Модульная анатомия

Цель

Разобраться в структуре загружаемых модулей Linux, больше узнать об устройстве ядра.

Ресурсы

Ноутбук Acer Aspire 3680 (старенький), дистрибутив Debian, голова и интернет.

Действия

Первый пункт

Нам нужно скачать, скомпилировать исходники ядра, с которым мы будем работать. Пользуемся возможностями Debian и получаем исходники ядра из репозитория Debian (всегда можно воспользоваться и http://kernel.org)

- sudo apt-get update
- sudo apt-get install liux-doc-2.6.32 linux-manual-2.6.32 linux-source-2.6.32
- cd /usr/src/
- tar jxf /usr/src/linux-source-2.6.32.tar.bz2
- sudo apt-get install build-essential fakeroot kernel-package
- make menuconfig
- sudo make-kpkg clean
- sudo fakeroot make-kpkg –initrd –append-to-version=-mine kernel_image kernel_headers
- sudo dpkg -i linux-image-2.6.32-mine-10.00.Custom_i386.deb
- sudo dpkg -i linux-headers-2.6.32-mine-10.00Custom_i386.deb
- sudo update-initramfs -c -k 2.6.32-mine
- sudo shutdown -r now

Дальше, если ядро сконфигурированно нормально, то загружаемся и вроде все, кстати update-iniramfs нужен только, если initrd.img-2.6.32-mine не создался на лету, когда устанавливался пакет

*1 советую обновить gcc (если пользуетесь stable, то обновлять с testing, в противном случае ядро не соберется, так как не будет нужных заголовочных файлов), вообще стоит использовать новые версии всех требуемых пакетов

*2 возможно придется доставить некоторые другие пакеты (см /usr/share/doc/kernel-package/Kernel.htm)

*3 ядро можно собрать и не “в стиле Debian”, а обычным образом - нет никакой разницы

*4 в даной версии ядра пришлось поправить файл /usr/src/linux-source-2.6.32/Documentation/lguest/lguest.c, в нем нужно было убрать строку #include <sys/eventfd.h> (21 строка), в противном случае оно просто отказывалось компилироваться, хотя я не понял почему (потомучто такое же ядро на другом компьютере собралось без проблем), но погуглив нашел, что такая проблема не только у меня, и что такое решение используют и другие, после такого решения мы получаем при компиляции implicit декларацию функции, короче если в этом месте будет ошибка, то когда она вылезет непонятно

Второй пункт

Проверим, что все работает, для этого напишем какой-нибудь бесполезный модуль, скомпилируем его и посмотрим, что получится

```
1   //hello-1.c
2   #include <linux/module.h>
3   #include <linux/kernel.h>
4
5   int init_module(void) {
6        printk(KERN_INFO "Hellow world\n");
7        return 0;
8   }
9
10  void cleanup_module(void) {
11      printk(KERN_INFO "Godbye world\n");
12  }
```

Теперь Makefile

```
1   obj-m += hello-1.c
2   all:
3        make -C /usr/src/linux-source-2.6.32 M=$(shell pwd) modules
4   clean:
5        make -C /usr/src/linux-source-2.6.32 M=$(shell pwd) clean
```

Теперь из каталога, в котором лежит исходник и Makefile делаем

- sudo make

Должны получить примерно следующее:

```
   1 make -C /usr/src/linux-source-2.6.32
M=/home/mirovingen/Interested/kernel_programming/src/Hellow_World modules
   2 make[1]: Entering directory `/usr/src/linux-source-2.6.32'
   3   CC [M]
/home/mirovingen/Interested/kernel_programming/src/Hellow_World/hello-1.o
   4   Building modules, stage 2.
   5   MODPOST 1 modules
```

```
   6   CC
/home/mirovingen/Interested/kernel_programming/src/Hellow_World/hello-1.mod.
o
   7   LD [M]
/home/mirovingen/Interested/kernel_programming/src/Hellow_World/hello-1.ko
   8 make[1]: Leaving directory `/usr/src/linux-source-2.6.32'
```

Если все получилось, у нас в каталоге должен появиться файл hello-1.ko, далее загружаем и выгружаем модуль:

```
   1 sudo insmod hello-1.ko
   2 sudo rmmod hello-1.ko
   3 sudo cat /var/log/messages | grep -i -e "world"
```

Должны получить примерно следующее:

```
   1 Jan 28 22:32:25 debian kernel: [ 6196.992494] Hellow world
   2 Jan 28 22:40:17 debian kernel: [ 6669.012286] Goodbye world
```

Теперь разбираемся, что мы сделали и что получили.

Исходный текст

#include <linux/module.h> - как написано, этот заголовочный файл должен быть у всех модулей, в нем есть объявления init_module и cleanup_module, еще куча структур описывающих состояние, версию и другую информацию о модулях. #include <linux/kernel.h> - тут тоже есть много чего полезного, например максимальное и минимальное значение определенного типа, printk объявлена в этом заголовочном файле, KERN_INFO также объявлена здесь: #define KERN_INFO “<6>” /* informational - она объявлена как строка, получается мы вызываем printk(“<6>” “text”); это тоже самое, что и printk(“<6>text”); ??? раньше не встречал такого варианта использования.

int init_module(void) - вызывается, когда мы загружаем модуль (insmod); void cleanup_module(void) - вызывается, когда код выгружается (rmmod); В данном случае мы не вольны выбирать имена функций входа и выхода, но если подключить заголовочный файл linux/init.h, то можно присваивать произвольное имя функциям входа и выхода (дальше будет пример)

printk - выводит значения, но она работает не совсем так как printf, как я понял, она выводит сообщение в какую-то очередь сообщений ядра, которую просмотреть скажем в xterm нельзя, поэтому для просмотра вывода мы заглядываем в файл /var/log/messages, который, кстати, довольно большой, и, наверно, очень полезный.

Прошу знатоков еще пояснить вот такой момент:

код возврата определяется программистом, т. е. он может не совпадать с какими-то принятыми в ОС значениями, для прикладных программ в этом нет ничего смертельного, но в данном случаем мы ведь не свободны в выборе кода возврата, так как потом система использует его значение? если да то какие еще коды существуют, и что они обозначают?

Makefile

Как написано формально Makefile должен содержать только строку obj-m += hello-1.o, но тогда простой командой make мы получим только ошибку, правильным вариантом вызова будет:

```
make -C /usr/src/linux-source-2.6.32 M=$(shell pwd) modules
```

$(shell pwd) - текущий каталог, можно прописать и руками. /usr/src/linux-source-2.6.32 - каталог с исходниками ядра. modules - не знаю зачем нужно, предполагаю, что эта команда показывает, что мы собираем именно модули ядра.