Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

Студент: Маринин И.С.

Группа: М8О–208Б–20

Вариант: 13

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021.

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

· Создании динамических библиотек.

· Создании программ, которые используют функции динамических библиотек.

**Задание**

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»;

Я не реализовал такую функцию.

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 13:

Функция 1: Вычисление функций:

float res1 = (cos(A + deltaX) - cos(A)) / deltaX

float res2 = (cos(A + deltaX) - cos(A - deltaX)) / (2 \* deltaX)

Функция 2: Подсчет площади плоской геометрической фигуры по двум сторонам. float Square(float A, float B). Фигура прямоугольник. Фигура прямоугольный треугольник.

**Общие сведения о программе**

Программа компилируется при помощи Makefile в 2 исполняемых файла dynLib.c, statLib.c и 2 библиотеки libstat.a, libdynn.so. В первом случае мы используем библиотеку, которая использует знания полученные во время компиляции (на этапе линковки). Во втором случае программа загружает библиотеки и взаимодействует с ними при помощи следующих системных вызовов:

1. **dlopen** – загружает динамическую библиотеку, имя которой указано первым аргументом, и возвращает прямой указатель на начало динамической библиотеки. Второй аргумент отвечает за разрешение неопределенных символов, возвращает 0 при успешном завершении и значение != 0 в случае ошибки.
2. **dlsym** – использует указатель на динамическую библиотеку – первый аргумент, возвращаемую dlopen, и оканчивающееся нулем символьное имя – второй аргумент, а затем возвращает адрес, указывающий, откуда загружается этот символ. Если символ не найден, то возвращаемым значением dlsym является NULL.
3. **dlclose –** уменьшает на единицу счетчик ссылок на указатель динамической библиотеки, передаваемый в качестве аргумента. Если нет других загруженных библиотек, использующих ее символы и если счетчик ссылок принимает нулевое значение, то динамическая библиотека выгружается.

**Общий метод и алгоритм решения**.

Создаем по два исполняемых файла и два heder-a с реализациями и объявлениями для каждой из двух функций, собираем из них библиотеку и используем 2-мя способами:

* + - 1. на этапе компиляции (стадия линковки) при помощи #include в программе static\_main.c
      2. при помощи загрузки библиотек при помощи dlopen в программе dynamic\_main.c

**Основные файлы программы**

**Makefile:**

CC = gcc

all: run

run: libstat.a libdynn.so static\_main.o dynamic\_main.o

$(CC) -o stat-prog static\_main.o -L. -lstat -lm

$(CC) -o dyn-prog dynamic\_main.o -L. -ldynn -ldl

libstat.a: static\_lib.o

ar rc libstat.a static\_lib.o

static\_lib.o: static\_lib.c

$(CC) -c static\_lib.c

static\_main.o: static\_main.c

$(CC) -c static\_main.c

libdynn.so: dynamic\_lib.o

$(CC) -shared -o libdynn.so dynamic\_lib.o

dynamic\_lib.o: dynamic\_lib.c

$(CC) -c -fPIC dynamic\_lib.c -lm

dynamic\_main.o: dynamic\_main.c

$(CC) -c dynamic\_main.c -lm

clean:

rm \*.o \*.a libdynn.so stat-prog dyn-prog

**static\_main.c:**

#include <stdio.h>

#include <stdlib.h>

#include "static\_lib.h"

int main() {

int c;

double x, dx;

while(scanf("%d", &c)) {

if (c == 1) {

scanf("%lf %lf", &x, &dx);

printf("[1]: Derivative = %lf\n", Derivative1(x, dx));

}

else if (c == 2) {

scanf("%lf %lf", &x, &dx);

printf("[2]: Derivative = %lf\n", Derivative2(x, dx));

}

else {

printf("ERROR\n");

break;

}

}

return 0;

}

**dynamic\_main.c:**

#include <stdio.h>

#include <dlfcn.h>

#include "dynamic\_lib.h"

int main() {

void\* handle = dlopen("libdynn.so", RTLD\_LAZY);

// загружаем динамическую библиотеку libdynn.so, и возвращаем прямой указатель на начало динамической библиотеки

// RTLD\_LAZY подразумевает разрешение неопределенных символов в виде кода, содержащегося в исполняемой динамической библиотеке

if(handle == NULL) {

printf("%s\n",dlerror());

return 1;

}

float (\*Square1) (float a, float b) = dlsym(handle, "Square1"); // Вызываем функцию Square1

float (\*Square2) (float a, float b) = dlsym(handle, "Square2");

int c;

float a, b;

while(scanf("%d", &c)) {

if (c == 1) {

scanf("%f %f", &a, &b);

printf("[1]: Square = %f\n", Square1(a, b));

}

else if (c == 2) {

scanf("%f %f", &a, &b);

printf("[2]: Square = %f\n", Square2(a, b));

}

else {

printf("ERROR\n");

break;

}

}

dlclose(handle);

return 0;

}

**dynamic\_lib.c:**

#include "dynamic\_lib.h"

float Square1(float a, float b) {

return a \* b;

}

float Square2(float a, float b) {

return (a \* b) / 2;

}

**static\_lib.c:**

#include "static\_lib.h"

double Derivative1(double x, double dx) {

return (cos(x + dx) - cos(x)) / dx;

}

double Derivative2(double x, double dx) {

return (cos(x + dx) - cos(x - dx)) / (2 \* dx);

}

**Пример работы**

ivanmarinin@MacBook-Air-Ivan src % ./stat-prog

1 0.4 0.01

[1]: Derivative = -0.394017

2 0.4 0.01

[2]: Derivative = -0.389412

1 5 0.1

[1]: Derivative = 0.943156

2 5 0.1

[2]: Derivative = 0.957327

0

ERROR

ivanmarinin@MacBook-Air-Ivan src % ./dyn-prog

1 4 5

[1]: Square = 20.000000

2 4 5

[2]: Square = 10.000000

1 100 100

[1]: Square = 10000.000000

2 100 100

[2]: Square = 5000.000000

1 3 3

[1]: Square = 9.000000

2 3 3

[2]: Square = 4.500000

0

ERROR

**Вывод**

В СИ можно использовать динамические библиотеки, в основном применяются, когда программист не хочет делиться со своим исходным кодом. Однако, каждая библиотека имеет свои недостатки. В этой лабораторной работе я научился их создавать и использовать на практике.