Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ**

Студент: Басова Татьяна Валентиновна

Группа: М8О–212Б–22

Вариант: 3

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

Цель работы

Целью является приобретение практических навыков в:

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

Задание

Требуется создать динамическую библиотеку, которая реализует определенный функционал. Далее использовать данную библиотеку 2-мя способами:

1. Во время компиляции (на этапе «линковки»/linking)
2. Во время исполнения программы, подгрузив библиотеку в память с помощью системных вызовов

В конечном итоге, программа должна состоять из следующих частей:

* Динамическая библиотека, реализующая заданных вариантом интерфейс;
* Тестовая программа, которая используют библиотеку, используя знания полученные на этапе компиляции;
* Тестовая программа, которая использует библиотеку, используя только местоположение динамической библиотеки и ее интерфейс.

Провести анализ между обоими типами использования библиотеки.

Вариант 16:

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) 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 linking.cpp и runtime.cpp. В программе используются следующие системные вызовы:

1. **dlopen** – загружает динамический общий объект (общую библиотеку) из файла, имя которого указано в строке filename (завершается null) и возвращает непрозрачный описатель на загруженный объект.
2. **dlsym** – функция возвращает адрес, по которому символ расположен в памяти(указывается одним из аргументов).
3. **dlclose** – уменьшает счётчик ссылок на динамически загружаемый общий объект, на который ссылается handle. Если счётчик ссылок достигает нуля, то объект выгружается. Все общие объекты, которые были автоматически загружены при вызове dlopen() для объекта, на который ссылается handle, рекурсивно закрываются таким же способом.

**Общий метод и алгоритм решения**.

Первый вариант исполнения реализуется в программе linking.cpp. При помощи cmake создаются 2 библиотеки с разными версиями имплементации функций. Далее одна из библиотек линкуется к исполняемому файлу.

Второй вариант исполнения реализуется в программе runtime.cpp. В нем поключение библиотек происходит при помощи системных вызовов. Также реализована возможность переключения между разными вариантами имплементации.

Различные имплементации представлены в файлах lib1.cpp и lib2.cpp.

**Основные файлы программы**

**linking.cpp:**

#include <iostream>

#include "lib.h"

int main(){

std::cout << "'1' - вызов функции подсчёта НОД для двух натуральных чисел." << "\n"

<< "'2' - вызов функции подсчета количества простых чисел на отрезке [A, B]" << "\n"

<< "'-1' - окончание работы программы." << "\n" << "\n"

<< "Введите номер команды: ";

int num;

std::cin >> num;

while (num != -1){

switch (num){

case 1:

int a, b;

std::cout << "Введите два натуральных числа: ";

std::cin >> a >> b;

std::cout << "НОД(" << a << "," << b << ") = " << GCF(a, b) << "\n" << "\n"

<< "Введите номер команды: ";

std::cin >> num;

break;

case 2:

int A, B;

std::cout << "Введите концы отрезка А и В: ";

std::cin >> A >> B;

std::cout << "Количество простых чисел на отрезке [A, B]: " << PrimeCount(A, B)<< "\n" << "\n"

<< "Введите номер команды: ";

std::cin >> num;

break;

default:

std::cout << "Команды с таким номером нет." << "\n";

return -1;

}

}

return 0;

}

**runtime.cpp:**

#include <iostream>

#include <dlfcn.h>

int main(){

std::cout << "'0' - переключение реализации контрактов (по умолчанию 1 реализация). " << "\n"

<< "'1' - вызов функции подсчёта НОД для двух натуральных чисел." << "\n"

<< "'2' - вызов функции подсчета количества простых чисел на отрезке [A, B]" << "\n"

<< "'-1' - окончание работы программы." << "\n" << "\n"

<< "Введите номер команды: ";

int num;

std::cin >> num;

int lib = 1;

auto handle = dlopen("./liblib1.so", RTLD\_LAZY);

char\* error;

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

int (\*GCF)(int, int);

GCF = reinterpret\_cast<decltype(GCF)>(dlsym(handle, "GCF"));

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

int (\*prime)(int, int);

prime = reinterpret\_cast<decltype(prime)>(dlsym(handle, "PrimeCount"));

error = dlerror();

if (error)

{

std::cerr << error << std::endl;

return 1;

}

while (num != -1){

switch (num){

case 0:

dlclose(handle);

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

switch (lib){

case 1:

std::cout << "Переключено на реализацию 2. " << "\n" << "\n";

lib = 2;

handle = dlopen("./liblib2.so", RTLD\_LAZY);

break;

case 2:

std::cout << "Переключено на реализацию 1. " << "\n" << "\n";

lib = 1;

handle = dlopen("./liblib1.so", RTLD\_LAZY);

break;

}

char\* error;

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

GCF = reinterpret\_cast<decltype(GCF)>(dlsym(handle, "GCF"));

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

prime = reinterpret\_cast<decltype(prime)>(dlsym(handle, "PrimeCount"));

error = dlerror();

if (error)

{

std::cerr << error << std::endl;

return 1;

}

std::cout << "Введите номер команды: ";

std::cin >> num;

break;

case 1:

int a, b;

std::cout << "Введите два натуральных числа: ";

std::cin >> a >> b;

std::cout << "НОД(" << a << "," << b << ") = " << GCF(a, b) << "\n" << "\n"

<< "Введите номер команды: ";

std::cin >> num;

break;

case 2:

int A, B;

std::cout << "Введите концы отрезка A и B: ";

std::cin >> A >> B;

std::cout << "Количество простых чисел на отрезке [A, B]: " << prime(A, B) << "\n" << "\n"

<< "Введите номер команды: ";

std::cin >> num;

break;

default:

std::cout << "Команды с таким номером нет." << "\n";

return -1;

}

}

dlclose(handle);

error = dlerror();

if (error != 0){

std::cout << error << "\n";

return -1;

}

return 0;

}

**lib1.cpp:**

#include "lib.h"

// Алгоритм Евклида

int GCF(int A, int B){

int max, min;

if (A > B){

max = A;

min = B;

}

else if(A < B){

max = B;

min = A;

}

else{

return A;

}

while (min != 0){

int q = min;

min = max % min;

max = q;

}

return max;

}

// Наивный алгоритм подсчета простых чисел

bool bruteforce(int n)

{

int i = 2;

while (i \* i <= n)

{

if (n % i == 0)

{

return true;

}

++i;

}

return false;

}

int PrimeCount(int A, int B)

{

int count = 0;

for (int i = A; i <= B; ++i)

{

if (not bruteforce(i))

{

count++;

}

}

return count;

}

**lib2.cpp:**

#include "lib.h"

// Наивный алгоритм

int GCF(int A, int B){

int min;

if (A > B){

min = B;

}

else if (A < B){

min = A;

}

else{

return A;

}

int q = min;

while (q != 0){

if (A % q == 0 && B % q == 0){

return q;

}

--q;

}

return -1;

}

// Решето Эратосфена

int PrimeCount(int A, int B)

{

bool sieve[B];

int count = 0;

for (int i = 1; i <= B; i++)

{ // Изначально все числа не вычеркнуты.

sieve[i] = true;

}

for (int i = 2; i <= B; i++)

{

if (sieve[i])

{ // если i не вычеркнуто

if (i >= A)

{

count++;

}

for (int j = i \* i; j <= B; j += i)

{ // вычеркиваем все кратные числа начиная с i^2

sieve[j] = false;

}

}

}

return count;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(lab\_3)

set(CMAKE\_CXX\_STANDARD\_REQUIRED ON)

set(CMAKE\_CXX\_STANDARD 17)

add\_library(lib1 SHARED lib1.cpp)

add\_library(lib2 SHARED lib2.cpp)

add\_executable(main1 linking.cpp)

target\_link\_libraries(main1 lib1)

add\_executable(main2 runtime.cpp)

**Пример работы**

**Test 1:**

**tatiana@tatiana-VirtualBox:~/Рабочий стол/os\_4/build$ ./main1**

**'1' - вызов функции подсчёта НОД для двух натуральных чисел.**

**'2' - вызов функции подсчета количества простых чисел на отрезке [A, B]**

**'-1' - окончание работы программы.**

**Введите номер команды: 1**

**Введите два натуральных числа: 2 4**

**НОД(2,4) = 2**

**Введите номер команды: 2**

**Введите концы отрезка А и В: 2 8**

**Количество простых чисел на отрезке [A, B]: 4**

**Введите номер команды: -1**

**Test 2:**

**tatiana@tatiana-VirtualBox:~/Рабочий стол/os\_4/build$ ./main2**

**'0' - переключение реализации контрактов (по умолчанию 1 реализация).**

**'1' - вызов функции подсчёта НОД для двух натуральных чисел.**

**'2' - вызов функции подсчета количества простых чисел на отрезке [A, B]**

**'-1' - окончание работы программы.**

**Введите номер команды: 1**

**Введите два натуральных числа: 4 5**

**НОД(4,5) = 1**

**Введите номер команды: 0**

**Переключено на реализацию 2.**

**Введите номер команды: 1**

**Введите два натуральных числа: 4 5**

**НОД(4,5) = 1**

**Введите номер команды: 2**

**Введите концы отрезка A и B: 1 3**

**Количество простых чисел на отрезке [A, B]: 2**

**Введите номер команды: 0**

**Переключено на реализацию 1.**

**Введите номер команды: 2**

**Введите концы отрезка A и B: 6 7**

**Количество простых чисел на отрезке [A, B]: 1**

**Введите номер команды: -1**

**Вывод**

Во время выполнения данной работы я познакомилась с динамическими библиотеками. Я изучила создание библиотек при помощи CMake, а также мною было изучено подключение библиотек 2 способами: на этапе компиляции и на этапе исполнения программы. Подключение библиотек на этапе компиляции позволяет динамически изменять функции в зависимости от подключенной библиотеки, но замедляет время выполнения, так как требуется время на подключение библиотек.