دانشگاه صنعتی امیرکبیر

# آزمایشگاه سیستم عامل

## جلسه سوم: برنامه‌نویسی ماژول‌های هسته و آشنایی با ساختمان‌های داده در هسته

مدرس: مینا یوسف‌نژاد

## ماژول‌های هسته

ماژول‌های هسته (Kernel Modules) قطعاتی از کد هستند که می‌توانند به صورت پویا به هسته سیستم‌عامل لینوکس بارگذاری و از آن تخلیه شوند. این قابلیت به توسعه‌دهندگان و مدیران سیستم اجازه می‌دهد تا قابلیت‌های جدیدی به هسته اضافه کرده یا قابلیت‌های موجود را تغییر دهند بدون نیاز به بازسازی یا راه‌اندازی مجدد هسته.

**مزایای استفاده از ماژول‌ها:**

- **پویایی:** امکان افزودن قابلیت‌های جدید به هسته بدون نیاز به بازسازی کل هسته.
- **بهینه‌سازی حافظه:** ماژول‌های غیرضروری می‌توانند بارگذاری نشده باقی بمانند، که مصرف حافظه را کاهش می‌دهد.
- **انعطاف‌پذیری:** توسعه‌دهندگان می‌توانند به سرعت تغییرات

# چرخه زندگی ماژول‌های هسته

ماژول‌های هسته دارای چرخه زندگی مشخصی هستند که شامل مراحل زیر می‌باشد:

- کامپایل: نوشتن و کامپایل کد ماژول به فایل باینری .ko
- بارگذاری: استفاده از ابزارهایی مانند insmod یا modprobe برای بارگذاری ماژول به هسته.
- فعال‌سازی: اجرای تابع بارگذاری init ماژول.
- استفاده: ماژول در حال اجرا و ارائه قابلیت‌های خود.
- تخلیه: استفاده از ابزارهایی مانند rmmod برای تخلیه ماژول از هسته.

## کدنویسی ماژول‌ها:

- تابع بارگذاری (Initialization Function) این تابع زمانی فراخوانی می‌شود که ماژول به هسته اضافه می‌شود.
- تابع تخلیه (Exit Function) این تابع زمانی فراخوانی می‌شود که ماژول از هسته خارج می‌شود.

هر ماژول هسته باید شامل هدرهای زیر باشد:

#include <linux/init.h> ← برای ماکروهای module_init و module_exit

#include <linux/module.h> ← برای تعریف ماژول‌ها

#include <linux/printk.h> ← برای استفاده از توابع پرینت

## تابع بارگذاری:

این تابع هنگام بارگذاری ماژول به هسته اجرا می‌شود. در این مثال، از تابع pr_info برای چاپ پیام استفاده شده است. این تابع پیام‌ها را در بافر سابقه هسته ذخیره می‌کند.

```c
static int simple_init(void)
{
    pr_info("Module loaded\n");
    // Equivalent: printk(KERN_INFO "Module loaded\n");
    return 0; // Returning 0 indicates the module was successfully loaded
}
```

## تابع تخلیه:

این تابع هنگام تخلیه ماژول از هسته اجرا می‌شود و برای انجام عملیات پاکسازی استفاده می‌گردد.

```
static void simple_exit(void)
{
    pr_info("exit module\n");
}
```

برای اینکه هسته بداند کدام توابع باید هنگام بارگذاری و تخلیه ماژول فراخوانی شوند، از ماکروهای module_init و module_exit استفاده می‌کنیم:

```
module_init(simple_init);
module_exit(simple_exit);
```

```c
#include <linux/init.h>        // For the module_init and module_exit macros
#include <linux/module.h>      // For defining modules
#include <linux/printk.h>      // For using print functions

/* This function is called when the module is loaded */
static int __init simple_init(void)
{
    pr_info("Loading simple module\n");
    return 0; // Returning 0 indicates successful loading
}

/* This function is called when the module is unloaded */
static void __exit simple_exit(void)
{
    pr_info("Unloading simple module\n");
}

/* Registering functions as entry and exit points */
module_init(simple_init);
module_exit(simple_exit);

/* Adding meta information to the module */
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("A simple module for testing");
MODULE_AUTHOR("Operating Systems Lab Group");
```

- برای مشاهده پیام‌هایی که با استفاده از pr_info یا printk ارسال شده‌اند، می‌توانید از دستور journalctl استفاده کنید.
- به عنوان مثال: مشاهده لاگ‌های ۵ دقیقه اخیر:

```
sudo journalctl --since "5 minutes ago"
```

- فیلتر کردن لاگ‌ها برای پیام‌های خاص (مثلاً پیام حذف ماژول):

```
sudo journalctl --since "5 minutes ago" | grep "Removing module "
```

# Makefile

```
obj-m += simple-module.o

PWD := $(CURDIR)

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

```
Sudo insmod simpleModule.ko
```

```
Sudo rmmod simpleModule
```

برای بارگذاری ماژول از دستور insmod استفاده کنید:

```
sudo insmod simple_module.ko
```

برای اطمینان از بارگذاری موفقیت‌آمیز، می‌توانید لاگ‌های سیستم را بررسی کنید:

```
sudo dmesg | tail
```

یا با استفاده از journalctl:

```
sudo journalctl --since "1 minute ago"
```

تخلیه ماژول از هسته:

```
sudo rmmod simple_module
```

| ویژگی | لینوکس | ویندوز | macOS |
| --- | --- | --- | --- |
| نام ماژول‌ها | Kernel Modules | Kernel-Mode Drivers | Kexts (Kernel Extensions) |
| زبان برنامه‌نویسی | C | C و C++ | C++ |
| چارچوب توسعه | استاندارد هسته لینوکس | Windows Driver Model (WDM) | IOKit |
| محیط توسعه | GCC, Makefile | Microsoft Visual Studio و WDK | Xcode و KDK |
| مدیریت حافظه | kfree، kmalloc | MmAllocate ، MmFree | IOKit Memory Management |
| ابزار دیباگینگ | dmesg ،printk ، GDB | WinDbg ،Visual Studio | Console ،Xcode Debugger |

# ساختمان داده‌هاي هسته

```
// Define your data structure
struct myType
{
  struct rb_node nodeName; // needed to be capable to use with
                           // kernel data structure of red black trees
(rbtree.h)

  char *someData; // user defined fields
  int someOtherData;
  // ...
};
```

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

| | | | |
|---|---|---|---|
| list_head | ليست پيوندي دوطرفه | <linux/types.h><br><linux/list.h> | مديريت process و device |
| rb_node, rb_root | پيادهسازي درخت قرمز-سياه | <linux/rbtree.h> | زمان بند و مديريت حافظه |
| hlist_head, hlist_node | ساخت hash table | <linux/hashtable.h> | مديريت اتصالات و مسيريابی در شبکه |
| Kfifo | ساخت يک صف FIFO | <linux/kfifo.h> | ارتباط بين فرايندها در لينوکس |

# درج عناصر در لیست پیوندي

```
LIST_HEAD(my_list);
```

```
struct birthday *b1;
b1 = kmalloc(sizeof(person), GFP_KERNEL);
b1->day = 2;
b1->month = 8;
b1->year = 1995;
INIT_LIST_HEAD(&b1->list);
```

```
list_add_tail(&b1->list, &my_list) // new node, head node
```

# پیمایش لیست پیوندي

```
Struct birthday *current;
list_for_each_entry(current, &my_list, list) {
  // some computation with the current node
}
```