Ime in priimek: **Datum:**

Linux jedro, moduli

Namen vaje je:

- Spoznavanje jedra OS, modulov in gonilnikov.
- Programiranje v jedru, rokovanje z moduli v jedru OS.

Jedrni moduli (*Kernel Module*) so deli kode, ki se lahko na zahtevo vključijo (*load*) v jedro in izključijo (*unload*) iz njega. Poznamo statične in dinamične module. Statični moduli so tisti, ki so že prevedeni z jedrom in vključeni vsakič, ko se sistem zažene. Dimamični moduli razširjajo funkcionalnost jedra, pri čemer sistema ni potrebno ponovno zaganjati. Primer modula je gonilnik (*device driver*), ki omogoča jedru dostop do strojne opreme priključene na sistem.

GRADIVA: https://github.com/matejkrenLTFE/POKS2022

Nekaj osnovnih ukazov za delo z moduli:

insmod *ime-modula.ko* - vključevanje modula v jedro

rmmod *ime-modula* - odstranimo modula iz jedra

modinfo *ime-modula.ko* - informacija o modulu

lsmod - prikaz vseh modulov vključenih v jedro

Za izvajanje *insmod* in *rmmod* potrebujemo superuser pravice, zato izvedemo *sudo su*.

V jedru ne moremo uporabljati sistemskih funkcij tako kot v uporabniškem prostoru. V jedru lahko za izpis teksta uporabljamo funkcijo ***printk***.

```
printk(KERN_ALERT "moj tekst\n");
```

Tekst iz jedra se po privzetih nastavitvah zapisuje v datoteko */var/log/**kern.log*** *(ali /var/log/messages).* Za pregled zadnjih vpisov v datoteko lahko uporabimo ukaz ***dmesg***

ali (najbolje na ločenem terminalu): ***tail –f /var/log/kern.log***

Kako prevesti modul?

Pripravimo **Makefile** z navodili za prevajanje(glej primer spodaj).

Modul nato prevedemo z ukazom ***make.***

Dokumentacija za modul se nahaja na koncu C kode modula. Primer za MODULE_LICENSE:

```
MODULE_LICENSE("GPL");
```

Ostali možni podatki za modul so še:

MODULE_DESCRIPTION, MODULE_SUPPORTED_DEVICE, MODULE_AUTHOR, ...

1.VAJA: Modul

Sestavite enostaven kernel modul **hello.c**, ki po vključitvi in izključitvi izpiše tekst s *printk*. Sestavite datoteko **Makefile** in modul prevedite, da dobite kernel objektni modul **hello.ko**:

```
make
```

Izpisi modula se nahajajo v **/var/log/kern.log** datoteki, najbolje da so prikazani na ločenem terminalu s:

```
tail -f /var/log/kern.log
```

Modul vključite v kernel z ukazom:

```
insmod hello.ko
```

Preverite stanje modulov:

```
lsmod
```

```
modinfo hello.ko
```

Modul odstranite s:

```
rmmod hello
```

Modulu popravite dokumentacijo in preverite z ukazom **`modinfo`**.

Primer kode za kernel modul **hello.c:**

```
/* hello.c - Demonstrates module documentation.  */
#include <linux/module.h>   /* Needed by all modules */
#include <linux/kernel.h>   /* Needed for KERN_INFO */
#include <linux/init.h>     /* Needed for the macros */

static int __init init_modul(void)
{
  printk(KERN_INFO "Hello, world 4\n");
  return 0;
}

static void __exit cleanup_modul(void)
{
  printk(KERN_INFO "Goodbye, world 4\n");
}

module_init(init_modul);
module_exit(cleanup_modul);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Author Name");     /* Who wrote this module? */
MODULE_DESCRIPTION("Demo modul");  /* What does this module do */
```

Primer za **Makefile** (https://cybersec.ltfe.org/Makefile):

```
obj-m += hello.o

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

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

V Makefile obvezno uporabljajte TAB in ne SPACE !

Opišite in razložite dogajanje :

2.VAJA: Modul za gonilnik (device driver) tipa 'Character device'

Kako naredimo modul 'Character device'

Pomoč: https://cybersec.ltfe.org/vaja2.c

Deklariramo globalno strukturo, kjer definiramo funkcije modula za komunikacijo.

Primer:

```
static struct file_operations fops = {
      .read = ime_funkcije_read,
      .write = ime_funkcije_write,
      .open = ime_funkcije_open,
      .release = ime_funkcije_release,
      .ioctl = ime_funkcije_ioctl
};
```

Za vse funkcije v strukturi moramo napisati pripadajoče funkcije. Prototipi in parametri funkcij so naslednji:

```
static int ime_funkcije_open (struct inode *node , struct file *file);
static int  ime_funkcije_release (struct inode *node, struct file *file);
static ssize_t  ime_funkcije_read (struct file *filp, const char *buff, size_t
len, loff_t * offset );
static ssize_t ime_funkcije_write (struct file *filp, const char *buff,
size_t len, loff_t * offset );
int ime_funkcije_ioctl ( struct inode *inode, struct file *file, unsigned int
ioctl_num, unsigned long ioctl_param);
```

Potem sledi koda posmeznih funkcij. Primer:

```
static int ime_funkcije_open (struct inode *node , struct file *file)
{
    printk(KERN_INFO "Device open\n");
    return 0;    /*return je obvezen v vseh funkcijah*/
}
```

Ob inicializaciji modula v funkciji *_init* moramo registrirati napravo kot character device, ki tako lahko postane dostopna preko navidezne datoteke naprave (*device file*).

To storimo s funkcijo : *register_chrdev().* Primer uporabe:

```
      MajorNum = register_chrdev(0, DEVICE_NAME, &fops);
      printk(KERN_INFO "chrdev registered MajorNum= %d \n", MajorNum);
```

DEVICE_NAME je ime naprave. *Primer:*

```
#define DEVICE_NAME  "chardev"
```

Funkcija register_chrdev() vrne številko ***MajorNum.*** To številko rabimo pri kreiranju naprave, zato jo izpišemo s printk.

Ob koncu dela z modulom, v funkciji exit, moramo modul odregistrirati:

```
int ret = unregister_chrdev(MajorNum, DEVICE_NAME);
```

Za prenos podatkov iz kernel v uporabniški prostor lahko uporabimo :

```
raw_copy_to_user(buff,msg,len);
```

Za prenos podatkov iz uporabniškega v kernel proctor lahko uporabimo :

```
raw_copy_from_user(msg,buff,len);
```

Kreiranje naprave 'character device'

Če hočemo modul uporabljati kot 'character device' moramo kreirati pripadajočo enoto (device file) v */dev* direktoriju.

```
mknod /dev/ime_naprave   c   254 0
```

254 je številka naprave *MajorNum*, ki jo vrne prijava modula kot character device.

Pregled in normalno zaporedje funkcij:

Iz ukazne vrstice	Funkcija v aplikaciji	Funkcija v jedru	Komentar
`insmod hello.ko`		`module_init register_chardev()`	Vključi modul Registrira device
`mknod dev/chardev1`			Naredi device file
	`open`	`device_open`	Odpre device file
	`ioctl`	`device_ioctl`	
	`read`	`device_read`	
	`write`	`device_write`	
	`close`	`device_release`	Zapre device file
`rmmod hello`		`module_exit unregister_chardev()`	Odstrani modul

Note: An *ioctl*, which means "input-output control" is a kind of device-specific system call. A driver can define an ioctl which allows a userspace application to send it orders. However, ioctls are not very flexible and tend to get a bit cluttered and can also be insecure. An alternative is the *sysfs* interface or *netlink*.

Literatura : The Linux Kernel Module Programming Guide

http://microcross.com/lkmpg.pdf

http://www.faqs.org/docs/kernel/index.html

Sestavite modul tipa »character device«. Dodajte mu funkcije za odpiranje (*open*), branje (*read*) in pisanje (*write*) ~~ter komunikacijo z modulom (*ioctl*)~~. V vsako funkcijo dodajte izpis.

Modul prevedite (**make**) in vključite (**insmod**) v jedro kot pri prvi vaji.

Po vključitvi modula kreirajte pripadajočo napravo (*device file*) z ukazom **mknod** - glej navodila zgoraj**.**

```
mknod /dev/ime_naprave  c  254 0
```

254 je stevilka *MajorNum*, ki jo vrne prijava modula kot character device.

Preizkusite modul z ukazi iz ukazne lupine (shell) preko datotečnega sistema:

`cat /dev/ime_naprave`	Ukaz kliče funkcije open, read in close naprave.
`echo MojText >/dev/ime_naprave`	Ukaz kliče funkcije open, write in close naprave.

Preverite stanje in izpise. Opišite in razložite dogajanje :

3.VAJA: Uporabniški program, ki uporablja gonilnik

V uporabniškem prostoru sestavite program, ki bo preizkusil delovanje modula iz prejšnje vaje in preverite komunikacijo med uporabniškim prostorom in modulom v jedru. Funkcionalnost programa naj bo naslednja:

Program naj odpre napravo z ukazom ***open***.

```
int open(const char *pathname, int flags);
```

The parameter flags is one of O_RDONLY (0), O_WRONLY (1) or O_RDWR (2) which request opening the file read-only, write-only or read/write, respectively.

```
file_desc = open(DEVICE_FILE_NAME, 0);
```

Kjer je *DEVICE_FILE_NAME* /dev/ime_naprave.

Nato naj se izvede nekaj branj iz naprave in vpisov v napravo:

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

Po uporabi modula moramo pred izhodom programa napravo tudi zapreti z ukazom:

```
int close(int fd);
```

Izboljšajte modul in uporabniški program iz primerov in stestirajte uporabo write in read funkcij.

V uporabniškem prostoru seveda prevajamo z **gcc**.

Spodaj je izhodiščni primer kode.

```
#include <fcntl.h>      /* open */
#include <sys/ioctl.h>  /* ioctl */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#define DEVICE_FILE_NAME "/dev/vaja2"

int main()
{
  int file_desc, ret_val;
  char msg[100];
  char msgSend[100]="Text from Application\n";
/* Open device */
  printf("Open device file: %s\n", DEVICE_FILE_NAME);
  file_desc = open(DEVICE_FILE_NAME, 2);
  if (file_desc < 0) {
    printf ("Can't open device file: %s\n", DEVICE_FILE_NAME);
    exit(-1);
  }
/* Read from device */
  read(file_desc, msg, 100);
  printf("Read from kernel modul : %s \n", msg);
  if (ret_val < 0)
  {
    printf ("Read from kernel failed:%d\n", ret_val);
    exit(-1);
  }
/* Write to device */
  write(file_desc, msgSend, strlen(msgSend));
  printf("Write to kernel modul: %s \n", msgSend);
  if (ret_val < 0)
  {
    printf ("Write to kernel failed:%d\n", ret_val);
    exit(-1);
  }
/* Close device */
  sleep(1);
  close(file_desc);
  printf("Close device file: %s\n", DEVICE_FILE_NAME);
}
```

Preverite stanje in izpise. Opišite in razložite dogajanje :