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-209Б-23

Студент: Магомедов Э.Х.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 26.12.24

Москва, 2024

**Постановка задачи**

**Вариант 8.**

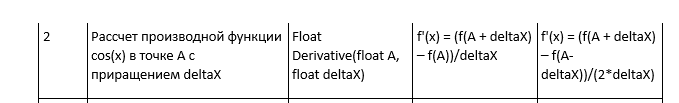
Требуется создать динамические библиотеки, которые реализуют заданный вариантом функционал. Далее использовать данные библиотеки 2-мя способами:

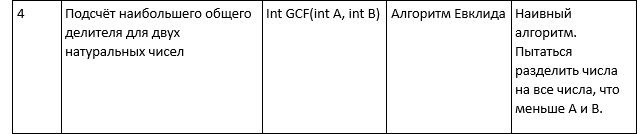
1. Во время компиляции (на этапе «линковки»/linking)
2. Во время исполнения программы. Библиотеки загружаются в память с помощью интерфейса ОС для работы с динамическими библиотеками

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 информацию полученные на этапе компиляции;
* Тестовая программа (*программа №2*), которая загружает библиотеки, используя только их относительные пути и контракты.

Контракты и реализации функций





**Общий метод и алгоритм решения**

Использованные системные вызовы:

● void\* dlopen(const char\* filename, int flag); – загружает динамическую библиотеку в память.  
● void\* dlsym(void\* handle, const char\* symbol); – возвращает указатель на функцию или переменную из загруженной библиотеки.  
● int dlclose(void\* handle); – освобождает ресурсы, связанные с загруженной библиотекой.

В рамках лабораторной работы были выполнены следующие основные этапы:

1. **Создание динамических библиотек:**
   * Реализованы две функции:
     + **SinIntegral:** Рассчитывает интеграл функции sin(x) на заданном отрезке [A, B] с шагом e методом прямоугольников и методом трапеций.
     + **Sort:** Сортирует целочисленный массив с использованием пузырьковой сортировки и быстрой сортировки (алгоритм Хоара).
   * Для каждой реализации функции была создана отдельная динамическая библиотека:
     + libSinIntegral\_Rect.so и libSinIntegral\_Trapezoid.so для функции SinIntegral.
     + libSort\_Bubble.so и libSort\_Quick.so для функции Sort.
2. **Разработка тестовых программ:**
   * **TestProgram1:** Использует динамические библиотеки на этапе компиляции (линковки).
   * **TestProgram2:** Загружает динамические библиотеки во время исполнения программы с помощью функций dlopen, dlsym и dlclose.
3. **Автоматизация сборки с помощью Makefile:**
   * Создан Makefile для упрощения процесса компиляции и сборки библиотек и тестовых программ.
   * Включены настройки RPATH для автоматического поиска динамических библиотек без необходимости ручной установки переменной окружения LD\_LIBRARY\_PATH.

**Makefile**

CXX = g++

CXXFLAGS = -std=c++11 -Wall -Wextra -fPIC -shared -O2

APPFLAGS = -std=c++11 -Wall -Wextra -O2

LDFLAGS = -ldl

# RPATH настройки

RPATH = -Wl,-rpath,\$$ORIGIN

# Библиотеки

LIBS = libSinIntegral\_Rect.so libSinIntegral\_Trapezoid.so libSort\_Bubble.so libSort\_Quick.so

all: $(LIBS) TestProgram1 TestProgram2

libSinIntegral\_Rect.so: SinIntegral\_Rect.cpp

    $(CXX) $(CXXFLAGS) -o $@ $<

libSinIntegral\_Trapezoid.so: SinIntegral\_Trapezoid.cpp

    $(CXX) $(CXXFLAGS) -o $@ $<

libSort\_Bubble.so: Sort\_Bubble.cpp

    $(CXX) $(CXXFLAGS) -o $@ $<

libSort\_Quick.so: Sort\_Quick.cpp

    $(CXX) $(CXXFLAGS) -o $@ Sort\_Quick.cpp

TestProgram1: TestProgram1.cpp

    $(CXX) $(APPFLAGS) -o $@ TestProgram1.cpp -L. -lSinIntegral\_Rect -lSort\_Bubble $(RPATH)

TestProgram2: TestProgram2.cpp

    $(CXX) $(APPFLAGS) -o $@ TestProgram2.cpp $(LDFLAGS) $(RPATH)

clean:

    rm -f $(LIBS) TestProgram1 TestProgram2

**Описание используемых флагов**

В Makefile используются следующие флаги компиляции и линковки:

* **Компилятор и стандарты:**
  + CXX = g++: Используется компилятор g++ для C++.
  + -std=c++11: Устанавливает стандарт C++11 для компиляции, обеспечивая доступ к современным возможностям языка.
* **Предупреждения и оптимизации:**
  + -Wall -Wextra: Включают все стандартные предупреждения и дополнительные предупреждения для выявления потенциальных ошибок в коде.
  + -O2: Включает уровень оптимизации O2 для повышения производительности создаваемых бинарных файлов.
* **Создание динамических библиотек:**
  + -fPIC: Генерирует позиционно-независимый код, необходимый для создания динамических библиотек.
  + -shared: Указывает компилятору создать динамическую библиотеку.
* **Настройка пути поиска библиотек:**
  + RPATH = -Wl,-rpath,\$$ORIGIN: Встраивает путь поиска библиотек непосредственно в исполняемый файл. $ORIGIN указывает на директорию, содержащую исполняемый файл, что позволяет динамическому загрузчику находить библиотеки без необходимости установки LD\_LIBRARY\_PATH.
* **Линковка с системными библиотеками:**
  + -ldl: Линкует программу с библиотекой dl, необходимой для динамической загрузки библиотек во время исполнения (используется в TestProgram2).

**Код программы**

**https://github.com/EATSLEEPSESH/os\_labs/tree/lab/os\_4**

**Вывод**

В ходе выполнения лабораторной работы были успешно созданы и использованы динамические библиотеки для реализации заданных функций. Разработанные тестовые программы демонстрируют два подхода к использованию динамических библиотек: статическую линковку во время компиляции и динамическую загрузку во время исполнения. Использование RPATH в Makefile позволило упростить процесс поиска библиотек, избавив от необходимости ручной настройки переменной окружения LD\_LIBRARY\_PATH. Анализ с помощью strace подтвердил корректность загрузки библиотек в обоих подходах. Работа показала преимущества динамической загрузки, такие как гибкость и возможность изменения функциональности без пересборки программ, а также подчеркнула важность правильной настройки путей поиска библиотек для обеспечения их корректной загрузки.