

Making a File System Module

Practical Class 6

**Systems and Storage Laboratory
Department of Computer Science and Engineering
Chung-Ang University**

Making a File System Module

- ❖ **Modules are pieces of code that can be loaded and unloaded into the kernel upon demand**
 - We aim to make a **file system** module
 - We can load or unload a file system with modification to the file system without kernel compile
 - The file system module should not be related with the running file system inside kernel
 - ✓ We need to rename the file system or source code inside file system
 - This makes implementation or modification for a file system easy

Making a File System Module

❖ Making a file system module

- Copying existing EXT4 file system module to my file system module (PXT4 file system)
- EXT4 file system includes two modules
 - ext4.ko
 - ✓ Body of file system (metadata, I/O operations)
 - jbd2.ko
 - ✓ Transaction processing

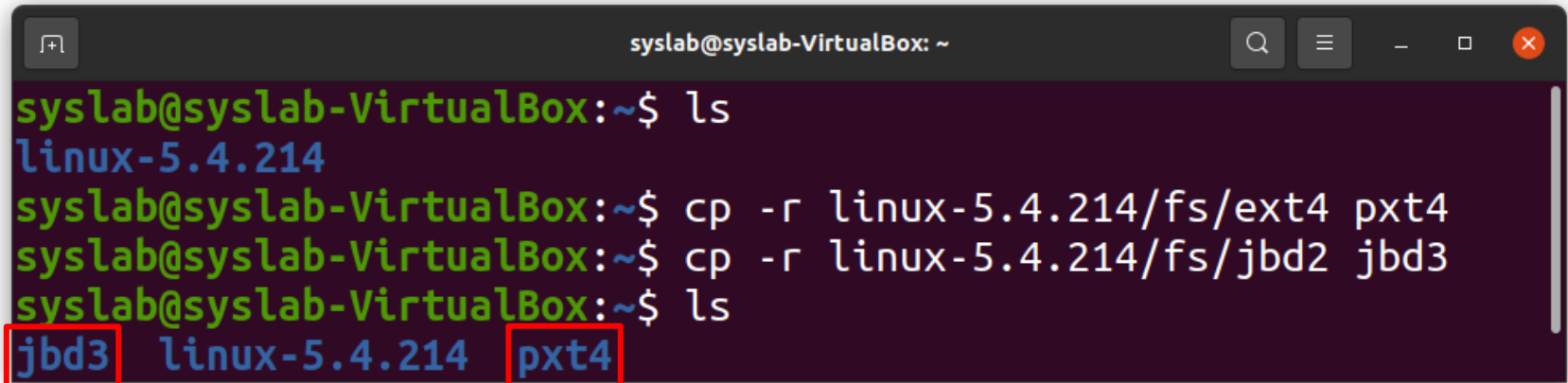
❖ To avoid conflict

- We change ext4.ko → pxt4.ko
- We change jbd2.ko → jbd3.ko

Copy original file system to our file system

- ❖ Copy the original ext4 and jbd2 to pxt4 and jbd3, respectively

```
$ cp -r <where linux-kernel is>/fs/ext4 pxt4  
$ cp -r <where linux-kernel is>/fs/jbd2 jbd3
```



```
syslab@syslab-VirtualBox: ~  
syslab@syslab-VirtualBox:~$ ls  
linux-5.4.214  
syslab@syslab-VirtualBox:~$ cp -r linux-5.4.214/fs/ext4 pxt4  
syslab@syslab-VirtualBox:~$ cp -r linux-5.4.214/fs/jbd2 jbd3  
syslab@syslab-VirtualBox:~$ ls  
jbd3 linux-5.4.214 pxt4
```

Modify codes

❖ Use 'cscope' to change some codes in files

```
/[pxt4 or jbd3 directory]$ cscope
```

- **Ctrl + A** : to select all the lines to change
- **Ctrl + D** : confirm and exit

```
Find this C symbol:  
Find this global definition:  
Find functions called by this function:  
Find functions calling this function:  
Find this text string:  
Change this text string: jbd2  
Find this egrep pattern:  
Find this file:  
Find files #including this file:  
Find assignments to this symbol:  
To: jbd3
```

- In each pxt4 and jbd3 directory, you have to change:
 - “jbd2” → “jbd3”, “ext4” → “pxt4”, “ext2” → “pxt2”
 - “JBD2” → “JBD3”, “EXT4” → “PXT4”, “EXT2” → “PXT2”

cscope checklist

❖ In **jbd3** directory:

1. jbd2 → jbd3
2. ext4 → pxt4
3. ext2 → pxt2
4. JBD2 → JBD3
5. EXT4 → PXT4
6. EXT2 → PXT2

❖ In **pxt4** directory:

7. jbd2 → jbd3
8. ext4 → pxt4
9. ext2 → pxt2
10. JBD2 → JBD3
11. EXT4 → PXT4
12. EXT2 → PXT2

Modify file name

❖ Use 'mv' to change name of files

```
/[pxt4 or jbd3 directory]$ mv <original name> <new name>
```

1. jbd2 → jbd3
2. ext4 → pxt4
3. JBD2 → JBD3
4. EXT4 → PXT4

❖ The result may look like these:

```
syslab@syslab-VirtualBox:~/jbd3$ ls
checkpoint.c  cscope.out  Kconfig     recovery.c  transaction.c
commit.c      journal.c   Makefile    revoke.c
```

```
syslab@syslab-VirtualBox:~/pxt4$ ls
acl.c          fsmap.h      migrate.c    symlink.c
acl.h          fsync.c      mmp.c        sysfs.c
balloc.c       hash.c       move_extent.c truncate.h
bitmap.c       ialloc.c     namei.c      verity.c
block_validity.c indirect.c   page-io.c    xattr.c
cscope.out     inline.c     pxt4_extents.h xattr.h
dir.c          inode.c      pxt4.h       xattr_security.c
extents.c      ioctl.c      pxt4_jbd3.c  xattr_trusted.c
extents_status.c Kconfig      pxt4_jbd3.h  xattr_user.c
extents_status.h Makefile     readpage.c
file.c         mballocc.c  resize.c
fsmapp.c       mballocc.h  super.c
```

Modify codes

❖ In `/[pxt4 dir]$ vim acl.h`

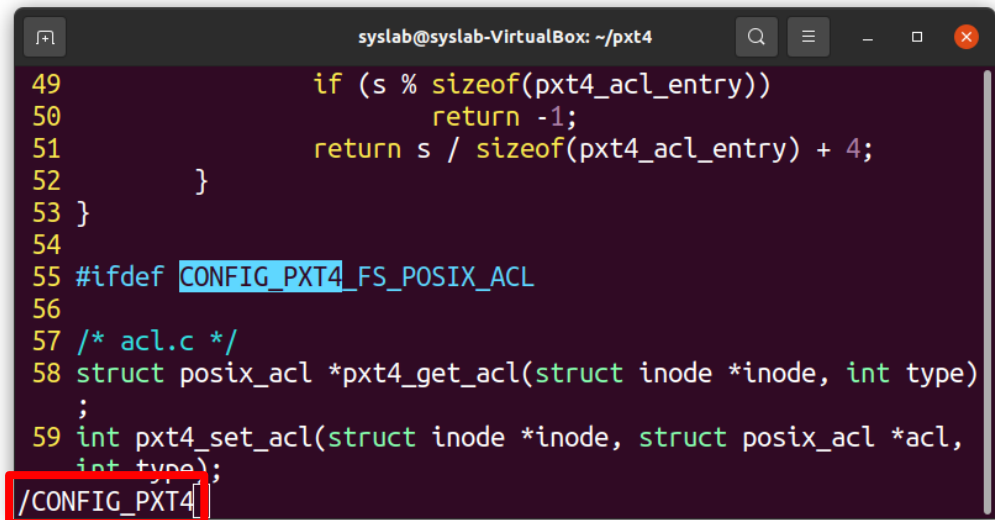
- Find and change
 - `#ifdef CONFIG_PXT4` → `#ifdef CONFIG_EXT4`

❖ In `/[pxt4 dir]$ vim xattr.h`

- Find and change
 - `#ifdef CONFIG_PXT4` → `#ifdef CONFIG_EXT4`

❖ In vim, you can easily find a text string with `/` (slash)

- `n`: find next
- `N`: find prev



The screenshot shows a terminal window with a vim editor. The code is displayed with line numbers 49 to 59. Line 55 contains the text `#ifdef CONFIG_PXT4_FS_POSIX_ACL`, where `CONFIG_PXT4` is highlighted in blue. At the bottom of the editor, a red rectangular box highlights the search command `/CONFIG_PXT4`.

```
49         if (s % sizeof(pxt4_acl_entry))
50             return -1;
51         return s / sizeof(pxt4_acl_entry) + 4;
52     }
53 }
54
55 #ifdef CONFIG_PXT4_FS_POSIX_ACL
56
57 /* acl.c */
58 struct posix_acl *pxt4_get_acl(struct inode *inode, int type)
59 ;
60 int pxt4_set_acl(struct inode *inode, struct posix_acl *acl,
61 int type);
62
63 /CONFIG_PXT4
```


Modify codes

❖ In `/[pxt4 dir]$ vim pxt4.h`

■ Find and change

- `#define EXT2` → `#define PXT2`
- `#define EXT3` → `#define PXT3`
- `cpu_to_le32(~EXT## ...)` → `cpu_to_le32(~PXT## ...)`

```
1880         return ((PXT4_SB(sb)->s_es->s_feature_compat & \
1881                 cpu_to_le32(~EXT##ver##_FEATURE_COMPAT_SUPP)) != 0); \
1882     } \
1883     static inline bool pxt4_has_unknown_ext##ver##_ro_compat_features(struct s
        uper_block *sb) \
        /cpu_to_le32(~EXT##
```

❖ In `/[pxt4 dir]$ vim super.c`

■ Find and change

- `pxt4_has_unknown_pxt4_incompat_`
→ `pxt4_has_unknown_ext4_incompat_`
- `pxt4_has_unknown_pxt4_ro_compat_`
→ `pxt4_has_unknown_ext4_ro_compat_`

Copy header files into linux directory

❖ In `/[linux kernel directory]/include/linux`

```
$ cp jbd2.h jbd3.h  
$ cp journal-head.h journal-head3.h
```

```
syslab@syslab-VirtualBox:~/linux-5.4.214/include/linux$ cp jbd2.h jbd3.h  
syslab@syslab-VirtualBox:~/linux-5.4.214/include/linux$ cp journal-head.h journal-head3.h
```

❖ In `/[linux kernel directory]/include/trace/events`

```
$ cp jbd2.h jbd3.h  
$ cp ext4.h pxt4.h
```

```
syslab@syslab-VirtualBox:~/linux-5.4.214/include/trace/events$ cp jbd2.h jbd3.h  
syslab@syslab-VirtualBox:~/linux-5.4.214/include/trace/events$ cp ext4.h pxt4.h
```

Modify codes in header files

- ❖ Find and change codes in each jbd3.h, pxt4.h and journal-head3.h
 - “jbd2” → “jbd3”, “ext4” → “pxt4”
 - “JBD2” → “JBD3”, “EXT4” → “PXT4”
- ❖ You can specify multiple files or directories to cscope at once

```
$ cscope <filename1> <filename2> ...
```

```
syslab@syslab-VirtualBox:~/linux-5.4.214/include/linux$ cscope jbd3.h journal-head3.h
```

```
syslab@syslab-VirtualBox:~/linux-5.4.214/include/trace/events$ cscope jbd3.h pxt4.h
```

Modify codes in header files

- ❖ In `/[linux kernel directory]/include/linux$ vim jbd3.h`
 - `#include <linux/journal-head.h>`
→ `#include <linux/journal-head3.h>`
- ❖ In `/[linux kernel directory]/include/uapi/linux$ vim magic.h`
 1. Copy line `#define EXT4_SUPER_MAGIC 0xEF53`
 2. Change “EXT4_SUPER_MAGIC” → “PXT4_SUPER_MAGIC”

```
#define EXT4_SUPER_MAGIC      0xEF53  
#define PXT4_SUPER_MAGIC     0xEF53
```

Modify Makefile in PXT4

❖ In `/[pxt4 dir]$ vim Makefile`

```
1 # SPDX-License-Identifier: GPL-2.0
2 #
3 # Makefile for the linux pxt4-filessystem routines.
4 #
5
6 obj-m += pxt4.o
7
8 pxt4-y := balloc.o bitmap.o block_validity.o dir.o pxt4_jbd3.o extents.o \
9         extents_status.o file.o fsmap.o fsync.o hash.o ialloc.o \
10        indirect.o inline.o inode.o ioctl.o mballoc.o migrate.o \
11        mmp.o move_extent.o namei.o page-io.o readpage.o resize.o \
12        super.o symlink.o sysfs.o xattr.o xattr_trusted.o xattr_user.o
13
14 pxt4-m += acl.o
15 pxt4-m += xattr_security.o
16 pxt4-m += verity.o
17
18 KDIR    := /lib/modules/$(shell uname -r)/build
19 PWD     := $(shell pwd)
20
21 default:
22     $(MAKE) -C $(KDIR) M=$(PWD) modules
23
24 clean:
25     rm *.mod.*
26     rm *.ko
27     rm *.o
```

Modify Makefile in JBD3

❖ In `/[jbd3 dir]$ vim Makefile`

```
1 # SPDX-License-Identifier: GPL-2.0-only
2 #
3 # Makefile for the linux journaling routines.
4 #
5
6 obj-m += jbd3.o
7
8 jbd3-objs := transaction.o commit.o recovery.o checkpoint.o revoke.o journal.o
9
10 KDIR      := /lib/modules/$(shell uname -r)/build
11 PWD       := $(shell pwd)
12
13 default:
14     $(MAKE) -C $(KDIR) M=$(PWD) modules
15
16 clean:
17     rm *.mod.*
18     rm *.ko
19     rm *.o
```

~
~
~

Move the directory and compile

❖ pxt4 refers to jbd3

- We need to move jbd3 directory into pxt4 directory

```
/[working directory]$ mv jbd3 ./pxt4
```

❖ Compile jbd3 module first

```
/[jbd3 dir]$ make
```

Copy module.symvers

- ❖ Copy 'Module.symvers' file from jbd3 directory into pxt4 directory

```
/[jbd3 dir]$ cp Module.symvers ../
```

- ❖ Now compile pxt4 module

```
/[pxt4 dir]$ make
```


Insert modules

❖ insmod command

- Inserts the module into kernel
- The version of kernel and module should be same
 - If not, there will be warning of '*invalid module format*'.

❖ Insert modules by following order

```
/[jbd3 dir]$ insmod jbd3.ko  
/[pxt4 dir]$ insmod pxt4.ko
```

❖ To check whether the modules are inserted, use

```
$ lsmod
```

```
syslab@syslab-VirtualBox:~/pxt4$ lsmod  
Module                Size  Used by  
pxt4                   757760  0  
jbd3                   118784  1 pxt4
```

Making file system for a device

❖ mkfs command

- Formats a device into the specified file system

```
$ mkfs [options] [-t <file system type>] <device name>
```

- Ex) `mkfs -t ext4 /dev/sdb`

❖ Watch out!

- **mkfs will erase all the data in the device.**
- **Make sure you are formatting the right device**

❖ Create a directory as mount point

```
/mnt$ mkdir <directory name>
```

Mounting a file system into a directory

❖ mount command

- Mounts a device into a directory

```
$ mount [options] <target device> <target directory>
```

❖ Mount your device with pxt4

```
/[pxt4 dir]$ sudo mount -t pxt4 <device name> /mnt/<directory name>
```

- Ex) mount -t pxt4 /dev/sdb /mnt/test

❖ To check whether the device is mounted, use

```
$ mount -l
```

Function Pointer

Appendix

Concept

❖ Function pointer

- Function pointers are like normal pointers, which have the capability to point to the address of a function. Simply, the '*function pointer*' is a pointer that points to a function.
- But, as opposed to referencing a data value, a function pointer points to executable code within memory. As to say, function pointer stores the starting address of executable code(function).
- Dereferencing the function pointer yields the referenced function, and the referenced function can be invoked, passed arguments just as in a normal function call.
 - Also known as an "indirect" call
- Also, unlike normal pointers, we do not allocate/de-allocate memory using function pointers.
- Even when calling the same function pointer, which function will be executed is determined by the connected address at runtime.

Basic usage

❖ Declaring function pointer

- Function pointer declaration

```
void (*func_ptr)(int, char*);
```

return type

name of function pointer

function parameters

- As you can see, the declaration of the *function pointer* looks similar to the *function prototype* in C.
- But the major difference is that the *function prototype* has no *asterisk sign* * before the function name, and also with no *parenthesis* () for the function name.
- So, do not confuse between **function pointer** with the *function prototype*. These two are just similar in shape.

Basic usage

❖ Comparison with normal function

■ Normal function declare

```
void func(int);
```

- The function `func()` takes an `int` parameter and returns `void` type (or nothing)

■ Function pointer declare

```
void (*func_ptr)(int);
```

- The function pointer `func_ptr` can be mapped with a function that takes an `int` parameter and returns `void` type
- **Caution)** without parentheses, `func_ptr` will become a function that returns `void *`

```
void *func_ptr(int);
```

return type

Basic usage

❖ Initializing function pointer

- All work the same way
 - Rvalue `func` is the address of function `func()`
 - No difference between `func` and `&func`
 - Asterisk marks (`*`) are ignored

```
func_ptr = func;  
func_ptr = &func;  
func_ptr = *func;  
func_ptr = *****func;
```

- Declare and initialize at the same time

```
void (*func_ptr)(int) = func;
```


Basic usage

❖ Calling a function

```
func(10);
```

❖ Calling a function with the function pointer

- `func_ptr` points to function `func()`
- Just like normal data pointers (`int *`, `char *`, etc), a function pointer with `*` on it dereferences function address `func`
- Parentheses and asterisk marks can be omitted
 - All work the same way

```
(*func_ptr)(10);  
func_ptr(10);
```

Basic usage

❖ Example

```
#include <stdio.h>

void func(int);

int main(void)
{
    void (*func_ptr)(int) = &func;

    (*func_ptr)(10);
    func(10);

    return 0;
}

void func(int a)
{
    printf("%d\n", a);
}
```

- Result output:



```
10
10
```

Usage Example - 1

❖ Declare function pointers inside struct

```
struct address_space_operations {  
    ...  
    int (*write_begin)(struct file *, struct address_space *mapping,  
                        loff_t pos, unsigned len, unsigned flags,  
                        struct page **pagep, void **fsdata);  
    int (*write_end)(struct file *, struct address_space *mapping,  
                     loff_t pos, unsigned len, unsigned copied,  
                     struct page *page, void *fsdata);  
    ...  
};
```

from include/linux/fs.h

- In Linux kernel, there is a struct that contains various function pointers
- These function pointers will be mapped into real functions dynamically

Usage Example - 1

❖ Map function pointers with real functions

```
static const struct address_space_operations ext4_aops = {  
    ...  
    .write_begin      = ext4_write_begin,  
    .write_end        = ext4_write_end,  
    ...  
};
```

from fs/ext4/inode.c

```
static const struct address_space_operations ext2_aops = {  
    ...  
    .write_begin      = ext2_write_begin,  
    .write_end        = ext2_write_end,  
    ...  
};
```

from fs/ext2/inode.c

```
static const struct address_space_operations fat_aops = {  
    ...  
    .write_begin      = fat_write_begin,  
    .write_end        = fat_write_end,  
    ...  
};
```

from fs/fat/inode.c

Usage Example - 1

❖ Dynamically determined target function

```
ssize_t generic_perform_write(struct file *file, ...)
{
    struct address_space *mapping = file->f_mapping;
    const struct address_space_operations *a_ops = mapping->a_ops;
    ...
    status = a_ops->write_begin(...);

    ...
    status = a_ops->write_end(...);
    ...
}
```

from mm/filemap.c

- Can't tell which function will be executed at the compile time
- Pointer of `struct address_space_operations a_ops` will be determined with the `<struct file *file>`'s filesystem

Usage Example - 2

❖ Load module function dynamically in the kernel space

- First, declare the function pointer from the **kernel side**

```
// Map this function pointer with the function implemented in pxt4 module  
void (*wb_queue_work_in_pxt4)(struct bdi_writeback *wb, struct wb_writeback_work *work) = NULL;  
EXPORT_SYMBOL(wb_queue_work_in_pxt4);
```

from fs/fs-writeback.c

- If `wb_queue_work_in_pxt4` is initialized, call the function that it points to
 - Also known as a callback function

```
static void wb_queue_work(struct bdi_writeback *wb, struct wb_writeback_work *work)  
{  
    if (wb_queue_work_in_pxt4) {  
        (*wb_queue_work_in_pxt4)(wb, work);  
        return;  
    }  
    ...  
}
```

from fs/fs-writeback.c

Usage Example - 2

❖ Load module function dynamically from kernel

- Declare the real function to be mapped with the function pointer `wb_queue_work_in_pxt4`

```
extern void (*wb_queue_work_in_pxt4)(struct bdi_writeback *, struct wb_writeback_work *);  
extern void pxt4_wb_queue_work(struct bdi_writeback *, struct wb_writeback_work *);
```

- Assign the function `pxt4_wb_queue_work()` to `wb_queue_work_in_pxt4` when the module is being installed
- Don't forget to nullify before exiting the module

```
static int __init pxt4_init_fs(void)  
{  
    ...  
    wb_queue_work_in_pxt4 = &pxt4_wb_queue_work;  
    ...  
}  
  
static void __exit pxt4_exit_fs(void)  
{  
    ...  
    wb_queue_work_in_pxt4 = NULL;  
    ...  
}  
...  
module_init(pxt4_init_fs)  
module_exit(pxt4_exit_fs)
```

from pxt4/super.c