Project 1 Report

Litao Zhou, 518030910407

# Project Description

In this project, we will create some kernel modules and load them into the Linux kernel. Then we modify the kernel module so that it creates an entry in the /proc file system. There are three parts of experiments. First we try to load the simple.c kernel module into a Linux Virtual Machine. Then based on simple.c we modify the program by calling several kernel functions. The modified programs are in number.c and jifhz.c. Finally, we will try to load hello.c, which will create an entry in the /proc file system, and further develop other two kernel modules jiffies.c and seconds.c.

# Loading and Removing Kernel Modules

## Experiment Results

1. Compile the given simple.c into kernel module and load it into the kernel.

文本

描述已自动生成

1. The loading info can be found in dmesg

文本

描述已自动生成

1. Remove our module from the kernel, and we can find the removing info from dmesg

文本

描述已自动生成

# Golden Ratio Prime and GCD

## Implementation

1. First include the kernel libraries that will be used. In this part, we use <linux/hash.h> <linux/gcd.h>.
2. Then we print the data required in the kernel using printk(). Note that the GOLDEN\_RATIO\_RATE is a long long unsigned number, and gcd() returns a long unsigned number. The datatype should be matched when printing.

The code we edited is shown below

#include <linux/hash.h>

#include <linux/gcd.h>

/\* This function is called when the module is loaded. \*/

static int simple\_init(void)

{

printk(KERN\_INFO "Loading Module\n");

printk(KERN\_INFO "Golden Ratio Prime:%llu\n", GOLDEN\_RATIO\_PRIME);

return 0;

}

/\* This function is called when the module is removed. \*/

static void simple\_exit(void) {

printk(KERN\_INFO "GCD(3300,24)=%lu\n", gcd(3300,24));

printk(KERN\_INFO "Removing Module\n");

}

## Result

Our module printed out the expected result when loading and removing the module

文本

描述已自动生成

# Jiffies and HZ

## Implementation

Similar as above, we include the related libraries so that we can access jiffies and HZ value. Then we print them out into the kernel message buffer. The code we edited is shown below

#include <linux/jiffies.h>

#include <linux/param.h>

/\* This function is called when the module is loaded. \*/

static int simple\_init(void)

{

printk(KERN\_INFO "Loading Module\n");

printk(KERN\_INFO "%d", HZ);

printk(KERN\_INFO "jiffies:%llu\n", get\_jiffies\_64());

return 0;

}

/\* This function is called when the module is removed. \*/

static void simple\_exit(void) {

printk(KERN\_INFO "jiffies:%llu\n", get\_jiffies\_64());

printk(KERN\_INFO "Removing Module\n");

}

## Result

Our module printed out the expected result when loading and removing the module. Since jiffies is the interrupt count since booting, and HZ is the frequency of the interrupt. We can calculate the running time of the module as follows

文本

描述已自动生成

The difference of the two jiffies value above in my experiment is . The running time of our module is , which coincides with the running time indicated by dmesg.

# Jiffies in /proc

## Implementation

1. We import the kernel libraries related to jiffies. Then we modify the content to be printed out to the buffer in proc\_read() function, as has been shown below.
2. The contents to be printed is just the present jiffies value, which can be implemented by calling get\_jiffies\_64() kernel function.
3. On my virtual machine, I have to rename the original copy\_to\_user() function into raw\_copy\_to\_user() so that the program can compile.

#include <linux/jiffies.h>

#define BUFFER\_SIZE 128

#define PROC\_NAME "jiffies"

......

static ssize\_t proc\_read(struct file \*file, char \_\_user \*usr\_buf, size\_t

count, loff\_t \*pos)

{

int rv = 0;

char buffer[BUFFER\_SIZE];

static int completed = 0;

if (completed) {

completed = 0;

return 0;

}

completed = 1;

rv = sprintf(buffer, "jiffies:%llu\n", get\_jiffies\_64());

// copies the contents of buffer to userspace usr\_buf

raw\_copy\_to\_user(usr\_buf, buffer, rv);

return rv;

}

## Result

We load the jiffies module we have created into the kernel. After checking that the module is successfully loaded, we read the /proc/jiffies several times, and it can be seen that the jiffies value printed out are dynamically increasing depending on the time we make a request.

文本

描述已自动生成

# Elapsed time in /proc

## Implementation

We can calculate the elapsed time by using the formula proposed above.

In order to obtain the difference between the current jiffies and the jiffies when the module is loaded, we need to store the loading jiffies. Therefore, we introduce a new static variable to store the initial jiffies. The load\_jif variable will be updated when the module is loaded.

#include <linux/jiffies.h>

#include <linux/param.h>

#define BUFFER\_SIZE 128

#define PROC\_NAME "seconds"

static u64 load\_jif;

......

static int proc\_init(void)

{

// creates the /proc/seconds entry

// the following function call is a wrapper for

// proc\_create\_data() passing NULL as the last argument

load\_jif = get\_jiffies\_64(); // <--- This is new

proc\_create(PROC\_NAME, 0, NULL, &proc\_ops);

printk(KERN\_INFO "/proc/%s created\n", PROC\_NAME);

return 0;

}

static ssize\_t proc\_read(struct file \*file, char \_\_user \*usr\_buf, size\_t

count, loff\_t \*pos)

{

int rv = 0;

char buffer[BUFFER\_SIZE];

static int completed = 0;

u64 elapsed\_time;

if (completed) {

completed = 0;

return 0;

}

completed = 1;

elapsed\_time = (get\_jiffies\_64() - load\_jif) / HZ;

rv = sprintf(buffer, "elapsed:%llus\n", elapsed\_time);

// copies the contents of buffer to userspace usr\_buf

raw\_copy\_to\_user(usr\_buf, buffer, rv);

return rv;

}

## Result

We load the seconds module we have created into the kernel. After checking that the module is successfully loaded, we read the /proc/seconds several times, and it can be seen that the seconds value printed out are dynamically increasing from 0s depending on the moment we make a request.

文本

描述已自动生成