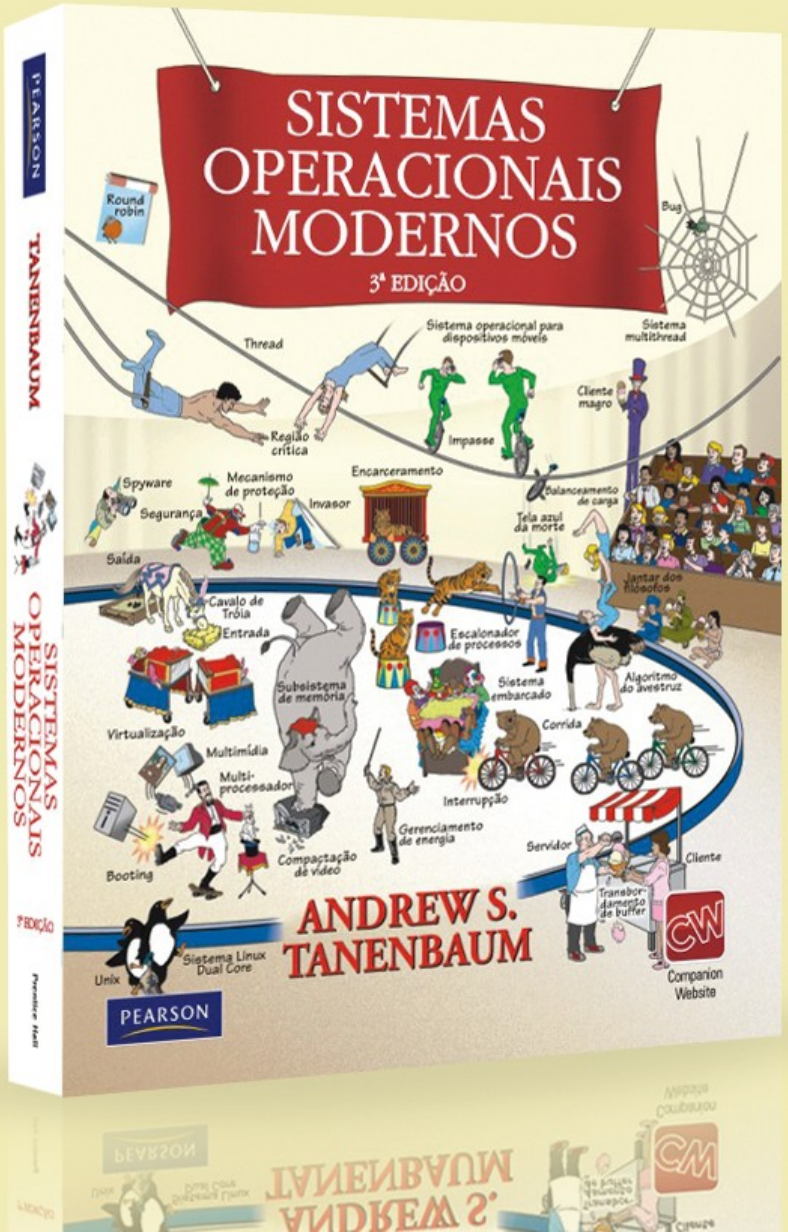
Sistemas operacionais modernos

Terceira edição
ANDREW S. TANENBAUM

Capítulo 5
Entrada/Saída



Capítulo 5: Sistemas de E/S

- Além das abstrações oferecidas pelo SO (processos, EE, arquivos) o mesmo também controla os dispositivos de E/S, emitindo comandos, interceptando interrupções, tratando erros, fornece interface dispositivo/sistema facilitando o uso. O código de E/S representa uma parte significativa de todo o SO.
- Organização de E/S
  - Parte 1
    - Princípios de Hardware de E/S
    - Software de E/S
      - Níveis de software de E/S
  - Parte 2
    - Dispositivos de E/S: Disco

Princípios de Hardware de E/S

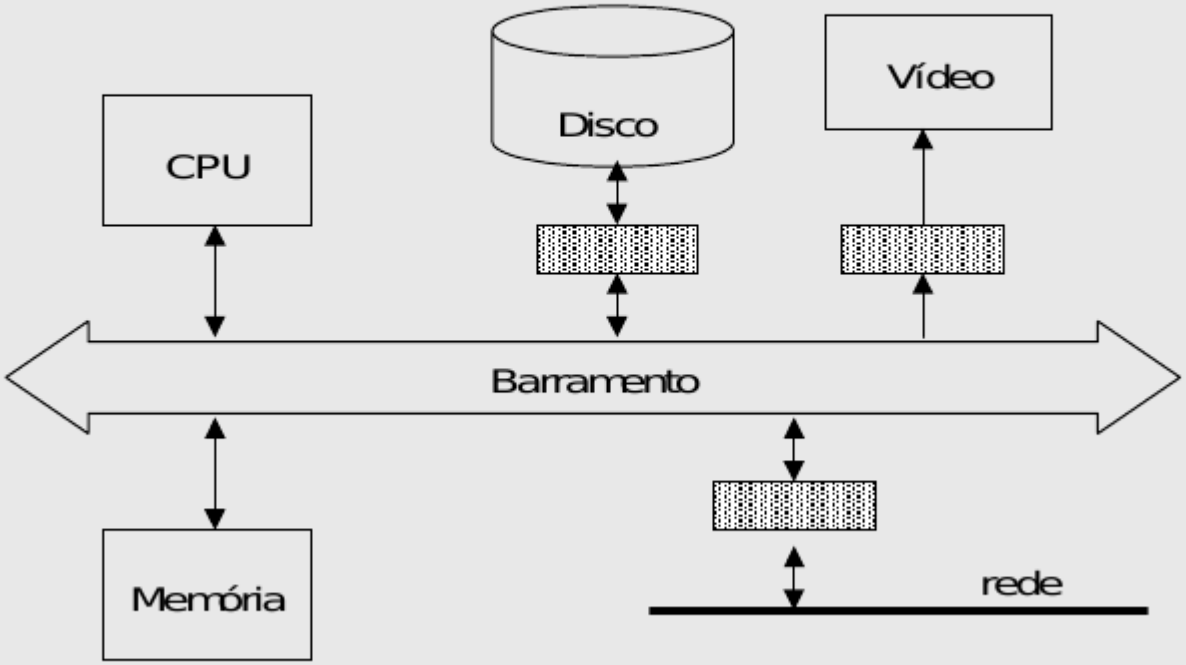
- Periférico é um dispositivo conectado a um computador de forma a possibilitar sua interação com o mundo externo
- Os periféricos são conectados ao computador através de um componente de hardware denominado de interface
- As interfaces são interconectadas aos barramentos internos de um computador
  - Elemento chave na coordenação da transferências de dados
- Interfaces se utilizam de um processador dedicado a realização e controle das operações de entrada e saída
  - Controladoras

Controladores de Dispositivos

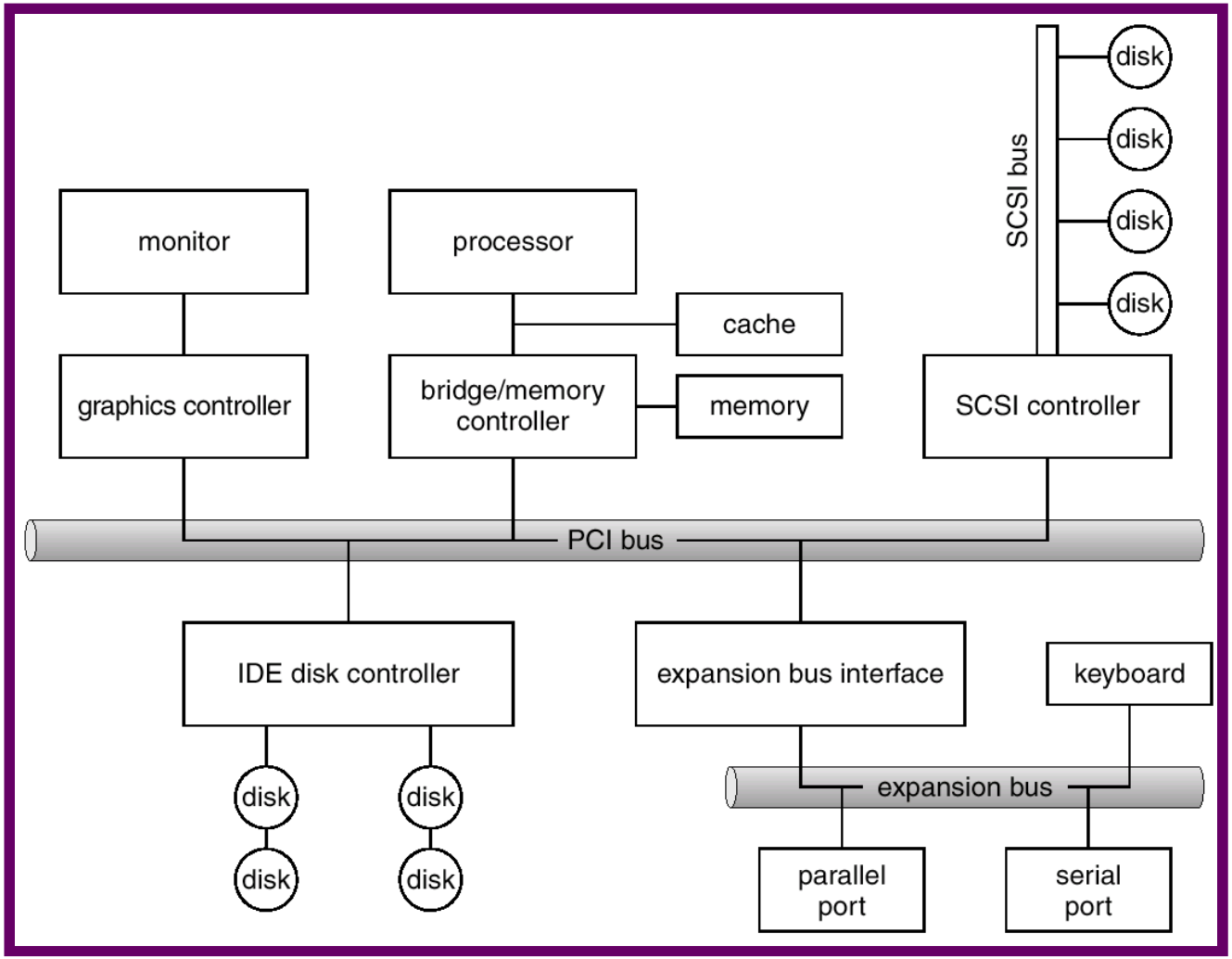
- Dispositivos de E/S tem componentes:
  - mecânico
  - eletrônico
- O componente eletrônico é o controlador do dispositivo
  - Placa controladora circuito impresso, conector para o dispositivo
  - Pode tratar com múltiplos dispositivos
- Interface entre controlador e dispositivo
  - Padrão oficial (ANSI,IEEE,ISO) – discos (IDE, SATA,SCSI,USB...)
  - Interface de nível baixo: unidade de disco entrega fluxo serial de bits (preâmbulo+4096bits(setor)+checksum-ECC)
- Tarefas do Controlador
  - converter um “stream” serial de bits em blocos de bytes
  - realizar correção de erros se necessário
  - tornar dados disponíveis para memória principal
  - SO inicia o controlador com parâmetros

Arquitetura de E/S

- Dispositivo de entrada e saída possui um parte mecânica e outra eletrônica



Arquitetura PC



Dispositivos de E/S

- Classificados como:
  - Orientado a caractere
    - Unidade de transferência é o caractere
      - e.g.; teclado, interface serial
  - Orientado a bloco
    - Unidade de transferência de dados é um bloco de caracteres (fixo)
      - e.g.; disco, fitas DAT
- Esquema de classificação não é perfeito pois alguns dispositivos não se enquadram nestas situações
  - e.g.; relógio, memória de vídeo mapeada em espaço de E/S

Dispositivos de E/S

- Apresentam características próprias
  - Taxa de transferência de dados
  - Complexidade de controle
  - Unidade de transferência
    - Caractere, bloco ou stream
  - Representação de dados
    - Esquemas de codificação
  - Tratamento de erros
    - Depende do tipo de dispositivo

Dispositivos de E/S

Dispositivo	Taxa de dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem 56 K	7 KB/s
Scanner	400 KB/s
Filmadora *camcorder* digital	3,5 MB/s
Rede sem fio 802,11g	6,75 MB/s
CD-ROM 52x	7,8 MB/s
Fast Ethernet	12,5 MB/s
Cartão flash compacto	40 MB/s
FireWire (IEEE 1394)	50 MB/s
USB 2.0	60 MB/s
Padrão SONET OC-12	78 MB/s
Disco SCSI Ultra 2	80 MB/s
Gigabit Ethernet	125 MB/s
Drive de disco SATA	300 MB/s
Fita Ultrium	320 MB/s
Barramento PCI	528 MB/s

Tabela 5.1 Algumas taxas de dados típicas de dispositivos, placas de redes e barramentos.

Interação SO Controlador

- Controladora é programada via registradores de configuração
  - Recebem ordens do processador
  - Fornecem estados de operação
  - Leitura e escrita de dados do periférico
- Registradores são "vistos" como posições de memória
  - E/S mapeada em espaço de E/S
  - E/S mapeada em espaço de memória

E/S Mapeada na memória (1)

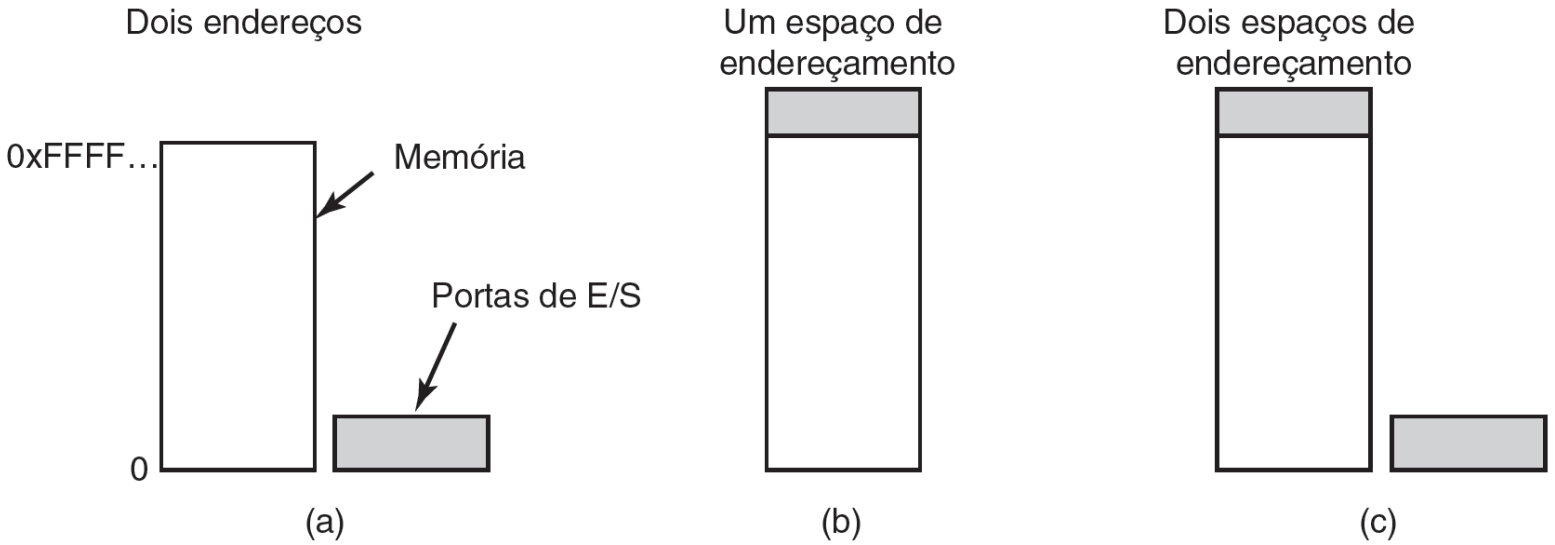


Figura 5.1 (a) Espaços de memória e E/S separados. (b) E/S mapeada na memória. (c) Híbrido.

- Espaço de E/S e memória separados
- E/S Mapeada na memória
- Híbrido

E/S: comunicação c/dispositivos

- Porta de E/S (IBM360)
  - Registrador associado a porta (inteiro 8,16 bits)
  - IN reg,porta ; OUT porta,reg
  - Espaços de endereçamento diferentes (figura)
  - Instruções IN r0,4 e MOV r0,4
- E/S Mapeada na memória (PDP11)
  - Mapear registradores no espaço de endereçamento da memória
  - Cada registrador é associado a endereço de memória único
- Esquema híbrido(Pentium)
  - Portas de E/S de 0 a 64K e endereços 640K a 1M buffers (dispositivos PC)
  - CPU coloca endereço nas linhas de endereço do barramento e emite READ
  - Segunda linha de sinal indica se E/S ou memória

E/S Mapeada na memória

- Vantagens da E/S mapeada
  - Não precisa instruções especiais (código específico) para ler/escrever nos registradores
  - Sem necessidade de mecanismo especial de proteção, não mapear EE dos registradores no EEV do usuário.
  - Registradores de controle são variáveis na memória, ou seja, Driver pode ser escrito totalmente em C
  - Instruções de referencia a memória referenciam registradores
    - Loop: TEST PORT_4 // verifica se a porta 4 é 0
      BEQ READY // se for 0, salta para
      BRANCH LOOP // senão, continua
      READY:

- Desvantagens
  - Utilização de cache é desastrosa (desabilitar)
  - Um espaço de endereçamento para todos, módulos de memória e dispositivos devem examinar referencias – responder
    - Barramento de memória de alta velocidade, dispositivos de E/S não conseguem examinar

E/S Mapeada em Memoria

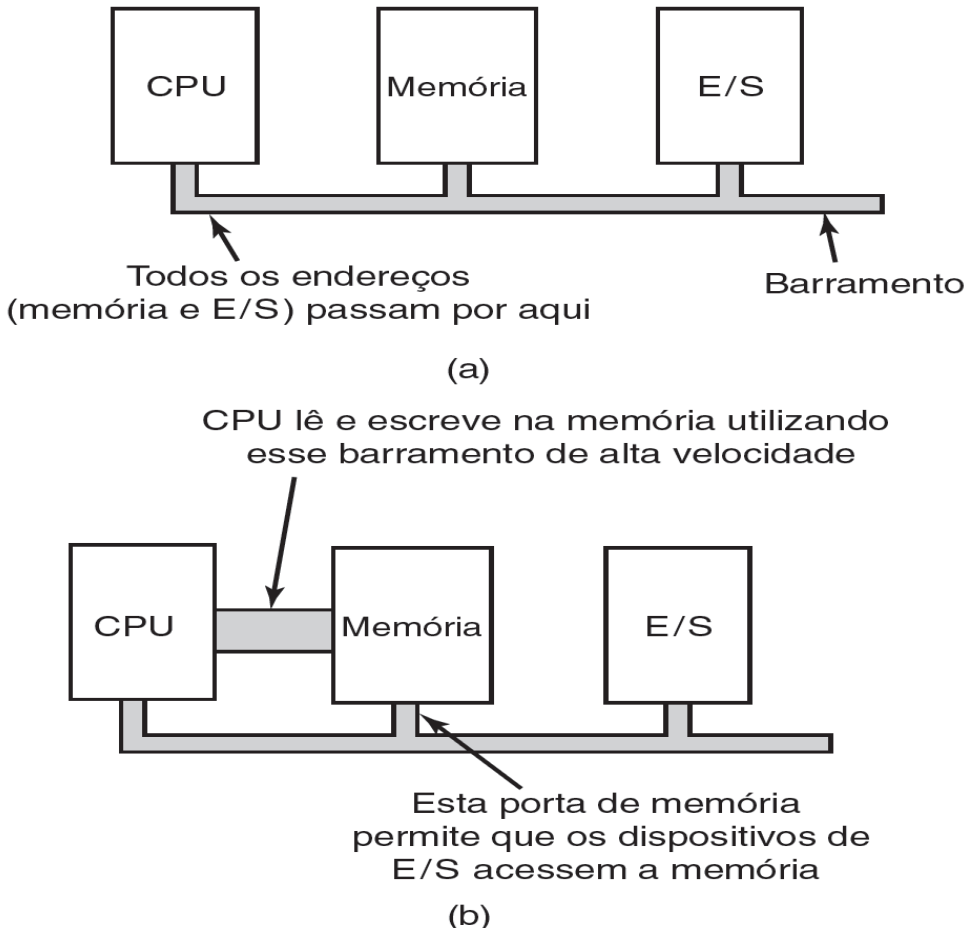


Figura 5.2 (a) Arquitetura com barramento único. (b) Arquitetura de memória com barramento duplo.

E/S Mapeada em Memoria

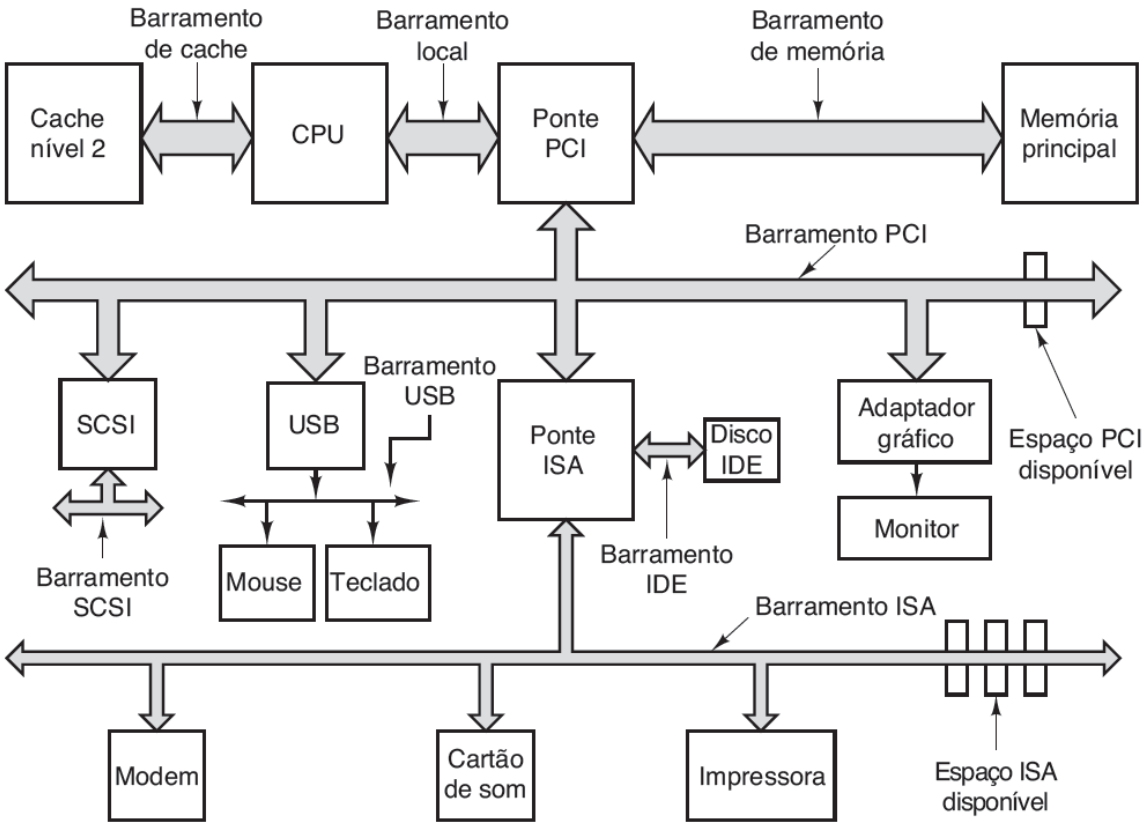


Figura 1.12 A estrutura de um sistema Pentium grande.

- Pentium : endereços reservados na inicialização em Ponte PCI, exemplo, 640K a 1M-1, são desviados para PCI.

Exemplo de acesso

- Controladora de impressão onde um registrador fornece o "status" da impressão (end. 315H) e outro corresponde ao envio do caracter a ser impresso (end. 312H).

Mapeado em memória	Mapeado em entrada e saída
`Le_status: mov AL, 315H`	`Le_status: in AL, 315H`
`Print_char: mov AL, 65H` `mov 312H, AL`	`Print_char: mov AL, 65H` `out 312H, AL`

Formas de realizar E/S

- E/S Programada
- E/S dirigida por Interrupção
- E/S usando DMA

Ciclo de E/S com Interrupções

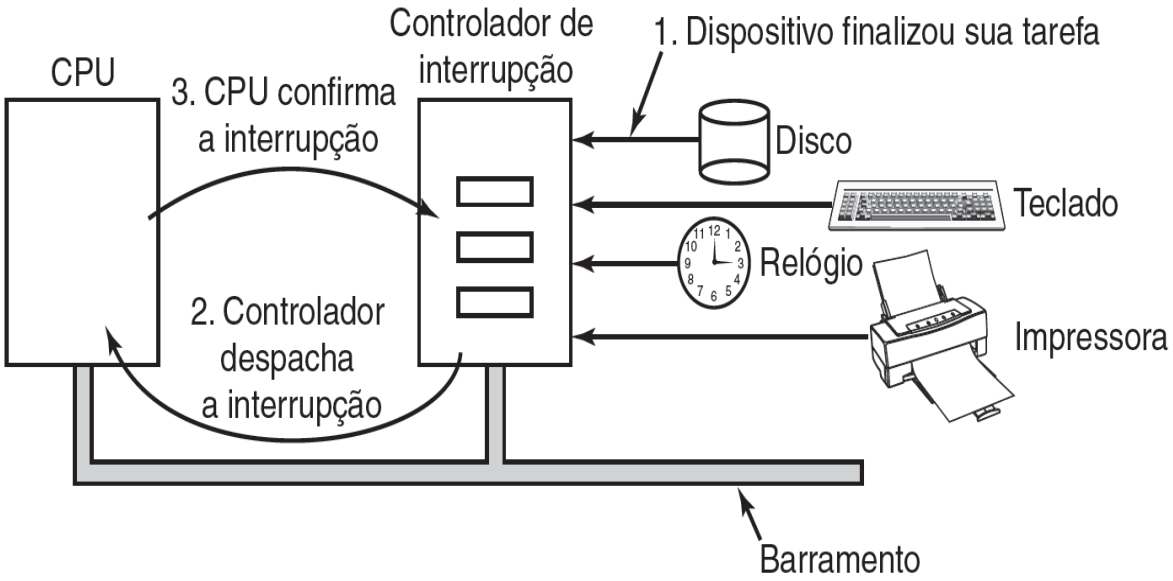


Figura 5.4 Como ocorre uma interrupção. As conexões entre os dispositivos e o controlador de interrupção atualmente utilizam linhas de interrupção no barramento, em vez de cabos dedicados.

Dispositivo gera interrupção, controlador de interrrupção executa ação adequada e coloca número nas linhas de endereço (índice do vetor de interrupções). Tratador de interrupção executa, confirma a interrupção.

Interrupções precisas e imprecisas

Uma interrupção que deixa a máquina em um estado bem definido é chamada de *interrrupção precisa*, (Walker,1995) ela possui 4 propriedades:

1. o PC é salvo em lugar conhecido

2. todas as instruções anteriores àquela apontada pelo PC foram totalmente executadas

3. nenhuma instrução posterior àquela apontada pelo PC foi executada

4. o estado de execução da instrução apontada pelo PC é conhecido

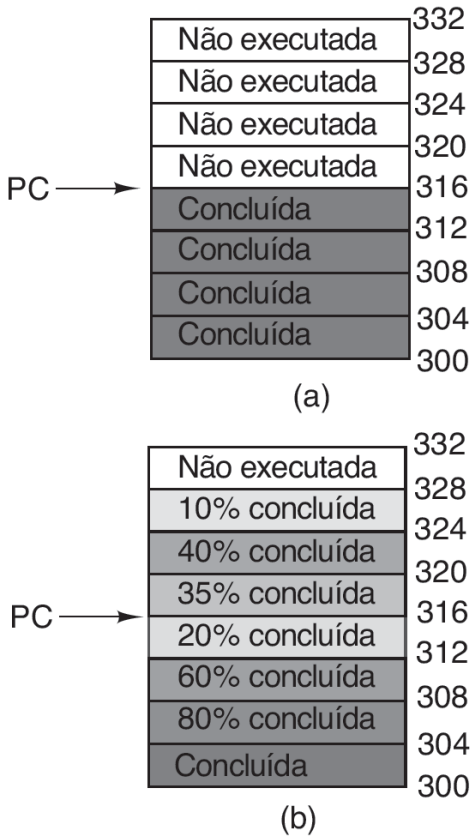


Figura 5.5 (a) Uma interrupção precisa. (b) Uma interrupção imprecisa.

Ciclo de E/S com transferência por DMA

O controlador de DMA é utilizado pelo SO para programar operações de E/S em dispositivos. Os registradores de controle são acessados pela CPU. Eles especificam porta de E/S, direção e unidade da transferência, bytes a transferir.

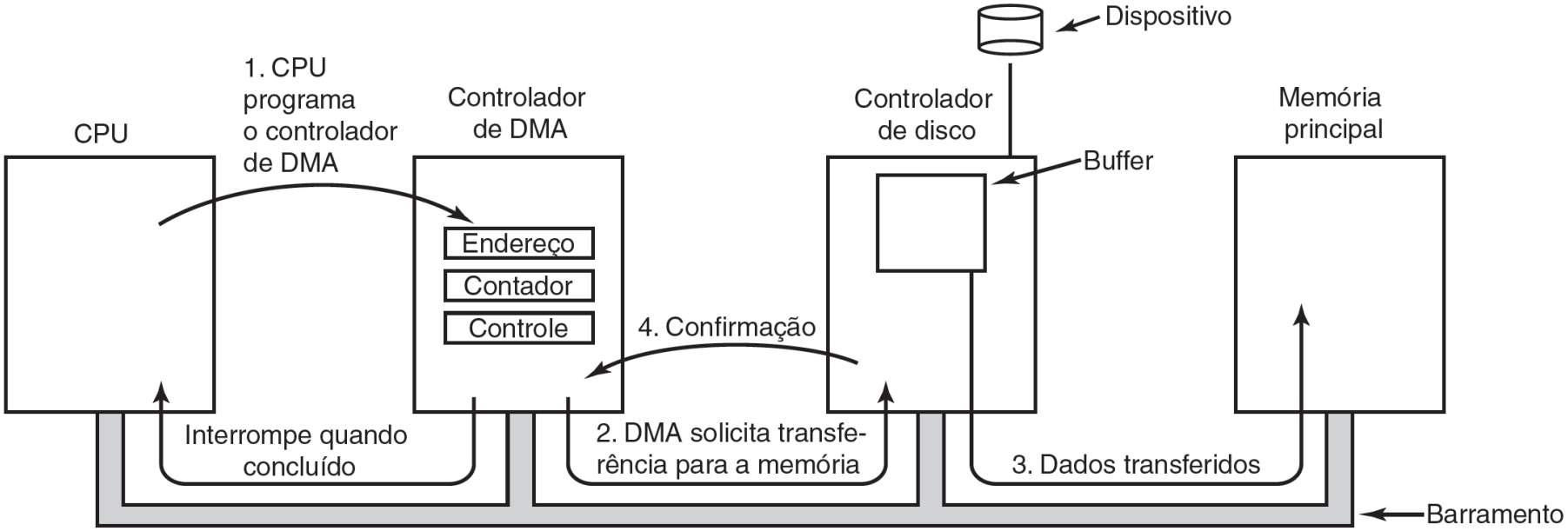


Figura 5.3 Operação de transferência utilizando DMA.

Ex: leitura de disco: programa o controlador DMA, requisita leitura, escrita na memória, confirma escrita, sinaliza CPU.

Formas de realizar E/S

- E/S Programada
  - Processo fica esperando (busy-waiting) pelo término da operação
- E/S dirigida por Interrupção
  - Comando de E/S é emitido
  - Processador continua executando
  - Modulo de E/S envia interrupção quando termina
- E/S usando DMA
  - Driver de E/S programa controlador DMA
  - Processador não participa da transferência

E/S Programada (1)

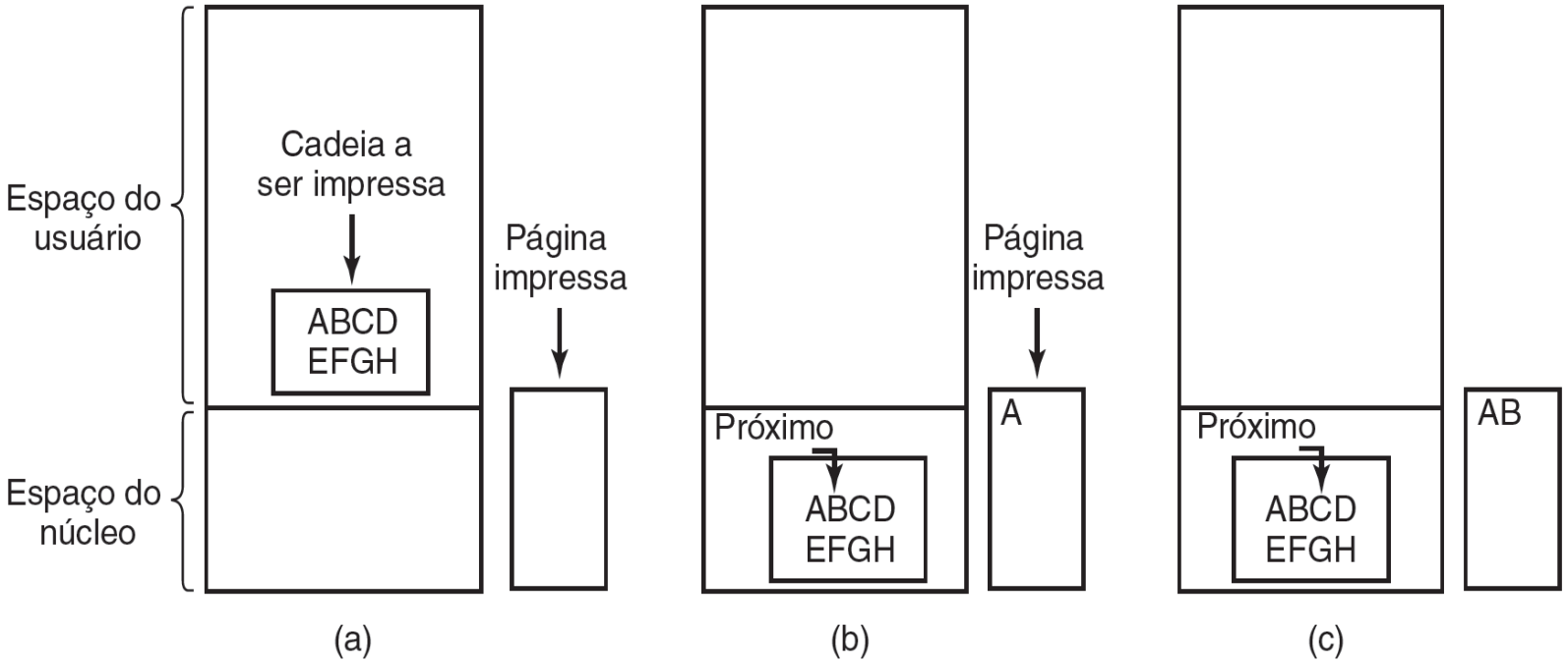


Figura 5.6 Estágios da impressão de uma cadeia de caracteres.

E/S Programada (2)

```
copy_from_user(buffer, p, cont);                    /* p é o buffer do núcleo */
for (i=0; i < count; i++) {                         /* executa o laço para cada caractere */
    while (*printer_status_reg !=READY) ;           /* executa o laço até a impressora estar pronta*/
    *printer_data_register = p[i];                  /* envia um caractere para a saída */
}
return_to_user();
```

Figura 5.7 Como é escrita uma cadeia de caracteres para a impressora usando E/S programada.

Dados são copiados para o núcleo, SO entra em laço de envio dos caracteres. A cada saída o processador fica verificando se o dispositivo esta pronto para o próximo.

E/S dirigida por Interrupção

```
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)

Figura 5.8 Como escrever uma cadeia de caracteres na impressora usando E/S orientada à interrupção. (a) Código executado quando é feita a chamada de sistema para impressão. (b) Rotina de tratamento da interrupção.

- Se impressora leva 10ms/c, processador pode chavear contexto
  - Código executado quando função de impressão e chamada
  - Rotina de tratamento da interrupção

E/S usando DMA

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

(a)

```
acknowledge_interrupt( );
unblock_user( );
return_from_interrupt( );
```

(b)

Figura 5.9 A impressão de uma cadeia de caracteres usando o DMA. (a) Código executado quando é feita a chamada de sistema print. (b) Rotina de tratamento da interrupção.

- A impressão de cada caracter causa uma interrupção, alternativa DMA
  - codigo executado na chamada
  - Rotina de atendimento da interrupção

Camadas de software de E/S

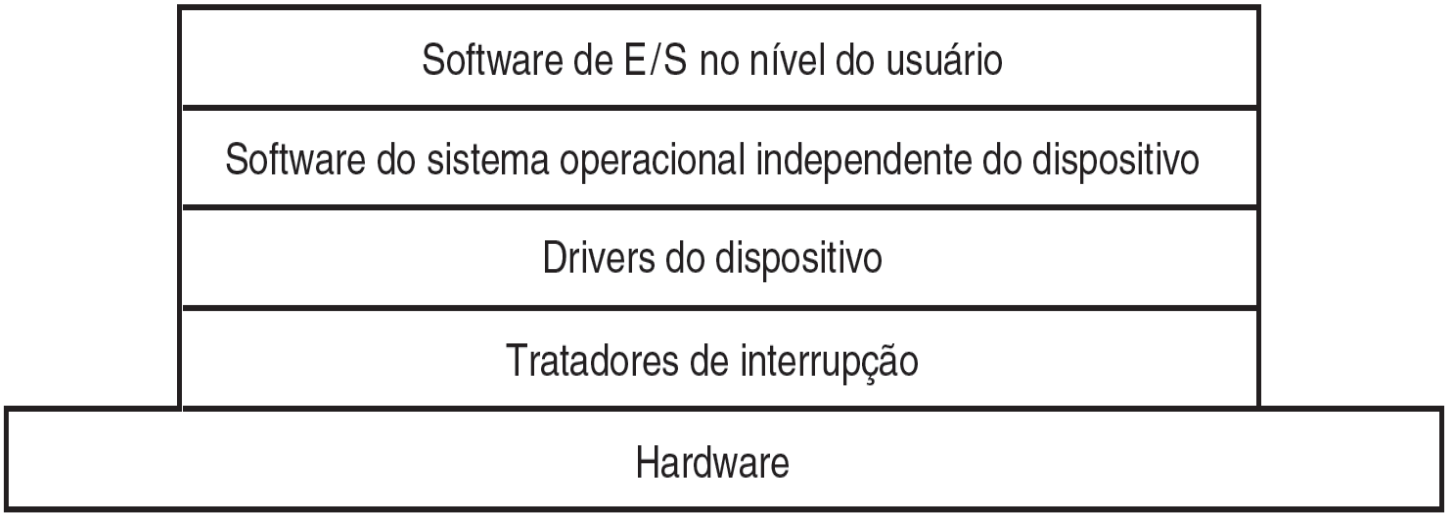


Figura 5.10 Camadas do software de E/S.

- O software de E/S é organizado em 4 camadas onde cada uma tem uma função bem definida para executar e uma interface bem definida para as camadas adjacentes.
- A funcionalidade e as interfaces diferem de sistema para sistema.

Tratadores de Interrupção (1)

- Tratadores de interrupção são parte do SO, usualmente bem escondidos
  - Tendo o driver iniciado uma operação de E/S o mesmo é bloqueado até que a operação se complete e uma interrupção ocorra (down, wait, receive)

- Ocorrendo a Interrupção a rotina de interrupção faz sua tarefa
  - Em seguida libera o driver que iniciou a operação de E/S (up, signal, send)
  - Processar uma interrupção não é apenas interceptar uma interrupção, sinalizar o driver e executar um IRET
  - Existe trabalho adicional realizado pelo SO

Tratadores de Interrupção (2)

- Passos que devem ser realizados em software depois da interrupção ser concluída:
  - Salvar regs ainda não salvos pelo hardware da interrupção.
  - Estabelece um contexto para rotina de tratamento da interrupção(configuração de TLB, MMU, tab. Páginas).
  - Estabelece uma pilha para rotina de tratamento da interrupção.
  - Sinaliza o controlador de interrupções. Senão existe um, reabilita interrupções.
  - Copiar registradores de onde foram salvos (pilha) para a tabela de processos.
  - Executa rotina de tratamento de interrupção. Ela extrai informações dos registradores do controlador do dispositivo que está interrompendo.
  - Escolhe próximo processo a executar. Se a interrupção deixou pronto algum processo de alta prioridade, este pode ser escolhido.
  - Estabelece contexto da MMU para próximo processo a executar. Algum ajuste na TLB também pode ser necessário.
  - Carrega registradores do novo processo, incluíndo PSW.
  - Inicializa a execução do novo processo.

Drivers de Dispositivos (1)

Tipos: Bloco / Caracter

Interface padrão

Kernel / usuário

Estáticos/dinâmicos

Reentrantes

Interação com o SO

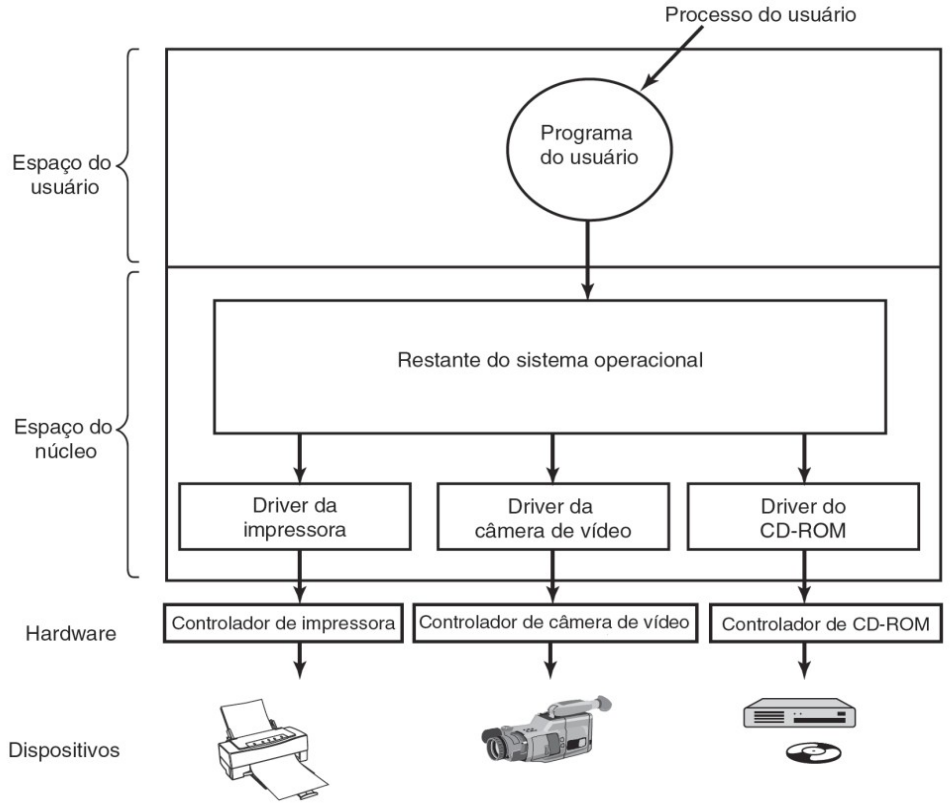


Figura 5.11 Posicionamento lógico dos drivers de dispositivos. Na verdade, toda comunicação entre os drivers e os controladores passa pelo barramento.

- Cada dispositivo de E/S precisa de algum código específico do dispositivo para controla-lo (driver do dispositivo)
- Código em geral escrito pelo fabricante
- Cada driver trata tipo ou classe de dispositivo

Software Independente de dispositivo (1)

Uniformizar interfaces para os drivers de dispositivos
Armazenar no buffer
Reportar erros
Alocar e liberar dispositivos dedicados
Providenciar um tamanho de bloco independente de dispositivo

Tabela 5.2 Funções do software de E/S independente de dispositivo.

Partes do software de E/S são independentes de dispositivo. As funções básicas de um software independente de dispositivo são executar as funções de E/S comuns para todos os dispositivos e fornecer uma interface uniforme para o software no nível de usuário.

Software Independente de dispositivo (2)

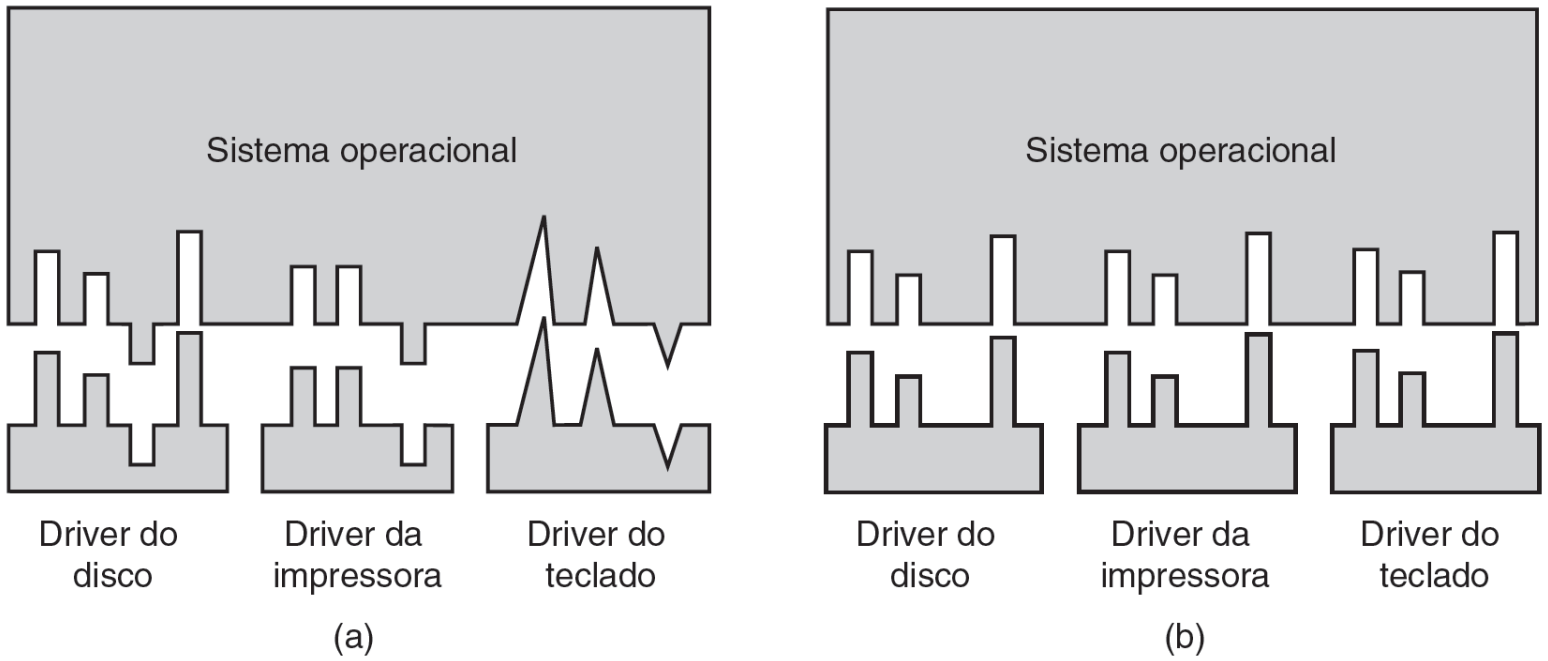


Figura 5.12 (a) Sem uma interface-padrão para o driver. (b) Com uma interface-padrão para o driver.

Uma questão importante em um sistema operacional é como fazer todos os dispositivos de E/S e drivers parecerem mais ou menos os mesmos.

(a) Com interfaces diferentes entre drivers e o restante do SO para cada driver.

- Funções diferentes para cada driver

(b) Com uma interface padrão do driver

- Cada classe (disco) define um conjunto de funções
- Nomeação dos dispositivos Ex. /dev/disk0

A tabela de dispositivos de caracteres

Dispositivo	Open	Close	Read	Write	Ioctl	Outros
Null	null	null	null	null	null	...
Memória	null	null	mem_read	mem_write	null	...
Teclado	k_open	k_close	k_read	error	k_ioctl	...
Terminal	tty_open	tty_close	tty_read	tty_write	tty_ioctl	...
Impressora	lp_open	lp_close	error	lp_write	lp_ioctl	...

Tabela 10.7 Algumas operações de arquivos para dispositivos de caracteres típicos.

Software Independente de dispositivo (3)

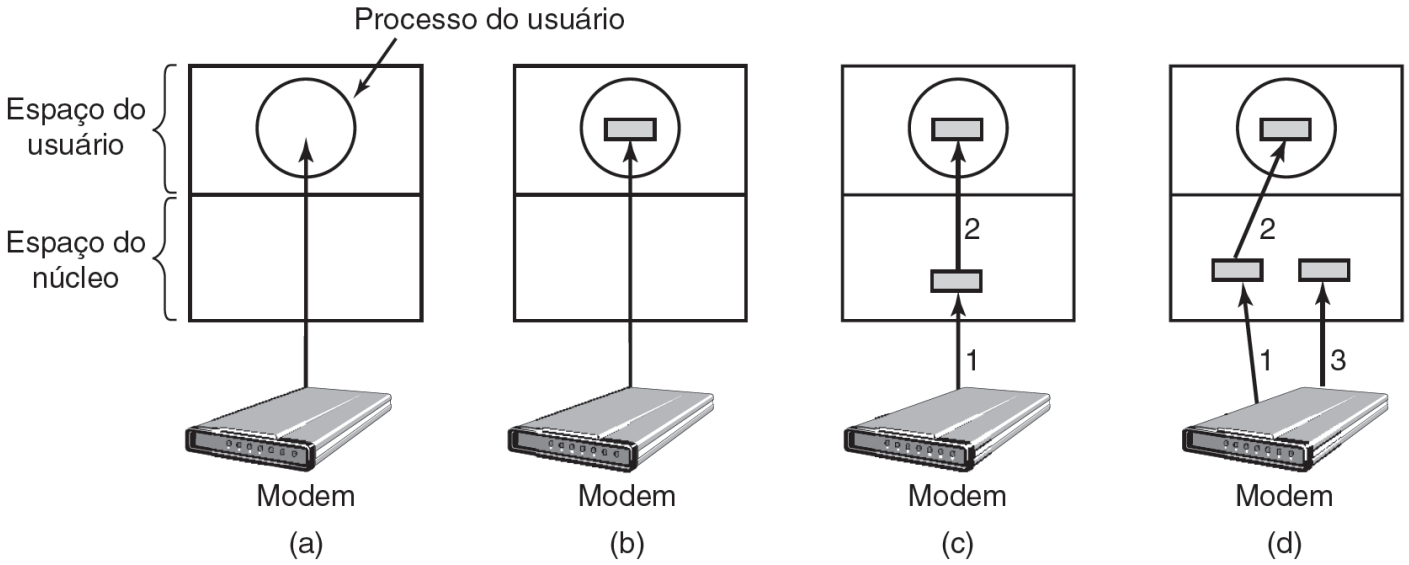


Figura 5.13 (a) Entrada não enviada para buffer. (b) Utilização de buffer no espaço do usuário. (c) Utilização de buffer no núcleo, seguido da cópia para o espaço do usuário. (d) Utilização de buffer duplicado no núcleo.

A utilização de buffer é uma questão importante para os dispositivos tanto de bloco como de caractere. Exemplo: modem.

(a) chamada read bloqueia e qdo o caracter chega é passado para o usuário

(b) Buffer no espaço de usuário – se buffer é paginado?

(c) Buffer no kernel seguido de cópia para o usuário – caracteres chegam durante paginação?

(d) Duplo buffer no kernel – alternativa buffer circular

Importancia de buffer na saída? (b) – write n caracteres

Software Independente de dispositivo (4)

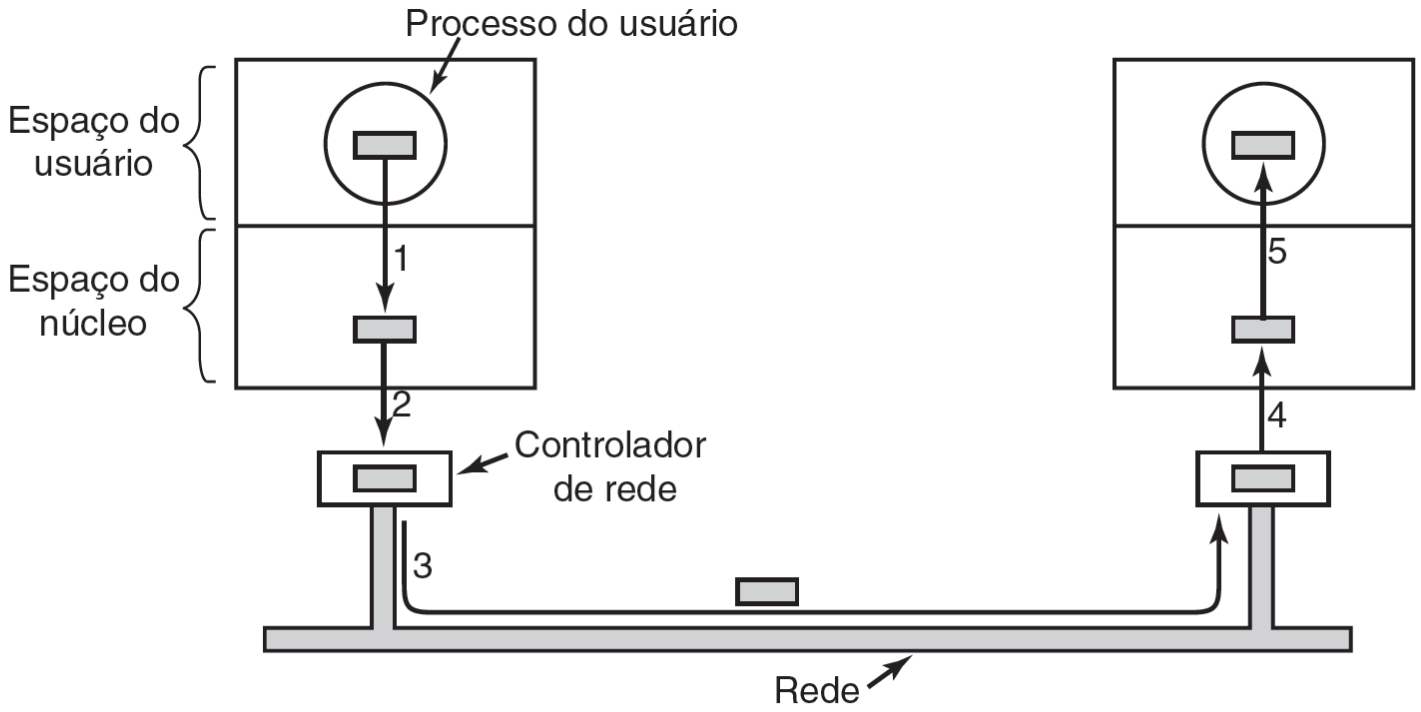


Figura 5.14 O trânsito na rede pode envolver muitas cópias de um pacote.

Utilização de buffer pode envolver muitas cópias e o desempenho pode cair

Software de E/S no espaço do usuário

- Bibliotecas de E/S - write(fd,buffer,nbytes), printf(“....”), scanf, ..
- sppoling (daemon) – gerencia dispositivos dedicados

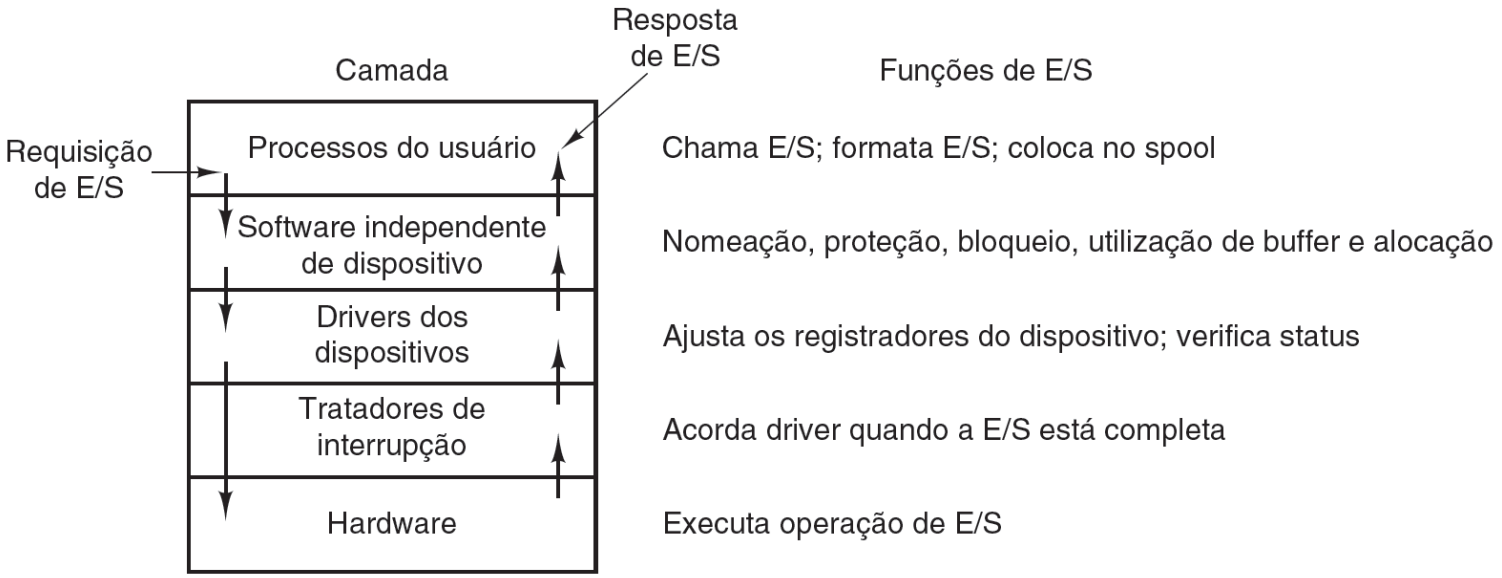


Figura 5.15 Camadas do sistema de E/S e as principais funções de cada camada.

UNIX SVR4 E/S

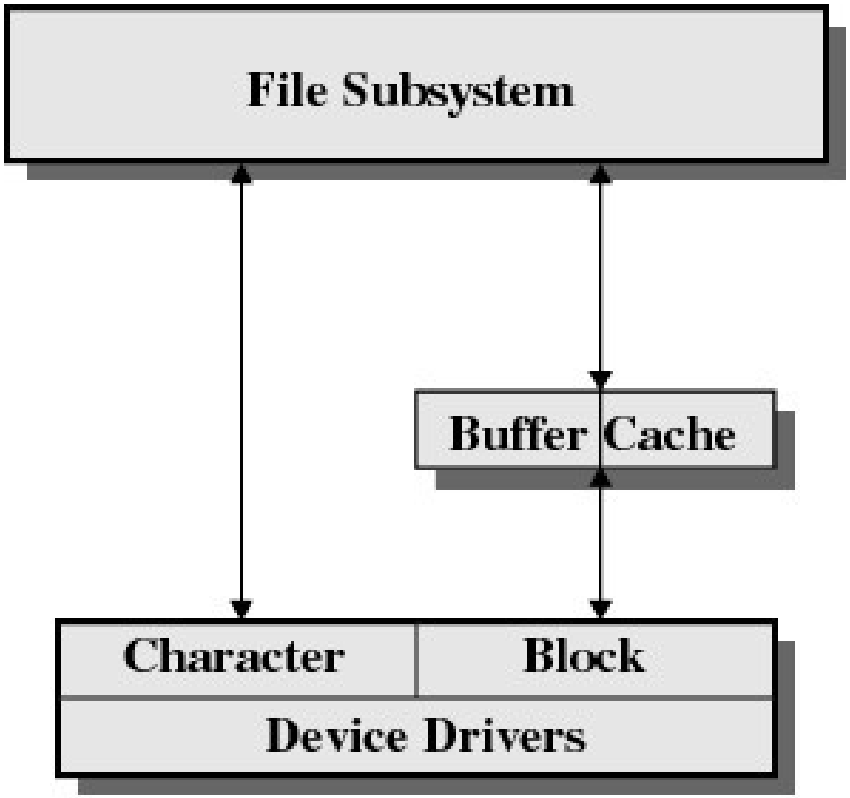


Figure 11.14 UNIX I/O Structure

Implementação de entrada/saída em Linux

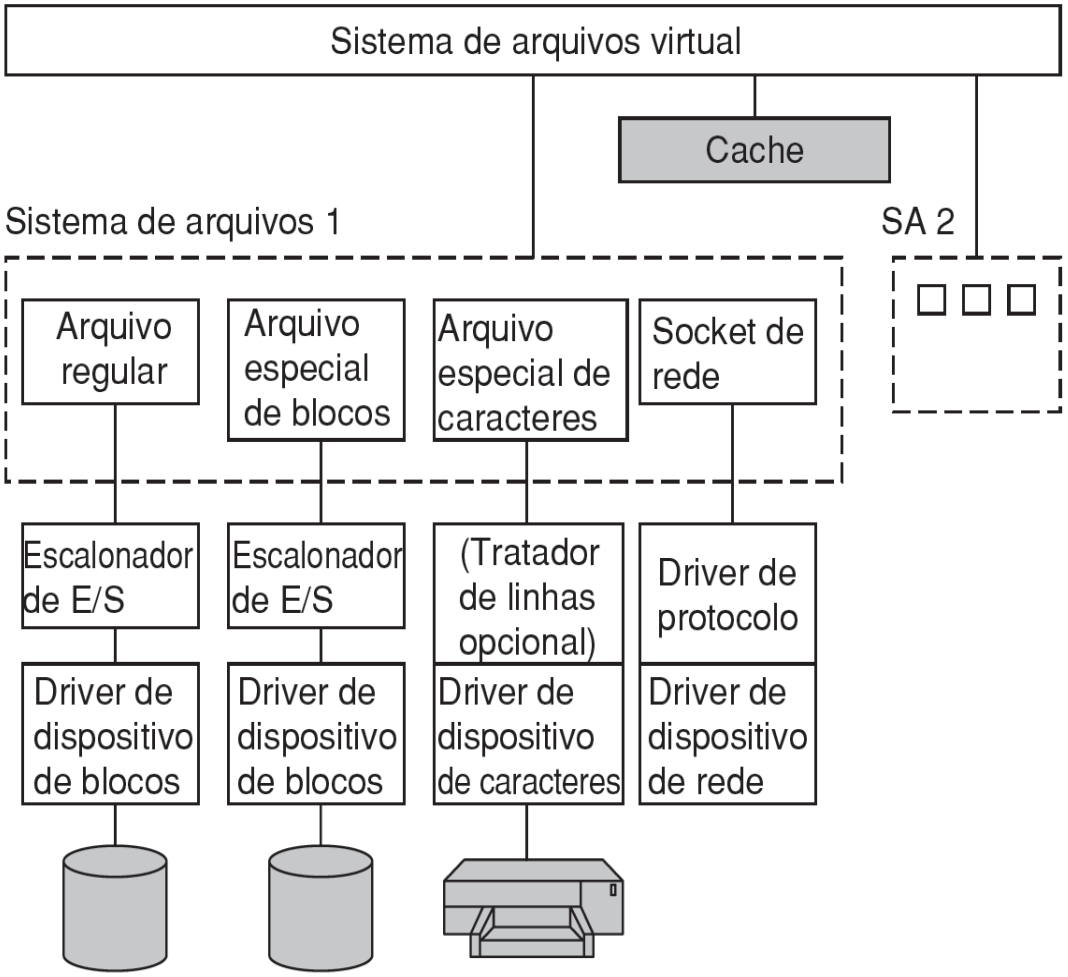
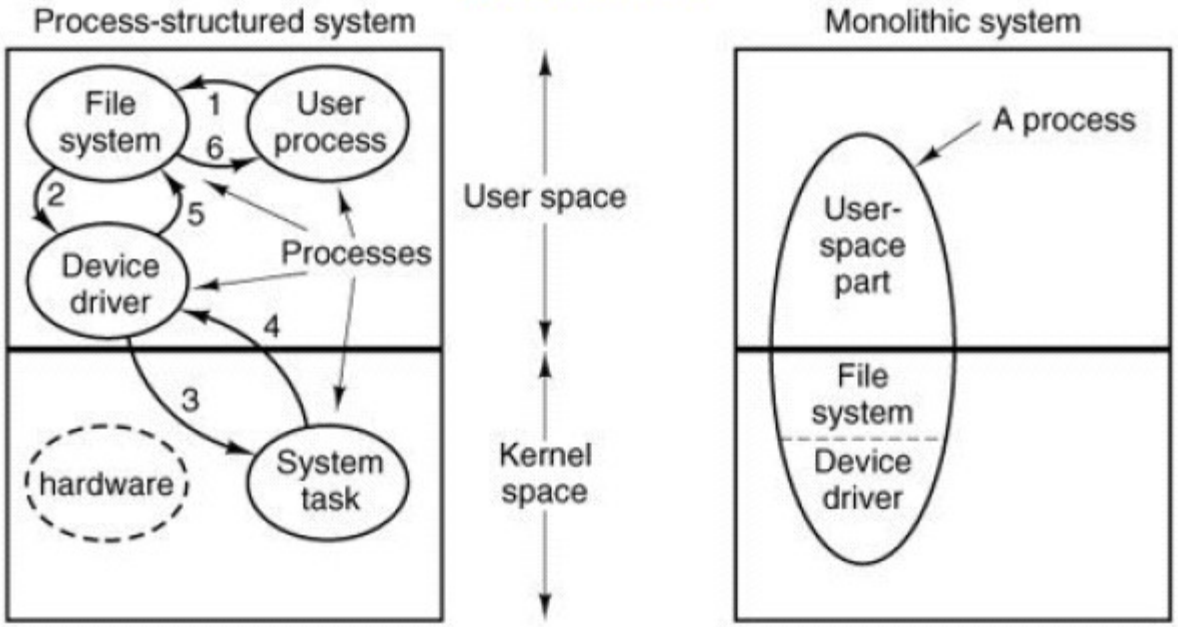


Figura 10.15 O sistema de E/S do Linux mostrando um sistema de arquivos em detalhes.

Figure 3-16. Two ways of structuring user-system communication.

[View full size image]



1–6 are request and reply messages between four independent processes.

(a)

The user-space part calls the kernel-space part by trapping. The file system calls the device driver as a procedure. The entire operating system is part of each process

(b)

Figure 3-18. Outline of the main procedure of an I/O device driver.

```
message mess;                                   /* message buffer*/

void io_driver() {
  initialize();                                 /* only done once, during system init.*/
  while (TRUE)  {
        receive(ANY, &mess);                    /* wait for a request for work*/
        caller = mess.source;                   /* process from whom message came*/
        switch(mess.type) {
            case READ:      rcode = dev_read(&mess); break;
            case WRITE:     rcode = dev_write(&mess); break;
            /* Other cases go here, including OPEN, CLOSE, and IOCTL*/
             default:       rcode = ERROR;
        }
        mess.type = DRIVER_REPLY;
        mess.status = rcode;                    /* result code*/
        send(caller,&mess);                     /* send reply message back to caller*/
  }
}
```

Figure 2-46. (a) Worst case for reading a block requires eleven messages. (b) Best case for reading a block requires four messages.