Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-216БВ-24

Студент: Лукиянчук А.О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.12.25

Москва, 2025

Постановка задачи

Вариант 16.

Требуется создать две динамические библиотеки, реализующие один и тот же интерфейс (контракт), но с различными алгоритмами обработки данных.

Контракты и реализации:

1. Подсчёт количества простых чисел на отрезке [a, b] (a, b – натуральные):

   Реализация №1: Наивный алгоритм. Проверить делимость текущего числа на все предыдущие числа.

   Реализация №2: Решето Эратосфена.

2. Подсчёт наибольшего общего делителя для двух натуральных чисел:

   Реализация №1: Алгоритм Евклида.

   Реализация №2: Наивный алгоритм: пытаться разделить числа на все числа, что меньше a и b.

Необходимо разработать две программы:

1. Программа №1: Использует одну из библиотек, связывание происходит на этапе компиляции. Программа жёстко зависит от наличия библиотеки при запуске.
2. Программа №2: Загружает библиотеки во время выполнения (runtime) по их отно- сительным путям. Программа должна поддерживать переключение между реализациями по команде пользователя без перезапуска.

Взаимодействие с пользователем осуществляется через консоль. Формат команд:

1. “0” – переключить реализацию (только для программы №2)
2. “1 a b” – подсчёт простых чисел на отрезке [a, b]
3. “2 a b” – вычисление НОД(a, b)
4. “q” – выход из программы

Общий метод и алгоритм решения

Использованные системные вызовы и функции:

- void* dlopen(const char* filename, int flags); – открывает динамическую библиотеку и возвращает дескриптор (handle).
- void* dlsym(void* handle, const char* symbol); – возвращает адрес символа (функции) в памяти загруженной библиотеки.

- int dlclose(void* handle); – уменьшает счётчик ссылок на библиотеку и выгружает её, если счётчик равен 0.
- char* dlerror(void); – возвращает текстовое описание последней ошибки, возникшей в функциях dl*.

Алгоритм работы:

Создание динамических библиотек:

- Созданы два файла исходного кода библиотек: libprime_gcd_impl1.c и libprime_gcd_impl2
- Оба файла компилируются с флагами -fPIC (позиционно-независимый код) и -shared для создания динамических библиотек libprime_gcd_impl1.so и libprime_gcd_impl2.so.
- Обе библиотеки реализуют одинаковый интерфейс, объявленный в заголовочном файле libprime_gcd.h.

Программа №1 (статическое связывание):

- Программа компилируется с явным указанием библиотеки libprime_gcd_impl1.so.
- Используется флаг линковщика -Wl, -rpath для указания загрузчику искать библиотеку в текущей директории.
- Функции библиотеки вызываются напрямую через заголовочный файл.
- При попытке переключения реализации программа информирует пользователя о фиксированной реализации.

Программа №2 (динамическая загрузка):

- Программа не линкуется с библиотеками на этапе компиляции.
- При запуске загружается первая библиотека с помощью dlopen().
- С помощью dlsym() получаются указатели на функции prime_count и gcd.
- По команде “0” текущая библиотека выгружается (dlclose()) и загружается альтернативная реализация.
- Все вызовы функций происходят через полученные указатели.
- Обрабатываются ошибки загрузки библиотек с помощью dlerror().

Алгоритмы реализаций:

- prime_count – наивный алгоритм проверяет каждое число на отрезке на делимость на все числа от 2 до n-1. Решето Эратосфена создаёт массив флагов и вычёркивает составные числа.
- gcd – алгоритм Евклида использует итеративное деление с остатком. Наивный ал- горитм перебирает все возможные делители от минимального числа вниз.

Код программы

[src/dynamic_program.c]

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

typedef int (*prime_count_func)(int, int);
typedef int (*gcd_func)(int, int);

int main() {
    void* lib_handle = NULL;
    prime_count_func prime_count = NULL;
    gcd_func gcd = NULL;

    int current_lib = 1;
    char input[256];

    lib_handle = dlopen("./libprime_gcd_impl1.so", RTLD_LAZY);
    if (!lib_handle) {
        fprintf(stderr, "Ошибка загрузки библиотеки 1: %s\n", dlerror());
        return 1;
    }

    prime_count = (prime_count_func)dlsym(lib_handle, "prime_count");
    gcd = (gcd_func)dlsym(lib_handle, "gcd");

    if (!prime_count || !gcd) {
        fprintf(stderr, "Ошибка загрузки функций: %s\n", dlerror());
        dlclose(lib_handle);
        return 1;
    }

    printf("Программа с динамической загрузкой библиотек\n");
    printf("Текущая реализация: %d\n", current_lib);
    printf("Команды:\n");
    printf("  0 - переключить реализацию\n");
    printf("  1 a b - подсчёт простых чисел на отрезке [a, b]\n");
    printf("  2 a b - НОД(a, b)\n");
    printf("  q - выход\n");

    while (1) {
        printf("\n> ");
        if (!fgets(input, sizeof(input), stdin)) break;

        if (input[0] == 'q') break;
```

```
        if (input[0] == '0') {
            dlclose(lib_handle);

            if (current_lib == 1) {
                lib_handle = dlopen("./libprime_gcd_impl2.so", RTLD_LAZY);
                current_lib = 2;
            } else {
                lib_handle = dlopen("./libprime_gcd_impl1.so", RTLD_LAZY);
                current_lib = 1;
            }

            if (!lib_handle) {
                fprintf(stderr, "Ошибка загрузки библиотеки: %s\n", dlerror());
                return 1;
            }

            prime_count = (prime_count_func)dlsym(lib_handle, "prime_count");
            gcd = (gcd_func)dlsym(lib_handle, "gcd");

            if (!prime_count || !gcd) {
                fprintf(stderr, "Ошибка загрузки функций: %s\n", dlerror());
                dlclose(lib_handle);
                return 1;
            }

            printf("Переключено на реализацию %d\n", current_lib);
            continue;
        }

        int command, a, b;
        if (sscanf(input, "%d %d %d", &command, &a, &b) == 3) {
            switch (command) {
                case 1: {
                    int result = prime_count(a, b);
                    if (result >= 0) {
                        printf("Количество простых чисел на отрезке [%d, %d]: %d\n", a, b,
result);
                    } else {
                        printf("Ошибка вычисления\n");
                    }
                    break;
                }
                case 2: {
                    int result = gcd(a, b);
                    printf("НОД(%d, %d) = %d\n", a, b, result);
                    break;
                }
                default:
```

```
                    printf("Неизвестная команда\n");
                }
            } else {
                printf("Неверный формат команды\n");
            }
        }

        if (lib_handle) {
            dlclose(lib_handle);
        }

        return 0;
}
[src/libprime_gcd_impl2.c]
#include "../include/libprime_gcd.h"
#include <stdlib.h>
#include <string.h>

int prime_count(int a, int b) {
    if (b < 2 || a > b) return 0;
    if (a < 2) a = 2;

    int size = b - a + 1;
    char* is_prime = (char*)malloc(size);
    if (!is_prime) return -1;

    memset(is_prime, 1, size);

    for (int p = 2; p * p <= b; p++) {
        int start = ((a + p - 1) / p) * p;
        if (start < p * p) start = p * p;

        for (int i = start; i <= b; i += p) {
            if (i >= a) {
                is_prime[i - a] = 0;
            }
        }
    }

    int count = 0;
    for (int i = 0; i < size; i++) {
        if (is_prime[i]) {
            count++;
        }
    }

    free(is_prime);
    return count;
```

```
}

int gcd(int a, int b) {
    int min = a < b ? a : b;
    for (int i = min; i >= 1; i--) {
        if (a % i == 0 && b % i == 0) {
            return i;
        }
    }
    return 1;
}
[src/libprime_gcd_impl1.c]
#include "../include/libprime_gcd.h"

static int is_prime_naive(int n) {
    if (n < 2) return 0;
    for (int i = 2; i < n; i++) {
        if (n % i == 0) return 0;
    }
    return 1;
}

int prime_count(int a, int b) {
    int count = 0;
    for (int i = a; i <= b; i++) {
        if (is_prime_naive(i)) {
            count++;
        }
    }
    return count;
}

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
[src/static_program.c]
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../include/libprime_gcd.h"

int main() {
    char input[256];
```

```
    int command;
    int lib_loaded = 1;

    printf("Программа со статической линковкой\n");
    printf("Используется реализация %d\n", lib_loaded);
    printf("Команды:\n");
    printf("  0 - переключить реализацию\n");
    printf("  1 a b - подсчёт простых чисел на отрезке [a, b]\n");
    printf("  2 a b - НОД(a, b)\n");
    printf("  q - выход\n");

    while (1) {
        printf("\n> ");
        if (!fgets(input, sizeof(input), stdin)) break;

        if (input[0] == 'q') break;

        if (input[0] == '0') {
            printf("В статически слинкованной программе реализация фиксирована на этапе
компиляции.\n");
            printf("Текущая реализация: %d\n", lib_loaded);
            continue;
        }

        int a, b;
        if (sscanf(input, "%d %d %d", &command, &a, &b) == 3) {
            switch (command) {
                case 1: {
                    int result = prime_count(a, b);
                    if (result >= 0) {
                        printf("Количество простых чисел на отрезке [%d, %d]: %d\n", a, b,
result);
                    } else {
                        printf("Ошибка вычисления\n");
                    }
                    break;
                }
                case 2: {
                    int result = gcd(a, b);
                    printf("НОД(%d, %d) = %d\n", a, b, result);
                    break;
                }
                default:
                    printf("Неизвестная команда\n");
            }
        } else {
            printf("Неверный формат команды\n");
        }
    }
```

```
    return 0;
}
[include/libprime_gcd.h]
#ifndef LIBPRIME_GCD_H
#define LIBPRIME_GCD_H

#ifdef __cplusplus
extern "C" {
#endif

int prime_count(int a, int b);
int gcd(int a, int b);

#ifdef __cplusplus
}
#endif

#endif
[Makefile]
CFLAGS = -Wall -Wextra -I./include
LDFLAGS = -ldl
BUILD_DIR = build
LIB_DIR = $(BUILD_DIR)

all: $(BUILD_DIR) libs programs

$(BUILD_DIR):
	mkdir -p $(BUILD_DIR)

libs: $(BUILD_DIR)/libprime_gcd_impl1.so $(BUILD_DIR)/libprime_gcd_impl2.so

$(BUILD_DIR)/libprime_gcd_impl1.so: src/libprime_gcd_impl1.c include/libprime_gcd.h
	$(CC) $(CFLAGS) -fPIC -shared -o $@ src/libprime_gcd_impl1.c

$(BUILD_DIR)/libprime_gcd_impl2.so: src/libprime_gcd_impl2.c include/libprime_gcd.h
	$(CC) $(CFLAGS) -fPIC -shared -o $@ src/libprime_gcd_impl2.c

programs: $(BUILD_DIR)/static_program $(BUILD_DIR)/dynamic_program

$(BUILD_DIR)/static_program: src/static_program.c $(BUILD_DIR)/libprime_gcd_impl1.so
	$(CC) $(CFLAGS) -o $@ src/static_program.c \
		-L$(BUILD_DIR) -lprime_gcd_impl1 \
		-Wl,-rpath,'$$ORIGIN'

$(BUILD_DIR)/dynamic_program: src/dynamic_program.c
	$(CC) $(CFLAGS) -o $@ src/dynamic_program.c $(LDFLAGS) \
		-Wl,-rpath,'$$ORIGIN'
```

```
test: all
	@echo "=== Тест статической программы ==="
	$(BUILD_DIR)/static_program <<< "1 1 20"
	@echo "\n=== Тест динамической программы ==="
	cd $(BUILD_DIR) && ./dynamic_program <<< "1 1 20"

run-static: all
	LD_LIBRARY_PATH=$(BUILD_DIR) $(BUILD_DIR)/static_program

run-dynamic: all
	cd $(BUILD_DIR) && LD_LIBRARY_PATH=. ./dynamic_program

clean:
	rm -rf $(BUILD_DIR)

install: libs
	sudo cp $(BUILD_DIR)/libprime_gcd_impl*.so /usr/local/lib/
	sudo cp include/libprime_gcd.h /usr/local/include/
	sudo ldconfig

.PHONY: all libs programs test clean install run-static run-dynamic
```

Протокол работы программы

Тестирование:

```
mkdir -p build
cc -Wall -Wextra -I./include -fPIC -shared -o build/libprime_gcd_impl1.so
src/libprime_gcd_impl1.c
cc -Wall -Wextra -I./include -fPIC -shared -o build/libprime_gcd_impl2.so
src/libprime_gcd_impl2.c
cc -Wall -Wextra -I./include -o build/static_program src/static_program.c \
	-Lbuild -lprime_gcd_impl1 \
	-Wl,-rpath,'$ORIGIN'
cc -Wall -Wextra -I./include -o build/dynamic_program src/dynamic_program.c -ldl \
	-Wl,-rpath,'$ORIGIN'
➜ lab4 cd build/
➜ build ./static_program
Программа со статической линковкой
Используется реализация 1
Команды:
  0 - переключить реализацию
  1 a b - подсчёт простых чисел на отрезке [a, b]
  2 a b - НОД(a, b)
  q - выход

> 1 1 100
```

```
Количество простых чисел на отрезке [1, 100]: 25

> 2 12345 67890
НОД(12345, 67890) = 15

> 0
В статически слинкованной программе реализация фиксирована на этапе компиляции.
Текущая реализация: 1

> q
➜ build ./dynamic_program
Программа с динамической загрузкой библиотек
Текущая реализация: 1
Команды:
  0 - переключить реализацию
  1 a b - подсчёт простых чисел на отрезке [a, b]
  2 a b - НОД(a, b)
  q - выход

> 1 100
Неверный формат команды

> 1 1 100
Количество простых чисел на отрезке [1, 100]: 25

> 2 12345 67890
НОД(12345, 67890) = 15

> 0
Переключено на реализацию 2

> 1 1 100
Количество простых чисел на отрезке [1, 100]: 25

> 2 12345 67890
НОД(12345, 67890) = 15

> q
➜ build strace -e trace=open,mmap,close,munmap,brk,fork,clone ./dynamic_program
[15:13:03]
brk(NULL)                               = 0x55b88c6e7000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f698b1d7000
mmap(NULL, 166763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f698b1ae000
close(3)                                = 0
mmap(NULL, 2169904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f698ae00000
mmap(0x7f698ae24000, 1511424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x24000) = 0x7f698ae24000
mmap(0x7f698af95000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x195000) =
```

```
0x7f698af95000
mmap(0x7f698b004000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x203000) = 0x7f698b004000
mmap(0x7f698b00a000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) =
0x7f698b00a000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f698b1ab000
munmap(0x7f698b1ae000, 166763)          = 0
brk(NULL)                               = 0x55b88c6e7000
brk(0x55b88c708000)                     = 0x55b88c708000
mmap(NULL, 16400, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f698b1d2000
mmap(0x7f698b1d3000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000)
0x7f698b1d3000
mmap(0x7f698b1d4000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7f698b1d4000
mmap(0x7f698b1d5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000)
0x7f698b1d5000
close(3)                                = 0
Программа с динамической загрузкой библиотек
Текущая реализация: 1
Команды:
  0 - переключить реализацию
  1 a b - подсчёт простых чисел на отрезке [a, b]
  2 a b - НОД(a, b)
  q - выход

> 0
munmap(0x7f698b1d2000, 16400)           = 0
mmap(NULL, 16424, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f698b1d2000
mmap(0x7f698b1d3000, 4096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1000)
0x7f698b1d3000
mmap(0x7f698b1d4000, 4096, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) =
0x7f698b1d4000
mmap(0x7f698b1d5000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000)
0x7f698b1d5000
close(3)                                = 0
Переключено на реализацию 2

> q
munmap(0x7f698b1d2000, 16424)           = 0
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы были изучены и применены механизмы работы с динамическими библиотеками в операционной системе Linux. Были созданы две дина-

мические библиотеки, реализующие одинаковый интерфейс с различными алгоритмами для подсчёта простых чисел и вычисления наибольшего общего делителя.

Были разработаны две программы, демонстрирующие различные подходы к использованию библиотек:

Статическое связывание: Библиотека линкуется на этапе компиляции, что обес- печивает более высокую производительность за счёт отсутствия накладных расходов на загрузку в runtime, но лишает гибкости.

Динамическая загрузка: Библиотеки загружаются во время выполнения про- граммы с использованием функций dlopen(), dlsym(), dlclose(), что позволяет переключаться между реализациями без перезапуска программы.

В процессе работы были изучены особенности компиляции динамических библиотек с флагами -fPIC и -shared, а также способы указания путей к библиотекам с помощью флага -Wl,-rpath.

Возникшие сложности включали необходимость корректной обработки ошибок при загрузке библиотек и обеспечение независимости разных реализаций алгоритмов. Эти проблемы были успешно решены с использованием функции dlerror() и тщательным тестированием граничных случаев.

Работа позволила получить практические навыки создания и использования динамических библиотек, что является важным для разработки модульных и расширяемых приложений в современных операционных системах. Была продемонстрирована разница в производительности между наивным алгоритмом и решетом Эратосфена, что подчерки- вает важность выбора правильных алгоритмов при реализации библиотечных функций.