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: Ергизов А.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 20.11.25

Москва, 2025

Постановка задачи

Вариант 28.

№5

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

Сигнатура функции: float pi(int k);

● Реализация №1: Ряд Лейбница

● Реализация №2: Формула Валлиса

№7

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 вызовы:

- void *dlopen(const char *filename, int flags);
  Загружает динамическую библиотеку в память процесса
- void *dlsym(void *handle, const char *symbol);
  Находит адрес функции/переменной в загруженной библиотеке
- int dlclose(void *handle);
  Выгружает библиотеку из памяти
- char *dlerror(void);
  Возвращает текст последней ошибки работы с DLL
- __attribute__((visibility("default")))
  Определяет, какие символы экспортируются из библиотеки

Созданы две динамические библиотеки, реализующие математические функции.

Функция area(float a, float b, int figure_type) - вычисление площади фигур.

Функция calculate_pi_leibniz(int iterations) - вычисление pi по ряду Лейбница.

Функция calculate_pi_wallis(int iterations) - вычисление pi по формуле Валлиса.

Программа №1: Статическое связывание (`static_main`)

Библиотеки линкуются на этапе компиляции

Прямой вызов функций библиотек

Не требует загрузки библиотек во время выполнения

Программа №2: Динамическое связывание (`dynamic_main`)

Библиотеки загружаются во время выполнения через `dlopen()`

Функции связываются через `dlsym()`

Поддерживает переключение между реализациями командой "0"

Код программы

Area.h

```
#ifndef AREA_H

#define AREA_H

#ifdef _WIN32
#ifdef BUILD_DLL
#define EXPORT __declspec(dllexport)
#else
#define EXPORT __declspec(dllimport)
#endif
#else
#define EXPORT __attribute__((visibility("default")))
#endif

#define RECTANGLE 1
#define TRIANGLE 2

EXPORT float area(float a, float b, int figure_type);
EXPORT const char *get_figure_name(int figure_type);

#endif
```

Pi.h

```
#ifndef PI_H

#define PI_H

#ifdef _WIN32
#ifdef BUILD_DLL
```

```
#define EXPORT __declspec(dllexport)
#else
#define EXPORT __declspec(dllimport)
#endif
#else
#define EXPORT __attribute__((visibility("default")))
#endif

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif // M_PI

EXPORT double calculate_pi_leibniz(int iterations);
EXPORT double calculate_pi_wallis(int iterations);

#endif
```

Area.c

```
#include <stdio.h>


#define BUILD_DLL
#include "../include/Area.h"

static float rectangle_area(float a, float b)
{
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return -1.0f;
}
return a * b;
}

static float triangle_area(float a, float b)
{
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return -1.0f;
}
return (a * b) / 2.0f;
}

EXPORT float area(float a, float b, int figure_type)
{
switch (figure_type)
{
case RECTANGLE:
```

```
return rectangle_area(a, b);
case TRIANGLE:
return triangle_area(a, b);
default:
printf("Ошибка: неизвестный тип фигуры (%d)\n", figure_type);
return -1.0f;
}
}

EXPORT const char *get_figure_name(int figure_type)
{
switch (figure_type)
{
case RECTANGLE:
return "прямоугольника";
case TRIANGLE:
return "треугольника";
default:
return "неизвестной фигуры";
}
}
```

Area2.c

```
#include <stdio.h>

#include <math.h>

#define BUILD_DLL
#include "../include/Area.h"

static float rectangle_area_alt(float a, float b)
{
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return -1.0f;
}

if (a == b)
{
printf("(Это квадрат!)\n");
}

return a * b;
}

static float triangle_area_alt(float a, float b)
{
if (a <= 0 || b <= 0)
```

```
{
printf("Ошибка: стороны должны быть положительными\n");
return -1.0f;
}
if (a == b)
{
printf("(Это равнобедренный прямоугольный треугольник!)\n");
}

return (a * b) / 2.0f;
}

EXPORT float area(float a, float b, int figure_type)
{
printf("[Реализация 2] ");

switch (figure_type)
{
case RECTANGLE:
return rectangle_area_alt(a, b);
case TRIANGLE:
return triangle_area_alt(a, b);
default:
printf("Ошибка: неизвестный тип фигуры (%d)\n", figure_type);
return -1.0f;
}
}

EXPORT const char *get_figure_name(int figure_type)
{
switch (figure_type)
{
case RECTANGLE:
return "прямоугольника (реализация 2)";
case TRIANGLE:
return "треугольника (реализация 2)";
default:
return "неизвестной фигуры";
}
}
```

Pi.c

```
#include <stdio.h>

#include <math.h>
#include <time.h>

#define BUILD_DLL
#include "../include/Pi.h"
```

```
EXPORT double calculate_pi_leibniz(int iterations)
{
double pi_quarter = 0.0;
int sign = 1;

if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return -1.0;
}

for (int i = 0; i < iterations; i++)
{
pi_quarter += sign * 1.0 / (2 * i + 1);
sign = -sign;
}

return 4 * pi_quarter;
}

EXPORT double calculate_pi_wallis(int iterations)
{
double pi_half = 1.0;

if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return -1.0;
}

for (int i = 1; i <= iterations; i++)
{
double numerator = 2.0 * i;
pi_half *= (numerator / (numerator - 1)) * (numerator / (numerator + 1));
}

return 2 * pi_half;
}
```

Pi2.c

```
#include <stdio.h>

#include <math.h>
#include <time.h>

#define BUILD_DLL
#include "../include/Pi.h"
```

```
EXPORT double calculate_pi_leibniz(int iterations)
{
double pi_quarter = 0.0;
int sign = 1;

if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return -1.0;
}

printf("[Реализация 2] Вычисление pi методом Лейбница...\n");

for (int i = 0; i < iterations; i++)
{
pi_quarter += sign * 1.0 / (2 * i + 1);
sign = -sign;
}

double result = 4 * pi_quarter;
printf(" Погрешность: %.10f\n", fabs(result - M_PI));

return result;
}

EXPORT double calculate_pi_wallis(int iterations)
{
double pi_half = 1.0;

if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return -1.0;
}

printf("[Реализация 2] Вычисление pi методом Валлиса...\n");

for (int i = 1; i <= iterations; i++)
{
double numerator = 2.0 * i;
pi_half *= (numerator / (numerator - 1)) * (numerator / (numerator + 1));
}

double result = 2 * pi_half;
printf(" Погрешность: %.10f\n", fabs(result - M_PI));

return result;
}
```

static_main.c

```c
#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#include "../include/Area.h"
#include "../include/Pi.h"

void print_help()
{
printf("Доступные команды:\n");
printf("0 - переключить реализацию (только для динамической версии)\n");
printf("1 <итерации> - вычислить pi методом Лейбница\n");
printf("2 <итерации> - вычислить pi методом Валлиса\n");
printf("3 <a> <b> - вычислить площадь прямоугольника\n");
printf("4 <a> <b> - вычислить площадь треугольника\n");
printf("help - показать эту справку\n");
printf("exit - выйти\n");
}

void handle_pi_leibniz(int iterations)
{
if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return;
}
double result = calculate_pi_leibniz(iterations);
printf("pi (Лейбниц, %d итераций) = %.10f\n", iterations, result);
}

void handle_pi_wallis(int iterations)
{
if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return;
}
double result = calculate_pi_wallis(iterations);
printf("pi (Валлис, %d итераций) = %.10f\n", iterations, result);
}

void handle_rectangle_area(float a, float b)
{
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return;
```

```
}
float result = area(a, b, RECTANGLE);
printf("Площадь прямоугольника (%.2f x %.2f) = %.2f\n", a, b, result);
}

void handle_triangle_area(float a, float b)
{
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return;
}
float result = area(a, b, TRIANGLE);
printf("Площадь треугольника (%.2f x %.2f) / 2 = %.2f\n", a, b, result);
}

int main()
{
char input[256];
char command[10];

printf("=== Программа №1 - Статическое связывание ===\n");
printf("Библиотеки загружены на этапе компиляции\n\n");
print_help();

while (1)
{
printf("\n> ");

if (fgets(input, sizeof(input), stdin) == NULL)
{
break;
}

input[strcspn(input, "\n")] = 0;

if (strlen(input) == 0)
{
continue;
}

int args_parsed = sscanf(input, "%9s", command);

if (args_parsed != 1)
{
continue;
}

if (strcmp(command, "exit") == 0)
```

```
{
printf("Выход...\n");
break;
}
else if (strcmp(command, "help") == 0)
{
print_help();
}
else if (strcmp(command, "0") == 0)
{
printf("В статической версии переключение реализаций недоступно\n");
printf("Используйте динамическую версию программы\n");
}
else if (strcmp(command, "1") == 0)
{
int iterations;
if (sscanf(input, "%*s %d", &iterations) == 1)
{
handle_pi_leibniz(iterations);
}
else
{
printf("Ошибка: укажите количество итераций\n");
printf("Использование: 1 <итерации>\n");
}
}
else if (strcmp(command, "2") == 0)
{
int iterations;
if (sscanf(input, "%*s %d", &iterations) == 1)
{
handle_pi_wallis(iterations);
}
else
{
printf("Ошибка: укажите количество итераций\n");
printf("Использование: 2 <итерации>\n");
}
}
else if (strcmp(command, "3") == 0)
{
float a, b;
if (sscanf(input, "%*s %f %f", &a, &b) == 2)
{
handle_rectangle_area(a, b);
}
else
{
printf("Ошибка: укажите две стороны\n");
printf("Использование: 3 <a> <b>\n");
}
```

```
}
else if (strcmp(command, "4") == 0)
{
float a, b;
if (sscanf(input, "%*s %f %f", &a, &b) == 2)
{
handle_triangle_area(a, b);
}
else
{
printf("Ошибка: укажите две стороны\n");
printf("Использование: 4 <a> <b>\n");
}
}
else
{
printf("Неизвестная команда: %s\n", command);
printf("Введите 'help' для справки\n");
}
}

return 0;
}
```

dinamyc_main.c

```
#include <stdio.h>

#include <stdlib.h>
#include <string.h>
#include <dlfcn.h>

typedef double (*pi_func_t)(int);
typedef float (*area_func_t)(float, float, int);
typedef const char *(*info_func_t)(int);

typedef struct
{
void *handle;
pi_func_t leibniz;
pi_func_t wallis;
area_func_t area;
info_func_t figure_name;
const char *name;
} Library;

Library current_lib = {0};
int current_implementation = 1;

int load_library(int impl_number)
```

```
{
const char *lib_paths[] = {
"./lib/libarea.so",
"./lib/libarea2.so"};

if (impl_number < 1 || impl_number > 2)
{
printf("Ошибка: доступны реализации 1 и 2\n");
return 0;
}

if (current_lib.handle)
{
dlclose(current_lib.handle);
current_lib.handle = NULL;
}

current_lib.handle = dlopen(lib_paths[impl_number - 1], RTLD_LAZY);
if (!current_lib.handle)
{
printf("Ошибка загрузки библиотеки %s: %s\n",
lib_paths[impl_number - 1], dlerror());
return 0;
}

current_lib.leibniz = (pi_func_t)dlsym(current_lib.handle,
"calculate_pi_leibniz");
current_lib.wallis = (pi_func_t)dlsym(current_lib.handle,
"calculate_pi_wallis");
current_lib.area = (area_func_t)dlsym(current_lib.handle, "area");
current_lib.figure_name = (info_func_t)dlsym(current_lib.handle,
"get_figure_name");

char *error;
if ((error = dlerror()) != NULL)
{
printf("Ошибка загрузки функций: %s\n", error);
dlclose(current_lib.handle);
current_lib.handle = NULL;
return 0;
}

current_lib.name = lib_paths[impl_number - 1];
current_implementation = impl_number;

printf("Загружена реализация %d: %s\n", impl_number, lib_paths[impl_number -
1]);
return 1;
}
```

```
void print_help()
{
printf("Доступные команды:\n");
printf("0 - переключить реализацию (текущая: %d)\n", current_implementation);
printf("1 <итерации> - вычислить pi методом Лейбница\n");
printf("2 <итерации> - вычислить pi методом Валлиса\n");
printf("3 <a> <b> - вычислить площадь прямоугольника\n");
printf("4 <a> <b> - вычислить площадь треугольника\n");
printf("help - показать эту справку\n");
printf("exit - выйти\n");
}

void handle_pi_leibniz(int iterations)
{
if (!current_lib.handle || !current_lib.leibniz)
{
printf("Ошибка: библиотека не загружена\n");
return;
}
if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return;
}
double result = current_lib.leibniz(iterations);
printf("pi (Лейбниц, %d итераций) = %.10f\n", iterations, result);
}

void handle_pi_wallis(int iterations)
{
if (!current_lib.handle || !current_lib.wallis)
{
printf("Ошибка: библиотека не загружена\n");
return;
}
if (iterations <= 0)
{
printf("Ошибка: количество итераций должно быть положительным\n");
return;
}
double result = current_lib.wallis(iterations);
printf("pi (Валлис, %d итераций) = %.10f\n", iterations, result);
}

void handle_rectangle_area(float a, float b)
{
if (!current_lib.handle || !current_lib.area || !current_lib.figure_name)
{
printf("Ошибка: библиотека не загружена\n");
return;
```

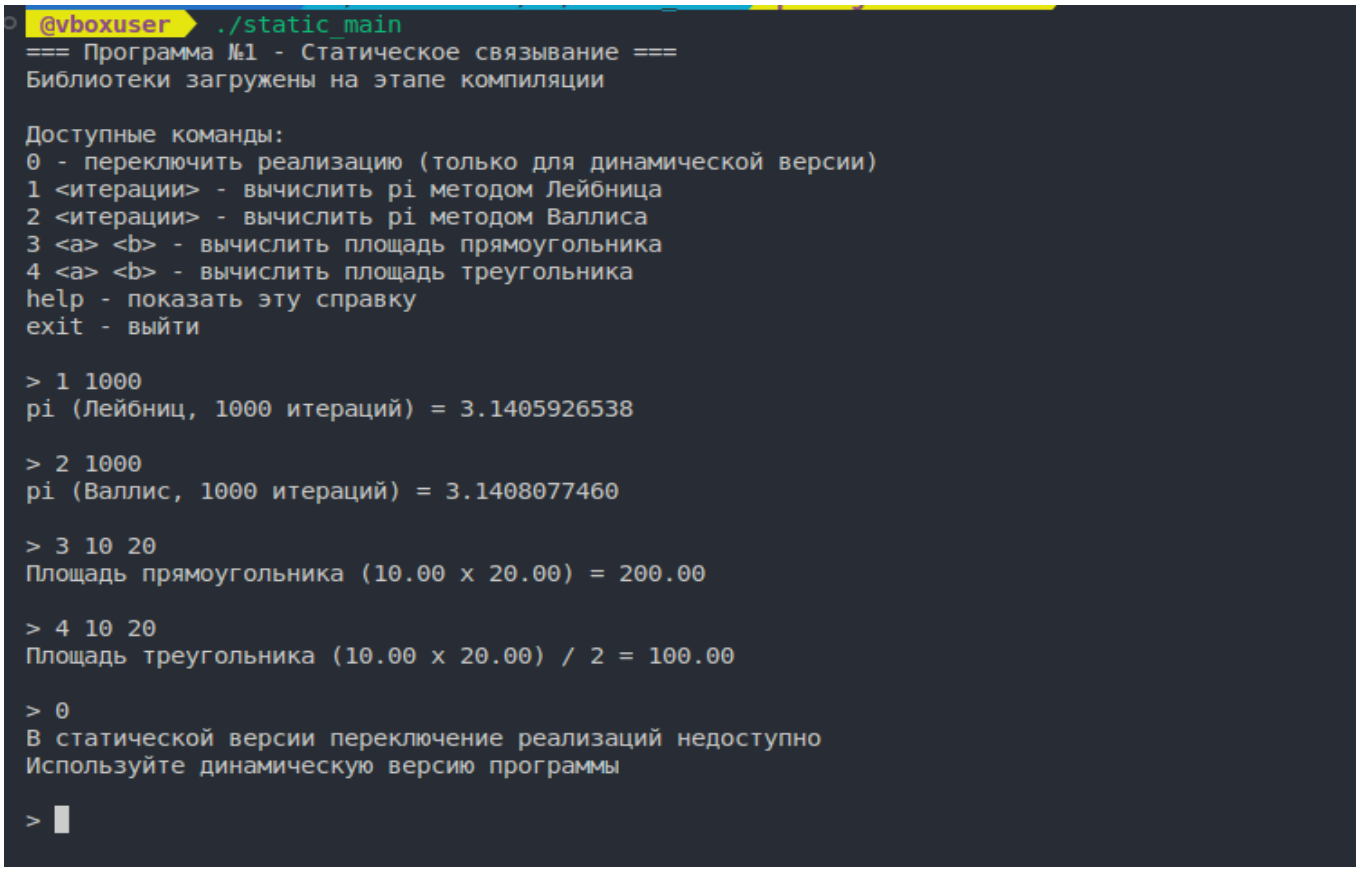
```
}
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return;
}
float result = current_lib.area(a, b, 1);
const char *figure_name = current_lib.figure_name(1);
printf("Площадь %s (%.2f x %.2f) = %.2f\n", figure_name, a, b, result);
}

void handle_triangle_area(float a, float b)
{
if (!current_lib.handle || !current_lib.area || !current_lib.figure_name)
{
printf("Ошибка: библиотека не загружена\n");
return;
}
if (a <= 0 || b <= 0)
{
printf("Ошибка: стороны должны быть положительными\n");
return;
}
float result = current_lib.area(a, b, 2);
const char *figure_name = current_lib.figure_name(2);
printf("Площадь %s (%.2f x %.2f) / 2 = %.2f\n", figure_name, a, b, result);
}

void switch_implementation()
{
int new_impl = (current_implementation == 1) ? 2 : 1;
if (load_library(new_impl))
{
printf("Реализация переключена на %d\n", new_impl);
}
else
{
printf("Не удалось переключить реализацию\n");
}
}

int main()
{
char input[256];
char command[10];

printf("=== Программа №2 - Динамическое связывание ===\n");
printf("Библиотеки загружаются во время выполнения\n\n");

if (!load_library(1))
```

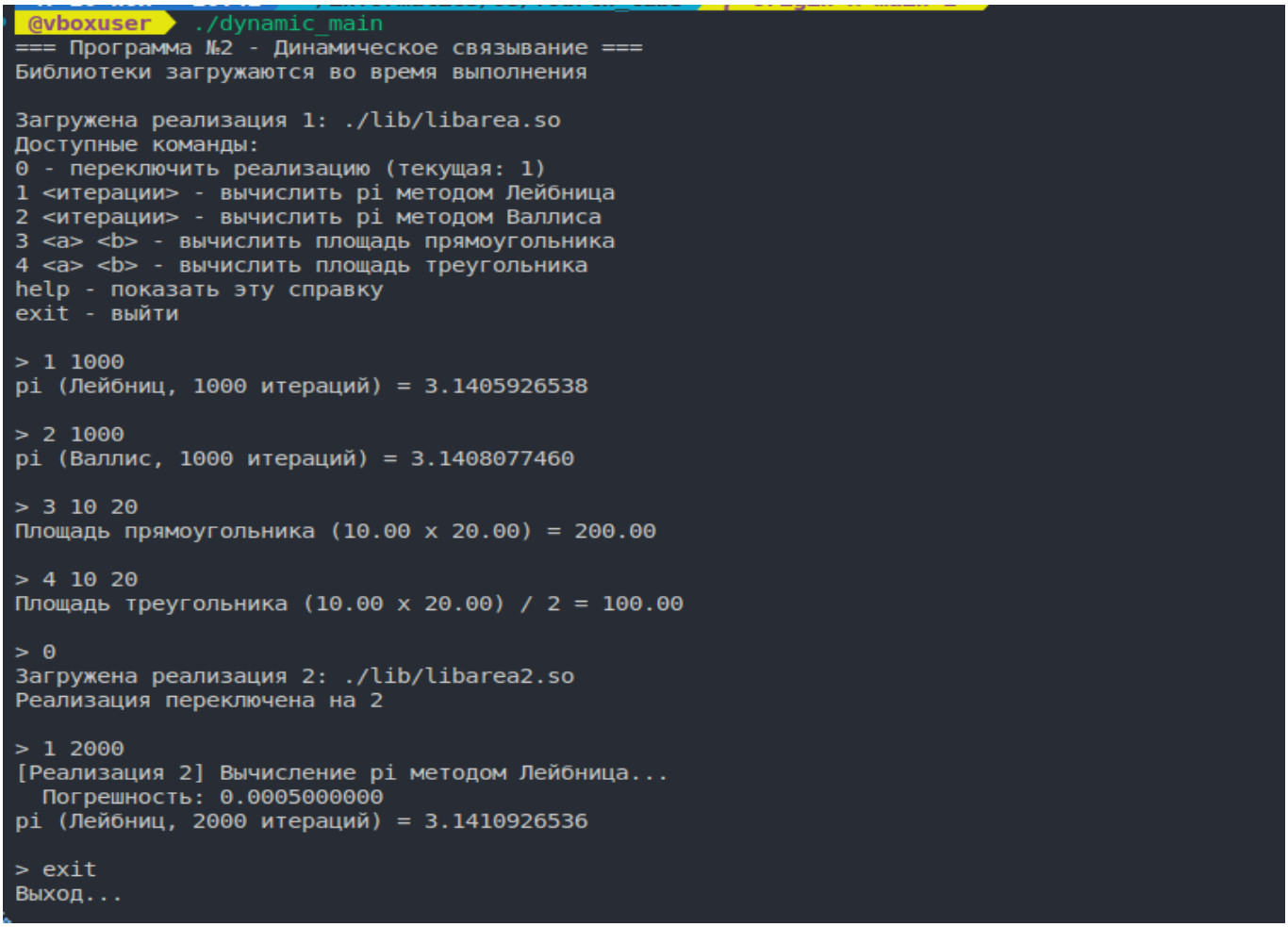
```
{
printf("Не удалось загрузить библиотеку по умолчанию\n");
return 1;
}

print_help();

while (1)
{
printf("\n> ");

if (fgets(input, sizeof(input), stdin) == NULL)
{
break;
}

input[strcspn(input, "\n")] = 0;

if (strlen(input) == 0)
{
continue;
}

int args_parsed = sscanf(input, "%9s", command);

if (args_parsed != 1)
{
continue;
}

if (strcmp(command, "exit") == 0)
{
printf("Выход...\n");
break;
}
else if (strcmp(command, "help") == 0)
{
print_help();
}
else if (strcmp(command, "0") == 0)
{
switch_implementation();
}
else if (strcmp(command, "1") == 0)
{
int iterations;
if (sscanf(input, "%*s %d", &iterations) == 1)
{
handle_pi_leibniz(iterations);
}
```

```
else
{
printf("Ошибка: укажите количество итераций\n");
printf("Использование: 1 <итерации>\n");
}
}
else if (strcmp(command, "2") == 0)
{
int iterations;
if (sscanf(input, "%*s %d", &iterations) == 1)
{
handle_pi_wallis(iterations);
}
else
{
printf("Ошибка: укажите количество итераций\n");
printf("Использование: 2 <итерации>\n");
}
}
else if (strcmp(command, "3") == 0)
{
float a, b;
if (sscanf(input, "%*s %f %f", &a, &b) == 2)
{
handle_rectangle_area(a, b);
}
else
{
printf("Ошибка: укажите две стороны\n");
printf("Использование: 3 <a> <b>\n");
}
}
else if (strcmp(command, "4") == 0)
{
float a, b;
if (sscanf(input, "%*s %f %f", &a, &b) == 2)
{
handle_triangle_area(a, b);
}
else
{
printf("Ошибка: укажите две стороны\n");
printf("Использование: 4 <a> <b>\n");
}
}
else
{
printf("Неизвестная команда: %s\n", command);
printf("Введите 'help' для справки\n");
}
}
```

```
if (current_lib.handle)
{
dlclose(current_lib.handle);
}

return 0;
}
```

Протокол работы программы





Вывод

В ходе лабораторной работы были успешно освоены механизмы работы с динамическими библиотеками в ОС Linux. Реализованы две программы, демонстрирующие различные подходы к использованию библиотек: статическое связывание на этапе компиляции и динамическая загрузка во время выполнения.