proc file system

reporter: cyj@os.nctu.edu.tw

outline

- Introduction
- One example
- Managing procfs entries
- Communicating with userland
- lookup()
- kernel version: 2.4.18

introduction

- virtual file system
  - exists only in memory
  - an interface to internal data structure
    - /proc/modules, /proc/interrupts, /proc/pid…
- file system structure
  - ext2 file system structure: inode
  - proc file system structure: proc_dir_entry

len = sprintf(page, "jiffies = %ld\n", jiffies);

example

```
module_init(init_procfs_example);
static int __init init_procfs_example(void) {
    /* create directory */
    example_dir = proc_mkdir(MODULE_NAME, NULL);
    example_dir->owner = THIS_MODULE;
    /* create jiffies using convenience function */
    jiffies_file = create_proc_read_entry("jiffies", 0444, example_dir,
            proc_read_jiffies, NULL);
    jiffies_file->owner = THIS_MODULE;
    /* create tty device */ /* (5, 0) already exist */
    tty_device = proc_mknod("tty", S_IFCHR | 0666, example_dir,
            MKDEV(5, 0));
    symlink->owner = THIS_MODULE;
```

proc_example

example

```
foo_file = create_proc_entry("foo", 0644, example_dir);
strcpy(foo_data.name, "foo");
strcpy(foo_data.value, "foo");
foo_file->data = &foo_data;
foo_file->read_proc = proc_read_foobar;
foo_file->write_proc = proc_write_foobar;
foo_file->owner = THIS_MODULE;
```

```
struct fb_data_t {
char name[FOOBAR_LEN + 1];
char value[FOOBAR_LEN + 1]; };
```

```
len = sprintf(page, "%s = '%s'\n",
fb_data->name, fb_data->value);
```

```
copy_from_user(fb_data->value, buffer,
len))
```

example

```
/* create symlink */
    symlink = proc_symlink("jiffies_too", example_dir, "jiffies");
    symlink->owner = THIS_MODULE;


module_exit(cleanup_procfs_example);
static void __exit cleanup_procfs_example(void){
    remove_proc_entry("jiffies_too", example_dir);
    remove_proc_entry("tty", example_dir);
    remove_proc_entry("bar", example_dir);
    remove_proc_entry("foo", example_dir);
    remove_proc_entry("jiffies", example_dir);
    remove_proc_entry(MODULE_NAME, NULL);
}
```

output

```
> /sbin/insmod proc_example.o
> cd /proc/procfs_example/
> ls
    bar  foo  jiffies  jiffies_too  tty
> cat jiffies
    jiffies = 27626141
> cat jiffies_too
    jiffies = 27626442
> cat foo
    foo = 'foo‘
> echo –n ‘‘abc’’ > foo
> cat foo
    foo = ‘abc‘
```

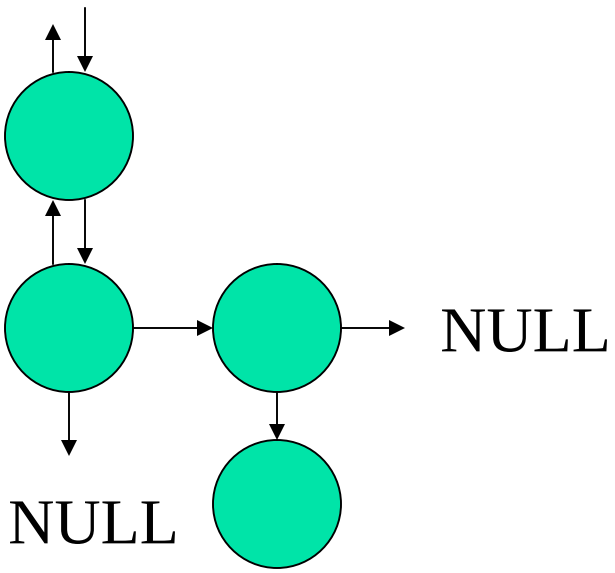
Managing procfs entries

struct proc_dir_entry

```
struct proc_dir_entry {
  unsigned short low_ino;                    /* inode number */
  unsigned short namelen;
  const char name;
  mode_t mode;
  nlink_t nlink                    /* file( nlink = 1), dir(nlink > 1) */
  uid_t uid;
  gid_t gid;
  unsigned long size;
  struct inode_operations * proc_iops;
  struct file_operations * proc_fops;
  get_info_t *get_info;
  struct module *owner;            // entry->owner = THIS_MODULE;
  struct proc_dir_entry *next, *parent, *subdir;        /* internal structure */
  void *data;
```

struct proc_dir_entry

```
read_proc_t *read_proc;
write_proc_t *write_proc;
atomic_t count;        /* use count */
int deleted;           /* delete flag */
kdev_t  rdev;};
```



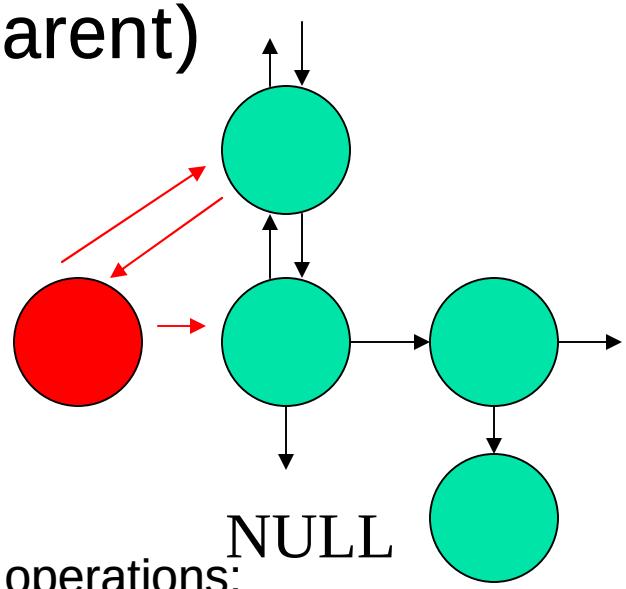
inode allocations in proc-fs

00000000	reserved
00000001-00000fff (goners)	static entries
001	root-ino
00001000-00001fff	dynamic entries
0001xxxx-7fffxxxx pid 1-7fff	pid-dir entries for
80000000-ffffffff	unused

static int proc_register(struct proc_dir_entry * dir, struct proc_dir_entry * dp);

- add dp(child) to dir(parent)

```
{
 i = make_inode_number();
dp->low_ino = i;
dp->next = dir->subdir;
dp->parent = dir;
dir->subdir = dp;
if (S_ISDIR(dp->mode)) {
    if (dp->proc_iops == NULL) {
        dp->proc_fops = &proc_dir_operations;
        dp->proc_iops = &proc_dir_inode_operations;
    }
    dir->nlink++;
```



static int proc_register(struct proc_dir_entry * dir, struct proc_dir_entry * dp);

```
} else if (S_ISLNK(dp->mode))
{
    if (dp->proc_iops == NULL)
        dp->proc_iops = &proc_link_inode_operations;
} else if (S_ISREG(dp->mode)) {
    if (dp->proc_fops == NULL)
        dp->proc_fops = &proc_file_operations;
}
return 0;}
```

static int make_inode_number(void)

- provide an unused inode number

```
static unsigned long proc_alloc_map[(PROC_NDYNAMIC +
    BITS_PER_LONG - 1) / BITS_PER_LONG];

static int make_inode_number(void)
{
    i = find_first_zero_bit(proc_alloc_map, PROC_NDYNAMIC);
    set_bit(i, proc_alloc_map);
    i += PROC_DYNAMIC_FIRST;
    return i;
}
```

static struct proc_dir_entry *proc_create(struct proc_dir_entry **parent, const char *name, mode_t mode, nlink_t nlink)

- **Creating a regular file**

```
if (!(*parent) && xlate_proc_name(name, parent, &fn) != 0)
    goto out;
ent = kmalloc(sizeof(struct proc_dir_entry) + len + 1,
    GFP_KERNEL);
memset(ent, 0, sizeof(struct proc_dir_entry));
memcpy(((char *) ent) + sizeof(struct proc_dir_entry), fn, len + 1);
ent->name = ((char *) ent) + sizeof(*ent);
ent->namelen = len;
ent->mode = mode;
ent->nlink = nlink;
return ent;
```

static int xlate_proc_name(const char *name, struct proc_dir_entry **ret, const char **residual)

- Not important
- eg: name: tty/driver/serial, return /proc/tty/driver proc_dir_entry && serial

```
struct proc_dir_entry *proc_symlink(const char *name,
struct proc_dir_entry *parent, const char *dest)
```

- **Creating a symlink**

```
ent = proc_create(&parent,name, (S_IFLNK | S_IRUGO |
    S_IWUGO | S_IXUGO),1);

ent->data = kmalloc((ent->size=strlen(dest))+1, GFP_KERNEL);
strcpy((char*)ent->data,dest);
proc_register(parent, ent);
return ent;
```

struct proc_dir_entry *proc_mknod(const char *name, mode_t mode, struct proc_dir_entry *parent, kdev_t rdev)

- Creating a device

```
ent = proc_create(&parent,name,mode,1);
ent->rdev = rdev;
proc_register(parent, ent);
```

struct proc_dir_entry *proc_mkdir(const char *name, struct proc_dir_entry *parent)

- **Creating a directory**

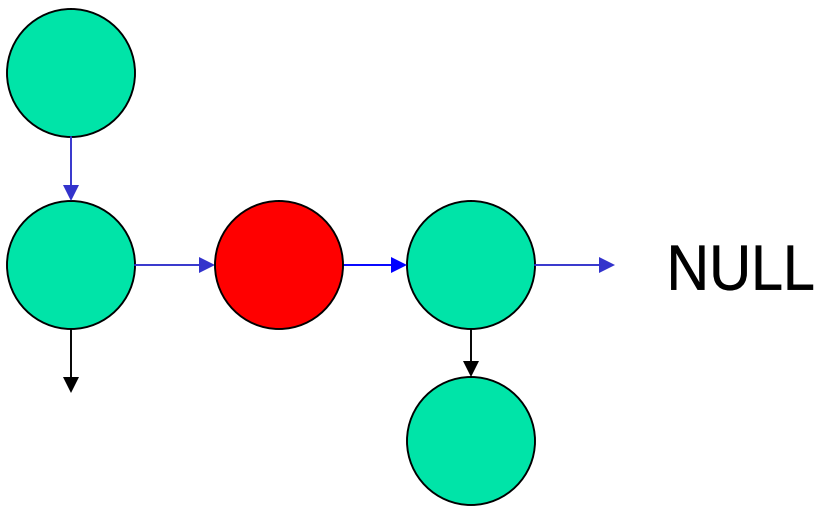
```
ent = proc_create(&parent,name, (S_IFDIR | S_IRUGO |
    S_IXUGO),2);
ent->proc_fops = &proc_dir_operations;
ent->proc_iops = &proc_dir_inode_operations;
proc_register(parent, ent);
```

void remove_proc_entry(const char *name, struct proc_dir_entry *parent)

- **Removing an entry**

```
for (p = &parent->subdir; *p; p=&(*p)->next ) {
    if (!proc_match(len, fn, *p)) continue;
    de = *p; *p = de->next; de->next = NULL;
    if (S_ISDIR(de->mode)) parent->nlink--;
    clear_bit(de->low_ino - PROC_DYNAMIC_FIRST, proc_alloc_map);
    proc_kill_inodes(de); de->nlink = 0;
    if (!atomic_read(&de->count)) free_proc_entry(de);
    else { de->deleted = 1;
            printk("remove_proc_entry: %s/%s busy, count=%d\n", parent-
    >name,          de->name, atomic_read(&de->count)); }
    break;
}
```

pointer to pointer



Communicating with userland

- Instead of reading (or writing) information directly from kernel memory, procfs works with *call back functions* for files: functions that are called when a specific file is being read or written.

```
struct proc_dir_entry* entry;
entry->read_proc = read_proc_foo;
entry->write_proc = write_proc_foo;
```

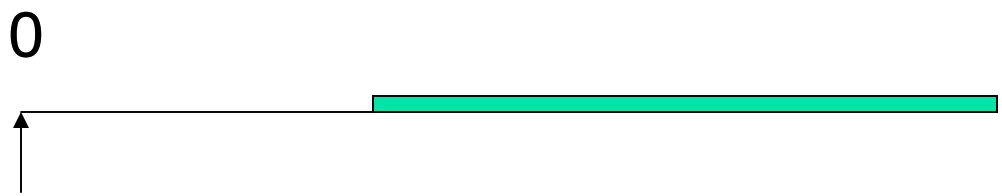
int (*read_proc)(char *page, char **start, off_t offset, int count, int *eof, void *data);

- page
- offset
- count
- eof
- start
  - The start argument is there to implement large data files, but it can be ignored.
- data
  - **A single call back for many files**

example of a single call back for many files

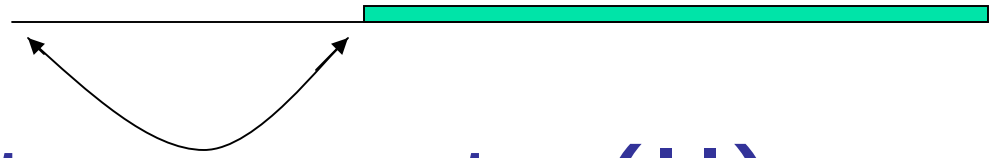
```
struct proc_dir_entry* entry;
struct my_file_data *file_data;
file_data = kmalloc(sizeof(struct my_file_data), GFP_KERNEL);
entry->data = file_data;

int foo_read_func(char *page, char **start, off_t off, int count, int
    *eof, void *data) {
    int len;
    if(data == file_data) { /* special case for this file */ }
    else { /* normal processing */ }
    return len; }
```



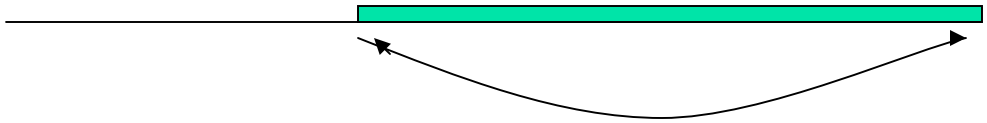
start parameter(I)

- *start = NULL
  - for files which are no longer than the buffer
  - ignore offset parameter
  - if not EOF, call again with the requested offset advanced by the number of bytes absorbed



start parameter(II)

- *start = an unsigned long value less than the buffer address but greater than zero(number of tokens)
  - for a large file consisting of a series of blocks which you want to count and return as wholes, Eg. array of structures
  - the caller will use it to increment filp->f_pos independently of the amount of data you return, thus making f_pos an internal record number of your *read_proc* or *get_info*procedure
  - Put the data of the requested offset at the beginning of the buffer
  - if not EOF, call again with the requested offset advanced by *start



start parameter(III)

- *start = an address within the buffer(address)
  - Put the data of the requested offset at *start.
  - If not EOF, call again with the requested offset advanced by the number of bytes absorbed.

static ssize_t proc_file_read(struct file * file, char * buf, size_t nbytes, loff_t *ppos)

```
while ((nbytes > 0) && !eof) {
    count = MIN(PROC_BLOCK_SIZE, nbytes);
    n = dp->read_proc(page, &start, *ppos, count, &eof, dp->data);
    if (!start) {
        /* * For proc files that are less than 4k */
        start = page + *ppos; n -= *ppos;
    }

    n -= copy_to_user(buf, start < page ? page : start, n);  // return #
    uncopied data
    *ppos += start < page ? (long)start : n; /* Move down the file */
    nbytes -= n; buf += n; retval += n;
}
```

static ssize_t proc_file_write(struct file * file, const char * buffer, size_t count, loff_t *ppos)

```
struct inode *inode = file->f_dentry->d_inode;
struct proc_dir_entry * dp;

dp = (struct proc_dir_entry *) inode->u.generic_ip;

return dp->write_proc(file, buffer, count, dp->data);
```

static loff_t proc_file_lseek(struct file * file, loff_t offset, int orig)

```
case 0:
    if (offset < 0) return -EINVAL;
    file->f_pos = offset;
    return(file->f_pos);
case 1:
    if (offset + file->f_pos < 0) return -EINVAL;
    file->f_pos += offset;
    return(file->f_pos);
```

default file_operations

```
static struct file_operations proc_file_operations = {
    llseek: proc_file_lseek,
    read: proc_file_read,
    write: proc_file_write,
};
```

struct dentry *proc_lookup(struct inode * dir, struct dentry *dentry)

```
de = (struct proc_dir_entry *) dir->u.generic_ip;
for (de = de->subdir; de ; de = de->next) {
    …
    if (!memcmp(dentry->d_name.name, de->name, de->namelen))
    {
        int ino = de->low_ino;
        inode = proc_get_inode(dir->i_sb, ino, de);
    break;
    }
}

d_add(dentry, inode);
```

```
static struct inode_operations
proc_dir_inode_operations = {
        lookup: proc_lookup, };
```

struct inode * proc_get_inode(struct super_block * sb, int ino, struct proc_dir_entry * de)

```
inode = iget(sb, ino);
inode->u.generic_ip = (void *) de;

inode->i_mode = de->mode;
inode->i_uid = de->uid;
inode->i_gid = de->gid;

inode->i_size = de->size;

inode->i_nlink = de->nlink;
…
return inode;
```

References

- http://lxr.linux.no/source/fs/proc
- http://www.kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html
- http://www.xml.com/ldd/chapter/book/ch04.html