Using the proc filesystem

Lecturer:
Ing. Luca Pizzamiglio
Politecnico di Milano, DEI
luca.pizzamiglio@gmail.com

© 2005 - Luca Pizzamiglio

Overview

- Introduction
- Managing procfs entries
- Communicating with userland
- Tips and tricks

 © 2005 - Luca Pizzamiglio

The proc filesystem

- `proc` is a virtual filesystem
  - It is not associated with a block device but exists only in memory
- Files in procfs allow userland programs to access certain information from the kernel
  - Debug purposes
- `/proc/sys` are sysctl files
  - They don't belong to procfs and are governed by a completely different API

 © 2005 - Luca Pizzamiglio

Managing procfs entries (1/3)

- If you want to use any of the procfs functions, be sure to include the correct header file

```
#include <linux/proc_fs.h>
```

- Creating a regular file

```
struct proc_dir_entry* create_proc_entry(const char*
  name, mode_t mode, struct proc_dir_entry* parent);
```

  - NULL as *parent* parameter to create the file in the root of the proc
  - It returns a pointer to the freshly created struct
    - NULL if some error has happened
  - It is possible to pass a path that spans multiple directories

```
create_proc_entry("foo/bar/info")
```

    - It creates the bar directory, if necessary, with 755 permissions
  - `create_proc_read_entry` to be able only to read the file

 © 2005 - Luca Pizzamiglio

Managing procfs entries (2/3)

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

- It creates a symlink in the procfs directory parent that points from *name* to *dest*

```
ln -s name dest
```

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

- It creates a device file *name* with mode *mode* in the procfs directory *parent*
  - *Mode* parameter must contain S_IFBLK or S_IFCHR

```
mknod --mode=mode
```


© 2005 - Luca Pizzamiglio

Managing procfs entries (3/3)

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

- It creates a directory *name* in *parent*

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

- It removes the entry *name* in the directory *parent*
  - Entries are removed by name
  - This function doesn't recursively remove entries
    - Be sure to free the *data* entry from the struct proc_dir_entry

© 2005 - Luca Pizzamiglio

Communicating with userland

- Procfs works with callback functions for files

```
struct proc_dir_entry* entry;

entry -> read_proc = read_proc_foo;
entry -> write_proc = write_proc_foo;
```

- `create_proc_read_entry` creates and initialize the procfs entry in one single call

 © 2005 - Luca Pizzamiglio

Reading and writing

```
int read_func(char* page, char** start, off_t off, int
   count, int* eof, void *data)
```

- It is used to read data from the kernel
- Writes its information to *page*
  - It should start writing at *off* and write at most *count* bytes
  - *eof* used to signal that the end of the file has been reached (1 in the memory location it points to)
  - It returns the number of bytes written into the page

```
int write_func(struct file* file, const char* buffer,
   long count, void *data)
```

- It should read *count* bytes at maximum from the *buffer*
- Buffer doesn't live in kernel memory space
  - `copy_from_user` to copy it

© 2005 - Luca Pizzamiglio

A single callback

- Useful when a large number of almost identical files is used
  - It distinguish between files using the *data* field in struct proc_dir_entry
    - It is a void*, so it can be initialized with anything
    - Be sure to free data when removing the procfs entry

```
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
```

 © 2005 - Luca Pizzamiglio

Inside the callback

```
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 */
   } else {
       /* normal processing */
   }
   return len;
}
```


© 2005 - Luca Pizzamiglio

Tips and tricks

- `create_proc_read_entry` to create and initialize an entry with a single call
- If procfs is being used from within a module, be sure to set the owner field in the struct `proc_dir_entry` to `THIS_MODULE`

```
struct proc_dir_entry* entry;
entry->owner = THIS_MODULE;
```

- Change the mode and/or ownership of the entry

```
struct proc_dir_entry* entry;
entry->uid = 0;
entry->mode = S_IWUSR | S_IRUSR | S_IRGRP
  | S_IROTH;
```


© 2005 - Luca Pizzamiglio