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

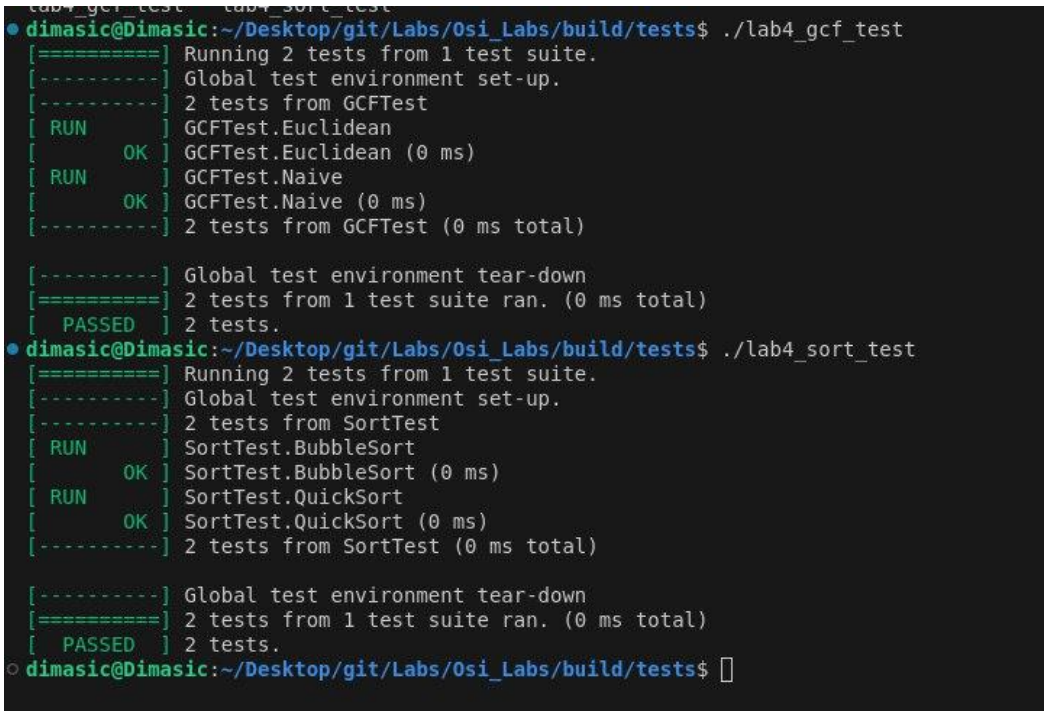
```
#include <gtest/gtest.h>
#include "sort.h"
#include <algorithm>
#include <time.h>
```

```
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:


dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/build/tests$ ./lab4_gcf_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from GCFTest
[RUN] GCFTest.Euclidean
[OK] GCFTest.Euclidean (0 ms)
[RUN] GCFTest.Naive
[OK] GCFTest.Naive (0 ms)
[----------] 2 tests from GCFTest (0 ms total)
[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[PASSED] 2 tests.
dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/build/tests$ ./lab4_sort_test
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from SortTest
[RUN] SortTest.BubbleSort
[OK] SortTest.BubbleSort (0 ms)
[RUN] SortTest.QuickSort
[OK] SortTest.QuickSort (0 ms)
[----------] 2 tests from SortTest (0 ms total)
[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[PASSED] 2 tests.
dimasic@Dimasic:~/Desktop/git/Labs/Osi_Labs/build/tests$