

Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и
прикладной математики
Кафедра вычислительной математики и
программирования

Лабораторная работа №5 по курсу
"Операционные системы"

ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ

Студент: Сеимов Максим Сергеевич

Группа: М8О-208Б-21

Вариант: 8

Преподаватель: Соколов Андрей Алексеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

1 Постановка задачи

Цель работы

Целью лабораторной работы №5 является приобретение практических навыков в:

- Создании динамических библиотек
- Создании 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 использует одну из библиотек, используя знания, полученные на этапе компиляции;
- Тестовая программа (*программа №2*), которая загружает библиотеки, используя только их местоположение и контракты.

Пользовательский ввод для обеих программ должен быть организован следующим образом:

1. Если пользователь вводит команду «0», то программа переключает одну реализацию контрактов на другую (необходимо только для программы №2);
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.

Вариант 8

В варианте №8 нужно реализовать две функции:

1. Расчет интеграла функции $\sin x$ на отрезке $[A, B]$ с шагом e . Реализации:
 - (a) Подсчет интеграла методом прямоугольников.
 - (b) Подсчет интеграла методом трапеций.
2. Отсортировать целочисленный массив. Реализации:
 - (a) Пузырьковая сортировка.
 - (b) Быстрая сортировка Хоара.

Общие сведения о программе

Программа представлена следующими исполняемыми файлами (компилируемыми в папке `src/bin` с помощью утилиты **make**):

- **lab5_1** - доступна только первая реализация обеих функций. Библиотеки подключены на этапе линковки.
- **lab5_2** - доступна только вторая реализация обеих функций. Библиотеки подключены на этапе линковки.
- **lab5** - доступны обе реализации обеих функций. Библиотеки подключаются и переключаются во время исполнения программы.

Исходные файлы:

- **src/headers:**
 - **interface.h** - заголовочный файл для реализации интерфейса программы
- **src/lib:**
 - **src/lib/src_files:**
 - * **1st_implementation.c** - первая реализация обеих функций
 - * **2nd_implementation.c** - вторая реализация обеих функций
 - **functions.h** - заголовочный файл с именами и сигнатурами функций
- **interface.c** - реализация пользовательского интерфейса
- **main.c** - реализация первой версии программы (реализации подключаются на этапе линковки)
- **main_2.c** - вторая версия программы (реализации меняются в процессе выполнения с помощью системных вызовов `dlopen`, `dlsym` и `dlclose`).

Основные файлы программы

functions.h:

```
1 | #ifndef __FUNC_H__
2 | #define __FUNC_H__
3 |
4 | #include <math.h>
5 | #include <stdio.h>
6 | #include <stdlib.h>
7 |
8 | double sin_integral(double, double, double);
9 | int *sort(int*, int);
10 |
11 | #endif
```

1st_implementation.c:

```
1 #include "../functions.h"
2
3 double sin_integral(double a, double b, double e) {
4     if (a >= b) {
5         printf("Invalid segment borders\n");
6         return 0;
7     }
8     if (e > (b - a)) {
9         printf("Step is bigger than segment size\n");
10        return 0;
11    }
12    double i = 0;
13    double result = 0;
14    while (a + i <= b) {
15        result += sin(a + i) * e;
16        i += e;
17    }
18    printf("\nCalculated via rectangle method: ");
19    return result;
20 }
21
22 int *sort(int *b, int n) {
23     int *a = (int *)calloc(sizeof(int), n);
24     for (int i = 0; i < n; ++i) {
25         a[i] = b[i];
26     }
27     for (int i = 0; i < n - 1; ++i) {
28         for (int j = 0; j < n - i - 1; ++j) {
29             if (a[j + 1] < a[j]) {
30                 int k = a[j];
31                 a[j] = a[j + 1];
32                 a[j + 1] = k;
33             }
34         }
35     }
36     printf("\nSorted by bubble sort algorithm: \n");
37     return a;
38 }
```

2nd_implementation.c:

```
1 #include "../functions.h"
2
3 double sin_integral(double a, double b, double e) {
4     if (a >= b) {
5         printf("Invalid segment borders\n");
6         return 0;
7     }
8     if (e > (b - a)) {
9         printf("Step is bigger than segment size\n");
10        return 0;
11    }
12    double i = 0;
13    double result = 0;
14    while (a + i < b) {
15        result += (sin(a + i) + sin(a + i + e)) * e / 2;
16        i += e;
17    }
18    printf("\nCalculated via trapezoid method: ");
19    return result;
```

```

20 }
21
22 void quicksort(int* a, int first, int last) {
23     int pivot = a[(first + last) / 2];
24     int i = first, j = last;
25     do {
26         while(a[i] < pivot) {
27             i++;
28         }
29         while(a[j] > pivot) {
30             j--;
31         }
32         if (i <= j) {
33             if (i < j) {
34                 int temp = a[i];
35                 a[i] = a[j];
36                 a[j] = temp;
37             }
38             i++;
39             j--;
40         }
41     } while(i <= j);
42     if (i < last) {
43         quicksort(a, i, last);
44     }
45     if (j > first) {
46         quicksort(a, first, j);
47     }
48 }
49
50 int *sort(int *b, int n) {
51     int *a = calloc(sizeof(int), n);
52     for (int i = 0; i < n; ++i) {
53         a[i] = b[i];
54     }
55     quicksort(a, 0, n - 1);
56     printf("\nSorted by quicksort algorithm\n");
57     return a;
58 }

```

interface.h:

```

1  #ifndef __INTERFACE_H__
2  #define __INTERFACE_H__
3
4  #include <stdio.h>
5  #include <string.h>
6  #include <stdlib.h>
7
8  #define STR_LEN 16
9  #define ARR_SIZE_INIT 32
10
11 typedef enum {CHANGE_IMP, EXEC_1, EXEC_2, EXIT, UNKNOWN} command;
12
13 command get_command();
14 void execute_command(command com);
15
16 #endif

```

interface.c:

```
1 #include "../headers/interface.h"
2 #include "../lib/functions.h"
3 #include <unistd.h>
4
5 int read_number(int *result) {
6     char *buf = calloc(sizeof(char), STR_LEN);
7     char c = 0;
8     short i = 0;
9
10    if (read(0, &c, 1) < 0) {
11        exit(1);
12    }
13    if ((c == ' ') || (c == '\n') || (c == '\0'))
14        return -1;
15    while (c != ' ' && c != '\n') {
16        buf[i++] = c;
17        read(0, &c, 1);
18    }
19    *result = atoi(buf);
20    free(buf);
21    if (c == '\n') {
22        return 0;
23    }
24    return 1;
25 }
26
27 command get_command() {
28     int command_number;
29     read_number(&command_number);
30     switch (command_number) {
31         case 0:
32             return CHANGE_IMP;
33         case 1:
34             return EXEC_1;
35         case 2:
36             return EXEC_2;
37         case -1:
38             return EXIT;
39         default:
40             return UNKNOWN;
41     }
42 }
43
44 void execute_command(command com) {
45     switch (com) {
46         case EXEC_1:
47             double a, b, e;
48             scanf("%lf %lf %lf", &a, &b, &e);
49             double result = sin_integral(a, b, e);
50             printf("%lf\n\n", result);
51             break;
52         case EXEC_2:
53             int curr, status, i = 0, arr_size = ARR_SIZE_INIT;
54             int *arr = malloc(arr_size * sizeof(int));
55             do {
56                 status = read_number(&curr);
57                 if (status == -1) {
58                     break;
59                 }
```

```

60         arr[i++] = curr;
61         if (i == arr_size) {
62             arr_size *= 2;
63             arr = realloc(arr, arr_size * sizeof(int));
64         }
65     } while (status != 0);
66     arr = realloc(arr, i * sizeof(int));
67     int *sorted = sort(arr, i);
68     free(arr);
69     for (int j = 0; j < i; ++j) {
70         printf("%d ", sorted[j]);
71     }
72     printf("\n\n");
73     break;
74     case UNKNOWN:
75         printf("Invalid command!\n");
76         break;
77     case EXIT:
78         break;
79 }
80 }

```

main.c:

```

1  #include "../headers/interface.h"
2
3  int main() {
4      command input;
5      do {
6          input = get_command();
7          execute_command(input);
8      } while (input != EXIT);
9      return 0;
10 }

```

main_2.c:

```

1  #include <unistd.h>
2  #include <dlfcn.h>
3
4  #include "../headers/interface.h"
5
6  int read_number(int *result) {
7      char *buf = calloc(sizeof(char), STR_LEN);
8      char c = 0;
9      short i = 0;
10
11     if (read(0, &c, 1) < 0) {
12         exit(1);
13     }
14     if ((c == ' ') || (c == '\n') || (c == '\0'))
15         return -1;
16     while (c != ' ' && c != '\n') {
17         buf[i++] = c;
18         read(0, &c, 1);
19     }
20     *result = atoi(buf);
21     free(buf);
22     if (c == '\n') {
23         return 0;
24     }
25     return 1;

```

```

26 }
27
28 command get_command() {
29     int command_number;
30     read_number(&command_number);
31     switch (command_number) {
32         case 0:
33             return CHANGE_IMP;
34         case 1:
35             return EXEC_1;
36         case 2:
37             return EXEC_2;
38         case -1:
39             return EXIT;
40         default:
41             return UNKNOWN;
42     }
43 }
44
45 int main() {
46     command input;
47     int implementation = 1;
48     void *handle = dlopen("/home/iddqd/Projects/C/OS/os_lab_5/src/lib/libFIRST.so",
49                          RTLD_LAZY);
49     double(*sin_integral)(double, double, double) = dlsym(handle, "sin_integral");
50     int(*sort)(int*, int) = dlsym(handle, "sort");
51
52     printf("\nYou are using implementation number %d\n\n", implementation);
53
54     do {
55         input = get_command();
56         if (input == CHANGE_IMP) {
57             if (implementation == 1) {
58                 dlclose(handle);
59                 implementation = 2;
60                 handle = dlopen("/home/iddqd/Projects/C/OS/os_lab_5/src/lib/libSECOND.so",
61                              RTLD_LAZY);
62                 sin_integral = dlsym(handle, "sin_integral");
63                 sort = dlsym(handle, "sort");
64             }
65             else {
66                 dlclose(handle);
67                 implementation = 1;
68                 handle = dlopen("/home/iddqd/Projects/C/OS/os_lab_5/src/lib/libFIRST.so",
69                              RTLD_LAZY);
70                 sin_integral = dlsym(handle, "sin_integral");
71                 sort = dlsym(handle, "sort");
72             }
73             printf("\nYou are using implementation number %d\n\n", implementation);
74         }
75         switch (input) {
76             case EXEC_1:
77                 double a, b, e;
78                 scanf("%lf %lf %lf", &a, &b, &e);
79                 double result = (*sin_integral)(a, b, e);
80                 printf("%lf\n\n", result);
81                 break;
82             case EXEC_2:
83                 int curr, status, i = 0, arr_size = ARR_SIZE_INIT;
84                 int *arr = malloc(arr_size * sizeof(int));

```



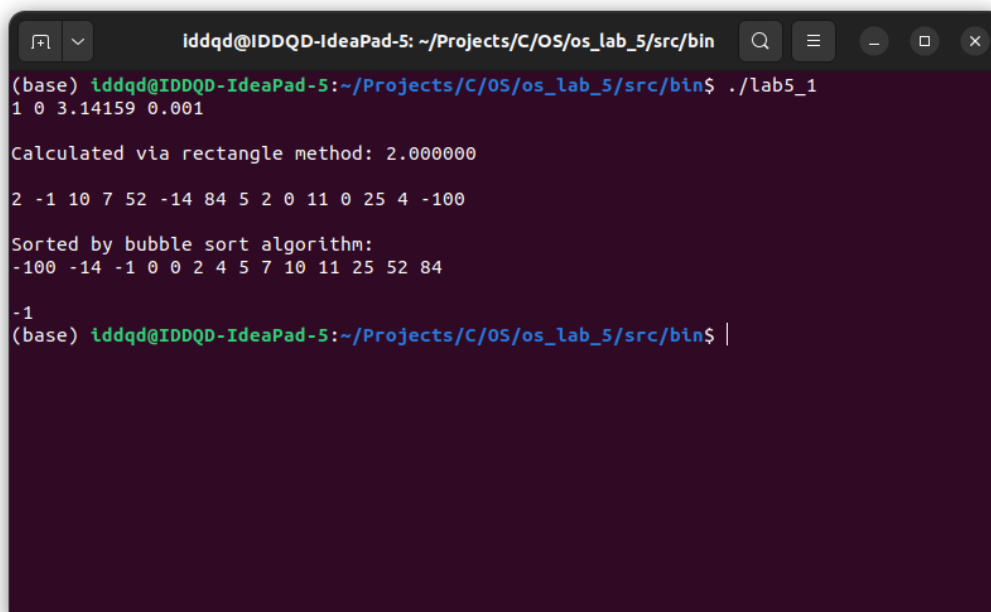
```

83         do {
84             status = read_number(&curr);
85             if (status == -1) {
86                 break;
87             }
88             arr[i++] = curr;
89             if (i == arr_size) {
90                 arr_size *= 2;
91                 arr = realloc(arr, arr_size * sizeof(int));
92             }
93         } while (status != 0);
94         arr = realloc(arr, i * sizeof(int));
95         int *sorted = (*sort)(arr, i);
96         free(arr);
97         for (int j = 0; j < i; ++j) {
98             printf("%d ", sorted[j]);
99         }
100        printf("\n\n");
101        break;
102    case UNKNOWN:
103        printf("Invalid command!\n");
104        break;
105    case EXIT:
106        break;
107    }
108    } while (input != EXIT);
109    dlclose(handle);
110    return 0;
111 }

```

Пример работы

Запуск программ на нескольких тестах:



```

(base) iddq@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_5/src/bin$ ./lab5_1
1 0 3.14159 0.001

Calculated via rectangle method: 2.000000

2 -1 10 7 52 -14 84 5 2 0 11 0 25 4 -100

Sorted by bubble sort algorithm:
-100 -14 -1 0 0 2 4 5 7 10 11 25 52 84

-1
(base) iddq@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_5/src/bin$

```

Рис. 1: Запуск программы lab5_1

```
iddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_5/src/bin
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_5/src/bin$ ./lab5_2
1 0 6.28 0.01

Calculated via trapezoid method: 0.000023

2 13 7 65 95 0 -1 65 74 29 4 6 0 -21

Sorted by quicksort algorithm
-21 -1 0 0 4 6 7 13 29 65 65 74 95

-1
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_5/src/bin$ |
```

Рис. 2: Запуск программы lab5_2

```
iddqd@IDDQD-IdeaPad-5: ~/Projects/C/OS/os_lab_5/src/bin
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_5/src/bin$ ./lab5
You are using implementation number 1
1 -3.14 3.14 0.01

Calculated via rectangle method: 0.000000

2 165 28 -14 3 9 -13 0 984 0 54 -32

Sorted by bubble sort algorithm:
-32 -14 -13 0 0 3 9 28 54 165 984

0

You are using implementation number 2
1 -3.14159 0 0.01

Calculated via trapezoid method: -1.999948

2 165 28 -14 3 9 -13 0 984 0 54 -32

Sorted by quicksort algorithm
-32 -14 -13 0 0 3 9 28 54 165 984

0

You are using implementation number 1
1 -3.14159 0 0.01

Calculated via rectangle method: -1.999990

-1
(base) iddq@IDDQD-IdeaPad-5:~/Projects/C/OS/os_lab_5/src/bin$ |
```

Рис. 3: Запуск программы lab5

Вывод

В ходе выполнения задания были изучены принципы работы динамических библиотек. Я научился компилировать программы, подключая динамические библиотеки во время линковки, а также использовать системные вызовы `dlopen`, `dlsym` и `dlclose`.