Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

**Динамические библиотеки**

Студент: Ползикова Алина Владимировна

Группа: М8О–208Б–21

Вариант: 8

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

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

## Задание

Требуется создать динамические библиотеки, которые реализуют определенный функционал. Далее использовать данные библиотеки двумя 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 следующее:

* Динамические библиотеки, реализующие контракты, которые заданы вариантом;
* Тестовая программа No1, которая использует одну из библиотек, используя знания, полученные на этапе компиляции;
* Тестовая программа No2, которая загружает библиотеки, используя 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 на другую (необходимо только для программы No2);
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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Описание | Сигнатура | Реализация 1 | Реализация 2 |
| 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 |
| 4 | Подсчёт наибольшего общего делителя для двух натуральных чисел | Int GCF(int A, int B) | Алгоритм Евклида | Наивный алгоритм. Пытаться разделить числа на все числа, что меньше A и B. |

**Общие сведения о программе**

Программа компилируется из файлов main.c, main\_dyn.c, realisation1.c, realisation2.c. Также используется заголовочные файлы: stdio.h, realisation.h, dlfcn.h. В программе используются следующие системные вызовы:

1. **dlopen()** – загружает (открывает) динамическую библиотеку. Возвращает указатель на загруженную библиотеку, в случае ошибки возвращает NULL;
2. **dlsym()** – получение адреса функции или переменной из библиотеки. Возвращает адрес функции, в случае ошибки возвращает NULL;
3. **dlerror()** – возвращает понятную человеку строку, описывающую последнюю ошибку, которая произошла при вызове одной из функции dlopen, dlsym, dlclose. Возвращает NULL если не возникло ошибок с момента инициализации или с момента ее последнего вызова;
4. **dlclose()** – уменьшает на единицу счетчик ссылок на указатель динамической библиотеки. Возвращает 0 при удачном завершении и ненулевой результат при ошибке;

**Общий метод и алгоритм решения**

Описываем решения в библиотечных файлах, создаём общий заголовочный файл. Нам не потребуется два, так как в обеих реализациях одни и те же функции, соответственно, между двумя заголовочными файлами не было бы различия. Далее собираем всё в исполняемый файл.

**Основные файлы программы**

====================== main\_dyn.c ======================

#include <dlfcn.h>

#include <stdio.h>

#include <stdlib.h>

#define check(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { printf("%s", MSG); return 1; }

#define check\_wrong(VALUE, WRONG\_VAL, MSG) if (VALUE == WRONG\_VAL) { printf("%s", MSG); return 1; }

// it is very important to make prefix "lib" and set extension ".so"

const char\* DYN\_LIB\_1 = "./libDyn1.so";

const char\* DYN\_LIB\_2 = "./libDyn2.so";

const char\* SIN\_INTEGRAL\_NAME = "SinIntegral";

const char\* SORT\_NAME = "Sort";

int main(int argc, const char\*\* argv) {

int dynLibNum = 1;

void\* handle = dlopen(DYN\_LIB\_1, RTLD\_LAZY);

check\_wrong(handle, NULL, "Error opening dynamic library!\n")

float (\*Integral)(float, float, float);

int\* (\*Sort)(int\*);

Integral = dlsym(handle, SIN\_INTEGRAL\_NAME);

Sort = dlsym(handle, SORT\_NAME);

char\* error = dlerror();

check(error, NULL, error)

int q;

size\_t size;

int\* array;

float A, B, e;

printf("Enter query: 0) change realisation 1) integral sin(x)dx 2) sort array\n");

while (scanf("%d", &q) > 0) {

switch (q) {

case 0:

check(dlclose(handle), 0, "Error closing dynamic library!\n")

if (dynLibNum) {

handle = dlopen(DYN\_LIB\_2, RTLD\_LAZY);

} else {

handle = dlopen(DYN\_LIB\_1, RTLD\_LAZY);

}

check\_wrong(handle, NULL, "Error opening dynamic library!\n")

Integral = dlsym(handle, SIN\_INTEGRAL\_NAME);

Sort = dlsym(handle, SORT\_NAME);

error = dlerror();

check(error, NULL, error)

/\* switch between 0 and 1 \*/

dynLibNum = dynLibNum ^ 1;

break;

case 1:

printf("enter A, B, e: ");

check(scanf("%f %f %f", &A, &B, &e), 3, "Error reading floats!\n")

printf("Integral from %f to %f sin(x)dx = %f\n", A, B, Integral(A, B, e));

break;

case 2:

printf("enter array size: ");

check(scanf("%lu", &size), 1, "Error reading array size!\n")

array = malloc((size + 1) \* sizeof(int));

if (array == NULL) {

printf("Cannot allocate memmory\n");

return 2;

}

array[0] = (int) size;

printf("enter elements: ");

for (int i = 1; i < size + 1; ++i) {

check(scanf("%d", array + i), 1, "Error reading integer\n")

}

array = Sort(array);

printf("Sorted array: ");

for (int i = 1; i < size + 1; ++i) {

printf("%d ", array[i]);

}

printf("\n");

free(array);

break;

default:

printf("End.\n");

check(dlclose(handle), 0, "Error closing dynamic library!\n")

return 0;

}

}

}

====================== main.c ======================

#include "realisation.h"

#include <stdio.h>

#include <stdlib.h>

#define check(VALUE, OKVAL, MSG) if (VALUE != OKVAL) { printf("%s", MSG); return 1; }

int main(int argc, const char\*\* argv) {

int q;

printf("Enter query: 1) integral sin(x)dx 2) sort array\n");

while (scanf("%d", &q) > 0) {

if (q == 1) {

float A, B, e;

printf("enter A, B, e: ");

check(scanf("%f %f %f", &A, &B, &e), 3, "Error reading floats!\n")

printf("Integral from %f to %f sin(x)dx = %f\n", A, B, SinIntegral(A, B, e));

} else if (q == 2) {

size\_t size;

int\* array;

printf("enter array size: ");

check(scanf("%lu", &size), 1, "Error reading array size!\n")

array = malloc((size + 1) \* sizeof(int));

if (array == NULL) {

printf("Cannot allocate memmory\n");

return 2;

}

array[0] = (int) size;

printf("enter elements: ");

for (int i = 1; i < size + 1; ++i) {

check(scanf("%d", array + i), 1, "Error reading integer\n")

}

array = Sort(array);

printf("Sorted array: ");

for (int i = 1; i < size + 1; ++i) {

printf("%d ", array[i]);

}

printf("\n");

free(array);

} else {

printf("End.\n");

return 0;

}

}

}

====================realisation1.c =====================

#include "realisation.h"

#include <math.h>

#include <stdio.h>

void swap\_int(int\* x, int\* y) {

int tmp = \*x;

\*x = \*y;

\*y = tmp;

}

float SinIntegral(float a, float b, float e) {

float square = 0;

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

square += e \* sin(i);

}

return square;

}

int\* Sort(int\* array) {

int size = array[0];

array++;

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size - 1; ++j) {

if (array[j] > array[j + 1]) {

swap\_int(array + j, array + j + 1);

}

}

}

return --array;

}

====================realisation2.c =====================

#include "realisation.h"

#include "math.h"

void swap\_int(int\* x, int\* y) {

int tmp = \*x;

\*x = \*y;

\*y = tmp;

}

float SinIntegral(float A, float B, float e){

float square = 0;

for (float i = A; i < B; i += e) {

square += e \* ((sin(i) + sin(i + e)) / 2);

}

return square;

}

void QuickSort(int\* array, int l, int r) {

if (l + 1 >= r) {

return;

}

int i = l, j = r - 2;

while (i < j) {

if (array[i] < array[r - 1]) {

++i;

} else if (array[j] > array[r - 1]) {

--j;

} else {

swap\_int(&array[i], &array[j]);

++i;

--j;

}

}

if (array[i] < array[r - 1]) {

++i;

}

swap\_int(array + i, array + r - 1);

QuickSort(array, l, i);

QuickSort(array, i + 1, r);

}

int\* Sort(int\* array) {

int size = array[0];

array++;

QuickSort(array, 0, size);

return --array;

}

====================realisation.h =====================

#ifndef LAB\_5\_REALISATION\_H

#define LAB\_5\_REALISATION\_H

float SinIntegral(float A, float B, float e);

int\* Sort(int\* array);

#endif //LAB\_5\_REALISATION\_H

======================== test1.txt ========================

1

167 85637

2

46578 465713

0

1

167 85637

2

46578 465713

5

======================== test2.txt ========================

1

100000 4635892

2

28372 35556

10

======================== test3.txt ========================

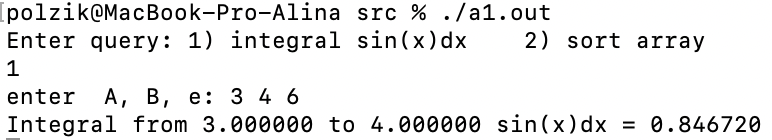
1

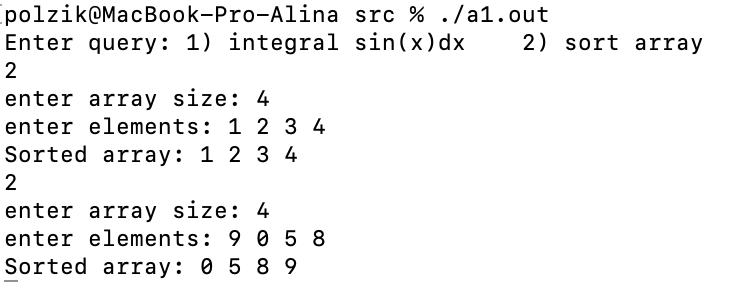
1 8888888

2

10000 100000

**Пример работы**

****

****

**Вывод**

Во время выполнения работы я изучила основы работы с динамическими библиотеками на операционных системах Linux, реализовала программу, которая использует созданные динамические библиотек. Выяснил некоторые различия в механизмах работы динамических и статических библиотек. Также я смогла сделать вывод что, использование библиотек добавляет модульность программе, что упрощает дальнейшую поддержку кода.