Device Drivers

Drivers de Dispositivos no Linux

Gustavo Kishima
7971473

Drivers de Dispositivo

- “Caixas pretas” especiais
- Modularidade no Linux
- Pra quê te quero?
- Onde estão?
  - Gerenciamento de Processos
  - Gerenciamento de Memória
  - Sistema de Arquivos
  - Controle de Dispositivos (AQUI!)
  - Rede

O Papel do Driver

- Mecanismos X Políticas
  - Mecanismo?
  - Política?
  - Driver deve fornecer mecanismos!
  - Políticas são para altos níveis!

Devices (Dispositivos)

- *Character devices*
  - São lidos como um fluxo de *bytes*
  - Acessados por nós do sistema de arquivos (*filesystem nodes*)
  - Não podem ser lidos de forma aleatória, como um arquivo, devido a sua natureza de *data stream*
  - Há exceções! Quando se recebe um *frame* de dados através de um *lseek*¸ por exemplo
- *Block devices*
  - Acessados tais quais os *char devices* (*filesystem nodes*)
  - É um dispositivo que pode hospedar um sistema de arquivos
  - Na maioria dos sistemas *Unix*, lida apenas com *I/O* que transferem blocos de dado
  - *Linux* permite, contudo, que se possa escrever e ler num *stream*, como um *char device*
  - Assim, a única diferença entre eles no *Linux* é como os dados são manejados na interface *kernel/driver*

Devices (Dispositivos)

- *Network Interfaces (devices)*
  - Transações na rede são efetuadas por interfaces, que é um dispositivo que pode trocar dados com outros *hosts*
  - Pode ser *hardware* ou puro *software*, como a interface de *loopback*
  - Embora a maioria das conexões seja orientada a *stream* (especialmente aquelas que usam *TCP/IP*), interfaces de rede são desenhadas para trabalhar com pacotes
  - Devido a isso, é difícil mapear uma interface de rede para um *node* no *filesystem*
  - Um nome único é associado a interface (*eth0*), mas ela não possui entrada no sistema de arquivos
  - O *driver* não sabe das conexões individuais, ele apenas gerencia pacotes

Módulos

- Extensão do Kernel
- Pode ser adicionado ou removido em tempo de execução
- Onde um ponteiro errado pode ter consequências CATASTRÓFICAS
- Segmentation Fault é pouco!
- Embora ter uma funcionalidade não seja uma regra, lembre do O'Reilly!

"Good programmers, nonetheless, usually create a different module for each new functionality they implement, because decomposition is a key element of scalability and extendability" (O'Reilly)

Módulos

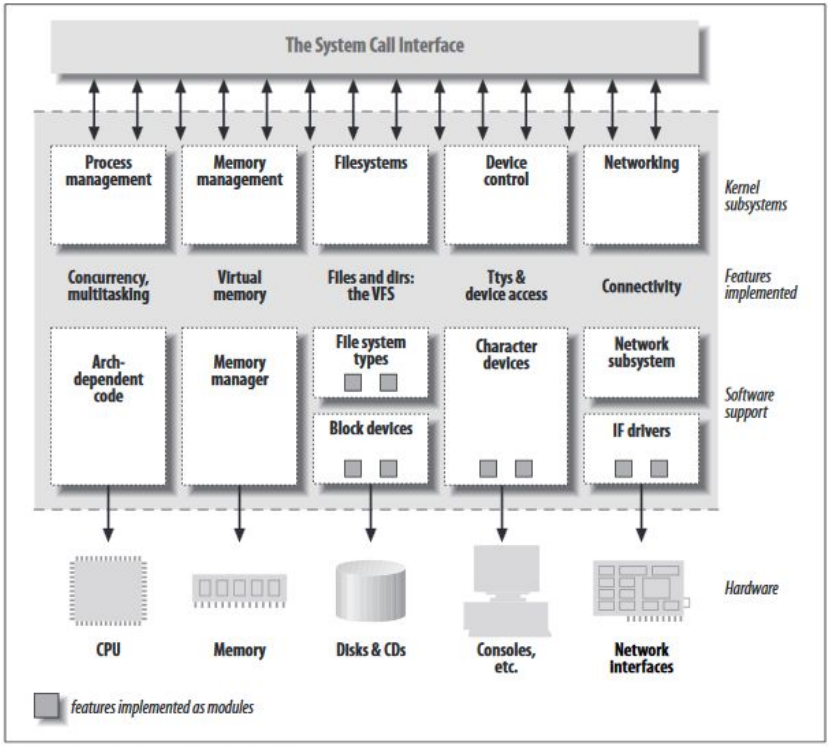

Figure 1-1. A split view of the kernel

Fonte: O'Reilly Linux Device Drivers; 3ª Edição

Módulos x Aplicações

- **Aplicações**: Executam uma ou duas funções e param
- **Módulos**: Registra-se como disponível e espera chamadas
  - Similar a programação orientada a eventos
- **Aplicações**: Podem chamar funções que não definem
- **Módulos**: Só podem chamar funções definidas por ele ou pelo “*kernel*”
  - Há pouquíssimas exceções

User Space x Kernel Space

- SO deve disponibilizar recursos do hardware aos programas e proteger os recursos de acessos não-autorizados
  - Só possível com reforço da CPU
  - No Unix dividido em dois níves (podem haver mais)
- Módulos funcionam em Kernel Space (tudo é permitido)
- Aplicações funcionam em User Space (acesso a recursos regulado por processador)
- Unix transfere execução de usuário para kernel quando há chamada de sistema ou interrupção de kernel
  - Chamada de sistema no caso roda em contexto do processo
  - Interrupção é independente dos processos
- *Driver* pode causar interrupção ou ser chamada do sistema

Concorrência no Kernel

- Aplicações comuns, com exceção das multithread, são sequenciais
  - “Kernel code does not run in such a simple world, and even the simplest kernel modules must be written with the idea that many things can be happening at once.” (O’Reilly)
- Dispositivos em geral podem causar interrupções a qualquer momento!
- E seu driver pode ser requisitado por mais de uma aplicação a qualquer momento!
- O código do *kernel* deve ser capaz de rodar em mais de um contexto ao mesmo tempo

Criando um módulo!

- Cabeçalhos do kernel são necessários
- Se estivermos falando sério, é bom ter várias versões de compiladores e outras ferramentas
- Devem sempre limpar tudo o que fizeram quando saírem!

“Olá mundo” dos módulos

```
#include <linux/module.h>       /* Needed by all modules */
#include <linux/kernel.h>       /* Needed for KERN_INFO */

int init_module(void)
{
        printk(KERN_INFO "Hello world 1.\n");

        /*
         * A non 0 return means init_module failed; module can't be loaded.
         */
        return 0;
}

void cleanup_module(void)
{
        printk(KERN_INFO "Goodbye world 1.\n");
}

/* Fonte: http://www.tldp.org/LDP/lkmpg/2.6/html/x121.html  */
```

Makefile

```
obj-m += hello-1.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



#Fonte : http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html
```

Informações, adicionando e removendo

```
/* Para compilar */

visitante@ubuntu-kernel-builder:~$ make

/* Verificando informações, adicionando ao kernel, removendo do kernel */

visitante@ubuntu-kernel-builder:~$ modinfo hello-1.ko

visitante@ubuntu-kernel-builder:~$ sudo insmod hello-1.ko

visitante@ubuntu-kernel-builder:~$ sudo rmmod  hello-1.ko

/* Para verificar as mensagens */

visitante@ubuntu-kernel-builder:~$ sudo tail /var/log/syslog
```

Referências

- https://www.kernel.org/ (Baixe seu kernel!)
- http://www.oreilly.com/openbook/linuxdrive3/book/ (Teoria geral)
- http://kernelnewbies.org/KernelBuild
- https://wiki.ubuntu.com/Kernel/BuildYourOwnKernel
- https://help.ubuntu.com/community/Kernel/Compile/
- http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html (Compilar módulos)
- https://help.ubuntu.com/community/Installation/MinimalCD#A64-bit_PC_.28amd64.2C_x86_64.29_.28Recommended.29 (Ubuntu Minimal)