

# Desenvolvimento de *Device Driver* para Comunicação com o Hardware

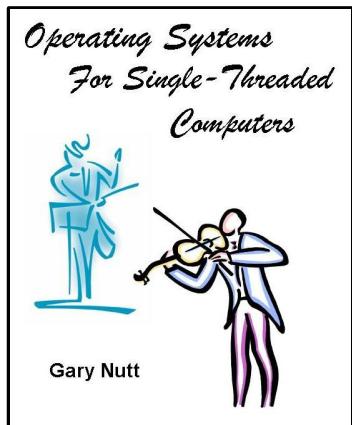
*Talles Marcelo G. de A. Barbosa*

Da aula passada...

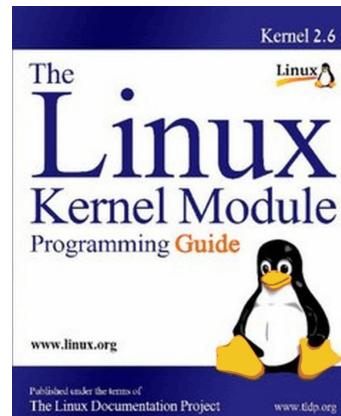
Um dispositivo de E/S é um Sistema Embarcado?



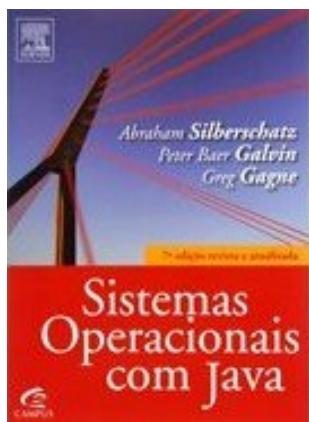
## Referências sobre o tema



Nutt G. *Operating Systems for Single-Threaded Computers*. Disponível em: [www.cs.colorado.edu/~nutt/ebBooklets/ST-Systems/index.html](http://www.cs.colorado.edu/~nutt/ebBooklets/ST-Systems/index.html). Em: 25/11/2010

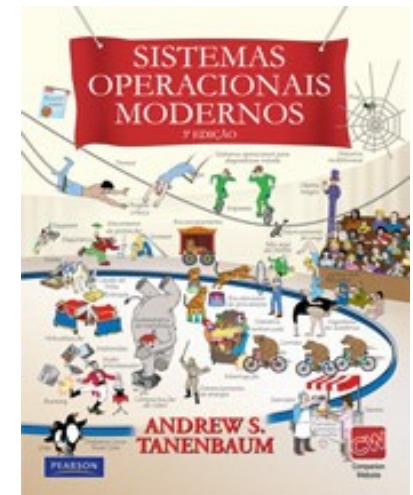
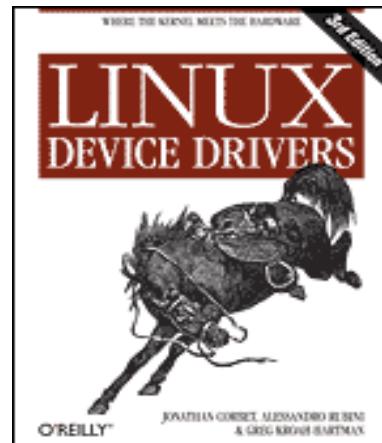


Salzman J. P., Burian M. e Pomerantz O. *The Linux Kernel Module Programming Guide*. Disponível em: <http://tldp.org/LDP/lkmpg/2.6/html/>. Em: 25/11/2010



Silberschatz A., Galvin B. P. E Gagne G. *Sistemas Operacionais com Java*. 7a. Edição. Editora Elsevier, 2008.

Salzman J. P., Burian M. e Pomerantz O. *Linux Device Drivers*. Disponível em: Em: 25/11/2010



Silberschatz A., Galvin B. P. E Gagne G. *Sistemas Operacionais com Java*. 7a. Edição. Editora Elsevier, 2008.

## Objetivos da Aula

Possibilitar ao expectador a compreensão de aspectos básicos relacionados aos seguintes assuntos:

- O conceito de *Device Driver*
- A utilidade da modularização
- Implementação de *Device Drivers* no Linux
- Desenvolvimento de *Device Drivers* tipo *character device*

Pré-requisitos:

- Compreender conceitos básicos e terminologia de organização de sistemas de computação, especialmente, de sistemas operacionais
- Conhecimento básico de programação em linguagem de montagem e em linguagem C
- Conhecimento básico sobre o Linux.

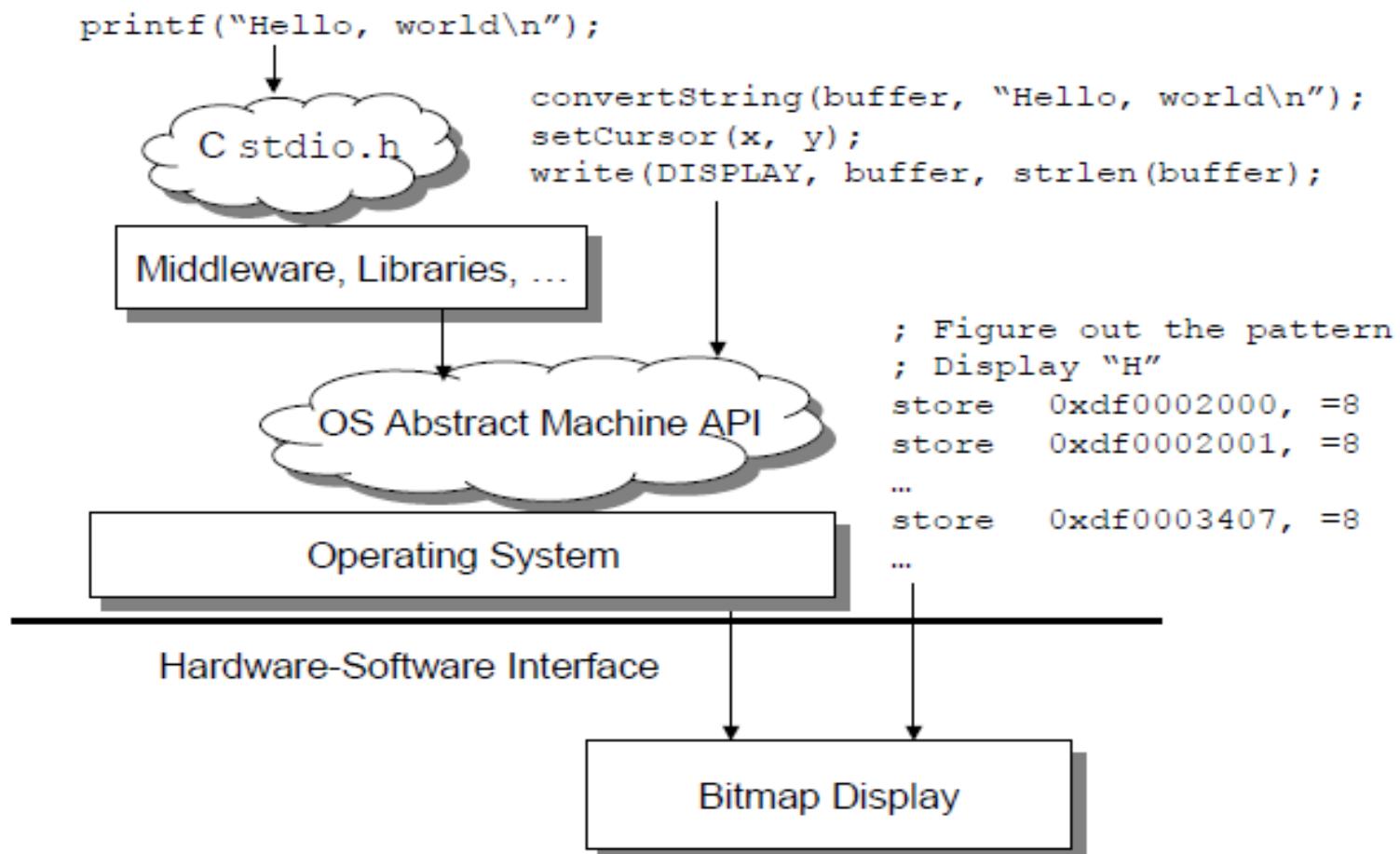
## Conteúdo

- Definições e Conceitos
- Estudo de casos: *Device Drivers* no Linux
  - Modelos de Implementação
  - Módulos do Kernel
  - Desenvolvimento de *drivers* tipo *character device*
  - Exemplos

## Definições: Sistema Operacional

*An OS is the part of the software that interacts directly with the computer hardware to export an abstraction of the hardware behavior using an **application programming interface (API)***

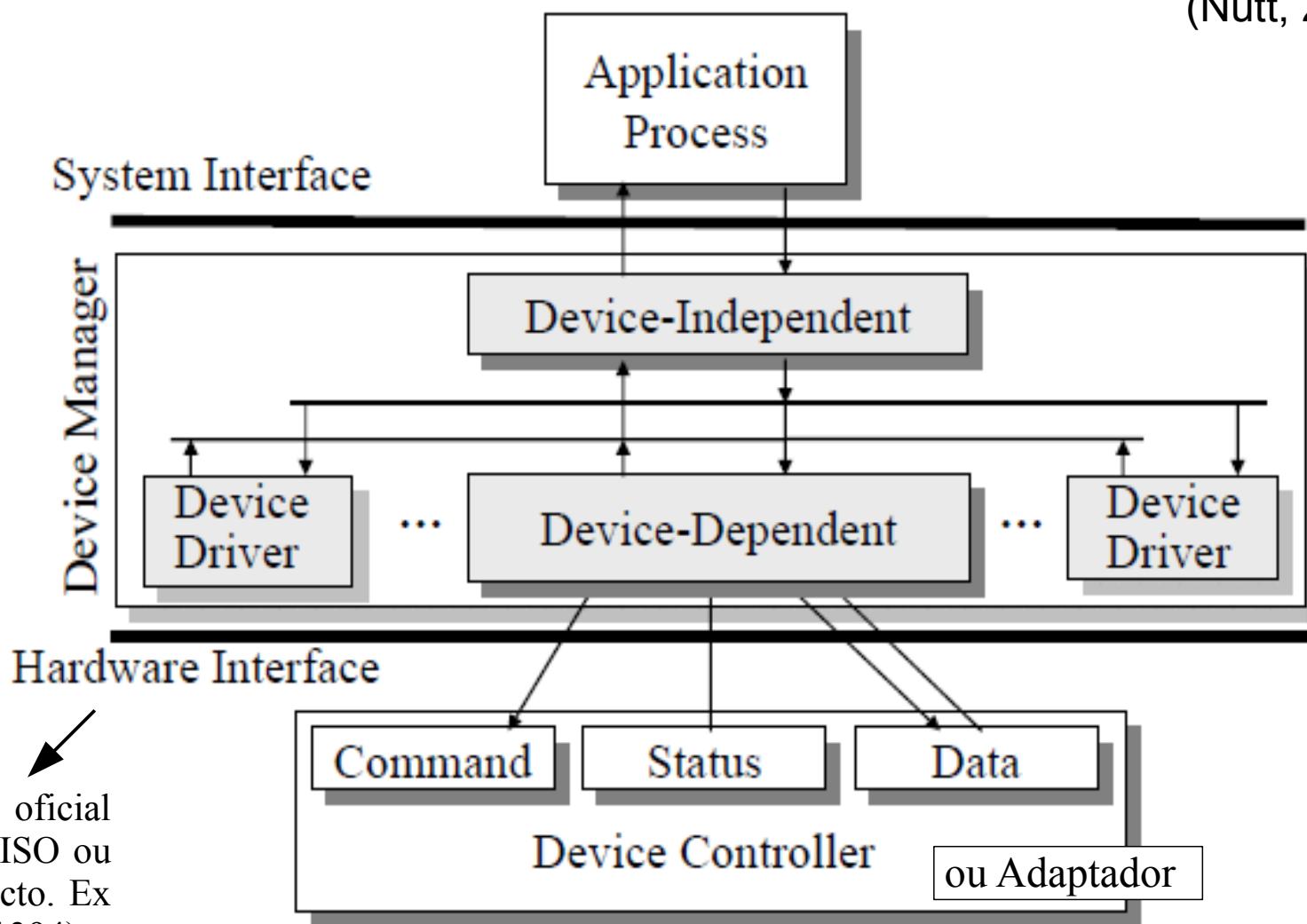
(Nutt, 2008)



## Definições: *Device Driver*

*A device driver is a collection of device-dependent functions that abstracts the operation of the device to a fixed interface.*

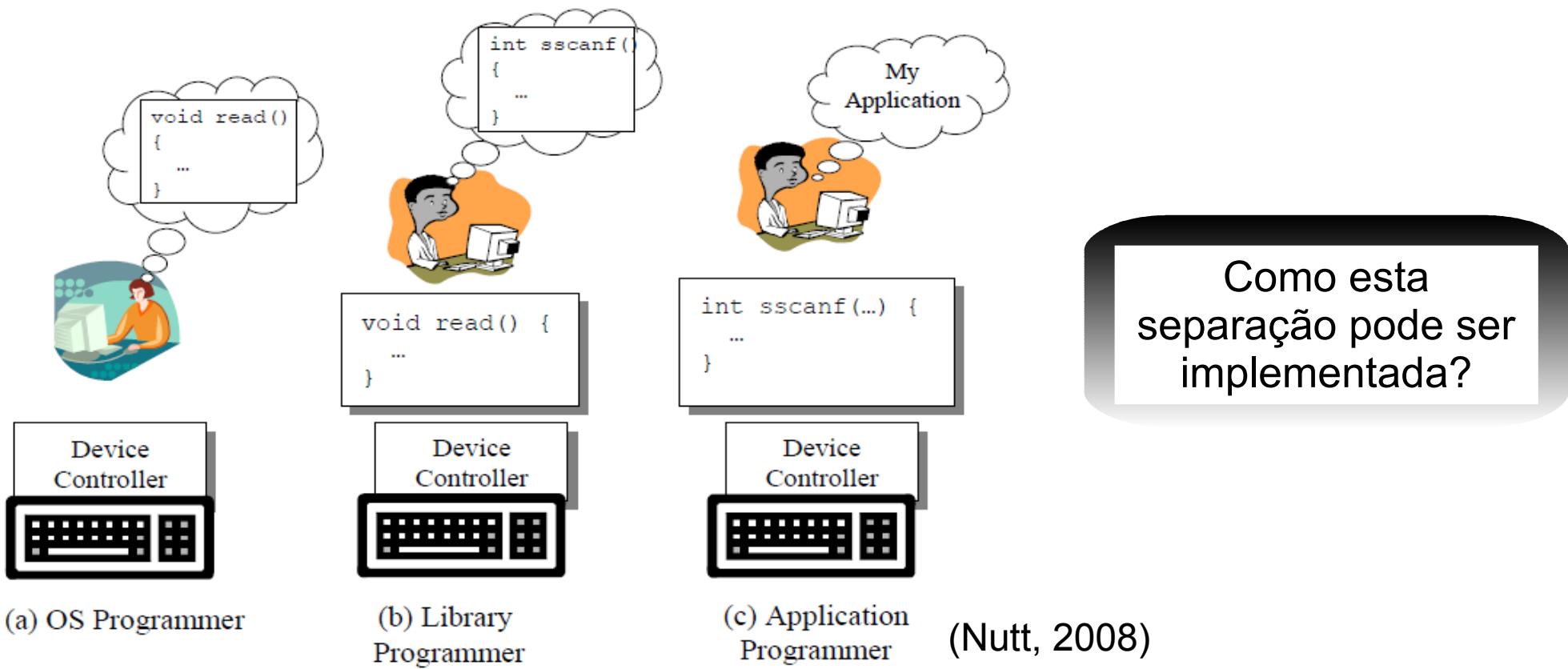
(Nutt, 2008)



## Definições: *Device Driver*

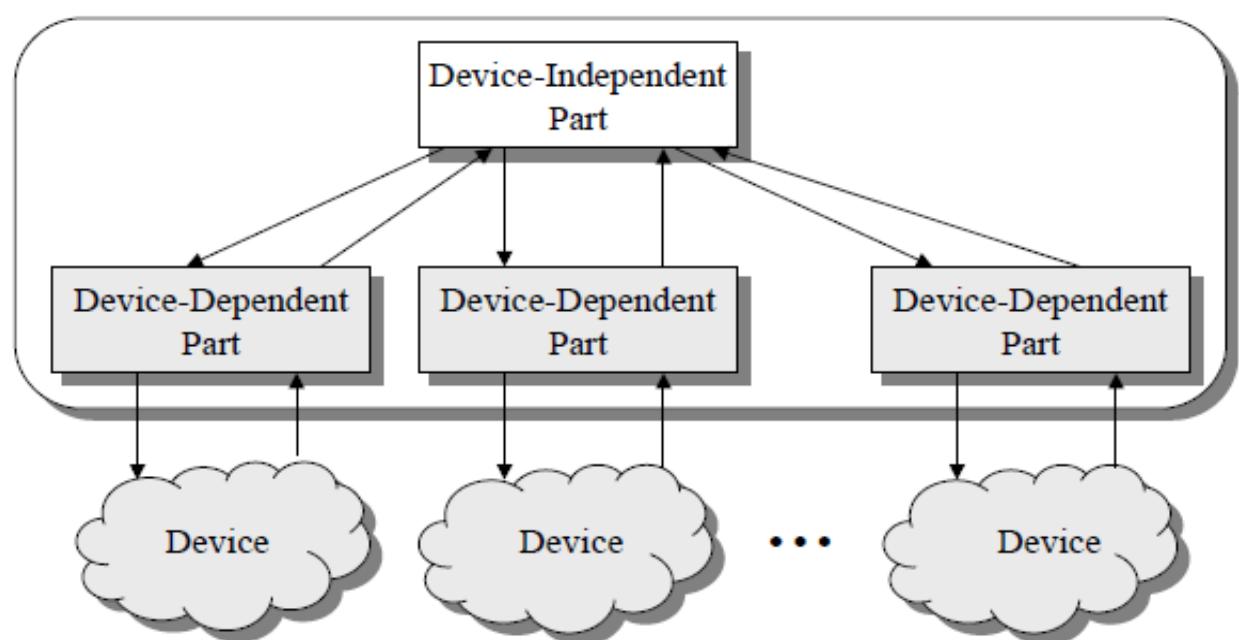
Por que separar em *device-independent* e *device-dependent* ?

- Independência da plataforma (do hardware)
  - Modificabilidade
  - Reuso
- Aumento de desempenho (customização do fabricante)



## Definições: *Device Manager*

O Gerenciador de dispositivos: *device drivers* + infraestrutura de gerência



(Nutt, 2008)

Mecanismos + Políticas  
coerentes ao propósito  
da cada Sistema  
Operacional

Uniform interfacing for device drivers
Buffering
Error reporting
Allocating and releasing dedicated devices
Providing a device-independent block size

(Tanenbaum *et al.*, 2010)

Uma alternativa para implementação: **padronização de interfaces**  
Ex. IEEE Portable Operating System Interface for UNIX (POSIX) - IEEE Std 1003.1-1988

## Parêntesis: O padrão



Certified by IEEE and The Open Group

**POSIX.1**: incorpora o padrão ANSI C (IEEE Std 1003.1-1988 ou ISO/IEC 9945)

**POSIX.1b** - IEEE Std 1003.1b-1993: basicamente os mecanismos para IPC

**POSIX.1c** - IEEE Std 1003.1c-1995: basicamente suporte à biblioteca <pthreds.h>

**POSIX 2** - IEEE Std 1003.2-1992: interpretador de comandos (shell) e alguns utilitários

**POSIX: 2001** - IEEE Std 1003.1-2001: equalizou com o *Single Unix Specification 3*

**POSIX: 2004** - IEEE Std 1003.1-2004: atualização do POSIX:2001

**POSIX: 2008**: IEEE Std 1003.1-2008: reune definições de base, interfaces de sistema cabeçalhos, comandos e utilitários

## Em relação aos Sistemas Operacionais

**Fully POSIX-compliant**: A/UX, Solaris, HP-UX, IRIX, MAC OS X, MINIX, QNX,...

**Mostly POSIX-compliant**: FreeBSD, OpenBSD, OpenSolaris, LINUX (maioria das distribuições – *Linux Standard Base*),...

**Compliant via compatibility feature**: eCOS, OpenVMS, Symbian,...

**POSIX for Windows**: windows NT Kernel quando instalado o INTERIX (*Windows Services for UNIX (SFU) or Subsystem for UNIX-based Applications (SUA)*), substituto do Microsoft POSIX Subsystem; CygWin (<http://www.cygwin.com/>), UWIN (da AT&T): <http://www2.research.att.com/sw/tools/uwin/>

## Definições: padronização de interfaces

UNIX	Win32	Description
fork	CreateProcess	Create a new process
waitpid	WaitForSingleObject	Can wait for a process to exit
execve	(none)	CreateProcess = fork + execve
exit	ExitProcess	Terminate execution
open	CreateFile	Create a file or open an existing file
close	CloseHandle	Close a file
read	ReadFile	Read data from a file
write	WriteFile	Write data to a file
lseek	SetFilePointer	Move the file pointer
stat	GetFileAttributesEx	Get various file attributes
mkdir	CreateDirectory	Create a new directory
rmdir	RemoveDirectory	Remove an empty directory
link	(none)	Win32 does not support links
unlink	DeleteFile	Destroy an existing file
mount	(none)	Win32 does not support mount
umount	(none)	Win32 does not support mount
chdir	SetCurrentDirectory	Change the current working directory
chmod	(none)	Win32 does not support security (although NT does)
kill	(none)	Win32 does not support signals
time	GetLocalTime	Get the current time

## Definições: passagem de parâmetros

Ex. Linux

teste.c

```
# include <unistd.h>      /*acesso às interfaces POSIX padrão POSIX.1*/
char msg []="Aula de SO\n";
Int cnt, len=12

/*protótipo da chamada: ssize_t write(int fd, const void *buff, size_t count); */

/* exemplo de uso: write (1,msg, len); /
```

Modo	Artefato	comandos da linguagem	
Usuário	Processo do Usuário	<code>write (1, msg, len)</code>	<code>syscall (write, 1, msg, len)</code>
	libc	<code>movl len, %edx</code> <code>movl \$msg, %ecx</code> <code>movl \$1, %ebx</code> <code>movl \$4, %eax</code>	#Passagem dos #argumentos da libc para #os registradores da CPU
		<code>int 0x80</code>	#trap para o chaveamento de modo
	<i>kernel</i>	Acessa à operação "write ( )" implementada pelo <i>driver</i>	
	<i>device driver</i>	Acessa diretamente os registradores do controlador para executar a operação solicitada	

# Desenvolvimento de *Device Driver* para Comunicação com o Hardware

```
#include <syscall.h>

char msg[]="Aula de SO\n";
int cnt, len=12;

int main(){
    syscall(SYS_write, 1, msg, len);
    return 0;
}
```

Alternativa para implementar uma chamada no âmbito do Usuário

usando uma função geral para chamadas ao sistema:  
`syscall(...)`, definida em:  
`<syscall.h>`

## Em assembler

```
char msg[]="Aula de SO\n";
int cnt, len=12;
int main(){
```

Mais de 6 argumentos, usa-se a pilha e um ponteiro para indexar o bloco

```
__asm__ ("movl len, %edx #define o tamanho da mensagem \n"
         "movl $msg, %ecx #ponteiro para a mensagem \n"
         "movl $1, %ebx #descriptor utilizado \n"
         "movl $4, %eax #identificador da chamada (sys_write) \n"
         "int $0x80 #mudança para modo supervisor \n"
         "movl %eax, cnt #retorna o no. de bytes transferidos ou -1
em caso de erro \n"
         );
return 0; }
```

Acorrepondência entre `SYS_write = 4`  
está em `<syscall.h>`

## Definições: gerenciador de dispositivos (cont.)

Modelo *Unix-like*: dispositivos são tratados como arquivos e acessados pelas mesmas interfaces

```
{..int dispositivo = open(/dev/audio, O_RDONLY);...}
```

descriptor  
retornado

/\*acessa o i-node (recebido como parâmetro)  
e descobre que o arquivo associado é um  
dispositivo de E/S\*/

Gerenciador de arquivos

resultado da operação

consulta

Tabela de  
chaveamento  
do dispositivo  
*trap table*

/\*obtém o endereço da rotina  
que implementa o driver do  
dispositivo\*/

aciona

/\*uma vez iniciado o dispositivo,  
um campo da tabela de arquivos  
é adicionado. O índice desse  
campo (descriptor) é retornado ao  
processo\*/

Driver da placa de som



## Definições: gerenciador de dispositivos (cont.)

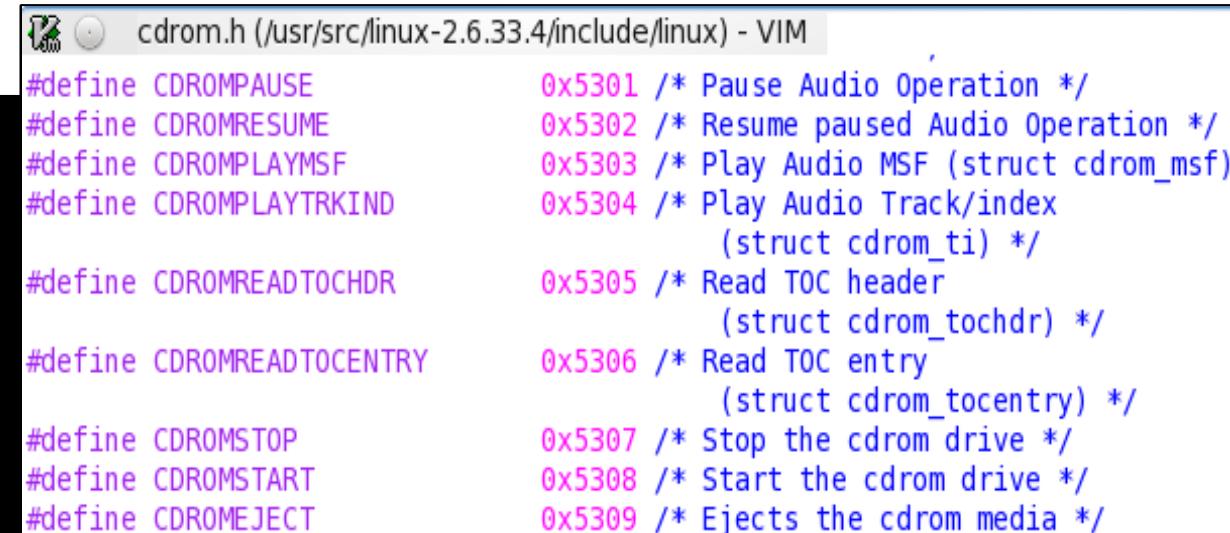
Interface de propósito geral: **IOCTL (SHORT FOR INPUT OUTPUT CONTROL)**

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/cdrom.h>

int main(void) {
    int fd;

    /* Abre o Drive de Cd-Rom para leitura */
    if ((fd = open("/dev/cdrom", O_RDONLY)) < 0) {
        perror("Erro ao abrir o CD-ROM");
        Exit(1);

    /* Envia o comando ao CD-ROM */
    if (ioctl(fd, CDROMEJECT, 0) < 0) {
        perror("Erro ao acessar o CD-ROM");
        exit(1);
    }
}
```



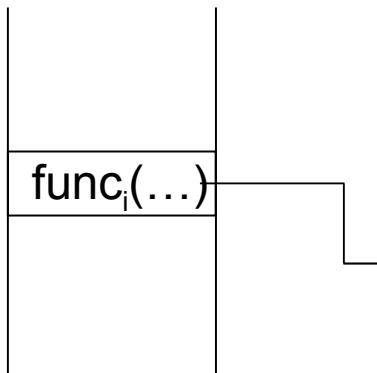
#define CDROMPAUSE	0x5301 /* Pause Audio Operation */
#define CDROMRESUME	0x5302 /* Resume paused Audio Operation */
#define CDROMPLAYMSF	0x5303 /* Play Audio MSF (struct cdrom_msf)
#define CDROMPLAYTRKIND	0x5304 /* Play Audio Track/index (struct cdrom_ti) */
#define CDROMREADTOCHDR	0x5305 /* Read TOC header (struct cdrom_tochdr) */
#define CDROMREADTOCENTRY	0x5306 /* Read TOC entry (struct cdrom_tocentry) */
#define CDROMSTOP	0x5307 /* Stop the cdrom drive */
#define CDROMSTART	0x5308 /* Start the cdrom drive */
#define CDROMEJECT	0x5309 /* Ejects the cdrom media */

```
#gcc -o cdrom cdrom.c
```

```
./cdrom
```

## Definições: gerenciador de dispositivos (cont.)

Trap Table



**Problema:** para adicionar um novo dispositivo, o mapeamento precisa ser editado e recompilado com os demais artefatos do SO

*i=indexa o dispositivo em questão*

```
dev_func_i(devID, ...) {  
    // Processing common to all devices  
    ...  
    switch(devID) {  
        case dev0: dev0_func_i(...);  
                    break;  
        case dev1: dev1_func_i(...);  
                    break;  
        ...  
        case devM: devM_func_i(...);  
                    break;  
    };  
    // Processing common to all devices  
    ...  
}
```

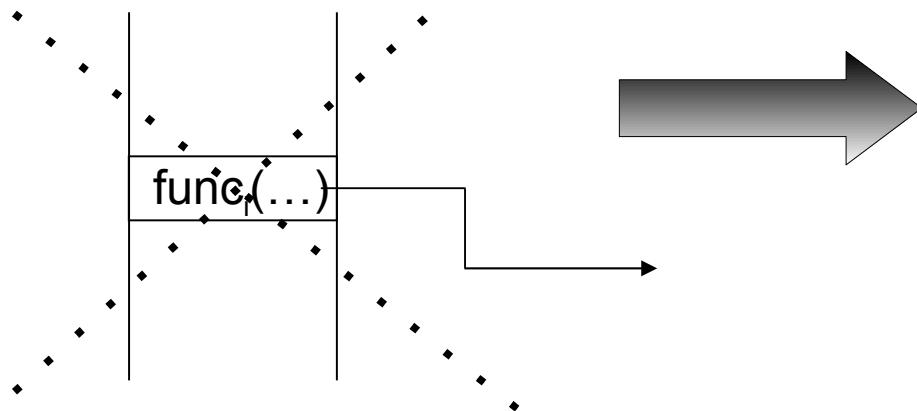
## Definições: gerenciador de dispositivos (cont.)

**Solução:** utilizar uma *irection table* para possibilitar a ligação dinâmica dos device drivers sem perder a independência do hardware

**Vantagem:** endereço pode ser reescrito na tabela em tempo de execução

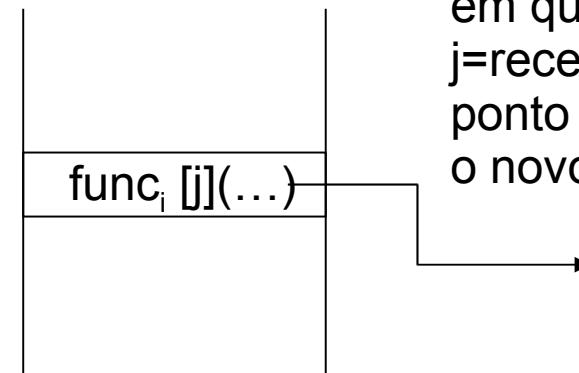
### Reconfigurable Device Driver

Trap Table



(modificado Nutt, 2008)

Trap Table

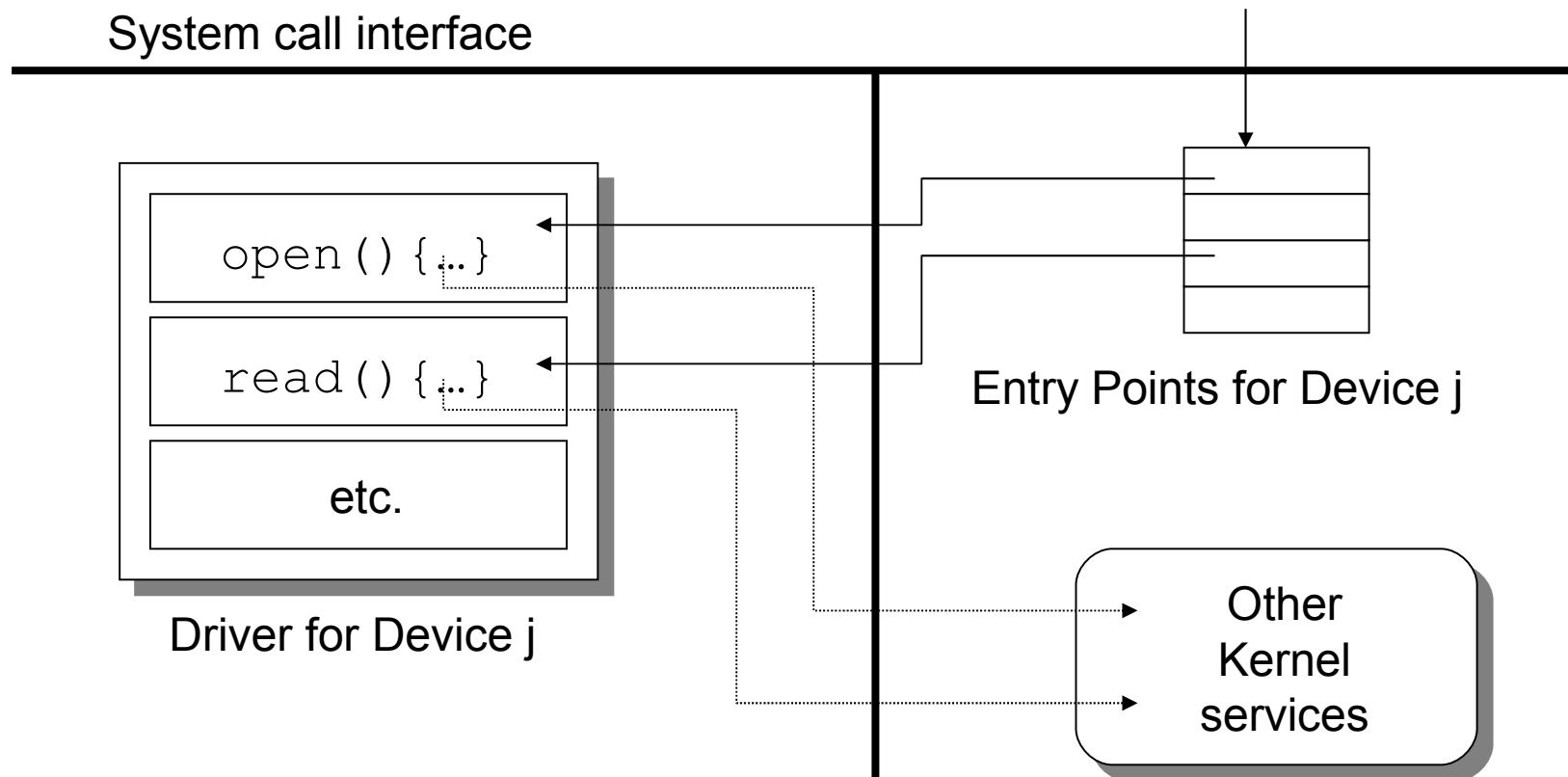


i=indexa o dispositivo em questão  
j=recebe o endereço do ponto de entrada para o novo dispositivo

**E para os dispositivos que utilizam interrupções?**  
Outra tabela para indexar o endereço do tratador de interrupções do novo dispositivo

## Definições: gerenciador de dispositivos (cont.)

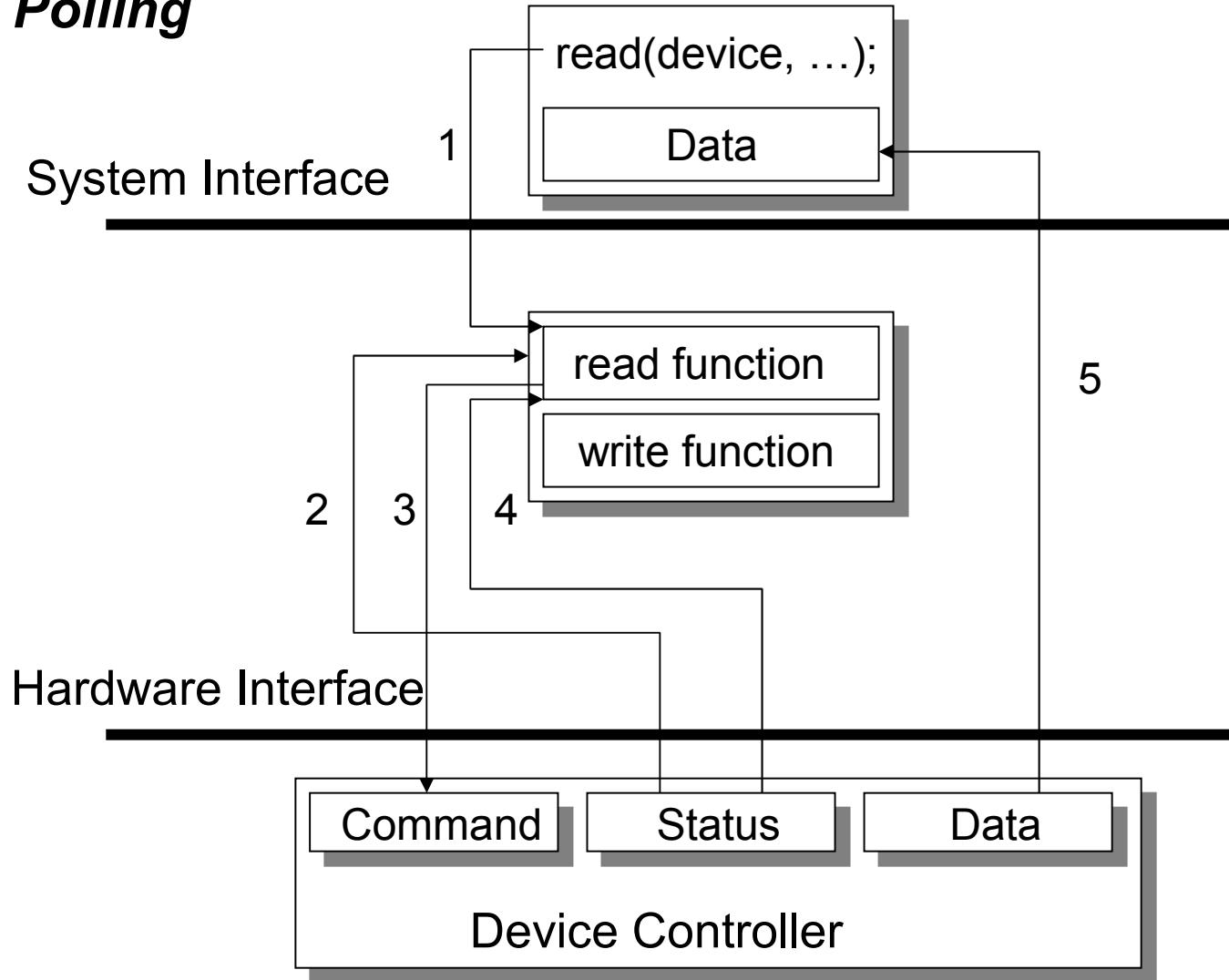
### *Reconfigurable Device Driver*



(Nutt, 2008)

## Definições: estratégias de E/S

### E/S com *Polling*



(Nutt, 2008)

## Definições: estratégias de E/S

### E/S com *Polling*

```
wait: IN    STATUS, %AX
      CMP   %AX, 0
      JZ    ready
      JMP   wait
ready:
loop:
      MOV   $LENGTH, %EDX
      MOV   (%ECX), %EAX
      MOV   %EAX, (%EBX)
      INC   %EBX
      INC   %ECX
      DEC   %EDX
      JNZ   loop
```

Espera que o valor do registrador de STATUS (do controlador) seja nulo (igual a 0). Isso indica que a informação está disponível

**Transfere dados para a Memória Principal**

Armazena a qtde. de dados a transferir

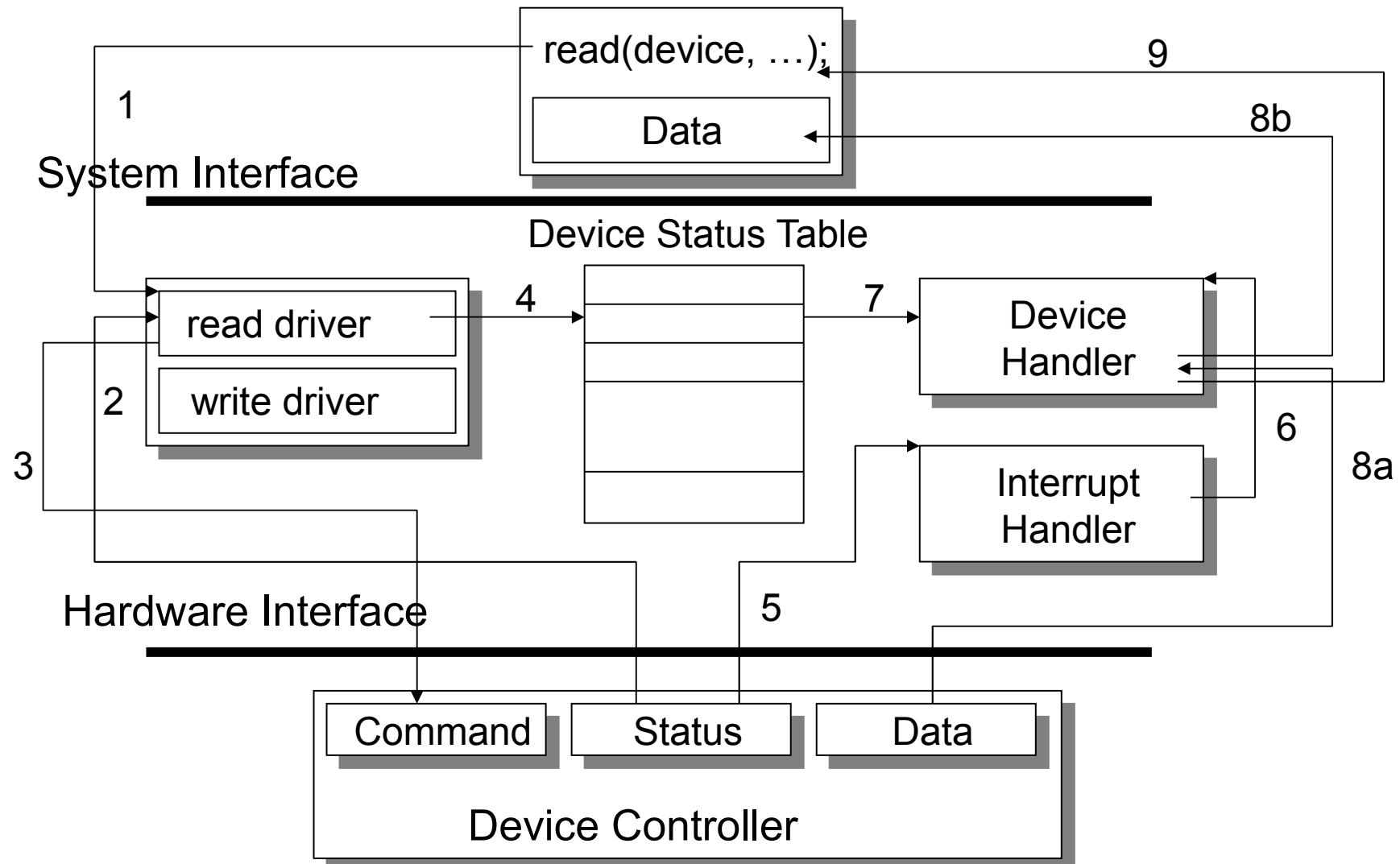
Localização do buffer apontado por ECX

Localização da memória apontada por EBX

\*também conhecida como **Espera Ocupada** ou, em inglês, **Busy Waiting**

## Definições: estratégias de E/S

### E/S com Interrupções



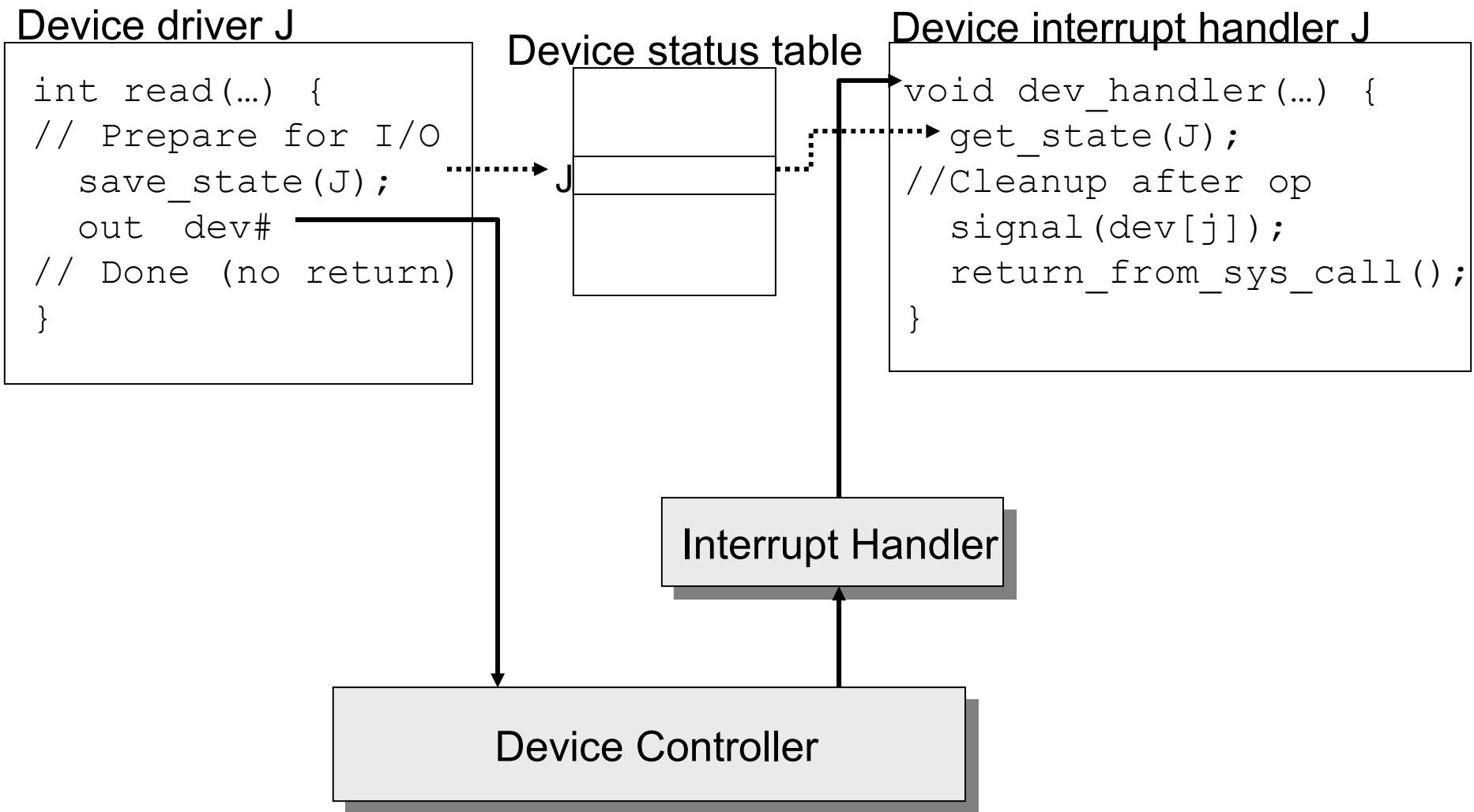
## Definições: estratégias de E/S

### E/S com Interrupções

```
talles@ubuntu:/proc$ cat /proc/interrupts
          CPU0
 0:      147  IO-APIC-edge      timer
 1:      960  IO-APIC-edge      i8042
 3:      1  IO-APIC-edge
 4:      1  IO-APIC-edge
 6:      5  IO-APIC-edge      floppy
 7:      0  IO-APIC-edge      parport0
 8:      0  IO-APIC-edge      rtc0
 9:      0  IO-APIC-fasteoi    acpi
 12:    2017  IO-APIC-edge      i8042
 14:      0  IO-APIC-edge    ata_piix
 15:  11131  IO-APIC-edge    ata_piix
 16:    3384  IO-APIC-fasteoi  Ensoniq AudioPCI
 17:    9202  IO-APIC-fasteoi  ehci_hcd:usb1, ioc0
 18:      53  IO-APIC-fasteoi  uhci_hcd:usb2
 19:    667  IO-APIC-fasteoi    eth0
 24:      0  PCI-MSI-edge    pciehp
 25:      0  PCI-MSI-edge    pciehp
 26:      0  PCI-MSI-edge    pciehp
 27:      0  PCI-MSI-edge    pciehp
 28:      0  PCI-MSI-edge    pciehp
 29:      0  PCI-MSI-edge    pciehp
 30:      0  PCI-MSI-edge    pciehp
```

## Definições: estratégias de E/S

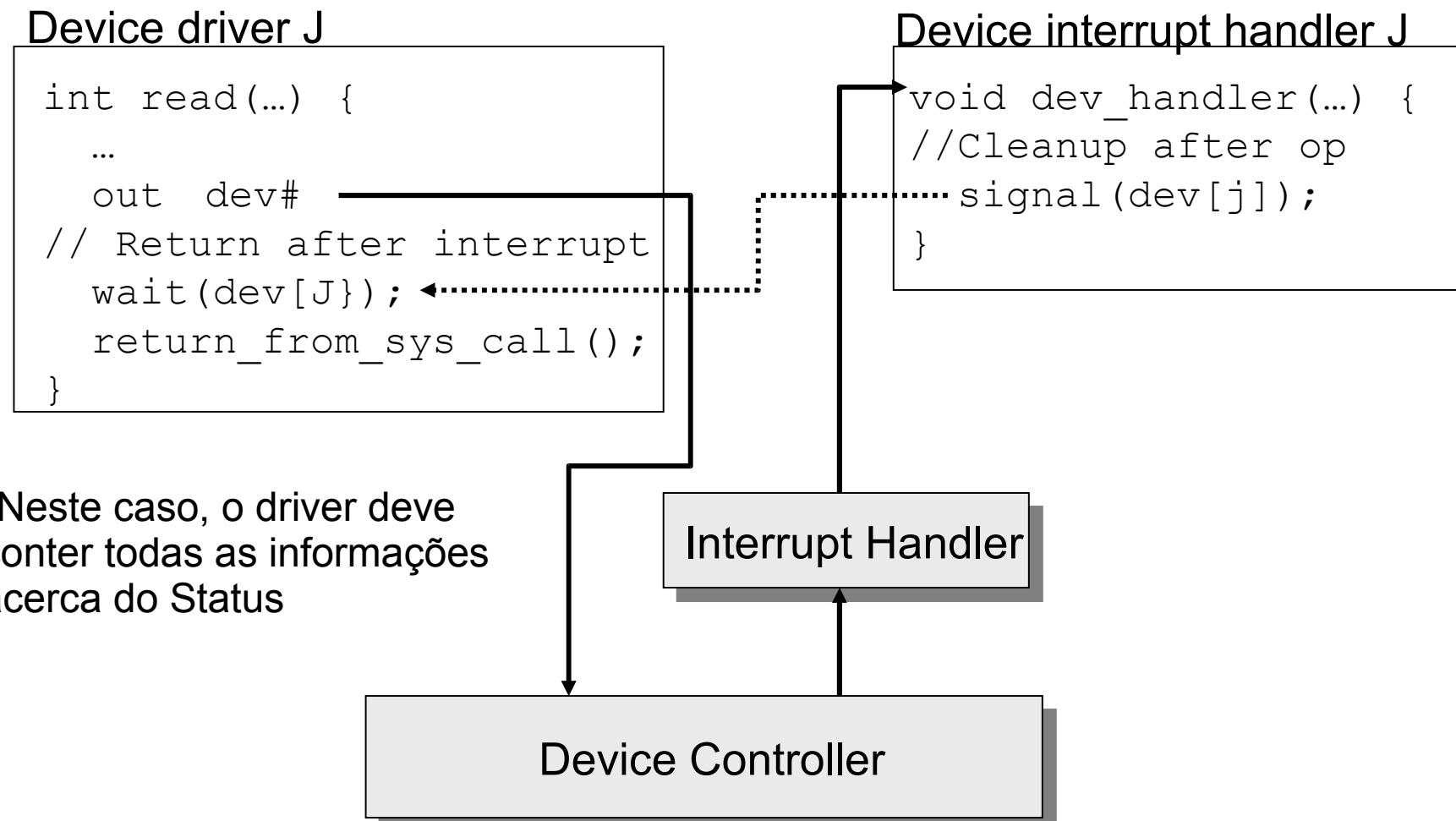
**Tratamento de Interrupções:** uso de uma tabela de status dos dispositivos



(Nutt, 2008)

## Definições: estratégias de E/S

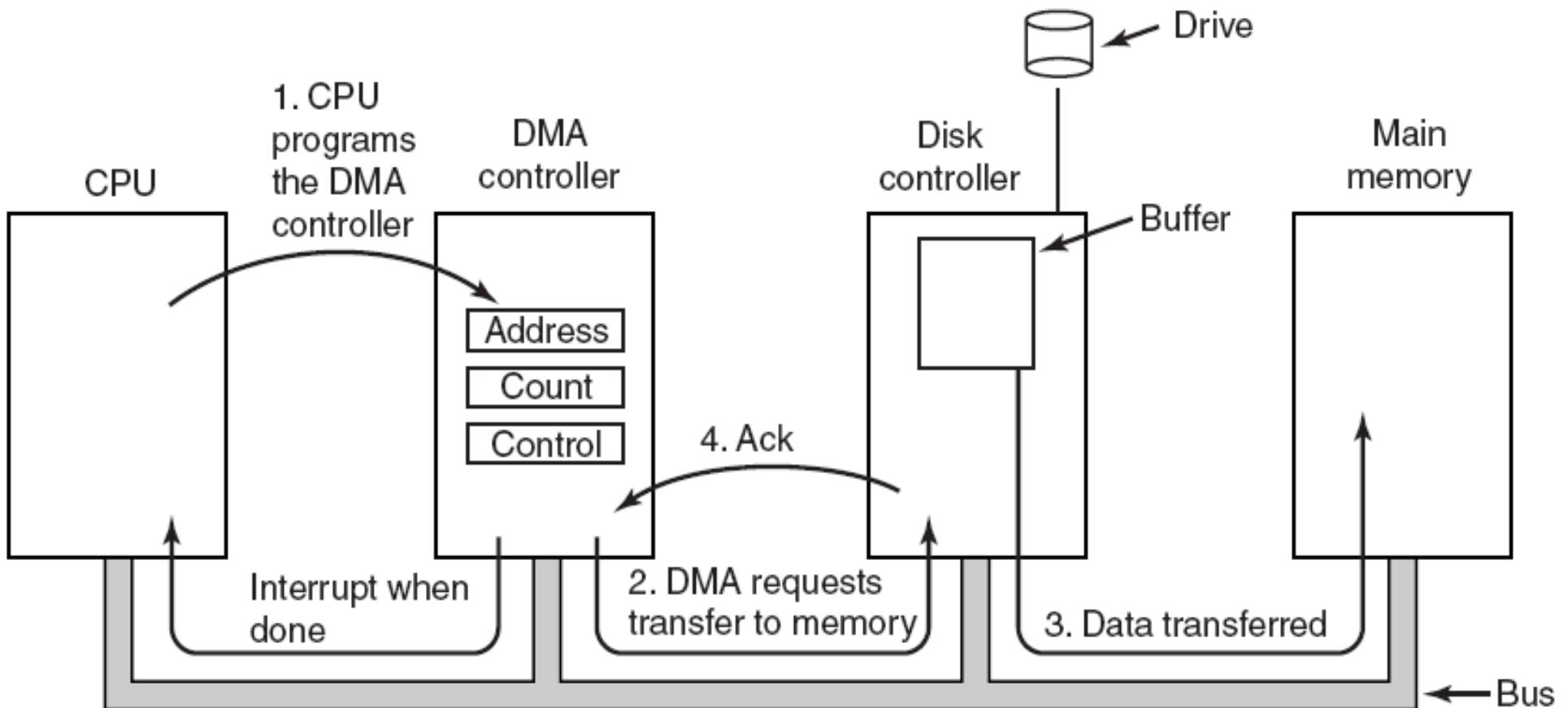
**Tratamento de Interrupções:** o driver é suspenso até o retorno do tratador



(Nutt, 2008)

## Definições: estratégias de E/S

### E/S com DMA



(Tanenbaum *et al.*, 2010)

## Definições: comunicação com o controlador

**Como a CPU se comunica com os registradores dos controladores?**

De acordo com Tanenbaum (Tanenbaum, 2008), há duas possibilidades:

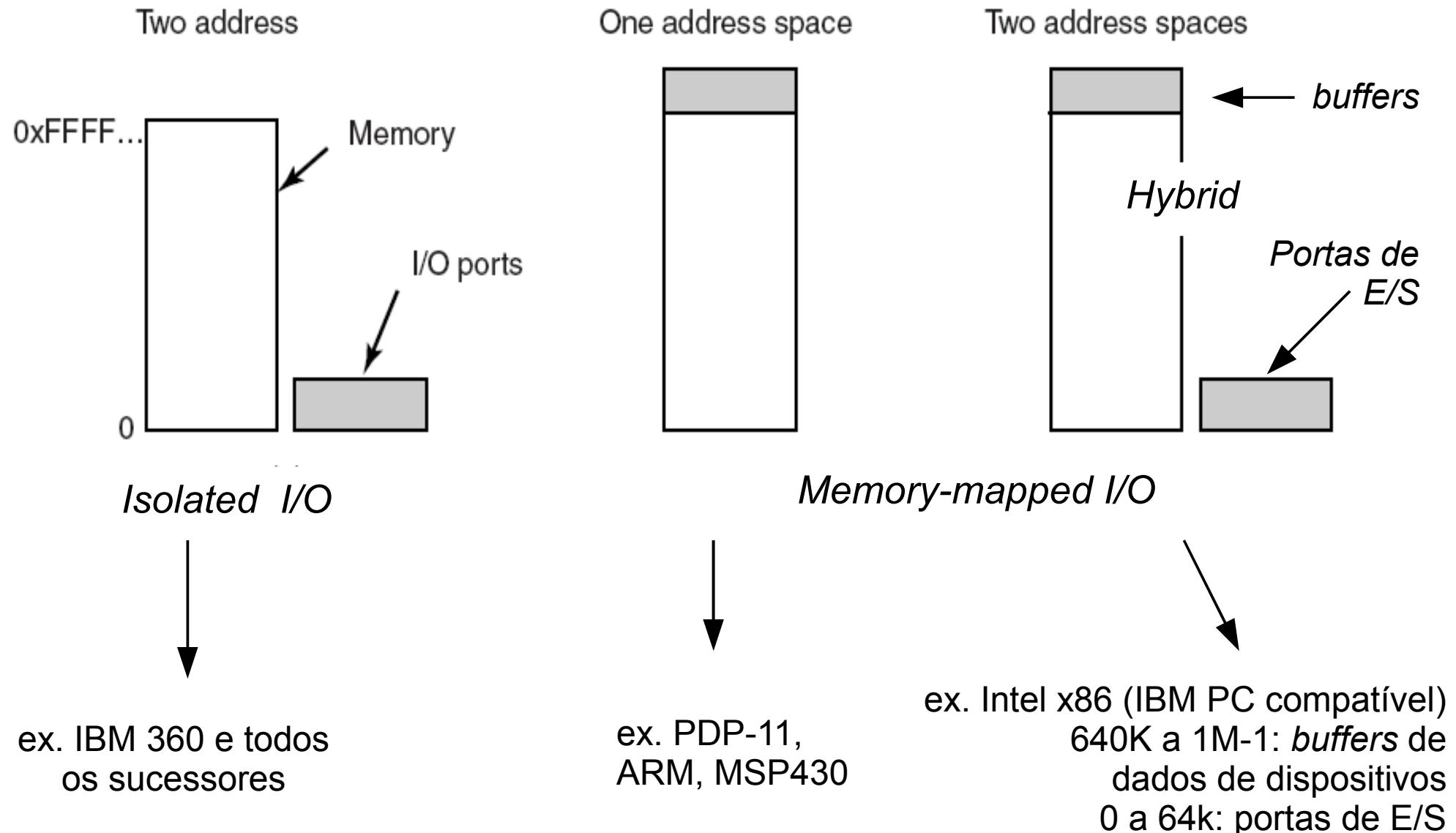
1) **E/S isolada ou *isolated I/O***. Dois espaços de endereçamento diferentes: um para a memória e outro para as portas de E/S.

2) **E/S mapeada em memória**. Apenas um espaço de endereçamento para memória e portas de E/S.

Uma variante: **Modelo híbrido**, mapeia no mesmo espaço de endereçamento a memória e os buffers dos controladores de periférico e em outro espaço as portas de E/S (outros registradores do controlador)

## Definições: comunicação com o controlador

(Tanenbaum, 2008)



## Definições: comunicação com o controlador

E/S Mapeada em Memória no MSP430 (total: 64KB)

Endereço	Descrição
0x0000 a 0x000F	SFR - Registradores de funções especiais (controle de interrupções e de ativação dos módulos internos)
0x0010 a 0x00FF	Registradores de controle de periféricos (acesso de 8 bits)
0x0100 a 0x01FF	Registradores de controle de periféricos (acesso de 16 bits)
0x0200 a 0x09FF	Memória RAM (até 2 Kbytes). Nos <i>chips</i> com mais de 2 Kbytes de RAM, essa área é espelhada nos endereços 0x1100 a 0x18FF
0x0A00 a 0x0BFF	Área não implementada
0x0C00 a 0x0FFF	ROM de <i>BOOT</i> ( <i>Bootstrap loader</i> )
0x1000 a 0x10FF	<i>FLASH</i> (256 bytes - <i>Information Memory</i> )
0x1100 a 0x38FF	Memória RAM (até 8 Kbytes) ou <i>FLASH</i> , dependendo do modelo do <i>chip</i>
0x3900 a 0xFFDF	Memória <i>FLASH</i>
0xFFE0 a 0xFFFFD	15 vetores de interrupção
0xFFFFE a 0xFFFFF	Vetor de <i>reset</i>

Von Neumann clássica com três barramentos:  
**Endereços:** 16bits  
→ 65536 endereços  
Pode endereçar bytes ou palavras de 16bits individualmente. Entretanto, em operações de 16bits é possível acessar apenas endereços pares.

**Dados:** 16bits

**Controle:** 8bits

Possui 16 registradores (R0-R15), todos de 16bits

Conjunto instruções de largura variável: 27 instruções físicas e 24 emuladas → RISC para alguns autores

## Definições: comunicação com o controlador

E/S Mapeada em Memória no ARM

```
DEV1 EQU 0x1000
LDR r1,#DEV1 ;
LDR r0,[r1] ;
LDR r0,#8 ;
STR r0,[r1] ;
```

Um pseudo-operador EQU para representar um nome simbólico a localização do dispositivo de E/S em memória

Configura o endereço do dispositivo

Lê do registrador do dispositivo

prepara o valor para escrita no DEV1

Escreve “8“ no dispositivo

Fonte: Wolf W. Computers as Components. 2a.  
Edição. Morgan Kaufmann, 2008.

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): exemplo teclado

```
#include <stdio.h>
#include <sys/io.h>

#define SCAN 0x60 →
#define STATUS 0x64 →

int main () {
    unsigned char s1, s2;
    if (ioperm(SCAN, 5, 1))
        {perror ("erro");
         exit(1);}

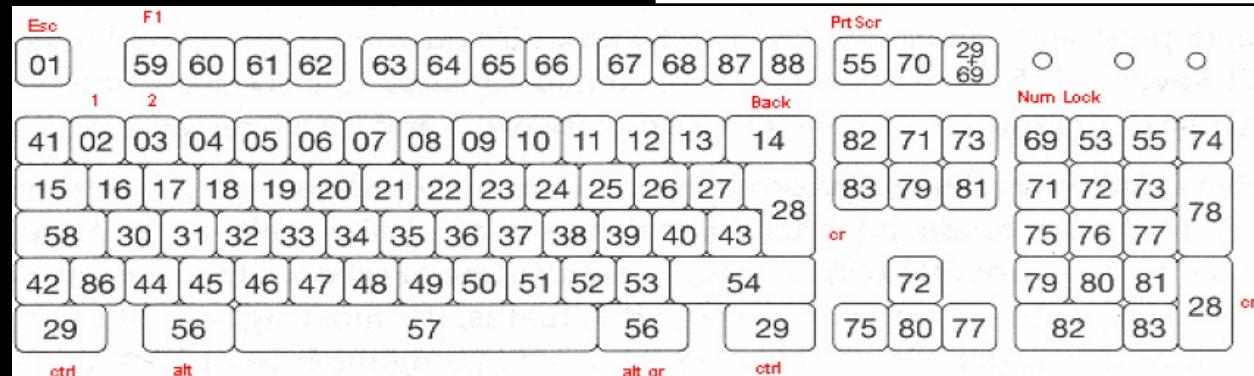
    for (;;) {
        s1 = inb(SCAN);
        s2 = inb(STATUS);
        if (s1 == 2) {
            printf("FIM\n\n");
            break; exit(0);}
        else{
            printf("Tecla Pressionada: %d \n", s1);
            printf("STATUS: %d \n", s2);
        }
    }
    return 0; }
```

**Registrador de Dados (SCancode): 8bits**

bit mais significativo = 1: *breakcode* (tecla liberada)  
bit mais significativo = 0: *make code* (tecla pressionada)

**Registrador de STATUS:**

bit 1 = 1: indica que um novo SCancode está disponível no *buffer* de saída  
bit 5 = 1: indica que o SCancode veio do teclado



# gcc -O teclado.c -o teclado

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): permissão

Apenas para endereços de E/S mapeados entre 0x0000 e 0x03FF

```
#include <sys/io.h> /*para glibc ou <unistd.h> para libc5*/  
  
if (ioperm(SCAN, 5, 1)) {perror ("erro"); exit(1);}
```

endereço de início

access = 1: oferece permissão  
access = 0: retira permissão

Limite de endereço: a partir de 0x60 até 0x64

Alternativa para endereçar portas de E/S de 0x0000 a 0xFFFF

```
#include <sys/io.h> /*para glibc ou <unistd.h> para libc5*/  
  
if (iopl(3)) {perror ("erro"); exit(1);}
```

nível normal para um processo é 0  
nível máximo é 3

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): endereços

Endereços de portas de E/S

```
root@darkstar:/proc# cat /proc/ioports
0000-001f : dma1
0020-0021 : pic1
0040-0043 : timer0
0050-0053 : timer1
0060-0060 : keyboard
0064-0064 : keyboard
0070-0071 : rtc0
0080-008f : dma page reg
00a0-00a1 : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : 0000:00:07.1
  0170-0177 : piix
01f0-01f7 : 0000:00:07.1
  01f0-01f7 : piix
02f8-02ff : serial
0320-0323 : wd7000
0350-0353 : wd7000
0376-0376 : 0000:00:07.1
  0376-0376 : piix
0378-037a : parport0
03c0-03df : vesafb
03f2-03f2 : floppy
03f4-03f5 : floppy
03f6-03f6 : 0000:00:07.1
  03f6-03f6 : piix
03f7-03f7 : floppy
03f8-03ff : serial
0cf0-0cf1 : pnp 00:01
0cf8-0cff : PCI conf1
```

Endereços de *buffers* de controladores

```
root@darkstar:~# cat /proc/iomem
00000000-0000ffff : reserved
00010000-0009f7ff : System RAM
0009f800-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000ca000-000cbfff : reserved
  000ca000-000cafff : Adapter ROM
000dc000-000e3fff : reserved
000e8000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-306effff : System RAM
  00100000-007a8436 : Kernel code
  007a8437-00a1551f : Kernel data
  00a94000-00b30673 : Kernel bss
306f0000-306fefff : ACPI Tables
306ff000-306fffff : ACPI Non-volatile Storage
30700000-307fffff : System RAM
40000000-40007fff : 0000:00:0f.0
40008000-4000bffff : 0000:00:10.0
d0000000-d7ffffff : 0000:00:0f.0
  d0000000-d7ffffff : vesafb
d8000000-d87fffff : 0000:00:0f.0
d8800000-d881ffff : 0000:00:10.0
  d8800000-d881ffff : mpt
d8820000-d883ffff : 0000:00:10.0
```

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): comandos

```
#include <asm/delay.h>
#include <sys/io.h> /*ou <asm/io.h>*/

int main (void)
{
    ioperm (0x378, 3, 1);

    for (;;) {
        outb (1, 0x378);
        usleep (100000);
        outb (2, 0x378);
        usleep (100000);
        outb (4, 0x378);
        usleep (100000);
        outb (8, 0x378);
        usleep (100000);
    }
    return 0;
}
```

Utilizando o registrador de dados do controlador da porta paralela (mapeado em memória) para o envio de dados

X	Porta de E/S (bits)
b	8
w	16
l	32

Em C:  
Int valor;  
valor = inX (0xPORTA\_DE\_E/S);  
outX (valor, 0xPORTA\_DE\_E/S);

Em assembler:  
IN PORTAES, %ah;  
OUT %ah, PORTAES

## Definições: comunicação com o controlador

### Como esses modelos funcionam?

ex. CPU quer ler dados de um dispositivo de E/S

- 1) Insere o endereço do registrador do dispositivo nas linhas de endereço do barramento.
- 2) Emite um sinal de leitura sobre uma linha de controle do barramento.
- 3) Um segundo canal (linha) é usado para informar se o espaço sendo requisitado é de E/S ou de memória. Se for de memória, ela responderá à requisição. Caso contrário, o dispositivo de E/S responderá à requisição

Obs. No modelo de E/S Mapeado em memória, cada módulo de memória e cada dispositivo de E/S comparam as linhas de endereços com a faixa de endereços associada a cada um e o responsável responde à requisição

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): vantagens

De acordo com Tanenbaum (Tanenbaum, 2008):

a) **Facilidade de acesso e rapidez**: podem ser acessados como variáveis, usando qualquer linguagem de programação. Caso contrário, o endereço do controlador do dispositivo deverá ser carregado no registrador do processador e testado, acrescentando mais uma linha de código.

```
Wait: TEST ADDR ;  
      JZ ready ;  
      JMP wait  
  
Ready:  
      #transfere dados
```

uma linha de  
código a mais

```
Wait: IN PORT, %AH ;  
      CMP %AH, 0 ;
```

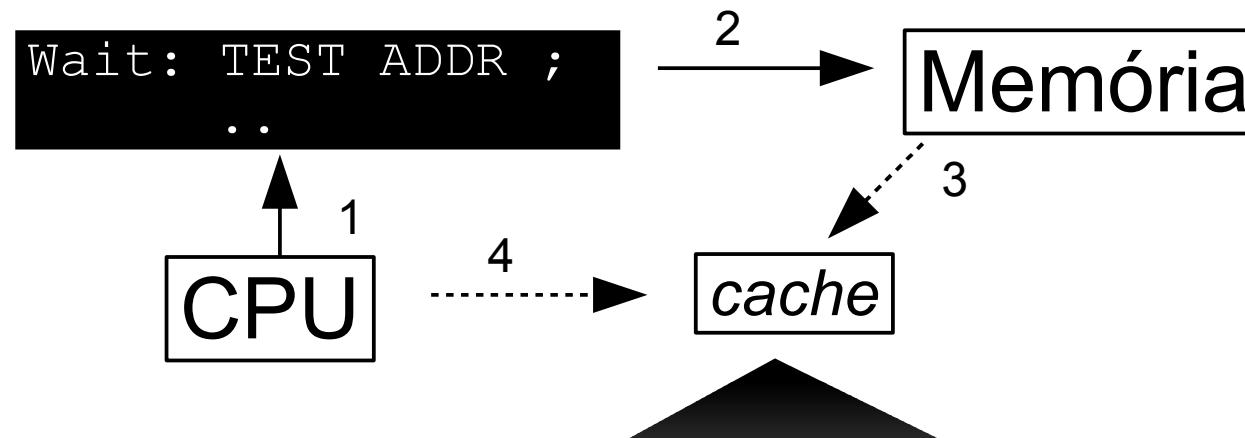
b) **Proteção**: além da permissão para uso, concedida apenas ao superusuário, o SO não mapeia a área de registradores de E/S na memória virtual do processo. Ou seja, processo filhos não herdarão as permissões concedidas aos pais.

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): desvantagens

De acordo com Tanenbaum (Tanenbaum, 2008):

a) **Caching**: o uso de cache pode colocar a CPU num ciclo infinito.



A CPU não consegue saber se o dispositivo está pronto, porque não é testado o registrador de controle do dispositivo.

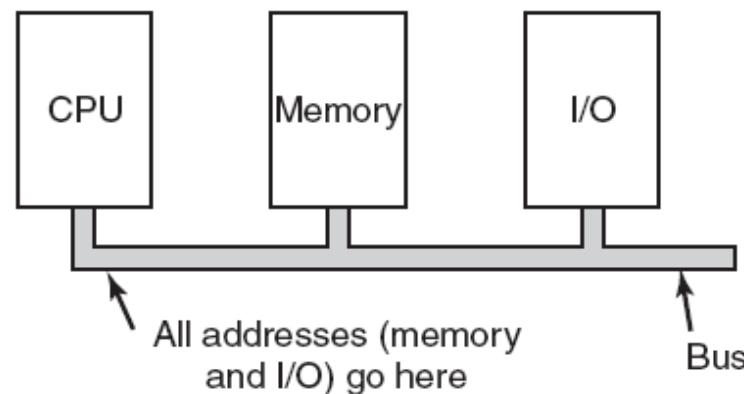
**Solução:** o hardware deverá ser capaz de desligar o caching para as páginas onde estão mapeados os registradores dos controladores

**Consequência:** SO mais complexo

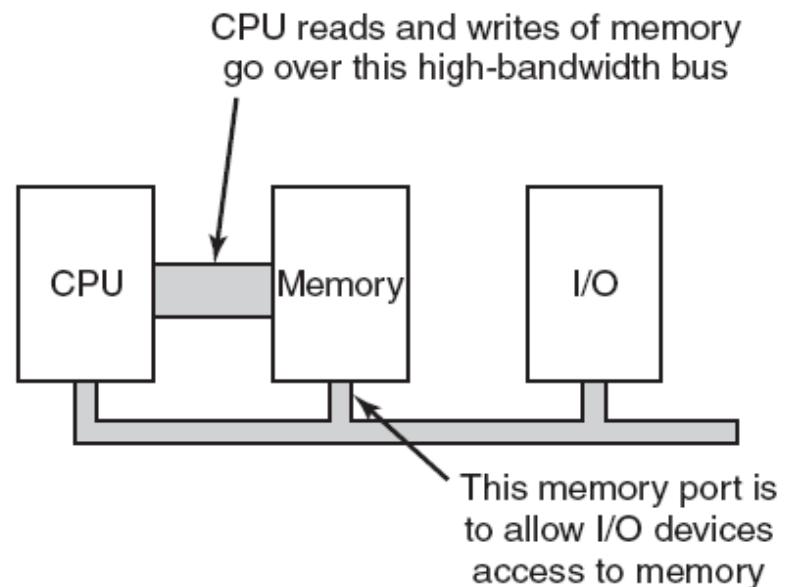
## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): desvantagens

b) **Dificuldade em sistemas multibarramentos:** dispositivos não conseguem “ver” os endereços colocados no barramento de sistema (entre a memória e a CPU)



*A single-bus architecture*



*A dual-bus memory architecture*

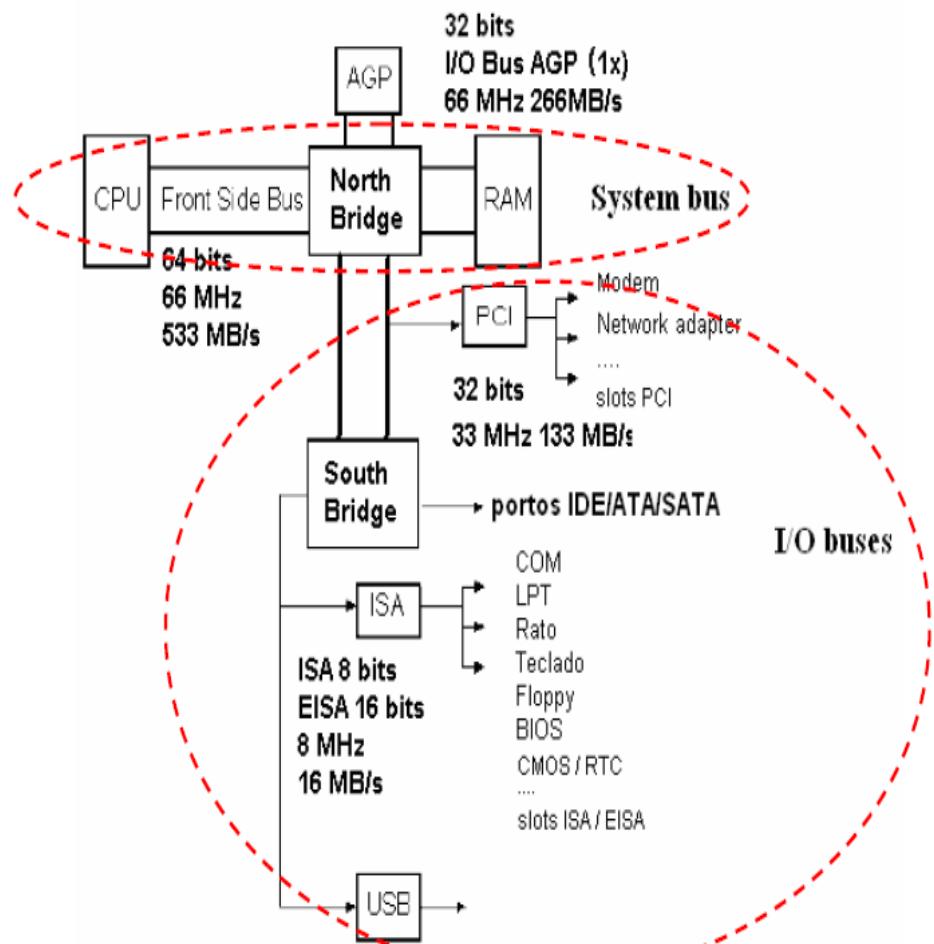
**Solução1:** testar o mesmo endereço por todos os barramentos, até que algum dispositivo ou memória respondam

**Consequência:** maior complexidade do hardware

## Definições: comunicação com o controlador

E/S mapeada em memória no Linux (Intel x86): desvantagens

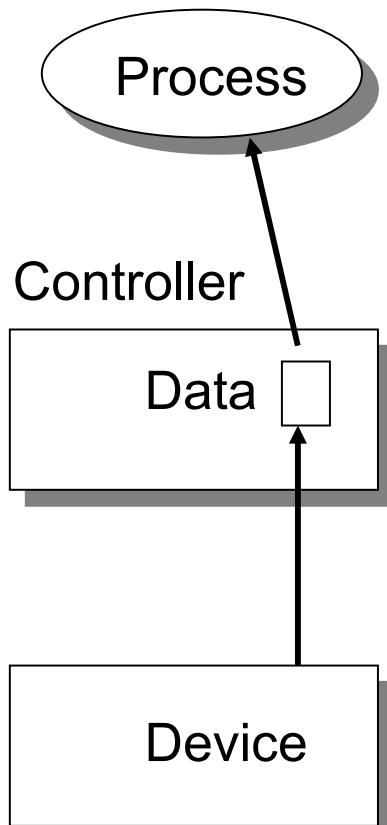
### b) Dificuldade em sistemas multibarramentos:



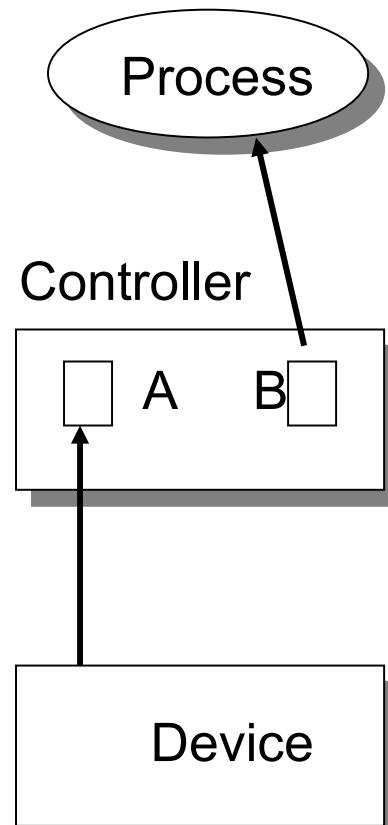
**Solução2:** filtrar os endereços no *chipset* (adotada em sistemas x86)

**Consequência:** na inicialização do PC é necessário determinar quais endereços correspondem ao mapeamento dos controladores e quais se referem a endereços de memória principal.

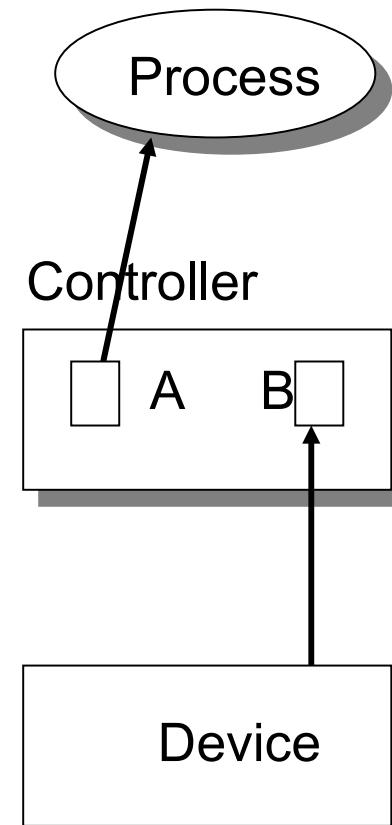
## Definições: *Buffering*



Unbuffered



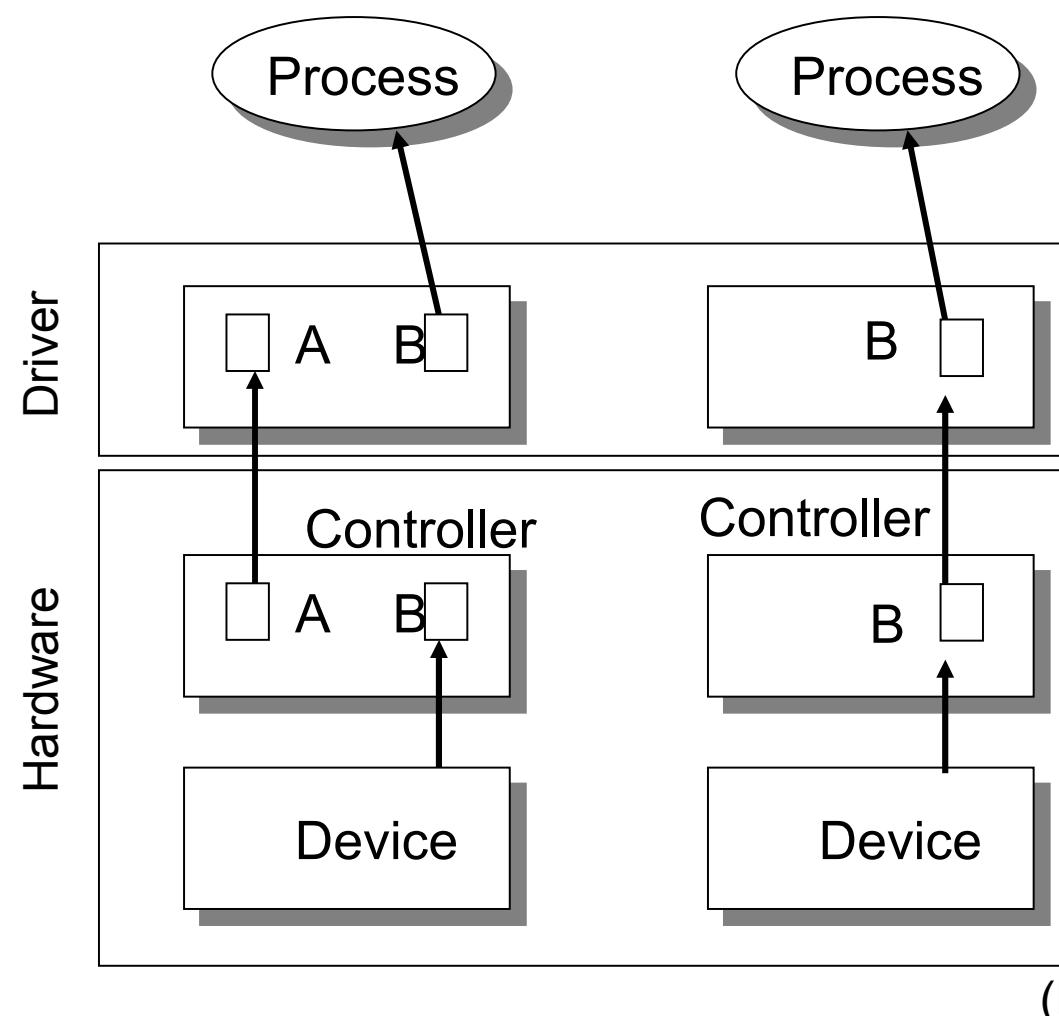
Process reads  $b_{i-1}$   
Controller reads  $b_i$



Process reads  $b_i$   
Controller reads  $b_{i+1}$

## Definições: estratégias de E/S

**Buffering:** aumentar o desempenho, diminuindo o tempo para executar as operações de E/S



- *Input buffering* x *Double Buffering?*
- Em Hardware e/ou em Software?
- Aumentar a quantidade de buffers?
- Como organizá-los?
  - Alternativa: *circular buffering*

## Definições: classificações

De acordo com Tanenbaum (Tanenbaum, 2008, p.217), em geral, são usadas duas classificações:

- 1) **Dispositivos de Bloco**, contêm vários blocos de dados que podem ser endereçados independentemente.
- 2) **Dispositivos de Caractere**, os quais geram ou aceitam uma sequência de caracteres.

\*A maioria dos SO define uma interface padrão para os drivers de bloco e para os drivers de caractere. **Essas interfaces constituem de um conjunto de chamadas.**

```
talles@ubuntu:~$ ls -l /dev
total 0
crw-rw----+ 1 root audio      14,  12 2010-12-24 17:43 adsp
crw-----  1 root video      10, 175 2010-12-24 17:43 agpgart
crw-rw----+ 1 root audio      14,   4 2010-12-24 17:43 audio
drwxr-xr-x  2 root root      640 2010-12-24 17:42 block
drwxr-xr-x  2 root root      80  2010-12-24 17:42 bsg
drwxr-xr-x  3 root root      60  2010-12-24 17:42 bus
lrwxrwxrwx  1 root root      3  2010-12-24 17:43 cdrom1 -> sr0
lrwxrwxrwx  1 root root      3  2010-12-24 17:43 cdrw1 -> sr0
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos:

- Dispositivos de Caracteres (*char dev*)
  - Acessados em */dev/disp*
  - Conjunto de bytes, chamado de STREAM
  - Em geral, apenas acesso sequencial. Ao contrário dos demais arquivos de dados.
  - Em geral, implementam pelo menos as seguintes operações: *open ()*, *close ()*, *read ()*, *write ()*
  - Ex. */dev/tty*, */dev/lp0*, ...

```
root@darkstar:/proc# cat /proc/devices
Character devices:
 1 mem
 2 pty
 3 ttyp
 4 /dev/uc/0
 4 tty
 4 ttys
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 6 lp
 7 vcs
 9 st
10 misc
13 input
14 sound
21 sg
189 usb_device
246 rtc
247 hidraw
248 usb_endpoint
249 usbmon
250 gdth
251 megaraid_sas_ioctl
252 megadev_legacy
253 aac
254 bsg
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos de Bloco (*block dev*)
  - Pode armazenar um sistema de arquivos, da mesma forma que um disco.
  - Originalmente, para transferências de blocos de pelo menos 512 bytes. Entretanto, no Linux é possível o acesso sequencial (por STREAM) a esses dispositivos
  - A diferença em relação aos char dev está na organização interna dos dados e na interface de programação
  - Ex. /dev/sda... hda

```
Block devices:  
 1 ramdisk  
 2 fd  
 3 ide0  
259 blkext  
 7 loop  
 8 sd  
 9 md  
11 sr  
22 ide1  
65 sd  
66 sd  
67 sd  
68 sd  
69 sd  
70 sd  
71 sd  
80 i2o_block  
128 sd  
129 sd  
130 sd  
131 sd  
132 sd  
133 sd  
134 sd  
135 sd  
253 device-mapper  
254 mdp
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos de Rede
  - Não são mapeados em descritores como “/dev/...” pelo sistema de arquivos, mas, identificados por um nome único
    - Ex ETH0, Wlan0,...
  - A comunicação entre o kernel e o dispositivo é baseada na troca de pacotes (formados por múltiplos bytes e cabeçalhos de rede), ao invés da API convencional

```
talles@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e8:0b:cc
          inet  addr:192.168.40.130  Bcast:192.168.40.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fee8:bcc/64 Scope:Link
                    UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                    RX packets:117  errors:0  dropped:0  overruns:0  frame:0
                    TX packets:40  errors:0  dropped:0  overruns:0  carrier:0
                    collisions:0  txqueuelen:1000
                    RX bytes:24915 (24.9 KB)  TX bytes:5274 (5.2 KB)
                    Interrupt:19  Base address:0x2000
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos de Barramento: PCI

Número de dispositivo (5 bits)

Número de função (3 bits)

\*Cada periférico é identificado por um interiro de 32 bits

Número de Barramento (8 bits)

Domínio (16 bits): 0-ffff  
Omitido neste exemplo

```
root@darkstar:~# lspci
00:00.0 RAM memory: nVidia Corporation MCP67 Memory Controller (rev a2)
00:01.0 ISA bridge: nVidia Corporation MCP67 ISA Bridge (rev a2)
00:01.1 SMBus: nVidia Corporation MCP67 SMBus (rev a2)
00:01.2 RAM memory: nVidia Corporation MCP67 Memory Controller (rev a2)
00:01.3 Co-processor: nVidia Corporation MCP67 Co-processor (rev a2)
00:02.0 USB Controller: nVidia Corporation MCP67 OHCI USB 1.1 Controller (rev a2)
00:02.1 USB Controller: nVidia Corporation MCP67 EHCI USB 2.0 Controller (rev a2)
00:04.0 USB Controller: nVidia Corporation MCP67 OHCI USB 1.1 Controller (rev a2)
00:04.1 USB Controller: nVidia Corporation MCP67 EHCI USB 2.0 Controller (rev a2)
00:06.0 IDE interface: nVidia Corporation MCP67 IDE Controller (rev a1)
00:07.0 Audio device: nVidia Corporation MCP67 High Definition Audio (rev a1)
00:08.0 PCI bridge: nVidia Corporation MCP67 PCI Bridge (rev a2)
00:09.0 IDE interface: nVidia Corporation MCP67 AHCI Controller (rev a2)
00:0a.0 Ethernet controller: nVidia Corporation MCP67 Ethernet (rev a2)
00:0c.0 PCI bridge: nVidia Corporation MCP67 PCI Express Bridge (rev a2)
00:0d.0 PCI bridge: nVidia Corporation MCP67 PCI Express Bridge (rev a2)
00:12.0 VGA compatible controller: nVidia Corporation C67 [GeForce 7150M / nForce 630M] (rev a2)
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos de Barramento: PCI e USB

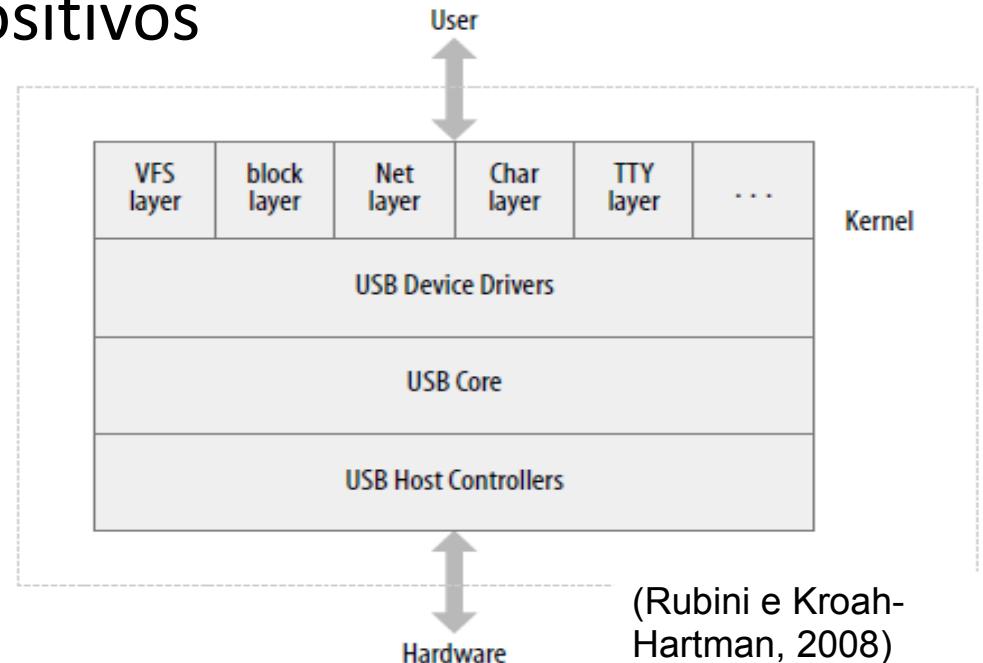
```
root@darkstar:~# ls /sys/bus/pci/drivers
3w-9xxx/          b43-pci-bridge/   k8temp/          pata_amd/          pata_jmicron/      pata_sil680/
3w-sas/           bfa/           lpfc/           megaraid/        pata_artop/        pata_marvell/
3w-xxxx/          cciss/          megaraid_legacy/  megaraid_sas/    pata_atiixp/      pata_mpiix/
DAC960/           cs5535/          mpt2sas/          mptfc/           pata_atp867x/      pata_netcell/
HDA\ Intel/       dc395x/          mptsas/          mptspi/          pata_cnd640/      pata_ninja32/
PCI_I2O/          dm3191d/         mvsas/           nForce2_smbus/  pata_cnd64x/      pata_ns87410/
PMC\ MaxRAID/    ehci_hcd/       nvidia/          ohci1394/        pata_cs5520/      pata_ns87415/
aacraid/          fnic/           ohci_hcd/        parport_pc/     pata_cs5530/      pata_oldpiix/
advansys/         forcedeth/      ohci_hcd/        pata_acpi/      pata_cs5536/      pata_opti/
ahci/             gdth/           ohci_hcd/        pata_ali/       pata_cypress/    pata_optidma/
aic79xx/          hpsa/           ohci_hcd/        pata_efar/      pata_hpt366/      pata_pdc2027x/
aic7xxx/          hptiop/          ohci_hcd/        ohci1394/       pata_hpt37x/      pata_pdc202xx_old/
aic94xx/          inia100/         ohci_hcd/        parport_pc/    pata_hpt3x2n/    pata_radisys/
arcmsr/           initio/          ohci_hcd/        pata_acpi/     pata_hpt3x3/      pata_rdc/
ata_generic/     ioapic/          ohci_hcd/        pata_ali/       pata_it8213/      pata_rz1000/
ata_piix/         ipr/            ohci_hcd/        pata_acpi/     pata_it821x/      pata_sch/
atp870u/          ispl760/         ohci_hcd/        pata_ali/       pata_serverworks/ pata_serverworks/
root@darkstar:~# ls /sys/bus/usb/drivers
hiddev/          ums-alauda/     ums-freecom/     ums-karma/      ums-sddr55/      usb-storage/    usbserial/
hub/             ums-cypress/    ums-isd200/     ums-onetouch/   ums-sddr55/      usbfs/          usbserial_generic/
pl2303/          ums-datafab/   ums-jumpshot/   ums-sddr09/     usb/           usbhid/         uvcvideo/
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

#### USB

- *USB Serial Port (char dev)*
- *USB Pen Drive (block dev)*
- *USB Ethernet Interface (net dev)*



(Rubini e Kroah-Hartman, 2008)

```
root@darkstar:~# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 004: ID 0781:5567 SanDisk Corp.           ← Pen drive
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 04f2:b015 Chicony Electronics Co., Ltd VGA 24fps UVC Webcam
Bus 003 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 003 Device 002: ID 04f3:0216 Elan Microelectronics Corp.
Bus 003 Device 003: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port ← Conversor
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Conversor  
USB/Serial

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Conversor USB/Serial

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
    unsigned char teste=1;
    FILE *desc_arq;
    size_t testeread;
    char bufferteste [1000];

    desc_arq = fopen("/dev/ttyUSB0", "w+");
    if (desc_arq < 0) {exit(-1);}

    fwrite (&teste, 1, 1, desc_arq);

    rewind(desc_arq);
    testeread = fread (bufferteste, sizeof(bufferteste), 1, desc_arq);
    printf (">>>>: %s", bufferteste);
    fclose(desc_arq);

    return 0;
}
```

Envia a String “1“ para /dev/ttyUSB0

## Estudo de casos: *Device Drivers* no Linux

### Identificadores de dispositivos

- **Número Maior (Major Device Number)**: identifica o *driver* em uso
- **Número Menor (Minor Device Number)**: identifica o dispositivo ou o componente controlado pelo *driver*

```
root@darkstar:~# ls -l /dev/sda1
brw-rw---- 1 root disk 8, 1 2010-12-02 09:14 /dev/sda1
root@darkstar:~# _
```

### Criação/remoção de novos identificadores de dispositivos

```
root@darkstar:~# mknod /dev/aulaC c 12 2
root@darkstar:~# ls -l /dev/aulaC
crw-r--r-- 1 root root 12, 2 2010-12-02 09:19 /dev/aulaC
```

```
root@darkstar:~# rm /dev/aulaC
root@darkstar:~# ls -l /dev/aulaC
/bin/ls: cannot access /dev/aulaC: No such file or directory
root@darkstar:~# _
```

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos Especiais

```
root@darkstar:~# ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 2010-12-15 09:02 /dev/null
```

- Possui número maior = 1 e está associado ao *Linux Kernel Memory Device* ao invés de um *device driver*
- Na prática serve para descarte de dados da saída padrão de um programa ou quando a leitura de um descriptor de arquivo exige um *end-of-file* ou *return 0*.

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos Especiais

```
root@darkstar:~# ls -l /dev/zero
crw-rw-rw- 1 root root 1, 5 2010-12-15 09:02 /dev/zero
root@darkstar:~# ls -l /dev/full
crw-rw-rw- 1 root root 1, 7 2010-12-15 09:02 /dev/full
```

- */dev/zero*: um arquivo infinito preencido com zeros
- */dev/full*: para tratamento de estouro de buffer. Por exemplo, uma escrita para */dev/full* falha e indica *errno= ENOSPC*, que comumente significa que o(s) buffer(s) do dispositivo está cheio quando se tenta uma operação de escrita

```
root@darkstar:~# od -t x1 /dev/zero
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
```

\* “-t x1“ para imprimir o conteúdo em hexadecimal

## Estudo de casos: *Device Drivers* no Linux

### Principais Classes de Dispositivos

- Dispositivos Especiais

```
root@darkstar:~# ls -l /dev/random
crw-rw-rw- 1 root root 1, 8 2010-12-15 09:02 /dev/random
```

- Provê acesso ao serviço de geração de números aleatórios do kernel
- Aspécito interessante: o linux utiliza o intervalo médio entre ações de entrada provocadas pelo usuário (como os movimento de teclado e mouse) para a geração de uma stream de números aleatórios. Ao contrário da função rand () da biblioteca padrão que utiliza a mesma semente para a geração de número aleatórios.

```
root@darkstar:~# od -t x1 /dev/random
00000000 82 f4 c5 09 16 8c 1a e7 23 77 08 b4 88 08 e3 dd
00000020 7b 4e 18 6a 82 9c 2b f3 ae 5b 28 3e af 11 f9 61
00000040 6f 43 d0 b8 c7 75 d0 54 0e ac 5d f4 ad e1 ee 4d
00000060 df f5 16 2c 21 2e 56 27 54 3b cd 8a f9 31 5f 85
0000100  db 37 d4 6a 31 1f f4 cd d3 e0 c4 36 3e 3e 73 2a
0000120 6d 2b 54 70 92 56 dc b8 b6 74 45 9a 86 50 d9 95
0000140 a6 e5 90 17 34 1c 64 24 2d 81 d5 8d 97 84 c3 db
0000160 5c 17 77 48 8f f2 cf 36 d8 a5 e9 74 8b 4f 6b 07
0000200 21 ab 58 a5 29 7e d7 9a d4 88 71 b8 83 d7 3e 23
0000220 77 2b 78 ff d7 06 f1 e8 a0 34 21 bc 25 4b 36 59
0000240 e3 6c 31 91 c3 5c d9 86 9f 7f 10 9c aa a2 b9 64
0000260 bc 88 fb fa 37 ca 30 6c 3d 5b 7a d9 6b 9e f9 4c
0000300 99 dd 95 74 8c ad 24 d2 84 91 8f 08 1d b0 3f bf
0000320 20 c4 b7 38 2c 41 19 bc a2 8e d2 4b 6d ff ea 53
0000340 1b 45 14 b7 b4 ed b1 c9 00 ac 25 74 b8 82 eb b6
0000360 d6 79 3b 57 f8 5f 37 0e 99 f1 0a 50 5b c8 e9 07
```

## Estudo de casos: *Device Drivers* no Linux

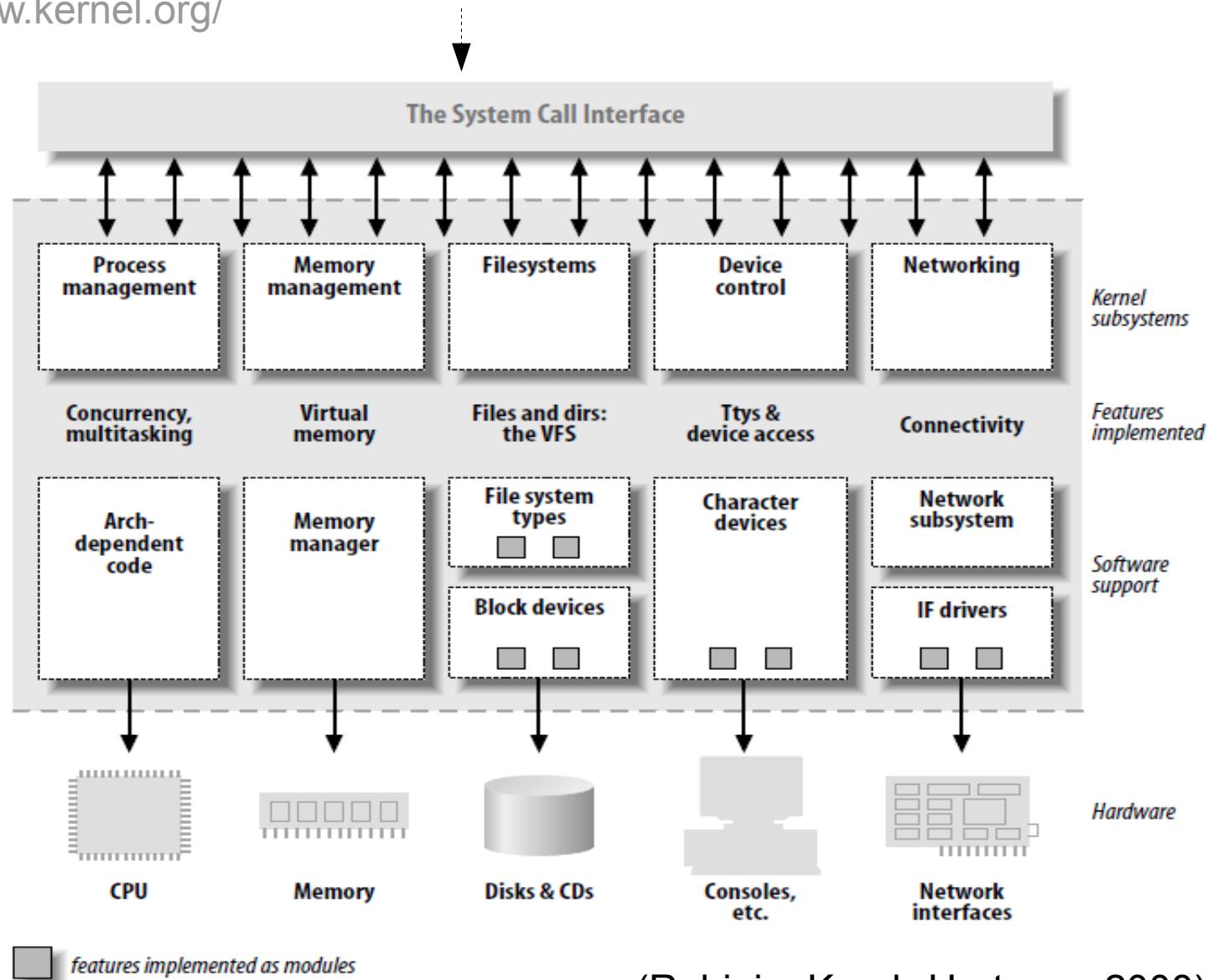
Kernel:

<http://www.kernel.org/>

Abstração

+

Execução



(Rubini e Kroah-Hartman, 2008)

# Estudo de casos: *Device Drivers* no Linux

Kernel: Abstração + Execução



## 1) Arquitetura de Software monolítica

“O sistema operacional inteiro é executado como um único programa em modo supervisor” (Tanenbaum, 2010)

Na prática, uma imagem do sistema é gerada em tempo de compilação e carregada para a memória no momento da inicialização da máquina.

Ex. Em geral, /boot/vmlinuz

## 2) Suporte à modificações em tempo de execução por meio dos MÓDULOS do Kernel

“Um módulo é uma unidade independente de software que pode ser projetada, implementada e incluída (instalada) em tempo de execução”

## Estudo de casos: *Device Drivers* no Linux

Kernel:

Abstração

+

Execução

<http://www.kernel.org/>

Left	File	Command	Options	Right
< /boot				Name
..				
README.initrd				
System.map				
System.map-generic-2.6.29.6				
System.map-generic-smp-2.6.29.6-smp				
System.map-huge-2.6.29.6				
System.map-huge-smp-2.6.29.6-smp				
boot.0800				
boot_message.txt				
config				
config-generic-2.6.29.6				168   Nov 26 23:05
config-generic-smp-2.6.29.6-smp				28   Nov 26 22:37
config-huge-2.6.29.6				96042   Aug 17 2009
config-huge-smp-2.6.29.6-smp				96218   Aug 17 2009
diag1.img				96158   Aug 17 2009
map				96246   Aug 17 2009
slack.bmp				5040   Nov 29 2008
umlinuz				79872   Dec 1 12:16
				15754   Feb 21 2008
				29   Nov 26 22:37
umlinuz-generic-2.6.29.6				2386000   Aug 17 2009
umlinuz-generic-smp-2.6.29.6-smp				2495952   Aug 17 2009
umlinuz-huge-2.6.29.6				4768048   Aug 17 2009
umlinuz-huge-smp-2.6.29.6-smp				4940304   Aug 17 2009

## Estudo de casos: *Device Drivers* no Linux

Kernel: Execução + Abstração



### 1) CPU no modo supervisor

- Teoricamente, o acesso à memória, instruções privilegiadas,... permitido!
- Na prática, o linux altera o modo de operação quando a aplicação executar uma chamada ao sistema ou a aplicação for suspensa por uma interrupção de hardware (mais de dois modos na família x86)

### 2) *Multiprogramming Kernel* a partir da versão 2.6: kernel preemptivo

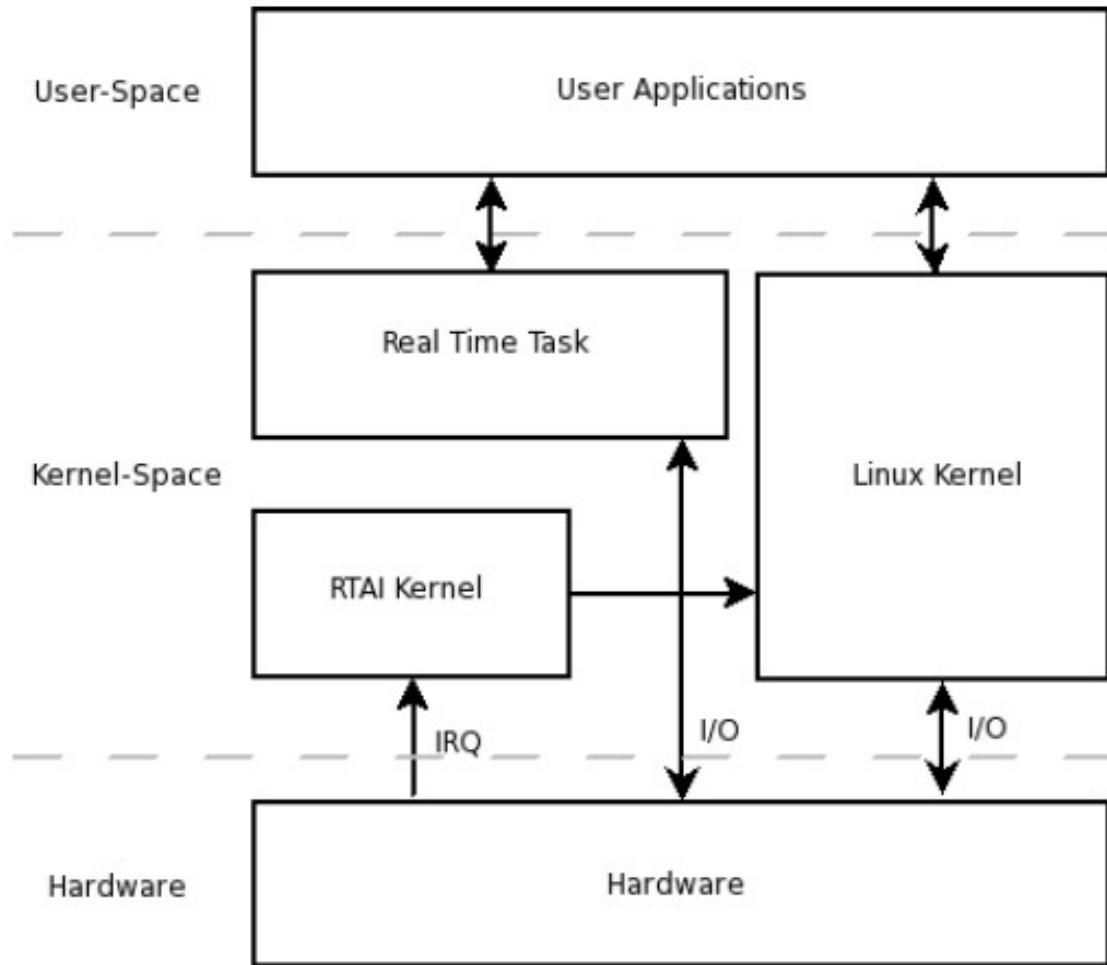
- Drivers concorrentes e reentrantes: um driver deve suportar que ele pode ser chamado uma segunda vez antes que a primeira chamada tenha sido concluída
- Entretanto, certas áreas de código não podem ser interrompidas quando em execução
- Módulos podem implementar drivers de dispositivos, mas, também sistemas de arquivos, protocolos de rede, ferramentas...

Originalmente, *single-threaded kernel*, ou seja, apenas uma única thread era executada por vez. Além disso, uma thread não podia executar uma chamada ao sistema e ser interrompida pelo escalonador (preempção)

## Estudo de casos: *Device Drivers* no Linux

### Ex. RTAI - the RealTime Application Interface for Linux

<https://www.rtai.org/>



Uma vez instalado o RTAI o escalonamento de processo deixa de ser responsabilidade do kernel e passa a ser do RTAI.

Disponibiliza comandos para gerenciamento de tarefas de tempo real, que não são interrompidas pelo só

## Estudo de casos: *Device Drivers* no Linux

### Módulos do Kernel

Module	Size	Used by
lp	9316	0
fuse	54008	1
ppdev	7200	0
psmouse	41676	0
serio_raw	5024	0
evdev	9152	2
snd_ens1371	20608	0
gameport	9960	1 snd_ens1371
snd_rawmidi	19040	1 snd_ens1371
snd_seq_device	6088	4 snd_seq_dummy, snd_seq_oss, snd_seq, snd_rawmidi
snd_ac97_codec	100128	1 snd_ens1371
ac97_bus	1372	1 snd_ac97_codec
parport_pc	24036	1
snd_pcm	68128	3 snd_pcm_oss, snd_ens1371, snd_ac97_codec
parport	30700	3 lp, ppdev, parport_pc

tamanho em bytes

Contador de utilização.  
Utilizado por outros  
módulos?

#cat /dev/module?

Identifica o nome. Normalmente, o nome  
criado em /dev/dispositivo.

Dependências:

Partport

parport\_pc  
Lp  
ppdev

## Estudo de casos: *Device Drivers* no Linux

### Módulos do Kernel

```
talles@ubuntu:~$ modinfo parport
filename:      /lib/modules/2.6.32-26-generic/kernel/drivers/parport/parport.ko
license:       GPL
srcversion:    31954FF5B7203B722EFEA39
depends:
vermagic:     2.6.32-26-generic SMP mod_unload modversions 586
talles@ubuntu:~$
```

```
talles@ubuntu:~$ modinfo lp
filename:      /lib/modules/2.6.32-26-generic/kernel/drivers/char/lp.ko
license:       GPL
alias:        char-major-6-*
srcversion:   84EA21D13BD2C67171AC994
depends:      parport
vermagic:    2.6.32-26-generic SMP mod_unload modversions 586
parm:         parport:array of charp
parm:         reset:bool
talles@ubuntu:~$
```

## Estudo de casos: *Device Drivers* no Linux

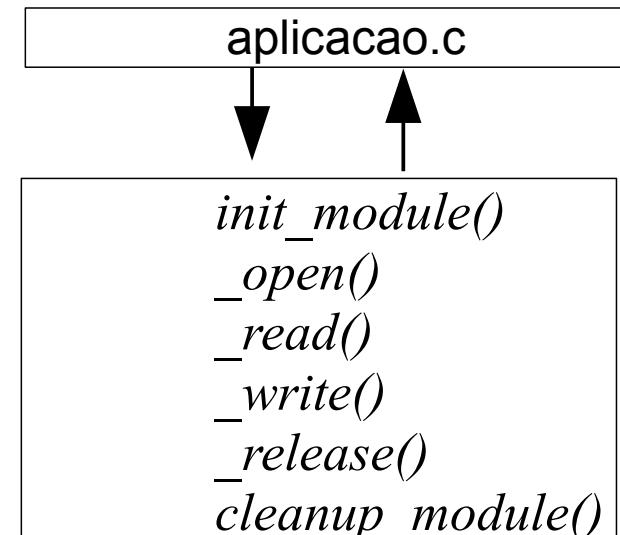
### Exemplo 1: um "hello world"

#### Objetivos:

- Implementar um *device driver* do tipo *character device* (como um módulo do kernel) que deve contabilizar quantas vezes foi invocado.
- Escrever o Makefile para a compilação do módulo.
- Implementar um programa de usuário para acessar o *device driver*.

#### Estrutura elementar de um módulo

- 1) Inclusão dos cabeçalhos
- 2) Definições e declarações globais
- 3) Inicialização
- 4) Demais funcionalidades
- 5) Encerramento



chardev.c

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

*chardev.c*

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>

MODULE_LICENSE ("GPL");
MODULE_AUTHOR ("prof. Talles");
MODULE_DESCRIPTION ("Exercício de CMP1240");

int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *);

#define DEVICE_NAME "chardev"
#define BUF_LEN 50

static int Major;
static int Device_Open = 0; /* para verificar se o dispositivo está aberto
                           sendo usado por outro processo */
static char msg[BUF_LEN];
static char *msg_Ptr;
```

→ **# modinfo chardev.ko**

→ **O nome do driver.**  
# cat /proc/devices

→ **Tamanho máximo da msg enviadas para a aplicação**

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

*chardev.c*

```
static struct file_operations fops = {  
    .read = device_read,  
    .write = device_write,  
    .open = device_open,  
    .release = device_release  
};  
  
int init_module(void) { Major = register_chrdev(0, DEVICE_NAME, &fops);  
  
if (Major < 0) {printk(KERN_ALERT "Erro no registro %d\n", Major); return Major;}  
  
printk(KERN_ALERT "Device driver registrado com sucesso!\n");  
printk(KERN_ALERT "Identificador do dispositivo: %d \n", Major);  
printk(KERN_ALERT "Para criar link: mknod /dev/%s c %d 0\n", DEVICE_NAME, Major);  
printk(KERN_ALERT "Remova o driver quando acabar: rmmod chardev\n");  
printk(KERN_ALERT "Remova o fichario /dev quando acabar: rm /dev/chardev\n");  
printk(KERN_INFO "Exercicio CMP1240\n");  
return SUCCESS; }
```

Registro do Dispositivo:  
Num maior, nome e pont. para a *struct*

<linux/kernel.h>

```
#define KERN_EMERG    "<0>"    /* System is unusable */  
#define KERN_ALERT    "<1>"    /* action must be taken immediately */  
#define KERN_CRIT    "<2>"    /* critical conditions */  
#define KERN_ERR     "<3>"    /* error conditions */  
#define KERN_WARNING  "<4>"    /* warning conditions */  
#define KERN_NOTICE   "<5>"    /* normal but significant condition */  
#define KERN_INFO    "<6>"    /* informational */  
#define KERN_DEBUG   "<7>"    /* debug-level messages */
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

*chardev.c*

```
void cleanup_module(void)
/* executada quando o módulo é descarregado # rmmod chardev */
{unregister_chrdev(Major, DEVICE_NAME);
 printk (KERN_ALERT "Device driver removido com sucesso!\n");}

static int device_open(struct inode *inode, struct file *file)
/* executada sempre que o dispositivo é acessado static impede que ela
seja visível fora do arquivo onde foi definida */
{
static int counter = 0; /*inicia o contador de acessos */
if (Device_Open)
return -EBUSY;
/*impede a abertura por múltiplos processo (reentrância)*/
/* pode ser substituído por um spin_lock()/spin_unlock <linux/spinlock.h
*/
Device_Open++; /* incrementa variável global para controle de acesso */

sprintf(msg, "Device driver acessado pela %d vez \n", counter++);
msg_Ptr = msg;
try_module_get(THIS_MODULE);
/*incrementa uma variável global definida em <linux/module.h> */
return 0; }
```

## Estudo de casos: *Device Drivers* no Linux

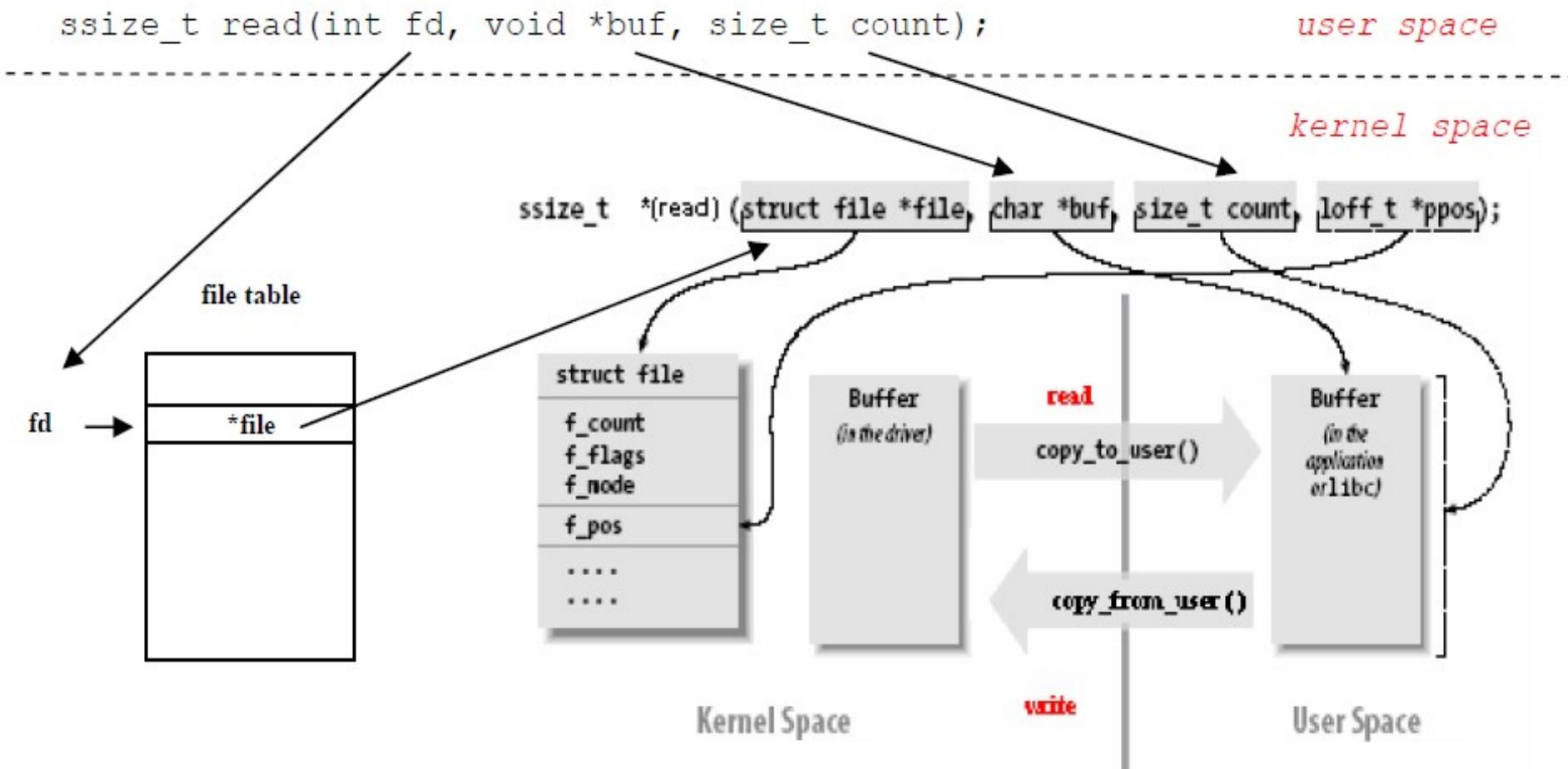
### Exemplo1: um "hello world"

*chardev.c*

```
static int device_release(struct inode *inode, struct file *file){  
/* contrário do open () */  
Device_Open--; /* reinicia o controle de concorrencia */  
printk(KERN_ALERT "Sempre executa release ()\n");  
module_put(TTHIS_MODULE);  
return 0; }  
  
static ssize_t device_read(struct file *filp, char *buffer, size_t  
length, loff_t * offset){  
  
int bytes_read = 0;  
if (*msg_Ptr == 0) return 0;  
while (length && *msg_Ptr) {put_user(* (msg_Ptr++), buffer++);  
/*copia dados do buffer do kernel para o buffer do usuário */  
length--;  
bytes_read++; }  
  
return bytes_read;  
}
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo 1: um "hello world"



modificado (Rubini e Kroah-Hartman, 2008)

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

*chardev.c*

```
static ssize_t
device_write(struct file *filp, const char *buff, size_t len,
loff_t * off)
{
    printk(KERN_ALERT "Operacao write() nao implementada\n");
    return -EINVAL;
}
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

aplicacao.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void) {

    unsigned char teste=1;
    FILE *desc_arq;
    size_t testeread;
    char bufferteste [100];

    desc_arq = fopen ("/dev/chardev", "w+");
    if (desc_arq < 0) {printf ("/dev/chardev"); exit (-1);}

    fwrite (&teste, 1, 1, desc_arq); rewind (desc_arq);

    testeread = fread (bufferteste, sizeof(bufferteste), 1, desc_arq);

    printf ("Lido do buffer do kernel: %s", bufferteste);

    fclose (desc_arq);

    return 0;
}
```

# Estudo de casos: *Device Drivers* no Linux

## Exemplo1: um "hello world"

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

```
obj-m += chardev.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
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

```
root@darkstar:~/modulos# ls
Makefile  aplicacao.c  chardev.c
root@darkstar:~/modulos# make
make -C /lib/modules/2.6.33.4-smp/build M=/root/modulos modules
make[1]: Entering directory `/usr/src/linux-2.6.33.4'
  CC [M]  /root/modulos/chardev.o
Building modules, stage 2.
MODPOST 1 modules
  CC      /root/modulos/chardev.mod.o
  LD [M]  /root/modulos/chardev.ko
make[1]: Leaving directory `/usr/src/linux-2.6.33.4'
root@darkstar:~/modulos# ls
Makefile      aplicacao.c  chardev.ko      chardev.mod.o  modules.order
Module.symvers  chardev.c   chardev.mod.c  chardev.o
root@darkstar:~/modulos# █
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

```
root@darkstar:~/modulos# insmod chardev.ko
root@darkstar:~/modulos# lsmod
Module           Size  Used by
chardev          1976  0
snd_seq_dummy    1107  0
snd_seq_oss      25580 0
snd_seq_midi_event  4620  1 snd_seq_oss
snd_seq          42857  5 snd_seq_dummy, snd_seq_oss, snd_seq_midi_event
snd_seq_device   4543  3 snd_seq_dummy, snd_seq_oss, snd_seq
snd_pcm_oss      33917  0
snd_mixer_oss    13399  1 snd_pcm_oss
ipv6            229909  14
cpufreq_ondemand  6917  1
speedstep_lib    2683  0
powernow_k8      10516  1
freq_table       2027  2 cpufreq_ondemand, powernow_k8
lp                7161  0
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

```
root@darkstar:~/modulos# dmesg
supported from D0 D1 D2 D3hot D3cold
pci 0000:00:04.0: PME# disabled
pci 0000:00:04.1: reg 10: [mem 0xf6489400-0xf64894ff]
ata5.00: configured for MWDMA2
ata5.00: qc timeout (cmd 0xa0)
ata5.00: TEST_UNIT_READY failed (err_mask=0x5)
ata5.00: disabled
ata5: soft resetting link
ata5: EH complete
Device driver registrado com sucesso!
Identificador do dispositivo: 244
Criar link para o dispositivo: mknod /dev/chardev c 244 0
Remova o driver quando acabar: rmmod chardev
Remova o ficherio /dev quando acabar: rm /dev/chardev
Exercicio CMP1240
root@darkstar:~/modulos# █
```

## Estudo de casos: *Device Drivers* no Linux

Exemplo1: um "hello world"

```
root@darkstar:~/modulos# mknod /dev/chardev c 244 0
root@darkstar:~/modulos# ls -l /dev/chardev
crw-r--r-- 1 root root 244, 0 2010-12-22 19:32 /dev/chardev
root@darkstar:~/modulos# cat /dev/chardev
Device driver acessado pela 0 vez
root@darkstar:~/modulos# cat /dev/chardev
Device driver acessado pela 1 vez
root@darkstar:~/modulos# cat /dev/chardev
Device driver acessado pela 2 vez
root@darkstar:~/modulos# █
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo1: um "hello world"

```
root@darkstar:~/modulos# ls
Makefile      aplicacao.c  chardev.ko      chardev.mod.o  modules.order
Module.symvers  chardev.c   chardev.mod.c  chardev.o
root@darkstar:~/modulos# gcc -o aplicacao aplicacao.c
root@darkstar:~/modulos#
```

```
root@darkstar:~/modulos# ls
Makefile      aplicacao*  chardev.c      chardev.mod.c  chardev.o
Module.symvers  aplicacao.c  chardev.ko      chardev.mod.o  modules.order
root@darkstar:~/modulos# ./aplicacao
Lido do buffer do kernel: Device driver acessado pela 3 vez
z·root@darkstar:~/modulos# ./aplicacao
Lido do buffer do kernel: Device driver acessado pela 4 vez
·root@darkstar:~/modulos# ./aplicacao
Lido do buffer do kernel: Device driver acessado pela 5 vez
·root@darkstar:~/modulos# ./aplicacao
Lido do buffer do kernel: Device driver acessado pela 6 vez
y·root@darkstar:~/modulos# █
```

## Estudo de casos: *Device Drivers* no Linux

Exemplo1: um "hello world"

```
root@darkstar:~/modulos# rmmod chardev
root@darkstar:~/modulos# █
```

```
root@darkstar:~/modulos# rm /dev/chardev
root@darkstar:~/modulos# █
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

#### Objetivos:

- Com base no exemplo anterior modificar a função `_write()` para o envio de dados pela porta paralela.
- Escrever o Makefile para a compilação do módulo.
- Implementar um programa de usuário para acessar o *device driver*. Este programa deverá abstrair as funções do dispositivo para o usuário.

#### Para simplificação do hardware:

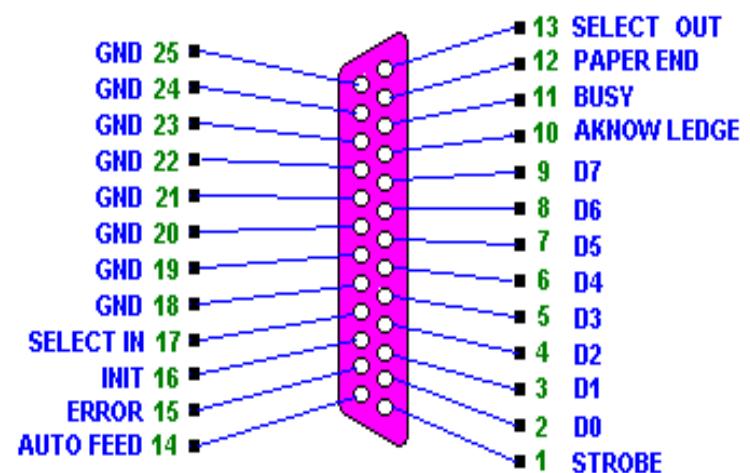
- Apenas envio de dados
- Utilização de apenas dois motores-de-passo unipolares de baixa potência
  - desnecessidade de transdutores, Conversores A/D e D/A, Debounce, multiplexadores, ponte-H, gerador de PWM e circuitos de proteção: buffers, optoacopladores, relês...



## Estudo de casos: *Device Drivers* no Linux

### Porta Paralela

Porta	BIT	Direção	Pinos	Nome
DADOS	7	OUT	9	D7
	6	OUT	8	D6
	5	OUT	7	D5
	4	OUT	6	D4
	3	OUT	5	D3
	2	OUT	4	D2
	1	OUT	3	D1
	0	OUT	2	D0
STATUS	7	IN	11(L)	BUSY
	6	IN	10	*ACKNLG
	5	IN	12	PAPER OUT
	4	IN	13	SELECT
	3	IN	15	*ERROR
CONTROLE	5	-	*OE 378h	-
	4	-	Hab IRQ 7	-
	3	IN/OUT	17(L)	*SLCT IN
	2	IN/OUT	16	*INIT
	1	IN/OUT	14(L)	*AUTO FEED
	0	IN/OUT	1(L)	*STROBE



Tensão:

Nível lógico baixo: 0-0,4v

Nível lógico alto: 3,1-5v

Corrente:

Sink: 24mA

Source: 2,6mA

## Estudo de casos: *Device Drivers* no Linux

### Porta Paralela: leitura

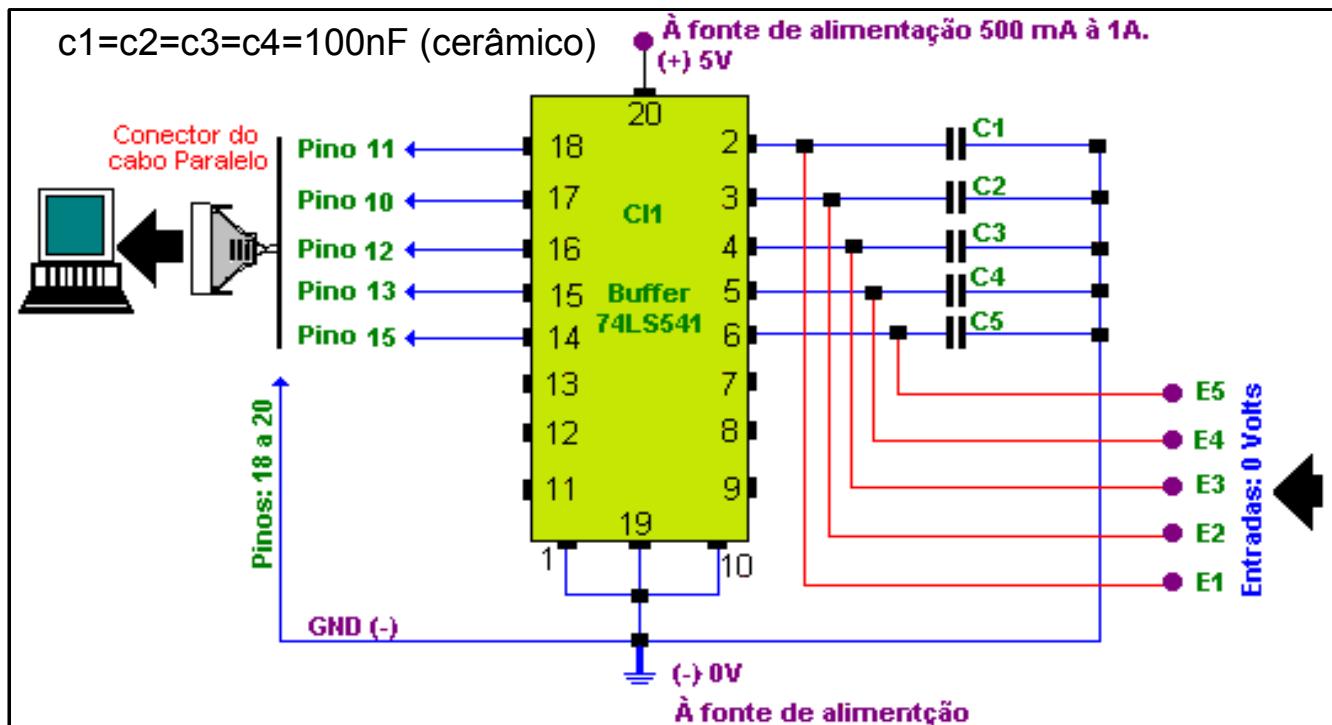
```
#define base 0x379

main(int argc, char **argv)
{
    int value;

    if (ioperm(base,1,1))
        fprintf(stderr, "%x\n", base), exit(1);

    value = inb(base);
    printf("0x%x $i \n",base,value);
}

/* gcc -O lpt_read.c -o lpt_read */
```



Fonte: <http://www.rogercom.com/>

## Estudo de casos: *Device Drivers* no Linux

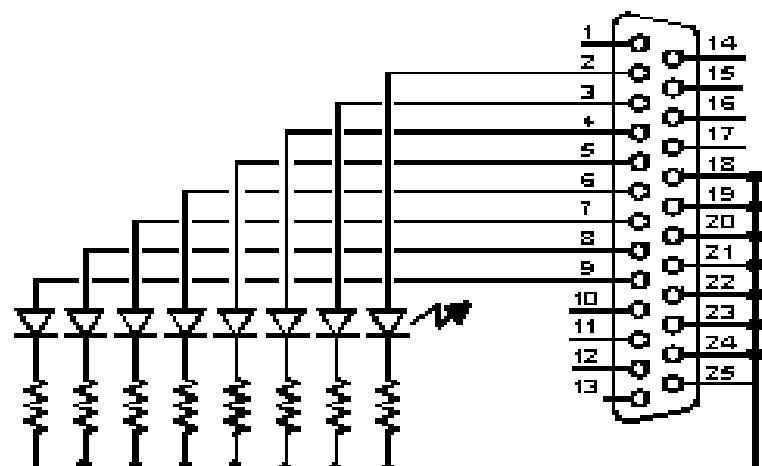
### Porta Paralela: escrita

Decimal	Hexadecimal	Binário	Pino/Fio ativo (5V)
128	80	10000000	9 - D7
64	40	01000000	8 - D6
32	20	00100000	7 - D5
16	10	00010000	6 - D4
8	8	00001000	5 - D3
4	4	00000100	4 - D2
2	2	00000010	3 - D1
1	1	00000001	2 - D0

Fonte: <http://www.rogercom.com/>

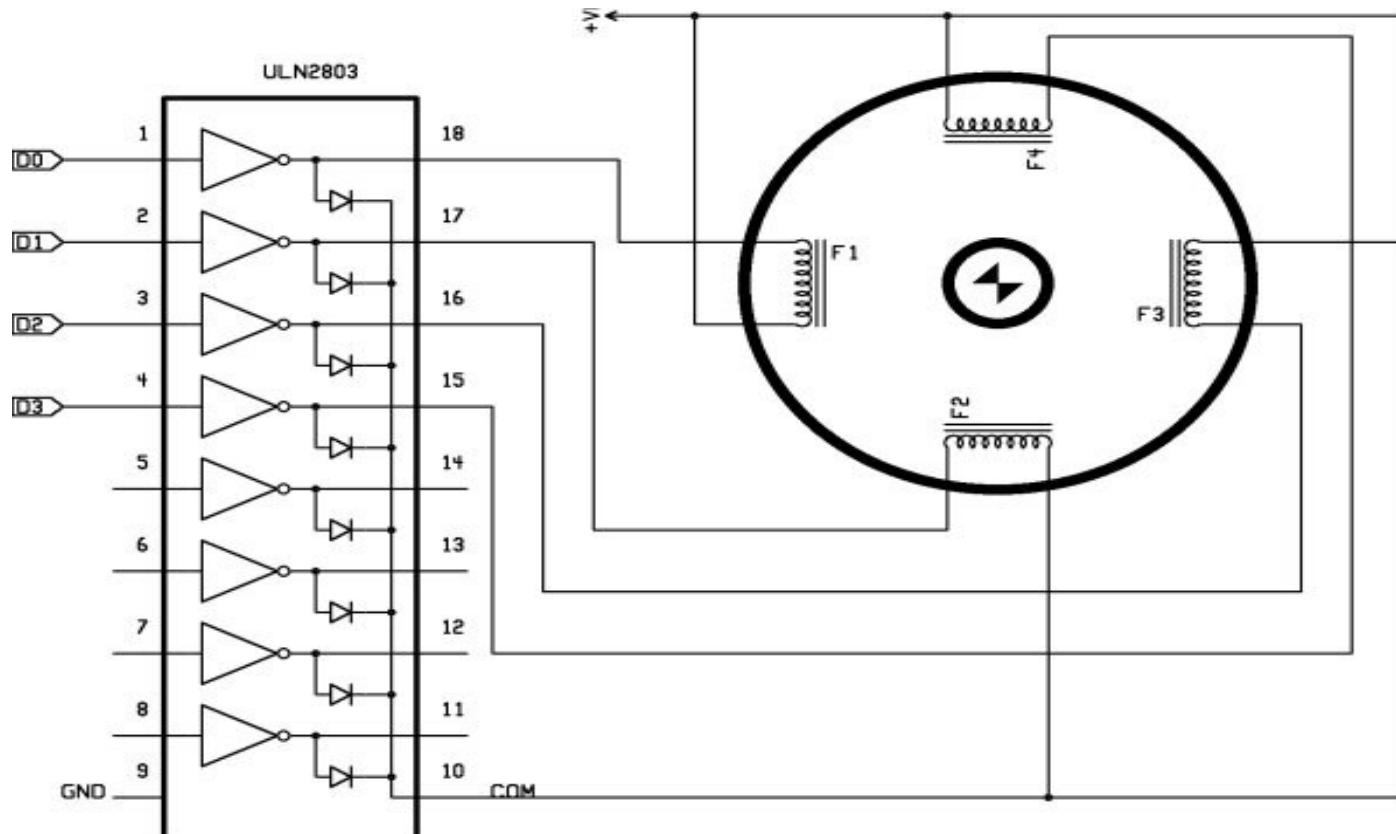
```
..
main(void) {
if( ioperm(0x378, 3, 1) ) { exit(1); }
/*libera endereços: 0x378, 0x379, 0x37A */

        outb(255, 0x378);
/*Todos os pinos --> Registrador de Dados */}
```

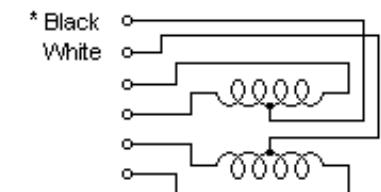


## Estudo de casos: *Device Drivers* no Linux

### Motores de passo unipolar

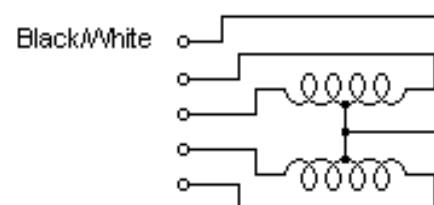


Typical 6-Wire Configuration



\* Sometimes 2 Red

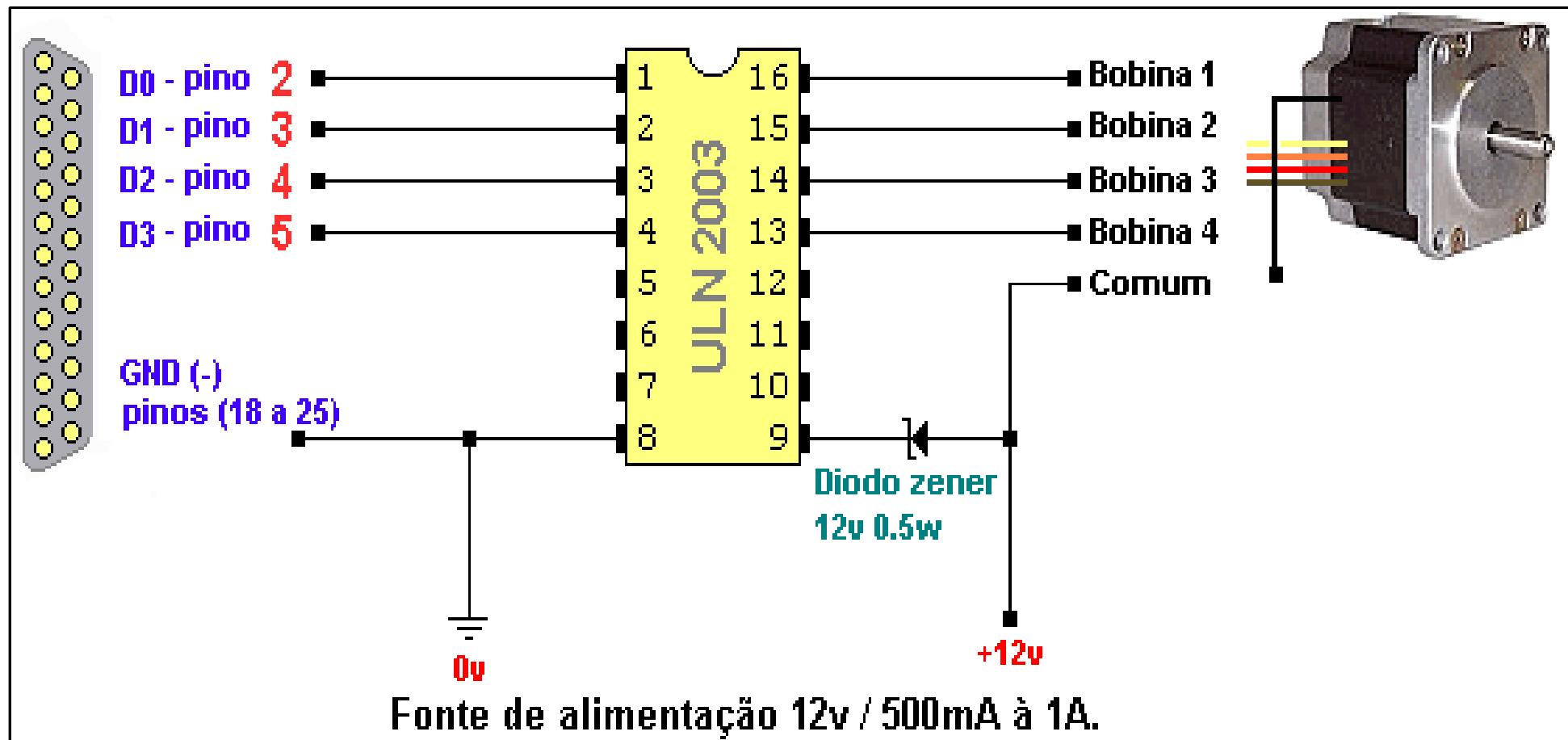
Typical 5-Wire Configuration



Fonte: <http://www.stepperworld.com/Tutorials/pgUnipolarTutorial.htm>

## Estudo de casos: *Device Drivers* no Linux

### Motores de passo unipolar



Fonte: <http://www.rogercom.com/>

## Estudo de casos: *Device Drivers* no Linux

### Motores de passo unipolar: tipos de acionamento

<u>Sequence</u>	<u>Name</u>	<u>Description</u>
0001 0010 0100 1000	Wave Drive, One-Phase	Consumes the least power. Only one phase is energized at a time. Assures positional accuracy regardless of any winding imbalance in the motor.
0011 0110 1100 1001	Hi-Torque, Two -Phase	Hi Torque - This sequence energizes two adjacent phases, which offers an improved torque-speed product and greater holding torque.
0001 0011 0010 0110 0100 1100 1000 1001	Half-Step	Half Step - Effectively doubles the stepping resolution of the motor, but the torque is not uniform for each step. (Since we are effectively switching between Wave Drive and Hi-Torque with each step, torque alternates each step.) This sequence reduces motor resonance which can sometimes cause a motor to stall at a particular resonant frequency. Note that this sequence is 8 steps.

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
#include <linux/module.h>
#include <linux/errno.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/sched.h>
#include <linux/ioport.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/delay.h>
#include <linux/time.h>
#include <linux/unistd.h>
#include <asm/io.h>
#include <asm/segment.h>
#include <asm/system.h>
#include <asm/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Alunos: Alexandre, Allan, Luiz e Yure");
MODULE_DESCRIPTION("Para controle de um guindaste");
MODULE_VERSION("1.0 - Beta");

#define DEVICE_FILE_NAME    "motor"
#define DEVICE_MINOR_NUM    0

static int Major = 61;
static int PORTA=0x378;
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
static int guindaste_open(struct inode *inode, struct file *file);
static int guindaste_release(struct inode *inode, struct file *file);
static size_t guindaste_write(struct file *filp, const char *buffer, size_t len, loff_t
*off);

static struct file_operations guin=
{
    .open=guindaste_open,
    .write=guindaste_write,
    .release=guindaste_release
};

int init_module(void) {
printk(KERN_ALERT,"GUINDASTE: Carregando o modulo... PRIMEIRO");
if(check_region(PORTA,3)<0) {
    printk(KERN_ALERT,"GUINDASTE: Erro ao requisitar porta");
    return -1;
}

request_region(PORTA, 3, DEVICE_FILE_NAME);

if(register_chrdev(Major,DEVICE_FILE_NAME,&guin)) {
    printk(KERN_ALERT "GUINDASTE: Erro ao carregar o modulo.\n");
    return -1;
    printk(KERN_ALERT "GUINDASTE: Modulo carregado com sucesso\n");
return 0;
}
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
void cleanup_module(void)
{
    printk(KERN_INFO "GUINDASTE: Removido modulo...\n");
    unregister_chrdev(Major, DEVICE_FILE_NAME);
    release_region(PORTA, 3);
    printk(KERN_ALERT "GUINDASTE: Modulo removido com sucesso.\n");
}

static int guindaste_open(struct inode *inode, struct file *file)
{
    printk(KERN_ALERT "Driver Braco Mecanico: Abrindo arquivo\n");
    try_module_get(THIS_MODULE);
    /*não deixa q o modulo seja descarregado a qualquer momento*/
    return 0;
} }
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
static size_t guindaste_write(struct file *filp, const char *buffer,
size_t len, loff_t *off)
{
    unsigned int movimento=0, velocidade=10000;
    char cmov;
    copy_from_user(&cmov, buffer, 1);
    printk(KERN_INFO "NUMERO %d", buffer);
    movimento=cmov-48;
    if(movimento==255)
    {
        if(velocidade==10000)
            return;
        velocidade-=1000;
    }
    else if(movimento==0)
    {
        velocidade+=1000;
        return;
    }
    else {
        outb(movimento, PORTA);
    }
    printk(KERN_INFO "GUINDASTE: Movimento: %d", movimento);
}
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
static int guindaste_release(struct inode *inode, struct file *file)
{
    printk(KERN_ALERT "GUINDASTE: Fechando arquivo.\n");
    module_put(THIS_MODULE);
    return 0;
}
```

*MakeFile*

```
obj-m += motor.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
    mknod /dev/motor c 61 0
    insmod motor.ko

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

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

*motor.c*

```
static int guindaste_release(struct inode *inode, struct file *file)
{
    printk(KERN_ALERT "GUINDASTE: Fechando arquivo.\n");
    module_put(THIS_MODULE);
    return 0;
}
```

*MakeFile*

```
obj-m += motor.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
    mknod /dev/motor c 61 0
    insmod motor.ko

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

## Estudo de casos: *Device Drivers* no Linux

### Exemplo2: um guindaste

```
bash-4.1# make
make -C /lib/modules/2.6.33.4/build M=/root/Desktop/driver_final/driver modules
make[1]: Entering directory `/usr/src/linux-2.6.33.4'
  Building modules, stage 2.
    MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-2.6.33.4'
mknod /dev/motor c 61 0
insmod motor.ko
bash-4.1#
```

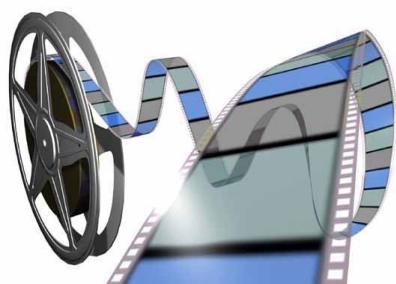
```
bash-4.1# lsmod
Module           Size  Used by
motor            2061  0
snd_seq_dummy    1487  0
snd_seq_oss      29884  0
snd_seq_midi_event  5620  1 snd_seq_oss
snd_seq           52643  5 snd_seq_dummy,snd_seq_oss,snd_seq_midi_event
snd_seq_device    5459  3 snd_seq_dummy,snd_seq_oss,snd_seq
snd_seqoss        28897  0
```

## Estudo de casos: *Device Drivers* no Linux

### Exemplo 2: um guindaste

```
bash-4.1# insmod motor.ko  
bash-4.1#
```

```
bash-4.1# mknod /dev/motor c 61 0  
bash-4.1#
```

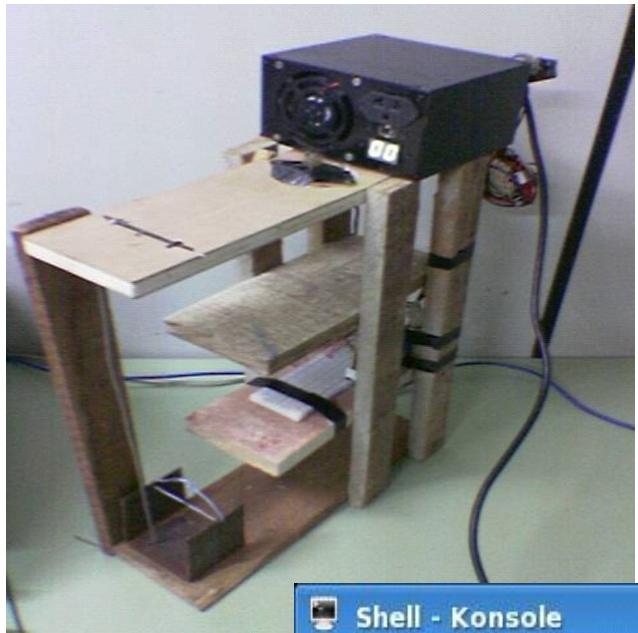


vídeo



## Estudo de casos: *Device Drivers* no Linux

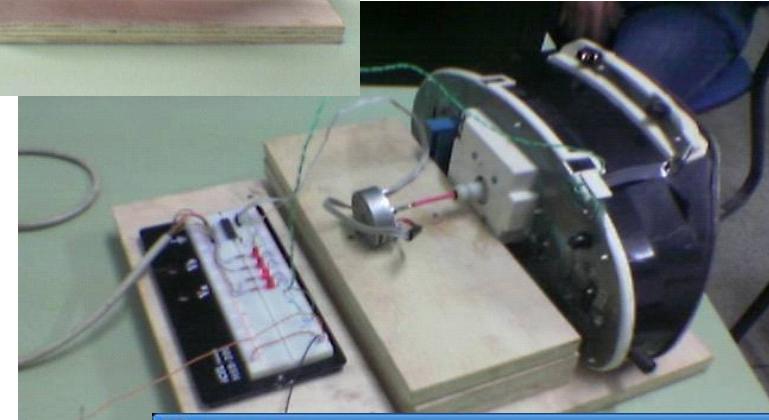
Outros trabalhos da mesma turma:



Shell - Konsole

```
Sessão Editar Ver Favoritos Configurações Ajuda
Shell Shell Núm. 2

root@linux:~/elevador# ./elevador
Informe o Andar Atual: 0
0o - 0o andar;
1o - 1o andar;
2o - 2o andar;
3 - para sair;
Digite a opção: 2
```



Terminal

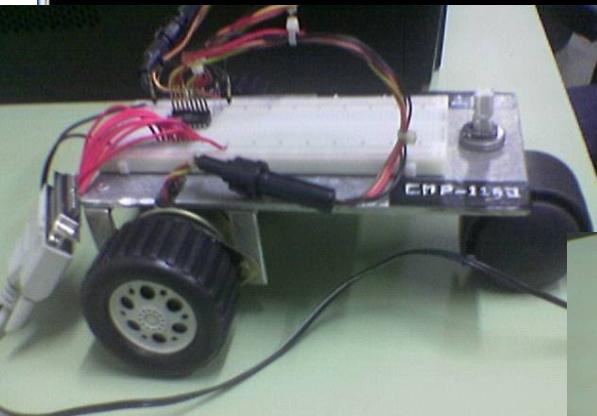
```
File Edit View Terminal Go Help
*****
* Simulador de Velocidade do Automóvel *
* INFORME A OPÇÃO DESEJADA:
*
* 1 - Mover Marcador a 40 Km/h
* 2 - Mover Marcador a 35 Km/h
* 3 - Mover Marcador a 30 Km/h
* 4 - Mover Marcador a 25 Km/h
* 0 - Encerrar Aplicação
*
*****
Opção:1
```

## Estudo de casos: *Device Drivers* no Linux

Outros trabalhos da mesma turma:

Shell - Konsole

```
*****  
**          MOTORHEAD          **  
*****  
** BEM VINDO AO PAINEL DE CONTROLE  **  
*****  
** Digite o comando desejado e tecle enter  **  
**  
** Opções de comando:  
**   w - Andar para frente  
**   s - Andar para tras  
**   a - Gira para a esquerda  
**   d - Gira para a direita  
**   p - Encerrar o programa  
*****
```

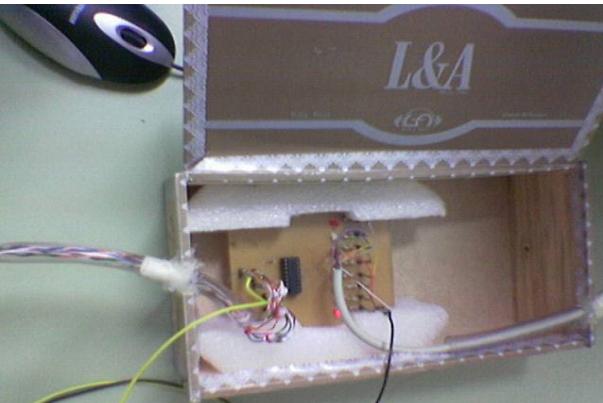


Terminal

```
File Edit View Terminal Go Help  
LIQUIDIFICADOR CLISQUEI  
ENTRE COM A VELOCIDADE  
1 - VELOCIDADE 1  
2 - VELOCIDADE 2  
3 - GIRAR AO CONTRARIO NA VELOCIDADE 1  
4 - GIRAR AO CONTRARIO NA VELOCIDADE 2  
0 - SAIR
```

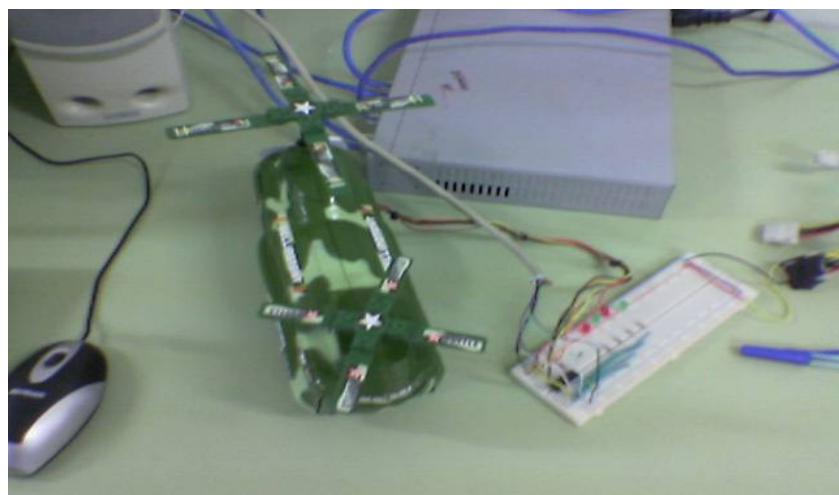
## Estudo de casos: *Device Drivers* no Linux

Outros trabalhos da mesma turma:



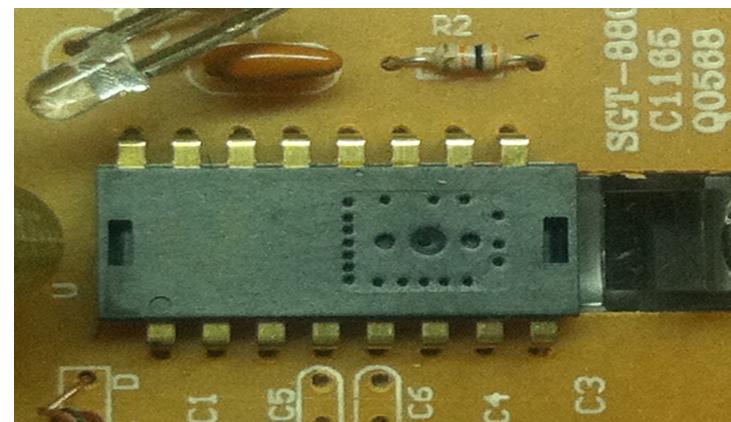
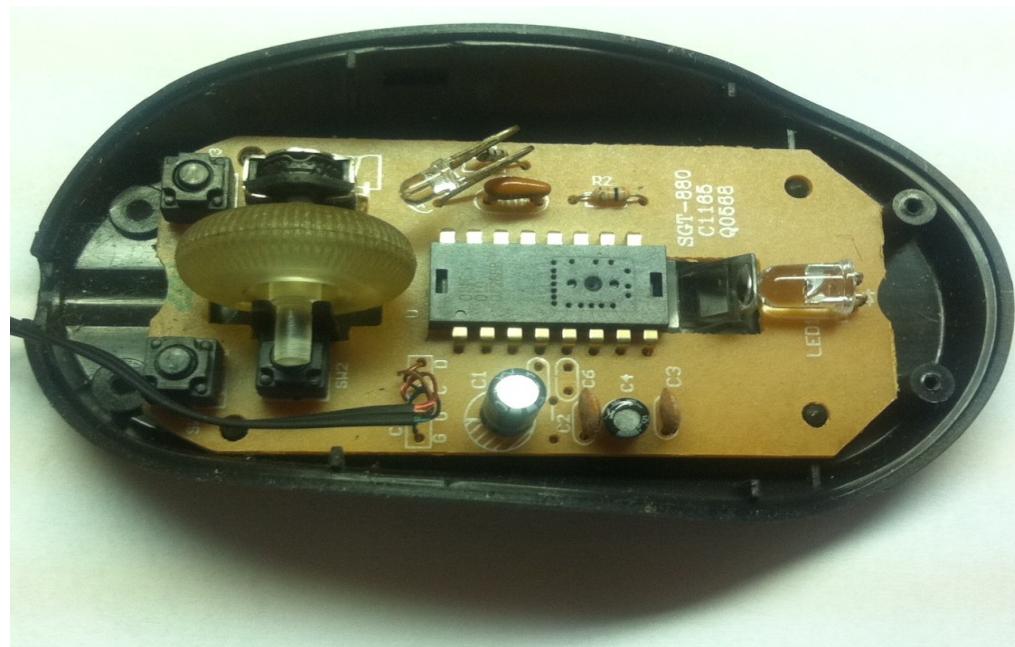
## Estudo de casos: *Device Drivers* no Linux

Outros trabalhos da mesma turma:



- Atividade Complementar CMP1240 (N1)
- Máximo 1,0 ponto acrescido à nota N1
- Objetivo: implementar um dispositivo de E/S
  - Confeccionar a placa de circuito impresso
    - Fazer o desenho da placa no Eagle ou no Proteus
    - Placa de única face
    - Para corroer e perfurar a placa, procurar o monitor de CMP1240, na sala 504E.
  - Não é necessário utilizar um controlador dedicado
    - Pode-se utilizar o controlador da porta paralela, porta serial, um controlador PS2 ou um controlador USB embutido em mouse, teclado ou qualquer outro.
  - Implementar o driver como um módulo do kernel
  - Implementar um programa de aplicação

- Para os alunos que já cursaram CMP1160 com o professor Talles, o dispositivo em questão deve ser para entrada de dados
- Uma sugestão: abstrair o hardware usado num mouse antigo



C1165-11