Sistemas Operacionais

Gerenciamento de Entrada e Saída

Prof. Arliones Hoeller

arliones.hoeller@ifsc.edu.br

Junho de 2014

baseado no material do Prof. Fröhlich em
http://www.lisha.ufsc.br/~guto

Gerenciamento de E/S (*I/O*)

- Sistemas interativos são muitas vezes mais preocupados com E/S que processamento

- Dispositivos de entrada e saída (E/S)
  - Variam muito em funcionalidade e velocidade
  - Interfaces padrão de software e hardware ajudam a incorporar novos dispositivos
  - Novos dispositivos são introduzidos constantemente

- Driver de dispositivo (*device driver*)
  - Ponte entre subsistemas do SO e dispositivos
  - Encapsulam particularidades de um dispositivo entregando uma interface uniforme

Hardware de E/S

- Porta
  - Ponto de conexão entre *Host* e dispositivos de E/S
- Barramento (*Bus*)
  - Conjunto de fios compartilhados e protocolo que permite a conexão simultânea de vários dispositivos ao *Host*
- Controlador
  - Controla a operação de portas, barramentos e dispositivos
  - Varia de eletrônica simples a processadores complexos
  - Interage com *Host* através de registradores
    - Controle, estado, envio/recebimento de dados
    - Portas de E/S, mapeado em memória, mapeado em registrador da CPU

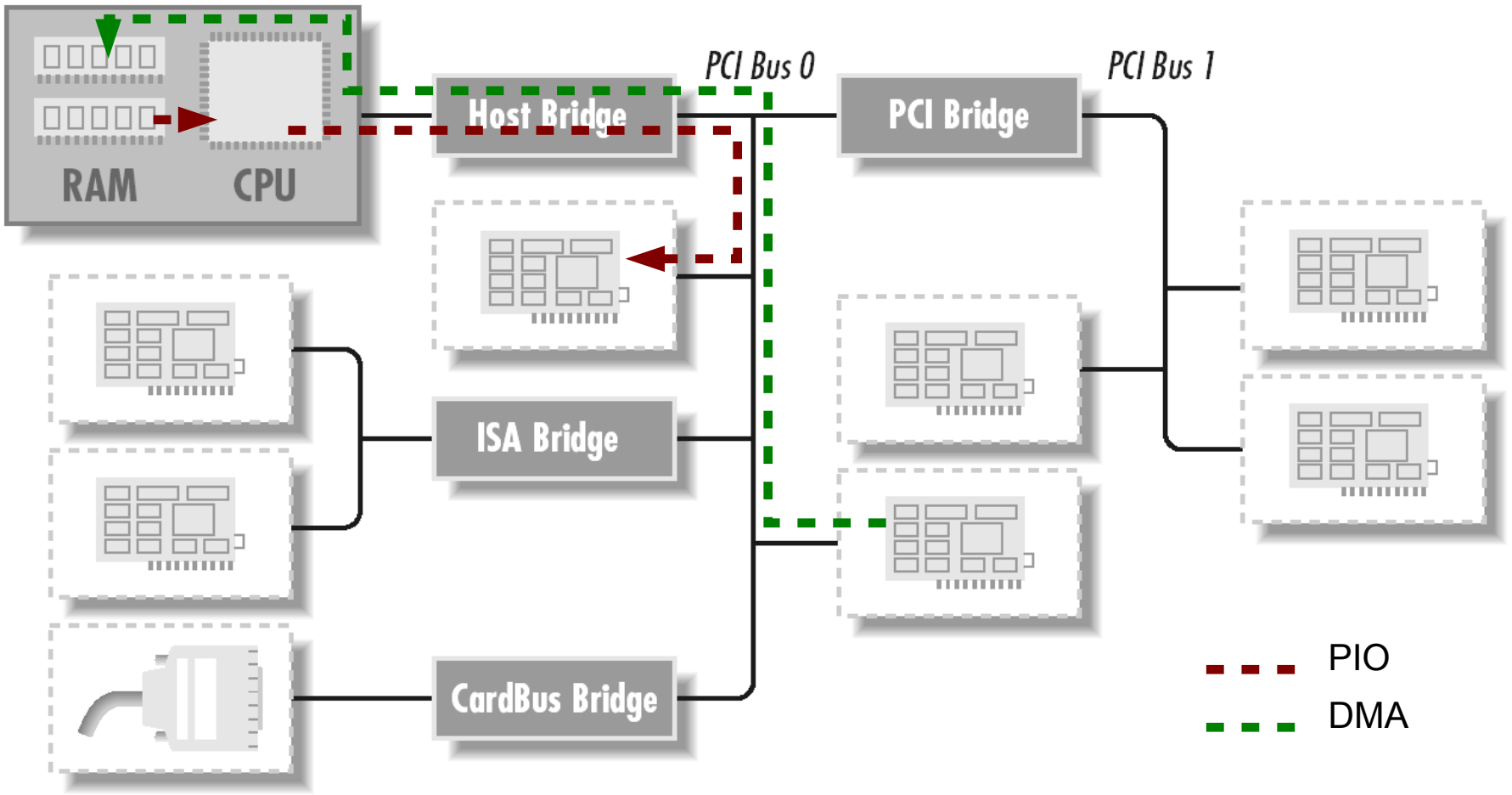
Operação de I/O

- *Polling*
  - *Host* verifica constantemente um registrador de estado para determinar o estado de um dispositivo
  - Espera ocupada (*Busy-waiting*)
    - Laço lendo um registrador de estado
    - Overhead em sistemas multitarefa
    - Simplicidade e eficiência em sistemas com única tarefa
- Interrupções
  - Evita espera ocupada
  - Dispositivo de E/S recebe uma requisição de serviço e gera uma interrupção quando a requisição é completada
  - Transparente aos processos

Transferência de dados em dispositivos de E/S

- E/S Programada
  - Dados são transferidos para/do dispositivo de E/S quando a CPU escreve/lê registradores de dados ou *buffers* de memória do controlador do dispositivo
  - Uma palavra por vez
- Direct Memory Access (DMA)
  - Acesso Direto à Memória
  - Dados são transferidos por circuito dedicado (Controlador de DMA) sem assistência da CPU
    - Ponteiros de origem e destino + contador
    - Transferências multi-palavras (*burst*)
    - Interrupções quando concluído ou na ocorrência de erro
  - Concorre com CPU pela memória
  - Armadilha
    - Tradução de endereço lógico -> físico ou
    - Address translation logical -> physical or DVMA

Hardware de E/S



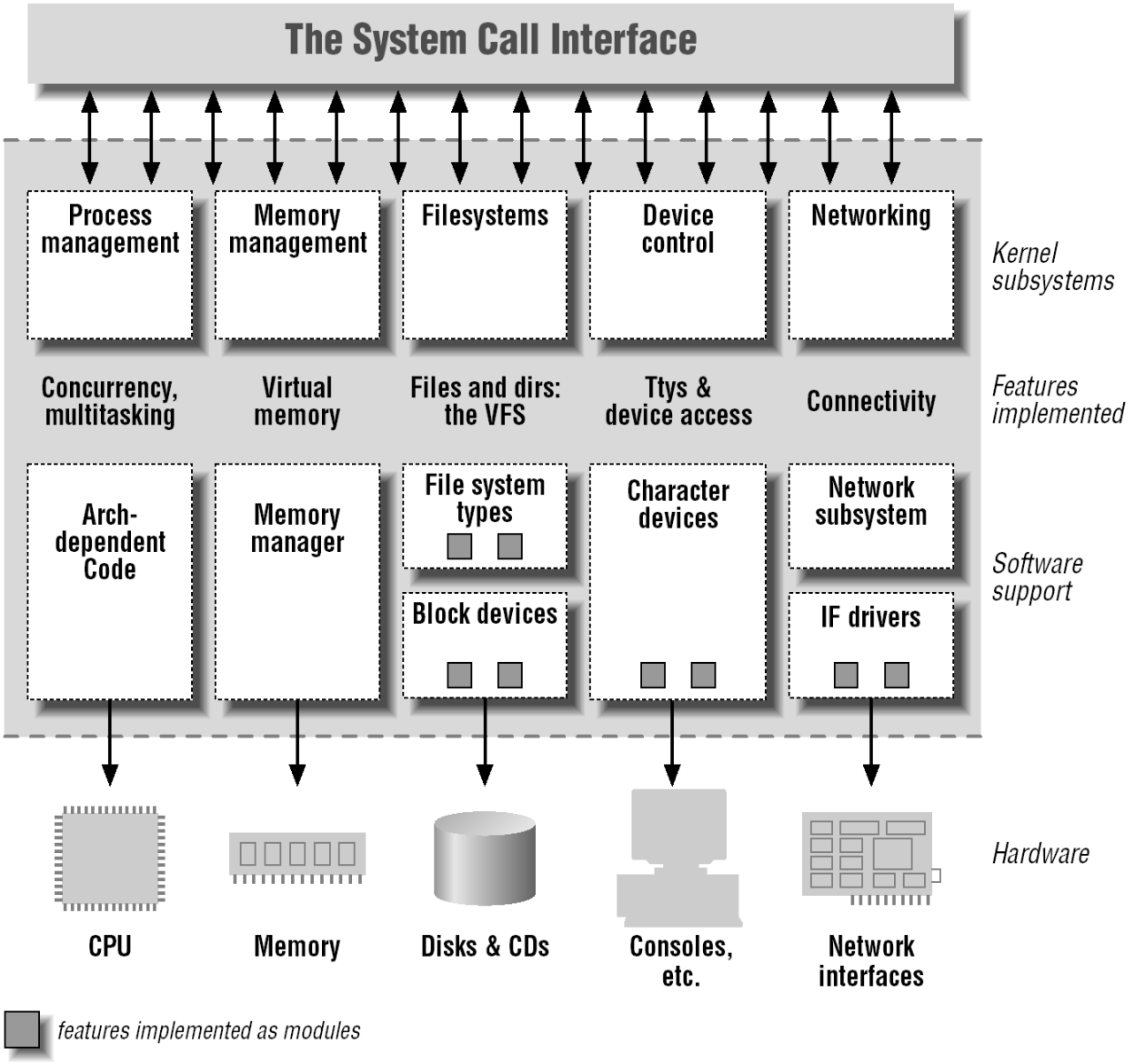
Interface de E/S para Aplicações

- Indireta via subsistemas de E/S
  - Um disco pode ser acessado indiretamente através dos arquivos contidos neles
  - Uma placa de rede pode ser acessada indiretamente através da pilha TCP/IP (*socket*)
- Pseudo-arquivo
  - Drivers de dispositivos se tornam tratadores de operações em “arquivos especiais” que são plugados ao sistema de arquivos (ex: /dev/mouse, /dev/sda, etc)
- Chamas de sistemas específicas
  - SO provê chamadas de sistema específicas para interagir com dispositivos de E/S (ex: man ioctl)

Drivers de Dispositivos no Unix (Linux)

- Módulo do núcleo do SO (*kernel*) que trata a interação com um dispositivo de hardware específico, escondendo seus detalhes operacionais atrás de uma interface comum
- Três categorias básicas
  - Caractere
  - Bloco
  - Rede

Visão Geral de Kernel: LINUX



Dispositovos de Hardware

- Acessível via pseudo-arquivos em `/dev`
- Kernel redireciona operações em pseudo-arquivos para o serviço apropriado de um driver considerando números *major* e *minor*
  - *Major*
    - Identifica um driver dentro do kernel (8 bits)
  - *Minor*
    - Identifica um dispositivo (unidade) dentro do driver

Char Devices

- Byte streams (ex: `/dev/console`, `/dev/ttyS0`, `/dev/st0`)
- Opera de forma similar a arquivos comuns
  - Sem possibilidade de busca para trás (*seek*)

Block Devices

- Dispositivos acessíveis através de blocos no nível de E/S
- Dispositivos relacionados a sistemas de arquivos (ex: discos)
- Compartilham uma interface comum com char devices, mas com semântica diferente
  - Orientado a bloco (acessar bytes individualmente é um desperdício)
  - É possível realizar varredura (*seek*)
- Operações adicionais para suporte a sistemas de arquivos

Net Devices

- Não é completamente compatível com a interface de pseudo-arquivos
  - Normalmente não é um nó em um sistema de arquivos
  - Integração com uma pilha de protocolos
- Interface genérica de redes
  - Operações relacionadas a comunicação (ex: send, receive, marshaling de pacotes, tratamento de time-out, coleção de estatísticas)
  - Otimizado para integração com TCP/IP

Módulo Hello World

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

int init_module(void)
{
  printk(KERN_INFO "Hello world\n");
  return 0;
}

void cleanup_module(void)
{
  printk(KERN_INFO "Bye world\n");
}
```

```
$ vim hello.c
$ vim Makefile
$ make
$ sudo insmod hello.ko
$ dmesg
$ sudo rmmod hello.ko
$ dmesg
```

```
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
```

Estrutura de um Módulo

- Inicialização

  `int init_module(void)`
  - Ponto de entrada do Módulo
  - Chamado no carregamento (por *insmod*)
  - Executa registro do Módulo
- Finalização

  `void cleanup_module(void)`
  - Ponto de saída do Módulo
  - Chamado no descarregamento (por *rmmod*)
  - Desfaz o registro do Módulo

Registro de um Módulo

- Amarra o módulo à interface de syscall do kernel
- Registro

```
int register_chrdev(unsigned int major, const char *name, struct file_operations *fops)
```

- Desregistro

```
int unregister_chrdev(unsigned int major, const char *name)
```

- Pseudo arquivo

```
mknod /dev/devname0 c major minor
```

Parâmetros de um Módulo

- Acessíveis externamente por variáveis globais ao módulo
- Declarado pela macro module_param(), que recebe 3 argumentos: nome, tipo e permissões

```
int irq = 10;
char * name = “Unknown”;
module_param( irq , int , S_IRUSR | S_IWUSR );
module_param( name, charp , 0000 );
```

- Definidos no carregamento

```
insmod mod.ko irq=9 name=”The Server”
```

Informações de um Módulo

- Declarações do módulo visíveis externamento para fornecer informações úteis aos clientes
- Macros

```
MODULE_AUTHOR("Somebody");
MODULE_DESCRIPTION("This module doesn't do
  anything");
MODULE_PARM_DESC(irq, "Device IRQ (3/4)");
```

struct file_operations

```
struct file_operations {
  struct module *owner;
  loff_t (*llseek) (struct file *, loff_t, int);
  ssize_t (*read) (struct file *, char *, size_t, ...
  ssize_t (*write) (struct file *, const char *, ...
  int (*readdir) (struct file *, void *, filldir_t);
  unsigned int (*poll) (struct file *, struct ...
  int (*ioctl) (struct inode *, struct file *, ...
  int (*mmap) (struct file *, struct vm_area_struct *);
  int (*open) (struct inode *, struct file *);
  int (*flush) (struct file *);
  int (*release) (struct inode *, struct file *);
  int (*fsync) (struct file *, struct dentry *, ...
  int (*fasync) (int, struct file *, int);
  int (*lock) (struct file *, int, struct file_lock *);
  :
  :
};
```

Dicas de Programação

- Sem bibliotecas padrão (e cabeçalhos)
  - *printk* ao invés de *printf*

```
#include <linux/x.h>
#include <asm/y.h>
```

- Sinalize código de kernel

```
#define __KERNEL__
```

- Evite conflitos de nomes
  - Variáveis locais em funções (*static*)
  - Símbolos com prefixos (_mod_name)

```
EXPORT_NO_SYMBOLS;
```

Mais Dicas de Programação

- Código de kernel executa dentro do contexto do processo de usuário que o chama

```
#include <asm/uaccess.h>
unsigned long copy_to_user(to, from, count);
unsigned long copy_from_user(to, from, count);
```

- Alocação de memória no kernel

```
#include <linux/malloc.h>
void *kmalloc(unsigned int size, int priority);
void kfree(void *obj);
```