آزمایشگاه سیستم عامل

دستورکار ۱۱: برنامه‌نویسی در هسته لینوکس

در این دستورکار چگونگی ایجاد ماژول هسته و بارگذاری آن روی هسته لینوکس آموزش داده می‌شود. می‌توان برای نوشتن این برنامه‌ها به زبان C از یک ویرایشگر استفاده شود. ماژول‌های هسته در آدرس /lib/modules قرار گرفته‌اند و با پسوند **.ko** و در نسخه‌های قدیمی‌تر با پسوند **.o** مشخص می‌شوند.

مزیت نوشتن ماژول‌های هسته، این است که یک روش آسان برای تعامل با هسته ایجاد گردد. لذا این امکان فراهم است که برنامه‌ای نوشته شود که مستقیماً توابع هسته را فراخوانی کند. از آنجایی که این برنامه‌ها در هسته بارگذاری می‌شود، هر خطایی در کد برنامه می‌تواند باعث خرابی سیستم شود. بنابراین، بهتر است از ماشین مجازی برای اجرای این دستور کار استفاده کنید.

بخش اول: ایجاد ماژول‌های هسته

اولین بخش این دستورکار، مراحل ایجاد و درج ماژول در هسته لینوکس است. تمامی ماژول‌های هسته‌ای که در حال حاضر در سیستم بارگذاری شده‌اند، با دستور زیر فهرست می‌شوند:

```
lsmod
```

```
Module                  Size  Used by
snd_intel8x0           40960  2
snd_ac97_codec        131072  1 snd_intel8x0
ac97_bus               16384  1 snd_ac97_codec
snd_pcm                98304  2 snd_intel8x0,snd_ac97_codec
snd_seq_midi           16384  0
intel_powerclamp       16384  0
crct10dif_pclmul       16384  0
crc32_pclmul           16384  0
ghash_clmulni_intel    16384  0
pcbc                   16384  0
snd_seq_midi_event     16384  1 snd_seq_midi
snd_rawmidi            32768  1 snd_seq_midi
joydev                 24576  0
aesni_intel           188416  0
aes_x86_64             20480  1 aesni_intel
crypto_simd            16384  1 aesni_intel
snd_seq                65536  2 snd_seq_midi,snd_seq_midi_event
glue_helper            16384  1 aesni_intel
cryptd                 24576  3 crypto_simd,ghash_clmulni_intel,aesni_intel
intel_rapl_perf        16384  0
```

برنامه زیر یک ماژول هسته بسیار ساده را نشان می‌دهد که در موقع بارگذاری و برداشتن ماژول هسته، پیام‌های مناسبی را چاپ می‌کند:

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
/* this function is called when the module is loaded*/
int simple_init(void)
{
       printk(KERN_INFO "Loading Module\n");
       return 0;
}
/* this function is called when the module is removed*/
void simple_exit(void)
{
       printk(KERN_INFO "Removing Module\n");
}
/* Macros for registering module entry and exit points.
*/
module_init(simple_init);
module_exit(simple_exit);
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("simple module");
MODULE_AUTHOR("SGG");
```

تابع simple_init()، نقطه ورود ماژول است که معرف تابعی است که موقع بارگذاری ماژول در هسته احضار می‌شود. به طور مشابه تابع simple_exit() نقطه خروج ماژول است که موقع حذف ماژول از هسته فراخوانی می‌شود. تابع نقطه ورود ماژول باید یک مقدار صحیح برگرداند، صفر معرف موفقیت‌آمیز بودن عملیات و مقادیر دیگر معرف خطاست. تابع نقطه خروج ماژول، void برمی‌گرداند. به هیچ یک از نقاط ورود یا خروج پارامتری ارسال نمی‌شود. دو ماکروی module_init() و module_exit() برای ثبت نقاط ورود و خروج ماژول در هسته استفاده می‌شوند:

```
module_init()
module_exit()
```

توجه گردد که چگونه هر دو تابع نقاط ورود و خروج، تابع printk() را فراخوانی می‌کنند. تابع printk() معادل هسته تابع printf() است. هرچند خروجی آن به یک بافر سابقه هسته فرستاده می‌شود که محتوی آن می‌تواند توسط فرمان dmesg خوانده شود. فرق میان printk() و printf() در این است که printk() امکان می‌دهد یک پرچم الویت مشخص گردد تا مقادیر آن در فایل سرآیند <linux/printk.h> مشخص گردد. در اینجا، الویت KERN_INFO تنظیم شده است که به عنوان یک پیغام اطلاعاتی تعریف می‌شود.

خطوط آخر MODULE_LICENSE()، MODULE_DESCRIPTION() و MODULE_AUTHOR() معرف جزییات مربوط به مجوز نرم‌افزار، توصیف ماژول و نویسنده است. این کار تجربه استاندارد در نوشتن ماژول‌های هسته به حساب می‌آید.

برای کامپایل ابتدا فایلی به نام Makefile ایجاد کرده و دستورات زیر را در این فایل تایپ کرده و آن را ذخیره کنید. Makefile مجموعه دستورات لازم برای کامپایل یک پروژه نرم افزاری است که در قالبی خاص در یک فایل متنی ذخیره شده است.

```
obj-m += نام فایل خروجی
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

در ادامه، برای کامپایل دستور make را در خط فرمان وارد کنید. make برنامه ای است که از روی Makefile کار کامپایل پروژه را انجام می دهد. در کل دو مزیت عمده می توان برای make در نظر گرفت:

۱- با استفاده از Makefile کارهای سخت (نوشتن دستورات پیچیده کامپایل) فقط یک بار انجام می‌شود.

۲- make از روی برچسب زمانی فایل‌های سورس و Object تشخیص می دهد که کدام‌یک از فایل‌ها نیاز به دوباره کامپایل شدن دارند و به این ترتیب از کامپایل مجدد و غیر ضروری فایل‌هایی که از زمان آخرین کامپایل تغییری نکرده‌اند، جلوگیری می‌شود. چرا که فرایند کامپایل به ویژه در کامپیوترهای قدیمی و با پروژه‌های بزرگ کاری بسیار وقت‌گیر است.

make بعد از فراخوانی به ترتیب دنبال یک فایل با یکی از نام‌های GNUmakefile، makefile یا Makefile می‌گردد تا دستورات کامپایل را از داخل آن خوانده و اجرا کند.

پس از کامپایل، فایل‌های متعددی تولید می‌شود که فایل simple.ko معرف ماژول هسته کامپایل شده است. مرحله بعدی، درج این ماژول را در هسته لینوکس روشن می‌سازد.

بخش دوم: بارگذاری و حذف ماژول‌های هسته

ماژول‌های هسته با استفاده از فرمان insmod بارگذاری می‌شوند که به صورت زیر اجرا می‌شود:

```
sudo insmod simple.ko
```

به منظور بررسی اینکه ماژول بارگذاری شده است یا خیر، فرمان lsmod را اجرا می‌گردد و ماژول simple را جستجو می‌کند. توجه شود نقطه ورود ماژول در موقع درج ماژول در هسته احضار می‌شود. برای بررسی محتوی این پیغام در بافر سابقه هسته، فرمان dmesg استفاده می‌شود.

```
dmesg
```

بایستی پیغام "Loading Module" را مشاهده گردد. به‌منظور برداشتن ماژول هسته، فرمان rmmod استفاده می‌شود.

```
sudo rmmod simple
```

با بررسی فرمان dmesg اطمینان از برداشته شدن ماژول، حاصل می‌شود. چون بافر سابقه هسته می‌تواند به سرعت پر شود، بهتر است به تناوب بافر را خالی شود. این کار می‌تواند به صورت زیر انجام می‌شود:

```
sudo  dmesg -c
```

بخش سوم: ساختمان داده‌های هسته

این بخش شامل اصلاح ماژول هسته است، طوری که از ساختمان داده لیست پیوندی هسته استفاده می‌کند. هسته لینوکس چند نمونه از ساختمان داده‌های مختلف را پوشش می‌دهد که در این دستور کار استفاده از لیست پیوندی دوطرفه چرخشی بررسی می‌شود که برای توسعه‌دهندگان هسته فراهم است. آنچه در این بخش بررسی می‌شود در فایل سرآیند <linux/list.h> فراهم است. بنابراین، در ابتدا باید یک ساختار (struct) تعریف گردد که شامل عناصری است که در لیست پیوندی درج می‌شوند.

```c
struct birthday {
	int day;
	int month;
	int year;
	struct list_head list;
}
```

در این قطعه کد به عضو struct list_head list توجه شود. این رکورد در فایل سرآیند <linux/types.h> تعریف می‌شود. هدف آن، گذاشتن لیست پیوندی در میان گره‌های سازنده‌ی لیست است. رکورد list_head کاملاً ساده بوده و فقط دو عنصر دارد (next و prev)، که به گره‌های قبلی و بعدی در لیست اشاره می‌کند. با گذاشتن لیست پیوندی در میان رکوردها، لینوکس این امکان را فراهم می‌کند که مدیریت ساختمان داده در اختیار توابع ماکرو قرار گیرد.

به منظور درج عناصر در لیست پیوندی ماکروی LIST_HEAD یک شی birthday_list اعلان می‌کند که به عنوان اشاره‌گری به ابتدای لیست استفاده می‌گردد:

```c
static LIST_HEAD(birthday_list);
```

این ماکرو متغیر birthday_list را که از نوع struct list_head است، تعریف و مقداردهی می‌کند. نمونه‌های struct birthday را به صورت زیر ایجاد کرده و مقداردهی می‌شود.

```c
struct birthday *person;
person = kmalloc(sizeof(person), GFP_KERNEL);
person->day = 2;
person->month = 8;
person->year = 1995;
INIT_LIST_HEAD(&person->list);
```

تابع kmalloc() معادل هسته‌ای تابع سطح کاربری malloc() برای تخصیص حافظه می‌باشد، که در اینجا حافظه به هسته تخصیص داده می‌شود. پرچم GFP_KERNEL تخصیص معمول حافظه هسته را نشان می‌دهد). دقت داشته باشید برای استفاده از kmalloc() می‌بایست از کتابخانه <linux/slab.h> استفاده کنید. ماکروی INIT_LIST_HEAD، عضو list در struct birthday را مقدار اولیه می‌دهد. در ادامه، می‌توان این نمونه را با استفاده از ماکروی list_add_tail() به انتهای لیست پیوندی پیوندی اضافه کرد.

```c
list_add_tail(&person->list, &birthday_list);
```

برای پیمایش لیست پیوندی از تابع list_for_each_entry() استفاده می‌شود که سه پارامتر زیر را می‌پذیرد:

- اشاره‌گر به رکوردی که پیمایش روی آن صورت می‌گیرد.
- اشاره‌گر به سر لیستی که پیمایش روی آن صورت می‌گیرد.
- نام متغیر شامل رکورد list_head

کد زیر این ماکرو را نشان می‌دهد.

```c
struct birthday *ptr;
list_for_each_entry(ptr, &birthday_list, list) {
	/*on each iteration ptr points to the next birthday struct*/
}
```

تمرین

۱- مراحل بخش اول و دوم را دنبال کنید تا یک ماژول هسته را ایجاد، بارگذاری و بردارید. ضمن بررسی محتوی بافر سابقه هسته مطمئن شوید مراحل کار را به درستی انجام داده‌اید.

۲- با توجه به بخش سوم در نقطه ورود ماژول، یک لیست پیوندی شامل پنج عنصر struct birthday ایجاد کنید. لیست پیوندی را پیمایش کنید و محتوای آن را به بافر سابقه هسته انتقال دهید. فرمان dmesg را احضار کنید تا مطمئن شوید که به محض بار شدن ماژول هسته، لیست حذف می‌شود.

- ✓ برای هریک سوالات خروجی هر مرحله را اسکرین شات گرفته و در قالب یک گزارش تحویل دهید.