Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 4 по курсу «Операционные системы»

Динамические библиотеки

Студент: Арусланов К. А.
Преподаватель: Миронов Е. С.
Группа: М8О-203Б-23
Дата:
Оценка:
Подпись:

Москва, 2024

Условие

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.

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.

Задание

Отсортировать целочисленный массив:

1. Пузырьковая сортировка
2. Сортировка Хоара

Рассчет производной функции cos(x) в точке A с приращением deltaX:

1. f'(x) = (f(A + deltaX) – f(A))/deltaX

2. f'(x) = (f(A + deltaX) – f(A-deltaX))/(2*deltaX)

Метод решения

Были написаны две библиотеки (например, с разными реализациями сортировки или вычисления производной), причём программу №1 компоновали статически (при линковке), а программу №2 загружали библиотеку динамически (через dlopen/dlsym). В результате мы показали, как одна программа может на этапе компиляции жёстко связаться с нужным вариантом функций, а другая программа — выбирать реализацию в рантайме (например, переключаться между реализациями с помощью dlclose и dlopen).

Код программы

functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#ifdef __cplusplus
extern "C" { // Совместимость для динамической загрузки
#endif

    float Derivative(float A, float deltaX);
    int* Sort(int* array, int size);

#ifdef __cplusplus
}
#endif

#endif
```

lib1.cpp

```
#include <cmath>
#include "functions.h"

extern "C" {
    float Derivative(float A, float deltaX) {
        return (cosf(A + deltaX) - cosf(A)) / deltaX;
    }

    int* Sort(int* array, int size) {
        // Пузырьковая сортировка
        for (int i = 0; i < size - 1; ++i) {
            for (int j = 0; j < size - i - 1; ++j) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
```

```
                }
            }
        }
        return array;
    }
}
```

lib2.cpp

```
#include <cmath>
#include "functions.h"

extern "C" {
    float Derivative(float A, float deltaX) {
        return (cosf(A + deltaX) - cosf(A - deltaX)) / (2 * deltaX);
    }

    int Partition(int* array, int low, int high) {
        int pivot = array[high]; // Опорный элемент
        int i = low - 1;

        for (int j = low; j < high; ++j) {
            if (array[j] < pivot) {
                ++i;
                // Обмен элементов
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
            }
        }
        // Обмен опорного элемента
        array[high] = array[i + 1];
        array[i + 1] = pivot;
        return i + 1;
    }

    void QuickSort(int* array, int low, int high) {
        if (low < high) {
            int pi = Partition(array, low, high);

            QuickSort(array, low, pi - 1);
            QuickSort(array, pi + 1, high);
        }
    }

    int* Sort(int* array, int size) {
        // Сортировка Хоара
        QuickSort(array, 0, size - 1);
        return array;
    }
```

```
}
```

program1.cpp

```
#include <iostream>
#include <vector>
#include "functions.h"

int main() {
    std::cout << "Программа №1 (статическая линковка)\n";

    while (true) {
        std::cout << "\nВведите команду:\n";
        std::cout << "1 A deltaX - вычислить производную cos(A) с приращением deltaX\n";
        std::cout << "2 N elem1 elem2 ... elemN - отсортировать массив из N элементов\n";
        std::cout << "3 - выйти\n";
        int command;
        std::cin >> command;

        if (command == 1) {
            float A, deltaX;
            std::cin >> A >> deltaX;
            float result = Derivative(A, deltaX);
            std::cout << "Производная cos(" << A << ") = " << result << "\n";
        } else if (command == 2) {
            int N;
            std::cin >> N;
            std::vector<int> vect(N);
            for (int i = 0; i < N; ++i) {
                std::cin >> vect[i];
            }
            Sort(vect.data(), N);
            std::cout << "Отсортированный массив: ";
            for (int i = 0; i < N; ++i) {
                std::cout << vect[i] << " ";
            }
            std::cout << "\n";
        } else if (command == 3) {
            break;
        } else {
            std::cout << "Неверная команда\n";
        }
    }

    return 0;
}
```

program2.cpp

```
#include <iostream>
#include <dlfcn.h>
#include <cstring>
#include <vector>

using DerivativeFunc = float(*)(float, float);
using SortFunc = int*(*)(int*, int);

int main() {
    std::cout << "Программа №2 (динамическая загрузка библиотек)\n";

    const char* pathToLib1 = std::getenv("PATH_TO_LIB1");
    if (!pathToLib1) {
        std::cerr << "Переменная окружения PATH_TO_LIB1 не установлена" << std::endl;
        return 1;
    }

    const char* pathToLib2 = std::getenv("PATH_TO_LIB2");
    if (!pathToLib2) {
        std::cerr << "Переменная окружения PATH_TO_LIB2 не установлена" << std::endl;
        return 1;
    }

    const char* lib_paths[] = {pathToLib1, pathToLib2};
    int current_lib = 0;
    void* handle = dlopen(lib_paths[current_lib], RTLD_LAZY);

    if (!handle) {
        std::cerr << "Ошибка загрузки библиотеки: " << dlerror() << "\n";
        return 1;
    }

    DerivativeFunc Derivative = reinterpret_cast<DerivativeFunc>(dlsym(handle,
"Derivative"));
    SortFunc Sort = reinterpret_cast<SortFunc>(dlsym(handle, "Sort"));

    char* error;
    if ((error = dlerror()) != nullptr) {
        std::cerr << "Ошибка получения символа: " << error << "\n";
        dlclose(handle);
        return 1;
    }

    while (true) {
        std::cout << "\nТекущая библиотека: " << lib_paths[current_lib] << "\n";
        std::cout << "Введите команду:\n";
        std::cout << "0 - переключить реализацию\n";
        std::cout << "1 A deltaX - вычислить производную cos(A) с приращением deltaX\n";
        std::cout << "2 N elem1 elem2 ... elemN - отсортировать массив из N элементов\n";
        std::cout << "3 - выйти\n";
        int command;
        std::cin >> command;
```

```
        if (command == 0) {
            // Переключаем библиотеку
            dlclose(handle);
            current_lib = 1 - current_lib;
            handle = dlopen(lib_paths[current_lib], RTLD_LAZY);
            if (!handle) {
                std::cerr << "Ошибка загрузки библиотеки: " << dlerror() << "\n";
                return 1;
            }
            Derivative = (DerivativeFunc)dlsym(handle, "Derivative");
            Sort = (SortFunc)dlsym(handle, "Sort");

            if ((error = dlerror()) != nullptr) {
                std::cerr << "Ошибка получения символа: " << error << "\n";
                dlclose(handle);
                return 1;
            }
        } else if (command == 1) {
            float A, deltaX;
            std::cin >> A >> deltaX;
            float result = Derivative(A, deltaX);
            std::cout << "Производная cos(" << A << ") = " << result << "\n";
        } else if (command == 2) {
            int N;
            std::cin >> N;
            std::vector<int> vect(N);
            for (int i = 0; i < N; ++i) {
                std::cin >> vect[i];
            }
            Sort(vect.data(), N);
            std::cout << "Отсортированный массив: ";
            for (int i = 0; i < N; ++i) {
                std::cout << vect[i] << " ";
            }
            std::cout << "\n";
        } else if (command == 3) {
            break;
        } else {
            std::cout << "Неверная команда\n";
        }
    }

    dlclose(handle);
    return 0;
}
```

CMakeLists.txt

```
add_library(lib1 SHARED src/lib1.cpp)
```

```
set_target_properties(lib1 PROPERTIES OUTPUT_NAME "lib1" PREFIX "")
target_include_directories(lib1 PRIVATE include)

add_library(lib2 SHARED src/lib2.cpp)
set_target_properties(lib2 PROPERTIES OUTPUT_NAME "lib2" PREFIX "")
target_include_directories(lib2 PRIVATE include)

add_executable(program1 program1.cpp)
add_executable(program2 program2.cpp)

target_include_directories(program1 PRIVATE include)

target_link_libraries(program1 PRIVATE lib1)
```

lab4_lib1_test.cpp

```
#include <gtest/gtest.h>
#include <cmath>
#include <functions.h>

TEST(DerivativeTest, Implementation1) {
    // Тестируем Derivative из lib1
    float A = 0.5f;
    float deltaX = 0.01f;

    float result = Derivative(A, deltaX);
    float expected = (cosf(A + deltaX) - cosf(A)) / deltaX;

    EXPECT_NEAR(result, expected, 1e-5);
}

TEST(SortTest, Implementation1) {
    // Тестируем Sort из lib1
    int array[] = {5, 2, 3, 1, 4};
    int expected[] = {1, 2, 3, 4, 5};
    int size = sizeof(array) / sizeof(array[0]);

    int* result = Sort(array, size);

    for (int i = 0; i < size; ++i) {
        EXPECT_EQ(result[i], expected[i]);
    }
}

int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

lab4_lib2_test.cpp

```
#include <gtest/gtest.h>
#include <cmath>
#include <functions.h>

TEST(DerivativeTest, Implementation2) {
    // Тестируем Derivative из lib2
    float A = 0.5f;
    float deltaX = 0.01f;

    float result = Derivative(A, deltaX);
    float expected = (cosf(A + deltaX) - cosf(A - deltaX)) / (2 * deltaX);

    EXPECT_NEAR(result, expected, 1e-5);
}

TEST(SortTest, Implementation2) {
    // Тестируем Sort из lib2
    int array[] = {5, 2, 3, 1, 4};
    int expected[] = {1, 2, 3, 4, 5};
    int size = sizeof(array) / sizeof(array[0]);

    int* result = Sort(array, size);

    for (int i = 0; i < size; ++i) {
        EXPECT_EQ(result[i], expected[i]);
    }
}

int main(int argc, char **argv) {
    ::testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}
```

Выводы

Использование динамических библиотек даёт гибкость: можно подключать разные реализации без пересборки, обновлять их «на лету» или даже переключаться во время исполнения. Однако статика упрощает распространение программы (не нужно таскать отдельный .so), но ограничивает обновление. Лабораторная работа продемонстрировала основы механизма dlopen/dlsym в Unix и различия между статической и динамической линковкой.