**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные Системы»

**Лабораторная работа № 5**

Тема: Создание динамических библиотек и

создание программ, которые используют функции динамических библиотек

Студент: Семин Александр Витальевич

Группа: М8О-206Б-20

Преподаватель: Соколов Андрей Алексеевич

Дата:

Оценка:

Москва, 2021

1. **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:

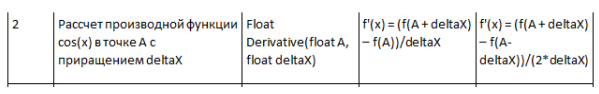
* Динамические библиотеки, реализующие контракты, которые заданы вариантом;
* Тестовая программа (программа №1), которая используют одну из библиотек, используя знания полученные на этапе компиляции;
* Тестовая программа (программа №2), которая загружает библиотеки, используя только их местоположение и контракты.

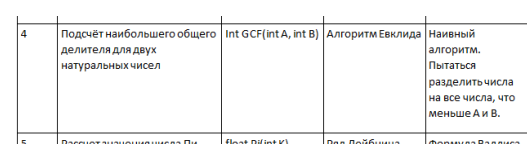
Провести анализ двух типов использования библиотек. Пользовательский ввод для обоих программ должен быть организован следующим образом:

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 10





1. **Набор тестов**

*Первый тестовый набор:*

progger@asus:~/Desktop/OS\_labs/l5$ ./prog1

1. Рассчет производной функции cos(x) в точке A с приращением deltaX

USAGE: float Derivative(float A, float deltaX)

2. Подсчёт наибольшего общего делителя для двух натуральных чисел A B

USAGE: int GCF(int A, int B)

1 0.123 0.01

-0.127650

2 14 49

7

*Второй тестовый набор:*

progger@asus:~/Desktop/OS\_labs/l5$ ./prog2

1. Рассчет производной функции cos(x) в точке A с приращением deltaX

USAGE: float Derivative(float A, float deltaX)

2. Подсчёт наибольшего общего делителя для двух натуральных чисел A B

USAGE: int GCF(int A, int B)

1 0.123 0.01

-0.127650

0

Switched to version 2

1 0.123 0.01

-0.122688

2 49 14

7

1. **Листинг программы**

**functions.h**

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

float Derivative(float A, float deltaX);

int GCF(int A, int B);

#endif

**lib.c**

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

//Рассчет производной функции cos(x) в точке A с приращением deltaX

float Derivative(float A, float deltaX) {

float res = (cos(A+deltaX) - cos(A)) / deltaX;

return res;

}

//Подсчёт наибольшего общего делителя для двух натуральных чисел A B

//Алгоритм Евклидa

int GCF(int A, int B) {

while (A != B) {

if (A > B)

A -= B;

else B -= A;

}

return A;

}

**prog1.c**

##include <stdio.h>

#include "functions.h"

void menu() {

printf("1. Рассчет производной функции cos(x) в точке A с приращением deltaX\n");

printf("USAGE: float Derivative(float A, float deltaX)\n\n");

printf("2. Подсчёт наибольшего общего делителя для двух натуральных чисел A B\n");

printf("USAGE: int GCF(int A, int B)\n\n");

}

int main() {

int cmd;

menu();

while(scanf("%d", &cmd) != EOF) {

if (cmd == 1) {

float x, y;

if(scanf("%f %f", &x, &y) != 2) {

printf("Invalid arguments!\n");

continue;

}

printf("%f\n", Derivative(x, y));

} else if (cmd == 2) {

int x, y;

if(scanf("%d%d", &x, &y) != 2) {

printf("Invalid arguments!\n");

continue;

}

printf("%d\n", GCF(x, y));

} else {

printf("Invalid command!\n");

menu();

}

}

}

**lib2.c**

#include <math.h>

//Рассчет производной функции cos(x) в точке A с приращением deltaX

float Derivative(float A, float deltaX) {

float res = (cos(A+deltaX) - cos(A-deltaX)) / (2\*deltaX);

return res;

}

//Подсчёт наибольшего общего делителя для двух натуральных чисел A B

//Наивный алгоритм.

int GCF(int A, int B) {

int prev\_nod = 0;

int nod = 1;

while (nod < A && nod < B) {

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

prev\_nod = nod;

}

nod++;

}

return prev\_nod;

}

**prog2.c**

#include <stdio.h>

#include <dlfcn.h>

void menu() {

printf("1. Рассчет производной функции cos(x) в точке A с приращением deltaX\n");

printf("USAGE: float Derivative(float A, float deltaX)\n\n");

printf("2. Подсчёт наибольшего общего делителя для двух натуральных чисел A B\n");

printf("USAGE: int GCF(int A, int B)\n\n");

}

int main() {

float (\*Derivative)(float, float);

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

void\* l1\_handler = dlopen("./lib1.so", RTLD\_LAZY);

void\* l2\_handler = dlopen("./lib2.so", RTLD\_LAZY);

if (!l1\_handler || !l2\_handler) {

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

return -1;

}

Derivative = dlsym(l1\_handler,"Derivative");

GCF = dlsym(l1\_handler,"GCF");

int ver = 0;

int cmd;

menu();

while (scanf("%d", &cmd) != EOF) {

if (cmd == 0) {

ver ^= 1;

if (ver == 0) {

Derivative = dlsym(l1\_handler,"Derivative");

GCF = dlsym(l1\_handler,"GCF");

} else {

Derivative = dlsym(l2\_handler,"Derivative");

GCF = dlsym(l2\_handler,"GCF");

}

printf("Switched to version %d\n", ver + 1);

} else if (cmd == 1) {

float x, y;

if (scanf("%f %f", &x, &y) != 2) {

printf("Invalid arguments!\n");

continue;

}

printf("%f\n", Derivative(x, y));

} else if (cmd == 2) {

int x, y;

if (scanf("%d %d", &x, &y) != 2) {

printf("Invalid arguments!\n");

continue;

}

printf("%d\n", GCF(x, y));

} else {

printf("Invalid command!\n");

menu();

}

}

dlclose(l1\_handler);

dlclose(l2\_handler);

}

**Makefile:**

all: prog1 prog2

lib1.so: lib1.c

gcc -shared lib1.c -o lib1.so -lm -Wall

lib2.so: lib2.c

gcc -shared lib2.c -o lib2.so -lm -Wall

prog2: lib1.so lib2.so prog2.c

gcc prog2.c -ldl -o prog2 -Wall

prog1: lib1.so prog1.c

gcc prog1.c -L "/home/progger/Desktop/OS\_labs/l5" -Wl,-R. -l1 -o prog1 –Wall

1. **Выводы**

В процессе выполнения работы я реализовал распределенную систему по асинхронной обработке запросов. В программе используется протокол передачи данных tcp, в котором общение между процессами происходит через определенные порты.

Обмен сообщений происходит посредством функций библиотеки ZMQ, а в частности, ее паттерном «Request – Reply». Это один из самых простых и прямолинейных паттернов, который своей реализацией очень напоминает pipe.

Также хорошей тренировкой стала реализация идеально сбалансированного бинарного дерева на С++.