Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Информационные технологии и прикладная математика»

Лабораторная работа №5
по курсу «Операционные системы»

«Динамические библиотеки»

Студент: Минеева Светлана Алексеевна

Группа: М8О-210Б-21

Преподаватель: Миронов Е.С

Вариант №14
Оценка: ____
Дата: 28.11.2022
Подпись: _____

Москва, 2022

Содержание:

1. Цель работы

Целью является приобретение практических навыков в:

- Создание программ динамических библиотек;
- Создание программ, которые используют функции динамических библиотек.

2. 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.

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.

3. Вариант задания

Вариант №14:

№	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

4. Общие сведения о программе

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

Основные библиотечные вызовы в программе:

1. void *dlopen(const char *filename, int flag) – загружает динамическую библиотеку, имя которой указано в строке filename, и возвращает прямой указатель на начало загруженной библиотеки;
2. const char *dlerror(void) – возвращает строку символов с нулевым конечком, которая описывает последнюю ошибку, произошедшую во время обработки динамического связывания; если с момента последнего вызова dlerror() ошибок динамического связывания не было, dlerror() вернет NULL;
3. void *dlsym(void *handle, char *symbol) – принимает указатель на начало загруженной библиотеки, возвращаемой dlopen(), и имя символа, и возвращает адрес, по которому этот символ загружается в память;
4. int dlclose(void *handle) – 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.

5. Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо:

1. Изучить работу с библиотеками, принципы работы dlopen, dlerror, dlsym, dlclose;
2. Реализовать две библиотеки, используя две разных реализации функций;
3. Реализовать две программы (для работы с динамическими и статическими библиотеками);
4. Отладить программы и протестировать на тестах.

6. Текст программы

realization.h

```
#ifndef REALIZATION_H
#define REALIZATION_H

extern float derivative(float point, float increment);
extern char* translation(long numeric);

#endif
```

first_realization.c

```
#include "realization.h"
#include <stdlib.h>
#include <math.h>

float derivative(float point, float increment) {
  return (cosf(point + increment) - cosf(point)) / increment;
}

char* translation(long numeric) {
  char* result = (char*)malloc(sizeof(char));
  int index = 0;

  while(numeric > 0) {
    result[index] = numeric % 2 + '0';
    index += 1;
    numeric /= 2;
    result = realloc(result, (index + 1) * sizeof(char));
  }

  result[index] = '\0';
  char symbol;

  for(int permutation_index = 0; permutation_index < index / 2; ++permutation_index) {
    symbol = result[permutation_index];
    result[permutation_index] = result[index - permutation_index - 1];
    result[index - permutation_index - 1] = symbol;
  }
```

```
  return result;
}
```

second_realization.c

```
#include "realization.h"
#include <stdlib.h>
#include <math.h>

float derivative(float point, float increment) {
  return (cosf(point + increment) - cosf(point - increment)) / (2 * increment);
}

char* translation(long numeric){
  char* result = (char*)malloc(sizeof(char));
  int index = 0;

  while(numeric > 0) {
    result[index] = numeric % 3 + '0';
    index += 1;
    numeric /= 3;
    result = realloc(result, (index + 1) * sizeof(char));
  }

  result[index] = '\0';
  char symbol;

  for(int permutation_index = 0; permutation_index < index / 2; ++permutation_index) {
    symbol = result[permutation_index];
    result[permutation_index] = result[index - permutation_index - 1];
    result[index - permutation_index - 1] = symbol;
  }

  return result;
}
```

static_main.c

```
#include "realization.h"
#include <stdio.h>

int main() {
  int command = 0;

  while(scanf("%d", &command) != EOF) {
    switch (command) {
      case 1: {
        float point, increment;

        if(scanf("%f %f", &point, &increment) == 2) {
          printf("%.6f\n", derivative(point, increment));
```

```
        }

        break;
      }

      case 2: {
        long numeric;

        if(scanf("%ld", &numeric) == 1) {
          printf("%s\n", translation(numeric));
        }

        break;
      }

      default: {
        printf("Invalid request\n");
      }
    }
  }

  return 0;
}
```

dynamic_main.c

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

typedef enum {
  first_contract,
  second_contract,
} contracts;

contracts contract = first_contract;

const char* first_library_name = "libfirst.so";
const char* second_library_name = "libsecond.so";

float (*derivative)(float, float) = NULL;
char* (*translation)(long) = NULL;

void* lib_handle = NULL;

void load_lib(contracts contract) {
  const char* name;

  switch (contract) {
    case first_contract: {
      name = first_library_name;
      break;
    }
```

```
    case second_contract: {
      name = second_library_name;
      break;
    }
  }

  lib_handle = dlopen(name, RTLD_LAZY);

  if (lib_handle == NULL) {
    perror("dlopen");
    exit(EXIT_FAILURE);
  }
}

void load_contract() {
  load_lib(contract);
  derivative = dlsym(lib_handle, "derivative");
  translation = dlsym(lib_handle, "translation");
}

void change_contract() {
  dlclose(lib_handle);

  switch(contract) {
    case first_contract: {
      contract = second_contract;
      break;
    }

    case second_contract: {
      contract = first_contract;
      break;
    }
  }

  load_contract();
}

int main() {
  load_contract();
  int command = 0;

  while (scanf("%d", &command) != EOF) {
    switch(command) {
      case 0: {
        change_contract();
        printf("Contract has been changed\n");

        switch(contract) {
          case first_contract: {
            printf("Contract is first\n");
            break;
```

```
          }

          case second_contract: {
            printf("Contract is second\n");
          }
        }

        break;
      }

      case 1: {
        float point, increment;

        if(scanf("%f %f", &point, &increment) == 2) {
          printf("%.6f\n", derivative(point, increment));
        }

        break;
      }

      case 2: {
        long numeric;

        if(scanf("%ld", &numeric) == 1) {
          printf("Translation from 10 to ");

          switch(contract) {
            case first_contract: {
              printf("2");
              break;
            }

            case second_contract: {
              printf("3");
            }
          }

          printf(" number system: %s\n", translation(numeric));
        }

      break;
      }

      default: {
        printf("Invalid request\n");
      }
    }
  }

  return 0;
}
```

7. Демонстрация работы программы

```
Last login: Sun Nov 27 15:08:38 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-MacBook:~ macbookpro$ gcc -fPIC -lm -c first_realization.c -o first.o
MacBook-Pro-MacBook:~ macbookpro$ gcc -fPIC -lm -c second_realization.c -o second.o
MacBook-Pro-MacBook:~ macbookpro$ gcc -shared -o libfirst.so first.o
MacBook-Pro-MacBook:~ macbookpro$ gcc -shared -o libsecond.so second.o
MacBook-Pro-MacBook:~ macbookpro$ sudo cp libfirst.so /usr/local/lib
Password:
MacBook-Pro-MacBook:~ macbookpro$ sudo cp libsecond.so /usr/local/lib
MacBook-Pro-MacBook:~ macbookpro$ gcc static_main.c -lfirst -lm -o first_static
MacBook-Pro-MacBook:~ macbookpro$ gcc static_main.c -lsecond -lm -o second_static
MacBook-Pro-MacBook:~ macbookpro$ gcc dynamic_main.c -ldl -lm -o dynamic
MacBook-Pro-MacBook:~ macbookpro$ ./first_static
1 1.9 0.01
-0.944668
1 0 0.1
-0.049958
2 15
1111
2 2
10
MacBook-Pro-MacBook:~ macbookpro$ ./second_static
1 0.6 0.0001
-0.564754
1 0 0.1
0.000000
2 15
120
2 3
10
MacBook-Pro-MacBook:~ macbookpro$ ./dynamic
1 1.9 0.01
-0.944668
2 15
Translation from 10 to 2 number system: 1111
0
Contract has been changed
Contract is second
1 0 0.1
0.000000
2 15
Translation from 10 to 3 number system: 120
0
Contract has been changed
Contract is first
2 2
Translation from 10 to 2 number system: 10
```

MacBook-Pro-MacBook:~ macbookpro$

8. Вывод

Я создала динамические библиотеки, которые реализуют определенный функционал, создала программы, которые используют функции динамических библиотек. Мною были освоены такие библиотечные вызовы, как dlopen(), dlerror(), dlsym(), dlclose().

Динамические библиотеки можно загружать в ходе выполнения программы, а также на этапе линковки. Их загрузка во время выполнения программы упрощает компиляцию программы, а также уменьшает размер исполняемых файлов в случае реально высокой сложности задачи. Используя библиотеки, можно писать более сложные вещи, которые используют уже написанные ранее и хранящиеся в библиотеках простые функции, структуры.