Методические указания по лабораторной работе «Виртуальная файловая система /proc»

1. Виртуальная файловая система /proc

Папки (каталоги) и файлы виртуальной файловой системы /proc не хранятся на диске. Они создаются динамически при обращении к ним.

Файловая система /proc фактически представляет собой интерфейс ядра, который позволяет получать информацию о процессах и ресурсах, которые они используют. При этом используется стандартный интерфейс файловой системы и системных вызовов. Из этого следует, что управление доступом к адресному пространству осуществляется при помощи обычных прав доступа – на чтение, на запись и выполнение.

Данные о каждом процессе хранятся в поддиректории с именем, которым является идентификатор процесса: /proc/<PID>. В поддиректории процесса находятся файлы и поддиректории, содержащие данные о процессе (табл.1):

Таблица – файлы и поддиректории /proc/<PID>

Элемент	Тип	Содержание
cmdline	файл	Указывает на директорию процесса
cwd	символическая ссылка	Указывает на директорию процесса
environ	файл	Список окружения процесса
exe	символическая ссылка	Указывает на образ процесса (на его файл)
fd	директория	Ссылки на файлы, которые «открыл» процесс
root	символическая ссылка	Указывает на корень файловой системы процесса
stat	файл	Информация о процессе

Процесс может получить свой идентификатор с помощью функции getpid().

Другой способ – использовать ссылку self: /proc/self.

2. Загружаемы модули ядра и виртуальная файловая система /proc

Файлы и поддиректории файловой системы /proc могут создаваться, их можно регистрировать и прекращать их регистрацию. Поэтому /proc часто используются загружаемыми модулями ядра. Файлы и

поддиректории файловой системы /proc используют структуру proc_dir_entry:

```
struct proc_dir_entry {

   const char *name;          // имя виртуального файла

   mode_t mode;               // режим доступа

   uid_t uid;             // уникальный номер пользователя -
                                  // владельца файла

   uid_t uid;             // уникальный номер группы, которой
                                  // принадлежит файл

   struct inode_operations *proc_iops; // функции-обработчики операций с
inode

   struct inode_operations *proc_iops; // функции-обработчики операций с
файлом

   struct proc_dir_entry *parent;      // Родительский каталог

   ...

   read_proc_t *read_proc;         // функция чтения из /proc

   write_proc_t *write_proc;       // функция записи в /proc

   void *data;            // Указатель на локальные данные

   atomic_t count;            // счетчик ссылок на файл

   ...

};
```

proc_dir_entry (9) (Linux man: Ядро)

```
struct proc_dir_entry {
    unsigned short low_ino;
    unsigned short namelen;
    const char *name;
    mode_t mode;
    nlink_t nlink;
    uid_t uid;
```

```
        gid_t gid;
        unsigned long size;
        struct inode_operations * ops;
        int (*get_info)(char *buffer, char **start,
              off_t offset, int length, int unused);
        void (*fill_inode)(struct inode *);
        struct proc_dir_entry *next, *parent, *subdir;
        void *data;
};
```

ow_ino : номер inode для директории. Для **proc_register** этот номер должен быть уникальным в файловой системе /proc, значения определены в <*linux/proc_fs.h*>. Для **proc_register_dynamic** номер inode назначаются динамически.
namelen : длина имени

name : уникальное имя виртуального файла **(**имя данного узла).
mode : тип и права доступа к узлу.
The node's type and permissions. Взяты из <*linux/stat.h*>.
nlink : число линков к узлу. Инициализировать до 2, если режим включает S_IFDIR, 1 в противном случае.
uid : идентификатор пользователя **(**uid), которому принадлежит файл (узел), обычно 0.
gid : идентификатор группы (gid), которой принадлежит узел, обычно 0.
size : устанавливает размер узла, значение будет отображаться как размер inode в списках и будет возвращено stat. Если размер не нужен, то его устанавливают равным нулю.
ops : определяет набор операций inode для узла / proc. Для узла каталога, если нет особых требований, используются & proc_dir_inode_operations. Для листового узла, если нет специальных требований, устанавливается значение NULL.
get_info : если определено, то вызывается, когда узел считывается. Должно быть равным NULL для узлов каталога. ПРИМЕЧАНИЕ. Если нужно вернуть большие объемы данных, то функция возвращает данные блоками а затем переместить себя на следующий вызов, используя offsetvariable. См. Ip_masq_procinfo, например, код с большим выходом.
fill_inode : динамически заполняет характеристики inode во время операций с каталогом. Обычно не требуется и устанавливается в NULL. См. Proc_pid_fill_inode, например, код - next, parent, subdir . Поддерживается подпрограммами / proc. Исходное значение не имеет значения, установлено значение NULL.
data : непрозрачный указатель, который может использоваться обработчиками proc для передачи локальных данных. Допускается устанавливать свободно при вызове proc_register, обычно NULL. Этот указатель копируется в поле inode.de_generic inode (by proc_get_inode), поэтому он доступен для всех процедур proc, которые передаются inode.

```
#include <linux/proc_fs.h>
int proc_register(struct proc_dir_entry * parent, struct proc_dir_entry * child);
int proc_unregister(struct proc_dir_entry * parent, int inode);
int proc_register_dynamic(struct proc_dir_entry * parent,
struct proc_dir_entry * child);
```

Функции proc_register() добавляют файл или каталог в файловую систему / proc. Они связывают процедуры обработки с каждым узлом дерева / proc. Функция proc_register() добавляет узел-потомок к узлу-предку.

3 . Файловая система /proc: создание файлов, доступных для чтения

Linux предоставляет ядру и модулям ядра дополнительный механизм передачи информации заинтересованным в ней процессам -- это файловая система /proc. Первоначально она создавалась с целью получения сведений о процессах (отсюда такое название). Теперь она интенсивно используется и самим ядром, которому есть что сообщить! Например, /proc/modules -- список загруженных модулей, /proc/meminfo -- статистика использования памяти.

Методика работы с файловой системой /proc очень похожа на работу драйверов с файлами устройств: создаётся структура со всей необходимой информацией, включая указатели на функции-обработчики (в нашем случае имеется только один обработчик, который обслуживает чтение файла в /proc). Функция init_module() регистрирует структуру, а module_exit() отменяет регистрацию.

Основная причина, по которой используется proc_register_dynamic состоит в том, что номер inode, для создаваемого файла, заранее неизвестен, поэтому ядро может определить его самостоятельно, чтобы предотвратить возможные конфликты. В обычных файловых системах, размещенных на диске, не в памяти, как /proc, inode указывает на то место в дисковом пространстве, где размещена индексная запись (index node, сокращенно -- inode) о файле. Inode содержит все необходимые сведения о файле, например права доступа, указатель на первый блок с содержимым файла.

Начиная с ядра 3.10 больше не поддерживается функция create_proc_entry(). Вместо нее используются функции:

```
#include <linux/types.h>
#include <linux/fs.h>

extern struct proc_dir_entry *proc_create_data(const char *, umode_t, struct
                        proc_dir_entry *, const struct file_operations *, void
                        *);

struct proc_dir_entry *proc_create(const char *name, umode_t mode, struct
                proc_dir_entry *parent, const struct file_operations *proc_fops);
```

В то время как отдельные элементы proc_dir_entry должны были быть инициализированы в некоторых более ранних версиях ядра, разработчики, работающие с более новой версией ядра, используют

структуру, которая хорошо знакома из разработки драйверов, struct file_operations, чтобы назначить методы доступа : open(), read(), write().

Структура file_operations используется для определения обратных вызовов (call back) чтения и записи.

Рассмотрим пример, который отображает текущее значение jiffies (счетчик тиков) всякий раз, когда он меняется. Рассмотрим загружаемый модуль jif.c (для 64-разрядной системы):

```
#include <linux/module.h>
 #include <linux/init.h>
 #include <linux/kernel.h>

 #include <linux/fs.h>      // for basic filesystem
 #include <linux/proc_fs.h> // for the proc filesystem
 #include <linux/seq_file.h>// for sequence files
 #include <linux/jiffies.h> // for jiffies

 static struct proc_dir_entry* jif_file;

 static int
 jif_show(struct seq_file *m, void *v)
 {
     seq_printf(m, "%llu\n",
         (unsigned long long) get_jiffies_64());
     return 0;
 }

 static int
 jif_open(struct inode *inode, struct file *file)
 {
     return single_open(file, jif_show, NULL);
 }

 static const struct file_operations jif_fops = {
     .owner= THIS_MODULE,
     .open  = jif_open,
     .read  = seq_read,
     .llseek     = seq_lseek,
     .release    = single_release,
 };

 static int __init
 jif_init(void)
 {
     jif_file = proc_create("jif", 0, NULL, &jif_fops);
```

```
    if (!jif_file) {
        return -ENOMEM;
    }

    return 0;
}

static void __exit
jif_exit(void)
{
    remove_proc_entry("jif", NULL);
}

module_init(jif_init);
module_exit(jif_exit);

MODULE_LICENSE("GPL");
```

Как только модуль будет загружен, убедитесь, что теперь существует следующий файл proc:

```
# ls -l /proc/jif
 -r--r--r--. 1 root root 0 2009-08-10 19:48 /proc/jif
 #
```

Для получения значения текущего значения соответствующей переменной jiffies ядра:

```
$ cat /proc/jif
 4329225958
 $ cat /proc/jif
 4329226854
 $ cat /proc/jif
 4329227174
 $ cat /proc/jif
 4329227486
 $ cat /proc/jif
 4329227798
 $ cat /proc/jif
 4329228078
 $
```

Родительский аргумент может быть NULL для корня / proc root или нескольких других значений, в зависимости от того, где нужно разместить файл. В таблице 2 перечислены некоторые другие родительские proc_dir_entrys, которые можно использовать, а также их расположение в файловой системе.

Table 2. Shortcut переменных proc_dir_entry

proc_dir_entry	Filesystem location
proc_root_fs	/proc
proc_net	/proc/net
proc_bus	/proc/bus
proc_root_driver	/proc/driver

Можно также создавать каталоги в файловой системе / proc, используя proc_mkdir(), а также символические ссылки с proc_symlink(). Для простых /proc-записей, для которых требуется только функция чтения, используется create_proc_read_entry(), которая создает запись /proc и инициализирует функцию read_proc в одном вызове. Прототипы этих функций:

```
extern struct proc_dir_entry *proc_symlink(const char *,
                                           struct proc_dir_entry *, const char *);

extern struct proc_dir_entry *proc_mkdir(const char *, struct proc_dir_entry *);

struct proc_dir_entry *create_proc_read_entry( const char *name, mode_t mode,
                    struct proc_dir_entry *base, read_proc_t *read_proc, void *data );
```

Задание на лабораторную работу

Написать программу – загружаемый модуль ядра (LKM) – которая поддерживает чтение из пространства пользователя и запись в пространство пользователя. После загрузки модуля пользователь может загружать в него строки с помощью команды echo, а затем автоматически считывать их с помощью команды cat.

Функция init (например, init_fortune_module) выделяет пространство для «горшка с печеньем» используя vmalloc(), а затем очищает его с помощью memset().

```
cookie_pot = (char *)vmalloc( MAX_COOKIE_LENGTH );
  if (!cookie_pot)
{
   ret = -ENOMEM;
  } else
 {
```

```
   memset( cookie_pot, 0, MAX_COOKIE_LENGTH );
 proc_entry = create_proc_entry( "fortune", 0644, NULL );
   if (proc_entry == NULL)
{
     ret = -ENOMEM;
     vfree(cookie_pot);
     printk(KERN_INFO "fortune: Couldn't create proc entry\n");
   } else
{
     cookie_index = 0;
     next_fortune = 0;
  ...
}
```

Когда cookie_pot выделен и пуст, создается proc_dir_entry в корне / proc root, который называется, например, fortune. Когда proc_entry успешно создан, инициализируюся локальные переменные и структура proc_dir_entry.

Файл cookie_pot представляет собой страницу длиной (4 КБ) и управляется двумя индексами. Первый, cookie_index, определяет, где будет записываться следующий файл cookie. Переменная next_fortune определяет, где следующий файл cookie будет считаться для вывода. Я просто переношу next_fortune в начало, когда все состояния были прочитаны.

Запись нового файла cookie_pot : если буфер для записи не доступен, то возвращается -ENOSPC, который передается пользовательскому процессу. В противном случае это пространство существует, и используется функция copy_from_user() для копирования пользовательского буфера непосредственно в файл cookie_pot. Затем увеличивается файл cookie_index (в зависимости от длины пользовательского буфера) и NULL завершает строку. Наконец, возвращается количество символов, фактически записанных в файл cookie_pot.

```
ssize_t fortune_write( struct file *filp, const char __user *buff,

                    unsigned long len, void *data ) {
       ...
if (copy_from_user( &cookie_pot[cookie_index], buff, len )) {

   return -EFAULT;
  }
 ...
}
```

Чтение: поскольку объявляемый буфер уже находится в пространстве ядра, можно манипулировать им напрямую и использовать sprintf для записи. Если индекс next_fortune больше, чем cookie_index (следующая позиция для записи), то next_fortune уменьшается до нуля, что зацикливает буфер. После того, как строка будет записана в пользовательский буфер, индекс next_fortune увеличивается на длину последней написанной строки.

```
int fortune_read( char *page, char **start, off_t off,
                  int count, int *eof, void *data )
{
  int len;
       ...
len = sprintf(page, "%s\n", &cookie_pot[next_fortune]);
  next_fortune += len;
  return len;
}
```

Листинг 1. Демонстрация работы «fortune cookie» LKM

```
[root@plato]# insmod fortune.ko

[root@plato]# echo "Success is an individual proposition.  Thomas Watson" > /proc/fortune

[root@plato]# echo "If a man does his best, what else is there?  Gen. Patton" > /proc/fortune

[root@plato]# echo "Cats: All your base are belong to us.  Zero Wing" > /proc/fortune

[root@plato]# cat /proc/fortune

Success is an individual proposition.  Thomas Watson

[root@plato]# cat /proc/fortune

If a man does his best, what else is there?  Gen. Patton

[root@plato]#
```

В программе необходимо создать поддиректорию и символическую ссылку.