Лабораторная работа «Загружаемые модули ядра»

Цель работы: знакомство с базовыми принципами разработки и взаимодействия с загружаемыми модулями ядра ОС Linux.

Теоретическая часть

Макросы module_init и module_init

Загружаемые модули ядра должны содержать два макроса **module_init** и **module_exit**.

Макрос module_init

Служит для регистрации функции инициализации модуля. Макрос принимает имя функции в качестве фактического параметра. В результате эта функция будет вызываться, при загрузке модуля в ядро. Передаваемая функция должна соответствовать следующему прототипу:

```
int func_init(void);
```

Функция возвращает значение типа int. Если функция инициализации завершилась успешно, то возвращается значение ноль. В случае ошибки возвращается ненулевое значение.

Как правило, функция инициализации предназначена для запроса ресурсов и выделения памяти под структуры данных и т.п.. Так как функция инициализации редко вызывается за пределами модуля, ее обычно не нужно экспортировать и можно объявить с ключевым словом static.

Если функция не передается в макрос, то код, который необходимо выполнить при загрузке модуля может быть написан в самом макросе.

Определение макроса module_init:

```
#define __initcall(fn) \
static initcall_t __initcall_##fn __init_call = fn
#define __init_call __attribute__ \
((unused,__section__ ("function_ptrs")))
#define module_init(x)  __initcall(x);
```

Макрос module_exit

Макрос служит для регистрации функции, которая вызывается при удалении модуля из ядра. Обычно эта функция выполняет задачу освобождения ресурсов. После завершения функции модуль выгружается.

Функция завершения должна соответствовать прототипу:
```
void func_exit(void);
```

Функцию завершения, как и инициализации, можно объявить как static.

Пример простого загружаемого модуля ядра:
```
static int __init md_init( void )
{
   printk( "Module md loaded!\n" );
   return 0;
}

static void __exit md_exit( void )
{
   printk( "Module unloaded!\n" );
}

module_init( md_init );
module_exit( md_exit );
```

Коментарии к примеру:

В предложенном коде используются макросы __init, __exit а также функция printk. Макрос __init используются для обозначения функций, встраиваемых в ядро, или инициализированных данных, которые ядро воспринимает как указание о том, что функция используется только на этапе инициализации и освобождает использованные ресурсы после. Это позволяет экономить ресурсы системы.

В ядре макрос определяется следующим образом:
```
#define __init  __section(.init.text) \
__cold  __latent_entropy __noinitretpoline
```

The __exit macro causes the omission of the function when the module is built into the kernel, and like __exit, has no effect for loadable modules. Again, if you consider when the cleanup function runs, this makes complete sense; built-in drivers don't need a cleanup function, while loadable modules do.
В ядре макрос определяется следующим образом:
```
#define __exit  __section(.exit.text) \
__exitused __cold notrace
```

Макросы **__init** и **__exit** определены в linux/init.h.
https://elixir.bootlin.com/linux/v3.14/source/include/linux/init.h

Если файл будет статически скомпилирован с образом ядра, то данная функция не будет включена в образ и никогда не будет вызвана (так как если нет модуля, то код никогда не может быть удален из памяти).

Функция printk определена в ядре Linux и доступна модулям. Функция ведёт себя аналогично библиотечной функции printf. Став частью ядра модуль не может вызывать обычные библиотечные функции, поэтому ядро предоставляет модулю функцию printk.

Функция printk позволяет отправлять сообщения в системный журнал. Сама функция не производит запись в системный журнал, а записывает сообщение в специальный буфер ядра. Из буфера ядра записанные сообщения могут быть прочитаны демоном протоколирования.

Пример использования функции printk :

```
int var = 1;
printk(KERN_INFO "var = %d\n", var);
```

где `KERN_INFO` - уровень протоколирования сообщения.
Система поддерживает восемь уровней протоколирования, показанных в таблице 1. Все константы определены в файле linux/kernel.h.

Таблица 1

Уровень	**Константа**	**Описание**
7	`KERN_DEBUG`	Отладочные сообщения, самый низкий приоритет
6	`KERN_INFO`	Информационные сообщения
5	`KERN_NOTICE`	Это уже не информационное сообщение, но еще и не предупреждение
4	`KERN_WARNING`	Предупреждение: скоро может пойти что-то не так
3	`KERN_ERR`	Возникла ошибка
2	`KERN_CRIT`	Возникла критическая ошибка
1	`KERN_ALERT`	Тревога — система скоро "развалится"
0	`KERN_EMERG`	Система "развалилась" (система больше не может использоваться)

Стоит обратить внимание на то, что после уровня протоколирования не стоит запятая, так как она является частью строки форматирования, это сделано в системе для экономии памяти стека при вызове функции.

Макросы для установки информации о модуле

Макросы MODULE_LICENSE, MODULE_AUTHOR, MODULE_DESCRIPTION, MODULE_SUPPORTED_DEVICE, MODULE_VERSION — это макросы, которые «дают» linux информацию о модуле, которую потом можно получить с помощью команды modinfo. Описание макросов содержится в файле linux/module.h.

Макрос MODULE_LICENSE

Макрос используется для того, чтобы сообщить ядру, под какой лицензией распространяется исходный код модуля, что влияет на то, к функциям и переменным (символы ядра) он может получить доступ в ядре. Например, модуль под лицензией GPLv2 имеет доступ ко всем символам ядра. Без этой декларации при загрузке модуля ядро выводит предупреждение.

Поддерживаемые варианты:

Параметр	Значение
"GPL"	[GNU Public License v2 or later]
"GPL v2"	[GNU Public License v2]
"GPL and additional rights"	[GNU Public License v2 rights and more]
"Dual BSD/GPL"	[GNU Public License v2 or BSD license choice]
"Dual MIT/GPL"	[GNU Public License v2 or MIT license choice]
"Dual MPL/GPL"	[GNU Public License v2 or Mozilla license choice]
"Proprietary"	[Non free products]

Загрузка в память модуля, для которого лицензия не соответствует GPL, приведет к установке в ядре флага tainted (испорчено), который служит для информационных целей (например, в сообщениях об ошибках).

Пример использования:
```
MODULE_LICENSE("GPL");
```

Макрос MODULE_AUTHOR

Позволяет указать автора модуля. Значение этого макроса служит только для информационных целей.

Пример использования:
```
MODULE_AUTHOR("Author");
```

Макрос MODULE_DESCRIPTION

Позволяет указать описание модуля. Значение этого макроса служит только для информационных целей.

Макрос MODULE_SUPPORTED_DEVICE

Помещает запись, описывающую, какое устройство поддерживается этим модулем. Комментарии в источниках ядра предполагают, что в конечном итоге эти параметры могут использоваться для автоматической загрузки модулей, но в настоящее время такое использование не производится.

Макрос MODULE_VERSION

Позволяет указать версию модуля. Значение этого макроса служит только для информационных целей.

Экспортируемые символы

Экспортируемыми символами называются данные и функции, которыми могут пользоваться другие модули ядра. При загрузке модули динамически компонуются с ядром, в коде модулей могут вызываться только те функции ядра, которые явно экспортируются для использования.

Макрос EXPORTOL

В ядре экспортирование осуществляется с помощью специального макроса EXPORT_SYMBOL(). Функции, которые экспортируются, доступны для использования модулям, остальные функции не могут быть вызваны из модулей. Для экспортируемых данных правило аналогично.

Пример использования:
```
int md1_data = 42; // экспортируемые данные

extern int md1_func(int n) // экспортируемая функция
{
    return n * 2;
```

```
}

EXPORT_SYMBOL(md1_data);
EXPORT_SYMBOL(md1_func);
```

Для того, чтобы другие модули могли использовать эспортированные символы, они должны “знать” их определение. Для этого применяются заголовочные файлы.

Пример заголовочного файла:
```
extern int md1_data;
extern int md1_func(int n);
```

Макрос EXPORT_SYMBOL_GPL
Иногда необходимо, чтобы символы были доступны только для модулей, имеющих соответствующую лицензию GPL. Для этого используют макрос EXPORT_SYMBOL_GPL().

Команды linux для работы с загружаемыми модулями

В ОС Linux существуют специальные команды для работы с загружаемыми модулями ядра.

insmod
Загружает модуль в ядро из конкретного файла, если модуль зависит от других модулей, которые не загружены в ядро, то выдает ошибку и не загружает модуль. Только суперпользователь может загрузить модуль в ядро.

Пример использования: `sudo insmod ./md.ko`

lsmod
Выводит список модулей, загруженных в ядро.

modinfo
Извлекает информацию из модулей ядра (лицензия, автор, описание и т.д.).

Пример использования:
```
modinfo md.ko
filename:       /home/alexander/Develop/OS/Kernel/TASK1/md.ko
author:         Sychev Svyatoslav
license:        GPL
srcversion:     10E5B7B44C5B0320F3FC7B2
depends:
name:           md
vermagic:       4.13.0-32-generic SMP mod_unload
```

rmmod
Команда используется для выгрузки модуля из ядра, в качестве параметра передается имя файла модуля. Только суперпользователь может выгрузить модуль из ядра.

Пример использования: `sudo rmmod md1.ko`

dmesg
Команда для вывода буфера сообщений ядра в стандартный поток вывода. Сообщения содержат информацию о драйверах устройств, загружаемых в ядро во время загрузки системы, а также при подключении аппаратного обеспечения к системе.

Замечание: системный журнал используется большим числом процессов, и для того чтобы было легче обнаружить сообщения конкретного модуля, рекомендуется в их начало помещать некоторый идентификатор (к примеру знак + или имя модуля). В этом случае просмотреть сообщения от модуля можно при помощи команды:
```
dmesg | tail -n60 | grep +
```

Работа с деревом модулей
Команда для загрузки модуля из дерева ядра со всеми зависимостями:
```
sudo modprobe
```

Команда работает только с деревом модулей, загрузка возможно только по имени модуля, а не имени файла. Используется для подгрузки готовых модулей, включенных в дерево модулей текущей версии ядра.

Для получения списка всех модулей из дерева каталогов, нужно выполнить команду:
```
find /lib/modules/`uname -r` -name '*.ko'
```

Вместо `` `uname -r` `` подставляется текущая версия ядра.

Для выгрузки модуля и всех модулей, которые зависят от него, используется команда:
```
sudo modprobe -r
```

Для создания списка зависимостей модулей используется команда:
```
sudo depmod -a
```

Данная команда считывает каждый модуль из каталога `/lib/modules/`uname -r`` и определяет, какие символы они экспортируют. Результат работы команды хранится в файле modules.dep в каталоге `/lib/modules/`uname -r``.

Пример файла modules.dep:

...
md1.ko:
md2.ko: md1.ko
...

Замечание:
Модуль md2.ko использует символы из md1.ko, а модулю md1.ko для работы другие модули не нужны.

Задания на лабораторную работу

Задание 1

Реализовать загружаемый модуль ядра, осуществляющий перебор списка задач (struct task_struct) и в системный файл /var/log/messages выводит идентификатор каждого встреченного процесса и имя его исполняемого файла, идентификатор процесса предка и имя этого файла. При инициализации модуля следует также использовать символ current для вывода такой же информации о текущем процессе. При выгрузке модуля записывается “Good by”. Модуль должен собираться при помощи Make-файла.

Загружаемый модуль должен содержать:

- Указание лицензии GPL
- Указание автора

Задание 2

Реализовать три загружаемых модуля ядра:

- Вызываемый модуль md1
- Вызывающий модуль md2
- «Отладочный» модуль md3

Каждый загружаемый модуль должен содержать:

- Указание лицензии GPL
- Указание автора

Загружаемые модули должны собираться при помощи Make-файла (сборка командой make). **Вызов каждой функции модуля должен сопровождаться записью в системный журнал** информации, какая функция какого модуля была вызвана.

Модуль md1

Модуль md1 демонстрирует возможность создания экспортируемых данных и функций. Данный модуль ядра должен содержать:

- Экспортируемые строковые (char *) и численные (int) данные.
- Экспортируемые функции возвращающие строковые и числовые значения.

Например:

  - Функция, возвращающая в зависимости от переданного целочисленного параметра различные строки (на усмотрение студента);
  - Функция, производящая подсчет факториала переданного целочисленного параметра;
  - Функция возвращающая 0;

Модуль md2

Модуль md2 демонстрирует использование данных и функций экспортируемых первым модулем (md1).

Данный модуль должен при загрузке:

- Вызывать все экспортированные модулем md1 процедуры и вывести в системный журнал возвращаемые ими значения с указанием имени вызванной процедуры.
- Вывести в системный журнал все экспортированные модулем md1 данные.

Модуль md3

Модуль md3 демонстрирует сценарий некорректного завершения установки модуля, и возможность использования загружаемого модуля в качестве функции выполняемой в пространстве ядре.

Процедура инициализации этого загружаемого модуля должна возвращать ненулевое значение и выводить в системный журнал данные и возвращаемые значения экспортированных модулем md1 процедур (аналогично md2).

Данный модуль включен в работу для проработки вопросов, связанных с отладкой модулей ядра.

Make-файл

Make-файл должен быть написан так, чтобы при вызове команды make происходила компиляция всех реализованных загружаемых модулей. Это позволит упростить процесс компиляции. Также Make-файл должен содержать правило clean для очистки директории от промежуточных файлов компиляции.

Пример Make-файла предназначенного для сборки и компиляции загружаемого модуля ядра:

```
ifneq ($(KERNELRELEASE),)
	obj-m   := md.o
else
	CURRENT = $(shell uname -r)
	KDIR = /lib/modules/$(CURRENT)/build
	PWD = $(shell pwd)
default:
	$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
	@rm -f *.o .*.cmd .*.flags *.mod.c *.order
	@rm -f .*.*.cmd *~ *.*~ TODO.*
	@rm -fR .tmp*
	@rm -rf .tmp_versions
disclean: clean
	@rm *.ko *.symvers

endif
// $(MAKE) - вызов MAKE в режиме ядра.
```