Midterm (Deadline: November 13, 11:59 pm)

Started: Nov 6 at 2:40pm

Quiz Instructions

This is a take-home midterm exam. You will have consecutive 60 minutes to finish. Once the time is up, it will automatically be submitted.

Question 1

1 pts

The Linux kernel is a monolithic kernel.

● True

○ False

Question 2

1 pts

The Linux kernel is a microkernel.

○ True

● False

Question 3

1 pts

Linux has the following license(s):

☑ GPLv3

☑ GPLv2 and onwards

□ BSD

☑ GPLv2

Question 4

1 pts

Linux drivers are both in user space and kernel space.

○ True

● False

Question 5

1 pts

The completely fair scheduler (CFS) allots equal wall time to processes within a scheduling period.

● True

○ False

Question 6

1 pts

The completely fair scheduler (CFS) allots equal virtual time to processes within a scheduling period.

● True

○ False

Question 7

1 pts

We have a scheduling period of 50 ms. We have 3 processes: 2 with nice = 0, 1 with nice = 6. What is the wall-time of the nice 6 process? In other words, how long does it *actually* run? (rounded to nearest integer)

● 6 ms

○ 17 ms

○ 22 ms

○ 10 ms

Question 8

1 pts

Which of the following is the correct order for interfacing between systems:

○ User Applications<->Hardware<->Operating System

● User Applications<->Operating System<->Hardware

○ Hardware<->User Applications<->Operating System

Question 9

1 pts

Select the following that could be considered part of an operating system:

☑ Terminal

☑ Process handling

☑ GUI (Graphical user interface)

☑ File management

☑ Process scheduling

☑ Networking

Question 10

1 pts

Select the following that could be considered part of the kernel:

☐ GUI (Graphical user interface)

☐ Terminal

☑ Networking

☑ Processing scheduling

☑ Process handling

☑ File management

Question 11

1 pts

User applications never enter the kernel; Instead, the kernel may act on behalf of applications via system calls.

● True

○ False

Question 12

1 pts

When a system call is activated, the processor is interrupted and the input arguments into the system call handler are passed from user space like a normal function call.

● True

○ False

Question 13

1 pts

The memory space for the kernel is one space while applications each have their own memory space.

● True

○ False

Question 14

1 pts

The fork glibc (GNU C Library) function calls the system call fork.

● True

○ False

Question 15

1 pts

The primary core types of modules are:

○ char, block, serial

● char, block, network

○ USB, char, network

○ char, network, video

Question 16

1 pts

Select each of the following that is true about kernel modules:

☑ insmod causes the init function to be called

☑ rmmod causes the exit function to be called

Question 17

1 pts

In terms of analogy, /dev is more like the packaging and labeling of a box while /sys is more like opening and accessing the box.

○ True

● False

Question 18

1 pts

Character devices appear as a file in /dev while block devices do not appear as a file in /dev

○ True

● False

Question 19

1 pts

Network interfaces have a file associated with them.

○ True

● False

Question 20

1 pts

systemd is part of the Linux kernel.

○ True

● False

Question 21

1 pts

systemd operates in user space.

○ True

● False

Question 22

1 pts

Package code is updated by sudo apt update.

● True

○ False

Question 23

1 pts

Consider the following module code, which is in a file naksu.c (and compiled to naksu.ko).

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>

#define DEVICE_NAME "onepiece"
#define CLASS_NAME "treasure"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("(Captain) Jack Sparrow");
MODULE_DESCRIPTION("Greatest module in the sea!");
MODULE_VERSION("0.000001");

static int times = 10;
module_param(times, int, S_IRUGO);

static int __init hello_init(void){
    printk(KERN_INFO "I’m great but Captain Barbossa is %d times more of a
team leader TBH\n", times);
    return 0;
}

static void __exit hello_exit(void){
    printk(KERN_INFO "Time to go to Tortuga and chill\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

The following files or directories will exist in the file system after this module is inserted:

☐ /dev/naksu

☐ /sys/class/treasure/onepiece

☑ /sys/module/naksu

☑ /dev/onepiece

Question 24 1 pts

Consider the same module code as the last question, which is in a file naksu.c (and compiled to naksu.ko).

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

```
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>

#define DEVICE_NAME "onepiece"
#define CLASS_NAME "treasure"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("(Captain) Jack Sparrow");
MODULE_DESCRIPTION("Greatest module in the sea!");
MODULE_VERSION("0.000001");

static int times = 10;
module_param(times, int, S_IRUGO);

static int __init hello_init(void){
    printk(KERN_INFO "I'm great but Captain Barbossa is %d times more of a
team leader TBH\n", times);
    return 0;
}

static void __exit hello_exit(void){
    printk(KERN_INFO "Time to go to Tortuga and chill\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

The following is true about the use of the S_IRUGO flag:

☐ Only root can view the value of times

☑ Only root can change the value of times

☑ All users are able to view the value of times

☐ All users are able to modify the value of times

Question 25

1 pts

Consider the module code (different from before), which is in a file naksu.c (and compiled to naksu.ko).

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
#include <linux/uaccess.h>
#define DEVICE_NAME "onepiece"
#define CLASS_NAME "treasure"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;

static int device_open(struct inode *, struct file *);

static struct file_operations fops =
{
 .open = device_open
};

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abraham Lincoln");
MODULE_DESCRIPTION("Greatest module in the world!");
MODULE_VERSION("0.000001");

static int multiplier = 10;
module_param(multiplier, int, S_IRUGO);

static int __init hello_init(void){
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    mescharClass = class_create(THIS_MODULE, CLASS_NAME);
    mescharDevice = device_create(mescharClass, NULL, MKDEV(majorNumber,
0), NULL, DEVICE_NAME);
    printk(KERN_INFO "Oh hi mark - I love Lisa %dX more than you do\n", mu
ltiplier);
    return 0;
}

static void __exit hello_exit(void){
    device_destroy(mescharClass, MKDEV(majorNumber,0));
    class_unregister(mescharClass);
    class_destroy(mescharClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multi
plier);
}

static int device_open(struct inode *inodep, struct file *filep){
    printk(KERN_INFO "You’re tearing me apart, Lisa!\n");
    return 0;
}

module_init(hello_init);
module_exit(hello_exit);
```

The following files will exist in the file system after this module is inserted:

☑ /sys/class/treasure/onepiece

☑ /dev/onepiece

□ /dev/treasure/onepiece

☑ /sys/module/naksu

Question 26

1 pts

Consider the module code (same as the previous question), which is in a file naksu.c (and compiled to naksu.ko).

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/device.h>
```

```
#include <linux/uaccess.h>
#define DEVICE_NAME "onepiece"
#define CLASS_NAME "treasure"

static int majorNumber;
static struct class* mescharClass = NULL;
static struct device* mescharDevice = NULL;

static int device_open(struct inode *, struct file *);

static struct file_operations fops =
{
 .open = device_open
};

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Abraham Lincoln");
MODULE_DESCRIPTION("Greatest module in the world!");
MODULE_VERSION("0.000001");

static int multiplier = 10;
module_param(multiplier, int, S_IRUGO);

static int __init hello_init(void){
    majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
    mescharClass = class_create(THIS_MODULE, CLASS_NAME);
    mescharDevice = device_create(mescharClass, NULL, MKDEV(majorNumber,
0), NULL, DEVICE_NAME);
    printk(KERN_INFO "Oh hi mark - I love Lisa %dX more than you do\n", mu
ltiplier);
    return 0;
}

static void __exit hello_exit(void){
    device_destroy(mescharClass, MKDEV(majorNumber,0));
    class_unregister(mescharClass);
    class_destroy(mescharClass);
    unregister_chrdev(majorNumber, DEVICE_NAME);
    printk(KERN_INFO "sad, but still love Lisa %dX more than you\n", multi
plier);
}

static int device_open(struct inode *inodep, struct file *filep){
    printk(KERN_INFO "You're tearing me apart, Lisa!\n");
    return 0;
}

module_init(hello_init);
module_exit(hello_exit);
```

The following function is actually not required in the above code.

○ device_destroy()

○ unregister_chrdev()

● class_unregister()

○ class_destroy()

Question 27

1 pts

Using the open() C standard library function is an example of direct use of a system call.

○ True

● False

Question 28

1 pts

Working with a character device named "coolDevice: through a driver would involve:

□ Using register_blkdev, class_create, and device_create to set up the device in the driver

☑ Using the C function open() on /dev/coolDevice when accessing the device from user space

☑ Linking the system call open with the open() function in the driver

Question 29

1 pts

The major number of a driver is unique compared to other major numbers of different drivers.

● True

○ False

Question 30

1 pts

The minor number of a device is unique compared to other minor numbers of devices of different drivers.

○ True

● False

Quiz saved at 3:13pm Submit Quiz