

MC504 - Sistemas Operacionais

Entrada e Saída

Pipes

Islene Calciolari Garcia

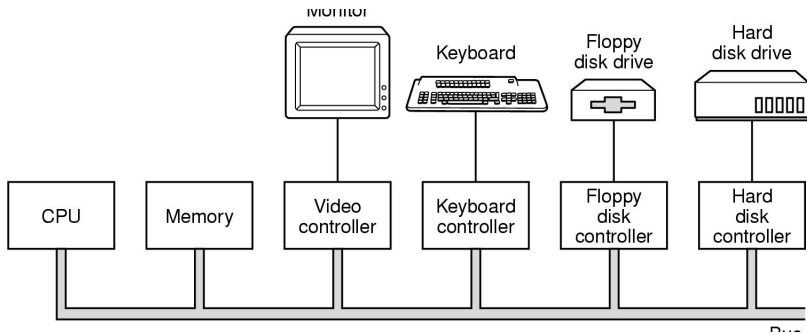
Instituto de Computação - Unicamp

Primeiro Semestre de 2014

Sumário

1 Introdução

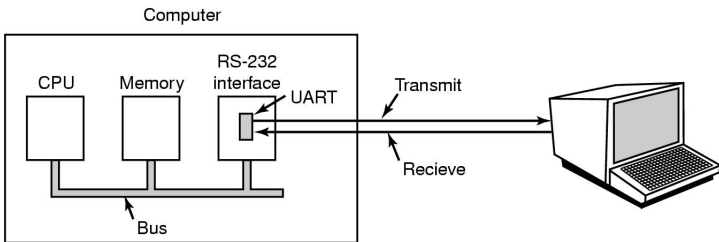
Dispositivos de I/O e controladores



Tanenbaum: Figura 1.5

O sistema operacional deve interagir com os controladores

Character device

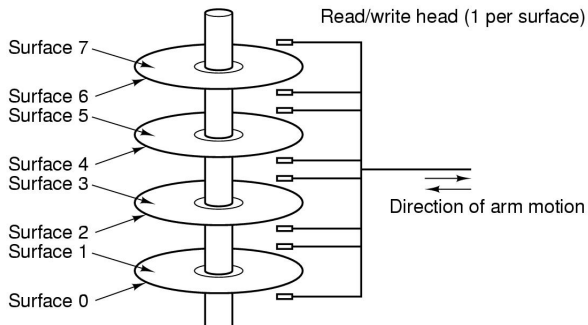


Tanenbaum: Figura 5.34

Acesso sequencial, caractere a caractere

```
Execute ls -l /dev
```

Block device

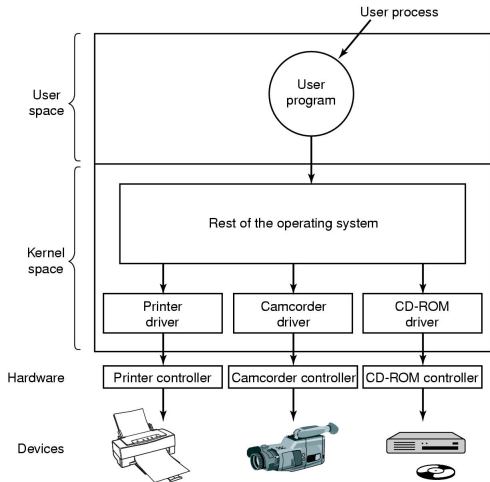


Tanenbaum: Figura 1.8

Acesso não sequencial a blocos de informação

Execute `ls -l /dev`

Device drivers



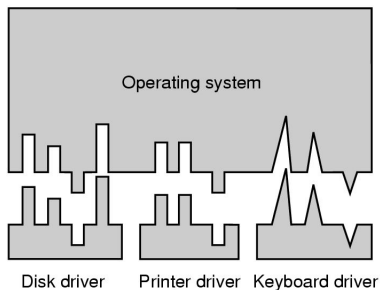
Tanenbaum: Figura 5.11

Device drivers

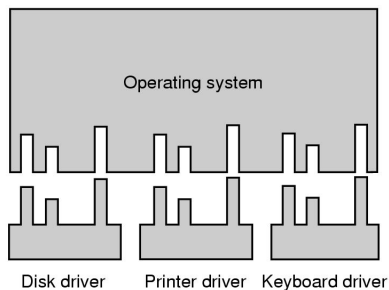
- Software que “conversa” com o controlador
- Os fabricantes devem fornecer dados detalhados para a escrita dos device drivers
- Como acoplar um device driver ao kernel:
 - relink e reboot
 - entrada em um arquivo e reboot
 - on-the-fly
veja o comando `lsmod`

Device drivers

Sem ou com uma interface padrão



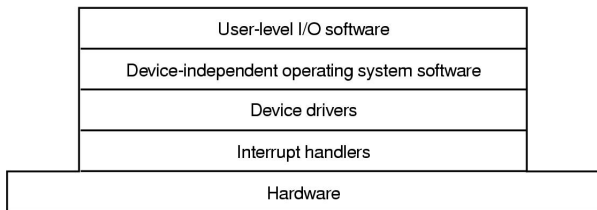
(a)



(b)

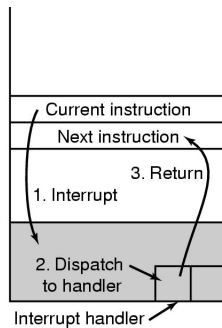
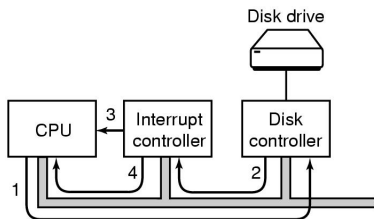
Tanenbaum: Figura 5.13

Camadas de software



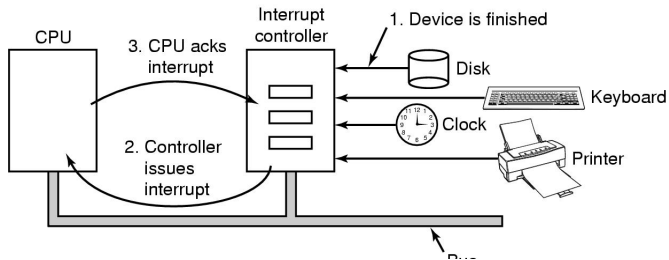
Tanenbaum: Figura 5.10

Tratamento de interrupções



Tanenbaum: Figura 1.10

Tratamento de interrupções



Tanenbaum: Figura 5.5

Como programar os dispositivos?

- Instruções especiais

```
IN REG, PORT
```

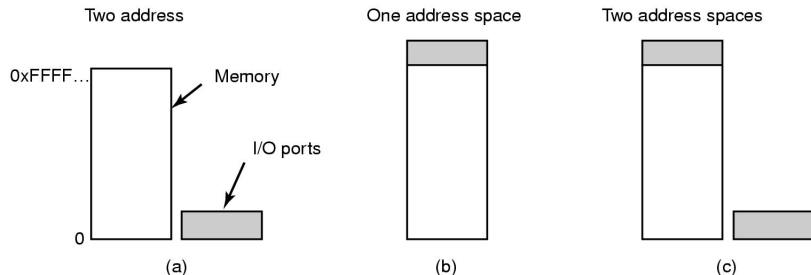
```
OUT PORT, REG
```

- Memory-mapped I/O

```
MOV REG, ADDR
```

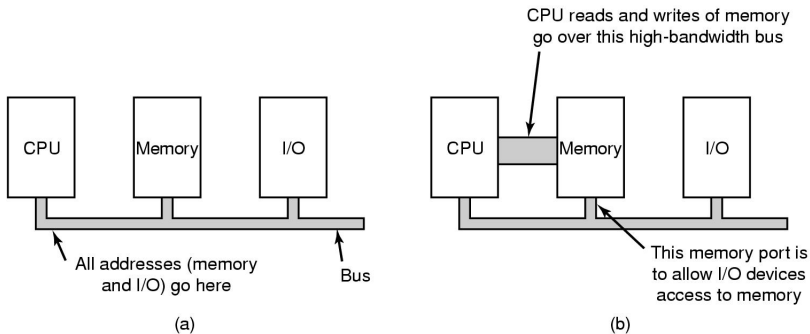
Conforme o valor de ADDR, a instrução MOV fará acesso a uma palavra de memória ou dispositivo

Como programar os dispositivos?



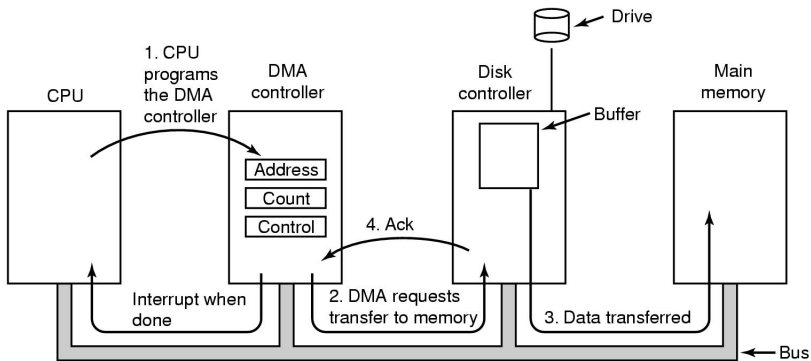
Tanenbaum: Figura 5.2

Barramento simples e dual



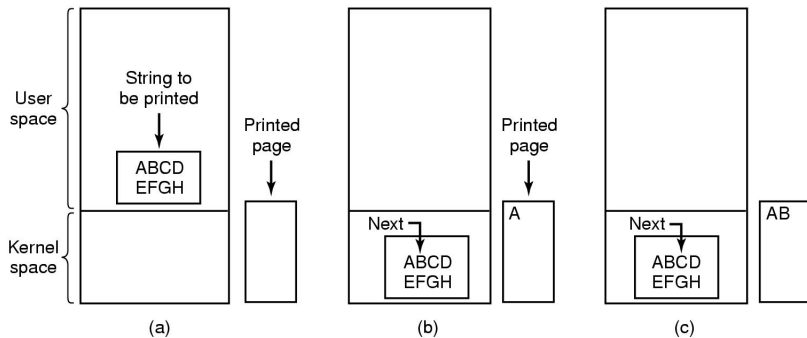
Tanenbaum: Figura 5.3

Direct Memory Access (DMA)



Tanenbaum: Figura 5.4

Imprimindo uma string



Tanenbaum: Figura 5.6

Imprimindo uma string

Programmed I/O

```
copy_from_user(buffer, p, count);
for (i = 0; i < count; i++) {
    while (*printer_status_reg != READY) ;
    *printer_data_register = p[i];
}
return_to_user();
```

/ p is the kernel bufer */*
/ loop on every character */*
/ loop until ready */*
/ output one character */*

Tanenbaum: Figura 5.7

Trecho de código do kernel

Imprimindo uma string

Interrupt-driven I/O

```
copy_from_user(buffer, p, count);
enable_interrupts();
while (*printer_status_reg != READY);
*printer_data_register = p[0];
scheduler();
```

(a)

```
if (count == 0) {
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count - 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

(b)

Tanenbaum: Figura 5.8

(a) Trecho de código do kernel

(b) Tratador da interrupção

Imprimindo uma string

DMA

```
copy_from_user(buffer, p, count);  
set_up_DMA_controller();  
scheduler();
```

(a)

```
acknowledge_interrupt();  
unlock_user();  
return_from_interrupt();
```

(b)

(a) Trecho de código do kernel

(b) Tratador de interrupção

Pipes

```
$ grep xxx log.txt > log-xxx.txt
```

```
$ wc -l log-xxx.txt
```

```
$ rm log-xxx.txt
```

```
$ grep xxx log.txt | wc -l
```

pipe()

```
int pipe (int FILEDES[2])
```

The 'pipe' function creates a pipe and puts the file descriptors for the reading and writing ends of the pipe (respectively) into 'FILEDES[0]' and 'FILEDES[1]'.

Veja o código: `mypipe.c`

Pipe com entrada e saída padrão?

```
int dup2(int oldfd, int newfd);
```

dup2 makes newfd be the copy of oldfd, closing newfd first if necessary. After successful return of dup or dup2, the old and new descriptors may be used interchangeably.

Veja o código: `mypipe2.c`

Processos conectados de maneira transparente

```
$ cm1 <args1> | cmd2 <args2>
```

- A modificação da entrada e saída padrão deve ser feita antes da chamada a `execve()`.
- Veja o código: `minishell.c`

popen()

```
FILE *popen(const char *command,  
            const char *type);  
int  pclose(FILE *stream);
```

The popen() function opens a process by creating a pipe, forking, and invoking the shell. Since a pipe is by definition unidirectional, the type argument may specify only reading or writing, not both; the resulting stream is correspondingly read-only or write-only.

Veja o código: `mypopen.c` e `mypopen2.c`

Programando um device driver

- Veja a série: Device drivers de Anil Kumar Pugalia
- Exemplo número 1: `ofd.c`
- Desafio: como implementar um device driver com comportamento de pipe?