Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу

«Операционные системы»

ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ

Студент: Белоносов Кирилл Алексеевич
Группа: М8О–208Б–21
Вариант: 14
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022.

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:

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.

Общие сведения о программе

Программа компилируется из файла main.c. Также я дополнительно реализовал 2 динамических библиотеки, для расчета производной и для расчета интеграла.

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.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы dlsym, dlopen, dlclose.
2. Написать библиотеку для вычисления производной
3. Написать библиотеку для вычисления интеграла
4. В файле main1.c подключить библиотеку на этапе компиляции.
5. В файле main2.c загрузить библиотечные функции в runtime, с помощью dlsym, dlopen, dlclose.

Основные файлы программы

der1.c:

```
#include <math.h>
float Derivative(float A, float deltaX) {
   return (cosf(A + deltaX) - cosf(A))/deltaX;
}
```

der2.c

```
#include <math.h>
float Derivative(float A, float deltaX) {
   return (cos(A + deltaX) - cos(A - deltaX))/(2*deltaX);
}
```

trans1.c

```
#include <stdlib.h>
char* translation(long x) {
   char s[64];
   int length = 0;
   for(int i = 0; x > 0; ++i) {
      long a = x % 2;
      s[i] = a + '0';
      length++;;
      x /= 2;
   }
   char * r = (char *)malloc((length + 1)*sizeof(char));
   for(int i = length - 1; i >= 0; --i) {
```

```
        r[i] = s[length - 1 - i];
    }
    r[length] = '\0';
    return r;
}
```

trans2.c

```
#include <stdlib.h>
char* translation(long x) {
    char s[64];
    int length = 0;
    for(int i = 0; x > 0; ++i) {
        long a = x % 3;
        s[i] = a + '0';
        length++;
        x /= 3;
    }
    char * r;
    r = (char *)malloc((length + 1)*sizeof(char));
    for(int i = length - 1; i >= 0; --i) {
        r[i] = s[length -1 - i];
    }
    r[length] = '\0';
    return r;
}
```

prog1.h

```
char* translation(long x);
float Derivative(float A, float deltaX);
```

main1.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "prog1.h"
int main(int argc, char * argv[]) {
```

```
    int func = 0;
    long x = 0;
    float A = 0;
    float deltaX = 0;
    if(argc > 2) {
        func = atoi(argv[1]);
        if(func == 1) {
            A = atof(argv[2]);
            deltaX = atof(argv[3]);
            printf("Derivative at a point %f with delta %f is %f\n", A, deltaX, Derivative(A,
deltaX));
        }
        if(func == 2) {
            x = atoi(argv[2]);
            printf("%s\n", translation(x));
        }
    }
}
```

main2.c

```
#include <stdio.h>
#include <dlfcn.h>
#include <math.h>
#include <stdlib.h>
int main(int argc, char * argv[]) {
    void * handler1;
    void * handler2;
    float(*Derivative)(float, float);
    char *(*translation)(long);
    if (argc > 2) {
        if(atoi(argv[1]) == 0) {
            if(atoi(argv[2]) == 1) {
                handler1 =
dlopen("/home/kirill/localProjects/OSlabs/lab5/build/src/include/libder2.so", RTLD_LAZY);
                if (!handler1) {
```

```
            fprintf(stderr, "Open of dynamic library der2 with error: %s\n", dlerror());
            exit(-1);
        }
        Derivative = dlsym(handler1, "Derivative");
        printf("Derivative is %f\n", (*Derivative)(atof(argv[3]), atof(argv[4])));
        dlclose(handler1);
    } else {
        handler2 = dlopen("./include/libtrans2.so", RTLD_LAZY);
        if (!handler2) {
            fprintf(stderr, "Open of dynamic library trans2 with error: %s\n", dlerror());
            exit(-2);
        }
        translation = dlsym(handler2, "translation");
        char * answer = (*translation)(atoi(argv[3]));
        printf("result is %s\n", answer);
        dlclose(handler2);
    }
}
else if(atoi(argv[1]) == 1) {
    handler1 = dlopen("./include/libder1.so", RTLD_LAZY);
    if (!handler1) {
        fprintf(stderr, "Open of dynamic library der1 with error: %s\n", dlerror());
        exit(-1);
    }
    Derivative = dlsym(handler1, "Derivative");
    printf("Derivative is %f\n", (*Derivative)(atof(argv[2]), atof(argv[3])));
    dlclose(handler1);
}
else if(atoi(argv[1]) == 2) {
    handler2 = dlopen("./include/libtrans1.so", RTLD_LAZY);
    if (!handler2) {
        fprintf(stderr, "Open of dynamic library trans1 with error: %s\n", dlerror());
        exit(-2);
```

```
            }
            translation = dlsym(handler2, "translation");
            char * answer = (*translation)(atoi(argv[2]));
            printf("result is %s\n", answer);
            dlclose(handler2);
        }
    }
}
```

Пример работы

```
kirill@kirill:~/localProjects/OSlabs/lab5/test$ ./run.sh
[ 16%] Built target trans1
[ 33%] Built target der1
[ 50%] Built target main1
[ 66%] Built target main2
[ 83%] Built target der2
[100%] Built target trans2
> Tests first program
Derivative at a point 10.000000 with delta 0.000100 is 0.544786
Derivative at a point 1.000000 with delta 0.010000 is -0.844157
Derivative at a point 323.000000 with delta 0.000010 is 0.000000
11
101
1010
> Tests second program
Derivative is 0.544786
Derivative is -0.844157
result is 101
result is 1010
Derivative is 0.544760
Derivative is -0.841456
```

```
result is 10
result is 21
```

Вывод

В данной лабораторной работе я познакомился с принципами создания и использования динамических библиотек, реализовал две программы которые используют 2 разных подхода к использованию функций динамической библиотеки: подключение библиотеки на этапе линковки или подключение библиотеки в рантайме, с помощью системных вызовов. Работа с библиотеками является очень важным навыком, необходимым программисту.