Файловая система sysfs для экспорта объектов ядра

Patrick Mochel <mochel@osdl.org>
Mike Murphy <mamurph@cs.clemson.edu>

Обновление: 16 августа 2011
Оригинал: 10 января 2003

Что это такое

Файловая система sysfs создается в ОЗУ компьютера и реализована исходно на базе ramfs. Она обеспечивает возможность экспорта структур данных ядра, их атрибутов и связей между объектами в пользовательское пространство. Файловая система sysfs непосредственно связана с инфраструктурой kobject (см. документ Documentation/kobject.txt).

Использование sysfs

Поддержка sysfs определяется опцией CONFIG_SYSFS при сборке ядра. Для доступа к файловой системе используется команда **mount -t sysfs sysfs /sys**

Создание структуры каталогов

Для каждого регистрируемого у системе объекта kobject создается каталог в файловой системе sysfs. Этот каталог создается, как субкаталог родительского объекта kobject, что дает в результате представление иерархии объектов для пользовательского пространства. Каталоги верхнего уровня в sysfs представляют «корни» иерархии объектов.

В sysfs сохраняется указатель на объект kobject, который реализует каталог в объекте sysfs_dirent, связанном с каталогом. В прошлом указатель на этот объект kobject использовался sysfs для прямого учета открытия и закрытия файлов kobject. В современных реализациях sysfs implementation счетчик ссылок kobject изменяется только непосредственно функцией sysfs_schedule_callback().

Атрибуты

Атрибуты объектов kobject могут экспортироваться в форме обычных файлов и sysfs будет предавать файловые операции ввода-вывода методам, определенным для соответствующего атрибута, обеспечивая способ чтения и записи значений атрибутов ядра.

Атрибуты следует помещать в текстовые файлы ASCII (предпочтительно с единственным значением в каждом файле). Однако использование отдельного файла для каждого значения не эффективно, поэтому допустимо включение в один файл массива однотипных значений.

Смешивание типов с использованием множества строк данных усложняет формат файлов и работу с ними. В результате данные могут удаляться или переписываться без ведома.

Определение атрибута весьма просто:

```
struct attribute {
        char * name;
        struct module *owner;
        umode_t mode;
};

int sysfs_create_file(struct kobject * kobj, const struct attribute * attr);
void sysfs_remove_file(struct kobject * kobj, const struct attribute * attr);
```

«Голые» атрибуты не обеспечивают способов чтения и записи значений. Подсистемам рекомендуется определять собственные структуры атрибутов и функции для работы с ними.

Например, модель драйвера определяет структуры **device_attribute** вида:

```
struct device_attribute {
        struct attribute attr;
        ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
        ssize_t (*store)(struct device *dev, struct device_attribute *attr, const char *buf, size_t count);
};

int device_create_file(struct device *, const struct device_attribute *);
void device_remove_file(struct device *, const struct device_attribute *);
```

Эта же модель определяет функцию для работы с атрибутами:

```
#define DEVICE_ATTR(_name, _mode, _show, _store) \
struct device_attribute dev_attr_##_name = __ATTR(_name, _mode, _show, _store)
```

Например, декларирование

```
static DEVICE_ATTR(foo, S_IWUSR | S_IRUGO, show_foo, store_foo);
```

эквивалентно выполнению операции

```
static struct device_attribute dev_attr_foo = {
        .attr    = {
                .name = "foo",
                .mode = S_IWUSR | S_IRUGO,
                .show = show_foo,
                .store = store_foo,
        },
};
```

Специфические для подсистемы вызовы (Callback)

Когда подсистема определяет новый тип атрибута, она должна реализовать набор операций sysfs для передачи вызовов на чтение и запись методам просмотра и сохранения значений владельцу атрибута.

```
struct sysfs_ops {
        ssize_t (*show)(struct kobject *, struct attribute *, char *);
        ssize_t (*store)(struct kobject *, struct attribute *, const char *, size_t);
};
```

[подсистемам следует иметь уже определенную в качестве деструктора для данного типа структуру **kobj_type**, в которой хранится указатель sysfs_ops; см. документацию kobject]

При чтении или записи в файл sysfs вызывает подходящий для данного типа метод. Этот метод транслирует базовую структуру kobject и указатели на атрибуты структуры в соответствующие типы указателей и вызывает нужные методы. Например,

```
#define to_dev(obj) container_of(obj, struct device, kobj)
#define to_dev_attr(_attr) container_of(_attr, struct device_attribute, attr)
static ssize_t dev_attr_show(struct kobject *kobj, struct attribute *attr, har *buf) {
                struct device_attribute *dev_attr = to_dev_attr(attr);
                struct device *dev = to_dev(kobj);
                ssize_t ret = -EIO;

                if (dev_attr->show)
                        ret = dev_attr->show(dev, dev_attr, buf);
                if (ret >= (ssize_t)PAGE_SIZE) {
                        print_symbol("dev_attr_show: %s returned bad count\n",
                        (unsigned long)dev_attr->show);
                }
                return ret;
}
```

Чтение и запись атрибутов

Для чтения и записи атрибутов при их декларировании должны задаваться методы show() и store(). Методы следует делать достаточно простыми:

```
ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t (*store)(struct device *dev, struct device_attribute *attr,
        const char *buf, size_t count);
```

Иными словами, методам следует принимать в качестве параметров лишь объект, атрибут и буфер.

Sysfs выделяет буфер размера (PAGE_SIZE) и передает его методу. Sysfs будет вызывать соответствующий метод один раз при каждой операции чтения или записи. Это требует от реализации метода следующего поведения:

- При операции **read** методу **show()** следует заполнять буфер целиком. С учетом того, что атрибут содержит лишь одно экспортируемое значение или массив однотипных значений здесь не должно возникать сложностей. В результате из пользовательского пространства становится возможным частичное считывание и перемещение вперед по файлу (seek). Если приложение вернется к началу файла или выполнит операцию pread с нулевым смещением, метод **show()** будет вызван снова для повторного заполнения буфера.
- При записи (write) sysfs предполагает, что буфер передается целиком при первой операции записи. Содержимое буфера sysfs целиком передает методу **store()**.

  При записи в файлы sysfs пользовательским процессам следует сначала считать файл целиком, изменить нужное значение и записать весь буфер обратно в файл.

  Реализациям методов работы с атрибутами следует применять идентичные буферы для чтения и записи значений.

Примечания

- при записи метод **show()** вызывается независимо от текущей позициив файле;
- размер буфера всегда составляет **PAGE_SIZE** байтов (на i386 это 4096);
- методам **show()** следует возвращать число байтов, записанных в буфер)значение, возвращаемое **scnprintf()**);
- методам **show()** следует всегда использовать **scnprintf()**;
- методам **store()** следует возвращать число использованных байтов буфера (если использовались все байты, просто возвращается аргумент **count**);
- методы **show()** и **store()** могут возвращать коды ошибок (особенно при получении некорректного значения);
- объекты, переданные методам, помещаются в память с использованием счетчика ссылок sysfs, однако указываемый физический объект (например, устройство) может отсутствовать в системе и при необходимости следует проверять его наличие.

Пример очень простой и естественной реализации атрибута устройства показан ниже:

```
static ssize_t show_name(struct device *dev, struct device_attribute *attr, char *buf) {
        return scnprintf(buf, PAGE_SIZE, "%s\n", dev->name);
}

static ssize_t store_name(struct device *dev, struct device_attribute *attr, const char *buf, size_t count) {
        snprintf(dev->name, sizeof(dev->name), "%.*s", (int)min(count,
                sizeof(dev->name) - 1), buf);
        return count;
}

static DEVICE_ATTR(name, S_IRUGO, show_name, store_name);
```

Отметим, что практические реализации не должны позволять задание имен устройств из пользовательского пространства.

Схема каталогов верхнего уровня

Каталог sysfs показывает соотношения между структурами данных ядра. Субкаталоги верхнего уровня имеют вид:

```
block/
bus/
class/
dev/
devices/
firmware/
net/
fs/
```

Каталог **devices/** представляет дерево файлов, связанных с устройствами, и отображает их напрямую во внутреннее дерево устройств ядра, представляющее собой иерархию структур **device**. В каталоге **bus/** содержится плоская схема различных типов шин в ядре. Каталог для каждой из шин включает два субкаталога:

```
devices/
drivers/
```

В субкаталоге **devices/** размещаются символьные ссылки для каждого обнаруженного в системе устройства на соответствующие устройства в каталоге устройств корневой файловой системы (**root/**).

В каталоге **drivers/** содержится каталог для каждого драйвера устройства, который загружается для конкретной шины (предполагается, что каждый драйвер работает только с одним типом шины).

Дерево **fs/** содержит каталоги для некоторых файловых систем. В настоящее время система, желающая экспортировать атрибуты, должна создавать свою иерархию в дереве **fs/** (см, например документ Documentation/filesystems/fuse.txt в дистрибутиве ядра).

Каталог **dev/** содержит два субкаталога **char/** и **block/**, в каждом из которых размещаются символьные ссылки с именами вида **<major>:<minor>**. Эти ссылки указывают на каталоги sysfs для каждого из устройств. Структура **/sys/dev** обеспечивает возможность быстрого поиска интерфейса sysfs для устройства по результатам операции **stat**.

Дополнительную информацию о конкретных свойствах драйверов можно найти в каталоге Documentation/driver-model/.

Примечание: Этот раздел пока не завершен.

Текущие интерфейсы

Ниже перечислены интерфейсные уровни, существующие в sysfs на текущий момент.

Устройства (include/linux/device.h)

Структура

```
struct
 device_attribute {
        struct attribute attr;
        ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
        ssize_t (*store)(struct device *dev, struct device_attribute *attr,
                         const char *buf, size_t count);
};
```

Декларирование

```
DEVICE_ATTR(_name, _mode, _show, _store);
```

Создание и удаление

```
int device_create_file(struct device *dev, const struct device_attribute * attr);
void device_remove_file(struct device *dev, const struct device_attribute * attr);
```

Драйверы шин (include/linux/device.h)

Структура

```
struct bus_attribute {
        struct attribute attr;
        ssize_t (*show)(struct bus_type *, char * buf);
        ssize_t (*store)(struct bus_type *, const char * buf, size_t count);
};
```

Декларирование

```
BUS_ATTR(_name, _mode, _show, _store)
```

Создание и удаление

```
int bus_create_file(struct bus_type *, struct bus_attribute *);
void bus_remove_file(struct bus_type *, struct bus_attribute *);
```

Драйверы устройств (include/linux/device.h)

Структура

```
struct driver_attribute {
    struct attribute        attr;
    ssize_t (*show)(struct device_driver *, char * buf);
    ssize_t (*store)(struct device_driver *, const char * buf, size_t count);
};
```

Декларирование

```
DRIVER_ATTR(_name, _mode, _show, _store)
```

Создание и удаление

```
int driver_create_file(struct device_driver *, const struct driver_attribute *);
void driver_remove_file(struct device_driver *, const struct driver_attribute *);
```

Документация

Структура каталога **sysfs** и атрибуты в каждом из субкаталогов определяют бинарный интерфейс (ABI) между ядром и пользовательским пространством. Как и для всякого ABI здесь важно обеспечить стабильность и надлежащее документирование. Все новые атрибуты **sysfs** должны документироваться в Documentation/ABI. Дополнительную информацию можно найти в документе Documentation/ABI/README.

Перевод на русский язык
Николай Малых
nmalykh@protocols.ru