Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ**

Студент: Жаворонков Михаил Николаевич

Группа: М8О–212Б–22

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**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:

* 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»;
* «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;
* «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.

**Функция 1:**

Описание: Подсчёт количества простых чисел на отрезке [A, B] (A, B - натуральные)

Сигнатура: Int PrimeCount(int A, int B)

Реализация 1: Наивный алгоритм. Проверить делимость текущего числа на все предыдущие числа.

Реализация 2: Решето Эратосфена

**Функция 2:**

Описание: Отсортировать целочисленный массив

Сигнатура: int \*sort(int\* array)

Реализация 1: пузырьковая сортировка

Реализация 2: сортировка Хоара

**Общие сведения о программе**

*func.hpp* – Заголовочный файл для функций;

*realization1.cpp* – Содержит реализацию 1, данных функций;

*realization2.cpp* – Содержит реализацию 2, данных функций;

*main\_1.cpp* - Тестовая программа (программа №1), которая используют одну из библиотек, используя знания полученные на этапе компиляции;

*main\_2.cpp* - Тестовая программа (программа №2), которая загружает библиотеки, используя только их местоположение и контракты;

CMakeLists.txt– Компиляция программ main\_1.c и main\_2.c;

**Общий метод и алгоритм решения**.

Для реализации, поставленной задачи необходимо:

1. Реализовать заданные функции двумя способами, в соответствии с заданием
2. Написать тестовую программу №1 использовать данные библиотеки во время компиляции
3. Написать тестовую программу №2 использовать данные библиотеки Во время исполнения программы.
4. Скомпилировать и запустить программу.

**Основные файлы программы**

**func.h**

#pragma once

int PrimeCount(int A, int B);

int \*sort(int\*array,int size);

**realization1.c**

#include <stdio.h>

#include "funcs.h"

#include <math.h>

#include <stdbool.h>

int isprime(int n){

if(n<=1){

return 0;

}

int sqrtN=sqrt(n);

for(int i = 2 ; i <=sqrtN;++i ){

if(n%i==0){

return 0;

}

}

return 1;

}

int PrimeCount(int A,int B){

int counter=0;

for(int i = A ; i<=B;++i){

if(isprime(i)){

counter+=1;

}

}

return counter;

}

int\* sort(int\* array, int size) {

bool swapped;

do {

swapped = false;

for (int i = 0; i < size - 1; ++i) {

if (array[i] > array[i + 1]) {

int tmp = array[i];

array[i] = array[i + 1];

array[i + 1] = tmp;

swapped = true;

}

}

} while (swapped);

return array;

}

**realization2.cpp**

#include <stdio.h>

#include <math.h>

#include "funcs.h"

#include <stdlib.h>

#include <stdbool.h>

int PrimeCount(int A,int B){ // решето

int N=B-A+1;

int\* massiv = (int\*)malloc(N \* sizeof(int));

if (massiv == NULL) {

printf("Memory allocation failed.\n");

return -1;

}

int counter=0;

for(int i=2;i\*i<N;++i){

if(massiv[i]==0){

for(int k=i\*i;k<N;k+=i){

massiv[k]=1;

}

}

}

for(int i=2;i<N;++i){

if(massiv[i]==0){

counter+=1;

}

}

return counter;

}

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int partition(int \*array, int low, int high) {

int pivot = array[low];

int left = low + 1;

int right = high;

while (true) {

while (left <= right && array[left] < pivot) {

++left;

}

while (right >= left && array[right] > pivot) {

--right;

}

if (left <= right) {

swap(&array[left], &array[right]);

} else {

break;

}

}

swap(&array[low], &array[right]);

return right;

}

void quicksort(int \*array, int low, int high) {

if (low < high) {

int pivotIndex = partition(array, low, high);

quicksort(array, low, pivotIndex - 1);

quicksort(array, pivotIndex + 1, high);

}

}

int \*sort(int \*array, int size) {

quicksort(array, 0, size - 1);

return array;

}

**main\_1.c**

#include <stdio.h>

#include <stdlib.h>

#include "funcs.h"

int main(){

printf("1 arg1 arg2 - PrimeCount function\n");

printf("2 arg1 your array - Sort function\n");

printf("3 - exit\n");

int command;

int \*array = NULL;

printf("Input command: ");

scanf("%d",&command);

while(command != 3){

switch (command){

case 1:

int x,y;

printf("\nInput args: ");

scanf("%d %d",&x,&y);

printf("-----------------\n");

printf("PrimeCount(%d, %d) = %d\n",x,y,PrimeCount(x,y));

printf("-----------------\n");

printf("\nInput command: ");

scanf("%d",&command);

break;

case 2:

int size;

printf("Input the size of your array: ");

scanf("%d", &size);

array = (int \*)malloc(size \* sizeof(int));

printf("Input your array: ");

for (int i = 0; i < size; ++i) {

scanf("%d", &array[i]);

}

int \*sortedArray = sort(array, size);

printf("Sorted array: ");

for (int i = 0; i < size; ++i) {

printf("%d ", sortedArray[i]);

}

printf("\n");

free(sortedArray);

printf("\nInput command: ");

scanf("%d",&command);

break;

case 3:

break;

default:

printf("\nIncorrect Input\n");

exit(1);

break;

}

}

}

**main\_2.c**

#include <stdio.h>

#include <dlfcn.h>

#include <string.h>

#include <stdlib.h>

int main(){

printf("0 - Change realization\n");

printf("1 arg1 arg2 - PrimeCount function\n");

printf("2 arg1 your array - Sort function\n");

printf("3 - exit\n");

int flag\_lib = 1; // номер текущей библиотеки

void \*library\_handler; // прямой указатель на начало динамической библиотеки

int \*array = NULL;

int size;

library\_handler = dlopen("librealization1.so",RTLD\_LAZY);

if (!library\_handler){

// если ошибка, то вывести ее на экран

fprintf(stderr,"dlopen() error: %s\n", dlerror());

exit(1); // в случае ошибки закончить работу программу

};

int command;

printf("Input command: ");

scanf("%d",&command);

while(command != 3){

switch (command){

case 0:

dlclose(library\_handler); // закрываю текущую библиотеку

if(flag\_lib == 1){

library\_handler = dlopen("librealization2.so",RTLD\_LAZY);

flag\_lib = 2;

}

else{

library\_handler = dlopen("librealization1.so",RTLD\_LAZY);

flag\_lib = 1;

}

printf("\nREALIZATION CHANGED\n");

printf("Input command: ");

scanf("%d",&command);

break;

case 1:

int x,y;

printf("\nInput args: ");

scanf("%d %d",&x,&y);

printf("-----------------\n");

int (\*func\_PrimeCount)(int, int); // указатель на функцию

char name1[] = "PrimeCount";

func\_PrimeCount = dlsym(library\_handler,name1);

if (!func\_PrimeCount){

fprintf(stderr,"dlopen() error: %s\n", dlerror());

exit(1);

};

if(flag\_lib == 1) printf("\nPrimeCount realization №1\n");

else printf("\nPrimeCount realization №2\n");

printf("PrimeCount(%d, %d) = %d\n",x,y,(\*func\_PrimeCount)(x, y));

printf("-----------------\n");

printf("\nInput command: ");

scanf("%d",&command);

break;

case 2:

printf("Input the size of your array: ");

scanf("%d", &size);

array = (int \*)malloc(size \* sizeof(int));

printf("Input your array: ");

for (int i = 0; i < size; ++i) {

scanf("%d", &array[i]);

}

void (\*func\_sort)(int \*, int);

char name2[]="sort";

if(flag\_lib == 1) printf("\nSort realization №1\n");

else printf("\nSort realization №2\n");

func\_sort = (void (\*)(int \*, int))dlsym(library\_handler, name2);

if (!func\_sort) {

fprintf(stderr, "sort function error: %s\n", dlerror());

return 1;

}

func\_sort(array, size);

printf("Sorted array: ");

for (int i = 0; i < size; ++i) {

printf("%d ", array[i]);

}

printf("\n");

free(array);

printf("\nInput command: ");

scanf("%d",&command);

break;

case 3:

break;

default:

printf("\nIncorrect Input\n");

exit(1);

break;

}

}

dlclose(library\_handler);

}

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.10)

project(OS\_lab4)

set(CMAKE\_C\_STANDARD 11)

add\_library(realization1 SHARED realization1.c)

target\_link\_libraries(realization1 PRIVATE m)

add\_library(realization2 SHARED realization2.c)

target\_link\_libraries(realization2 PRIVATE m)

add\_executable(first main\_1.c)

target\_link\_libraries(first PRIVATE realization1)

add\_executable(second main\_2.c)

target\_link\_libraries(second PRIVATE realization2)

**Пример работы**

**zhavkk@zhavkk:~/OSlabs/lab4$ ./first**

**1 arg1 arg2 - PrimeCount function**

**2 arg1 your array - Sort function**

**3 - exit**

**Input command: 1 1 25**

**Input args: -----------------**

**PrimeCount(1, 25) = 9**

**-----------------**

**Input command: 1 4 100**

**Input args: -----------------**

**PrimeCount(4, 100) = 23**

**-----------------**

**Input command: 2 5 5 4 3 2 1**

**Input the size of your array: Input your array: Sorted array: 1 2 3 4 5**

**Input command: 2**

**Input the size of your array: 5**

**Input your array: 5 4 3 2 1**

**Sorted array: 1 2 3 4 5**

**Input command: 3**

**zhavkk@zhavkk:~/OSlabs/lab4$ ./second**

**0 - Change realization**

**1 arg1 arg2 - PrimeCount function**

**2 arg1 your array - Sort function**

**3 - exit**

**Input command: 1 1 25**

**Input args: -----------------**

**PrimeCount realization №1**

**PrimeCount(1, 25) = 9**

**-----------------**

**Input command: 1 5 100**

**Input args: -----------------**

**PrimeCount realization №1**

**PrimeCount(5, 100) = 23**

**-----------------**

**Input command: 2 5 5 4 3 2 1**

**Input the size of your array: Input your array:**

**Sort realization №1**

**Sorted array: 1 2 3 4 5**

**Input command: 2**

**Input the size of your array: 5**

**Input your array: 5 3 4 2 1**

**Sort realization №1**

**Sorted array: 1 2 3 4 5**

**Input command: 0**

**REALIZATION CHANGED**

**Input command: 1 1 29**

**Input args: -----------------**

**PrimeCount realization №2**

**PrimeCount(1, 29) = 3**

**-----------------**

**Input command: 2 5 5 4 3 2 1**

**Input the size of your array: Input your array:**

**Sort realization №2**

**Sorted array: 1 2 3 4 5**

**Input command: 3**

**zhavkk@zhavkk:~/OSlabs/lab4$**

**Вывод**

В результате лабораторной работы мной были получены следующие основные части: динамические библиотеки, реализующие заданные контракты, тестовая программа №1, которая использует одну из библиотек на этапе компиляции, и тестовая программа №2, которая загружает библиотеки только по их местоположению.

Использование динамической библиотеки во время компиляции означает, что код библиотеки подключается к программе во время ее компиляции. Это позволяет компилятору проверить правильность использования функций и переменных из библиотеки и включить необходимые зависимости в исполняемый файл.

С другой стороны, использование динамической библиотеки во время исполнения программы означает, что код библиотеки загружается и подключается к программе только во время ее выполнения. Это позволяет сократить размер исходного кода программы и обеспечить более гибкую работу с библиотеками, поскольку можно обновлять их без перекомпиляции программы.

Таким образом, выбор между использованием библиотек на этапе компиляции или на этапе исполнения зависит от требований конкретного проекта и наличия нужного функционала в библиотеках.