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: Ядров Артем Леонидович
Группа: М8О-208Б-20
Вариант: 14
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/Yadroff/OS/tree/master/2_lab

Постановка задачи

Цель работы

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

N	Описание	Сигнатура	Реализация 1	Реализация 2
2	Рассчет 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

Общие сведения о программе

Программа компилируется в двух файлах: static_main.c и dynamic_main.c

Используемые библиотечные вызовы:

void *dlopen(const char *filename, int flag);	Загружает динамическую библиотеку, имя которой указано в строке filename и возвращает прямой указатель на начало загруженной библиотеки.
const char *dlerror(void);	Возвращает указатель на начало строки, описывающей ошибку, полученную на предыдущем вызове.
void *dlsym(void *handle, char *symbol);	Получает параметр handle, который является выходом вызова dlopen и параметр symbol, который является строкой, в которой содержится название символа, который необходимо загрузить из библиотеки. Возвращает указатль на область памяти, в которой содержится необходимый символ.
int dlclose(void *handle);	Уменьшает счетчик ссылок на указатель handle и если он равен нулю, то освобождает библиотеку.

Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить работу с библиотеками.
2. Реализовать две библиотеки согласно заданию.
3. Реализовать две программы (для работы с динамическими и статическими библиотеками).

Исходный код

realization.h

```
#ifndef INC_5_LAB_LIBRARY_H
#define INC_5_LAB_LIBRARY_H

extern float Derivative(float A, float deltaX);
extern char* Translation(long x);
#endif//INC_5_LAB_LIBRARY_H
```

1realization.c

```
#include "realization.h"

#include <malloc.h>
#include <math.h>

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

char *Translation(long x) {
  char *res = (char *)malloc(sizeof(char));
  int n = 0;
  while (x > 0) {
        res[n++] = x % 2 + '0';
        x /= 2;
        res = realloc(res, (n + 1) * sizeof(char));
  }
  res[n] = '\0';
  char t;
  for (int i = 0; i < n / 2; ++i) {
        t = res[i];
```

```
        res[i] = res[n - i - 1];
        res[n - i - 1] = t;
  }
  return res;
}
```

2realization.c

```
//
// Created by Temi4 on 21.11.2021.
//

#include "realization.h"
#include <malloc.h>
#include <math.h>

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

char* Translation(long x){
  char *res = (char *)malloc(sizeof(char));
  int n = 0;
  while (x > 0) {
        res[n++] = x % 3 + '0';
        x /= 3;
        res = realloc(res, (n + 1) * sizeof(char));
  }
  res[n] = '\0';
  char t;
```

```
  for (int i = 0; i < n / 2; ++i) {
        t = res[i];
        res[i] = res[n - i - 1];
        res[n - i - 1] = t;
  }
  return res;
}
```

static_main.c

```
//
// Created by Temi4 on 21.11.2021.
//

#include "realization.h"

#include <stdio.h>

int main(){
  int cmd = 0;
  while (scanf("%d", &cmd) != EOF){
        switch (cmd) {
          case 1: {
                float A, deltaX;
                if (scanf("%f %f", &A, &deltaX) == 2) {
                  printf("%.6f\n", Derivative(A, deltaX));
                }
                break;
          }
          case 2:{
```

```
                long x;
                if (scanf("%ld", &x) == 1){
                  printf("%s\n", Translation(x));
                }
                break;
            }
            default:{
                printf("Wrong answer\n");
            }
          }
  }
  return 0;
}
```

dynamic_main.c

```
//
// Created by Temi4 on 21.11.2021.
//

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

typedef enum{
  FIRST,
  SECOND,
} CONTRACT;

CONTRACT cont = FIRST;
```

```
const char* libName1 = "libFirst.so";
const char* libName2 = "libSecond.so";

float (*Derivative)(float, float) = NULL;
char* (*Translation)(long) = NULL;

void *libHandle = NULL;

void loadLibs(CONTRACT contract){
  const char* name;
  switch (contract) {
  case FIRST:{
        name = libName1;
        break;
  }
  case SECOND:{
        name = libName2;
        break;
  }
  }
  libHandle = dlopen(name, RTLD_LAZY);
  if (libHandle == NULL){
        perror("dlopen");
        exit(EXIT_FAILURE);
  }
}

void loadContract(){
  loadLibs(cont);
```

```
  Derivative = dlsym(libHandle, "Derivative");
  Translation = dlsym(libHandle, "Translation");
}

void changeContract(){
  dlclose(libHandle);
  switch (cont) {
        case FIRST:{
              cont = SECOND;
              break;
        }
        case SECOND:{
          cont = FIRST;
          break;
        }
  }
  loadContract();
}

int main(){
  loadContract();
  int cmd = 0;
  while (scanf("%d", &cmd) != EOF){
        switch (cmd) {
          case 0:{
          changeContract();
          printf("Contract has been changed\n");
          switch (cont) {
                case FIRST: {
```

```
            printf("Contract is First\n");
            break;
          }
          case SECOND:{
            printf("Contract is Second\n");
          }
        }
        break;
      }
      case 1:{
        float A, deltaX;
        if (scanf("%f %f", &A, &deltaX) == 2){
          printf("%.6f\n", Derivative(A, deltaX));
        }
        break;
      }
      case 2:{
        long x;
        if (scanf("%ld", &x) == 1){
          printf("Translation from 10 to ");
          switch (cont) {
            case FIRST:{
              printf("2");
              break;
            }
            case SECOND:{
              printf("3");
            }
          }
```

```
                printf(" base %s\n", Translation(x));
            }
            break;
          }
          default:{
            printf("Wrong answer\n");
          }
          }
    }
    return 0;
}
```

Сборка программы

```
[Temi4@localhost src]$ gcc -fPIC -lm -c FirstRealization.c -o First.o
[Temi4@localhost src]$ gcc -fPIC -lm -c SecondRealization.c -o Second.o
[Temi4@localhost src]$ gcc -shared -o libFirst.so First.o
[Temi4@localhost src]$ gcc -shared -o libSecond.so Second.o
[Temi4@localhost src]$ sudo cp libFirst.so /usr/lib
[Temi4@localhost src]$ sudo cp libSecond.so /usr/lib
[Temi4@localhost src]$ gcc static_main.c -lFirst -lm -o first_static
[Temi4@localhost src]$ gcc static_main.c -lSecond -lm -o second_static
[Temi4@localhost src]$ gcc dynamic_main.c -ldl -lm -o dynamic
```

Демонстрация работы программы

```
[Temi4@localhost src]$ ./FirstStatic
2 10
1010
1 0 0.1
-0.049958
1 0 0.0001
0.000000
1 0 0.001
```

-0.000477
1 1.57 0.001
-1.000047
[Temi4@localhost src]$ ./SecondStatic
2 10
101
1 0 0.001
0.000000
1 1.57 0.001
-1.000046
[Temi4**@**localhost src**]**$ ./dynamic
1 0 0.001
-0.000477
1 1.57 0.001
-1.000047
2 10
Translation from 10 to 2 base 1010
0
Contract has been changed
Contract is Second
1 0 0.001
0.000000
1 1.57 0.001
-1.000046
2 10
Translation from 10 to 3 base 101

Выводы

В ходе лабораторной работы я познакомился с созданием динамических библиотек в ОС Linux, а также с возможностью загружать эти библиотеки в ходе выполнения программы. Динамические библиотеки помогают уменьшить размер исполняемых файлов. Загрузка динамических библиотек во время выполнения также упрощает компиляцию. Однако также можно подключить библиотеку к программе на этапе линковки. Она все равно загрузится при выполнении, но теперь программа будет изначально знать что и где искать. Если библиотека находится не в стандартной для динамических библиотек директории, необходимо также сообщить линкеру, чтобы тот передал необходимый путь в исполняемый файл. При помощи библиотек мы можем писать более сложные вещи, которые используют простые функции,

структуры и т.п., написанные ранее и сохраненные в различных библиотеках.