Лабораторная работа № 5
по курсу "Операционные системы":

Выполнил студент группы 08-208 МАИ *Куликов Алексей.*

Цель работы

Целью является приобретение практических навыков в:

1. Создание динамических библиотек
2. Создание программ, которые используют функции динамических библиотек

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 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.

Провести анализ между обоими типами использования библиотеки.

В качестве конкретного варианта задания предлагается создание динамиеской библиотеки реализующая интерфейс взаимодействия с вектором 32-битных целочисленных переменных. (вариант 6).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Описание программы

Программа состоит из 3-х частей:

- модуль динамической библиотеки, реализующей тип данных вектор и определенные для него операции;

- модуль, взаимодействующий с динамической библиотекой с подгрузкой во время инициализации процесса;
- модуль, взаимодействующий с динамической библиотекой во время работы программы с помощью библиотеки `dlfcn.h`.

В модуле библиотеки реализован тип данных вектор и стандартные операции над ним.

Модуль `init_time` взаимодействует с библиотекой после линковки на этапе инициацлизации процесса, исполняющего программу. Внутри просто осуществляются всевозможные операции над вектором для тестирования.

Внутри модуля `full_dynamic` осуществляются линковка во время исполнения программы посредством библиотеки `dlfcn.h`.

Внутри файла глобальным образом объявляются указатели на функции с сигнатурами, соответсвующими функциям в библиотеке. Далее в функции `dl_init` получаем доступ к динамической библиотеке с помощью функции `dlopen`. Теперь можно получить доступ к функциям дмнамической библиотеки по их именам. Это осуществляется функцией `dlsym`.

После можно использовать функции динамической библиотеки. Далее так же осуществляются всевозможные операции над вектором для тестирования.

После работы с фунциями динамической библиотеки в функции `dl_init` закрываем доступ нашей программы к динамической библиотеке с помощью `dlclose`.

Листинг

`vector.h`

```
#ifndef VECTOR_H
#define VECTOR_H

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

const int coef = 2;

typedef struct {
    int *data;
    size_t capacity;
    size_t size;
} vector;

int vector_init(vector *vec, size_t cap);
int vector_resize(vector *vec, size_t new_cap);
void vector_push_back(vector *vec, int val);
void vector_pop_back(vector *vec);
size_t vector_size(const vector *vec);
size_t vector_capacity(const vector *vec);
int vector_insert(vector *vec, size_t ind, int val);
int vector_erase(vector *vec, size_t ind);
int vector_is_empty(const vector *vec);
int vector_put(vector *vec, size_t ind, int val);
int vector_fetch(const vector *vec, size_t ind);
void vector_destroy(vector *vec);
#endif
```

vector.c

```
#include "vector.h"
int vector_init(vector *vec, size_t cap){
    vec->data = (int *)malloc(sizeof(int) * cap);
    if (!vec->data)
        return 0;
    vec->capacity = cap;
    vec->size = 0;
    return 1;
}
int vector_resize(vector *vec, size_t new_cap){
    vec->capacity = new_cap;
    vec->data = (int *)realloc(vec->data, vec->capacity * sizeof(int));
    if (!vec->data)
        return 0;
    return 1;
}
int vector_push_back(vector *vec, int val){
    if(vec->size >= vec->capacity)
        if(!vector_resize(vec, vec->capacity * coef))
            return 0;
    vec->data[vec->size] = val;
    ++vec->size;
    return 1;
}
int vector_pop_back(vector *vec){
    --vec->size;
    if (vec->size * coef * coef <= vec->capacity)
        if(!vector_resize(vec, vec->capacity / coef))
            return 0;
}
size_t vector_size(const vector *vec){
    return vec->size;
}
size_t vector_capacity(const vector *vec){
    return vec->capacity;
}
int vector_insert(vector *vec, size_t ind, int val){
    if (ind < 0 || ind > vec->size)
        return 0;
    if(vec->size + 1 > vec->capacity)
        if(!vector_resize(vec, vec->capacity * coef))
            return 0;
    for(size_t i = vec->size; i > ind; --i)
        vec->data[i] = vec->data[i-1];
    vec->data[ind] = val;
    ++vec->size;

    return 1;
}
int vector_erase(vector *vec, size_t ind){
    if (ind < 0 || ind > vec->size)
        return 0;
    for (size_t i = ind + 1; i < vec->size; ++i)
        vec->data[i-1] = vec->data[i];
    --vec->size;
    if (vec->size * coef * coef <= vec->capacity)
        if(!vector_resize(vec, vec->capacity / coef))
            return 0;
    return 1;
}
int vector_is_empty(const vector *vec){
    return vec->size == 0;
}
```

```
int vector_put(vector *vec, size_t ind, int val){
    if(ind < 0 || ind >= vec->size)
        return 0;
    vec->data[ind] = val;
    return 1;
}
int vector_fetch(const vector *vec, size_t ind){
    if (ind < 0 || ind >= vec->size)
        return -1;
    return vec->data[ind];
}
void vector_destroy(vector *vec){
    free(vec->data);
    vec->capacity = 0;
    vec->size = 0;
    vec->data = NULL;
}
```

init_time_main.c

```
#include <stdio.h>

#include "vector.h"

int main(){
    printf("init time linked dynamic library interaction\n");

    vector v;
    printf("vector initialization\n");
    vector_init(&v, 4);

    printf("vector cap: %ld\n", vector_capacity(&v));
    printf("vector size: %ld\n", vector_size(&v));

    printf("push 3 elements\n");
    vector_push_back(&v, 1);
    vector_push_back(&v, 2);
    vector_push_back(&v, 3);

    printf("vector cap: %ld\n", vector_capacity(&v));
    printf("vector size: %ld\n", vector_size(&v));

    printf("push extra 2 elements\n");
    vector_push_back(&v, 2);
    vector_push_back(&v, 3);

    printf("vector cap: %ld\n", vector_capacity(&v));
    printf("vector size: %ld\n", vector_size(&v));

    printf("pop 3 elements\n");
    vector_pop_back(&v);
    vector_pop_back(&v);
    vector_pop_back(&v);

    printf("vector cap: %ld\n", vector_capacity(&v));
    printf("vector size: %ld\n", vector_size(&v));

    printf("vector contains: ");
    size_t size = vector_size(&v);
    for(size_t i = 0; i < size; ++i)
        printf("%d ", vector_fetch(&v, i));
    printf("\n");

    printf("insert 100 on 1th position in vector\n");
    vector_insert(&v, 1, 100);
```

```
    printf("vector contains: ");
    size = vector_size(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", vector_fetch(&v, i));
    printf("\n");

    printf("erase 0th element in vector\n");
    vector_erase(&v, 0);

    printf("vector contains: ");
    size = vector_size(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", vector_fetch(&v, i));
    printf("\n");

    printf("put 111 on 0th position in vector\n");
    vector_put(&v, 0, 111);

    printf("vector contains: ");
    size = vector_size(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", vector_fetch(&v, i));
    printf("\n");

    printf("vector destroying\n");
    vector_destroy(&v);

    return 0;
}
```

full_dynamic_main.c

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

#include "vector.h"

int (*vec_init)(vector *vec, size_t cap);
int (*vec_resize)(vector *vec, size_t new_cap);
int (*vec_push_back)(vector *vec, int val);
int (*vec_pop_back)(vector *vec);
size_t (*vec_size)(const vector *vec);
size_t (*vec_capacity)(const vector *vec);
int (*vec_insert)(vector *vec, size_t ind, int val);
int (*vec_erase)(vector *vec, size_t ind);
int (*vec_is_empty)(const vector *vec);
int (*vec_put)(vector *vec, size_t ind, int val);
int (*vec_fetch)(const vector *vec, size_t ind);
void (*vec_destroy)(vector *vec);

void *dl_handler;

void dl_init(){
    dl_handler = dlopen("libvector.so", RTLD_LAZY);
    if(!dl_handler){
        fprintf(stderr, "ERROR: %s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    vec_init = dlsym(dl_handler, "vector_init");
    vec_resize = dlsym(dl_handler, "vector_resize");
    vec_push_back = dlsym(dl_handler, "vector_push_back");
    vec_pop_back = dlsym(dl_handler, "vector_pop_back");
    vec_size = dlsym(dl_handler, "vector_size");
    vec_capacity = dlsym(dl_handler, "vector_capacity");
```

```
    vec_insert = dlsym(dl_handler, "vector_insert");
    vec_erase = dlsym(dl_handler, "vector_erase");
    vec_is_empty = dlsym(dl_handler, "vector_is_empty");
    vec_put = dlsym(dl_handler, "vector_put");
    vec_fetch = dlsym(dl_handler, "vector_fetch");
    vec_destroy = dlsym(dl_handler, "vector_destroy");
}

void dl_fini(){
    dlclose(dl_handler);
}

int main(){
    printf("fully dynamic linked dynamic library interaction\n");
    dl_init();
    vector v;
    printf("vector initialization\n");
    (*vec_init)(&v, 4);

    printf("vector cap: %ld\n", (*vec_capacity)(&v));
    printf("vector size: %ld\n", (*vec_size)(&v));

    printf("push 3 elements\n");
    (*vec_push_back)(&v, 1);
    (*vec_push_back)(&v, 2);
    (*vec_push_back)(&v, 3);

    printf("vector cap: %ld\n", (*vec_capacity)(&v));
    printf("vector size: %ld\n", (*vec_size)(&v));

    printf("push extra 2 elements\n");
    (*vec_push_back)(&v, 2);
    (*vec_push_back)(&v, 3);

    printf("vector cap: %ld\n", (*vec_capacity)(&v));
    printf("vector size: %ld\n", (*vec_size)(&v));

    printf("pop 3 elements\n");
    (*vec_pop_back)(&v);
    (*vec_pop_back)(&v);
    (*vec_pop_back)(&v);

    printf("vector cap: %ld\n", (*vec_capacity)(&v));
    printf("vector size: %ld\n", (*vec_size)(&v));

    printf("vector contains: ");
    size_t size = (*vec_size)(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", (*vec_fetch)(&v, i));
    printf("\n");

    printf("insert 100 on 1th position in vector\n");
    (*vec_insert)(&v, 1, 100);

    printf("vector contains: ");
    size = (*vec_size)(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", (*vec_fetch)(&v, i));
    printf("\n");

    printf("erase 0th element in vector\n");
    (*vec_erase)(&v, 0);

    printf("vector contains: ");
    size = (*vec_size)(&v);
```

```
    for (size_t i = 0; i < size; ++i)
        printf("%d ", (*vec_fetch)(&v, i));
    printf("\n");

    printf("put 111 on 0th position in vector\n");
    (*vec_put)(&v, 0, 111);

    printf("vector contains: ");
    size = (*vec_size)(&v);
    for (size_t i = 0; i < size; ++i)
        printf("%d ", (*vec_fetch)(&v, i));
    printf("\n");

    printf("vector destroying\n");
    (*vec_destroy)(&v);

    dl_fini();
    return 0;
}
```

Демонстрация работы

```
$ make init_time
$ ./it_main
init time linked dynamic library interaction
vector initialization
vector cap: 4
vector size: 0
push 3 elements
vector cap: 4
vector size: 3
push extra 2 elements
vector cap: 8
vector size: 5
pop 3 elements
vector cap: 4
vector size: 2
vector contains: 1 2
insert 100 on 1th position in vector
vector contains: 1 100 2
erase 0th element in vector
vector contains: 100 2
put 111 on 0th position in vector
vector contains: 111 2
vector destroying

$ make full_dinamic
$ ./fd_main
```

```
fully dynamic linked dynamic library interaction
...
```

Strace

```
...
openat(AT_FDCWD, "/usr/lib/libvector.so", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0
\0\0\3\0>\0\1\0\0\0000\10\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=12560, ...}) = 0
mmap(NULL, 2105416, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE,
3, 0) = 0x7f89acbc2000
mprotect(0x7f89acbc4000, 2093056, PROT_NONE) = 0
mmap(0x7f89acdc3000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1000) = 0x7f89acdc3000
close(3)                                = 0
mprotect(0x7f89acdc3000, 4096, PROT_READ) = 0
munmap(0x7f89ad5ca000, 144477)          = 0
write(1, "vector initialization\n", 22) = 22
...
write(1, "vector destroying\n", 18)     = 18
munmap(0x7f89acbc2000, 2105416)         = 0
...
```

Выводы

Что это такое?

Динамическая библиотека – специальным образом организованный файл, содержащий машинный код.

Суть динамических библиотек состоит в том, что машинный код из них не включается в сам исполняемый файл, а связывается с исполняемым либо на этапе инициализации процесса, либо уже во время исполнения программы.

Зачем оно нужно?

Динамические библиотеки используются по приведенным ниже причинам. Достоинства:

- экономия памяти (один экземпляр библиотеки в ОЗУ в один момент времени для кучи программ);
- модульность (для исправления ошибок и/или обновлений нужно не пересобирать исполняемые файлы, а всего лишь заменть на новые файлы динамических библиотек);

- компактный исполняемый файл.

Недостатки:

- конфликт версий (потребность разных программ в разных версиях библиотеки, в которых, возможно, отличается поведение функций, интерфейсы и т.п)

Если библиотека будет совместно использоваться несколькими исполняемыми файлами, часто имеет смысл сделать ее динамической, чтобы уменьшить размер исполняемых файлов.

Так же имеет смысл выносить редкоиспользуемые функции в динамические библиотеки, чтобы не тратить память в пустую.

С помощью явной компоновки можно сделать систему плагинов.

Если вдруг надо использовать библиотеки на разных языках(что маловероятно), то тут тоже в помощь приходят динамические библиотеки. Главное чтоб компилятор позволял создавать их.

Линковка во время создания процесса (неявная компоновка)

Операционная система во время создания процесса, в котором будет исполняться программа, подгружает (если уже не загружена) библиотеку и линкует объекты исполняемого файла и динамической библиотеки. Далее в программе все работает так, как будто производилась статическая линковка. Т.е. использование объектов в библиотеке так, как будто были в исходных кодах до компиляции. Никаких указателей на функции и т.д. и т.п.

Линковка програмным путем во время выполнения программы (явная компоновка)

Во время создания процесса никаких подгрузок и связываний не происходит. В коде программы явно прописаны действия для загрузки и связывания динамической библиотеки тогда, когда это нужно. Такую услугу предаставляет библиотека `dlfcn.h`, которая в свою очередь, работает через загрузчик динамических библиотек операционной системы. Получение доступа к функциям происходит так:

Просим `dlfcn.h` при помощи `dlopen` открыть нам доступ (подгрузить для нас, если надо) к динамической библиотеке с указанием пути к месту ее расположения. `dlopen` предоставляет нам указатель на область, куда подгружена библиотека. Далее по этому указателю мы с помощью `dlsym`, зная имя функции, получаем указатель на требуемую функцию. Этот указаель мы записываем в переменную-указатель соответствующего типа функции. После этого мы можем свободно оперировать этой функцией разыменовывая указатель и передавая аргументы в функцию.

Когда нам уже не нужны функции данной библиотеки мы просим `dlclose` закрыть нам доступ к этой библиотеке. Далее операционная система смотрит нужна ли еще какому-нибудь процессу данная библиотека. Если не нужна, то выгружает ее из оперативной памяти, иначе оставляет для использования другими процессами.

Особенности

Так как мы создаем разделяемую динамическую библиотеку нужно чтобы происходила генерация позиционно независимого кода на этапе компиляции. Для этого нужно компилировать исходные файлы для библиотеки с ключом `-fPIC`. Без этого программы не смогли бы совместно использовать динамическую библиотеку потому что каждая из них находится в своем адресном пространстве и отсчитывать адреса для переходов, переменных и т.д. следует именно от начала конкретного адресного пространства. Т.е. должна быть относительная адресация.

Так же на этапе линковки следует указать флаг `-shared`.

Простого создания библиотеки не достаточно для того, чтобы использующие ее программы заработали. Нужно так же либо положить ее туда, где системны линковщик ищет динамические библиотеки, либо указать ему где ее искать. Можно положить ее в каталог `/usr/lib`. Так же существуют еще несколько путей. Либо задать значение переменной `D_LIBRARY_PATH` пути, где следует их искать. Пути задаются через знак двоеточия.