# Operating System Design & Implementation

Lab 12: File system driver (Ramfs)

TA: 周昆霖、吳崧銘

### Objective:

#### In this Lab, you can learn

* Communicating with kernel code via /proc file system.
* Understanding how ramfs handles read/write requests.
* Encrypting and decrypting files in ramfs

### Experiment:

* 12-1 Handling /proc access requests in kernel code.
* 12-2 Registering your own “pass through” handlers of file read/write in ramfs.
* 12-3 Implementing simple file encryption/decryption using XOR operations.

Exp. 12-1 Handling /proc access requests in kernel code

* + /proc file system provides a simple interface between kernel code and user program. Kernel code handles read/write requests from user programs via the “fake files” in the /proc directory.
  + An example is provided in <http://tuxthink.blogspot.tw/2011/02/creating-readwrite-proc-entry.html>

Step1: Register a proc file using *create\_proc entry()*, which takes 3 arguments:

*proc file name, file permissions, pathname of the file*

It returns a pointer to structure ‘struct proc\_dir\_entry’, in which you can register your own read/write handlers of your proc file.

For example, you can register a proc file in init\_ramfs\_fs() of fs/ramfs/inode.c with the following

proc\_entry = create\_proc\_entry( “flag”, 0644, NULL);

The above command will create the ‘flag’ entry, and you assign

‘NULL’ to the path so the new file will be /proc/flag

Step2: To remove a file from /proc, use the *remove\_proc\_entry()* function.

Add it in exit\_ramfs\_fs().

Step3: Create two functions *my\_read()* and *my\_write()* to handle read/write requests to /proc/flag. Register your callback functions after creating the proc file, as follows:

proc\_entry->read\_proc = my\_read;

proc\_entry->write\_proc = my\_write;

Step4: When the proc file is read or written by a user program, the proc file handler will be called and a pointer to the user space buffer is provided.

Notice: Use *copy\_from\_user()* and *copy\_to\_user()* in your handlers to access the user buffer, as the buffer may not be directly accessible in the kernel code.

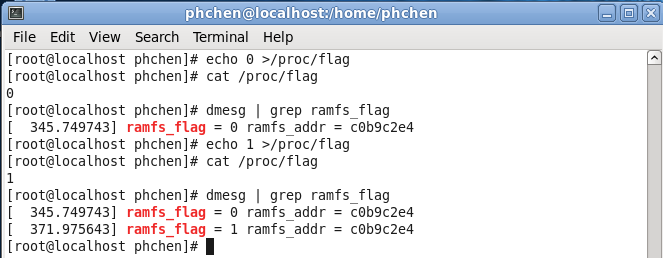
Step5: You may need to include some header files, such as asm/uaccess.h, linux/string.h, linux/vmalloc.h, and linux/proc\_fs.h

Step6: Implement your own *my\_read()* and *my\_write()*. Add a global variable, i.e., ramfs\_flag, to indicate whether encryption is enabled.

e.g. if /proc/flag is written with “1”, then set *ramfs\_flag = 1* in *my\_write()*. When /proc/flag is read, return the value of flag in my\_read().

Exp. 12-1 Demonstration:

Your kernel code must print out the flag value in my\_read() so you can check whether the flag value has been changed successfully.



Note : Put your proc handler in inode.c.

Exp. 12-2 Registering your own “pass through” handlers of file read/write in ramfs.

#### In this lab, you will register your “pass through” handlers for ramfs read/write.

Step1: Mount command : mount –t ramfs none [DIRECTORY]

e.g. mount –t ramfs none /mnt/ramfs

Step2: You should modify two source files:

“fs/ramfs/file-mmu.c”, and “fs/ramfs/internal.h.”

* In file-mmu.c : replace the ramfs read/write handlers with your own “pass through” handlers

.

e.g. modify “.aio\_read = generic\_file\_aio\_read” to “.aio\_read = my\_aio\_read”

and “aio\_write =“generic\_file\_aio\_write” to

“.aio\_write = my\_aio\_write”

Note:

1. your handler must call the original ramfs handlers so that requests will “pass through” your handlers.

2. you may have to include </linux/uio.h> for access to struct iovec

Step3: Print some messages using printk in your handlers to verify whether your handlers are called correctly.

Exp. 12-2 Demonstration: 

Exp. 12-3 Implementing simple file encryption and decryption using XOR operations.

* Adding encryption and decryption code in the pass through handlers

Step1: Set ramfs\_flag = 1 (by writing to /proc/flag)

Step2: Write “hello” to a file “test”. Each character should be XOR’ed with a key value (0x25 in this example)

Step3: Set ramfs\_flag = 0 to turn off the encryption (by writing to /proc/flag)

Step4: Read the file, its content should be “M@IIJ”

Step5: Turn on encryption again

Step6: Read the file, its content should be “hello”

Note:

1. You should declare “ramfs\_flag” in ramfs driver as a global variable.
2. For read, your ramfs driver XOR data before returning to user space.
3. For write, your ramfs driver XOR data before calling the original write handler.

Exp. 12-3 Demonstration:

