Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

**Тема работы**

**«Динамические библиотеки»**

Студент: Павлов Иван Дмитриевич

Группа: М8О-207Б-21

Вариант: 12

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Pavloffff/MAI\_OS/tree/main/lab5

**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*).
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 12**:

Контракт 1:

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

Float Derivative(float A, float deltaX)

Реализация 1:

f'(x) = (f(A + deltaX) – f(A))/deltaX

Реализация 2:

f'(x) = (f(A + deltaX) – f(A-deltaX))/(2\*deltaX)

Контракт 2:

Рассчет значения числа е (основание натурального логарифма)

Float E(int x)

Реализация 1:

(1 + 1/x) ^ x

Реализация 2:

Сумма ряда по n от 0 до x, где элементы ряда равны: (1/(n!))

Система сборки: CMake.

Вариант 3:

Возможность сборки всех таргетов с ASAN без переопределения CMake флагов, если указана соответствующая переменная и ОС имеет поддержку ASAN.

**Общие сведения о программе**

Программа состоит из двух интерфейсов (main1.c и main2.c), каждый из них реализован по-разному, в соответствии с заданием. Также каждая реализация контрактов представляет из себя отдельный файл: lib1.c и lib2.c. Для объявления необходимых функций также используется заголовочный файл lib.h. Так как все собирается с помощью CMake, то в проекте присутствует CMakeLists.txt.

**Общий метод и алгоритм решения**

Объявим необходимые функции внутри файла lib.h. Используем спецификатор хранения extern, который сообщает компилятору, что находящиеся за ним типы и имена переменных объявляются где-то в другом месте.

Так как по заданию необходимо подключать библиотеки на этапе линковки, то подключать lib.h в реализации lib1.c и lib2.c не следует. В этих файлах просто напишем логику работы необходимых функций. Важно, чтобы они назывались также, как и те, что объявлены в lib.h.

Используемые алгоритмы:

* Косинус — сумма ряда Тейлора;
* Факториал — факториал «Деревом»;
* Возведение в степень — алгоритм «бинарного» возведения в степень.

Интерфейс 1:

Подключаем lib.h и пользуемся функциями так, как будто библиотека обычная. Различия наступают в сборке программы. Если бы мы собирали такой код в терминале, то прописали бы gcc -c -fpic lib1.c. Опция -fpic - требует от компилятора, при создании объектных файлов, порождать позиционно-независимый код. Формат позиционно-независимого кода позволяет подключать исполняемые модули к коду основной программы в момент её загрузки. Далее gcc -shared -o liblib1.so lib1.o -lm. Опция -shared - указывает gcc, что в результате должен быть собран не исполняемый файл, а разделяемый объект — динамическая библиотека.

Интерфейс 2:

Воспользуемся системными вызовами из библиотеки <dlfcn.h>.

Функция dlopen открывает динамическую библиотеку (объект .so) по названию.

Функция dlsym - обоаботчик динамически загруженного объекта вызовом dlopen.

Функция dlclose, соответственно, закрывает динамическую библиотеку.

Собираем с помощью gcc -L. -Wall -o main.out main2.c -llib2 -llib1. Флаг -L. Означает, что поиск файлов библиотек будет начинаться с текущей директории.

Система сборки:

ASAN — это Address Sanitizer, инструмент, с помощью которого можно ловить RE связанные с неправильным обращением к памяти. Наиболее логичный способ их интеграции в СMake — интегрировать их как типы сборки CMake, чтобы программы были созданы оптимально для санитайзеров. Для получения оптимальных результатов эти типы сборки игнорируют все другие флаги компилятора.

**Исходный код**

**lib.h**

#ifndef \_\_LIB\_H\_\_

#define \_\_LIB\_H\_\_

extern float Derivative(float A, float deltaX);

extern float E(int x);

#endif

**lib1.c**

#include <stdio.h>

const float PI = 3.1415926;

float Cos(float x)

{

int y = 100;

int div = (int) (x / PI);

x = x - (div \* PI);

char sign = 1;

if (div % 2 != 0) {

sign = -1;

}

float result = 1.0;

float inter = 1.0;

float num = x \* x;

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

float comp = 2.0 \* i;

float den = comp \* (comp - 1.0);

inter \*= num / den;

if (i % 2 == 0) {

result += inter;

} else {

result -= inter;

}

}

return sign \* result;

}

float Derivative(float A, float deltaX)

{

printf("\nCalculation of derivative function f(x) = Cos(x)\n");

printf("in point %f with approximation %f\n", A, deltaX);

printf("by formula f'(x) = (f(A + deltaX) – f(A))/deltaX\n");

printf("cos(A) = %f\n", Cos(A));

float dfdx = (Cos(A + deltaX) - Cos(A)) / deltaX;

return dfdx;

}

float binPow(float x, int y)

{

float z = 1.0;

while (y > 0) {

if (y % 2 != 0) {

z \*= x;

}

x \*= x;

y /= 2;

}

return z;

}

float E(int x)

{

printf("\nCalculation value of number e (base of natural logarithm)\n");

printf("with approximation %d\n", x);

printf("by formula e(x) = (1 + 1/x) ^ x\n");

float mant = (float) 1 + ((float) 1 / (float) x);

float e = binPow(mant, x);

return e;

}

**lib2.c**

#include <stdio.h>

const float PI = 3.1415926;

float Cos(float x)

{

int y = 100;

int div = (int) (x / PI);

x = x - (div \* PI);

char sign = 1;

if (div % 2 != 0) {

sign = -1;

}

float result = 1.0;

float inter = 1.0;

float num = x \* x;

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

float comp = 2.0 \* i;

float den = comp \* (comp - 1.0);

inter \*= num / den;

if (i % 2 == 0) {

result += inter;

} else {

result -= inter;

}

}

return sign \* result;

}

float Derivative(float A, float deltaX)

{

printf("\nCalculation of derivative function f(x) = cos(x)\n");

printf("in point %f with approximation %f\n", A, deltaX);

printf("by formula f'(x) = (f(A + deltaX) – f(A-deltaX))/(2\*deltaX)\n");

printf("cos(A) = %f\n", Cos(A));

float dfdx = (Cos(A + deltaX) - Cos(A - deltaX)) / (2 \* deltaX);

return dfdx;

}

int prodTree(int l, int r)

{

if (l > r) {

return 1;

}

if (l == r) {

return l;

}

if (l - r == 1) {

return l \* r;

}

int m = (l + r) / 2;

return prodTree(l, m) \* prodTree(m + 1, r);

}

int fact(int n)

{

if (n < 0) {

return 0;

}

if (n == 0) {

return 1;

}

if (n == 1 || n == 2) {

return n;

}

return prodTree(2, n);

}

float machineEpsilon(void)

{

float e = 1.0f;

while (1.0f + e / 2.0f > 1.0f)

e /= 2.0f;

return e;

}

float E(int x)

{

printf("\nCalculation value of number e (base of natural logarithm)\n");

printf("with approximation %d\n", x);

printf("by sum of row by n from 0 to x f(n) = (1/(n!))\n");

float e = 0;

for (int n = 0; n <= x; n++) {

float tmp = ((float) 1 / fact(n));

float ftmp = tmp > 0 ? tmp : (float) (-1) \* tmp;

if (ftmp <= machineEpsilon()) {

printf("Approximation can not work because of mashine Epsilon of float is %.8f\n", machineEpsilon());

break;

}

e += tmp;

}

return e;

}

**main1.c**

#include <stdio.h>

#include "lib.h"

int main(int argc, char const \*argv[])

{

printf("\nWrite:\n [command] [arg1] ... [argN]\n");

printf("\nIf you want to take derivation of f(x) = cos(x), write 1 [point] [delta]\n");

printf("\nIf you want to calculate number e (base of natural logarithm), write 2 [approximation]\n\n");

int command = 0;

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

switch (command) {

case 1:

float A, deltaX;

scanf("%f%f", &A, &deltaX);

printf("Answer: %f\n", Derivative(A, deltaX));

break;

case 2:

int x;

scanf("%d", &x);

printf("Answer: %f\n", E(x));

break;

default:

printf("wrong command\n");

break;

}

printf("\nWrite:\n [command] [arg1] ... [argN]\n");

printf("\nIf you want to take derivation of f(x) = cos(x), write 1 [point] [delta]\n");

printf("\nIf you want to calculate number e (base of natural logarithm), write 2 [approximation]\n\n");

}

return 0;

}

**main2.c**

#include <stdio.h>

#include <dlfcn.h>

#include "lib.h"

const char\* lib1 = "./liblib1.so";

const char\* lib2 = "./liblib2.so";

int main(int argc, char const \*argv[])

{

printf("\nWrite:\n [command] [arg1] ... [argN]\n");

printf("\nIf you want to change methods of calculation, write 0\n");

printf("\nIf you want to take derivation of f(x) = cos(x), write 1 [point] [delta]\n");

printf("\nIf you want to calculate number e (base of natural logarithm), write 2 [approximation]\n");

int command = 0;

int link = 0;

void \*currentLib = dlopen(lib1, RTLD\_LAZY);

printf("\nCurrent lib is %d\n\n", link);

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

float (\*E)(int x);

Derivative = dlsym(currentLib, "Derivative");

E = dlsym(currentLib, "E");

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

switch (command) {

case 0:

dlclose(currentLib);

if (link == 0) {

currentLib = dlopen(lib2, RTLD\_LAZY);

} else {

currentLib = dlopen(lib1, RTLD\_LAZY);

}

link = !link;

Derivative = dlsym(currentLib, "Derivative");

E = dlsym(currentLib, "E");

break;

case 1:

float A, deltaX;

scanf("%f%f", &A, &deltaX);

printf("Answer: %f\n", Derivative(A, deltaX));

break;

case 2:

int x;

scanf("%d", &x);

printf("Answer: %f\n", E(x));

break;

default:

printf("wrong command\n");

break;

}

printf("\nWrite:\n [command] [arg1] ... [argN]\n");

printf("\nIf you want to change methods of calculation, write 0\n");

printf("\nIf you want to take derivation of f(x) = cos(x), write 1 [point] [delta]\n");

printf("\nIf you want to calculate number e (base of natural logarithm), write 2 [approximation]\n");

printf("\nCurrent lib is %d\n\n", link);

}

return 0;

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.8 FATAL\_ERROR)

project(main LANGUAGES C)

set(BUILD\_WITH\_ASAN 1)

add\_library(

lib1 SHARED

./include/lib.h

./src/lib1.c

)

add\_library(

lib2 SHARED

./include/lib.h

./src/lib2.c

)

add\_executable(main1 ./src/main1.c)

target\_include\_directories(main1 PRIVATE ./include)

target\_link\_libraries(main1 PRIVATE lib1 m)

add\_executable(main2 ./src/main1.c)

target\_include\_directories(main2 PRIVATE ./include)

target\_link\_libraries(main2 PRIVATE lib2 m)

add\_executable(main ./src/main2.c)

target\_include\_directories(main PRIVATE ./include m)

if (${BUILD\_WITH\_ASAN})

message("-- Adding sanitizers")

target\_compile\_options(main PRIVATE -fsanitize=address)

target\_link\_options(main PRIVATE -fsanitize=address)

target\_compile\_options(main1 PRIVATE -fsanitize=address)

target\_link\_options(main1 PRIVATE -fsanitize=address)

target\_compile\_options(main2 PRIVATE -fsanitize=address)

target\_link\_options(main2 PRIVATE -fsanitize=address)

endif()

**Демонстрация работы программы**

ggame@ggame:~/OS/ready/lab5/build$ ./main

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 0

0

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 1

1 2 0.0001

Calculation of derivative function f(x) = cos(x)

in point 2.000000 with approximation 0.000100

by formula f'(x) = (f(A + deltaX) – f(A-deltaX))/(2\*deltaX)

cos(A) = -0.416147

Answer: -0.908971

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 1

2 1000

Calculation value of number e (base of natural logarithm)

with approximation 1000

by sum of row by n from 0 to x f(n) = (1/(n!))

Approximation can not work because of mashine Epsilon of float is 0.00000012

Answer: 2.718282

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 1

0

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 0

1 3.14 0.0001

Calculation of derivative function f(x) = Cos(x)

in point 3.140000 with approximation 0.000100

by formula f'(x) = (f(A + deltaX) – f(A))/deltaX

cos(A) = -0.999999

Answer: -0.003576

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 0

2 1000

Calculation value of number e (base of natural logarithm)

with approximation 1000

by formula e(x) = (1 + 1/x) ^ x

Answer: 2.717042

Write:

[command] [arg1] ... [argN]

If you want to change methods of calculation, write 0

If you want to take derivation of f(x) = cos(x), write 1 [point] [delta]

If you want to calculate number e (base of natural logarithm), write 2 [approximation]

Current lib is 0

**Выводы**

В ходе лабораторной работы я познакомился с созданием динамических библиотек в ОС Linux, а также с возможностью загружать эти библиотеки в ходе выполнения программы.