Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №5 по курсу
«Операционные системы»

ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ

Студент: Лукманова Аэлита
Группа: М8О–201Б–19
Вариант: 13
Преподаватель: Миронов Е. С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

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 No1*), которая используют одну из библиотек, используя знания полученные на этапе компиляции;
- Тестовая программа (*программа No2*), 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 No2*). 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 34:
7: Подсчет площади плоской геометрической фигуры по двум сторонам. Фигуры прямоугольник и прямоугольный треугольник.

2: Расчет производной cosx в точке A с приращением deltaX.

Общие сведения о программе

Программа использует динамические библиотеки, которые компилируются из файлов Derivative.c и Square.c. Также используется заголовочные файлы: stdio.h, stdlib.h, stdint.h, stdbool.h, string.h, dlfcn.h.

В программе используются следующие системные вызовы:

1. **dlopen** – загружает динамическую библиотеку с указанным именем. В случае неуспеха возвращает NULL.
2. **dlclose –** уменьшает на единицу счетчик ссылок на указатель динамической библиотеки *handle*. Если нет других загруженных библиотек, использующих ее символы и если счетчик ссылок принимает нулевое значение, то динамическая библиотека выгружается. В случае успеха возвращает 0, иначе ненулевой результат.
3. **dlsym** – использует указатель на динамическую библиотеку, возвращаемую dlopen, и оканчивающееся нулем символьное имя, а затем возвращает адрес, указывающий на нужный символ. В случае неуспеха dlsym возвращает NULL.
4. **dlerror** – возвращает сообщение об ошибке, если ошибки не произошло, то возвращает NULL.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить ключи компилятора для создания динамических библиотек.
2. Изучить работу dlopen, dlclose, dlsym, dlerror.
3. Изучить принципы работы динамических библиотек, динамической линковки и динамической загрузки.
4. Реализовать функции для динамических библиотек.
5. Реализовать обработку не валидных данных и обработку ошибок в программе.

Основные файлы программы

Square.h:

```
#pragma once

float squareOfFigure(int typeOfFigure, float side1, float side2);
```

Square.c:

```
#include "Square.h"

#include <stdio.h>

#define EXPORT __attribute__((visibility("default")))


// Initializer.
__attribute__((constructor))
static void initializer(void) {
    printf("[%s] initializer()\n", __FILE__);
}

// Finalizer.
__attribute__((destructor))
static void finalizer(void) {
    printf("[%s] finalizer()\n", __FILE__);
}

//внутренняя функция, которая будет использоваться функцией из библиотеки
//не будем делать проверку на невалидные данные
float squareOfRect(float side1, float side2) {
    return side1 * side2;
}

//здесь будем делать проверку на невалидные данные, т.к. это функция библиотеки
EXPORT
float squareOfFigure(int typeOfFigure, float side1, float side2) {
    if ((side1 <= 0) || (side2 <= 0)) {
        printf("Проверьте введенные стороны. Они должны быть положительными
числами\n");
        return 0;
    }
    switch (typeOfFigure) {
        case 1:
            return squareOfRect(side1, side2);
            break;
```

```
        case 2:
            return 1./2*squareOfRect(side1, side2);
            break;
        default:
            printf("Ошибка в вводе. Первый аргумент должен быть \"1\"(площадь прямо-
угольника) или \"2\"(площадь треугольника)");
            break;
    }
    return 0;
}
```

SquareDependent.c:

```
#include <stdio.h>
#include <unistd.h>
#include "Square.h"

int main(int argc, char **argv) {
    printf("[start_test]\n");

    int typeOfFigure;
    float side1;
    float side2;
    float result;
    char mode;

    printf("Hello, you are in the dependent realization of library with calculating
square\n");
    printf("Do you want to change mode to runtime-loaded library? Y/N\n");
    scanf("%s", &mode);
    if (mode == 'Y' || mode == 'y') {
        execv("./SquareRuntime", NULL);
    }
    printf("If you want to know square of rectangle input 1 and if of triangle input
2:\n");
    scanf("%d", &typeOfFigure);

    printf("First side:\n");
    scanf("%f", &side1);

    printf("Second side:\n");
    scanf("%f", &side2);

    result = squareOfFigure(typeOfFigure, side1, side2);
    if (result == 0) {
        printf("Error: Wrong input");
    } else {
        printf("Result: %f\n", result);
    }


    printf("[end_test]\n");
    return 0;
}
```

SquareRuntime.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <string.h>
#include <unistd.h>
#include "Square.h"


int main(int argc, char **argv) {
    printf("[start_test]\n");

    // Open the library.
```

```
    char *lib_name = "./libSquare.dylib";
    void *lib_handle = dlopen(lib_name, RTLD_NOW);

    if (lib_handle) {
        printf("[%s] dlopen(\"%s\", RTLD_NOW): Successful\n", __FILE__, lib_name);
    }
    else {
        printf("[%s] Unable to open library: %s\n",
            __FILE__, dlerror());
        exit(EXIT_FAILURE);
    }

    // Get the symbol addresses.
    float (*squareOfFigure)(int, float, float) = dlsym(lib_handle, "squareOfFigure");
    if (squareOfFigure) {
        printf("[%s] dlsym(lib_handle, \"squareOfFigure\"): Successful\n", __FILE__);
    }
    else {
        printf("[%s] Unable to get symbol: %s\n",
            __FILE__, dlerror());
        exit(EXIT_FAILURE);
    }



    int typeOfFigure;
    float side1;
    float side2;
    float result;
    char mode;

    printf("Hello, you are in the runtime-loaded realization of library with
calculating square\n");
    printf("Do you want to change mode to dependent library? Y/N\n");
    scanf("%s", &mode);
    if (mode == 'Y' || mode == 'y') {
        execv("./SquareDependent", NULL);
    }

    printf("If you want to know square of rectangle input 1 and if of triangle input
2:\n");
    scanf("%d", &typeOfFigure);

    printf("First side:\n");
    scanf("%f", &side1);

    printf("Second side:\n");
    scanf("%f", &side2);

    result = squareOfFigure(typeOfFigure, side1, side2);
    if (result == 0) {
        printf("Error: Wrong input");
    } else {
        printf("Result: %f\n", squareOfFigure(typeOfFigure, side1, side2));
    }



    // Close the library.
    if (dlclose(lib_handle) == 0) {
        printf("[%s] dlclose(lib_handle): Successful\n", __FILE__);
    }
    else {
        printf("[%s] Unable to open close: %s\n",
            __FILE__, dlerror());
    }

    printf("[end_test]\n");
    return 0;
}
```

Derivative.h:

```
#pragma once
float derivativeCosWay1(float A, float deltaX);
float derivativeCosWay2(float A, float deltaX);
```

Derivative.c:

```
#include "Square.h"

#include <stdio.h>

#define EXPORT __attribute__((visibility("default")))


#include "Derivative.h"

#include <math.h>
#include <stdio.h>

#define EXPORT __attribute__((visibility("default")))


__attribute__((constructor))
static void initializer(void) {
    printf("[%s] initializer()\n", __FILE__);
}

__attribute__((destructor))
static void finalizer(void) {
    printf("[%s] finalizer()\n", __FILE__);
}


EXPORT
float derivativeCosWay1(float A, float deltaX) {
    return (cos(A+deltaX)-cos(A))/(deltaX);
}

EXPORT
float derivativeCosWay2(float A, float deltaX) {
    return (cos(A+deltaX)-cos(A-deltaX))/(2*deltaX);
}
```

DerivativeDependent.c:

```
#include <stdio.h>
#include <unistd.h>

#include "Derivative.h"


int main(int argc, char **argv) {
    printf("[start_test]\n");

    printf("Hello, you are in the dependent realization of library with calculating
derivative of cosine\n");
    printf("Do you want to change mode to runtime-loaded library? Y/N\n");
    char libMode;
    scanf("%s", &libMode);
    if (libMode == 'Y' || libMode == 'y') {
        if (execv("./DerivativeRuntime", NULL)) // todo - remove if
        {
            printf("ERROR: Failed to change mode to runtime-loaded library\n");
        }
    }

    printf("Input 1 or 2 to calculate derivative of cosine in point A and delta x with
1st way or 2nd way:\n");
```

```
    int way;
    if (scanf("%d", &way) != 1 || !(way == 1 || way == 2) ) {
        printf("Wrong input. The number needs to be \"1\"(1st way) or \"2\"(2nd way)
\n");
        return 0;
    }

    printf("Point A:\n");
    float A;
    if (scanf("%f", &A) != 1) {
        printf("Wrong input. The point needs to be float number\n");
        return 0;
    }

    printf("delta x:\n");
    float deltaX;
    if ((scanf("%f", &deltaX) != 1) || (deltaX==0)) {
        printf("Wrong input. Delta needs to be float non zero number\n");
        return 0;
    }

    if (way == 1) {
        printf("Result: %f\n", derivativeCosWay1(A, deltaX));
    } else if (way == 2) {
        printf("Result: %f\n", derivativeCosWay2(A, deltaX));
    }

    printf("[end_test]\n");
    return 0;
}
```

DerivativeRuntime.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <string.h>
#include <unistd.h>

#include "Derivative.h"


int main(int argc, char **argv) {
    printf("[start_test]\n");

    // Open the library.
    char *lib_name = "./libDerivative.dylib";
    void *lib_handle = dlopen(lib_name, RTLD_NOW);

    if (lib_handle) {
        printf("[%s] dlopen(\"%s\", RTLD_NOW): Successful\n", __FILE__, lib_name);
    }
    else {
        printf("[%s] Unable to open library: %s\n",
            __FILE__, dlerror());
        exit(EXIT_FAILURE);
    }

    // Get the symbol addresses.
    float (*derivativeCosWay1)(float, float) = dlsym(lib_handle, "derivativeCosWay1");
    if (derivativeCosWay1) {
        printf("[%s] dlsym(lib_handle, \"derivativeOfCos\"): Successful\n", __FILE__);
    }
    else {
        printf("[%s] Unable to get symbol: %s\n",
            __FILE__, dlerror());
        exit(EXIT_FAILURE);
    }
```

```
    float (*derivativeCosWay2)(float, float) = dlsym(lib_handle, "derivativeCosWay2");
    if (derivativeCosWay2) {
        printf("[%s] dlsym(lib_handle, \"derivativeOfCos\"): Successful\n", __FILE__);
    }
    else {
        printf("[%s] Unable to get symbol: %s\n",
            __FILE__, dlerror());
        exit(EXIT_FAILURE);
    }



    printf("Hello, you are in the runtime-loaded realization of library with
calculating derivative of cosine\n");
    printf("Do you want to change mode to dependent library? Y/N\n");
    char libMode;
    scanf("%s", &libMode);
    if (libMode == 'Y' || libMode == 'y') {
        if (execv("./DerivativeDependent", NULL)) // TODO: - remove if
        {
            printf("ERROR: Failed to change mode to dependent library\n");
        }
    }

    printf("Input 1 or 2 to calculate derivative of cosine in point A and delta x with
1st way or 2nd way:\n");
    int way;
    if (scanf("%d", &way) != 1 || !(way == 1 || way == 2) ) {
        printf("Wrong input. The number needs to be \"1\"(1st way) or \"2\"(2nd way)
\n");
        return 0;
    }

    printf("Point A:\n");
    float A;
    if (scanf("%f", &A) != 1) {
        printf("Wrong input. The point needs to be float number\n");
        return 0;
    }

    printf("delta x:\n");
    float deltaX;
    if ((scanf("%f", &deltaX) != 1) || (deltaX==0)) {
        printf("Wrong input. Delta needs to be float non zero number\n");
        return 0;
    }

    if (way == 1) {
        printf("Hey");
        printf("Result: %f\n", derivativeCosWay1(A, deltaX));
    } else if (way == 2) {
        printf("Hey");
        printf("Result: %f\n", derivativeCosWay2(A, deltaX));
    }



    // Close the library.
    if (dlclose(lib_handle) == 0) {
        printf("[%s] dlclose(lib_handle): Successful\n", __FILE__);
    }
    else {
        printf("[%s] Unable to open close: %s\n",
            __FILE__, dlerror());
    }

    printf("[end_test]\n");
    return 0;
}
```

Пример работы

```
aelitalukmanova@MacBook-Pro-Aelita os-5 % cd os-5
aelitalukmanova@MacBook-Pro-Aelita os-5 % ls
Derivative   Square
aelitalukmanova@MacBook-Pro-Aelita os-5 % cd Square
aelitalukmanova@MacBook-Pro-Aelita Square % clang -dynamiclib -std=gnu99 Square.c
-current_version 1.0 -compatibility_version 1.0 -fvisibility=hidden -o libSquare.dylib
aelitalukmanova@MacBook-Pro-Aelita Square % clang SquareDependent.c libSquare.dylib -o
SquareDependent
aelitalukmanova@MacBook-Pro-Aelita Square % clang SquareRuntime.c -o SquareRuntime
aelitalukmanova@MacBook-Pro-Aelita Square % ./SquareDependent
[Square.c] initializer()
[start_test]
Hello, you are in the dependent realization of library with calculating square
Do you want to change mode to runtime-loaded library? Y/N
n
If you want to know square of rectangle input 1 and if of triangle input 2:
1
First side:
2 3
Second side:
Result: 6.000000
[end_test]
[Square.c] finalizer()
aelitalukmanova@MacBook-Pro-Aelita Square % ./SquareDependent
[Square.c] initializer()
[start_test]
Hello, you are in the dependent realization of library with calculating square
Do you want to change mode to runtime-loaded library? Y/N
y
[start_test]
[Square.c] initializer()
[SquareRuntime.c] dlopen("./libSquare.dylib", RTLD_NOW): Successful
[SquareRuntime.c] dlsym(lib_handle, "squareOfFigure"): Successful
Hello, you are in the runtime-loaded realization of library with calculating square
Do you want to change mode to dependent library? Y/N
n
If you want to know square of rectangle input 1 and if of triangle input 2:
2
First side:
2
Second side:
3
Result: 3.000000
[Square.c] finalizer()
[SquareRuntime.c] dlclose(lib_handle): Successful
[end_test]
aelitalukmanova@MacBook-Pro-Aelita Square % ./SquareRuntime
[start_test]
[Square.c] initializer()
[SquareRuntime.c] dlopen("./libSquare.dylib", RTLD_NOW): Successful
[SquareRuntime.c] dlsym(lib_handle, "squareOfFigure"): Successful
Hello, you are in the runtime-loaded realization of library with calculating square
Do you want to change mode to dependent library? Y/N
n
If you want to know square of rectangle input 1 and if of triangle input 2:
1
First side:
2
Second side:
3
Result: 6.000000
[Square.c] finalizer()
[SquareRuntime.c] dlclose(lib_handle): Successful
[end_test]

aelitalukmanova@MacBook-Pro-Aelita Derivative % clang -dynamiclib -std=gnu99
Derivative.c -current_version 1.0 -compatibility_version 1.0 -fvisibility=hidden -o
libDerivative.dylib
aelitalukmanova@MacBook-Pro-Aelita Derivative % clang DerivativeDependent.c lib-
Derivative.dylib -o DerivativeDependent
aelitalukmanova@MacBook-Pro-Aelita Derivative % ./DerivativeDependent
```

```
[Derivative.c] initializer()
[start_test]
Hello, you are in the dependent realization of library with calculating derivative of
cosine
Do you want to change mode to runtime-loaded library? Y/N
n
Input 1 or 2 to calculate derivative of cosine in point A and delta x with 1st way or
2nd way:
2
Point A:
2
delta x:
1
Result: -0.765147
[end_test]
[Derivative.c] finalizer()
aelitalukmanova@MacBook-Pro-Aelita Derivative % clang DerivativeRuntime.c -o
DerivativeRuntime
aelitalukmanova@MacBook-Pro-Aelita Derivative % ./DerivativeRuntime
[start_test]
[Derivative.c] initializer()
[DerivativeRuntime.c] dlopen("./libDerivative.dylib", RTLD_NOW): Successful
[DerivativeRuntime.c] dlsym(lib_handle, "derivativeOfCos"): Successful
[DerivativeRuntime.c] dlsym(lib_handle, "derivativeOfCos"): Successful
Hello, you are in the runtime-loaded realization of library with calculating
derivative of cosine
Do you want to change mode to dependent library? Y/N
n
Input 1 or 2 to calculate derivative of cosine in point A and delta x with 1st way or
2nd way:
1
Point A:
2
delta x:
1
HeyResult: -0.573846
[Derivative.c] finalizer()
[DerivativeRuntime.c] dlclose(lib_handle): Successful
[end_test]
```

Вывод

Динамические библиотеки, в отличии от статических, позволяют сделать зависящие от них приложения меньше по памяти за счет того, что динамическую библиотеку нужно лишь раз выгрузить в память, чтобы ей пользовались все, кто от нее зависит. Существует два способа использования динамических библиотек: динамическая компоновка в момент загрузки и динамическая загрузка на этапе исполнения. В первом случае всю работу по загрузке необходимых зависимостей выполняют операционная система и динамический компоновщик операционной системы, а во втором случае мы сами можем явно в коде указать какие библиотеки и когда подгрузить, что дает нам большую гибкость в действиях.