OS 2016

Homework1: Kernel Module

(Due date: 2016/10/20 23:59:59)

Requirements

- Write a Kernel Module
  - Create multiple sysfs entries during module **init**
  - Remove all of these entries when module **exit**

- Create the following 3 sysfs entries in the kernel module
  1. /sys/kernel/hw1/swap_string
  2. /sys/kernel/hw1/calc
  3. /sys/kernel/hw1/sum_tree

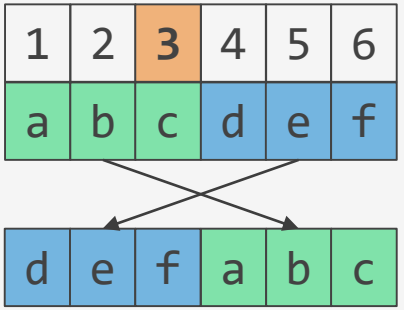
  Access to the entries are described in slides 3-5.

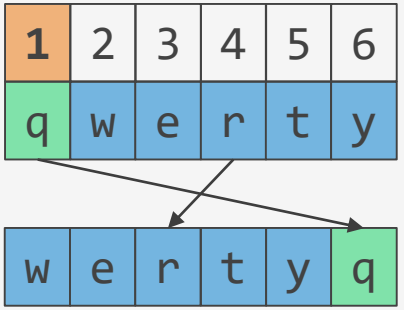
- The module handles 2 parameters, as described in slide 6.

Requirements - /sys/kernel/hw1/swap_string

- Swap the string by the specified index
- The input will be a given positive number **n** with a **string**
  - **n** is the index starts from 1, and will not bigger than the size of the string
  - For example:

```
$ echo "3 abcdef" > /sys/kernel/hw1/swap_string
$ cat /sys/kernel/hw1/swap_string
defabc
$ echo "3 abcdef" > /sys/kernel/hw1/swap_string
$ echo "1 qwerty" > /sys/kernel/hw1/swap_string
$ cat /sys/kernel/hw1/swap_string
wertyq
$ echo "6 a0b1c2" > /sys/kernel/hw1/swap_string
$ cat /sys/kernel/hw1/swap_string
a0b1c2
```





Requirements - /sys/kernel/hw1/calc

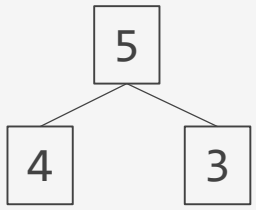
- A simple calculator with 5 operators: +, -, *, / and %
  - The operands must be integer(no floating points)
- Make sure the operator precedence: (*, /, %) > (+, -)
  - For example:

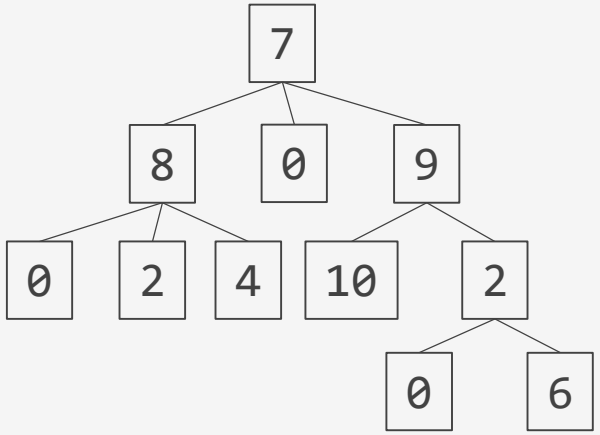
```
$ echo "16 % 3 + 2 * 3 - 5" > /sys/kernel/hw1/calc
$ cat /sys/kernel/hw1/calc
2
$ echo "-3" > /sys/kernel/hw1/calc
$ cat /sys/kernel/hw1/calc
-3
$ echo "5 / 2" > /sys/kernel/hw1/calc
$ cat /sys/kernel/hw1/calc
2
$ echo "2 / 3 * 2" > /sys/kernel/hw1/calc
$ cat /sys/kernel/hw1/calc
0
```

Requirements - /sys/kernel/hw1/sum_tree

- Input a given string that represents a tree's layout, you must output all of the sums from root to the leaves

```
(5
  (4
   3
  )
)
```






```
(7
  (8
    (0
     2
     4
    )
   0
   9
    (10
     2
      (0
       6
      )
    )
  )
)
```

```
$ echo "(7(8(0 2 4)0 9(10 2(0 6))))" > /sys/kernel/hw1/sum_tree
$ cat /sys/kernel/hw1/sum_tree
15, 17, 19, 7, 26, 18, 24
$ echo "(5(4 3))" > /sys/kernel/hw1/sum_tree
$ cat /sys/kernel/hw1/sum_tree
9, 8
```

Requirements (cont.)

- The module must handle 2 parameters
  - **mask**
    - int type, default: 111
    - a bitmask to decide whether a sysfs entries will be appeared or not
    - Example:
      - 101 for **swap_string** and **sum_tree** appear
      - 011 for **calc** and **sum_tree** appear
    - **Notice**: The input string starts from **0** will be treated as octal numbers
  - **name[1-3]**
    - charp type
    - to customize the entry's name
    - it will be the default name if you don't pass the parameter

```
$ insmod hw1.ko mask=110 name1="q1"
$ ls /sys/kernel/hw1
calc    q1
$ rmmod hw1
$ insmod hw1.ko mask=011 name1="foo" name2="bar"
$ ls /sys/kernel/hw1
bar     sum_tree
```

Background Knowledge

A Hello World Module

hw1.c

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

MODULE_LICENSE("Dual BSD/GPL");

static int __init create_hello(void)
{
  printk(KERN_ALERT "Hello!\n");
  return 0;
}

static void __exit cleanup_hello(void)
{
  printk(KERN_ALERT "Goodbye!\n");
}

module_init(create_hello);
module_exit(cleanup_hello);
```

printk

- Similar to the standard C library function – printf
  - But no floating point support
- Need a priority string to define log message's level

KERN_EMERG	system is unusable
KERN_ALERT	action must be taken immediately
KERN_CRIT	critical conditions
KERN_ERR	error conditions
KERN_WARNING	warning conditions
KERN_NOTICE	normal but significant condition
KERN_INFO	informational
KERN_DEBUG	debug-level messages

  - Example:

```
printk(KERN_INFO "x:%d, y:%d, z:%d\n", x, y, z);
```

Compiling Modules

Makefile

```
ifneq ($(KERNELRELEASE),)
    obj-m := hw1.o

else
    KERNELDIR ?= /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)

all:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules

clean:
    rm -rf *.o *.ko *.mod.c

endif
```

- After compile, there will be '*hw1.**ko***'

Compiling/Loading Modules

- Make your module

```
$ make
```

- Display module's information

```
$ modinfo hw1.ko
```

- Load the module into Kernel

```
$ insmod hw1.ko
```

- Print Kernel's message buffer

```
$ dmesg
```

- List all module

```
$ lsmod
```

- Unload the module

```
$ rmmod hw1
```

Module Parameters

- First, include the ‘moduleparam.h’
- Use ‘module_param’ macro to defined parameters

```
module_param(name, type, permission_mask)
```

  - Type
    - bool, invbool(0: true), charp, short, int, long, ushort, uint, ulong
  - Permission Mask - define at <linux/stat.h>
    - With a `S_I` prefix
    - R – read, W – write, X – execute
    - U or USR – user, G or GRP – group, O or OTH – others
  - Examples

S_IRUSR	Readable by user
S_IXGRP	Executable by group
S_IRWXO	Readable, writable and executable by others
S_IWUGO	Writable by user, group and others

Examples

hw1.c

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

static char *name = "World";
static unsigned int age = 0;

module_param(name, charp, S_IRUGO);
module_param(age, uint, S_IRUGO);

. . .

  printk(KERN_ALERT "Hello %s(%u)!\n", name, age);

. . .
```

```
$ insmod hw1.ko name="Candy" age=18
$ dmesg | tail
. . .
[ 123.456789] Hello Candy(18)!
```

sysfs

- sysfs is a pseudo filesystem
  - mounted at ‘/sys’

- A way to export kernel data structures, their attributes, and the linkages between them to user space

Kernel	**Userspace**
Kernel Objects(kobject)	Directories
Attributes(kobj_attribute)	Files
Relationships	Symbolic links

Creating a Kernel Object

- To create a simple kobject

```
struct kobject *kobject_create_and_add(char *name,
    struct kobject *parent);
```

- Example

hw1.c

```
#include <linux/kobject.h>
static struct kobject *hw1_kobject;

. . .
     hw1_kobject = kobject_create_and_add("hw1", kernel_kobj);
```

  - **kernel_kobj** is the '/sys/kernel'
  - so the **hw1_kobject** will be '/sys/kernel/hw1'

Creating an Attribute

- The attribute can be created by the macro

```
__ATTR(_name, _mode, _show, _store);
```

  - **_name** will be the attribute(file) name
  - **_mode** is the Permission Mask
  - **_show** and **_store** is the callback function while you read or write to the file

```
static ssize_t show(struct kobject *kobj,
    struct kobj_attribute *attr, char *buf);
static ssize_t store(struct kobject *kobj,
    struct kobj_attribute *attr, const char *buf, size_t count);
```

    - You must return how many bytes (ssize_t) you read from/write to the ***buf**
    - **count** contains the length of the **buf**

- Example:

```
struct kobj_attribute hw1_attribute =
    __ATTR(q1, 0660, q1_show, q1_store);
```

Adding an Attribute to a Kernel Object

- To add the attribute to Kernel Object

```
int sysfs_create_file(struct kobject *kobj,
    const struct attribute *attr);
```

- Example

```
error = sysfs_create_file(hw1_kobject, &hw1_attribute.attr);
```

Reference

- Linux Cross Reference
  - sysfs.h?v=4.4
  - kobject.h?v=4.4
- Kernel Documentation
  - filesystems/sysfs.txt
  - kobject.txt
- http://www.tldp.org/LDP/lkmpg/2.6/html/
- http://pradheepshrinivasan.github.io/2015/07/02/Creating-an-simple-sysfs/