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: Губеев Д.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 25.11.25

Москва, 2025

Постановка задачи

Вариант 13.

Функция 1: **Расчет производной функции cos(x) в точке a с приращением dx:**
Сигнатура функции: float cos_derivative(float a, float dx);
- Реализация №1: f’(x) = (f(a + dx) - f(a)) / dx
- Реализация №2: f’(x) = (f(a + dx) - f(a - dx)) / (2dx)

Функция 2: **Подсчет площади плоской геометрической фигуры по двум сторонам:**
Сигнатура функции: float area(float a, float b);
- Реализация №1: Фигура прямоугольник
- Реализация №2: Фигура прямоугольный треугольник

Общий метод и алгоритм решения

Использованные системные вызовы:

- void* dlopen(const char filename, int flag); - загружает динамическую библиотеку в память и возвращает дескриптор. При ошибке возвращает NULL.
- void dlsym(void *handle, const char *symbol); - ищет адрес функции или переменной в загруженной библиотеке. Возвращает указатель на найденный символ либо NULL при ошибке.
- int dlclose(void handle); - закрывает загруженную динамическую библиотеку, уменьшает счётчик ссылок и освобождает ресурсы. Возвращает 0 при успехе, ненулевое значение при ошибке.

Программа 1 статически привязывается к одной из библиотек на этапе компиляции и получает доступ к её функциям напрямую. Пользователь вводит команды, после чего программа вызывает соответствующие функции вычисления производной или площади из подключённой библиотеки. После обработки всех команд программа завершается, не меняя реализацию функций во время работы.

Программа 2 загружает две динамические библиотеки с различными реализациями функций, используя механизм dlopen. Затем она получает адреса нужных функций с помощью dlsym, чтобы выполнять вычисления производной и площади без привязки к конкретной реализации. Далее программа принимает команды пользователя, вызывая соответствующие функции выбранной библиотеки. Предусмотрено переключение текущей реализации по команде, позволяющее выбрать другую библиотеку без перезапуска программы. После завершения работы все библиотеки корректно закрываются с помощью dlclose, а программа завершается.

Код программы

library_1.c

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

// NOTE: MSVC compiler does not export symbols unless annotated
```

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

EXPORT float cos_derivative(float a, float dx) {
    if (dx == 0.0f) {
        return NAN;
    }
    return (cosf(a + dx) - cosf(a)) / dx;
}
EXPORT float area(float a, float b) {
    return a * b;
}
```

library_2.c

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

// NOTE: MSVC compiler does not export symbols unless annotated
#ifdef _MSC_VER
#define EXPORT __declspec(dllexport)
#else
#define EXPORT
#endif

EXPORT float cos_derivative(float a, float dx) {
    if (dx == 0.0f) {
        return NAN;
    }
    return (cosf(a + dx) - cosf(a - dx)) / (2.0f * dx);
}
EXPORT float area(float a, float b) {
    return 0.5f * a * b;
}
```

library.h

```c
#ifndef __LIBRARY_1_H
#define __LIBRARY_1_H

float cos_derivative(float a, float dx);
float area(float a, float b);

#endif // __LIBRARY_1_H
```

prog1.c

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

```c
#include <unistd.h>

void print(char msg[]) { write(STDOUT_FILENO, msg, strlen(msg)); }

void print_error(char msg[]) { write(STDERR_FILENO, msg, strlen(msg)); }

int read_float(char **token, char *arg_name, char **endptr, float *out_value) {
    *token = strtok(NULL, " ");
    if (*token == NULL) {
        char output[256];
        snprintf(output, sizeof(output), "error: missing argument '%s'\n",
                 arg_name);
        print_error(output);
        return 0;
    }
    float value = strtof(*token, endptr);
    if (*endptr == *token) {
        char output[256];
        snprintf(output, sizeof(output), "error: invalid argument '%s'\n",
                 arg_name);
        print_error(output);
        return 0;
    }
    *out_value = value;
    return 1;
}

int main() {
    print("Программа 1 (линковка на этапе компиляции)\n");
    print("Используется реализация 1:\n");
    print("  cos_derivative(a, dx) = (cos(a+dx) - cos(a)) / dx\n");
    print("  area(a, b) - площадь прямоугольника\n\n");
    print("Формат ввода:\n");
    print("  1 a dx  -> вычислить производную cos(x) в точке a с шагом dx\n");
    print("  2 a b   -> вычислить площадь фигуры по сторонам a и b\n");
    print("Ctrl+D для выхода.\n\n");

    while (1) {
        char buffer[1024];
        ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
        if (bytes == -1) {
            print_error("error: failed to read from standard input\n");
            return -1;
        }
        if (bytes == 0) {
            print("Завершение работы программы.\n");
            break;
        }

        buffer[bytes] = '\0';

        char *token;
        token = strtok(buffer, " ");
```

```
        char *endptr;
        int cmd = strtol(token, &endptr, 10);
        if (endptr == token) {
            print_error("error: invalid command\n");
            continue;
        }

        if (cmd == 1) {
            float a;
            if (!read_float(&token, "a", &endptr, &a)) {
                continue;
            }
            float dx;
            if (!read_float(&token, "dx", &endptr, &dx)) {
                continue;
            }

            float result = cos_derivative(a, dx);
            char output[256];
            int len = snprintf(output, sizeof(output),
                               "cos_derivative(%f, %f) = %f\n", a, dx, result);
            write(STDOUT_FILENO, output, len);
        } else if (cmd == 2) {
            float a;
            if (!read_float(&token, "a", &endptr, &a)) {
                continue;
            }

            float b;
            if (!read_float(&token, "b", &endptr, &b)) {
                continue;
            }

            float result = area(a, b);
            char output[256];
            int len = snprintf(output, sizeof(output), "area(%f, %f) = %f\n", a,
                               b, result);
            write(STDOUT_FILENO, output, len);
        } else {
            print_error("error: unknown command\n");
        }
    }
}
```

prog2.c

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <dlfcn.h>  // dlopen, dlsym, dlclose, RTLD_*
#include <unistd.h> // write
```

```c
#define sizeof_array(a) (sizeof(a) / sizeof((a)[0]))

typedef float (*cos_derivative_func)(float a, float dx);
typedef float (*area_func)(float a, float b);

static float func_impl_stub(float x, float y) {
    (void)x;
    (void)y;
    return 0.0f;
}

void print(char msg[]) { write(STDOUT_FILENO, msg, strlen(msg)); }

void print_error(char msg[]) { write(STDERR_FILENO, msg, strlen(msg)); }

int read_float(char **token, char *arg_name, char **endptr, float *out_value) {
    *token = strtok(NULL, " ");
    if (*token == NULL) {
        char output[256];
        snprintf(output, sizeof(output), "error: missing argument '%s'\n",
                 arg_name);
        print_error(output);
        return 0;
    }
    float value = strtof(*token, endptr);
    if (*endptr == *token) {
        char output[256];
        snprintf(output, sizeof(output), "error: invalid argument '%s'\n",
                 arg_name);
        print_error(output);
        return 0;
    }
    *out_value = value;
    return 1;
}

typedef struct {
    char *path;
    void *handle;
    cos_derivative_func cos_derivative;
    area_func area;
    char *description;
} Library;

int load_library(Library *lib) {
    lib->handle = dlopen(lib->path, RTLD_NOW | RTLD_LOCAL);
    if (!lib->handle) {
        print_error("error: failed to load library\n");
        return -1;
    }

    lib->cos_derivative =
        (cos_derivative_func)dlsym(lib->handle, "cos_derivative");
```

```
    if (!lib->cos_derivative) {
        print_error(
            "warning: failed to find cos_derivative function implementation\n");
        lib->cos_derivative = func_impl_stub;
    }
    lib->area = (area_func)dlsym(lib->handle, "area");
    if (!lib->area) {
        print_error("warning: failed to find area function implementation\n");
        lib->area = func_impl_stub;
    }

    return 0;
}

void print_current_library_info(int idx, const Library *lib) {
    printf("Текущая реализация: %d (%s)\n", idx + 1, lib->path);
    printf("%s\n", lib->description);
    printf("\n");
}

int populate_and_load_libs(Library *libs) {
    libs[0].path = "./library_1.so";
    libs[0].handle = NULL;
    libs[0].cos_derivative = NULL;
    libs[0].area = NULL;
    libs[0].description =
        "  cos_derivative(a, dx) = (cos(a+dx) - cos(a)) / dx\n"
        "  area(a, b) - площадь прямоугольника";
    libs[1].path = "./library_2.so";
    libs[1].handle = NULL;
    libs[1].cos_derivative = NULL;
    libs[1].area = NULL;
    libs[1].description =
        "  cos_derivative(a, dx) = (cos(a+dx) - cos(a-dx)) / (2*dx)\n"
        "  area(a, b) - площадь прямоугольного треугольника";

    for (size_t i = 0; i < 2; i++) {
        if (load_library(&libs[i]) != 0) {
            print_error("error: failed to load one of the libraries\n");
            for (size_t j = 0; j < i; j++) {
                if (libs[j].handle) {
                    dlclose(libs[j].handle);
                }
            }
            return -1;
        }
    }
    return 0;
}

int main() {
    Library libs[2];
    if (populate_and_load_libs(libs) != 0) {
        return -1;
```

```
    }

    int current = 0;

    print(
        "Программа 2 (динамическая загрузка библиотек во время выполнения)\n");
    print_current_library_info(current, &libs[current]);
    print("Формат ввода:\n");
    print("  0       -> переключить реализацию контрактов (между 1 и 2)\n");
    print("  1 a dx  -> вычислить производную cos(x) в точке a с шагом dx\n");
    print("  2 a b   -> вычислить площадь фигуры по сторонам a и b\n");
    print("EOF (Ctrl+D) для выхода.\n\n");

    while (1) {
        char buffer[1024];
        ssize_t bytes = read(STDIN_FILENO, buffer, sizeof(buffer) - 1);
        if (bytes == -1) {
            print_error("error: failed to read from standard input\n");
            return -1;
        }
        if (bytes == 0) {
            print("Завершение работы программы.\n");
            break;
        }

        buffer[bytes] = '\0';

        char *token;
        token = strtok(buffer, " ");

        char *endptr;
        int cmd = strtol(token, &endptr, 10);
        if (endptr == token) {
            print_error("error: invalid command\n");
            continue;
        }

        if (cmd == 0) {
            current = (current + 1) % (int)sizeof_array(libs);
            print_current_library_info(current, &libs[current]);
        } else if (cmd == 1) {
            float a;
            if (!read_float(&token, "a", &endptr, &a)) {
                continue;
            }
            float dx;
            if (!read_float(&token, "dx", &endptr, &dx)) {
                continue;
            }

            float result = libs[current].cos_derivative(a, dx);
            char output[256];
            int len = snprintf(output, sizeof(output),
                               "cos_derivative(%f, %f) = %f\n", a, dx, result);
```

```
            write(STDOUT_FILENO, output, len);
        } else if (cmd == 2) {
            float a;
            if (!read_float(&token, "a", &endptr, &a)) {
                continue;
            }

            float b;
            if (!read_float(&token, "b", &endptr, &b)) {
                continue;
            }

            float result = libs[current].area(a, b);
            char output[256];
            int len = snprintf(output, sizeof(output), "area(%f, %f) = %f\n", a,
                               b, result);
            write(STDOUT_FILENO, output, len);
        } else {
            print_error("error: unknown command\n");
        }
    }

    for (size_t i = 0; i < sizeof_array(libs); i++) {
        dlclose(libs[i].handle);
    }
}
```

Протокол работы программы

Программа 1, входные и выходные данные:

```
1 3.14 0.001
cos_derivative(3.140000, 0.001000) = -0.001073
1 1.57 0.001
cos_derivative(1.570000, 0.001000) = -1.000047
2 3 4
area(3.000000, 4.000000) = 12.000000
2 4 5
area(4.000000, 5.000000) = 20.000000
```

Программа 2, входные и выходные данные:

```
Текущая реализация: 1 (./library_1.so)
  cos_derivative(a, dx) = (cos(a+dx) - cos(a)) / dx
  area(a, b) - площадь прямоугольника

1 3.14 0.001
cos_derivative(3.140000, 0.001000) = -0.001073
2 3 4
area(3.000000, 4.000000) = 12.000000
0
Текущая реализация: 2 (./library_2.so)
  cos_derivative(a, dx) = (cos(a+dx) - cos(a-dx)) / (2*dx)
  area(a, b) - площадь прямоугольного треугольника

1 3.14 0.001
cos_derivative(3.140000, 0.001000) = -0.001580
2 3 4
area(3.000000, 4.000000) = 6.000000
```

Вывод

Анализ двух подходов использования библиотек

1) Динамическая библиотека, подключенная на этапе линковки:

- Плюсы:
  - Простота использования: функции вызываются как обычные.
  - Типы проверяются на этапе компиляции.
  - Меньше шансов ошибиться в имени функции.
  - Быстрее при вызове.
- Минусы:
  - Жёстко зашитая реализация: переключить на другую библиотеку без перекомпиляции/перелинковки нельзя.
  - Библиотека должна быть доступна системе при запуске

2) Динамическая загрузка во время выполнения:

- Плюсы:
  - Можно подгружать и менять реализации во время работы программы
  - Можно реализовать архитектуру плагинов: подгружать любую библиотеку, соблюдающую контракт.
- Минусы:
  - Более сложный код: нужно следить за ошибками, закрывать загруженные библиотеки
  - Ошибки в имени символа проявятся только во время выполнения.
  - Накладные расходы на загрузку/разрешение символов.

В результате выполнения данной работы я научился создавать динамические библиотеки на языке C и использовать их двумя способами: статическим подключением на этапе линковки и динамической загрузкой с помощью функций dlopen, dlsym и dlclose. Также я освоил работу с контрактами

функций, организацию нескольких реализаций и переключение между ними во время выполнения программы. Проанализировал различия между статическим и динамическим подключением библиотек.