Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №4 по курсу

«Операционные системы»

Группа: М8О-216БВ-24

Студент: Настина М.О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.12.24

Москва, 2024

Постановка задачи

Вариант 13.

Требуется создать динамические библиотеки, которые реализуют функции: расчет производной функции cos(x) в точке a с приращением dx и подсчет площади плоской геометрической фигуры по двум сторонам.

Общий метод и алгоритм решения

Использованные системные вызовы:

● ssize_t write(int fd, const void *buf, size_t count); – вывод данных в файловый дескриптор (в данном случае в стандартный поток вывода 1).

● ssize_t read(int fd, void *buf, size_t count); – чтение данных из файлового дескриптора (в данном случае из стандартного потока ввода 0).

● void _exit(int status); – немедленное завершение процесса с указанным статусом.

● void *dlopen(const char *filename, int flags); – динамическая загрузка библиотеки.

● void *dlsym(void *handle, const char *symbol); – получение адреса символа из динамически загруженной библиотеки.

● int dlclose(void *handle); – закрытие динамической библиотеки.

● int dlerror(void); – получение строки с описанием последней ошибки динамического связывания (используется в сочетании с write, но сама dlerror не является системным вызовом).

Программа представляет собой бесконечный цикл, который ждет команд пользователя. В зависимости от команды, она либо переключает активную реализацию библиотеки, либо вычисляет производную косинуса или площадь, используя текущую загруженную библиотеку.

Код программы

dynamic_test.c:

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

#define BUF 128

typedef float (*der_f)(float, float);
typedef float (*area_f)(float, float);

void print_float(float x) {
    char buf[64];
    int len = 0;

    if (x < 0) {
        buf[len++] = '-';
        x = -x;
    }

    int ip = (int)x;
    float fp = x - ip;

    char tmp[16];
    int tlen = 0;
    if (ip == 0) tmp[tlen++] = '0';
    while (ip > 0) {
        tmp[tlen++] = '0' + (ip % 10);
```

```
        ip /= 10;
    }
    while (tlen > 0) buf[len++] = tmp[--tlen];

    buf[len++] = '.';

    for (int i = 0; i < 3; i++) {
        fp *= 10;
        int d = (int)fp;
        buf[len++] = '0' + d;
        fp -= d;
    }

    buf[len++] = '\n';
    write(1, buf, len);
}

int main() {
    void *lib = NULL;
    der_f der = NULL;
    area_f ar = NULL;
    int impl = 1;

    void load_lib(int num) {
        if (lib) dlclose(lib);

        const char *path = (num == 1) ? "impl1/libimpl1.so" : "impl2/libimpl2.so";
        lib = dlopen(path, RTLD_LAZY);
        if (!lib) {
            write(1, dlerror(), strlen(dlerror()));
```

```
        _exit(1);
    }

    der = (der_f)dlsym(lib, "cos_derivative");
    ar  = (area_f)dlsym(lib, "area");
    if (!der || !ar) {
        write(1, "Symbol not found\n", 17);
        _exit(1);
    }
}

load_lib(impl);

char buf[BUF];
while (1) {
    write(1, "> ", 2);
    int n = read(0, buf, BUF - 1);
    if (n <= 0) break;
    buf[n] = 0;

    char *cmd = strtok(buf, " \n");
    if (!cmd) continue;

    if (cmd[0] == '0') {
        impl = 3 - impl;
        load_lib(impl);
        write(1, "switched\n", 9);
    } else if (cmd[0] == '1') {
        char *s_a = strtok(NULL, " \n");
        char *s_dx = strtok(NULL, " \n");
```

```
            if (!s_a || !s_dx) continue;
            float a = strtof(s_a, NULL);
            float dx = strtof(s_dx, NULL);
            print_float(der(a, dx));
        } else if (cmd[0] == '2') {
            char *s_a = strtok(NULL, " \n");
            char *s_b = strtok(NULL, " \n");
            if (!s_a || !s_b) continue;
            float a = strtof(s_a, NULL);
            float b = strtof(s_b, NULL);
            print_float(ar(a, b));
        }
    }

    if (lib) dlclose(lib);
    return 0;
}
```

statik_test.c:

```
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include "contract.h"

#define BUF 128

void print_float(float x) {
    char buf[64];
    int len = 0;

    if (x < 0) {
```

```
    buf[len++] = '-';
    x = -x;
}

int ip = (int)x;
float fp = x - ip;

char tmp[16];
int tlen = 0;

if (ip == 0) {
    tmp[tlen++] = '0';
} else {
    while (ip > 0) {
        tmp[tlen++] = '0' + (ip % 10);
        ip /= 10;
    }
}

while (tlen > 0)
    buf[len++] = tmp[--tlen];

buf[len++] = '.';

for (int i = 0; i < 3; i++) {
    fp *= 10;
    int d = (int)fp;
    buf[len++] = '0' + d;
    fp -= d;
}
```

```
    buf[len++] = '\n';
    write(1, buf, len);
}

int main() {
    char buf[BUF];
    write(1, "1 a dx | 2 a b\n", 14);

    int n = read(0, buf, BUF - 1);
    buf[n] = 0;

    char *cmd = strtok(buf, " ");
    if (!cmd) return 0;

    if (cmd[0] == '1') {
        float a = strtof(strtok(NULL, " "), NULL);
        float dx = strtof(strtok(NULL, " "), NULL);
        float r = cos_derivative(a, dx);
        print_float(r);
    } else if (cmd[0] == '2') {
        float a = strtof(strtok(NULL, " "), NULL);
        float b = strtof(strtok(NULL, " "), NULL);
        float r = area(a, b);
        print_float(r);
    }
    return 0;
}
```

libimpl1.c:

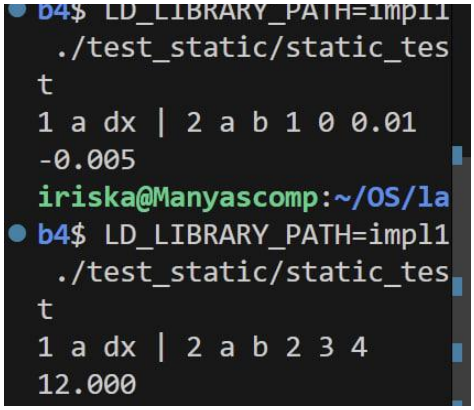
```
#include <math.h>
#include "contract.h"

float cos_derivative(float a, float dx) {
    return (cosf(a + dx) - cosf(a)) / dx;
}

float area(float a, float b) {
    return a * b;
}
```

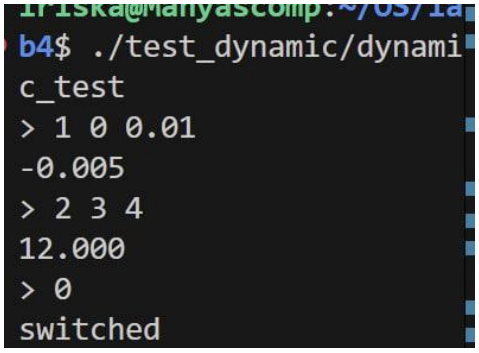
libimpl2.c:

```
#include <math.h>
#include "contract.h"

float cos_derivative(float a, float dx) {
    return (cosf(a + dx) - cosf(a - dx)) / (2.0f * dx);
}

float area(float a, float b) {
    return a * b / 2.0f;
}
```

Протокол работы программы

Тестирование:





Strace для динамического теста:

execve("./dynamic_test", ["./dynamic_test"], 0x7ffc...)

brk(NULL) = 0x55a1a2c6c000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8b7b7f6000

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

openat(AT_FDCWD, "impl1/libimpl1.so", O_RDONLY|O_CLOEXEC) = 3

write(1, "> ", 2) = 2

read(0, "0\n", 128) = 2

close(3) = 0

openat(AT_FDCWD, "impl2/libimpl2.so", O_RDONLY|O_CLOEXEC) = 3

write(1, "switched\n", 9) = 9

write(1, "> ", 2) = 2

read(0, "1 0 0.01\n", 128) = 9

write(1, "0.000\n", 6) = 6

write(1, "> ", 2) = 2

read(0, "0\n", 128) = 2

close(3) = 0

openat(AT_FDCWD, "impl1/libimpl1.so", O_RDONLY|O_CLOEXEC) = 3

write(1, "switched\n", 9) = 9

write(1, "> ", 2) = 2

read(0, "2 1 2\n", 128) = 6

write(1, "2.000\n", 6) = 6

write(1, "> ", 2) = 2

read(0, "", 128) = 0

close(3) = 0

exit_group(0) = ?

Strace для статического теста:

execve("./static_test", ["./static_test"], 0x7ffc...)

brk(NULL) = 0x55a1a2c6c000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8b7b7f6000

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|O_CLOEXEC) = 3

close(3) = 0

write(1, "1 a dx | 2 a b\n", 14) = 14

read(0, "1 0 0.01\n", 128) = 9

write(1, "0.000\n", 6) = 6

exit_group(0) = ?

Вывод

В ходе лабораторной работы были успешно исследованы и реализованы два подхода к компоновке программ. Статическая линковка создает самодостаточный исполняемый файл с включенным кодом библиотек, что увеличивает его размер, но обеспечивает независимость. Динамическая линковка продемонстрировала ключевые преимущества: гибкость "горячей замены" реализаций, модульность архитектуры и экономию ресурсов через разделяемые библиотеки. Практическое применение API динамической загрузки (dlopen, dlsym, dlclose) позволило создать систему плагинов, где две альтернативные математические реализации могут переключаться без перезапуска программы.