Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

Студент: Хомяков Иван Андреевич

Группа: М8О-207Б-21

Вариант: 28

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/EbumbaE/OS\_LAB/lab5

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Создание динамических библиотек
* Создание 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 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 обоих 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). Можно реализовать лабораторную работу без данной функции, но максимальная оценка в этом случае будет «хорошо»;
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 28:

функции:

5) Рассчет значения числа Пи при заданной длине ряда (K)

7) Подсчет площади плоской геометрической фигуры по двум сторонам.

Сборочная система:

5) CMake. Типы чисел, используемых при вычислениях, в библиотеках меняются в зависимости от операционной системы, на которой происходит сборка.

Windows: int, float

Linux: int -> long, float -> double

MacOS: int -> short, float -> long double

**Общие сведения о программе**

Я сделал обычное задание с makefile, потом переделал под задание с cmake. Проект под make можно найти в репозитории в истории коммитов:).

В программе используются следующие системные вызовы:

1. LoadLibrary – загружает указанный модуль в адресное пространство вызывающего процесса, возвращает дескриптор dll модуля.
2. GetProcAddress – извлекает адрес экспортированной функции из указанной библиотеки динамической компоновки, возвращает адрес функции, передаем дескриптор dll и имя функции.
3. FreeLibrary – освобождает загруженный модуль библиотеки динамической компоновки.

**Общий метод и алгоритм решения**

В lib1 и lib2 создадим функции вычисления числа pi и площади с разной реализацией. Будем использовать свои типы данных, чтобы подстраивать библиотеку под операционную систему. Также создадим разные реализации main в зависимости от задания.

MakeFile:

Статическая реализация main: обычный импорт заголовочного файла с именами функций, также обычный вызов функций. Всю остальную работу делает makefile: в зависимости от выбранной версии библиотеки создаем объектный файл lib.o, из него статическую библиотеку lib.a, которую линкуем с объектным файлом static.o и получаем готовую программу static.exe. Объектный файл static.o мы создаем из main\_st.c.

Динамическая реализация main: в makefile из объектных файлов lib1.o и lib2.o создаем общие библиотеки lib1.so и lib2.so (shared object). Их мы и будем подгружать в main\_dy.c при помощи функции LoadLibrary, из этой функции мы получим библиотечный дескриптор libHandle. Для этого нам нужно знать путь до so файлов. Также мы завели ссылки на функции Pi и Square, их мы получим из функции GetProcAddress, в которую мы передадим имена функций и libHandle. Не забудем закрывать дескриптор при помощи команды FreeLibrary, чтобы не допускать утечек памяти. Когда пользователь вводит 0, мы просто получаем ссылки на другие реализации функций.

CMakeFile:

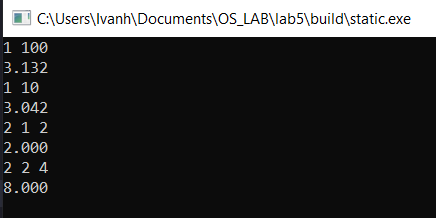
Чтобы переделать проект под дополнительное задание с cmake, в lib.h нужно создать свой тип при помощи typedef и использовать его в lib1.c и lib2.c. Информация об ос будет храниться в флаге OS, определяемом в cmake. В зависимости от этого флага по-разному будут определяться типы в lib.h. Добавляем библиотеки lib1 и lib2 как dll при помощи флага SHARED для main\_dy. Также сделаем статические библиотеки при помощи флага STATIC для main\_st. Для компиляции main\_st будем использовать target\_link\_libraries, чтобы связать его с библиотеками. Для main\_dy это делать не нужно, тк мы подгружаем их при помощи системных вызовов.

**Исходный код**

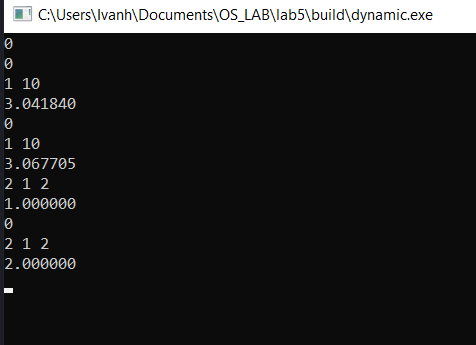
В репозитории.

**Демонстрация работы программы**

Статическое использование:



Динамическое использование:



**Выводы**

Научился работать с динамическими библиотеками, а также подгружать их во время работы программы. Полезный навык. Также было интересным и написать свой тип данных, который меняется в зависимости от операционной системы.