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: М80-211Бв-24

Студент: Скворцов Ю.И.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата:11.12.25

Москва, 2025

Постановка задачи

Вариант 6.

Расчет интеграла функции sin(x) на отрезке [A, B] с шагом e: Сигнатура функции: float sin_integral(float a, float b, float e); ● Реализация №1: Подсчет интеграла методом прямоугольников; ● Реализация №2: Подсчет интеграла методом трапеций.

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

dlopen(const char *filename, int flags) - Динамически загружает разделяемую библиотеку в память

dlsym(void *handle, const char *symbol) - Получает адрес символа (функции/переменной) из загруженной библиотеки

dlclose(void *handle) - Закрывает динамическую библиотеку, уменьшает счетчик ссылок

dlerror(void) - Возвращает строку с описанием последней ошибки dl функций

Работа программы:

Первая программа, со статической линковкой, работает по принципу "всё в одном". При её компиляции функции для вычисления интеграла и площади фигуры включаются прямо в исполняемый файл. Когда пользователь запускает эту программу, она сразу готова к работе — все необходимые алгоритмы уже находятся в её памяти. Программа запрашивает у пользователя команды: если ввести "1", она ожидает три числа — границы интервала и шаг, затем вычисляет интеграл синуса методом прямоугольников и выводит результат. Если ввести "2", она ждёт две стороны фигуры и вычисляет площадь прямоугольника. При этом реализация функций жёстко зафиксирована — нельзя переключиться на другие методы вычислений без перекомпиляции всей программы. Когда пользователь вводит "exit", программа завершает работу.

Вторая программа, с динамической загрузкой, устроена иначе. При запуске она не содержит реализаций вычислений — вместо этого она пытается загрузить внешнюю библиотеку libmath.so из текущей директории. Если библиотека найдена и успешно загружена, программа получает из неё адреса нужных функций. Здесь появляется возможность выбора: начальная реализация использует метод прямоугольников для интеграла и расчёт площади прямоугольника, но командой "0" можно переключиться на альтернативную реализацию — метод трапеций и площадь треугольника. Это переключение происходит динамически: программа ищет в той же библиотеке другие функции с соответствующими именами. Весь процесс происходит во время выполнения — программа не знает заранее, какие функции будут доступны, и обращается к библиотеке как к чёрному ящику, из которого можно извлекать разные реализации по мере необходимости. Когда библиотека больше не нужна, программа корректно выгружает её из памяти.

Код программы

static.c

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "integral.h"
#include "area.h"

int main() {
    char buffer[1024];
    char cmd[10];
    int len;
    while (1) {
        len = snprintf(buffer, sizeof(buffer),
                "Enter command (1 a b e / 2 a b / exit): ");
        write(STDOUT_FILENO, buffer, len);
        fflush(stdout);
        scanf("%s", cmd);

        if (strcmp(cmd, "exit") == 0) break;

        if (strcmp(cmd, "1") == 0) {
            float a, b, e;
            scanf("%f %f %f", &a, &b, &e);
            float result = sin_integral_rect(a, b, e);
            len = snprintf(buffer, sizeof(buffer),
                    "Integral sin(x) on [%.2f, %.2f]: %.6f\n",
                    a, b, result);
            write(STDOUT_FILENO, buffer, len);
        }
        else if (strcmp(cmd, "2") == 0) {
```

```
            float a, b;
            scanf("%f %f", &a, &b);
            float result = area_rect(a, b);
            len = snprintf(buffer, sizeof(buffer),
                    "Area (%.2f x %.2f): %.6f\n",
                    a, b, result);
            write(STDOUT_FILENO, buffer, len);
        }
        else {
            len = snprintf(buffer, sizeof(buffer), "Unknown command\n");
            write(STDOUT_FILENO, buffer, len);
        }
    }

    return 0;
}
```

dynamic.c

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

typedef float (*integral_func)(float, float, float);
typedef float (*area_func)(float, float);

int main() {
    char buffer[1024];
```

```
    int len;
    void *lib = dlopen("./libmath.so", RTLD_LAZY);
    if (!lib) {
        len = snprintf(buffer, sizeof(buffer),
                "Error lin loading: %s\n", dlerror());
        write(STDOUT_FILENO, buffer, len);
        return 1;
    }

    integral_func sin_integral = (integral_func)dlsym(lib, "sin_integral_rect");
    area_func area = (area_func)dlsym(lib, "area_rect");
    int impl = 1;

    char cmd[10];

    while (1) {
        len = snprintf(buffer, sizeof(buffer), "\nCurrent realisation: %d\n",
impl);
        write(STDOUT_FILENO, buffer, len);

        len = snprintf(buffer, sizeof(buffer),
                "Enter command (0 / 1 a b e / 2 a b / exit): ");
        write(STDOUT_FILENO, buffer, len);
        fflush(stdout);
        scanf("%s", cmd);

        if (strcmp(cmd, "exit") == 0) break;

        if (strcmp(cmd, "0") == 0) {
```

```
            impl = (impl == 1) ? 2 : 1;


            if (impl == 1) {
                sin_integral = (integral_func)dlsym(lib, "sin_integral_rect");
                area = (area_func)dlsym(lib, "area_rect");
            } else {
                sin_integral = (integral_func)dlsym(lib, "sin_integral_trap");
                area = (area_func)dlsym(lib, "area_triangle");
            }


            len = snprintf(buffer, sizeof(buffer),
                    "Switched to realisation %d\n", impl);
            write(STDOUT_FILENO, buffer, len);
            continue;
        }


        if (strcmp(cmd, "1") == 0) {
            float a, b, e;
            scanf("%f %f %f", &a, &b, &e);
            float result = sin_integral(a, b, e);
            len = snprintf(buffer, sizeof(buffer), "Result: %.6f\n", result);
            write(STDOUT_FILENO, buffer, len);
        }
        else if (strcmp(cmd, "2") == 0) {
            float a, b;
            scanf("%f %f", &a, &b);
            float result = area(a, b);
            len = snprintf(buffer, sizeof(buffer), "Result: %.6f\n", result);
            write(STDOUT_FILENO, buffer, len);
```

```
        }
        else {
            len = snprintf(buffer, sizeof(buffer), "Unknown command\n");
            write(STDOUT_FILENO, buffer, len);
        }
    }

    dlclose(lib);
    return 0;
}
```

area.c

```
#include "area.h"

float area_rect(float a, float b) {
    return a * b;
}

float area_triangle(float a, float b) {
    return (a * b) / 2.0f;
}
```

integral.c

```
#include "integral.h"
#include <math.h>

float sin_integral_rect(float a, float b, float e) {
    float result = 0.0;
    float step = (e > 0) ? e : 0.001f;
```

```
    for (float x = a; x < b; x += step) {
        float next_x = (x + step < b) ? x + step : b;
        float mid = (x + next_x) / 2.0f;
        result += sinf(mid) * (next_x - x);
    }

    return result;
}

float sin_integral_trap(float a, float b, float e) {
    float result = 0.0;
    float step = (e > 0) ? e : 0.001f;

    for (float x = a; x < b; x += step) {
        float next_x = (x + step < b) ? x + step : b;
        result += (sinf(x) + sinf(next_x)) * (next_x - x) / 2.0f;
    }

    return result;
}
```

Протокол работы программы

```
evilpig@DESKTOP-5BT55H9:/mnt/c/Users/EvilPig31/Desktop/C/os_labs/4$ ./static
Enter command (1 a b e / 2 a b / exit): 1 2 3 1
Integral sin(x) on [2.00, 3.00]: 0.598472
Enter command (1 a b e / 2 a b / exit): 2 3 4
Area (3.00 x 4.00): 12.000000
Enter command (1 a b e / 2 a b / exit): exit
```

```
evilpig@DESKTOP-5BT55H9:/mnt/c/Users/EvilPig31/Desktop/C/os_labs/4$ ./dynamic

Current realisation: 1
Enter command (0 / 1 a b e / 2 a b / exit): 1 2 3 1
Result: 0.598472

Current realisation: 1
Enter command (0 / 1 a b e / 2 a b / exit): 2 3 4
Result: 12.000000

Current realisation: 1
Enter command (0 / 1 a b e / 2 a b / exit): 0
Switched to realisation 2

Current realisation: 2
Enter command (0 / 1 a b e / 2 a b / exit): 1 2 3 1
Result: 0.525209

Current realisation: 2
Enter command (0 / 1 a b e / 2 a b / exit): 2 3 4
Result: 6.000000

Current realisation: 2
Enter command (0 / 1 a b e / 2 a b / exit): exit
```

Вывод

В ходе выполнения лабораторной работы были успешно изучены и практически реализованы два принципиально разных подхода к использованию библиотек в программировании.

Первый подход — статическая линковка — показал свою эффективность в ситуациях, где требуется максимальная производительность и предсказуемость. Все функции жёстко включаются в исполняемый файл на этапе компиляции, что устраняет любые накладные расходы на загрузку и разрешение символов во время выполнения. Однако этот подход страдает от недостатка гибкости: для изменения реализации необходимо перекомпилировать всю программу, а размер исполняемого файла увеличивается за счёт включённого кода библиотек.

Второй подход — динамическая загрузка — продемонстрировал принципиально иную философию. Библиотеки загружаются в память только при необходимости, что позволяет экономить ресурсы и обеспечивает невиданную гибкость. Возможность "горячей" замены реализаций, обновления функциональности без перекомпиляции основной программы и разделения кода между несколькими приложениями делает этот подход незаменимым в современных сложных системах. Однако за эту гибкость приходится платить: появляются накладные расходы на загрузку библиотек, усложняется отладка и распределение программ.

Практическая реализация обеих программ наглядно продемонстрировала, что выбор между статическим и динамическим подходом зависит от конкретных требований проекта. Для

небольших утилит с фиксированной функциональностью предпочтительна статическая линковка, тогда как для крупных модульных систем с требующейся гибкостью и возможностью расширения динамическая загрузка становится единственно верным решением. Оба подхода имеют право на существование и должны использоваться осознанно, с учётом компромиссов между производительностью, гибкостью и сложностью сопровождения.