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: Примаченко Александр Александрович
Группа: М8О-208Б-20
Вариант: 2
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Выводы

Репозиторий

https://github.com/SashaPaladin/OS/tree/main/5_lab

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

Общие сведения о программе

Программа компилируется в двух файлах: static_main.c и dynamic_main.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.

Исходный код

First.cpp

```
#include <iostream>

extern "C" int PrimeCount(int A, int B);
extern "C" float SinIntegral(float A, float B, float e);

int main(){
   int command;
   while((std::cout << "Enter command: ") && (std::cin >> command)){
      if(command == 1){
         std::cout << "Enter A and B: ";
         int a, b;
         std::cin >> a >> b;
         std::cout << "PrimeCount in [a; b] " << PrimeCount(a, b) << std::endl;
      }
      else if(command == 2){
         float A, B, e;
         std::cout << "Enter A, B, e: ";
         std::cin >> A >> B >> e;
         std::cout << "Integral value " << SinIntegral(A, B, e) << std::endl;
      }
   }
}
```

Second.cpp

```
#include <cstdlib>
#include <iostream>
#include <dlfcn.h>

int main(){
   std::cout << "Enter num library: ";
   int lib_num;
```

```
std::cin >> lib_num;
if(lib_num < 1 || lib_num > 2){
   std::cout << "error lib\n";
   exit(1);
}
--lib_num;
int command;
const char* libs[] = {"libd1.so", "libd2.so"};
void* library_handle;
library_handle = dlopen (libs[lib_num], RTLD_LAZY);
if(!library_handle){
   std::cout << "Error in dlopen\n";
   exit(1);
}

int (*PrimeCount)(int A, int B);
float (*SinIntegral)(float A, float B, float e);

PrimeCount = (int(*)(int, int))dlsym(library_handle, "PrimeCount");
SinIntegral = (float(*)(float, float, float))dlsym(library_handle, "SinIntegral");

std::cout << "Enter command 0, 1 or 2\n";
while(std::cin >> command) {
   switch (command) {
      case 0:
         dlclose(library_handle);
         lib_num = (lib_num + 1) % 2;
         library_handle = dlopen(libs[lib_num], RTLD_LAZY);
         if(!library_handle){
            std::cout << "Error in dlopen\n";
            exit(1);
         }
         PrimeCount = (int(*)(int, int))dlsym(library_handle, "PrimeCount");
         SinIntegral = (float(*)(float, float, float))dlsym(library_handle, "SinIntegral");
         std::cout << "Change contract\n";
         break;
      case 1:
         std::cout << "Enter A and B: ";
         int a, b;
         std::cin >> a >> b;
         std::cout << "PrimeCount in [a; b] " << PrimeCount(a, b) << std::endl;
         break;
      case 2:
         float A, B, e;
         std::cout << "Enter A, B, e: ";
         std::cin >> A >> B >> e;
         std::cout << "Integral value " << SinIntegral(A, B, e) << std::endl;
         break;
      default:
         std::cout << "Enter 0, 1 or 2!\n";
         break;
   }
}
```

```
    dlclose(library_handle);
}
```

Lib1.cpp

```
#include <cmath>

extern "C" int PrimeCount(int A, int B);
extern "C" float SinIntegral(float A, float B, float e);

int PrimeCount(int A, int B) {
    int count = 0;
    if (B < 2)
        return 0;
    if (A < 3) {
        A = 3;
        ++count;
    }
    for (int number = A; number <= B; ++number) {
        for (int divider = 2; divider < number; ++divider) {
            if (number % divider == 0)
                break;
            if (divider == number - 1)
                ++count;
        }
    }
    return count;
}


float SinIntegral(float A, float B, float e) {
    float rectangle_integral = 0;
    for(float step = A; step + e < B; step+= e)
    {
        float x1 = step;
        float x2 = (step + e < B)?step+e:B;
        rectangle_integral += 0.5*(x2-x1)*(sin(x1) + sin(x2));
    }

    return rectangle_integral;
}
```

Lib2.cpp

```
#include <vector>
#include <cmath>

extern "C" int PrimeCount(int A, int B);
extern "C" float SinIntegral(float A, float B, float e);

int PrimeCount(int A, int B){
    if (B < 2)
        return 0;
    if(A < 2)
        A = 2;
```

```
    int n = B;
    std::vector<char> prime(n + 1, true);
    prime[0] = prime[1] = false;
    for(int i = 2; i <= n; ++i){
        if(prime[i]){
            if(i * i <= n){
                for(int j = i * i; j <= n; j += i){
                    prime[j] = false;
                }
            }
        }
    }
    int count = 0;
    for(int i = A; i <= B; ++i)
        count += prime[i];
    return count;
}

float SinIntegral(float A, float B, float e) {
    float trapezoidal_integral = 0;
    for(float step = A; step + e < B; step+= e)
    {
        float x1 = step;
        float x2 = (step + e < B)?step+e:B;
        trapezoidal_integral += (x2-x1)*sin(x1) + 0.5*(x2-x1) * (sin(x2) - sin(x1));
    }

    return trapezoidal_integral;
}
```

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.