Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу

«Операционные системы»

Студент: Юнусов Руслан Асифович
Группа: М8О-209Б-24
Вариант: 24
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/Rissochek/OSLabs/tree/main/lab4

Постановка задачи

Цель работы

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 No1), которая используют одну из библиотек, используя знания полученные на этапе компиляции;

Тестовая программа (программа No2), 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 No2). 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

	Описание	Сигнатура	Реализация 1	Реализация 2
	Подсчёт наибольшего общего делителя для двух натуральных чисел	Int GCF(int A, int B)	Алгоритм Евклида	Наивный алгоритм. Пытаться разделить числа на все числа, что меньше A и B.
	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 в двух файлах: first_programm.c и second_programm.c

Используемые библиотечные вызовы:

void *dlopen(const char *filename, int flag);	Загружает динамическую библиотеку, имя которой указано в строке filename и возвращает прямой указатель на начало загруженной библиотеки.
const char *dlerror(void);	Возвращает указатель на начало строки, описывающей ошибку, полученную на предыдущем вызове.
void *dlsym(void *handle, char *symbol);	Получает параметр handle, который является выходом вызова dlopen и параметр symbol, который является строкой, в которой содержится название символа, который необходимо загрузить из библиотеки. Возвращает указатль на область памяти, в которой содержится необходимый символ.
int dlclose(void *handle);	Уменьшает счетчик ссылок на указатель handle и если он равен нулю, то освобождает библиотеку.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить работу с библиотеками.
2. Реализовать две библиотеки согласно заданию.

3. Реализовать две программы (для работы с динамическими и статическими библиотеками).

Исходный код

First_lib.c

```
#include <stdlib.h>
#include <stdio.h>
int GCF(int a, int b) {
    printf("Function is evclid_gfc\n");
    int c;
    while (b) {
        c = a % b;
        a = b;
        b = c;
    }
    return abs(a);
}

float Square(float a, float b){
    printf("Function is square_square\n");
    return a * b;
}
```

Second_lib.c

```
#include <stdio.h>
#include <stdlib.h>
```

```
int GCF(int a, int b){
    printf("Function is stupid_gfc\n");
    int minimal = 0;
    if (abs(a) < abs(b)){
        minimal = abs(a);
    }else{
        minimal = abs(b);
    }
    for (int i = minimal; i > 0; i--){
        if (a % i == 0 && b % i == 0){
            return i;
        }
    }
}

float Square(float a, float b){
    printf("Function is square_triangle\n");
    return 0.5*a*b;
}
```

first_programm.c

```
#include <stdio.h>
#include <stdlib.h>

extern int GCF(int a, int b);
extern float Square(float a, float b);
```

```
int main(int argc, char** argv){
    if (argc != 7){
        printf("Usage: %s 1 <arg1> <arg2> 2 <arg1> <arg2>\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    printf("Evclid algorithm result: %d\n", GCF(atoi(argv[2]), atoi(argv[3])));
    printf("Stupid algorithm result: %f\n", Square(atof(argv[5]), atof(argv[6])));
}
```

second_programm.c

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

typedef enum{
  first_realization,
  second_realization,
} contract;

contract cont = first_realization;

const char* lib_name_first = "./libs/First_lib.so";
const char* lib_name_second = "./libs/Second_lib.so";

float (*Square)(float, float) = NULL;
int (*GCF)(int, int) = NULL;

void *lib_chosen = NULL;
```

```
void load_libs(contract contract){
    const char* name;
    switch (contract) {
        case first_realization:{
            name = lib_name_first;
            break;
        }
        case second_realization:{
            name = lib_name_second;
            break;
        }
    }
    lib_chosen = dlopen(name, RTLD_LAZY);
    if (lib_chosen == NULL){
        perror("dlopen");
        exit(EXIT_FAILURE);
    }
}

void load_contract(){
  load_libs(cont);
  Square = dlsym(lib_chosen, "Square");
  GCF = dlsym(lib_chosen, "GCF");
}

void change_contract(){
  dlclose(lib_chosen);
  switch (cont) {
          case first_realization:{
```

```
                cont = second_realization;
                break;
            }
            case second_realization:{
              cont = first_realization;
              break;
            }
  }
  load_contract();
}

int main(){
    load_contract();
    int value = 0;
    while (scanf("%d", &value) != EOF){
        switch (value) {
            case 0:{
                change_contract();
                printf("contract has been changed\n");
                switch (cont) {
                case first_realization: {
                    printf("contract is first_realization\n");
                    break;
                }
                case second_realization:{
                    printf("contract is second_realization\n");
                }
            }
            break;
```

```
            }
            case 1:{
                int a, b;
                if (scanf("%d %d", &a, &b) == 2){
                    printf("%d\n", GCF(a, b));
                }
                break;
            }
            case 2:{
                float a, b;
                if (scanf("%f %f", &a, &b) == 2){
                    printf("%f\n", Square(a,b));
                }
            }
        }
    }
    dlclose(lib_chosen);
    return 0;
}
```

Сборка программы

gcc -fPIC -c First_lib.c -o First_lib.o

gcc -fPIC -c Second_lib.c -o Second_lib.o

gcc -shared -o First_lib.so First_lib.o

gcc -shared -o Second_lib.so Second_lib.o

gcc -c first_programm.c -o first.o

gcc -fsanitize=address first.o ./libs/First_lib.so -o first

gcc -fsanitize=address second_programm.c -o second

Демонстрация работы программы

```
parol1@riss:~/Labs/OSLabs/lab4$ ./first 1 2 3 2 5 6
Function is evclid_gfc
Evclid algorithm result: 1
Function is square_square
Stupid algorithm result: 30.000000

parol1@riss:~/Labs/OSLabs/lab4$ ./second
1 5 10
Function is evclid_gfc
5
2 5 3
Function is square_square
15.000000
0
contract has been changed
contract is second_realization
1 5 10
Function is stupid_gfc
5
2 5 3
Function is square_triangle
7.500000
```

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. Динамические библиотеки помогают уменьшить размер исполняемых файлов. Загрузка динамических библиотек во время выполнения также упрощает компиляцию. Однако также можно подключить библиотеку к программе на этапе линковки. Она все равно загрузится при выполнении, но теперь программа будет изначально знать что и где искать. Если библиотека находится не в стандартной для динамических

библиотек директории, необходимо также сообщить линкеру, чтобы тот передал необходимый путь в исполняемый файл. При помощи библиотек мы можем писать более сложные вещи, которые используют простые функции, структуры и т.п., написанные ранее и сохраненные в различных библиотеках.