3. БИБЛИОТЕКИ

3.1. Введение в библиотеки

Как уже неоднократно упоминалось, *библиотека* - это набор скомпонованных особым образом объектных файлов. Библиотеки подключаются к основной программе во время линковки. По способу компоновки библиотеки подразделяют на *архивы* (статические библиотеки, static libraries) и *совместно используемые* (динамические библиотеки, shared libraries). В Linux, кроме того, есть механизмы *динамической подгрузки* библиотек. Суть динамической подгрузки состоит в том, что запущенная программа может по собственному усмотрению подключить к себе какую-либо библиотеку. Благодаря этой возможности создаются программы с *подключаемыми плагинами*, такие как XMMS. В этой главе мы не будем рассматривать динамическую подгрузку, а остановимся на классическом использовании статических и динамических библиотек.

С точки зрения модели КИС, библиотека - это сервер. Библиотеки несут в себе одну важную мысль: возможность использовать одни и те же механизмы в разных программах. В Linux библиотеки используются повсеместно, поскольку это очень удобный способ "не изобретать велосипеды". Даже ядро Linux в каком-то смысле представляет собой библиотеку механизмов, называемых *системными вызовами*.

Статическая библиотека - это просто архив объектных файлов, который подключается к программе во время линковки. Эффект такой же, как если бы вы подключали каждый из файлов отдельно.

В отличие от статических библиотек, код совместно используемых (динамических) библиотек не включается в бинарник. Вместо этого в бинарник включается только *ссылка* на библиотеку.

Рассмотрим преимущества и недостатки статических и совместно используемых библиотек. Статические библиотеки делают программу более автономной: программа, скомпонованная со статической библиотекой может запускаться на любом компьютере, не требуя наличия этой библиотеки (она уже "внутри" бинарника). Программа, скомпонованная с динамической библиотекой, требует наличия этой библиотеки на том компьютере, где она запускается, поскольку в бинарнике не код, а ссылка на код библиотеки. Не смотря на такую зависимость, динамические библиотеки обладают двумя существенными преимуществами. Во-первых, бинарник, скомпонованный с совместно используемой библиотекой меньше размером, чем такой же бинарник, с подключенной к нему статической библиотекой (статически скомпонованный бинарник). Во-вторых, любая модернизация динамической библиотеки, отражается на всех программах, использующих ее. Таким образом, если некоторую библиотеку foo используют 10 программ, то исправление какой-нибудь ошибки в foo или любое другое улучшение библиотеки автоматически улучшает все программы, которые используют эту библиотеку. Именно поэтому динамические библиотеки называют совместно используемыми. Чтобы применить изменения, внесенные в статическую библиотеку, нужно пересобрать все 10 программ.

В Linux статические библиотеки обычно имеют расширение **.a** (Archive), а совместно используемые библиотеки имеют расширение **.so** (Shared Object). Хранятся библиотеки, как правило, в каталогах /lib и /usr/lib. В случае иного расположения (относится только к совместно используемым библиотекам), приходится указать каталог с этой библотекой в /etc/ld.so.conf и запустить ldconfig с правами суперпользователя, чтобы программа запустилась.

3.2. Пример статической библиотеки

Теперь давайте создадим свою собственную библиотеку, располагающую двумя функциями: h_world() и g_world(), которые выводят на экран "Hello World" и "Goodbye World" соответственно. Начнем со статической библиотеки.

Начнем с интерфейса. Создадим файл world.h:

```
/* world.h */
void h_world (void);
void g_world (void);
```

Здесь просто объявлены функции, которые будут использоваться.

Теперь надо реализовать серверы. Создадим файл h_world.c:

```
/* h_world.c */
#include <stdio.h>
#include "world.h"

void h_world (void)
{
        printf ("Hello World\n");
}
```

Теперь создадим файл g_world.c, содержащий реализацию функции g_world():

```
/* g_world.c */
#include <stdio.h>
#include "world.h"

void g_world (void)
{
        printf ("Goodbye World\n");
}
```

Можно было бы с таким же успехом уместить обе функции в одном файле (hello.c, например), однако для наглядности мы разнесли код на два файла.

Теперь создадим файл main.c. Это клиент, который будет пользоваться услугами сервера:

```
/* main.c */
#include "world.h"

int main (void)
{
        h_world ();
        g_world ();
}
```

Теперь напишем сценарий для make. Для этого создаем Makefile:

```
# Makefile for World project
```

```
binary: main.o libworld.a
	gcc -o binary main.o -L. -lworld

main.o: main.c
	gcc -c main.c

libworld.a: h_world.o g_world.o
	ar cr libworld.a h_world.o g_world.o

h_world.o: h_world.c
	gcc -c h_world.c

g_world.o: g_world.c
	gcc -c g_world.c

clean:
	rm -f *.o *.a binary
```

Не забывайте ставить табуляции перед каждым правилом в целевых связках.

Собираем программу:

```
$ make
gcc -c main.c
gcc -c h_world.c
gcc -c g_world.c
ar cr libworld.a h_world.o g_world.o
gcc -o binary main.o -L. -lworld
$
```

Осталось только проверить, работает ли программа и разобраться, что же мы такое сделали:

```
$ ./binary
Hello World
Goodbye World
$
```

Итак, в приведенном примере появились три новые вещи: опции **-l** и **-L** компилятора, а также команда **ar**. Начнем с последней. Как вы уже догадались, команда ar создает статическую библиотеку (архив). В нашем случае два объектных файла объединяются в один файл libworld.a. В Linux практически все библиотеки имеют префикс lib.

Как уже говорилось, компилятор gcc сам вызывает линковщик, когда это нужно. Опция -l, переданная компилятору, обрабатывается и посылается линковщику для того, чтобы тот подключил к бинарнику библиотеку. Как вы уже заметили, у имени библиотеки "обрублены" префикс и суффикс. Это делается для того, чтобы создать "видимое безразличие" между статическими и динамическими библиотеками. Но об этом речь пойдет в других главах книги. Сейчас важно знать лишь то, что и библиотека libfoo.so и библиотека libfoo.a подключаются к проекту опцией **-lfoo**. В нашем случае libworld.a "урезалось" до -lworld.

Опция -L указывает линковщику, где ему искать библиотеку. В случае, если библиотека располагается в каталоге /lib или /usr/lib, то вопрос отпадает сам собой и опция -L не требуется. В нашем случае библиотека находится в репозитории (в текущем каталоге). По умолчанию линковщик не просматривает текущий каталог в поиске библиотеки, поэтому опция -L. (точка означает текущий каталог) необходима.

3.3. Пример совместно используемой библиотеки

Для того, чтобы создать и использовать динамическую (совместно используемую) библиотеку, достаточно переделать в нашем проекте Makefile.

```
# Makefile for World project

binary: main.o libworld.so
	gcc -o binary main.o -L. -lworld -Wl,-rpath,.

main.o: main.c
	gcc -c main.c

libworld.so: h_world.o g_world.o
	gcc -shared -o libworld.so h_world.o g_world.o

h_world.o: h_world.c
	gcc -c -fPIC h_world.c

g_world.o: g_world.c
	gcc -c -fPIC g_world.c

clean:
	rm -f *.o *.so binary
```

Внешне ничего не изменилось: программа компилируется, запускается и выполняет те же самые действия, что и в предыдущем случае. Изменилась **внутренняя суть**, которая играет для программиста первоочередную роль. Рассмотрим все по порядку.

Правило для сборки binary теперь содержит пугающую опцию **-Wl,-rpath,.** Ничего страшного тут нет. Как уже неоднократно говорилось, компилятор gcc сам вызывает линковщик ld, когда это надо и передает ему нужные параметры сборки, избавляя нас от ненужной платформенно-зависимой волокиты. Но иногда мы все-таки должны вмешаться в этот процесс и передать линковщику "свою" опцию. 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). Возникает вопрос: что означает опция -rpath? Как уже говорилось, линковщик ищет библиотеки в определенных местах; обычно это каталоги /lib и /usr/lib, иногда /usr/local/lib. Опция -rpath просто добавляет к этому списку еще один каталог. В нашем случае это текущий каталог. Без указания опции -rpath, линковщик "молча" соберет программу, но при запуске нас будет ждать сюрприз: программа не запустится из-за отсутствия библиотеки. Попробуйте убрать опцию -Wl,-rpath,. из Makefile и пересоберите проект. При попытке запуска программа binary завершится с кодом возврата 127 (о кодах возврата будет рассказано в последующих главах). То же самое произойдет, если вызвать программу из другого каталога. Верните обратно -Wl,-rpath,., пересоберите проект, поднимитесь на уровень выше командой cd .. и попробуйте запустить бинарник командой world/binary. Ничего не получится, поскольку в новом текущем каталоге библиотеки нет.

Есть один способ не передавать линковщику дополнительных опций при помощи -Wl - это использование переменной окружения LD_LIBRARY_PATH. В последующих главах мы будем подробно касаться темы окружения (environment). Сейчас лишь скажу, что у каждого пользователя есть так называемое окружение (environment) представляющее собой набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, используемых программами. Чтобы посмотреть окружение, достаточно набрать команду env. Чтобы добавить в окружение переменную, достаточно набрать export ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, а чтобы удалить переменную из окружения, надо набрать export -n ПЕРЕМЕННАЯ. Будьте внимательны: export - это **внутреннаяя**

команда оболочки BASH; в других оболочках (csh, ksh, ...) используются другие команды для работы с окружением. Переменная окружения LD_LIBRARY_PATH содержит список дополнительных "мест", разделенных двоеточиеями, где линковщих должен искать библиотеку.

Не смотря на наличие двух механизмов передачи информации о нестандартном расположении библиотек, лучше помещать библиотеки в конечных проектах в /lib и в /usr/lib. Допускается расположение библиотек в подкаталоги /usr/lib и в /usr/local/lib (с указанем -Wl,-rpath). Но заставлять конечного пользователя устанавливать LD_LIBRARY_PATH почти всегда является плохим стилем программирования.

Следующая немаловажная деталь - это процесс создания самой библиотеки. Статические библиотеки создаются при помощи архиватора ar, а совместно используемые - при помощи gcc с опцией -shared. В данном случае gcc опять же вызывает линковщик, но не для сборки бинарника, а для создания динамической библиотеки.

Последнее отличие - опциии -fPIC (-fpic) при компиляции h_world.c и g_world.c. Эта опция сообщает компилятору, что объектные файлы, полученные в результате компиляции должны содержать **позиционно-независимый код** (PIC - Position Independent Code), который используется в динамических библиотеках. В таком коде используются не фиксированные позиции (адреса), а плавающие, благодаря чему код из библиотеки имеет возможность подключаться к программе в момент запуска.