Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-214БВ-24

Студент: Чернявская Алиса Алексеевна

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 17.11.25

Москва, 2025

Постановка задачи

Вариант 15.

Задание

Требуется создать динамические библиотеки, которые реализуют функции вычисления производной косинуса в точке с приращением двумя вариантами и сортировку массива двумя видами. . Далее использовать данные библиотеки 2-мя способами:

1. Во время компиляции (на этапе линковки)

2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками.

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).

“1 arg1 arg2 … argN”, где после “1” идут аргументы для первой функции (производная косинуса)

“2 arg1 arg2 … argM”, где после “2” идут аргументы для второй функции) (сортировка массива).

Общий метод и алгоритм решения

Использованные системные вызовы:

- LoadLibraryA() — загрузка динамической библиотеки (DLL) в адресное пространство процесса
- GetProcAddress() — получение адреса функции из загруженной библиотеки
- FreeLibrary() — выгрузка библиотеки из памяти и освобождение ресурсов
- GetStdHandle() — получение handle (дескриптора) стандартных потоков ввода/вывода
- WriteFile() — запись данных в файл или поток (аналог printf/puts)
- ReadFile() — чтение данных из файла или потока (аналог scanf/fgets)

В программе реализованы 2 типа используемых библиотек – статические (которые программа видит на этапе линковки) и динамические, которые программа загружает прямо во время выполнения. В первой программе библиотеки линкуются через командную строк (gcc program1.c cos1.c sort1.c -o program1.exe -lm -luser32). В первой программе на вход должно податься число 0, 1, или 2. Если пользователь ввел 0 – выводится информация об используемых библиотеках. Если пользователь вводит 1, то он также через пробел должен ввести два аргумента – точку и приращение, по которым потом считается производная косинуса с помощью заранее загруженных библиотек cos1.c, cos2.c. Если пользователь вводит 2 – он также следующим числом через пробел вводит количество элементов массива, а потом их количество. Какими библиотеками будут реализоваться функции, зависит от того, что мы слинковали. Дальше программа проверяет правильность введенных данных и вызывает нужные функции для расчетов. Потом выводится результат.

Вторая программа используется с загрузкой динамических библиотек, то есть заранее ОС не знает, какие функции будут использоваться. Сначала загружаются динамические библиотеки – записываются имена нужных библиотек (в зависимости от реализации) в специальные массивы. Затем эти массивы с помощью функции LoadLibraryA загружаются в память и мы получаем дескриптор библиотеки. Потом по дескрипторам получаем указатели на нужные функции с помощью GetProcAddress(), который находит указатель на функцию через ее имя. Затем в основном цикле программы пользователь так же вводит числа, как и в первой программе, отличие в том, что если введен 0 – переключается реализация функций. Происходят расчеты, результаты выводятся на экран и высвобождается вся память.

Код программы

cos1.c

```c
#include <math.h>

float cosdx(float a, float dx)
{
    return (cosf(a + dx) - cosf(a)) / dx;

 }
```

cos2.c

```c
#include <math.h>

float cosdx(float a, float dx)
{
    return (cosf(a + dx) - cosf(a - dx)) / (2 * dx);

 }
```

sort1.c

```c
//пузырьковая сортировка!!!

#include <stdlib.h>
#include <string.h>

int* sort(int* array, size_t n)
{
    if (n == 0) return NULL;

    // выделяем память под копию массива
    int* result = (int*)malloc(n * sizeof(int));
    if (!result) return NULL;
    memcpy(result, array, n * sizeof(int));//копируем исходный массив в динамический

    // реализация сортировки
    for (size_t i = 0; i < n - 1; i++)
    {
        for (size_t j = 0; j < n - i - 1; j++)
        {//тут сравниваем соседние, и если они не по порядку, меняем местами через temp
            if (result[j] > result[j + 1])
            {
                int temp = result[j];
                result[j] = result[j + 1];
                result[j + 1] = temp;
            }
        }
```

```
    }

    return result;

  }
```

sort2.c

```
//сортировка кучей!!!!

#include <stdlib.h>
#include <string.h>

static void quick_sort(int* arr, int low, int high) //2 и 3 аргументы это индексы
{
    if (low < high)
    {
        int central = arr[(low + high) / 2];//серединка
        int i = low - 1;//индекс слева
        int j = high + 1;//индекс справа

        while (1)
        {
            do { i++; }
            while (arr[i] < central);//поиск элемента больше серединки слева

            do { j--; }
            while (arr[j] > central);//поиск элемента меньше серединки справа

            if (i >= j)
                break;//если индексы пересеклись

            int temp = arr[i];//меняем местами
            arr[i] = arr[j];
            arr[j] = temp;
        }
        //рекурсия: сортирум левую и правую части)
        quick_sort(arr, low, j);
        quick_sort(arr, j + 1, high);
    }
}

int* sort(int* array, size_t n)
{
    if (n == 0) return NULL;

    // копируем массив в динамический как в пузырьковой сортировке
    int* result = (int*)malloc(n * sizeof(int));
    if (!result)
        return NULL;
    memcpy(result, array, n * sizeof(int));

    // выполняем сортировку
    quick_sort(result, 0, n - 1);

    return result;

  }
```

program1.c

```
//статическая линковка: подключение библиотеки на этапе компиляции, т.е. программа знает о
функциях заранее
//и ос сразу загружает библиотеки

#include "contract.h"
#include <stdlib.h>
#include <string.h>
```

```c
#include <stdio.h>
#include <windows.h>

// Функции ввода/вывода
void win_write(const char* str)
{
    HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);//дескриптор вывода
    DWORD bytesWritten;//количество записанных байт
    WriteFile(hStdout, str, (DWORD)strlen(str), &bytesWritten, NULL);//вывод
}

void win_write_float(float value) //вывод числа float для косинуса
{
    char buffer[64];
    snprintf(buffer, sizeof(buffer), "%.6f\n", value);
    win_write(buffer);
}

void win_write_array(int* arr, size_t n) //вывод массива
{
    char buffer[64];
    for (size_t i = 0; i < n; i++)
    {
        snprintf(buffer, sizeof(buffer), "%d ", arr[i]);
        win_write(buffer);
    }
    win_write("\n");
}

int win_read_line(char* buffer, size_t size)//ввод
{
    HANDLE hStdin = GetStdHandle(STD_INPUT_HANDLE);
    DWORD bytesRead;

    if (!ReadFile(hStdin, buffer, (DWORD)size - 1, &bytesRead, NULL))
    {
        return 0;
    }

    buffer[bytesRead] = '\0';

    // Убираем \r и \n
    char* cr = strchr(buffer, '\r');
    if (cr)
        *cr = '\0';
    char* lf = strchr(buffer, '\n');
    if (lf)
        *lf = '\0';

    return 1;
}

int main()
{
    char input[256];//буфер для ввода

    win_write(" program 1: static linking \n");
    win_write(" commands:\n");
    win_write("  0 - show library info\n");
    win_write("  1 a dx - derivative of cos(x) at point 'a' with step 'dx'\n");
    win_write("  2 n a1 a2 ... an - sort array of n numbers\n");
    win_write("  exit - exit program\n");

    while (1)
    {
        win_write("\n> ");

        if (!win_read_line(input, sizeof(input)))
        {
```

```
        break;
    }

    // если выход
    if (strcmp(input, "exit") == 0)
    {
        win_write("exiting\n");
        break;
    }

    if (strlen(input) == 0) //может быть введен только 1 символ
    {
        continue;
    }

    // если введен 0: вывод информации
    if (input[0] == '0')
    {
        win_write(" program is statically linked\n");
        win_write("  - cos1.dll (the 1st realization)\n");
        win_write("  - sort1.dll (bubble sort)\n");
        win_write("you can't switch realizations\n");
    }

    // Команда 1: производная
    else if (input[0] == '1')
    {
        float a = 0, dx = 0;
        int count = sscanf(input + 1, "%f %f", &a, &dx);

        if (count == 2) //должно быть введено 2 числа
        {
            float result = cosdx(a, dx);
            char msg[128];
            snprintf(msg, sizeof(msg), "cos'(%f) with dx=%f = ", a, dx);//запись в массив
            win_write(msg);//вывод в массив
            win_write_float(result);//вывод результата
        }
        else
        {
            //значит было введено не 2 числа
            win_write("you should print 2 numbers\n");
        }
    }
    // введено 2: сортировка
    else if (input[0] == '2')
    {
        char* token = strtok(input, " ");
        token = strtok(NULL, " "); // Пропускаем "2" - на вход подаются 2 и размер массива

        if (!token)
        {
            win_write("error: print the array size\n");
            continue;
        }

        int n = atoi(token);//преобразовываем элемент массива в число
        if (n <= 0 || n > 100)
        {
            win_write("error: please print array size between 1-100\n");
            continue;
        }
        //выделяем память под динамический массив
        int* array = (int*)malloc(n * sizeof(int));
        if (!array)
        {
            win_write("memory error\n");
            continue;
        }
```

```
            // считывание элементов массива
            int valid = 1;
            for (int i = 0; i < n; i++)
            {
                token = strtok(NULL, " ");//так получаем указатель на следующее число
                if (!token) //ввели слишком мало элементов массива
                {
                    win_write("not enough array elements\n");
                    valid = 0;
                    break;
                }
                array[i] = atoi(token);
            }

            if (valid)
            {
                win_write("printed array: ");
                win_write_array(array, n);

                int* sorted = sort(array, n);
                if (sorted)
                {
                    win_write("sorted array:   ");
                    win_write_array(sorted, n);
                    free(sorted);
                }
                else
                {
                    win_write("error in sorting\n");
                }
            }

            free(array);
        }
        else {
            win_write("print 0, 1, or 2.\n");
        }
    }

    return 0;

}
```

program2.c

```
//динамическая загрузка: библиотеки загружаются прямо во время выполнения и можно менять реализацию

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

// типы указатели на функции, т.к. компилятор не знает о них заранее
typedef float (*cosdx_func)(float, float);
typedef int* (*sort_func)(int*, size_t);

// глобальные переменные для загружаемых библиотек
HMODULE cos_lib = NULL;// дескриптор библиотеки производной
HMODULE sort_lib = NULL;//дескриптор библиотеки сортировки
cosdx_func cosdx_ptr = NULL;// указатель на функцию cosdx
sort_func sort_ptr = NULL;// указатель на функцию sort

int current_version = 1;// данная реализация библиотеки (линковка или днамическая)

// функции ввода/вывода
void win_write(const char* str)
{
```

```
    HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);//дескриптор вывода
    DWORD bytesWritten;//количество записанных байт
    WriteFile(hStdout, str, (DWORD)strlen(str), &bytesWritten, NULL);//вывод
}

void win_write_int(int value)//вывод числа
{
    char buffer[32];
    snprintf(buffer, sizeof(buffer), "%d", value);
    win_write(buffer);
}

void win_write_float(float value) //вывод float для косинуса
{
    char buffer[64];
    snprintf(buffer, sizeof(buffer), "%.6f", value);
    win_write(buffer);
}

void win_write_array(int* arr, size_t n) //вывод массива для сортировки
{
    char buffer[64];
    for (size_t i = 0; i < n; i++)
    {
        snprintf(buffer, sizeof(buffer), "%d ", arr[i]);
        win_write(buffer);
    }
}

int win_read_line(char* buffer, size_t size) //считывание
{
    HANDLE hStdin = GetStdHandle(STD_INPUT_HANDLE);
    DWORD bytesRead;

    if (!ReadFile(hStdin, buffer, (DWORD)size - 1, &bytesRead, NULL))
    {
        return 0;
    }

    buffer[bytesRead] = '\0';

    // убираем /n /r
    char* cr = strchr(buffer, '\r');
    if (cr) *cr = '\0';
    char* lf = strchr(buffer, '\n');
    if (lf) *lf = '\0';

    return 1;
}

// загружаем библиотеку
void load_libraries(int version)
{
    char cos_lib_name[64];
    char sort_lib_name[64];

    if (version == 1) //если введено 1 то первая реализация
    {
        strcpy(cos_lib_name, "cos1.dll");
        strcpy(sort_lib_name, "sort1.dll");
    }
    else
    {
        strcpy(cos_lib_name, "cos2.dll");
        strcpy(sort_lib_name, "sort2.dll");
    }

    // чистим старые библиотеки после смены реализации
    if (cos_lib)
```

```
        FreeLibrary(cos_lib);
    if (sort_lib)
        FreeLibrary(sort_lib);

    // Загружаем новые
    cos_lib = LoadLibraryA(cos_lib_name); //загружает DLL в память
    if (!cos_lib)
    {
        char error[128];
        snprintf(error, sizeof(error), "error dll %s\n", cos_lib_name);
        win_write(error);
        return;
    }

    sort_lib = LoadLibraryA(sort_lib_name);
    if (!sort_lib)
    {
        char error[128];
        snprintf(error, sizeof(error), "error dll %s\n", sort_lib_name);
        win_write(error);
        FreeLibrary(cos_lib);//выход из программы поэтому 1ую чистим тоже
        cos_lib = NULL;
        return;
    }

    // получаем указатели на функции через сисколлы
    cosdx_ptr = (cosdx_func)GetProcAddress(cos_lib, "cosdx");//находит адрес функции по её
имени
    sort_ptr = (sort_func)GetProcAddress(sort_lib, "sort");

    if (!cosdx_ptr || !sort_ptr)
    {
        win_write("cant find functions in libraries\n");
        FreeLibrary(cos_lib);
        FreeLibrary(sort_lib);
        cos_lib = NULL;
        sort_lib = NULL;
        return;
    }

    char msg[128];
    snprintf(msg, sizeof(msg), "got to version %d\n", version);
    win_write(msg);
}

int main()
{
    char input[256];

    win_write(" program 2: dynamic\n");
    win_write("commands:\n");
    win_write("  0 - choose version (1 or 2)\n");
    win_write("  1 a dx - derivative of cos(x)\n");
    win_write("  2 n a1 a2 ... an - sort array\n");
    win_write("  exit - exit program\n");

    // загрузка библиотек
    load_libraries(current_version);

    while (1)
    {
        win_write("\n> ");

        if (!win_read_line(input, sizeof(input)))
        {
            break;
        }

        // если введен выход
```

```
if (strcmp(input, "exit") == 0)
{
    win_write("exiting\n");
    break;
}

if (strlen(input) == 0)
{
    continue;
}

// если введен 0 переключаем версию
if (input[0] == '0')
{
    current_version = (current_version == 1) ? 2 : 1;
    load_libraries(current_version);
}
// введена 1 - ищем производную
else if (input[0] == '1')
{
    if (!cosdx_ptr)
    {
        win_write("error loading\n");
        continue;
    }

    float a = 0, dx = 0;
    int count = sscanf(input + 1, "%f %f", &a, &dx);

    if (count == 2)
    {
        float result = cosdx_ptr(a, dx);//вызов функции через указатель
        char msg[128];
        snprintf(msg, sizeof(msg), "cos'(%f) with dx=%f = ", a, dx);
        win_write(msg);
        win_write_float(result);
        win_write("\n");

        // показываем какая версия использовалась
        snprintf(msg, sizeof(msg), " (using version %d: ", current_version);
        win_write(msg);
        win_write(current_version == 1 ? "realization 1)\n" : "realization 2)\n");
    }
    else
    {
        win_write("you should print 2 numbers\n");
    }
}
// ведено 2 это сортировка
else if (input[0] == '2')
{
    if (!sort_ptr)
    {
        win_write("loading error\n");
        continue;
    }

    char* token = strtok(input, " ");
    token = strtok(NULL, " "); // пропускаем 2

    if (!token)
    {
        win_write("print the array size\n");
        continue;
    }

    int n = atoi(token);//преобразуем элемент массива в число
    if (n <= 0 || n > 100)
    {
```

```
                win_write("please print the array size 1-100\n");
                continue;
            }

            int* array = (int*)malloc(n * sizeof(int));
            if (!array)
            {
                win_write("memory error\n");
                continue;
            }

            // считываниеи элементов массива
            int valid = 1;
            for (int i = 0; i < n; i++)
            {
                token = strtok(NULL, " ");
                if (!token)//значит введено мало чисел
                {
                    win_write("not enough array elements\n");
                    valid = 0;
                    break;
                }
                array[i] = atoi(token);
            }

            if (valid)
            {
                win_write("printed array: ");
                win_write_array(array, n);
                win_write("\n");

                int* sorted = sort_ptr(array, n);//указатель на функцию сортировки
                if (sorted)
                {
                    win_write("sorted array:   ");
                    win_write_array(sorted, n);

                    char msg[64];
                    snprintf(msg, sizeof(msg), " (using version %d: ", current_version);
                    win_write(msg);
                    win_write(current_version == 1 ? "bubble sort)\n" : "quick sort)\n");

                    free(sorted);
                }
                else
                {
                    win_write("sorting error\n");
                }
            }

            free(array);
        }
        else
        {
            win_write("print 0, 1, or 2.\n");
        }
    }

    // освобождение библиотек перед выходом
    if (cos_lib) FreeLibrary(cos_lib);
    if (sort_lib) FreeLibrary(sort_lib);

    return 0;

}
```

contract.h

```
//здесь то, что должны делать библиотеки!!!
```

```c
#ifndef CONTRACT_H
#define CONTRACT_H

#include <stddef.h>

// Производная cos(x)
float cosdx(float a, float dx);

// Сортировка массива
int* sort(int* array, size_t n);

#endif
```

Протокол работы программы

Сборка:

```
C:\Users\Алиска\Documents\mai\osi\4>gcc -shared cos1.c -o cos1.dll -lm

C:\Users\Алиска\Documents\mai\osi\4>gcc -shared cos2.c -o cos2.dll -lm

C:\Users\Алиска\Documents\mai\osi\4>gcc -shared sort1.c -o sort1.dll

C:\Users\Алиска\Documents\mai\osi\4>gcc -shared sort2.c -o sort2.dll

C:\Users\Алиска\Documents\mai\osi\4>gcc program1.c cos1.c sort1.c -o program1.exe -lm -luser32

C:\Users\Алиска\Documents\mai\osi\4>gcc program2.c -o program2.exe -luser32

C:\Users\Алиска\Documents\mai\osi\4>program1.exe
```

Работа первой программы (со статическими библиотеками), в которую линкуем первые реализации функций:

```
C:\Users\Алиска\Documents\mai\osi\4>program1.exe
 program 1: static linking
 commands:
  0 - show library info
  1 a dx - derivative of cos(x) at point 'a' with step 'dx'
  2 n a1 a2 ... an - sort array of n numbers
  exit - exit program

> 2
error: print the array size

> 2 5 7 9 45 2 -5
printed array: 7 9 45 2 -5
sorted array:   -5 2 7 9 45

> 1 0.5 0.6
cos'(0.500000) with dx=0.600000 = -0.706644

> exit
exiting
```

Работа первой программы (со статическими библиотеками), в которую линкуем вторые реализации функций:

```
C:\Users\Алиска\Documents\mai\osi\4>gcc program1.c cos2.c sort2.c -o program1_v2.exe

C:\Users\Алиска\Documents\mai\osi\4>program1.exe
 program 1: static linking
 commands:
  0 - show library info
  1 a dx - derivative of cos(x) at point 'a' with step 'dx'
  2 n a1 a2 ... an - sort array of n numbers
  exit - exit program

> 1 54 0.4
cos'(54.000000) with dx=0.400000 = 0.707672

> 2 4 -8 -9 7 78
printed array: -8 -9 7 78
sorted array:   -9 -8 7 78
```

Работа второй программы (с динамическими библиотеками):

```
C:\Users\Алиска\Documents\mai\osi\4>program2.exe
 program 2: dynamic
commands:
  0 - choose version (1 or 2)
  1 a dx - derivative of cos(x)
  2 n a1 a2 ... an - sort array
  exit - exit program
got to version 1

> 1 56 0.4
cos'(56.000000) with dx=0.400000 = 0.339373
 (using version 1: realization 1)

> 2 4 -8 -9 7 78
printed array: -8 -9 7 78
sorted array:   -9 -8 7 78  (using version 1: bubble sort)

> 0
got to version 2

> 1 56 0.4
cos'(56.000000) with dx=0.400000 = 0.507756
 (using version 2: realization 2)

> 2 4 -8 -9 7 78
printed array: -8 -9 7 78
sorted array:   -9 -8 7 78  (using version 2: quick sort)

> exit
exiting
```

Вывод системных вызовов:

program2.exe	WriteFile (0x0000000000000054, 0x00007ff737e6...	TRUE	
program2.exe	GetStdHandle (STD_OUTPUT_HANDLE)	0x000000...	
program2.exe	WriteFile (0x0000000000000054, 0x00007ff737e6...	TRUE	
program2.exe	LoadLibraryA ("cos1.dll")	0x00007ff...	
ntdll.dll	└DllMain (0x00007ff89d3b0000, DLL_PROCESS...	TRUE	
program2.exe	LoadLibraryA ("sort1.dll")	0x00007ff...	
ntdll.dll	└DllMain (0x00007ff899ae0000, DLL_PROCESS_...	TRUE	
program2.exe	GetProcAddress (0x00007ff89d3b0000, "cosdx")	0x00007ff...	
program2.exe	GetProcAddress (0x00007ff899ae0000, "sort")	0x00007ff...	
program2.exe	GetStdHandle (STD_OUTPUT_HANDLE)	0x000000...	
program2.exe	WriteFile (0x0000000000000054, 0x000000d749df...	TRUE	
program2.exe	GetStdHandle (STD_OUTPUT_HANDLE)	0x000000...	
program2.exe	WriteFile (0x0000000000000054, 0x00007ff737e6...	TRUE	
program2.exe	GetStdHandle (STD_INPUT_HANDLE)	0x000000...	
program2.exe	ReadFile (0x0000000000000050, 0x000000d749dff...	TRUE	

12:09:50.6...	1	program2.exe	FreeLibrary (0x00007ff89d3b0000)	TRUE	
12:09:50.6...	1	program2.exe	FreeLibrary (0x00007ff899ae0000)	TRUE	

Вывод

В ходе лабораторной работы были реализованы и проанализированы два подхода к использованию библиотек: статическая линковка на этапе компиляции и динамическая загрузка во время выполнения программы.

Работа успешно продемонстрировала различия между статическим и динамическим связыванием. Оба подхода имеют свои области применения: статическая линковка предпочтительна для максимальной производительности, так как время не тратится на вызовы через указатели, а динамическая загрузка обеспечивает быстрое переключение между библиотеками без необходимости перекомпилирования. Выбор конкретного метода зависит от требований проекта: если критична скорость выполнения - выбираем статическую линковку; если важны возможность обновлений и расширяемость - динамическую загрузку.

Разработанные программы корректно управляют ресурсами (памятью, дескрипторами библиотек), что исключает утечки и обеспечивает стабильную работу.