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-210БВ-24

Студент: Тодуа Н.Г.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.11.25

Москва, 2025

Постановка задачи

Вариант 14.

Функция 1: Расчет производной функции cos(x) в точке a с приращением dx:
Реализация №1: f’(x) = (f(a + dx) - f(a)) / dx

Реализация №2: f’(x) = (f(a + dx) - f(a - dx)) / (2dx)

Функция 2: Расчет значения числа π при заданной длине ряда (k):

Реализация №1: Ряд Лейбница

Реализация №2: Формула Валлиса

Общий метод и алгоритм решения

Использованные функции POSIX API для работы с динамическими библиотеками:

- void *dlopen(const char *filename, int flags); – Загружает динамическую библиотеку (shared object) в адресное пространство вызывающего процесса.
- void *dlsym(void *handle, const char *symbol); – Получает адрес экспортируемой функции из загруженной библиотеки по её имени.
- int dlclose(void *handle); – Выгружает ранее загруженную библиотеку из памяти процесса.
- char *dlerror(void); – Возвращает строку с описанием последней ошибки, произошедшей при вызове функций

Работа с динамическими библиотеками реализована двумя способами: через линковку на этапе компиляции и через динамическую загрузку во время выполнения. Программа №1 (linktime_static1, linktime_static2) вызывает функции напрямую, будучи связанной с одной из динамических библиотек на этапе сборки. Программа №2 загружает динамические библиотеки вручную с помощью dlopen(), получает адреса функций через dlsym() и поддерживает переключение между разными реализациями по команде пользователя.

Код программы

contract.h

```
#ifndef CONTRACT_H
#define CONTRACT_H

float cos_derivative(float a, float dx);
float pi(int k);

#endif
```

library1.c

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

#ifdef __cplusplus
extern "C" {
#endif

float cos_derivative(float a, float dx) {
    if (dx <= 0.0f) {
        return 0.0f;
    }
    // Реализация №1: f'(x) = (f(a + dx) - f(a))/dx
    return (cosf(x: a + dx) - cosf(x: a)) / dx;
}

float pi(int k) {
    if (k <= 0) {
        return 0.0f;
    }

    // Реализация №1: Ряд Лейбница
    double sum = 0.0;
    for (int i = 0; i < k; ++i) {
        double term = 1.0 / (2 * i + 1);
        if (i % 2 == 1) {
            sum -= term;
        } else {
            sum += term;
        }
    }

    return (float)(4.0 * sum);
}

#ifdef __cplusplus
}
#endif
```

library2.c

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

#ifdef __cplusplus
extern "C" {
#endif

float cos_derivative(float a, float dx) {
    if (dx <= 0.0f) {
        return 0.0f;
    }
    // Реализация №2: f'(x) = (f(a + dx) - f(a - dx))/(2dx)
    return (cosf(x: a + dx) - cosf(x: a - dx)) / (2 * dx);
}

float pi(int k) {
    if (k <= 0) {
        return 0.0f;
    }

    // Реализация №2: Формула Валлиса
    double product = 1.0;
    for (int n = 1; n <= k; ++n) {
        double num = 2.0 * n;
        double term = (num / (num - 1)) * (num / (num + 1));
        product *= term;
    }

    return (float)(2.0 * product);
}

#ifdef __cplusplus
}
#endif
```

linktime.c

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

#include "contract.h"

int main() {
    printf(format: "\nCommands:\n");
    printf(format: "  1 a dx     - compute derivative of cos(x) at point a with increment dx\n");
    printf(format: "  2 k        - compute pi using k terms\n");
    printf(format: "\n");

    char buffer[256];
    while (1) {
        printf(format: "Enter command: ");
        if (fgets(s: buffer, n: sizeof(buffer), stream: stdin) == NULL) {
            break;
        }

        // Удаляем символ новой строки
        buffer[strcspn(s: buffer, reject: "\n")] = '\0';

        if (buffer[0] == '1') {
            float a, dx;
            if (sscanf(s: buffer + 1, format: "%f %f", &a, &dx) == 2) {
                float result = cos_derivative(a, dx);
                printf(format: "Result (cos derivative): %.6f\n", result);
            } else {
                printf(format: "Error: Invalid args for command 1. Expected: 1 a dx\n");
            }
            continue;
        }

        if (buffer[0] == '2') {
            int k;
            if (sscanf(s: buffer + 1, format: "%d", &k) == 1) {
                float result = pi(k);
                printf(format: "Result (pi approximation): %.6f\n", result);
            } else {
                printf(format: "Error: Invalid args for command 2. Expected: 2 k\n");
            }
            continue;
        }
```

```c
        if (buffer[0] == '\0') {
            continue;
        }

        printf(format: "Error: Unknown command. Use 1 or 2.\n");
    }

    return 0;
}
```

runtime.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dlfcn.h>

// Прототипы функций
typedef float (*CosDerivativeFunc)(float, float);
typedef float (*PiFunc)(int);

// Заглушки для функций
static float stub_cos_derivative(float a, float dx) {
    (void)a; (void)dx;
    return 0.0f;
}

static float stub_pi(int k) {
    (void)k;
    return 0.0f;
}

int main() {
    void* library = NULL;
    CosDerivativeFunc f_cos_derivative = stub_cos_derivative;
    PiFunc f_pi = stub_pi;

    printf(format: "Enter path to shared library (e.g., ./lib1.so): ");
    char lib_path[256];
    if (fgets(s: lib_path, n: sizeof(lib_path), stream: stdin) == NULL) {
        printf(format: "Error reading library path\n");
        return 1;
    }

    lib_path[strcspn(s: lib_path, reject: "\n")] = '\0';

    // Загрузка библиотеки
    library = dlopen(file: lib_path, mode: RTLD_LAZY);
    if (library == NULL) {
        printf(format: "Error: Failed to load library (%s). Using stubs.\n", dlerror());
    } else {
        f_cos_derivative = (CosDerivativeFunc)dlsym(handle: library, name: "cos_derivative");
        f_pi = (PiFunc)dlsym(handle: library, name: "pi");
```

```
        if (f_cos_derivative == NULL) {
            printf(format: "Error: 'cos_derivative' not found. Using stub.\n");
            f_cos_derivative = stub_cos_derivative;
        }
        if (f_pi == NULL) {
            printf(format: "Error: 'pi' not found. Using stub.\n");
            f_pi = stub_pi;
        }
    }

    printf(format: "\nCommands:\n");
    printf(format: "  0              - switch to another library\n");
    printf(format: "  1 a dx         - compute derivative of cos(x) at point a with increment dx\n");
    printf(format: "  2 k            - compute pi using k terms\n");
    printf(format: "\n");

    char buffer[256];
    while (1) {
        printf(format: "Enter command: ");
        if (fgets(s: buffer, n: sizeof(buffer), stream: stdin) == NULL) {
            break;
        }

        buffer[strcspn(s: buffer, reject: "\n")] = '\0';

        if (buffer[0] == '0') {
            if (library) {
                dlclose(handle: library);
                library = NULL;
            }

            printf(format: "Enter new library path: ");
            if (fgets(s: lib_path, n: sizeof(lib_path), stream: stdin) == NULL) {
                break;
            }
            lib_path[strcspn(s: lib_path, reject: "\n")] = '\0';

            library = dlopen(file: lib_path, mode: RTLD_LAZY);
            if (library == NULL) {
                printf(format: "Error: Failed to load new library (%s). Using stubs.\n", dlerror());
                f_cos_derivative = stub_cos_derivative;
```

```
            f_pi = stub_pi;
        } else {
            f_cos_derivative = (CosDerivativeFunc)dlsym(handle: library, name: "cos_derivative");
            f_pi = (PiFunc)dlsym(handle: library, name: "pi");

            if (f_cos_derivative == NULL) {
                printf(format: "Error: 'cos_derivative' not found. Using stub.\n");
                f_cos_derivative = stub_cos_derivative;
            }
            if (f_pi == NULL) {
                printf(format: "Error: 'pi' not found. Using stub.\n");
                f_pi = stub_pi;
            }
            printf(format: "Library switched successfully!\n");
        }
        continue;
    }

    if (buffer[0] == '1') {
        float a, dx;
        if (sscanf(s: buffer + 1, format: "%f %f", &a, &dx) == 2) {
            float result = f_cos_derivative(a, dx);
            printf(format: "Result (cos derivative): %.6f\n", result);
        } else {
            printf(format: "Error: Invalid args for command 1. Expected: 1 a dx\n");
        }
        continue;
    }

    if (buffer[0] == '2') {
        int k;
        if (sscanf(s: buffer + 1, format: "%d", &k) == 1) {
            float result = f_pi(k);
            printf(format: "Result (pi approximation): %.6f\n", result);
        } else {
            printf(format: "Error: Invalid args for command 2. Expected: 2 k\n");
        }
        continue;
    }

    if (buffer[0] == '\0') {
        continue;
    }

    printf(format: "Error: Unknown command. Use 0, 1, or 2.\n");
}

if (library) {
    dlclose(handle: library);
}

return 0;
}
```

Протокол работы программы

Тестирование:

```
[mafonchik@mafonchik-aspirea31544p lab4]$ ./runtime
Enter path to shared library (e.g., ./lib1.so): ./lib1.so

Commands:
  0             - switch to another library
  1 a dx        - compute derivative of cos(x) at point a with increment dx
  2 k           - compute pi using k terms

Enter command: 2 14
Result (pi approximation): 3.070255
Enter command: 2 100
Result (pi approximation): 3.131593
Enter command: 4
Error: Unknown command. Use 0, 1, or 2.
```

```
[mafonchik@mafonchik-aspirea31544p lab4]$ ./linktime_static1

Commands:
  1 a dx    - compute derivative of cos(x) at point a with increment dx
  2 k       - compute pi using k terms

Enter command: 2 10
Result (pi approximation): 3.041840
Enter command: 2 10000000
Result (pi approximation): 3.141593
Enter command: 1 0.1
Error: Invalid args for command 1. Expected: 1 a dx
Enter command: 1 0.1 1
Result (cos derivative): -0.541408
```

Вывод

Использование динамических библиотек двумя способами – на этапе линковки и во время выполнения – демонстрирует разницу в гибкости и моменте связывания: в первом случае зависимость от конкретной реализации жёстко фиксируется при компиляции программы, что упрощает код, но исключает возможность замены библиотеки без пересборки. Во втором, программа остаётся полностью независимой от реализации, загружая и связывая функции динамически в процессе работы, что позволяет переключаться между разными версиями библиотек на лету, обеспечивая расширяемость и поддержку плагинов, но требуя более сложной обработки ошибок и ручного управления памятью и дескрипторами. Таким образом, link-time linking предпочтителен для стабильных, закрытых систем, а run-time loading – для модульных и гибких приложений.