PhiInnovations

Linux Device Drivers

Autor: Diego Rodrigues Thuler

Agosto - 2011

Linux Device Drivers

- Conteúdo:
  - Device Drivers
  - User Space x Kernel Space
  - Modules
  - Char Driver
  - Tutorial
    - Primeiros passos
    - Hello World Driver
    - Memory Driver
    - Peripheral Driver

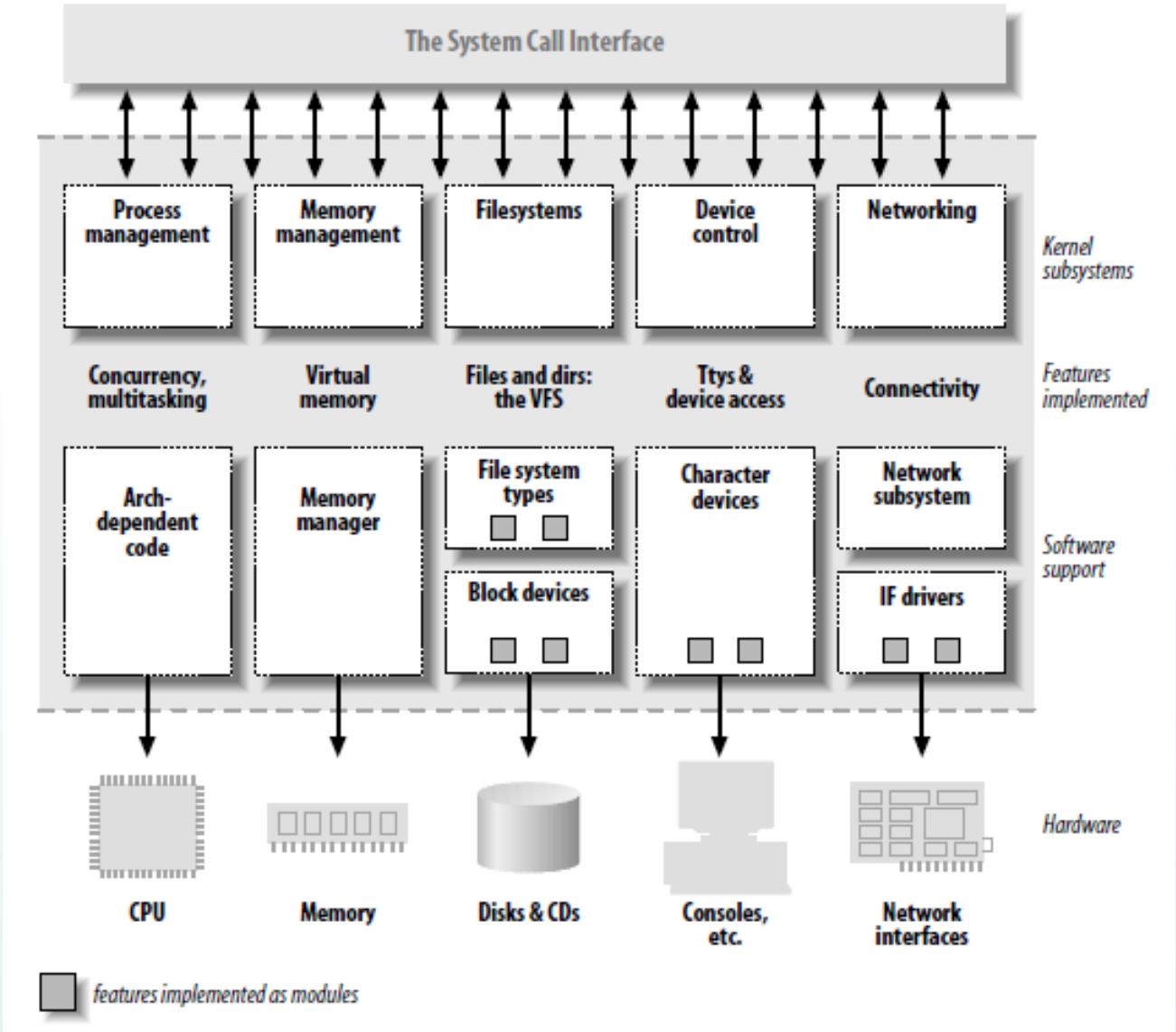
Device Drivers

“A função de um driver de dispositivo é aceitar requerimentos abstratos do software independente do dispositivo acima dele e cuidar para que a solicitação seja executada, permitindo que o software interaja com o dispositivo.”

Wikipedia

- Interface com o Hardware
- Modelo Caixa Preta
- Padronização de Interfaces
- Classes de drivers

Linux Device Drivers

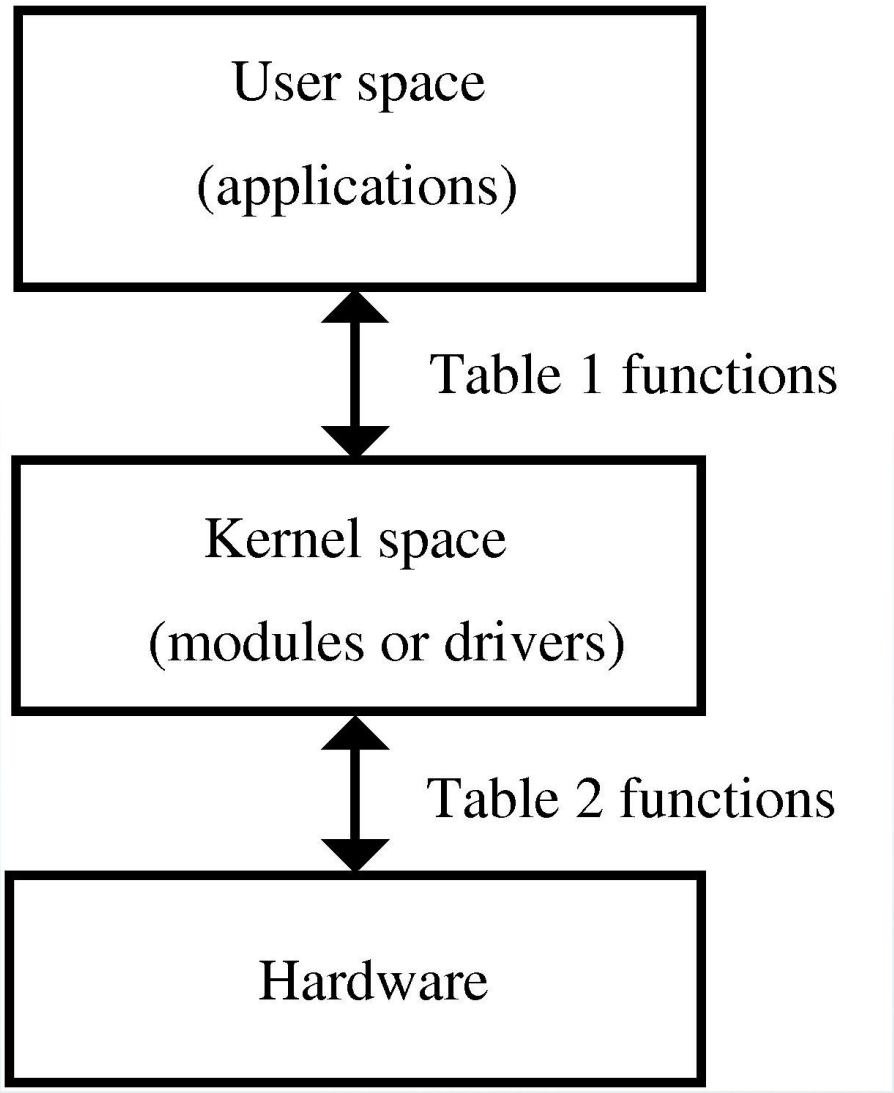


Linux Kernel

User Space x Kernel Space

- Kernel Space
  - Linux Kernel
  - Device Drivers
- User Space
  - Aplicações
  - Shell, GUI, etc

User Space x Kernel Space



User Space x Kernel Space

- Interface
  - Aplicação controlar dispositivo
  - Kernel API
  - Proteção do kernel
  - Modelo por classe - abstração

Modules

- Kernel driver x Module
- Execução em Kernel Space
- Carga em tempo de uso
- Flexibilização
- Compilação independente
- Compatibilidade com kernel

Char Driver

- Driver de caracter
- Popular
- Fluxo de bytes
- Transferência de dados
- Operações como arquivo
  - Open, close, read and write
- Exemplos: /dev/ttyS0, /dev/console
- Acesso sequencial

Char Driver Module

- Exporta funções de controle do driver para user space
- Passo a passo
- Tutorial

Events	User functions	Kernel functions
Load module		
Open device		
Read device		
Write device		
Close device		
Remove module		

Tutorial

- Compilando um módulo
  - Driver Nothing

<nothing.c> =

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

MODULE_LICENSE("Dual BSD/GPL");
```

- Compilando um módulo
  - Makefile

```
<Makefile1> =

obj-m := nothing.o
```

  - Comando Make

```
# make -C /usr/src/kernel-source-2.6.8 M=pwd modules
```

  - Objeto → nothing.ko

Primeiros Passos

- Carregando, listando e removendo módulo
  - Load Module

```
# insmod nothing.ko
```

  - List Modules

```
# lsmod
```

  - Remove Module

```
# rmmod nothing
```

- Primeiras funções de usuário

Events	User functions	Kernel functions
Load module	insmod	
Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	

Hello World Driver

<hello.c> =

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

MODULE_LICENSE("Dual BSD/GPL");

static int hello_init(void) {
  printk("<1> Hello world!\n");
  return 0;
}

static void hello_exit(void) {
  printk("<1> Bye, cruel world\n");
}

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

- module_init
- module_exit
- printk

Hello World Driver

- Makefile

  <Makefile1> =

  obj-m := nothing.o hello.o

- Saída → console

- Dmesg

Hello World Driver

- Primeiras funções de kernel: module_init e module_exit

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device		
Read device		
Write device		
Close device		
Remove module	rmmod	module_exit()

Memory Driver

- Driver completo
- Acesso à memória
- Didático

<memory initial> =

```
/* Necessary includes for device drivers */
#include <linux/init.h>
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h> /* printk() */
#include <linux/slab.h> /* kmalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <asm/system.h> /* cli(), *_flags */
#include <asm/uaccess.h> /* copy_from/to_user
*/

MODULE_LICENSE("Dual BSD/GPL");

...
```

Memory Driver

```
/* Declaration of memory.c functions */
int memory_open(struct inode *inode, struct file *filp);
int memory_release(struct inode *inode, struct file *filp);
ssize_t memory_read(struct file *filp, char *buf, size_t count, loff_t *f_pos);
ssize_t memory_write(struct file *filp, char *buf, size_t count, loff_t *f_pos);
void memory_exit(void);
int memory_init(void);

/* Structure that declares the usual file */
/* access functions */
struct file_operations memory_fops = {
  read: memory_read,
  write: memory_write,
  open: memory_open,
  release: memory_release
};
```

```
/* Declaration of the init and exit functions */
module_init(memory_init);
module_exit(memory_exit);

/* Global variables of the driver */
/* Major number */
int memory_major = 60;
/* Buffer to store data */
char *memory_buffer;
```

Memory Driver

- Estrutura FOPS
  - Funções do driver
- Major / Minor
  - Link do driver com sistema de arquivo
  - Criando arquivo de link

```
# mknod /dev/memory c 60 0
```

Memory Driver

<memory init module> =

```
int memory_init(void) {
  int result;

  /* Registering device */
  result = register_chrdev(memory_major, "memory", &memory_fops);
  if (result < 0) {
    printk("<1>memory: cannot obtain major number %d\n", memory_major);
    return result;
  }

  /* Allocating memory for the buffer */
  memory_buffer = kmalloc(1, GFP_KERNEL);
  if (!memory_buffer) {
    result = -ENOMEM;
    goto fail;
  }

  memset(memory_buffer, 0, 1);

  printk("<1>Inserting memory module\n");
  return 0;

  fail:
    memory_exit();
    return result;
}
```

Memory Driver

<memory exit module> =

```
void memory_exit(void) {

  /* Freeing the major number */
  unregister_chrdev(memory_major, "memory");

  /* Freeing buffer memory */
  if (memory_buffer) {
    kfree(memory_buffer);
  }

  printk("<1>Removing memory module\n");

}
```

Memory Driver

<memory open> =

```
int memory_open(struct inode *inode, struct file *filp) {

  /* Success */
  return 0;
}
```

<memory release> =

```
int memory_release(struct inode *inode, struct file *filp) {

  /* Success */
  return 0;
}
```

Memory Driver

- Abrindo: fopen → file_operations: open
- Fechando: fclose → file_operations: release

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device	fopen	file_operations: open
Read device		
Write device		
Close device	fclose	file_operations: release
Remove module	rmmod	module_exit()

Memory Driver

<memory read> =

```
ssize_t memory_read(struct file *filp, char *buf,
                    size_t count, loff_t *f_pos) {

  /* Transfering data to user space */
  copy_to_user(buf,memory_buffer,1);

  /* Changing reading position as best suits */
  if (*f_pos == 0) {
    *f_pos+=1;
    return 1;
  } else {
    return 0;
  }
}
```

Memory Driver

<memory write> =

```
ssize_t memory_write( struct file *filp, char *buf,
                      size_t count, loff_t *f_pos) {

  char *tmp;

  tmp=buf+count-1;
  copy_from_user(memory_buffer,tmp,1);
  return 1;
}
```

Memory Driver

- Lendo: fread → file_operations: read
- Escrevendo: fwrite → file_operations: write

Events	User functions	Kernel functions
Load module	insmod	module_init()
Open device	fopen	file_operations: open
Close device	fread	file_operations: read
Write device	fwrite	file_operations: write
Close device	fclose	file_operations: release
Remove module	rmmod	module_exit()

Memory Driver

<memory.c> =

<memory initial>
<memory init module>
<memory exit module>
<memory open>
<memory release>
<memory read>
<memory write>

- Testando
  - Carregando

```
# insmod memory.ko
```

  - Permissões

```
# chmod 666 /dev/memory
```

  - Escrevendo

```
# echo -n abcdef >/dev/memory
```

  - Lendo

```
# cat /dev/memory
```

Peripheral Driver

<parallel open> =

```
int parallel_open(struct inode *inode, struct file *filp) {

  if ( configurePins() == 0)
      return 0;
  else
      return -EIO;
}
```

<parallel release> =

```
int parallel_release(struct inode *inode, struct file *filp) {

  releasePins();
  return 0;
}
```

Peripheral Driver

<parallel read> =

```
ssize_t parallel_read(struct file *filp, char *buf,
                      size_t count, loff_t *f_pos) {

  int data, i;
  char *bufPos = buf;

  for(i=0;i<count;i++) {

    clearREpin();
    clearCSpin();
    data = readData();
    setCSpin();
    setREpin();

    copy_to_user(bufPos,&data,sizeof(int));
    buffPos += sizeof(int);
   }
   return count;

}
```

Peripheral Driver

<parallel write> =

```
ssize_t parallel_write( struct file *filp, char *buf,
                        size_t count, loff_t *f_pos) {

  int data, i;
  char *bufPos = buf;

  for(i=0;i<count;i++) {

    copy_from_user(&data,buffPos,sizeof(int));
    buffPos += sizeof(int);

    writeData(data);
    clearWEpin();
    clearCSpin();
    setCSpin();
    setWEpin();
   }
   return count;

}
```

Referências

- Linux Device Drivers - 3rd Edition (LDD3) - Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman - Ed. O'Reilly - 2005

- Linux Kernel

- Tutorial: http://www.freesoftwaremagazine.com/articles/drivers_linux#

Obrigado!

Perguntas?

diego.thuler@phiinnovations.com