Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

**Тема работы**

**“Динамические библиотеки”**

Студент: Ковриженков Дмитрий Олегович

Группа: М8О-203Б-23

Вариант: 26

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Постановка задачи**

**Задача:** 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.

**Задание 4.** Подсчёт наибольшего общего делителя для двух натуральных чисел. Сигнатура: Int GCF(int A, int B), Реализация 1: Алгоритм Евклида Реализация 2: Наивный алгоритм. Пытаться разделить числа на все числа, что меньше A и B.

**Задание 9.** Отсортировать целочисленный массив, Сигнатура: Int \* Sort(int \* array), Реализация 1: Пузырьковая сортировка 2: Сортировка Хоара

**Общие сведения**

В данной лабораторной работе реализуется работа с динамически загружаемыми библиотеками (dlopen, dlsym, dlclose) для вычисления наибольшего общего делителя (GCF) и сортировки массива. Программа предоставляет две реализации GCF: Алгоритм Евклида (GCF\_Euclidean) – эффективный метод нахождения НОД. Простой перебор (GCF\_Naive) – менее эффективный алгоритм, проверяющий все делители. Для сортировки массива реализованы: Пузырьковая сортировка (Sort\_Bubble) – простой, но медленный алгоритм. Быстрая сортировка (Sort\_Quick) – более эффективный метод разбиения массива. Используются два модуля-обертки (gcf\_wrapper.c и sort\_wrapper.c), позволяющие переключать используемую функцию на лету. В program1 (main.c): Статическая компоновка с библиотеками gcf и sort. Функции set\_gcf\_function() и set\_sort\_function() позволяют выбирать используемый алгоритм. В program2 (main.c): Динамическая загрузка библиотек с помощью dlopen(). dlsym() используется для получения адресов функций. Возможность переключения алгоритмов во время работы программы. Сборка выполняется с помощью CMake, создавая динамические библиотеки libgcf.so и libsort.so.

**Вывод**

В процессе выполнения лабораторной работы были изучены: Динамическая и статическая компоновка библиотек. Работа с dlopen() и dlsym() для динамической загрузки. Различные алгоритмы GCF и сортировки. Использование функций-оберток для динамического переключения алгоритмов. Данная лабораторная работа демонстрирует основы модульности кода и гибкости программных решений при использовании динамических библиотек.

**Приложение**

src/gcf\_euclidean.c

#include "gcf.h"

int GCF\_Euclidean(int A, int B) {

while (B != 0) {

int temp = B;

B = A % B;

A = temp;

}

return A;

}

src/gcf\_naive.c

#include "gcf.h"

int GCF\_Naive(int A, int B) {

if (A == 0 && B == 0) {

return 0;

}

if (A == 0) {

return (B > 0) ? B : -B;

}

if (B == 0) {

return (A > 0) ? A : -A;

}

if (A < 0) A = -A;

if (B < 0) B = -B;

int min = (A < B) ? A : B;

int gcf = 1;

for (int i = 1; i <= min; ++i) {

if (A % i == 0 && B % i == 0) {

gcf = i;

}

}

return gcf;

}

src/gcf\_wrapper.c

#include "gcf.h"

static GCF\_Function current\_gcf = GCF\_Euclidean;

void set\_gcf\_function(GCF\_Function func) {

current\_gcf = func;

}

GCF\_Function get\_gcf\_function() {

return current\_gcf;

}

int GCF(int A, int B) {

return current\_gcf(A, B);

}

src/sort\_bubble.c

#include "sort.h"

void Sort\_Bubble(int\* array, int size) {

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

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

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

int temp = array[j];

array[j] = array[j+1];

array[j+1] = temp;

}

}

}

}

src/sort\_quick.c

#include "sort.h"

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

int pivot = array[high];

int i = low -1;

for (int j = low; j < high; ++j) {

if (array[j] < pivot) {

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i+1];

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

array[high] = temp;

return i+1;

}

void quick\_sort(int\* array, int low, int high) {

if (low < high) {

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

quick\_sort(array, low, pi -1);

quick\_sort(array, pi +1, high);

}

}

void Sort\_Quick(int\* array, int size) {

quick\_sort(array, 0, size -1);

}

src/sort\_wrapper.c

#include "sort.h"

static Sort\_Function current\_sort = Sort\_Bubble;

void set\_sort\_function(Sort\_Function func) {

current\_sort = func;

}

Sort\_Function get\_sort\_function() {

return current\_sort;

}

void Sort(int\* array, int size) {

current\_sort(array, size);

}

include/ gcf.h

#ifndef GCF\_H

#define GCF\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

typedef int (\*GCF\_Function)(int, int);

void set\_gcf\_function(GCF\_Function func);

GCF\_Function get\_gcf\_function();

int GCF(int A, int B);

int GCF\_Euclidean(int A, int B);

int GCF\_Naive(int A, int B);

#ifdef \_\_cplusplus

}

#endif

#endif

include/ sort.h

#ifndef SORT\_H

#define SORT\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

typedef void (\*Sort\_Function)(int\*, int);

void set\_sort\_function(Sort\_Function func);

Sort\_Function get\_sort\_function();

void Sort(int\* array, int size);

void Sort\_Bubble(int\* array, int size);

void Sort\_Quick(int\* array, int size);

#ifdef \_\_cplusplus

}

#endif

#endif

programs/program1/main.c

#include <stdio.h>

#include "gcf.h"

#include "sort.h"

int main() {

set\_gcf\_function(GCF\_Euclidean);

set\_sort\_function(Sort\_Bubble);

int A = 48, B = 18;

int gcf = GCF(A, B);

printf("GCF(%d, %d) = %d\n", A, B, gcf);

int array[] = {5, 2, 9, 1, 5, 6};

int size = sizeof(array)/sizeof(array[0]);

printf("Original array: ");

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

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

}

printf("\n");

Sort(array, size);

printf("Sorted array: ");

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

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

}

printf("\n");

return 0;

}

programs/program1/ CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(program1 C)

include\_directories(${CMAKE\_SOURCE\_DIR}/include)

add\_executable(program1 main.c)

target\_link\_libraries(program1 PRIVATE gcf sort)

install(TARGETS program1 DESTINATION bin)

programs/program2/main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dlfcn.h>

#include "gcf.h"

#include "sort.h"

typedef void (\*SetGCFFunction)(GCF\_Function);

typedef void (\*SetSortFunction)(Sort\_Function);

typedef GCF\_Function (\*GetGCFFunction)();

typedef Sort\_Function (\*GetSortFunction)();

typedef int (\*GCF)(int, int);

typedef void (\*Sort)(int\*, int);

int main() {

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

if (!handle\_gcf) {

fprintf(stderr, "Ошибка загрузки libgcf.so: %s\n", dlerror());

exit(EXIT\_FAILURE);

}

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

if (!handle\_sort) {

fprintf(stderr, "Ошибка загрузки libsort.so: %s\n", dlerror());

dlclose(handle\_gcf);

exit(EXIT\_FAILURE);

}

dlerror();

SetGCFFunction set\_gcf\_function = (SetGCFFunction)dlsym(handle\_gcf, "set\_gcf\_function");

char\* error = dlerror();

if (error != NULL) {

fprintf(stderr, "Ошибка поиска set\_gcf\_function: %s\n", error);

dlclose(handle\_gcf);

dlclose(handle\_sort);

exit(EXIT\_FAILURE);

}

SetSortFunction set\_sort\_function = (SetSortFunction)dlsym(handle\_sort, "set\_sort\_function");

error = dlerror();

if (error != NULL) {

fprintf(stderr, "Ошибка поиска set\_sort\_function: %s\n", error);

dlclose(handle\_gcf);

dlclose(handle\_sort);

exit(EXIT\_FAILURE);

}

GCF GCF\_func = (GCF)dlsym(handle\_gcf, "GCF");

error = dlerror();

if (error != NULL) {

fprintf(stderr, "Ошибка поиска GCF: %s\n", error);

dlclose(handle\_gcf);

dlclose(handle\_sort);

exit(EXIT\_FAILURE);

}

Sort Sort\_func = (Sort)dlsym(handle\_sort, "Sort");

error = dlerror();

if (error != NULL) {

fprintf(stderr, "Ошибка поиска Sort: %s\n", error);

dlclose(handle\_gcf);

dlclose(handle\_sort);

exit(EXIT\_FAILURE);

}

printf("Динамическая загрузка библиотек успешна.\n");

set\_gcf\_function(GCF\_Euclidean);

set\_sort\_function(Sort\_Bubble);

char command[256];

while (1) {

printf("\nВведите команду:\n");

printf("0 - переключить реализации функций\n");

printf("1 A B - вычислить GCF(A, B)\n");

printf("2 size elem1 elem2 ... elemN - отсортировать массив\n");

printf("q - выход\n");

printf("Команда: ");

if (!fgets(command, sizeof(command), stdin)) {

break;

}

command[strcspn(command, "\n")] = 0;

if (strcmp(command, "q") == 0) {

break;

}

if (strcmp(command, "0") == 0) {

static int gcf\_switch = 0;

if (gcf\_switch == 0) {

set\_gcf\_function(GCF\_Naive);

printf("Реализация GCF переключена на Naive.\n");

gcf\_switch = 1;

} else {

set\_gcf\_function(GCF\_Euclidean);

printf("Реализация GCF переключена на Euclidean.\n");

gcf\_switch = 0;

}

static int sort\_switch = 0;

if (sort\_switch == 0) {

set\_sort\_function(Sort\_Quick);

printf("Реализация Sort переключена на Quick Sort.\n");

sort\_switch = 1;

} else {

set\_sort\_function(Sort\_Bubble);

printf("Реализация Sort переключена на Bubble Sort.\n");

sort\_switch = 0;

}

continue;

}

char\* token = strtok(command, " ");

if (token == NULL) {

continue;

}

if (strcmp(token, "1") == 0) {

char\* arg1 = strtok(NULL, " ");

char\* arg2 = strtok(NULL, " ");

if (arg1 == NULL || arg2 == NULL) {

printf("Недостаточно аргументов для команды 1.\n");

continue;

}

int A = atoi(arg1);

int B = atoi(arg2);

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC, &start);

int result = GCF\_func(A, B);

clock\_gettime(CLOCK\_MONOTONIC, &end);

double time\_taken = (end.tv\_sec - start.tv\_sec) \* 1e3 +

(end.tv\_nsec - start.tv\_nsec) / 1e6;

printf("GCF(%d, %d) = %d\n", A, B, result);

printf("Время выполнения GCF: %.6f ms\n", time\_taken);

}

else if (strcmp(token, "2") == 0) {

char\* arg\_size = strtok(NULL, " ");

if (arg\_size == NULL) {

printf("Недостаточно аргументов для команды 2.\n");

continue;

}

int size = atoi(arg\_size);

if (size <= 0) {

printf("Некорректный размер массива.\n");

continue;

}

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

if (!array) {

printf("Не удалось выделить память для массива.\n");

continue;

}

int valid = 1;

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

char\* elem = strtok(NULL, " ");

if (elem == NULL) {

printf("Недостаточно элементов для массива.\n");

free(array);

valid = 0;

break;

}

array[i] = atoi(elem);

}

if (!valid) {

continue;

}

printf("Исходный массив: ");

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

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

}

printf("\n");

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC, &start);

Sort\_func(array, size);

clock\_gettime(CLOCK\_MONOTONIC, &end);

double time\_taken = (end.tv\_sec - start.tv\_sec) \* 1e3 +

(end.tv\_nsec - start.tv\_nsec) / 1e6;

printf("Отсортированный массив: ");

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

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

}

printf("\n");

printf("Время выполнения сортировки: %.6f ms\n", time\_taken);

free(array);

}

else {

printf("Неизвестная команда.\n");

}

}

dlclose(handle\_gcf);

dlclose(handle\_sort);

return 0;

}

programs/program2/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.10)

project(program2 C)

include\_directories(${CMAKE\_SOURCE\_DIR}/include)

add\_executable(program2 main.c)

target\_link\_libraries(program2 PRIVATE gcf sort dl)

install(TARGETS program2 DESTINATION bin)

tests/gcf\_test.cpp

#include <gtest/gtest.h>

#include "gcf.h"

TEST(GCFTest, Euclidean) {

set\_gcf\_function(GCF\_Euclidean);

EXPECT\_EQ(GCF(48, 18), 6);

EXPECT\_EQ(GCF(100, 10), 10);

EXPECT\_EQ(GCF(7, 3), 1);

EXPECT\_EQ(GCF(0, 5), 5);

EXPECT\_EQ(GCF(5, 0), 5);

EXPECT\_EQ(GCF(0, 0), 0);

}

TEST(GCFTest, Naive) {

set\_gcf\_function(GCF\_Naive);

EXPECT\_EQ(GCF(48, 18), 6);

EXPECT\_EQ(GCF(100, 10), 10);

EXPECT\_EQ(GCF(7, 3), 1);

EXPECT\_EQ(GCF(0, 5), 5);

EXPECT\_EQ(GCF(5, 0), 5);

EXPECT\_EQ(GCF(0, 0), 0);

}

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

tests/sort\_test.cpp

#include <gtest/gtest.h>

#include "sort.h"

#include <algorithm>

#include <time.h>

bool is\_sorted\_correctly(const int\* sorted, const int\* expected, int size) {

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

if(sorted[i] != expected[i]) {

return false;

}

}

return true;

}

TEST(SortTest, BubbleSort) {

set\_sort\_function(Sort\_Bubble);

{

int array[] = {5, 2, 9, 1, 5, 6};

int expected[] = {1, 2, 5, 5, 6, 9};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int array[] = {3, 0, -1, 8, 7, 2};

int expected[] = {-1, 0, 2, 3, 7, 8};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int array[] = {1};

int expected[] = {1};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int\* array = nullptr;

int\* expected = nullptr;

int size = 0;

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

}

TEST(SortTest, QuickSort) {

set\_sort\_function(Sort\_Quick);

{

int array[] = {5, 2, 9, 1, 5, 6};

int expected[] = {1, 2, 5, 5, 6, 9};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int array[] = {3, 0, -1, 8, 7, 2};

int expected[] = {-1, 0, 2, 3, 7, 8};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int array[] = {1};

int expected[] = {1};

int size = sizeof(array)/sizeof(array[0]);

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

{

int\* array = nullptr;

int\* expected = nullptr;

int size = 0;

Sort(array, size);

EXPECT\_TRUE(is\_sorted\_correctly(array, expected, size));

}

}

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

Пример вывода:

