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: Баталин Дмитрий Андреевич

Группа: М8О–212Б–20

Вариант: 17

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 динамические библиотеки, которые реализуют определенный функционал. Далее использовать данные библиотеки 2-мя способами:

* Во время компиляции (на этапе «линковки»/linking)
* 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.

17 вариант)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | 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 |
| 5 | Рассчет значения числа Пи при заданной длине ряда (K) | float Pi(int K) | Ряд Лейбница | Формула Валлиса |

**Общие сведения о программе**

Программа состоит из двух папок: linking и runtime. В папке linking находятся файлы первой библиотеки:

1. mathfirst.h – заголовочный файл для первой динамической библиотеки, где реализованы функции float Pi(int K) с помощью ряда Лейбница и int PrimeCount(int A, int B) с помощью наивного алгоритма.
2. mathfirst.c – реализация mathfirst.h
3. program1.c – программа 1, подключающая динамические библиотеки на этапе линковки, с возможностью выбора, какая функция должна выполниться.
4. libMathfirst.so – динамическая библиотека mathfirst которая создана с помощью команды “gcc -o libMathfirst.so -shared -fPIC mathfirst.c”
5. program1 – скомпилированная программа 1 командой “gcc program1.c -L. -lMathfirst -o program1”

В папке runtime находятся файлы второй библиотеки и программы:

1. mathsecond.h – заголовочный файл для первой динамической библиотеки, где реализованы функции float Pi(int K) с помощью формулы Валлиса и int PrimeCount(int A, int B) с помощью решета эратосфена.
2. mathsecond.c – реализация mathsecond.h
3. program2.c – программа 2, подключающая динамические библиотеки во время исполнения программы, с возможностью выбора, какая функция должна выполниться и с возможностью смены библиотеки на другую при помощи команды 0.
4. libMathsecond.so – динамическая библиотека mathfirst которая создана с помощью команды “gcc -o libMathsecond.so -shared -fPIC mathsecond.c”
5. program2 – скомпилированная программа 1 командой “ gcc program2.c -o program2 -ldl”

**Общий метод и алгоритм решения**.

Для решения поставленной задачи необходимо:

1. Разобраться в теории динамических библиотек
2. Реализовать функции, заданные в варианте
3. Создать заголовочные файлы и файлы реализации для будущих библиотек
4. Создание самих библиотек
5. Компиляция двух программ

**Основные файлы программы**

mathfirst.h:  
#ifndef MATHFIRST\_H

#define MATHFIRST\_H

**float** **Pi**(**int** K);

**int** **PrimeCount**(**int** A, **int** B);

#endif

mathfirst.c:  
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "mathfirst.h"

**float** **Pi**(**int** K) {

**float** result = **0**;

**for** (**size\_t** i = **0**; i < K; i++) {

result += pow(-**1**, i) / (**2**\*i + **1**);

}

**return** result \* **4**;

}

**int** **PrimeCount**(**int** A, **int** B) {

**int** count = **0**;

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

**int** check = **1**;

**if** (i <= **1**) {

check = **0**; // Число не является простым

}

**for** (**int** j = **2**; j < i; j++) {

**if** (i % j == **0**) {

check = **0**; // Число не является простым

**break**;

}

}

**if** (check == **1**) {

count++;

}

}

**return** count;

}

program1.c:

#include <stdio.h>

#include <stdlib.h>

#include "mathfirst.h"

**int** **main**() {

**int** command;

**do** {

// Ввод команды

printf("Введите команду (1, 2)(-1 для завершения выполнения): ");

scanf("%d", &command);

**switch** (command) {

**case** **1**: {

**int** iterations = **1000**;

printf("Введите количество итераций вычисления числа Пи 1: ");

scanf("%d", &iterations);

**float** pi = Pi(iterations);

printf("Pi: %f**\n**", pi);

**break**;

}

**case** **2**: {

**int** arg1, arg2;

printf("Введите отрезок на котором нужно посчитать простые числа: ");

scanf("%d %d", &arg1, &arg2);

**int** count = PrimeCount(arg1, arg2);

printf("Простых чисел: %d**\n**", count);

**break**;

}

**case** -**1**: **break**;

**default:**

printf("Некорректная команда**\n**");

**break**;

}

} **while** (command != -**1**);

}

mathsecond.h

#ifndef MATHSECOND\_H

#define MATHSECOND\_H

**float** **Pi**(**int** K);

**int** **PrimeCount**(**int** A, **int** B);

#endif

mathsecond.c

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include "mathsecond.h"

**float** **Pi**(**int** K) {

**float** result = **1**;

**for** (**int** i = **1**; i < K; i++) {

result \*= (**4**\*i\*i) / (**4**\*i\*i - **1.0**);

}

**return** result \* **2**;

}

**int** **PrimeCount**(**int** A, **int** B) {

**if** (B < **2** || A > B) {

**return** **0**; // Нет простых чисел в заданном диапазоне

}

// Создаем массив для решета Эратосфена и инициализируем его

**bool** \*isPrime = (**bool** \*)malloc((B + **1**) \* **sizeof**(**bool**));

**for** (**int** i = **0**; i <= B; i++) {

isPrime[i] = true;

}

// Исключаем из решета числа 0 и 1

isPrime[**0**] = isPrime[**1**] = false;

// Применяем решето Эратосфена

**for** (**int** i = **2**; i \* i <= B; i++) {

**if** (isPrime[i]) {

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

isPrime[j] = false;

}

}

}

// Подсчитываем количество простых чисел в заданном диапазоне

**int** count = **0**;

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

**if** (isPrime[i]) {

count++;

}

}

// Освобождаем память, выделенную для решета

free(isPrime);

**return** count;

}

program2.c:

#include <stdio.h>

#include <stdlib.h>

#include <dlfcn.h>

#include "mathsecond.h"

**int** **main**() {

**void** \*library\_handler;

**float** (\*Pi)(**int** K);

**int** (\*PrimeCount)(**int** A, **int** B);

library\_handler = dlopen("../linking/libMathfirst.so", RTLD\_LAZY);

**if** (!library\_handler){

fprintf(stderr, "dlopen() error: %s**\n**", dlerror());

exit(**1**);

};

**int** command;

**int** contract = **0**;

**do** {

// Ввод команды

printf("Введите команду (0, 1, 2)(-1 для завершения выполнения): ");

scanf("%d", &command);

**switch** (command) {

**case** **0**:

**if** (contract == **0**) {

contract = **1**;

dlclose(library\_handler);

library\_handler = dlopen("libMathsecond.so", RTLD\_LAZY);

**if** (!library\_handler){

fprintf(stderr, "dlopen() error: %s**\n**", dlerror());

exit(**1**);

};

} **else** {

contract = **0**;

dlclose(library\_handler);

library\_handler = dlopen("../linking/libMathfirst.so", RTLD\_LAZY);

**if** (!library\_handler){

fprintf(stderr, "dlopen() error: %s**\n**", dlerror());

exit(**1**);

};

}

printf("Реализация переключения контрактов**\n**");

**break**;

**case** **1**: {

**int** iterations = **1000**;

printf("Введите количество итераций вычисления числа Пи 1: ");

scanf("%d", &iterations);

Pi = dlsym(library\_handler, "Pi");

**float** pi = (\*Pi)(iterations);

printf("Pi: %f**\n**", pi);

**break**;

}

**case** **2**: {

**int** arg1, arg2;

printf("Введите отрезок на котором нужно посчитать простые числа: ");

scanf("%d %d", &arg1, &arg2);

PrimeCount = dlsym(library\_handler, "PrimeCount");

**int** primeResult = (\*PrimeCount)(arg1, arg2);

printf("Простых чисел: %d**\n**", primeResult);

**break**;

}

**case** -**1**: **break**;

**default:**

printf("Некорректная команда**\n**");

**break**;

}

} **while** (command != -**1**);

}

**Пример работы**

moses@MacBook-Pro-Dmitrij linking % ./program1

Введите команду (1, 2): 1

Введите количество итераций вычисления числа Пи 1: 100

Pi: 3.131593

Введите команду (1, 2): 1

Введите количество итераций вычисления числа Пи 1: 1

Pi: 4.000000

Введите команду (1, 2): 1

Введите количество итераций вычисления числа Пи 1: 2

Pi: 2.666667

Введите команду (1, 2): 1

Введите количество итераций вычисления числа Пи 1: 3

Pi: 3.466667

Введите команду (1, 2): 1

Введите количество итераций вычисления числа Пи 1: 10000

Pi: 3.141498

Введите команду (1, 2): 2

Введите отрезок на котором нужно посчитать простые числа: 1 10

Простых чисел: 4

Введите команду (1, 2): 2

Введите отрезок на котором нужно посчитать простые числа: 1 11

Простых чисел: 5

Введите команду (1, 2): 2

Введите отрезок на котором нужно посчитать простые числа: 0 2

Простых чисел: 1

Введите команду (1, 2): 2

Введите отрезок на котором нужно посчитать простые числа: 10 10000

Простых чисел: 1225

Введите команду (1, 2): -1

moses@MacBook-Pro-Dmitrij linking %

moses@MacBook-Pro-Dmitrij runtime % ./program2

Введите команду (0, 1, 2)(-1 для завершения выполнения): 1

Введите количество итераций вычисления числа Пи 1: 100

Pi: 3.131593

Введите команду (0, 1, 2)(-1 для завершения выполнения): 0

Реализация переключения контрактов

Введите команду (0, 1, 2)(-1 для завершения выполнения): 1

Введите количество итераций вычисления числа Пи 1: 100

Pi: 3.133708

Введите команду (0, 1, 2)(-1 для завершения выполнения): 0

Реализация переключения контрактов

Введите команду (0, 1, 2)(-1 для завершения выполнения): 1

Введите количество итераций вычисления числа Пи 1: 100

Pi: 3.131593

Введите команду (0, 1, 2)(-1 для завершения выполнения): 2

Введите отрезок на котором нужно посчитать простые числа: 1 10

Простых чисел: 4

Введите команду (0, 1, 2)(-1 для завершения выполнения): 0

Реализация переключения контрактов

Введите команду (0, 1, 2)(-1 для завершения выполнения): 2

Введите отрезок на котором нужно посчитать простые числа: 1 10

Простых чисел: 4

Введите команду (0, 1, 2)(-1 для завершения выполнения): -1

moses@MacBook-Pro-Dmitrij runtime %

**Вывод**

Я научился работать с простейшими динамическими библиотеками в си. В процессе выполнения лабораторной работы я на своем личном опыте понял, чем отличаются статические и динамические библиотеки. Узнал, что во время использования динамических библиотек на этапе компиляции линковщик встраивает в программу описания требуемых функций и переменных, которые присутствуют в библиотеке. Но сами объектные файлы из библиотеки не присоединяются к программе. Присоединение этих объектных файлов осуществляет системный динамический загрузчик во время запуска программы. Загрузчик проверяет все библиотеки прилинкованные к программе на наличие требуемых объектных файлов, затем загружает их в память и присоединяет их в копии запущенной программы, находящейся в памяти. И самый главный плюс динамических библиотек состоит в том, что если другая запускаемая программа линкована с этой же загруженной динамической библиотекой, то она использует ту же копию библиотеки.   
Анализируя два вида использования динамических библиотек, можно сказать, что при подключении библиотеки на этапе линковки мы можем использовать функции внутри программы образом, очень сильно похожим на то, как используются функции из статических библиотек. А при подключении во время исполнения программы мы должны будем открыть библиотеку внутри нее, должны будем получить оттуда адрес функции, разыменовать ее и передать туда значения, что в общем не является очень сложной задачей, но занимает больше времени в реализации, чем первый способ. Благодаря подклчению библиотек в рантайме можно получать лишь названия функций и библиотек на вход, а внутри уже подключать их и использовать, что дает много возможностей для написания различных плагинов для программ, модернизации различных ее частей.