ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

ЛАБОРАТОРНАЯ РАБОТА № 3
«РАЗДЕЛЯЕМЫЕ БИБЛИОТЕКИ»

Автор: С.Н. Мамойленко

Новосисбирск - 2013

Оглавление

Цель работы

Укрепить навыки разработки приложений с использованием разделяемых библиотек, понять принципы конфигурирования подсистемы управления динамическими библиотеками.

Теоретическое введение

1. Динамические и статические библиотеки функций. Общие сведения

Отладка программного обеспечения трудоемкий и длительный процесс. Для упрощения этого процесса программное обеспечение обычно делится на логически оконченные блоки – функции, каждый из которых тестируется отдельно и независимо. Кроме того, такой подход позволяет разделить процесс разработки программного обеспечения между несколькими исполнителями.

Функции, отвечающие за выполнение логически связанных действий, объединяются в библиотеки. Например, библиотеки обработки изображений, стандартная библиотека функция языка Си и т.п. Очевидно, что при разработке нового программного обеспечения могут использоваться ранее подготовленные библиотеки функций. Тем самым уменьшается время, требуемое для разработки программного обеспечения.

Библиотеки функций в зависимости от способа их использования в процессе выполнения программного обеспечения можно разделить на два класса – статические и динамические. ***Статические библиотеки функций*** полностью включаются в исполняемый файл и загружаются в оперативную память в процессе загрузки программы на исполнение. При этом, если несколько программ используют одну и ту же библиотеку и эти программы одновременно исполняются на ЭВМ, то в памяти ЭВМ будут находиться несколько одинаковых копий функций. ***Динамические библиотеки функций*** загружаются в память ЭВМ по мере необходимости и могут разделяться (одновременно использоваться) несколькими выполняющимися программами. Такой способ использования библиотек функций позволяет избежать нерациональное использование памяти ЭВМ.

Работа с динамическими библиотеками требует выполнения двух операций: загрузки библиотеки в память ЭВМ и связывание функций (т.е. определения их адресов в памяти). В зависимости от способа реализации этих операций работу с динамическими библиотеками можно разделить на два типа: ***динамическая компоновка*** и ***динамическая загрузка***. В первом случае операции загрузки и связывания выполняются «автоматически» (т.е. за их выполнение несет ответственность системное программное обеспечение), а во втором случае эти операции закладываются в программу в процессе её разработки (ответственность за выполнение операция несет разработчик программного обеспечения).

2. Динамическая загрузка функций

Динамическая загрузка библиотек осуществляется с помощью специальных системных вызовов, которые размещены в библиотеке функций `dl` (см. рисунок **1**).

```
#include <dlfcn.h>

void *dlopen(const char *filename, int flag);
char *dlerror(void);
void *dlsym(void *handle, const char *symbol);
int dlclose(void *handle);
```

Рисунок **1** – Функции динамический загрузки библиотек

Функция `dlopen` загружает динамическую библиотеку в память ЭВМ. В результате функция возвращает указатель на дескриптор открытой библиотеки (специальная структура, описывающая

динамически разделяемый объект в памяти процесса). Используя этот дескриптор пользователь может осуществить динамическое связывание функций. Файл, в которой находится динамическая библиотека, указывается первым параметром функции. Вторым параметром указывается режим выполнения связывания имен функций. В случае, если в динамической библиотеке есть связи с другими библиотеками, то они загружаются рекурсивно.

Файл динамической библиотеки может быть указан в абсолютном и относительном виде. При использовании относительного имени поиск файла осуществляется в следующем порядке:

- для исполняемых файлов, записанных в формате ELF - если в исполняемом файле указаны пути, в которых необходимо искать динамические библиотеки, то файл ищется в них;
- проверяется содержимое переменной среды окружения LD_LIBRARY_PATH, в которой через двоеточие указываются каталоги для поиска динамических библиотек;
- используется содержимое файла `/etc/ld.so.cache` (о формировании этого файла будет сказано ниже);
- библиотека ищется в каталогах `/lib` и `/usr/lib`.

Если в качестве первого параметра указан NULL, то возвращается дескриптор, содержащий указатель на исполняемый процесс (т.е. в качестве динамической библиотеки будет использован исполняемый файл процесса).

После того, как библиотека будет загружена в память, системное программное обеспечение должно подготовиться для поиска в ней функций. Сделать это можно в двух режимах – сразу найти все имена функций библиотеки и их адреса в памяти (режим `RTLD_NOW`) или по мере вызова функции динамического связывания (`RTLD_LAZY`). Режим поиска сразу всех функций может быть также задан с применением переменной среды окружения `LD_BIND_NOW`, задав ей непустое значение. Дополнительно к режимам поиска имен функций могут быть заданы опции (через операцию бинарного сложения):

- `RTLD_GLOBAL` – использовать имена из этой библиотеки для разрешения имен из библиотек, загружаемых после указанной;
- `RTLD_LOCAL` – использовать имен из этой библиотеки только для ней. Этот режим используется «по умолчанию».
- `RTLD_NODELETE` – не выгружать библиотеку при вызове функции `dlclose()`;
- `RTLD_NOLOAD` – не загружать библиотеку. Используется для отладки.

Связывание функций (получение адреса функции) осуществляется с помощью вызова `dlsym`, которому в качестве первого параметра передается указатель на дескриптор, а вторым параметром строка, содержащая имя требуемой функции. Если функция найдена, то возвращается указатель на неё, иначе NULL.

В ситуации, когда несколько динамических библиотек используют функции с одинаковыми именами необходимо обеспечить механизм поиска нужной функции. Для этого в функции `dlsym` могут использоваться специальные дескрипторы – `RTLD_NEXT` и `RTLD_DEFAULT`. Первый дескриптор позволяет продолжить поиск требуемой функции в библиотеках, загруженных следующими по списку. Второй вызов ищет требуемую функцию в самой ранней загруженной библиотеке.

Функция `dlclose` закрывает динамическую библиотеку и при необходимости выгружает её из памяти ЭВМ.

Некоторым динамическим библиотекам необходимо выполнить какие-то действия перед началом или по окончании их использования. Для реализации этих механизмов используются специальные функции (см. рисунок **2** и рисунок **3**): `_init()` и `_fini()`, а также конструктор `__attribute ((constructor))` и деструктор библиотеки `__attribute ((destructor))`.

При компиляции разделяемой библиотеки следует помнить о том (см. рисунок **4**), что она должна быть скомпилирована в адресно-независимой форме (опция –fPIC), а линковка должна пройти без сборки исполняемого файла (опция –shared). Чтобы линковщик сделал все функции программы подготовил для динамического использования необходимо использовать опцию –rdynamic.

```
#include <stdio.h>
#include <stdlib.h>
#define __USE_GNU
#include <dlfcn.h>
static void * (*real_malloc) (size_t);
static int (*fu) (void);
void * malloc (size_t size){
  if (size > 100) { (*fu)(); return (NULL); }
  return (*real_malloc)(size);
}
static void __malloc_init (void) __attribute__ ((constructor));
static void __malloc_init (void){
  void * ptr;
  ptr = dlopen (NULL, RTLD_LAZY);
  if (ptr == NULL) abort();
  real_malloc = dlsym (RTLD_NEXT, "malloc");
  fu = dlsym (ptr, "func");
  if (NULL == real_malloc || NULL == fs){
   fprintf (stderr, "Error in `dlsym`: %s\n", dlerror ());
   return;
  }
}
```

Рисунок **2** – Пример динамической библиотеки функций

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

int func (void){
  printf ("func!!!!\n");
  return (1);
}

int main (void){
  void * ptr;
  ptr = malloc (1000);
  printf ("Результат выделения = %s\n",
                      (ptr == NULL)? "отказ" : "удача");
  return (0);
}
```

Рисунок **3** – Пример использования динамической библиотеки функций

```
$ gcc -fPIC -shared -o malloc.so malloc.c -ldl
$ gcc prog.c -o prog -rdynamic
$ ./prog
Результат выделения = удача
$ LD_PRELOAD="./malloc.so" ./prog
func!!!!
Результат выделения = отказ
$
```

Рисунок **4** – Компиляция динамической библиотеки и запуск программы с её использованием

3. Динамическая компоновка приложений

Автоматизация использования динамических библиотек заключается в использовании специальной программы, которая запускается сразу после загрузки исполняемого когда в память ЭВМ и выполняет открытие всех необходимых динамических библиотек и связывание всех используемых в программе функций. Такая программа в линуксе называется ld.so (или ld-linux.so).

Следует напомнить, что каждая динамическая библиотека имеет свое внутреннее имя, задаваемой опцией –soname линковщика!

Поиск динамических библиотек программой ld.so осуществляется в следующем порядке:

- Загружаются библиотеки, указанные в переменной среды окружения LD_PRELOAD;
- Просматриваются каталоги, указанные в параметрам исполняемого файла (DT_RPATH, DT_RUNPATH);
- Используются каталоги, указанные в переменной LD_LIBRARY_PATH;
- Просматривается файл /etc/ld.so.cache;
- Осуществляем поиск в стандартных каталогах - /lib и /usr/lib.

Файл /etc/ld.so.cache формируется утилитой ldconfig, которая для этого использует текстовый файл /etc/ld.so.conf и параметры командной строки. В файле /etc/ld.so.conf в каждой строке указывается путь к каталогу, которые необходимо просмотреть для поиска требуемой библиотеки.

Задание на лабораторную работу

1. Напишите программу, которая динамически выделяет 100 блоков памяти по 1000 байт каждый и затем освобождает их. Продемонстрируйте работоспособность Вашей программы.
2. Напишите динамическую библиотеку, в которой реализуйте две функции: `malloc` и `free`. Функция `malloc` выделяет (с использованием стандартной функции `malloc` из библиотеки `GLIBC`) запрашиваемый блок памяти до тех пор, пока не будет выделено 77 блоков. Далее любой вызов функции `malloc` приводит к ошибке выделения памяти. Функция `free` печатает на экране сообщение о количестве выделенных блоков памяти и освобождает запрашиваемый блок).
3. Используя принудительную загрузку библиотек продемонстрируйте работу созданной в п.2 библиотеки на примере созданной в п.1 программы.

Контрольные вопросы

1. Что такое динамическая библиотека? В чем её отличие от статической?
2. Что такое динамическое связывание и динамическая загрузка?
3. Для чего используется дескриптор RTLD_NEXT?
4. Какие переменные среды окружения используются загрузчиком динамических библиотек.