ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 5 по курсу

по курсу «Операционные системы»

Группа: М8о-207Б-18

Студент:

Тояков Артем Олегович

Преподаватель:

Миронов Евгений Сергеевич

Оценка:

Дата:

Москва, 2020

Оглавление:

ПОСТАНОВКА ЗАДАЧИ

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 конечном 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.

Вариант 4: BST, int

СТРУКТУРА ПРОГРАММЫ

В данной работе в моя программа состоит из двух файлов:

1. maindnm.c
2. mainstt.c
3. BST.c
4. BST.h
5. Makefile

ОПИСАНИЕ ПРОГРАММЫ

По способу компоновки библиотеки подразделяют на *архивы* (статические библиотеки, static libraries) и совместно используемые (динамические библиотеки, shared libraries). Для создания динамической в Makefile при создании динамической библиотеки используется параметр -shared. Также при сборке объектного файла для нашей структуры указан параметр -fpic(позиционно независимый код) который используется в динамических библиотеках. Также хотелось бы отметить что при сборке исполняемого файла

с использованием динамической библиотеки, при динамической компоновке в Makefile стоит указать -L. -ldeque(для линковщика при записи в объектный файл). Добавляет текущий каталог в просматриваемый линковщиком так как библиотеки располагаются в линукс либо в каталоге /lib или /usr/lib. -ldeque собственно название библиотеки. А соответственно -Wl,-rpath это параметры для поиска во время выполнения, благодаря которым он ищет библиотеку в текущем каталоге при создании объектного файла. Для этого используется опция компилятора -Wl,option,optargs,... Расшифровываю: передать линковщику (-Wl) опцию option с аргументами optargs. В нашем случае мы передаем линковщику опцию -rpath с аргументом . (точка, текущий каталог). Также вместо -Wl, -rpath можно использовать переменную окружения LD_LIBRARY_PATH. Также хотелось бы отметить, что при реализации maindnm.c использовались системные вызовы:

При вызове dlopen происходит автоматическое разрешение зависимостей между библиотеками. Это значит, что если некая библиотека использует другую библиотеку, функция загрузит и ее. dlopen возвращает дескриптор, используемый для дальнейшей работы с библиотекой. Прототип функции выглядит так: void *dlopen(const char *file, int mode);

По дескриптору с помощью функции dlsym находятся адреса символов библиотеки. Функция принимает в качестве параметра дескриптор и строковое

имя символа и возвращает искомый адрес: void *dlsym(void *restrict handle, const char *restrict name);

ЛИСТИНГ ПРОГРАММЫ

//maindnm.c

```
#include <dlfcn.h>
#include "BST.h"
#define Error_dlsym 5

void help() {
    printf("Add: press a \n");
    printf("Delete element: press d \n");
    printf("Print: press p \n");
    printf("Help: press h \n");
    printf("Exit: press e \n");
}

void* get(void *libHandle, char* name) {
    void* temp = dlsym(libHandle, name);
    if (temp == NULL) {
        fprintf(stderr, "%s\n", dlerror());
        exit(Error_dlsym);
    }
    return temp;
}

int main() {
    void *libHandle;
    libHandle = dlopen("./libBST.so", RTLD_LAZY);
    if (!libHandle) {
        fprintf(stderr, "%s\n", dlerror());
        exit(1);
    }
    node*(*create)(node *my_tree) = get(libHandle, "create");
    node*(*add)(node *my_tree, int inf) = get(libHandle, "add");
    node*(*delete)(node *my_tree, int inf) = get(libHandle, "delete");
    void(*print)(node *my_tree) = get(libHandle, "print");
    void(*tree_delete)(node *my_tree) = get(libHandle, "tree_delete");
    node *my_tree = NULL;
    int inf;
    char c;
    printf("Please, enter your firt info ");
    scanf("%d", &inf);
    my_tree = (*create)(my_tree);
    help();
    while (true) {
        scanf("%c", &c);
        switch(c) {
            case 'a':
                printf("Enter your element ");
                scanf("%d", &inf);
                (*add)(my_tree, inf);
            break;
            case 'd':
                printf("Enter your element ");
                scanf("%d", &inf);
                (*delete)(my_tree, inf);
            break;
            case 'p':
```

```
                (*print)(my_tree);
            break;
            case 'h':
                help();
            break;
            case 'e':
                (*tree_delete)(my_tree);
                my_tree = NULL;
                dlclose(libHandle);
                return 0;
            break;
        }
    }
    return 0;
}
```

//Makefile

```
cc = gcc
FLAGS = -g -Wall -Werror -Wextra

all: stt dnm

stt: libBST.so mainstt.o
	$(CC) $(FLAGS) -o stt mainstt.o -L. -lBST -Wl,-rpath,.

dnm: maindnm.o libBST.so
	$(CC) $(FLAGS) -rdynamic -o dnm maindnm.o -ldl

mainstt.o: mainstt.c
	$(CC) -c $(FLAGS) mainstt.c

maindnm.o: maindnm.c
	$(CC) -c $(FLAGS) maindnm.c

libBST.so: BST.o
	$(CC) $(FLAGS) -shared -o libBST.so BST.o

BST.o: BST.c
	$(CC) -c -fPIC $(FLAGS) BST.c

clean:
	rm *.o stt dnm *.so
```

РЕЗУЛЬТАТ РАБОТЫ

```
ARTOY@ARTOY:~/DESKTOP/LABS/3 SEM/OS/LAB5$ MAKE
CC -C -FPIC -G -WALL -WERROR -WEXTRA BST.C
CC -G -WALL -WERROR -WEXTRA -SHARED -O LIBBST.SO BST.O
CC -C -G -WALL -WERROR -WEXTRA MAINSTT.C
CC -G -WALL -WERROR -WEXTRA -O STT MAINSTT.O -L. -LBST -WL,-RPATH,.
CC -C -G -WALL -WERROR -WEXTRA MAINDNM.C
CC -G -WALL -WERROR -WEXTRA -RDYNAMIC -O DNM MAINDNM.O -LDL
ARTOY@ARTOY:~/DESKTOP/LABS/3 SEM/OS/LAB5$ ./DNM
PLEASE, ENTER YOUR FIRT INFO 1
ADD: PRESS A
DELETE ELEMENT: PRESS D
```

```
PRINT: PRESS P
HELP: PRESS H
EXIT: PRESS E
A
ENTER YOUR ELEMENT -5
P
1
-5
NULL
NULL
NULL
A
ENTER YOUR ELEMENT 10
A
ENTER YOUR ELEMENT 48
A
ENTER YOUR ELEMENT -10
A
ENTER YOUR ELEMENT -7
P
1
-5
-10
NULL
-7
NULL
NULL
NULL
10
NULL
48
NULL
NULL
D
ENTER YOUR ELEMENT 10
D
ENTER YOUR ELEMENT -5
P
1
-10
NULL
-7
NULL
NULL
48
NULL
```

```
NULL
E
```

ВЫВОД

В чем же отличие подключения динамической библиотеки на стадии линковки и во время исполнения? На стадии линковки адреса данных функций подгружаются в объектный файл и при запуске программы мы вызываем данные функции, если же во время исполнения, то мы вручную с помощью системных вызовов dlopen и тд. получаем и определяем адреса во время исполнения программы. Основными плюсами использования динамических библиотек является экономия памяти и возможность использования данной библиотеки несколькими программами и соответственно при изменении в библиотеке чего-либо в этих программах эти изменения будут отображены. Экономия памяти происходит за счет того, что в исполняемый файл подгружаются адреса на наши функции в динамической библиотеке.