Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу
«Операционные системы»

Студент: Останина Анна Андреевна
Группа: М8О-208Б-22
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

Репозиторий

https://github.com/Imariiii/os_labs

Цель работы

Создание динамических библиотек и 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.

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).

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 14:

2	Расчет производной функции cos(x) в точке A с приращением deltaX	Float Derivative(float A, float deltaX)	f'(x) = (f(A + deltaX) – f(A))/deltaX	f'(x) = (f(A + deltaX) – f(A-deltaX))/(2*deltaX)
8	Перевод числа x из десятичной системы счисления в другую	Char* translation(long x)	Другая система счисления двоичная	Другая система счисления троичная

Описание работы программы

Функции, написанные в результате выполнения лабораторной работы:

- Расчет производной функции cos(x) в точке A с приращением deltaX.
- Перевод числа x из десятичной системы счисления в другую.

В программе работы были использованы следующие системные вызовы:

- dlsym() - получение адреса функции из динамической библиотеки,
- dlopen() - открытие динамической библиотеки,
- dlclose() - закрытие динамической библиотеки,
- dlerror() - возвращение строки, описывающей последнюю ошибку, произошедшую при вызове функций из динамической библиотеки.

Исходный код

lib.h

```
#pragma once

constexpr int NUM_BUFFER_SIZE = 33;

extern "C" {
char* Translation(long x);
float Derivative(float a, float deltaX);
}
```

utils.h

```
#pragma once
#include <cstddef>

void ReverseString(char* string, std::size_t length);
```

lib1.cpp

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

#include "lib.h"
#include "utils.h"

#include <math.h>

char* Translation(long x) {
    if (x < 0) {
        return nullptr;
    }
    char* binary = (char*)malloc(NUM_BUFFER_SIZE);
    if (!binary) {
        return binary;
    }
    int i = 0;
    do {
        binary[i++] = '0' + (x & 1);
        x >>= 1;
    } while(x);
    binary[i] = '\0';
    ReverseString(binary, i);
    return binary;
}

float Derivative(float a, float deltaX) {
    return (cos(a + deltaX) - cos(a)) / deltaX;
}
```

lib2.cpp

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

#include "lib.h"
#include "utils.h"

#include <math.h>

char* Translation(long x) {
    if (x < 0) {
        return nullptr;
    }
    char* trinary = (char*)malloc(NUM_BUFFER_SIZE);
    if (!trinary) {
        return trinary;
    }
    int i = 0;
    do {
        trinary[i++] = '0' + (x % 3);
        x /= 3;
    } while (x);
    trinary[i] = '\0';
    ReverseString(trinary, i);
    return trinary;
}


float Derivative(float a, float deltaX) {
    return (cos(a + deltaX) - cos(a - deltaX)) / (2 * deltaX);
}
```

utils.cpp

```
#include "utils.h"
#include <cstddef>

void ReverseString(char* string, std::size_t length) {
    for (std::size_t i = 0; i < length >> 1; ++i) {
        char temp = string[i];
        string[i] = string[length - i - 1];
        string[length - i - 1] = temp;
    }
}
```

dynamic_main.cpp

```
#include <dlfcn.h>

#include <array>
#include <iostream>
#include <sstream>
#include <vector>

class MainRoutine {
   private:
    using Function1Ptr = float(*)(float, float);
    using Function2Ptr = char *(*)(long);

    const char *const Function1Name = "Derivative";
    const char *const Function2Name = "Translation";

    Function1Ptr Derivative;
    Function2Ptr Translation;
    size_t currentMode;
    std::array<void *, 2> dls;

    void SwitchMode() {
        currentMode = 1 - currentMode;
        Derivative = reinterpret_cast<Function1Ptr>(
            dlsym(dls[currentMode], Function1Name));
        Translation = reinterpret_cast<Function2Ptr>(
            dlsym(dls[currentMode], Function2Name));
    }

    static void InvalidArgs() {
        std::cout << "Invalid arguments\n";
    }

    static void MissingArgs() {
        std::cout << "Missing arguments\n";
    }

    void Function1(float a, float deltaX) {
        std::cout << Derivative(a, deltaX) << std::endl;
    }

    void Function2(long x) {
        char *str = Translation(x);
        std::cout << str << '\n';
        std::free(str);
    }

    static void Help() {
        std::cout << "  1. help - for help\n";
        std::cout << "  2. exit - exits program\n";
        std::cout << "  3. 1 <a> <deltaX> - calculates derivative of cos(x)
of a with specific deltaX\n";
```

```
        std::cout << "  4. 2 <x> - translate x to binary number system\n";
        std::cout << "  5. 0 - switch libs\n";
    }

    static std::vector<std::string> Tokenize(const std::string &line) {
        std::vector<std::string> result;
        std::stringstream ss(line);
        while (ss) {
            std::string token;
            ss >> token;
            if (!token.empty()) {
                result.push_back(token);
            }
        }
        return result;
    }

    bool ProceedLine(const std::string &line) {
        std::vector<std::string> tokens = Tokenize(line);
        if (tokens[0] == "help") {
            Help();
        } else if (tokens[0] == "exit") {
            return false;
        } else if (tokens[0] == "1") {
            if (tokens.size() != 3) {
                MissingArgs();
                return true;
            }
            try {
                Function1(std::atof(tokens[1].c_str()),
std::atof(tokens[2].c_str()));
            } catch (std::invalid_argument &) {
                InvalidArgs();
            }
        } else if (tokens[0] == "2") {
            if (tokens.size() != 2) {
                MissingArgs();
                return true;
            }
            try {
                long x = std::stol(tokens[1]);
                Function2(x);
            } catch (std::invalid_argument &) {
                InvalidArgs();
            }
        } else if (tokens[0] == "0") {
            SwitchMode();
            std::cout << "Switched\n";
        } else {
            InvalidArgs();
        }
        return true;
    }

public:
    MainRoutine() = delete;
    MainRoutine(const MainRoutine &) = delete;
    MainRoutine(MainRoutine &&) = delete;
    MainRoutine &operator=(const MainRoutine &) = delete;
    MainRoutine &operator=(MainRoutine &&) = delete;
    MainRoutine(const char *lib1Path, const char *lib2Path) {
        dls[0] = dlopen(lib1Path, RTLD_LOCAL | RTLD_LAZY);
        if (dls[0] == nullptr) {
```

```
            throw std::invalid_argument("Can't open first lib.");
        }
        dls[1] = dlopen(lib2Path, RTLD_LOCAL | RTLD_LAZY);
        if (dls[1] == nullptr) {
            throw std::invalid_argument("Can't open second lib.");
        }
        currentMode = 1;
        SwitchMode();
    }

    ~MainRoutine() {
        dlclose(dls[0]);
        dlclose(dls[1]);
    }

    void Start() {
        std::string line;
        std::cout << "> ";

        while (std::getline(std::cin, line)) {
            if (!ProceedLine(line)) {
                break;
            }
            std::cout << "> ";
        }
    }
};

int main() {
    char *pathToLib1 = getenv("PATH_TO_LIB1");
    char *pathToLib2 = getenv("PATH_TO_LIB2");
    if (pathToLib1 == nullptr) {
        std::cerr << "PATH_TO_LIB1 is not specified\n";
        exit(EXIT_FAILURE);
    }
    if (pathToLib2 == nullptr) {
        std::cerr << "PATH_TO_LIB2 is not specified\n";
        exit(EXIT_FAILURE);
    }
    MainRoutine routine(pathToLib1, pathToLib2);
    routine.Start();
}
```

static_main.cpp

```
extern "C" {
#include "lib.h"
}

#include <iostream>
#include <sstream>
#include <stdexcept>
#include <string>
#include <vector>

class MainRoutine {
   private:
    static void InvalidArgs() {
        std::cout << "Invalid arguments\n";
    }

    static void MissingArgs() {
        std::cout << "Missing arguments\n";
    }
```

```
    static void Help() {
        std::cout << "  1. help - for help\n";
        std::cout << "  2. exit - exits program\n";
        std::cout << "  3. 1 <a> <deltaX> - calculates derivative of cos(x)
of a with specific deltaX\n";
        std::cout << "  4. 2 <x> - translate x to binary number system\n";
    }

    static void Function1(float a, float deltaX) {
        std::cout << Derivative(a, deltaX) << std::endl;
    }

    static void Function2(long x) {
        char *str = Translation(x);
        std::cout << str << '\n';
        std::free(str);
    }

    static std::vector<std::string> Tokenize(const std::string &line) {
        std::vector<std::string> result;
        std::stringstream ss(line);
        while (ss) {
            std::string token;
            ss >> token;
            if (!token.empty()) {
                result.push_back(token);
            }
        }
        return result;
    }

    static bool ProceedLine(const std::string &line) {
        std::vector<std::string> tokens = Tokenize(line);
        if (tokens[0] == "help") {
            Help();
        } else if (tokens[0] == "exit") {
            return false;
        } else if (tokens[0] == "1") {
            if (tokens.size() != 3) {
                MissingArgs();
                return true;
            }
            try {
                Function1(std::atof(tokens[1].c_str()),
std::atof(tokens[2].c_str()));
            } catch (std::invalid_argument &) {
                InvalidArgs();
            }
        }

        else if (tokens[0] == "2") {
            if (tokens.size() != 2) {
                MissingArgs();
                return true;
            }
            try {
                Function2(std::stol(tokens[1]));
            } catch (std::invalid_argument &) {
                InvalidArgs();
            }
        } else {
            InvalidArgs();
```

```
        }
        return true;
    }

   public:
    static void Start() {
        std::string line;
        std::cout << "> ";
        while (std::getline(std::cin, line)) {
            if (!ProceedLine(line)) {
                break;
            }
            std::cout << "> ";
        }
    }
};

int main() {
    MainRoutine::Start();
}
```

lab4_lib1_test.cpp

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

#include <cstring>

#include "lib.h"

constexpr const float EPS = 1e-6;
constexpr const float PI = 3.14159265359;

bool EqualFloat(float lhs, float rhs) {
    return std::abs(lhs - rhs) < EPS;
}

TEST(FourthLabTests, DerivativeTest) {
    EXPECT_TRUE(EqualFloat(Derivative(PI, EPS), 0));
}

TEST(FourthLabTests, TranslationTest) {
    auto deleter = [](char *str) { std::free(str); };
    std::unique_ptr<char, decltype(deleter)> str(Translation(31), deleter);
    EXPECT_TRUE(std::strcmp(str.get(), "11111") == 0);
    str.reset(Translation(0));
    EXPECT_TRUE(std::strcmp(str.get(), "0") == 0);
    str.reset(Translation(888));
    EXPECT_TRUE(std::strcmp(str.get(), "1101111000") == 0);
    str.reset(Translation(-1));
    EXPECT_EQ(str.get(), nullptr);
}
```

lab4_lib2_test.cpp

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

#include <cstring>
#include <utility>

#include "lib.h"

constexpr const float EPS = 1e-6;
constexpr const float PI = 3.14159265359;

bool EqualFloat(float lhs, float rhs) {
```

```
    return std::abs(lhs - rhs) < EPS;
}

TEST(FourthLabTests, DerivativeTest) {
    EXPECT_TRUE(EqualFloat(Derivative(PI, EPS), 0));
}

TEST(FourthLabTests, TranslationTest) {
    auto deleter = [](char *str) { std::free(str); };
    std::unique_ptr<char, decltype(deleter)> str(Translation(0), deleter);
    EXPECT_TRUE(std::strcmp(str.get(), "0") == 0);
    str.reset(Translation(5));
    EXPECT_TRUE(std::strcmp(str.get(), "12") == 0);
    str.reset(Translation(888));
    EXPECT_TRUE(std::strcmp(str.get(), "1012220") == 0);
    str.reset(Translation(-1));
    EXPECT_EQ(str.get(), nullptr);
}
```

Консоль

```
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/lab4$ ./static_main
> 1 2 0.001
-0.909001
> 1 2 0.01
-0.907201
> 2 300
100101100
> 2 534
1000010110
> exit
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/lab4$ cat run.sh
export PATH_TO_LIB1="./liblib1.so"
export PATH_TO_LIB2="./liblib2.so"
./dynamic_main
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/lab4$ ./run.sh
> help
  1. help - for help
  2. exit - exits program
  3. 1 <a> <deltaX> - calculates derivative of cos(x) of a with specific
deltaX
  4. 2 <x> - translate x to binary number system
  5. 0 - switch libs
> 1 2 0.001
-0.909001
> 1 2 0.01
-0.907201
> 2 300
100101100
> 2 534
1000010110
> exit
```

Запуск тестов

```
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/tests$ ./lab4_lib1_test
Running main() from /home/anna/labs_3sem/os_labs/build/_deps/googletest-
src/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from FourthLabTests
[ RUN      ] FourthLabTests.DerivativeTest
```

```
[       OK ] FourthLabTests.DerivativeTest (0 ms)
[ RUN      ] FourthLabTests.TranslationTest
[       OK ] FourthLabTests.TranslationTest (0 ms)
[----------] 2 tests from FourthLabTests (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
anna@anna-virtual-machine:~/labs_3sem/os_labs/build/tests$ ./lab4_lib2_test
Running main() from /home/anna/labs_3sem/os_labs/build/_deps/googletest-
src/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from FourthLabTests
[ RUN      ] FourthLabTests.DerivativeTest
[       OK ] FourthLabTests.DerivativeTest (0 ms)
[ RUN      ] FourthLabTests.TranslationTest
[       OK ] FourthLabTests.TranslationTest (0 ms)
[----------] 2 tests from FourthLabTests (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

Выводы

В ходе данной лабораторной работы были получены навыки использования динамических библиотек в ОС Linux, которые позволяют программе загружать и использовать функции из библиотек во время выполнения, что обеспечивает гибкость и возможность изменения программы без перекомпиляции.

Также были получены знания про этапы сборки программы и особенности использования extern "C" при линковке файлов с общим include.