****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5 по курсу**

**«Операционные системы»**

Группа: М80-207Б-18

Студент: Ильминский Никита Сергеевич

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2020.

***Постановка задачи***Вариант 13:  
Создать динамическую библиотеку, которая реализует функционал работы со структурой данных вектор с элементами вещественного 64-битного типа.

Продемонстрировать использование библиотеки двумя 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.

Также необходимо провести анализ между обоими типами использования библиотеки.

***Алгоритм и описание программы***

Для выполнения данной задачи сначала необходимо, очевидно, написать саму библиотеку; весь код разделен на два файла vector.c (реализация) и vector.h (заголовочный файл). Для создания файла с библиотекой необходимо компилировать объектный файл vector.o (здесь будет использован флаг -fpic, дающий gcc знать, что необходимо компилировать Position Independent Code, то есть код, не зависящий от данного адреса расположения памяти; это важно для создания динамической библиотеки). После этого при помощи gcc и флага -shared можно создать сам файл библиотеки lib\*.so (название файла пользовательской написанной библиотеки должно начинаться с ‘lib’). Теперь, когда библиотека готова к использованию, можно писать код для двух реализаций использования библиотеки.

В файле linking.c стандартно опишем интерфейс работы с вектором, предварительно включив vector.h в код программы. Тоже самое сделаем в файле loading.c, однако здесь будем использовать также библиотеку dl (с хэдэром dlfcn.h), позволяющую при помощи функций dlopen (открытие библиотеки, RTLD\_LAZY – флаг, позволяющий перемещать части кода только по необходимости), dlsym (получение доступа к части кода библиотеки, например к функции, по дескриптору) и dlclose (закрытие библиотеки) работать с libvector.so; важно заметить, что необходимые функции библиотеки нельзя вызвать напрямую, их необходимо сначала объявить в качестве указателей на функции и получить доступ при помощи dlsym.

Завершающий этап – компиляция. Создадим Makefile, пропишем необходимые цели; файл link будет компилироваться при помощи ключей -L. -lvector (сам процесс линковки библиотеки), файл load – при помощи ключей -rdynamic и -ldl (экспорт символов из пользовательской библиотеки, подключение библиотеки dl). Созданные исполняемые файлы готовы к работе.

***Листинг***

*linking.c:*

#include "vector.h"

#include <string.h>

void menu() {

printf("Double vector. Type in:\n");

printf("'menu' for menu;\n");

printf("'push' to pushback;\n");

printf("'ins' to insert;\n");

printf("'er' to erase;\n");

printf("'del' to erase at idx;\n");

printf("'destr' to destroy;\n");

printf("'at' to get element at idx;\n");

printf("'size' for size;\n");

printf("'print' to print;\n");

printf("'quit' to quit\n\n");

}

int main() {

vector v;

vInit(&v);

menu();

char ch[16];

double val;

size\_t idx;

while (scanf("%15s", ch) == 1) {

if (strcmp(ch, "menu") == 0) {

menu();

continue;

}

if (strcmp(ch, "push") == 0) {

printf("Enter the element value: ");

scanf("%lf", &val);

vPushBack(&v, val);

continue;

}

if (strcmp(ch, "ins") == 0) {

printf("Enter the element value and idx: ");

scanf("%lf %ld", &val, &idx);

vInsert(&v, val, idx);

continue;

}

if (strcmp(ch, "er") == 0) {

vErase(&v);

continue;

}

if (strcmp(ch, "del") == 0) {

printf("Enter the idx: ");

scanf("%ld", &idx);

vEraseAt(&v, idx);

continue;

}

if (strcmp(ch, "destr") == 0) {

vDestroy(&v);

continue;

}

if (strcmp(ch, "at") == 0) {

printf("Enter the idx: ");

scanf("%ld", &idx);

val = vAt(&v, idx);

printf("%f\n", val);

continue;

}

if (strcmp(ch, "size") == 0) {

printf("%ld\n", vSize(&v));

continue;

}

if (strcmp(ch, "print") == 0) {

vPrint(&v);

continue;

}

if (strcmp(ch, "quit") == 0) {

break;

}

printf("Invalid option!\n");

}

return 0;

}

*loading.c:*

#include "vector.h"

#include <string.h>

#include <dlfcn.h>

#include <stdlib.h>

void\* getFunc(void\* handle, char\* fName) {

void\* f = dlsym(handle, fName);

if (f == NULL) {

printf("%s\n", dlerror());

exit(1);

}

return f;

}

void menu() {

printf("Double vector. Type in:\n");

printf("'menu' for menu;\n");

printf("'push' to pushback;\n");

printf("'ins' to insert;\n");

printf("'er' to erase;\n");

printf("'del' to erase at idx;\n");

printf("'destr' to destroy;\n");

printf("'at' to get element at idx;\n");

printf("'size' for size;\n");

printf("'print' to print;\n");

printf("'quit' to quit\n\n");

}

int main() {

void\* libHandle = dlopen("./libvector.so", RTLD\_LAZY);

if (!libHandle) {

printf("%s\n", dlerror());

exit(1);

}

vector v;

void (\*init)(vector\*) = getFunc(libHandle, "vInit");

size\_t (\*size)(const vector\*) = getFunc(libHandle, "vSize");

double (\*at)(const vector\*, size\_t) = getFunc(libHandle, "vAt");

void (\*pushback)(vector\*, double) = getFunc(libHandle, "vPushBack");

void (\*insert)(vector\*, double, size\_t) = getFunc(libHandle, "vInsert");

void (\*erase)(vector\*) = getFunc(libHandle, "vErase");

void (\*delete)(vector\*, size\_t) = getFunc(libHandle, "vEraseAt");

void (\*destroy)(vector\*) = getFunc(libHandle, "vDestroy");

void (\*print)(const vector\*) = getFunc(libHandle, "vPrint");

(\*init)(&v);

menu();

char ch[16];

double val;

size\_t idx;

while (scanf("%15s", ch) == 1) {

if (strcmp(ch, "menu") == 0) {

menu();

continue;

}

if (strcmp(ch, "push") == 0) {

printf("Enter the element value: ");

scanf("%lf", &val);

(\*pushback)(&v, val);

continue;

}

if (strcmp(ch, "ins") == 0) {

printf("Enter the element value and idx: ");

scanf("%lf %ld", &val, &idx);

(\*insert)(&v, val, idx);

continue;

}

if (strcmp(ch, "er") == 0) {

(\*erase)(&v);

continue;

}

if (strcmp(ch, "del") == 0) {

printf("Enter the idx: ");

scanf("%ld", &idx);

(\*delete)(&v, idx);

continue;

}

if (strcmp(ch, "destr") == 0) {

(\*destroy)(&v);

continue;

}

if (strcmp(ch, "at") == 0) {

printf("Enter the idx: ");

scanf("%ld", &idx);

val = (\*at)(&v, idx);

printf("%f\n", val);

continue;

}

if (strcmp(ch, "size") == 0) {

printf("%ld\n", (\*size)(&v));

continue;

}

if (strcmp(ch, "print") == 0) {

(\*print)(&v);

continue;

}

if (strcmp(ch, "quit") == 0) {

break;

}

printf("Invalid option!\n");

}

dlclose(libHandle);

return 0;

}

*Makefile:*

CC = gcc

CCKEYS = -Wall -Werror -g

all: link load

link: linking.c libvector.so

$(CC) $(CCKEYS) linking.c -o link -L. -lvector

load: loading.c libvector.so

$(CC) $(CCKEYS) -rdynamic loading.c -o load -ldl

libvector.so: vector.o

$(CC) $(CCKEYS) -shared -o libvector.so vector.o

vector.o: vector.c vector.h

$(CC) $(CCKEYS) vector.c -c -fpic

cleanL:

rm -rf libvector.so vector.o

cleanE:

rm -rf link load

***Демонстрация работы программ***

Прежде чем запускать программы, проведем их компиляцию при помощи утилиты make и заранее написанного Makefile’а:

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ls

Makefile linking.c loading.c vector.c vector.h

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# make link

gcc -Wall -Werror -g vector.c -c -fpic

gcc -Wall -Werror -g -shared -o libvector.so vector.o

gcc -Wall -Werror -g linking.c -o link -L. -lvector

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ls

Makefile libvector.so link linking.c loading.c vector.c vector.h vector.o

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# make load

gcc -Wall -Werror -g -rdynamic loading.c -o load -ldl

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ls

Makefile libvector.so link linking.c load loading.c vector.c vector.h vector.o

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5#

Производим запуск программ:

*link:*

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ./link

./link: error while loading shared libraries: libvector.so: cannot open shared object file: No such file or directory

Запуск программы, использующей линковку библиотеки, не был произведен, так как система не может найти данную библиотеку. Для этого нужно добавить рабочую директорию с библиотекой в список известных расположений; это возможно сделать, изменив файл /etc/ld.so.conf (что будет влиять на всю систему в целом) или используя специальную переменную среды LD\_LIBRARY\_PATH, в которой перечисляются все каталоги, содержащие пользовательские динамические библиотеки (однако это решение временное). Применим второй способ:

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# LD\_LIBRARY\_PATH=./

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# export LD\_LIBRARY\_PATH

Теперь программа работает:

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ./link

Double vector. Type in:

'menu' for menu;

'push' to pushback;

'ins' to insert;

'er' to erase;

'del' to erase at idx;

'destr' to destroy;

'at' to get element at idx;

'size' for size;

'print' to print;

'quit' to quit

push

Enter the element value: 1

push

Enter the element value: 2

push

Enter the element value: 600000

size

3

print

1.000000 2.000000 600000.000000

ins

Enter the element value and idx: 3.14 2

print

1.000000 2.000000 3.140000 600000.000000

at

Enter the idx: 2

3.140000

er

print

1.000000 2.000000 3.140000

del

Enter the idx: 0

print

2.000000 3.140000

destr

print

size

0

quit

Программа, использующая подгрузку библиотеки, запускается без излишних проблем:

*load:*

root@DESKTOP-5B8SU4Q:~/labs/3sem/os/5# ./load

Double vector. Type in:

'menu' for menu;

'push' to pushback;

'ins' to insert;

'er' to erase;

'del' to erase at idx;

'destr' to destroy;

'at' to get element at idx;

'size' for size;

'print' to print;

'quit' to quit

push

Enter the element value: 1

push

Enter the element value: 2

push

Enter the element value: 600000

size

3

print

1.000000 2.000000 600000.000000

ins

Enter the element value and idx: 3.1416 1

print

1.000000 3.141600 2.000000 600000.000000

at

Enter the idx: 1

3.141600

er

print

1.000000 3.141600 2.000000

del

Enter the idx: 0

print

3.141600 2.000000

destr

print

size

0

quit

***Вывод***

По результатам работы двух программ видно, что они работают одинаково правильно. В чем же различие между двумя способами использования динамических библиотек?  
 1. при использовании линковки, в отличии от способа подгрузки, системе необходимо дополнительно (не в самом коде программы, а через командную оболочку) передать данные о расположении библиотеки для возможности ее использования, о чем всегда нужно помнить программисту; при этом решение с использованием переменной среды LD\_LIBRARY\_PATH временно, а изменение файла системы /etc/ld.so.conf не всегда безопасно и доступно лишь пользователям с соответствующими правами. Открытие файла с библиотекой в самой программе (при подгрузке) является более удобным средством, однако сам код программы увеличивается и усложняется ввиду необходимости получения нужных “частей” кода из подгруженной библиотеки.

2. хотя и усложняя код, постепенная подгрузка является более эффективной в том плане, что библиотеку не нужно полностью загружать в память, ведь есть возможность подгружать нужные функции во время выполнения. Таким образом множество одновременно работающих программ способны использовать лишь одну библиотеку, в отличии от способа линковки, когда библиотечный файл копируется для каждой программы. Это, возможно, является очень важным преимуществом для программ, работающих с огромными библиотеками.

Однако, на мой взгляд, использование линковки библиотеки при компиляции является более удобным способом, нежели динамическая подгрузка. Несмотря на все недостатки первого способа, код становится заметно проще и более удобным для чтения, что очень важно для разработчика.

***Список литературы***

https://developer.ibm.com/tutorials/l-dynamic-libraries/

https://hackernoon.com/static-and-dynamic-libraries-fe5d23daffe3

https://medium.com/@Cu7ious/how-to-use-dynamic-libraries-in-c-46a0f9b98270

http://www.firststeps.ru/linux/r.php?7

http://www.firststeps.ru/linux/r.php?8