Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

Студент: Воробьева К.Н.
Группа: М8О-201Б-21
Вариант: №18
Преподаватель: Миронов Е.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## 1. Цель работы

Целью является приобретение практических навыков в:

- Создание динамических библиотек
- Создание программ, которые используют функции динамических библиотек

## 2. Постановка задачи

Требуется создать динамические библиотеки, которые реализуют определенный 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 знания полученные на этапе компиляции;
- Тестовая программа (программа №2), которая загружает библиотеки, используя только их местоположение и контракты.

*Провести анализ двух типов использования библиотек.*

Пользовательский ввод для обоих программ должен быть организован следующим образом:

1. 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»;
2. «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 её выполнения;
3. «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 её выполнения.

**Вариант №18:**

| | | | | |
|---|---|---|---|---|
| 3 | Подсчёт количества простых чисел на отрезке [A, B] (A, B - натуральные) | Int PrimeCount(int A, int B) | Наивный алгоритм. Проверить делимость текущего числа на все предыдущие числа. | Решето Эратосфена |
| 6 | Рассчет значения числа e(основание натурального логарифма) | Float E(int x) | (1 + 1/x) ^ x | Сумма ряда по n от 0 до x, где элементы ряда равны: (1/(n!)) |

## 3. Общие сведения о программе

Проект состоит из пяти исходных файлов main.c, dynamic_main.c, lib.h, first_lib.c, second_lib.c. Первые два — это программы, в который тестируются функции из библиотек. Третий файл – интерфейс библиотек. И, наконец, 4 и 5 — это файлы, в которых хранятся реализации функции библиотек. В first_lib.c хранятся функции для Реализации 1, а в second_lib.c для Реализации 2.
В программе используются такие команды, как:

**dlopen(const char* filename, int flag)** - загружает динамическую библиотеку, имя которой мы передаем, и возвращает прямой указатель на начало динамической библиотеки.
**dlclose(void* handle) -** уменьшает на единицу счетчик ссылок на указатель динамической библиотеки handle. Если нет других загруженных библиотек, использующих ее символы и если счетчик ссылок handle принимает нулевое значение, то динамическая библиотека выгружается.
**dlsym(void* handle, char* symbol) -** использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий, откуда загружается этот символ.

## 4. Общий метод и алгоритм решения

Для начала я реализовала две библиотеки: first_lib и second_lib. В первой из них реализовано вычисление экспоненты с помощью 1 замечательного предела и подсчет количества простых чисел на отрезке наивным

алгоритмом, а во второй соответственно вычисление экспоненты с помощью ряда Тейлора и подсчет количества простых чисел на отрезке с помощью решета Эратосфена. Далее были реализованы еще две программы: main и dynamic_main, в которых использовались наши библиотеки, но в первой программе библиотека подключалась на этапе линковке, поэтому для проверки 2 библиотек используются 2 разные программы. А во второй мы использовали динамическую библиотеку, которая использовалась непосредственно во время исполнения программы, причем я также реализовала возможность изменения подключаемой библиотеки, поэтому для проверки двух библиотек необходима лишь одна программа.

## 5. Исходный код

### lib.h

```
#ifndef LIB_H
#define LIB_H

extern int PrimeCount(int start, int finish);
extern float E(int argument);

#endif //LIB_H
```

### first_lib.c

```
#include "lib.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int PrimeCount(int start, int finish) {
   printf("PrimeCount with naive algoritm\n");

   if (start == 1) {
      ++start;
   }

   int prime_count = 0;
   int is_prime;

   for (start; start <= finish; ++start) {
      is_prime = 1;
      for (int number = 2; number < start; ++number) {
         if (start % number == 0) {
            is_prime = 0;
            break;
         }
      }

      if (is_prime == 1) {
         ++prime_count;
      }
   }
```

```
    return prime_count;
}

float E(int argument) {
    printf("Calculating a E with first wonderful limit\n");

    if (argument < 0) {
        return -1;
    }

    float e = 1.0;

    for (int index = 0; index < argument; ++index) {
        e *= 1 + 1 / (float)argument;
    }

    return e;
}
```

## second_lib.c

```
#include "lib.h"
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

int PrimeCount(int start, int finish) {
    printf("PrimeCount with sieve of Eratosthenes\n");

    int curent_number;
    int* prime_number_list;
    int is_prime;
    finish = finish - 1;
    prime_number_list = (int*) malloc(finish * sizeof(int));

    for (int index = 0; index < finish; ++index) {
        prime_number_list[index] = index + 2;
    }

    for (int index = 0; index < finish; ++index) {
        curent_number = prime_number_list[index];
        is_prime = 0;

        for (int second_index = index + 1; second_index < finish; ++second_index) {
            if (!(prime_number_list[second_index] % curent_number)) {
                for (int third_index = second_index; third_index < finish - 1; ++third_index) {
                    prime_number_list[third_index] = prime_number_list[third_index + 1];
                }

                is_prime = 1;
                --finish;
                --second_index;
            }
        }

        if (is_prime == 0) {
            break;
        }
    }

    int prime_count = 0;
    for (int index = 0; index < finish; ++index) {
        if (prime_number_list[index] >= start) {
            ++prime_count;
        }
    }

    free(prime_number_list);

    return prime_count;
```

```
}

float E(int argument) {
    printf("Calculating a E by Taylor series\n");

    if (argument < 0) {
        return -1;
    }

    float e = 1.0;
    float term = 1.0;

    for (int index = 1; index <= argument; ++index) {
        term /= index;
        e += term;
    }

    return e;
}
```

## main.c

```
#include "lib.h"
#include <stdio.h>

int main() {
    int command = 0;
    printf("To compute count of prime number on [A, B] enter -- 1.Args: start finish\n");
    printf("To compute E enter -- 2.Args: argument\n");

    while (scanf("%d", &command) != EOF) {
        switch (command) {
            case 1: {
                int start, finish;

                if (scanf("%d %d", &start, &finish) == 2) {
                    printf("Count: %d\n", PrimeCount(start, finish));
                }

                break;
            }
            case 2: {
                int argument;

                if (scanf("%d", &argument) == 1) {
                    printf("E is: %f\n", E(argument));
                }

                break;
            }
            default: {
                printf("This command is not supported, enter 1 or 2\n");
            }
        }
    }

    return 0;
}
```

## dynamic_main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <math.h>
#include <dlfcn.h>
```

```
typedef enum {
   first_contract,
   second_contract,
} contracts;

contracts contract = first_contract;

const char* first_library_name = "lib1.so";
const char* second_library_name = "lib2.so";

int (*PrimeCount)(int, int) = NULL;
float (*E)(int) = NULL;

void* library_handle = NULL;

void load_library(contracts contract) {
   const char* library_name;

   switch (contract) {
      case first_contract: {
         library_name = first_library_name;
         break;
      }
      case second_contract: {
         library_name = second_library_name;
         break;
      }
   }

   library_handle = dlopen(library_name, RTLD_LAZY);

   if (library_handle == NULL) {
      perror("dlopen error");
      exit(EXIT_FAILURE);
   }
}

void load_contract() {
   load_library(contract);
   PrimeCount = dlsym(library_handle, "PrimeCount");
   E = dlsym(library_handle, "E");
}

void change_contract() {
   dlclose(library_handle);

   switch (contract) {
      case first_contract: {
         contract = second_contract;
         break;
      }
      case second_contract: {
         contract = first_contract;
         break;
      }
   }

   load_contract();
}

int main() {
   load_contract();
   int command = 0;
   printf("To compute count of prime number on [A, B] enter -- 1.Args: start finish\n");
   printf("To compute E enter -- 2.Args: argument\n");
   printf("To change contract enter -- 0\n");

   while (scanf("%d", &command) != EOF) {
      switch (command) {
         case 0: {
```

```
            change_contract();
            printf("Contract has been changed\n");

            switch (contract) {
                case first_contract: {
                    printf("Contract is first\n");
                    break;
                }
                case second_contract: {
                    printf("Contract is second\n");
                    break;
                }
            }

            break;
        }
        case 1: {
            int start, finish;
            if (scanf("%d %d", &start, &finish) == 2){
                printf("Count: %d\n", PrimeCount(start, finish));
            }

            break;
        }
        case 2:{
            int argument;
            if (scanf("%d", &argument) == 1){
                printf("E is: %f\n", E(argument));
            }

            break;
        }
        default:{
            printf("This command is not supported, enter 1 or 2 or 0\n");
        }
    }
  }

  return 0;
}
```

## 6. Демонстрация работы программы

## Тест:

karina@MSI:~/projects/OS/build/lab5$ ./first_static

*To compute count of prime number on [A, B] enter -- 1.Args: start finish*

*To compute E enter -- 2.Args: argument*

*1 6 29*

*PrimeCount with naive algoritm*

*Count: 7*

*2 4000*

*Calculating a E with first wonderful limit*

*E is: 2.717744*

karina@MSI:~/projects/OS/build/lab5$ ./second_static

*To compute count of prime number on [A, B] enter -- 1.Args: start finish*

*To compute E enter -- 2.Args: argument*

*1 6 29*

*PrimeCount with sieve of Eratosthenes*

*Count: 7*

*2 4000*

*Calculating a E by Taylor series*

*E is: 2.718282*

karina@MSI:~/projects/OS/build/lab5$ ./dynamic

*To compute count of prime number on [A, B] enter -- 1.Args: start finish*

*To compute E enter -- 2.Args: argument*

*To change contract enter -- 0*

*1 1 20*

*PrimeCount with naive algoritm*

*Count: 8*

*2 4000*

*Calculating a E with first wonderful limit*

*E is: 2.717744*

*0*

*Contract has been changed*

*Contract is second*

*1 1 20*

*PrimeCount with sieve of Eratosthenes*

*Count: 8*

*2 4000*

*Calculating a E by Taylor series*

*E is: 2.718282*

## 7. Выводы

Проделав лабораторную работу, я научилась создавать динамические библиотеки в ОС Linux, а также узнала, что их можно использовать двумя способами: во время работы программы и во время компиляции. Вообще библиотеки позволяют использовать разработанный ранее программный код в различных программах. У каждой библиотеки должен быть свой заголовочный файл, в котором должен быть описан ее интерфейс, то есть

должны быть объявлены все функции, содержащиеся в библиотеке. Библиотеки бывают двух видов – статические и динамические. Код первых при компиляции полностью входит в состав исполняемого файла, что делает программу легко переносимой. Код динамических библиотек не входит в исполняемый файл, последний содержит лишь ссылку на библиотеку. Если динамическая библиотека будет удалена или перемещена в другое место, то программа работать не будет. С другой стороны, использование динамических библиотек позволяет сократить размер исполняемого файла. Также, если в памяти находится две программы, использующие одну и туже динамическую библиотеку, то последняя будет загружена в память лишь единожды. Но при работе с динамическими библиотеками нельзя забывать, что они компилируются особым образом. Они должны содержать так называемый позиционно-независимый код (position independent code). Наличие такого кода позволяет библиотеке подключаться к программе, когда последняя загружается в память.