ssize_t write(int fd, const void *buf, size_t count) – записывает данные в файловый дескриптор. Используется для вывода результатов в STDOUT_FILENO и ошибок в STDERR_FILENO.
- char *strtok(char *str, const char *delim) – разбивает строку на токены по указанным разделителям.
- int strcmp(const char *s1, const char *s2) – сравнивает две строки лексикографически.
- size_t strlen(const char *s) – вычисляет длину строки.
- char *strcpy(char *dest, const char *src) – копирует строку из источника в назначение.
- char *strcat(char *dest, const char *src) – добавляет строку источника к строке назначения.
- void *malloc(size_t size) – выделяет блок памяти указанного размера.
- void free(void *ptr) – освобождает ранее выделенный блок памяти.
- float strtof(const char *nptr, char **endptr) – преобразует строку в число с плавающей точкой.
- long int strtol(const char *nptr, char **endptr, int base) – преобразует строку в целое число в указанной системе счисления.

Лабораторная работа реализует два способа использования динамических библиотек: статическую линковку и динамическую загрузку.

Программа №1 использует статическую линковку с библиотекой library1.so. Программа предоставляет интерфейс для вычисления числа e и конвертации чисел в двоичную систему. Пользователь вводит команды:

- "0" - информация о реализации
- "1 x" - вычисление e(x)
- "2 x" - конвертация числа x

Программа №2 использует динамическую загрузку библиотек library1.so и library2.so через интерфейс dlopen/dlsym. При вводе команды "0" программа переключается между реализациями. Для управления библиотеками используются функции dlopen(), dlsym() и dlclose().

Обе программы используют низкоуровневый ввод-вывод через системные вызовы read() и write(). Реализованы вспомогательные функции для парсинга ввода и форматирования вывода.

Алгоритмы реализаций:

Функция e(x):

- Реализация №1: вычисление по формуле (1 + 1/x)^x
- Реализация №2: вычисление как суммы ряда 1/n! от n=0 до x

Функция convert(x):

- Реализация №1: перевод в двоичную систему
- Реализация №2: перевод в троичную систему

Оба алгоритма конвертации используют деление с остатком и динамическое выделение памяти под результирующую строку.

Код программы

Library.h

```h
#ifndef __LIBRARY_H
#define __LIBRARY_H

typedef float e_func(int x);
typedef char* convert_func(int x);

#endif
```

Library1.c

```c
#include "library.h"
#include <stdlib.h>

// макрос для кроссплатформенности
// В Windows нужен __declspec(dllexport) для экспорта функций из DLL
// В Linux/Unix функции экспортируются по умолчанию
// макрос EXPORT - видим функции вне библиотеки
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

// Реализация №1: e = (1 + 1/x)^x
EXPORT float e(int x) {
    if (x <= 0) return 0.0f;
    float temp = 1.0f + 1.0f / (float)x;
    float result = 1.0f;
```

```
    // Возведение в степень через умножение
    for (int i = 0; i < x; i++) {
        result *= temp;
    }
    return result;
}

// Реализация №1: Перевод в двоичную систему
EXPORT char* convert(int x) {
    if (x == 0) {
        char* result = malloc(2);
        result[0] = '0';
        result[1] = '\0';
        return result;
    }

    // Определяем длину результата
    int temp = abs(x);
    int length = 0;
    while (temp > 0) {
        temp /= 2;
        length++;
    }

    // Добавляем место для знака и нуль-терминатора
    if (x < 0) length++;
    char* result = malloc(length + 1);
    result[length] = '\0';

    // Заполняем строку с конца
    temp = abs(x);
    int index = length - 1;
    while (temp > 0) {
        result[index--] = (temp % 2) + '0';
        temp /= 2;
    }

    // Добавляем знак если нужно
    if (x < 0) {
        result[0] = '-';
    }

    return result;
}
```

Library2.c

```
#include "library.h"

#include <stdlib.h>
```

```c
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

// Реализация №2: e = сумма ряда 1/n! от n=0 до x
EXPORT float e(int x) {
    if (x < 0) return 0.0f;

    float result = 0.0f;
    float factorial = 1.0f;

    for (int n = 0; n <= x; n++) {
        if (n > 0) {
            factorial *= n;
        }
        result += 1.0f / factorial;
    }
    return result;
}

// Реализация №2: Перевод в троичную систему
EXPORT char* convert(int x) {
    if (x == 0) {
        char* result = malloc(2);
        result[0] = '0';
        result[1] = '\0';
        return result;
    }

    int temp = abs(x);
```

```
        int length = 0;
        while (temp > 0) {
            temp /= 3;
            length++;
        }

        if (x < 0) length++;
        char* result = malloc(length + 1);
        result[length] = '\0';

        temp = abs(x);
        int index = length - 1;
        while (temp > 0) {
            result[index--] = (temp % 3) + '0';
            temp /= 3;
        }

        if (x < 0) {
            result[0] = '-';
        }

        return result;
    }
```

Program1.c

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>// для read(), write(), STDIN_FILENO, STDOUT_FILENO

// Функции из библиотеки (линкуются на этапе компиляции)
// Компоновщик найдет эти функции в library1.so
extern float e(int x);
extern char* convert(int x);

void print(const char* msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}
```

```c
void print_error(const char* msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

char* int_to_string(int value) {
    if (value == 0) {
        char* result = malloc(2);
        result[0] = '0';
        result[1] = '\0';
        return result;
    }

    int is_negative = value < 0;
    unsigned int num = is_negative ? -value : value;
    int length = 0;
    unsigned int temp = num;

    while (temp > 0) {
        temp /= 10;
        length++;
    }

    if (is_negative) length++;
    char* result = malloc(length + 1);
    result[length] = '\0';

    temp = num;
    int index = length - 1;
    while (temp > 0) {
        result[index--] = (temp % 10) + '0';
        temp /= 10;
    }

    if (is_negative) {
        result[0] = '-';
    }

    return result;
}

// Конвертация float в строку (упрощенная)
char* float_to_string(float value) {
    // Целая часть
    int int_part = (int)value;
    // Дробная часть (2 знака)
    int frac_part = (int)((value - int_part) * 100);
    if (frac_part < 0) frac_part = -frac_part;

    char* int_str = int_to_string(int_part);
    char* result = malloc(strlen(int_str) + 5); // целая часть + точка + 2 цифры + '\0'

    strcpy(result, int_str);
    strcat(result, ".");
```

```
    // Добавляем дробную часть
    if (frac_part < 10) {
        strcat(result, "0");
    }
    char frac_str[3];
    frac_str[0] = (frac_part / 10) + '0';
    frac_str[1] = (frac_part % 10) + '0';
    frac_str[2] = '\0';
    strcat(result, frac_str);

    free(int_str);
    return result;
}

//Замена atoi
int parse_int(const char* str, int* result) {
    if (str == NULL || *str == '\0') return 0;

    int value = 0;
    int sign = 1;
    int i = 0;

    if (str[0] == '-') {
        sign = -1;
        i = 1;
    } else if (str[0] == '+') {
        i = 1;
    }

    // Парсинг цифр
    for (; str[i] != '\0'; i++) {
        if (str[i] >= '0' && str[i] <= '9') {
            value = value * 10 + (str[i] - '0');
        } else {
            return 0; // Не цифра
        }
    }

    *result = value * sign;
    return 1;
}

int main() {
    print("Программа №1 - Статическая линковка\n");
    print("Доступные команды:\n");
    print("0 - информация о реализации\n");
    print("1 x - вычисление числа e для x\n");
    print("2 x - конвертация числа x\n");
    print("exit - выход из программы\n\n");

    char buffer[256];
    char* token;
```

```
while (1) {
    print("> ");

    // Чтение ввода
    ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
    if (bytes <= 0) break;

    buffer[bytes] = '\0';

    // Удаление символа новой строки
    if (bytes > 0 && buffer[bytes - 1] == '\n'){
        buffer[bytes - 1] = '\0';
    }

    // Парсинг команды
    token = strtok(buffer, " ");
    if (token == NULL) continue;

    if (strcmp(token, "exit") == 0){
        break;
    }
    else if (strcmp(token, "0") == 0){
        print("Используется реализация №1 (статическая линковка)\n");
        print("e(x) = (1 + 1/x)^x\n");
        print("convert(x) = перевод в двоичную систему\n");
    }
    else if (strcmp(token, "1") == 0){
        token = strtok(NULL, " ");
        if (token == NULL) {
            print_error("error: missing argument for command '1'\n");
            continue;
        }

        int x;
        if (!parse_int(token, &x)) {
            print_error("error: invalid integer argument\n");
            continue;
        }

        float result = e(x);
        char* result_str = float_to_string(result);

        char* x_str = int_to_string(x);
        char output[256];
        strcpy(output, "e(");
        strcat(output, x_str);
        strcat(output, ") = ");
        strcat(output, result_str);
        strcat(output, "\n");

        print(output);

        free(x_str);
        free(result_str);
```

```
        }
        else if (strcmp(token, "2") == 0){
            token = strtok(NULL, " ");
            if (token == NULL) {
                print_error("error: missing argument for command '2'\n");
                continue;
            }

            int x;
            if (!parse_int(token, &x)){
                print_error("error: invalid integer argument\n");
                continue;
            }

            char* result = convert(x);

            char* x_str = int_to_string(x);
            char output[256];
            strcpy(output, "convert(");
            strcat(output, x_str);
            strcat(output, ") = ");
            strcat(output, result);
            strcat(output, "\n");

            print(output);

            free(x_str);
            free(result);
        }
        else {
            print_error("error: unknown command\n");
        }
    }

    return 0;
}
```

Program2.c

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <dlfcn.h> // Для динамической подгрузки библиотек

typedef float e_func(int x);
typedef char* convert_func(int x);

// Структура для библиотеки
typedef struct {
    void* handle;// Указатель на загруженную библиотеку
    e_func* e_ptr;// Указатель на функцию e
    convert_func* convert_ptr;// Указатель на функцию convert
    const char* description;// Описание реализации
} Library;
```

```c
void print(const char* msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}

void print_error(const char* msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

// Заглушки для функций - вызывается если библиотека не загружена
static float e_stub(int x) {
    print_error("error: function e not loaded\n");
    return 0.0f;
}

static char* convert_stub(int x) {
    print_error("error: function convert not loaded\n");
    char* result = malloc(2);
    result[0] = '0';
    result[1] = '\0';
    return result;
}

// Конвертация чисел в строки (аналогично program1.c)
char* int_to_string(int value) {
    if (value == 0) {
        char* result = malloc(2);
        result[0] = '0';
        result[1] = '\0';
        return result;
    }

    int is_negative = value < 0;
    unsigned int num = is_negative ? -value : value;
    int length = 0;
    unsigned int temp = num;

    while (temp > 0) {
        temp /= 10;
        length++;
    }

    if (is_negative) length++;
    char* result = malloc(length + 1);
    result[length] = '\0';

    temp = num;
    int index = length - 1;
    while (temp > 0) {
        result[index--] = (temp % 10) + '0';
        temp /= 10;
    }

    if (is_negative) {
```

```
        result[0] = '-';
    }

    return result;
}

char* float_to_string(float value) {
    int int_part = (int)value;
    int frac_part = (int)((value - int_part) * 100);
    if (frac_part < 0) frac_part = -frac_part;

    char* int_str = int_to_string(int_part);
    char* result = malloc(strlen(int_str) + 5);

    strcpy(result, int_str);
    strcat(result, ".");

    if (frac_part < 10) {
        strcat(result, "0");
    }
    char frac_str[3];
    frac_str[0] = (frac_part / 10) + '0';
    frac_str[1] = (frac_part % 10) + '0';
    frac_str[2] = '\0';
    strcat(result, frac_str);

    free(int_str);
    return result;
}

int parse_int(const char* str, int* result) {
    if (str == NULL || *str == '\0') return 0;

    int value = 0;
    int sign = 1;
    int i = 0;

    if (str[0] == '-') {
        sign = -1;
        i = 1;
    } else if (str[0] == '+') {
        i = 1;
    }

    for (; str[i] != '\0'; i++) {
        if (str[i] >= '0' && str[i] <= '9') {
            value = value * 10 + (str[i] - '0');
        } else {
            return 0;
        }
    }

    *result = value * sign;
    return 1;
```

```
}

int load_library(Library* lib, const char* path){
    //Загрузка библиотеки в память
    lib->handle = dlopen(path, RTLD_LAZY);// RTLD_LAZY - ленивая загрузка символов
    if (!lib->handle) {
        return 0;
    }

    lib->e_ptr = (e_func*)dlsym(lib->handle, "e");// Получение указателей на функции
    lib->convert_ptr = (convert_func*)dlsym(lib->handle, "convert");
    // Проверяем что обе функции найдены
    if (!lib->e_ptr || !lib->convert_ptr) {
        dlclose(lib->handle);
        return 0;
    }

    return 1;
}

void print_library_info(int index, const Library* lib){
    char num_str[2];
    num_str[0] = '1' + index;
    num_str[1] = '\0';

    print("Текущая реализация: ");
    print(num_str);
    print("\n");
    print(lib->description);
    print("\n\n");
}

int main() {
    Library libs[2]; //Массив для двух библиотек
    int current = 0; //Индекс текущей библиотеки

    // Инициализация библиотек
    libs[0].handle = NULL;
    libs[0].e_ptr = e_stub;
    libs[0].convert_ptr = convert_stub;
    libs[0].description = "e(x) = (1 + 1/x)^x\nconvert(x) = binary";

    libs[1].handle = NULL;
    libs[1].e_ptr = e_stub;
    libs[1].convert_ptr = convert_stub;
    libs[1].description = "e(x) = sum(1/n!) from n=0 to x\nconvert(x) = ternary";

    // Загрузка первой библиотеки по умолчанию
    if (!load_library(&libs[0], "./library1.so")) {
        print_error("error: failed to load library1.so\n");
    }

    print("Программа №2 - Динамическая загрузка\n");
    print("Доступные команды:\n");
```

```
print("0 - переключение реализации\n");
print("1 x - вычисление числа e для x\n");
print("2 x - конвертация числа x\n");
print("exit - выход из программы\n\n");

print_library_info(current, &libs[current]);

char buffer[256];
char* token;

while (1) {
    print("> ");

    ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
    if (bytes <= 0) break;

    buffer[bytes] = '\0';

    if (bytes > 0 && buffer[bytes - 1] == '\n') {
        buffer[bytes - 1] = '\0';
    }

    token = strtok(buffer, " ");
    if (token == NULL) continue;

    if (strcmp(token, "exit") == 0) {
        break;
    }
    else if (strcmp(token, "0") == 0) {
        current = (current + 1) % 2;

        // Пытаемся загрузить библиотеку если она еще не загружена
        if (!libs[current].handle) {
            const char* path = (current == 0) ? "./library1.so" : "./library2.so";
            if (!load_library(&libs[current], path)) {
                print_error("error: failed to load library\n");
                libs[current].e_ptr = e_stub;
                libs[current].convert_ptr = convert_stub;
            }
        }

        print_library_info(current, &libs[current]);
    }
    else if (strcmp(token, "1") == 0) {
        token = strtok(NULL, " ");
        if (token == NULL) {
            print_error("error: missing argument for command '1'\n");
            continue;
        }

        int x;
        if (!parse_int(token, &x)) {
            print_error("error: invalid integer argument\n");
            continue;
```

```
        }
        //Вызов функции через указатель
        float result = libs[current].e_ptr(x);

        char* result_str = float_to_string(result);
        char* x_str = int_to_string(x);

        char output[256];
        strcpy(output, "e(");
        strcat(output, x_str);
        strcat(output, ") = ");
        strcat(output, result_str);
        strcat(output, "\n");

        print(output);

        free(x_str);
        free(result_str);
    }
    else if (strcmp(token, "2") == 0) {
        token = strtok(NULL, " ");
        if (token == NULL) {
            print_error("error: missing argument for command '2'\n");
            continue;
        }

        int x;
        if (!parse_int(token, &x)) {
            print_error("error: invalid integer argument\n");
            continue;
        }

        char* result = libs[current].convert_ptr(x);
        char* x_str = int_to_string(x);

        char output[256];
        strcpy(output, "convert(");
        strcat(output, x_str);
        strcat(output, ") = ");
        strcat(output, result);
        strcat(output, "\n");

        print(output);

        free(x_str);
        free(result);
    }
    else {
        print_error("error: unknown command\n");
    }
}

// Закрытие библиотек
for (int i = 0; i < 2; i++) {
```

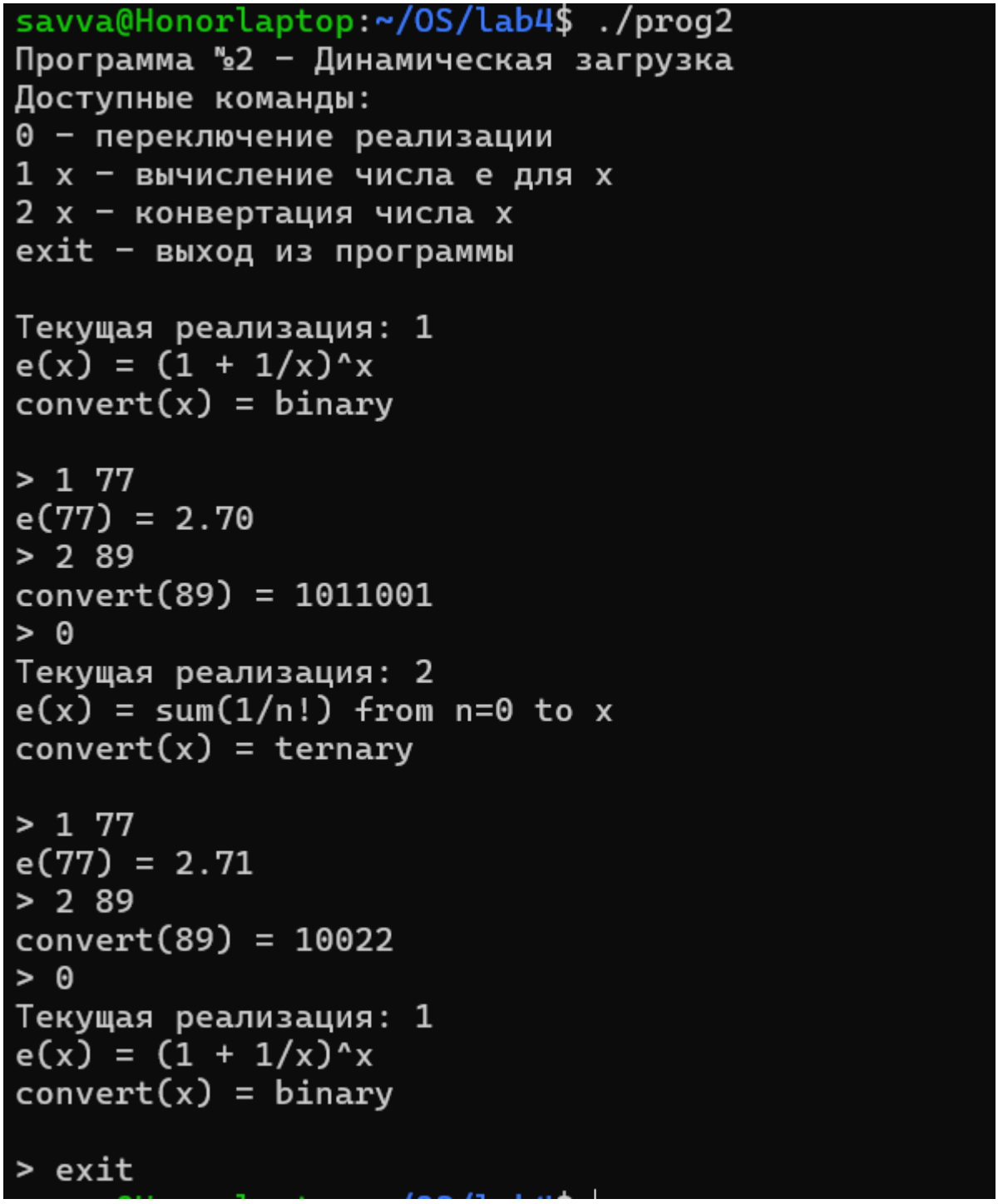
```
        if (libs[i].handle) {
            dlclose(libs[i].handle);
        }
    }

    return 0;
}
```

Протокол работы программы

```
savva@Honorlaptop:~/OS/lab4$ ./prog1
Программа №1 - Статическая линковка
Доступные команды:
0 - информация о реализации
1 x - вычисление числа e для x
2 x - конвертация числа x
exit - выход из программы

> 1 56
e(56) = 2.69
> 2 256
convert(256) = 100000000
> 0
Используется реализация №1 (статическая линковка)
e(x) = (1 + 1/x)^x
convert(x) = перевод в двоичную систему
> 0
Используется реализация №1 (статическая линковка)
e(x) = (1 + 1/x)^x
convert(x) = перевод в двоичную систему
> exit
```



Вывод

В ходе работы я освоил создание и использование динамических библиотек двумя способами. Научился компилировать библиотеки с флагами -fPIC и -shared, организовывать единые интерфейсы функций и работать с системными вызовами dlopen/dlsym для динамической загрузки.

Реализовал переключение между разными реализациями математических функций без перекомпиляции программы. Освоил низкоуровневый ввод-вывод через read/write и алгоритмы преобразования систем счисления.

Полученные навыки позволяют создавать гибкие приложения с модульной архитектурой и эффективно управлять библиотеками в операционной системе.