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: Харьков Павел Александрович
Группа: М8О–206Б–19
Вариант: 14
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021.

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Требуется создать динамическую библиотеку, которая реализует определенный функционал. Далее использовать данную библиотеку 2-мя способами:

1. Во время компиляции (на этапе «линковки»/linking)
2. Во время исполнения программы, подгрузив библиотеку в память с помощью системных вызовов

В итоге программа должна состоять из следующих частей:

- Динамическая библиотека, реализующая заданных вариантом интерфейс;
- Тестовая программа, которая используют библиотеку, используя знания полученные на этапе компиляции;
- Тестовая программа, которая использует библиотеку, используя только местоположение динамической библиотеки и ее интерфейс.

Пользовательский ввод для обоих программ должен быть организован следующим образом:

- 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»;
- «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;
- «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:

Функция 1: расчёт производной функции cos(x) в точке A с приращением deltaX (float Derivative(float A, float deltaX)):

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

Функция 2: перевод числа X из десятичной системы счисления (char* Translation(long X)) в:

- Двоичную систему счисления.
- Троичную систему счисления.

Общие сведения о программе.

Программы компилируется из файлов task1.c, task2, deriv.c, translation.c . Также используется заголовочные файлы: unistd.h, stdlib.h, stdio.h, math.h, dlfcn.h. В программе используются следующие системные вызовы:

- dlopen – загружает динамический общий объект (общую библиотеку) из файла, имя которого указано в строке filename (завершается null) и возвращает непрозрачный описатель на загруженный объект.
- dlsym – функция возвращает адрес, по которому символ расположен в памяти(указывается одним из аргументов).
- dlclose – уменьшает счётчик ссылок на динамически загружаемый общий объект, на который ссылается handle. Если счётчик ссылок достигает нуля, то объект выгружается. Все общие объекты, которые были автоматически загружены при вызове dlopen() для объекта, на который ссылается handle, рекурсивно закрываются таким же способом.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы dlsym, dlopen, dlclose.
2. Написать по 2 библиотеки, используя 2 разных реализации функций.
3. В файле test1.c подключить библиотеку на этапе компиляции.
4. В файле test2.c загрузить библиотечные функции в runtime, с помощью dlsym, dlopen, dlclose.
5. Отладить программу и протестировать на тестах.

Основные файлы программы.

task1.c

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

```
#include <stdlib.h>

#include "lib/first/first.h"

int main(){
    int command;
    printf("> ");
    while(scanf("%d", &command) == 1)
    {
        if(command == 1){
            float A = 1;
            float deltaX = 1;
            if(scanf("%f %f", &A, &deltaX) != 2){
                break;
            }
            printf("%f\n", Derivative(A, deltaX));
        }
        else if(command == 2){
            long x = 1;
            if(scanf("%ld", &x) != 1){
                break;
            }
            char* ans = Translation(x);
            if(ans[0] != '\0')
                printf("%s\n", ans);
            else
                printf("Under zero\n");
            free(ans);
        }
        else{
            printf("Wrong command!\n");
        }
        printf("> ");

    }

}
```

task2.c

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

int main(){
    int lib = 0;
    char* libs[] = {"./src/lib/first/libFirst.so", "./src/lib/second/libSec.so"};

    float (*Derivative)(float A, float deltaX);
```

```
char* (*Translation)(long x);

void* dlHandle = dlopen(libs[lib], RTLD_LAZY);
if (dlHandle == NULL){
   perror(dlerror());
   exit(-1);
}

int command;
printf("> ");
while (scanf("%d", &command) == 1) {
   if (command == 1) {
      float A, deltaX;
      if (scanf("%f %f", &A, &deltaX) != 2) {
         break;
      }
      Derivative = dlsym(dlHandle, "Derivative");
      if (Derivative == NULL){
         perror(dlerror());
         exit(-1);
      }
      printf("%f\n", Derivative(A, deltaX));
   } else if (command == 2) {
      long x;
      if (scanf("%ld", &x) != 1) {
         break;
      }
      Translation = dlsym(dlHandle, "Translation");
      if (Translation == NULL){
         perror(dlerror());
         exit(-1);
      }
      char* ans = Translation(x);
      if(ans[0] != '\0')
         printf("%s\n", ans);
      else
         printf("Under zero\n");
      free(ans);
   } else if (command == 0){
      if (dlclose(dlHandle) != 0){
         perror(dlerror());
         exit(-1);
      }
      lib = (lib == 0 ? 1 : 0);
      dlHandle = dlopen(libs[lib], RTLD_LAZY);
      if (dlHandle == NULL){
         perror(dlerror());
         exit(-1);
      }
      printf("Changed to another library\n");
```

```
        }else{
            printf("Wrong command!\n");
        }
        printf("> ");
    }

    if (dlclose(dlHandle) != 0){
        perror(dlerror());
        exit(-1);
    }

    return 0;
}
```

deriv.c

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

deriv.c

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

char* Translation(long x){
    int size = sizeof(x) * 8;
    char* str = (char*)malloc(sizeof(char) * (size+1));
    str[0] = '\0';
    if(x < 0){
        return str;
    }
    for(int i = size - 1; i >= 0; i--){
        str[size - i - 1] = ((x >> i) & 1) + '0';
    }

    return str;
}
```

deriv.c

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

deriv.c

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

```
char* Translation(long x){
    int size = sizeof(x) * 8;
    char* str = (char*)malloc(sizeof(char) * (size+1));
    str[0] = '\0';
    if(x < 0)
    {
        return str;
    }
    for(int i = 0; i <= size; i++){
        str[i] = '0';
    }
    int tmp = x;
    int length = 0;
    while(tmp > 0){
        tmp /= 3;
        length++;
    }
    tmp = x;
    for(int i = size; i > size - length; i--){
        str[i] = (tmp % 3) + '0';
        tmp /= 3;
    }

    return str;
}
```

makefile

```
CC=gcc
CFLAGS=-Wall -std=c99
LIBF=src/lib/first
LIBS=src/lib/second

all: task1 task2 $(LIBF)/libFirst.so $(LIBS)/libSec.so

task1: $(LIBF)/deriv.c $(LIBF)/translation.c $(LIBF)/first.h src/task1.c
    $(CC) $(CFLAGS) $(LIBF)/deriv.c $(LIBF)/translation.c src/task1.c -o task1 -lm

task2: src/task2.c $(LIBF)/libFirst.so $(LIBS)/libSec.so
    $(CC) $(CFLAGS) src/task2.c -ldl -o task2 -lm

$(LIBF)/libFirst.so: $(LIBF)/deriv.c $(LIBF)/translation.c
    $(CC) $(CFLAGS) -shared -fPIC $(LIBF)/deriv.c $(LIBF)/translation.c -
o $(LIBF)/libFirst.so -lm

$(LIBS)/libSec.so: $(LIBS)/deriv.c $(LIBS)/translation.c
    $(CC) $(CFLAGS) -shared -fPIC $(LIBS)/deriv.c $(LIBS)/translation.c -o $(LIBS)/libSec.so -
lm

clean:
    rm -f task1 task2 $(LIBF)/libFirst.so $(LIBS)/libSec.so
```

Пример работы программы.

```
pablo$ make
gcc -Wall -std=c99 src/lib/first/deriv.c src/lib/first/translation.c src/task1.c -o task1 -lm
gcc -Wall -std=c99 -shared -fPIC src/lib/first/deriv.c src/lib/first/translation.c -o
src/lib/first/libFirst.so -lm
gcc -Wall -std=c99 -shared -fPIC src/lib/second/deriv.c src/lib/second/translation.c -o
src/lib/second/libSec.so -lm
gcc -Wall -std=c99 src/task2.c -ldl -o task2 -lm
pablo$ cat tests/test1
1 10 0.001
2 15
pablo$ ./task1 < tests/test1
> 0.544667
> 00000000000000000000000000000000000000000000000000000000000001111
pablo$ cat tests/test2
1 10 0.001
2 15
0
1 10 0.001
2 15
pablo$ ./task2 < tests/test2
> 0.544667
> 00000000000000000000000000000000000000000000000000000000000001111
> Changed to another library
> 0.544250
> 00000000000000000000000000000000000000000000000000000000000000120
pablo$ ./task2
> 1 5 0.001
0.958711
> 0
Changed to another library
> 2 1
00000000000000000000000000000000000000000000000000000000000000001
```

Выводы.

Выполнив данную лабораторную работу, я научился создавать динамические библиотеки и подключать их и во время линковки, и во время выполнения программы. Динамические библиотеки очень полезны, так как они могут не подгружаться, если функции, которые они содержат, не используются в программе. Также, если изменить код функции в динамической библиотеке, то перекомпилировать надо будет только ее, а не все файлы, в которых она используется, в отличие от статической библиотеки.