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-212БВ-24

Студент: Федосов Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 28.11.25

Москва, 2025

Постановка задачи

Вариант 5.

Требуется создать динамические библиотеки, которые реализуют заданный вариантом функционал. Далее использовать данные библиотеки 2-мя способами:
1. Во время компиляции (на этапе линковки/linking)
2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками
В конечном итоге, в лабораторной работе необходимо получить следующие части:
● Динамические библиотеки, реализующие контракты, которые заданы вариантом;
● Тестовая программа (программа №1), которая используют одну из библиотек, используя информацию полученные на этапе компиляции;
● Тестовая программа (программа №2), которая загружает библиотеки, используя только их относительные пути и контракты.
Провести анализ двух типов использования библиотек.
Пользовательский ввод для обоих программ должен быть организован следующим образом:
● Если пользователь вводит команду «0», то программа переключает одну реализацию контрактов на другую (необходимо только для программы №2). Можно реализовать лабораторную работу без данной функции, но максимальная оценка в этом случае будет «хорошо»;
● “1 arg1 arg2 … argN”, где после “1” идут аргументы для первой функции, предусмотренной контрактами. После ввода команды происходит вызов первой функции, и на экране появляется результат ее выполнения;
● “2 arg1 arg2 … argM”, где после “2” идут аргументы для второй функции, предусмотренной контрактами. После ввода команды происходит вызов второй функции, и на экране появляется результат ее выполнения.

Общий метод и алгоритм решения

Использованные новые функции:

`dlopen()` - Загружает указанную .so библиотеку в память и возвращает её дескриптор.

`dlsym()` - Используя дескриптор, находит адрес функции или переменной по её имени (строке) и возвращает указатель.

`dlclose()` - Выгружает библиотеку, если она больше не используется другими частями программы.

Алгоритм решения:

Программа №1 (stat.c) — Статическая Компоновка:
Эта программа использует функции из динамических библиотек, но связывает их на этапе компиляции.

- Чтение ввода: Программа в бесконечном цикле читает команды из стандартного ввода (stdin) с помощью низкоуровневого системного вызова read().
- Парсинг команды: Используется функция sscanf для быстрого определения, какую команду (1 или 2) и с какими аргументами ввёл пользователь.
- Обработка команды:

а. Если команда 1, программа напрямую вызывает функцию sin_integral(a, b, step).
б. Если команда 2, программа напрямую вызывает функцию e(x).

- Выход: При вводе q цикл прерывается.

Во время компиляции компоновщик записывает в исполняемый файл, что ему нужны функции sin_integral и e из библиотек libfunc_sin_1.so и libfunc_e_1.so. Перед запуском операционная система (динамический загрузчик) должна найти эти библиотеки и подгрузить их в память.

Программа №2 (dynam.c) - Динамическая Компоновка:

- Чтение ввода: Аналогично, программа читает команды из stdin через read().
- Инициализация: В начале main вызывается switch_realization(), которая с помощью dlopen() загружает библиотеки версии 1 (libfunc_sin_1.so и libfunc_e_1.so) и получает указатели на функции (curr_sin_func, curr_e_func) с помощью dlsym().
- Обработка команды 0 (Переключение): При вводе 0 программа вызывает switch_realization(). Эта функция сначала выгружает текущие библиотеки (dlclose()). Затем она определяет новую версию и загружает соответствующие ей .so файлы (например, libfunc_sin_2.so) через dlopen(). А потом получает новые указатели на функции через dlsym().
- Обработка команд 1 и 2 (Вычисление): Команды парсятся с помощью sscanf. Вместо прямого вызова, программа использует указатели на функции (curr_sin_func(a, b, step) или curr_e_func(x)), которые указывают на версию, загруженную в данный момент.
- Выход: При выходе (q) программа обязательно вызывает dlclose() для освобождения всех загруженных библиотек из памяти.

Код программы

stat.c:

```c
#include <unistd.h>

#include <string.h>

#include "functions.h"
```

```c
#include <stdio.h>


// 1 <a> <b> <step> = function 1

// 2 <x> = function 2

// q = end


int main() {

   char buff[1024];

   int bytes;


   while ((bytes = read(STDIN_FILENO, buff, sizeof(buff) - 1)) > 0) {


       buff[bytes] = '\0';


       if (buff[bytes - 1] == '\n') {

           buff[bytes - 1] = '\0';

       }


       int command;

       float a, b, step;

       int x;

       int readAmount;


       if (strcmp(buff, "q") == 0) {

           break;

       }
```

```
        readAmount = sscanf(buff, "%d %f %f %f", &command, &a,
&b, &step);


        if (command == 1 && readAmount == 4) {
            float res = sin_integral(a, b, step);
            char output[256];
            int len = snprintf(output, sizeof(output) - 1,
"Результат (sin): %.10f \n", res);
            write(STDOUT_FILENO, output, len);
        }


        readAmount = sscanf(buff, "%d %d", &command, &x);


        if (command == 2 && readAmount == 2) {
            float res = e(x);
            char output[256];
            int len = snprintf(output, sizeof(output) - 1,
"Результат (e): %.10f \n", res);
            write(STDOUT_FILENO, output, len);
        }
    }


    return 0;
}
```

dynam.c:

```
#include <unistd.h>
```

```c
#include <stdio.h>

#include <string.h>

#include <dlfcn.h>


// 1 <a> <b> <step> = function 1

// 2 <x> = function 2

// q = end

// 0 = switch


typedef float (*sin_t)(float, float, float);

typedef float (*e_t)(int);


static sin_t curr_sin_func = NULL;

static e_t curr_e_func = NULL;

static int curr_version = 2;


static void *loaded_sin = NULL;

static void *loaded_e = NULL;


const char *SIN_FUNC_1_NAME = "./libfunc_sin_1.so";

const char *SIN_FUNC_2_NAME = "./libfunc_sin_2.so";


const char *E_FUNC_1_NAME = "./libfunc_e_1.so";

const char *E_FUNC_2_NAME = "./libfunc_e_2.so";


int load_func(void **library, void **function, const char
*lib_name, const char *func_name)
```

```
{
    *library = dlopen(lib_name, RTLD_LAZY);
    if (*library == NULL) {
        const char msg[] = "Ошибка: dlopen. \n";
        write(STDERR_FILENO, msg, sizeof(msg));
        return 0;
    }

    *function = dlsym(*library, func_name);
    if (*function == NULL) {
        const char msg[] = "Ошибка: dlsym. \n";
        write(STDERR_FILENO, msg, sizeof(msg));
        return 0;
    }

    return 1;
}

int switch_realization()
{
    if (loaded_sin) {
        dlclose(loaded_sin);
    }
    if (loaded_e) {
        dlclose(loaded_e);
    }
```

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

    const char *sin_lib_name = (curr_version == 1) ?
SIN_FUNC_1_NAME : SIN_FUNC_2_NAME;

    const char *e_lib_name = (curr_version == 1) ? E_FUNC_1_NAME
: E_FUNC_2_NAME;




    int success_sin = load_func(&loaded_sin, (void
**)&curr_sin_func, sin_lib_name, "sin_integral");

    int success_e = load_func(&loaded_e, (void **)&curr_e_func,
e_lib_name, "e");


    if (success_e && success_sin) {

        char output[256];

        int bytes = snprintf(output, sizeof(output) - 1,
"Переключено на реализацию №%d. \n", curr_version);

        write(STDOUT_FILENO, output, bytes);

        return 1;


    } else {

        const char msg[] = "Ошибка: не удалось поменять
реализацию контрактов. \n";

        write(STDERR_FILENO, msg, sizeof(msg));

        curr_e_func = NULL;

        curr_sin_func = NULL;

        return 0;

    }

}
```

```c
int main() {
    char buff[1024];
    int bytes;

    switch_realization();

    while ((bytes = read(STDIN_FILENO, buff, sizeof(buff) - 1)) > 0) {

        buff[bytes] = '\0';

        if (buff[bytes - 1] == '\n') {
            buff[bytes - 1] = '\0';
        }

        int command;
        float a, b, step;
        int x;
        int readAmount;

        if (strcmp(buff, "q") == 0) {
            break;
        }

        if (strcmp(buff, "0") == 0) {
            switch_realization();
            continue;
```

```
        }


        readAmount = sscanf(buff, "%d %f %f %f", &command, &a,
&b, &step);


        if (command == 1 && readAmount == 4) {
            if (!curr_sin_func) {
                if (loaded_sin) {
                    dlclose(loaded_sin);
                }
                if (loaded_e) {
                    dlclose(loaded_e);
                }
                const char msg[] = "Ошибка: не удалось загрузить
функцию. \n";
                write(STDERR_FILENO, msg, sizeof(msg));
                curr_e_func = NULL;
                return 0;
            }
            float res = curr_sin_func(a, b, step);
            char output[256];
            int len = snprintf(output, sizeof(output) - 1,
"Результат (sin): %.10f \n", res);
            write(STDOUT_FILENO, output, len);
        }


        readAmount = sscanf(buff, "%d %d", &command, &x);
```

```
        if (command == 2 && readAmount == 2) {
            if (!curr_e_func) {
                if (loaded_sin) {
                    dlclose(loaded_sin);
                }
                if (loaded_e) {
                    dlclose(loaded_e);
                }
                const char msg[] = "Ошибка: не удалось загрузить функцию. \n";
                write(STDERR_FILENO, msg, sizeof(msg));
                curr_sin_func = NULL;
                return 0;
            }
            float res = curr_e_func(x);
            char output[256];
            int len = snprintf(output, sizeof(output) - 1, "Результат (e): %.10f \n", res);
            write(STDOUT_FILENO, output, len);
        }
    }


    if (loaded_sin) {
        dlclose(loaded_sin);
    }
    if (loaded_e) {
        dlclose(loaded_e);
    }
```

```
    return 0;
}
```

func_e_1.c:

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


float e(int x)
{
    if (x <= 0) {
        return 1.0;
    }

    return powf(1.0 + (1.0 / (float)x), (float)x);
}
```

func_e_2.c:

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


long long fact(int n)
{
    if (n < 0) {
        return 0;
```

```
    }

    if (n == 0 || n == 1) {
        return 1;
    }

    long long res = 1;
    for (int i = 2; i < n; i++) {
        res *= i;
    }

    return res;
}

float e(int x)
{
    if (x < 0) {
        return 0.0;
    }

    float res = 0.0;

    for (int n = 0; n <= x; n++) {
        res += 1.0 / (float)fact(n);
    }

    return res;
```

```
}
```

func_sin_1.c:

```
#include "functions.h"

#include <math.h>


float sin_integral(float a, float b, float e)

{

   if (a > b) {

       return -sin_integral(b, a, e);

   }


   float res = 0.0;


   for (float x = a; x < b; x += e) {

       res += sinf(x) * e;

   }


   return res;

}
```

func_sin_2.c:

```
#include "functions.h"

#include <math.h>


float sin_integral(float a, float b, float e)

{
```

```
    if (a > b) {
        return -sin_integral(b, a, e);
    }

    float res = 0.0;

    for (float x = a; x < b; x += e) {
        res += (sinf(x) + sinf(x + e)) * e / 2.0;
    }

    return res;
}
```

functions.h:

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

float sin_integral(float a, float b, float e);

float e(int x);

#endif
```

Протокол работы программы

```
snadon@fedora:~/OS-Labs/Lab4$ ./static
1 0 1 0.01
Результат (sin): 0.4639011323
1 0 1 0.00001
Результат (sin): 0.4591229558
2 5
Результат (e): 2.4883205891
2 10
Результат (e): 2.5937430859
q
snadon@fedora:~/OS-Labs/Lab4$ ./dynamic
Переключено на реализацию №1.
1 0 1 0.1
Результат (sin): 0.4172410369
1 0 1 0.000001
Результат (sin): 0.4541524947
2 3
Результат (e): 2.3703706264
2 10
Результат (e): 2.5937430859
0
Переключено на реализацию №2.
1 0 1 0.1
Результат (sin): 0.4593146145
1 0 1 0.000001
Результат (sin): 0.4541530013
2 3
Результат (e): 3.5000000000
2 10
Результат (e): 3.7182817459
q
```

Вывод

В процессе выполнения лабораторной работы я составил программу, демонстрирующую использование статических и динамических библиотек. Я приобрел базовые навыки создания библиотек и их правильного использования в коде. Одной из основных сложностей была в импорте динамических библиотек.

Таким образом, мы можем сделать выводы: статическая компоновка хоть и может быть автономнее динамической, но явно уступает в расходе памяти и размере. Также нужно учитывать сложность обновления при статической компоновке. Динамическая компоновка хоть и показалась мне сложной, однако имеет больше преимуществ, по сравнению со статической.