

**Московский авиационный институт (национальный
исследовательский университет)**

Институт информационных технологий и прикладной математики
«Кафедра вычислительной математики и программирования»

**Лабораторная работа по предмету
"Операционные системы"
№4**

Студент: Филиппов А. М.

Преподаватель: Миронов Е.С.

Группа: М8О-201Б-24

Дата:

Оценка:

Подпись:

Москва, 2025

Оглавление

- 1. Цель работы**
- 2. Постановка задачи**
- 3. Общие сведения о программе**
- 4. Общий алгоритм решения**
- 5. Реализация**
- 6. Пример работы**
- 7. Вывод**

Цель работы

Целью является приобретение практических навыков в:

- Создание динамических библиотек
- Создание программ, которые используют функции динамических библиотек

Постановка задачи

Требуется создать динамические библиотеки, которые реализуют определенный функционал.

Далее использовать данные библиотеки 2-мя способами:

Во время компиляции (на этапе «линковки»/linking)

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 обеих 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.

5	Расчет значения числа Пи при заданной длине ряда (K)	float Pi(int K)	Ряд Лейбница	Формула Валлиса
9	Отсортировать целочисленный массив	Int * Sort(int * array)	Пузырьковая сортировка	Сортировка Хоара

Общие сведения о программе

Создадим две динамические библиотеки, которые реализуют контракты "Расчет значения числа Пи при заданной длине ряда (K)" и "Сортировка целочисленного массива".

Теперь создадим тестовую программу, которая будет использовать библиотеки на этапе компиляции.

Теперь создадим вторую тестовую программу, которая будет загружать библиотеки во время исполнения.

Общий алгоритм решения

Анализ двух типов использования библиотек

1. Во время компиляции (на этапе «линковки»/linking):

Программа №1 использует библиотеки на этапе компиляции.

Заголовочные файлы библиотек включаются в программу и функции вызываются напрямую. Это обеспечивает более простую и производительную работу программы, так как вызовы функций происходят непосредственно, без необходимости загрузки библиотек во время исполнения.

2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками: Программа №2 загружает библиотеки во время исполнения с помощью функций из библиотеки `dlfcn.h`, которые позволяют работать с динамическими библиотеками. Это предоставляет большую гибкость, так как библиотеки могут быть загружены или выгружены во время работы программы. Однако использование такого подхода более сложно, так как требуется явная загрузка библиотеки и определение типов функций с помощью указателей на функции.

В обоих случаях возможно переключить реализацию контрактов при помощи команды "0" в пользовательском вводе в программе №2. Это демонстрирует гибкость и возможность динамического изменения реализаций контрактов без необходимости перекомпиляции или перезапуска программы.

Реализация

lib.h

```
#ifndef MATH_LIB_H
#define MATH_LIB_H

float Pi(int K);

int *Sort(int *array);

#endif
```

lib1.c

```
#include "lib.h"
#include <math.h>
```

```
#include <stdlib.h>
```

```
// Implementation 1: Leibniz series for Pi
```

```
float Pi(int K) {
```

```
if (K <= 0)
```

```
return 0.0f;
```

```
float pi = 0.0f;
```

```
for (int n = 0; n < K; n++) {
```

```
float term = pow(-1, n) / (2 * n + 1);
```

```
pi += term;
```

```
}
```

```
return 4.0f * pi;
```

```
}
```

```
// Implementation 1: Bubble sort
```

```
int *Sort(int *array) {
```

```
if (array == NULL)
```

```
return NULL;
```

```
int size = array[0];
```

```
if (size <= 1)
```

```
return array;
```

```
int *data = array + 1;
```

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

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

```
if (data[j] > data[j + 1]) {
```

```
int temp = data[j];
```

```
data[j] = data[j + 1];
```

```
data[j + 1] = temp;
```

```
}
```

```
}
```

```
}
```

```
return array;
```

```
}
```

lib2.c

```
#include "lib.h"
```

```
#include <stdlib.h>
```

```
// Implementation 2: Wallis formula for Pi
```

```
float Pi(int K) {
```

```
if (K <= 0)
```

```
return 0.0f;
```

```

float pi = 1.0f;
for (int n = 1; n <= K; n++) {
float numerator = 2.0f * n;
float denominator = 2.0f * n - 1.0f;
pi *= (numerator / denominator) * (numerator / (denominator + 2.0f));
}
return 2.0f * pi;
}

```

```

static int partition(int *arr, int low, int high) {
int pivot = arr[low];
int i = low - 1;
int j = high + 1;

```

```

while (1) {
do {
i++;
} while (arr[i] < pivot);

```

```

do {
j--;
} while (arr[j] > pivot);

```

```

if (i >= j) {
return j;
}

```

```

int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
}

```

```

static void quickSort(int *arr, int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi);
quickSort(arr, pi + 1, high);
}
}

```

```

// Implementation 2: Hoare's sort (Quick Sort)
int *Sort(int *array) {
if (array == NULL)
return NULL;

```

```

int size = array[0];

```

```

if (size <= 1)
return array;

int *data = array + 1;

quickSort(data, 0, size - 1);

return array;
}

```

program1.c

```

#include "lib.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
int command;

printf(
"Commands:\n"
" 1 - Pi func\t\tusage: 1 <num_digits>\n"
" 2 - array sort func\tusage: 2 <N = size> <elem1> <elem2> ... <elemN>\n"
"-1 - exit\t\tusage: -1\n");

while (1) {
printf("Enter command: ");
scanf("%d", &command);

if (command == 1) {
int K;
scanf("%d", &K);
float result = Pi(K);
printf("Pi: %f\n", result);
} else if (command == 2) {
int size;
scanf("%d", &size);

if (size <= 0) {
printf("Invalid array size\n");
return 1;
}

int *array = (int *)malloc((size + 1) * sizeof(int));
if (array == NULL) {
printf("Memory allocation failed\n");
return 1;
}

```

```

}

array[0] = size;
for (int i = 1; i <= size; i++) {
    scanf("%d", &array[i]);
}

int *result = Sort(array);

printf("Result: ");
for (int i = 1; i <= size; i++) {
    printf("%d ", result[i]);
}
printf("\n");

free(array);
} else if (command == -1) {
    break;
} else {
    printf("Invalid command\n");
}
}

return 0;
}

```

program2.c

```

#include <dlfcn.h>
#include <stdio.h>
#include <stdlib.h>

typedef float (*PiFunction)(int);
typedef int *(*SortFunction)(int *);

int main() {
    void *lib_handle = NULL;
    PiFunction pi_func = NULL;
    SortFunction sort_func = NULL;

    int lib_number = 1;
    char lib_path[256];

    snprintf(lib_path, sizeof(lib_path), "./lib%d.so", lib_number);
    lib_handle = dlopen(lib_path, RTLD_LAZY);

    if (!lib_handle) {

```

```

fprintf(stderr, "Error opening library %s: %s\n", lib_path, dlerror());
return 1;
}

pi_func = (PiFunction)dlsym(lib_handle, "Pi");
sort_func = (SortFunction)dlsym(lib_handle, "Sort");

if (!pi_func || !sort_func) {
fprintf(stderr, "Error loading functions: %s\n", dlerror());
dlclose(lib_handle);
return 1;
}

int command;

printf(
"Commands:\n"
" 0 - switch libs\tusage: 0\n"
" 1 - Pi func\t\tusage: 1 <num_digits>\n"
" 2 - array sort func\tusage: 2 <N = size> <elem1> <elem2> ... <elemN>\n"
"-1 - exit\t\t\tusage: -1\n");

while (1) {
printf("Enter command: ");
scanf("%d", &command);

if (command == 0) {
dlclose(lib_handle);

lib_number = (lib_number == 1) ? 2 : 1;
snprintf(lib_path, sizeof(lib_path), "./lib%d.so", lib_number);

lib_handle = dlopen(lib_path, RTLD_LAZY);
if (!lib_handle) {
fprintf(stderr, "Error opening library %s: %s\n", lib_path, dlerror());
return 1;
}

pi_func = (PiFunction)dlsym(lib_handle, "Pi");
sort_func = (SortFunction)dlsym(lib_handle, "Sort");

if (!pi_func || !sort_func) {
fprintf(stderr, "Error loading functions: %s\n", dlerror());
dlclose(lib_handle);
return 1;
}

printf("Switched to lib%d.so\n", lib_number);

```

```

continue;
}

if (command == 1) {
int K;
scanf("%d", &K);
float result = pi_func(K);
printf("Pi: %f\n", result);
} else if (command == 2) {
int size;
scanf("%d", &size);

if (size <= 0) {
printf("Invalid array size\n");
continue;
}

int *array = (int *)malloc((size + 1) * sizeof(int));
if (array == NULL) {
printf("Memory allocation failed\n");
continue;
}

array[0] = size;
for (int i = 1; i <= size; i++) {
scanf("%d", &array[i]);
}

int *result = sort_func(array);

printf("Result: ");
for (int i = 1; i <= size; i++) {
printf("%d ", result[i]);
}
printf("\n");

free(array);
} else if (command == -1) {
break;
} else {
printf("Invalid command\n");
}
}

dlclose(lib_handle);
return 0;
}

```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(lab_04 C)
```

```
set(CMAKE_C_STANDARD 99)
set(CMAKE_C_STANDARD_REQUIRED ON)
```

```
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
set(CMAKE_POSITION_INDEPENDENT_CODE ON)
```

```
add_library(lib1 SHARED lib1.c)
set_target_properties(lib1 PROPERTIES OUTPUT_NAME "lib1" PREFIX "")
target_link_libraries(lib1 m) # m <= math.h
```

```
add_library(lib2 SHARED lib2.c)
set_target_properties(lib2 PROPERTIES OUTPUT_NAME "lib2" PREFIX "")
```

```
add_executable(program1 program1.c)
target_link_libraries(program1 lib1 m)
```

```
add_executable(program2 program2.c)
target_link_libraries(program2 dl) # dl <= dlfcn.h
```

Пример работы

./program1

Commands:

1 - Pi func usage: 1 <num_digits>
2 - array sort func usage: 2 <N = size> <elem1> <elem2> ... <elemN>
-1 - exit usage: -1

Enter command: 1 100

Pi: 3.131593

Enter command: 2 10 1 3 5 7 9 2 4 6 8 10

Result: 1 2 3 4 5 6 7 8 9 10

Enter command: -1

./program2

Commands:

0 - switch libs usage: 0
1 - Pi func usage: 1 <num_digits>
2 - array sort func usage: 2 <N = size> <elem1> <elem2> ... <elemN>
-1 - exit usage: -1

Enter command: 1 10

Pi: 3.041840

Enter command: 2 4 1 3 2 4

```
Result: 1 2 3 4
Enter command: 0
Switched to lib2.so
Enter command: 1 10
Pi: 3.067706
Enter command: 2 4 1 3 2 4
Result: 1 2 3 4
Enter command: 0
Switched to lib1.so
Enter command: -1
```

Вывод

Выполнив данную лабораторную работу, я приобрел практические навыки в создании динамических библиотек. Динамические библиотеки, хоть и замедляют загрузку программы, обладают существенными преимуществами перед статическими: нет необходимости копировать библиотеку для каждой отдельной программы, также в одном варианте возможно применять изменения библиотеки в уже запущенной программе. Существует два типа динамических библиотек. Первый из них предполагает линковку во время компиляции. При этом становится невозможно применять изменения, происходящие в библиотеке для запущенной программы. Эту проблему решает второй тип: библиотека, подгружаемая программой во время исполнения с помощью системных вызовов. Таким образом, использование динамических библиотек, подключаемых на этапе выполнения программы, является более гибким решением. Однако, при этом, нужно больше задумываться о безопасности.