Sistemas Operacionais: Conceitos e Mecanismos

VII - Gerência de Entrada/Saída

Prof. Carlos Alberto Maziero
DInf UFPR
http://www.inf.ufpr.br/maziero

4 de agosto de 2017

Este texto está licenciado sob a Licença *Attribution-NonCommercial-ShareAlike 3.0 Unported* da *Creative Commons* (CC). Em resumo, você deve creditar a obra da forma especificada pelo autor ou licenciante (mas não de maneira que sugira que estes concedem qualquer aval a você ou ao seu uso da obra). Você não pode usar esta obra para fins comerciais. Se você alterar, transformar ou criar com base nesta obra, você poderá distribuir a obra resultante apenas sob a mesma licença, ou sob uma licença similar à presente. Para ver uma cópia desta licença, visite `http://creativecommons.org/licenses/by-nc-sa/3.0/`.

Este texto foi produzido usando exclusivamente software livre: Sistema Operacional *GNU/Linux* (distribuições *Fedora* e *Ubuntu*), compilador de texto LaTeX 2$_\varepsilon$, gerenciador de referências *BibTeX*, editor gráfico *Inkscape*, criadores de gráficos *GNUPlot* e *GraphViz* e processador PS/PDF *GhostScript*, entre outros.

Sumário

Resumo

Este conteúdo está em elaboração. Ainda há muito o que escrever aqui...

1 Introdução

Um computador é constituído basicamente de um ou mais processadores, memória RAM e **dispositivos de entrada e saída**, também chamados de **periféricos**. Os dispositivos de entrada/saída permitem a interação do computador com o mundo exterior de várias formas, como por exemplo:

- interação com os usuários através de *mouse*, teclado, tela gráfica, tela de toque e *joystick*;
- escrita e leitura de dados em discos rígidos, SSDs, CD-ROMs, DVD-ROMs e *pen-drives*;
- impressão de informações através de impressoras e plotadoras;
- captura e reprodução de áudio e vídeo, como câmeras, microfones e alto-falantes;
- comunicação com outros computadores, através de redes LAN, WLAN, *Bluetooth* e de telefonia celular.

Esses exemplos são típicos de computadores pessoais e de computadores menores, como os *smartphones*. Já em ambientes industriais, é comum encontrar dispositivos de entrada/saída específicos para a monitoração e controle de máquinas e processos de produção, como tornos de comando numérico, braços robotizados e processos químicos. Por sua vez, o computador embarcado em um carro conta com dispositivos de entrada para coletar dados do combustível e do funcionamento do motor e dispositivos de saída para controlar a injeção eletrônica e a tração dos pneus, por exemplo. É bastante óbvio que um computador não tem muita utilidade sem dispositivos periféricos, pois o objetivo básico da imensa maioria dos computadores é receber dados, processá-los e devolver resultados aos seus usuários, sejam eles seres humanos, outros computadores ou processos físicos/químicos externos.

Os primeiros sistemas de computação, construídos nos anos 1940, eram destinados a cálculos matemáticos e por isso possuíam dispositivos de entrada/saída rudimentares, que apenas permitiam carregar/descarregar programas e dados diretamente na memória principal. Em seguida surgiram os terminais compostos de teclado e monitor de texto, para facilitar a leitura e escrita de dados, e os discos rígidos, como meio de armazenamento persistente de dados e programas. Hoje, dispositivos de entrada/saída dos mais diversos tipos podem estar conectados a um computador. A grande diversidade de dispositivos periféricos é um dos maiores desafios presentes na construção e manutenção de um sistema operacional, pois cada um deles tem especificidades e exige mecanismos de acesso específicos.

Este capítulo apresenta uma visão geral da operação dos dispositivos de entrada/-saída sob a ótica do sistema operacional. Inicialmente serão discutidas as principais

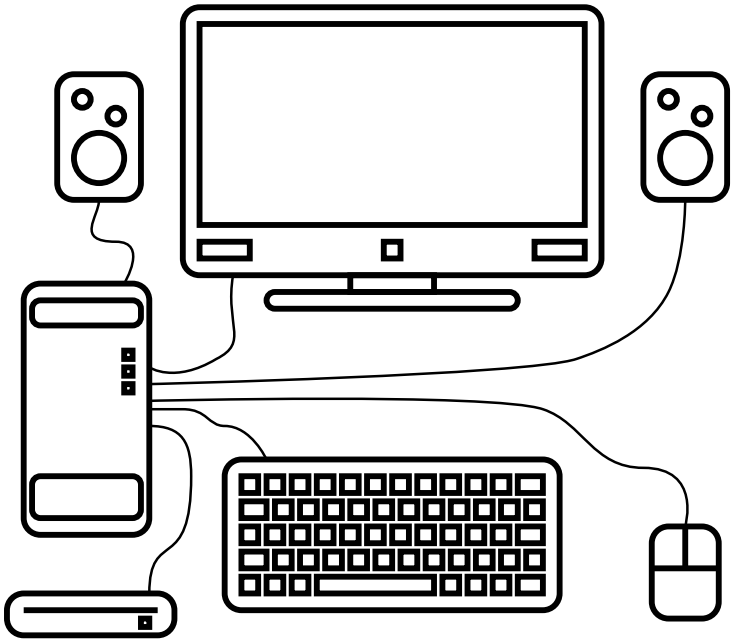

Figura 1: Dispositivos de entrada/saída.

características dos dispositivos de entrada/saída usualmente presentes nos computadores convencionais para a interação com o usuário, armazenamento de dados e comunicação. A seguir, a arquitetura de hardware e software responsável pela operação dos dispositivos de entrada/saída será detalhada. Na sequência, as principais estratégias de interação entre o software e o hardware serão apresentadas. Os *drivers*, componentes de software responsáveis pela interação do sistema operacional com o hardware de entrada/saída, terão sua estrutura e princípio de funcionamento abordados em seguida. Por fim, alguns subsistemas de entrada/saída específicos, considerados os mais relevantes nos computadores de uso geral atualmente em uso, serão estudados com maior profundidade.

2 Dispositivos de entrada/saída

Um dispositivo de entrada/saída realiza a interação entre os sistemas internos de um computador (processadores e memória) e o mundo exterior.

2.1 Componentes de um dispositivo

Conceitualmente, a entrada de dados em um computador inicia com um **sensor** capaz de converter uma informação externa (física ou química) em um sinal elétrico analógico. Como exemplos de sensores temos o microfone, as chaves internas das teclas de um teclado ou o foto-diodo de um leitor de DVDs. O sinal elétrico analógico fornecido pelo sensor é então aplicado a um **conversor analógico-digital** (CAD), que o transforma em informação digital (sequências de bits). Essa informação digital é

Tabela 1: Velocidades típicas de alguns dispositivos de entrada/saída.

Dispositivo	**velocidade**
Teclado	10 B/s
Mouse ótico	100 B/s
Interface infravermelho (IrDA-SIR)	14 KB/s
Interface paralela padrão	125 KB/s
Interface de áudio digital S/PDIF	384 KB/s
Interface de rede *Fast Ethernet*	11.6 MB/s
Chave ou disco USB 2.0	60 MB/s
Interface de rede *Gigabit Ethernet*	116 MB/s
Disco rígido SATA 2	300 MB/s
Interface gráfica *high-end*	4.2 GB/s

armazenada em um *buffer* que pode ser acessado pelo processador através de um **controlador de entrada**.

Uma saída de dados inicia com o envio de dados do processador a um **controlador de saída**, através do barramento. Os dados enviados pelo processador são armazenados em um *buffer* interno do controlador e a seguir convertidos em um sinal elétrico analógico, através de um **conversor digital-analógico** (CDA). Esse sinal será aplicado a um **atuador**[1] que irá convertê-lo em efeitos físicos perceptíveis ao usuário. Exemplos simples de atuadores são a cabeça de impressão e os motores de uma impressora, um alto-falante, uma tela gráfica, etc.

Vários dispositivos combinam funcionalidades tanto de entrada quanto de saída, como os discos rígidos: o processador pode ler dados gravados no disco rígido (entrada), mas para isso precisa ativar o motor que faz girar o disco e posicionar adequadamente a cabeça de leitura (saída). O mesmo ocorre com uma placa de áudio de um PC convencional, que pode tanto capturar quanto reproduzir sons. A Figura 2 mostra a estrutura básica de captura e reprodução de áudio em um computador pessoal, que é um bom exemplo de dispositivo de entrada/saída.

Como existem muitas possibilidades de interação do computador com o mundo exterior, também existem muitos tipos de dispositivos de entrada/saída, com características diversas de velocidade de transferência, forma de transferência dos dados e método de acesso. A **velocidade de transferência de dados** de um dispositivo pode ir de alguns bytes por segundo, no caso de dispositivos simples como teclados e *mouses*, a gigabytes por segundo, para algumas placas de interface gráfica ou de acesso a discos de alto desempenho. A Tabela 1 traz alguns exemplos de dispositivos de entrada/saída com suas velocidades típicas de transferência de dados.

[1]Sensores e atuadores são denominados genericamente *dispositivos transdutores*, pois transformam energia externa (como luz, calor, som ou movimento) em sinais elétricos, ou vice-versa.

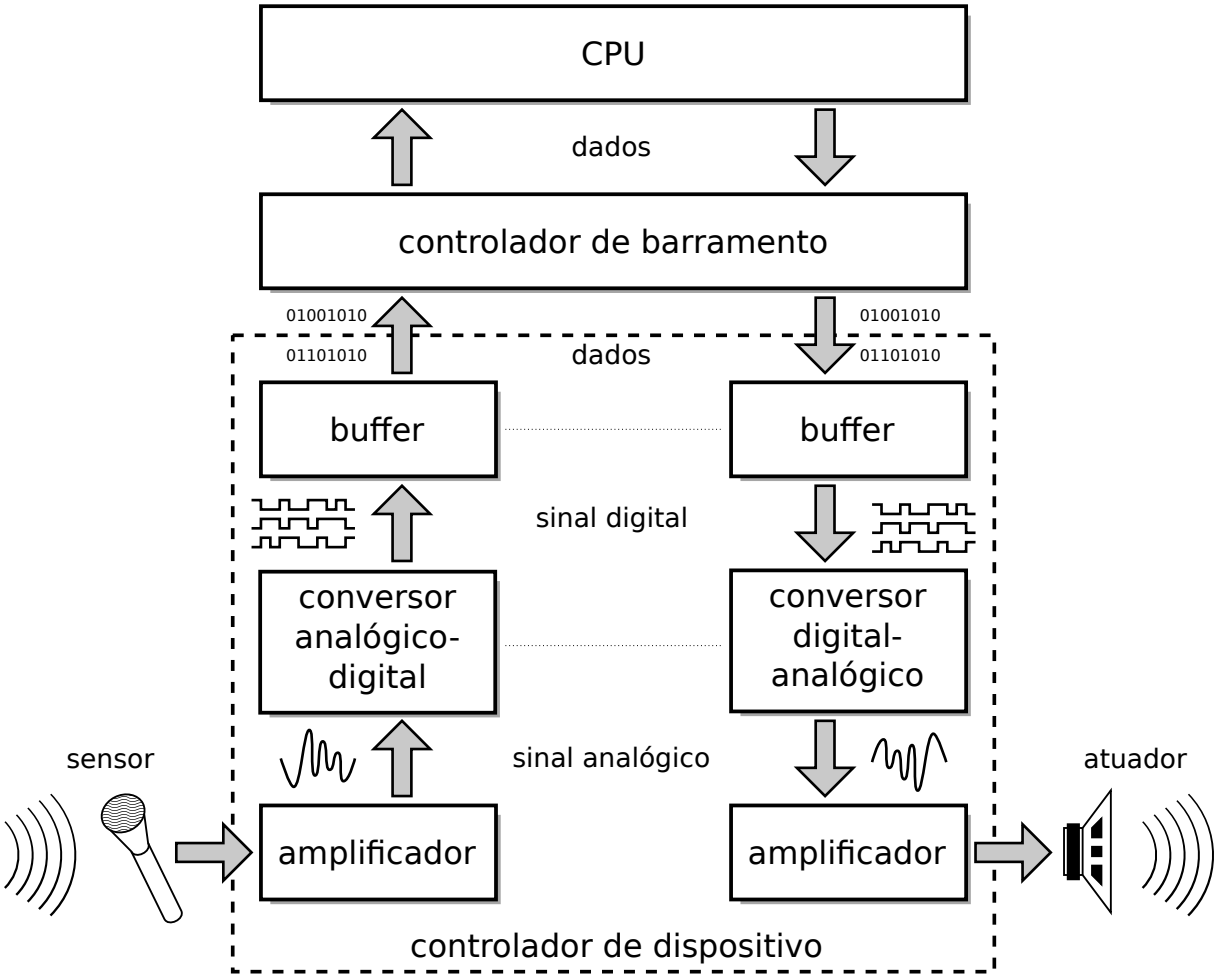

Figura 2: Estrutura básica da entrada e saída de áudio.

2.2 Barramentos

Historicamente, o acoplamento dos dispositivos de entrada/saída ao computador é feito através de barramentos, seguindo o padrão estabelecido pela arquitetura de *Von Neumann*. Enquanto o barramento dos primeiros sistemas era um simples agrupamento de fios, os barramentos dos sistemas atuais são estruturas de hardware bastante complexas, com circuitos específicos para seu controle. Além disso, a diversidade de velocidades e volumes de dados suportados pelos dispositivos fez com que o barramento único dos primeiros sistemas fosse gradativamente estruturado em um conjunto de barramentos com características distintas de velocidade e largura de dados.

O controle dos barramentos em um sistema *desktop* moderno está a cargo de dois controladores de hardware que fazem parte do *chipset*[2] da placa-mãe: a *north bridge* e a

[2]O *chipset* de um computador é um conjunto de controladores e circuitos auxiliares de hardware integrados à placa-mãe, que proveem serviços fundamentais ao funcionamento do computador, como o controle dos barramentos, acesso à BIOS, controle de interrupções, temporizadores programáveis e controladores *on-board* para alguns periféricos, como discos rígidos, portas paralelas e seriais e entrada/saída de áudio.

south bridge. A *north bridge*, diretamente conectada ao processador, é responsável pelo acesso à memória RAM e aos dispositivos de alta velocidade, através de barramentos dedicados como AGP (*Accelerated Graphics Port*) e PCI-Express (*Peripheral Component Interconnect*).

Por outro lado, a *south bridge* é o controlador responsável pelos barramentos e portas de baixa ou média velocidade do computador, como as portas seriais e paralelas, e pelos barramentos dedicados como o PCI padrão, o USB e o SATA. Além disso, a *south bridge* costuma integrar outros componentes importantes do computador, como controladores de áudio e rede *on-board*, controlador de interrupções, controlador DMA (*Direct Memory Access*), relógio de tempo real (responsável pelas interrupções de tempo usadas pelo escalonador de processos), controle de energia e acesso à memória BIOS. O processador se comunica com a *south bridge* indiretamente, através da *north bridge*.

A Figura 3 traz uma visão da arquitetura típica de um computador pessoal moderno. A estrutura detalhada e o funcionamento dos barramentos e seus respectivos controladores estão fora do escopo deste texto; informações mais detalhadas podem ser encontradas em [Patterson and Henessy, 2005].

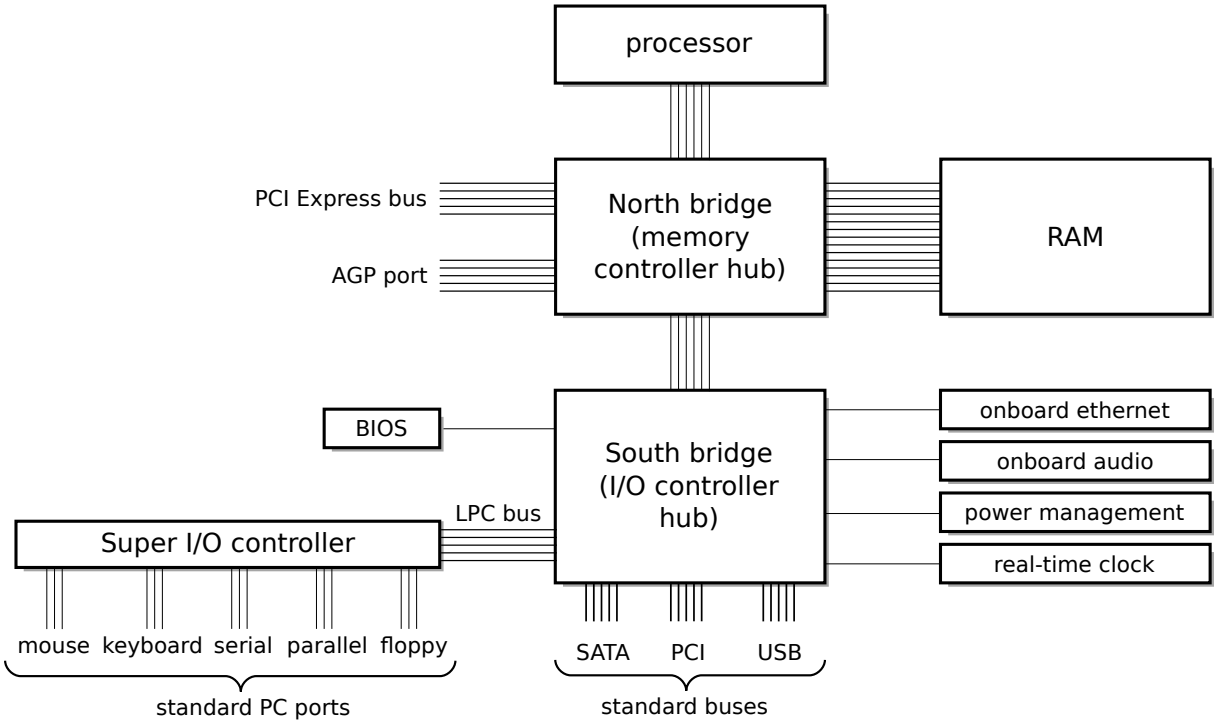

Figura 3: Arquitetura típica de um PC atual.

2.3 Interface de acesso

Para o sistema operacional, o aspecto mais relevante de um dispositivo de entrada/saída é sua interface de acesso, ou seja, a abordagem a ser usada para acessar o dispositivo, configurá-lo e enviar dados para ele (ou receber dados dele). Normalmente, cada dispositivo oferece um conjunto de registradores acessíveis através do barramento,

também denominados **portas de entrada/saída**, que são usados para a comunicação entre o dispositivo e o processador. As portas oferecidas para acesso a cada dispositivo de entrada/saída podem ser divididas nos seguintes grupos (conforme ilustrado na Figura 4):

- Portas de entrada (*data-in ports*): usadas pelo processador para receber dados provindos do dispositivo; são escritas pelo dispositivo e lidas pelo processador;
- Portas de saída (*data-out ports*): usadas pelo processador para enviar dados ao dispositivo; essas portas são escritas pelo processador e lidas pelo dispositivo;
- Portas de status (*status ports*): usadas pelo processador para consultar o estado interno do dispositivo ou verificar se uma operação solicitada ocorreu sem erro; essas portas são escritas pelo dispositivo e lidas pelo processador;
- Portas de controle (*control ports*): usadas pelo processador para enviar comandos ao dispositivo ou modificar parâmetros de sua configuração; essas portas são escritas pelo processador e lidas pelo dispositivo.

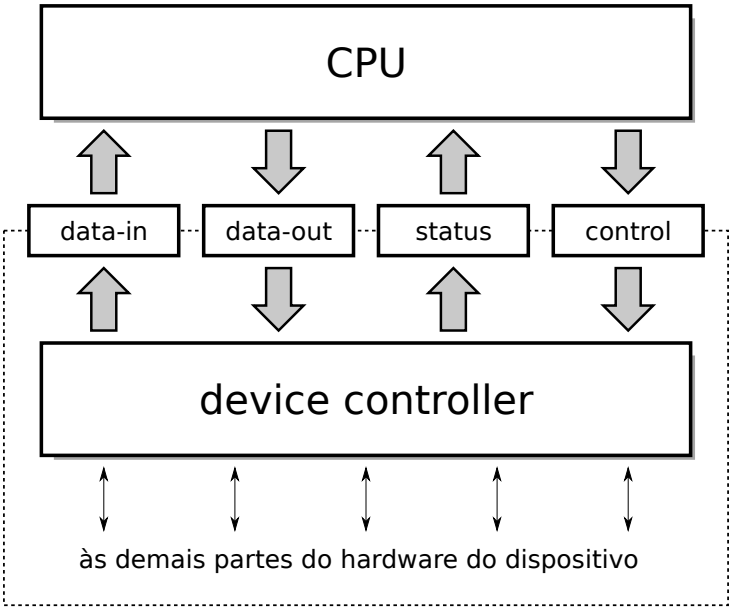

Figura 4: Portas de interface de um dispositivo de entrada/saída.

O número exato de portas e o significado específico de cada uma dependem do tipo de dispositivo considerado. Um exemplo simples de interface de acesso a dispositivo é a interface paralela, geralmente usada para acessar impressoras mais antigas. As portas de uma interface paralela operando no modo padrão (SPP - *Standard Parallel Port*) estão descritas a seguir [Patterson and Henessy, 2005]:

- P_0 (*data port*): porta de saída, usada para enviar bytes à impressora; pode ser usada também como porta de entrada, se a interface estiver operando em modo bidirecional;

- P_1 (*status port*): porta de status, permite ao processador consultar vários indicadores de status da interface paralela ou do dispositivo ligado a ela. O significado de cada um de seus 8 bits é:

  0. reservado;
  1. reservado;
  2. $\overline{nIRQ}$: se 0, indica que o controlador gerou uma interrupção (Seção 2.5);
  3. *error*: há um erro interno na impressora;
  4. *select*: a impressora está pronta (*online*);
  5. *paper_out*: falta papel na impressora;
  6. $\overline{ack}$: se 0, indica que um dado foi recebido (gera um pulso em 0 com duração de ao menos $1\mu s$);
  7. *busy*: indica que o controlador está ocupado processando um comando.

- P_2 (*control port*): porta de controle, usada para configurar a interface paralela e para solicitar operações de saída (ou entrada) de dados através da mesma. Seus 8 bits têm o seguinte significado:

  0. $\overline{strobe}$: informa a interface que há um dado em P_0 (deve ser gerado um pulso em 0, com duração de ao menos $0,5\mu s$);
  1. *auto_lf*: a impressora deve inserir um *line feed* a cada *carriage return* recebido;
  2. *reset*: a impressora deve ser reiniciada;
  3. *select*: a impressora está selecionada para uso;
  4. *enable_IRQ*: permite ao controlador gerar interrupções (Seção 2.5);
  5. *bidirectional*: informa que a interface será usada para entrada e para saída de dados;
  6. reservado;
  7. reservado.

- P_3 a P_7: estas portas são usadas nos modos estendidos de operação da interface paralela, como EPP (*Enhanced Paralel Port*) e ECP (*Extended Capabilities Port*).

O algoritmo básico implementado pelo hardware interno do controlador da interface paralela para coordenar suas interações com o processador está ilustrado no fluxograma da Figura 5. Considera-se que os valores iniciais dos flags de status da porta P_1 são $nIRQ = 1$, $error = 0$, $select = 1$, $paper\_out = 0$, $ack = 1$ e $busy = 0$.

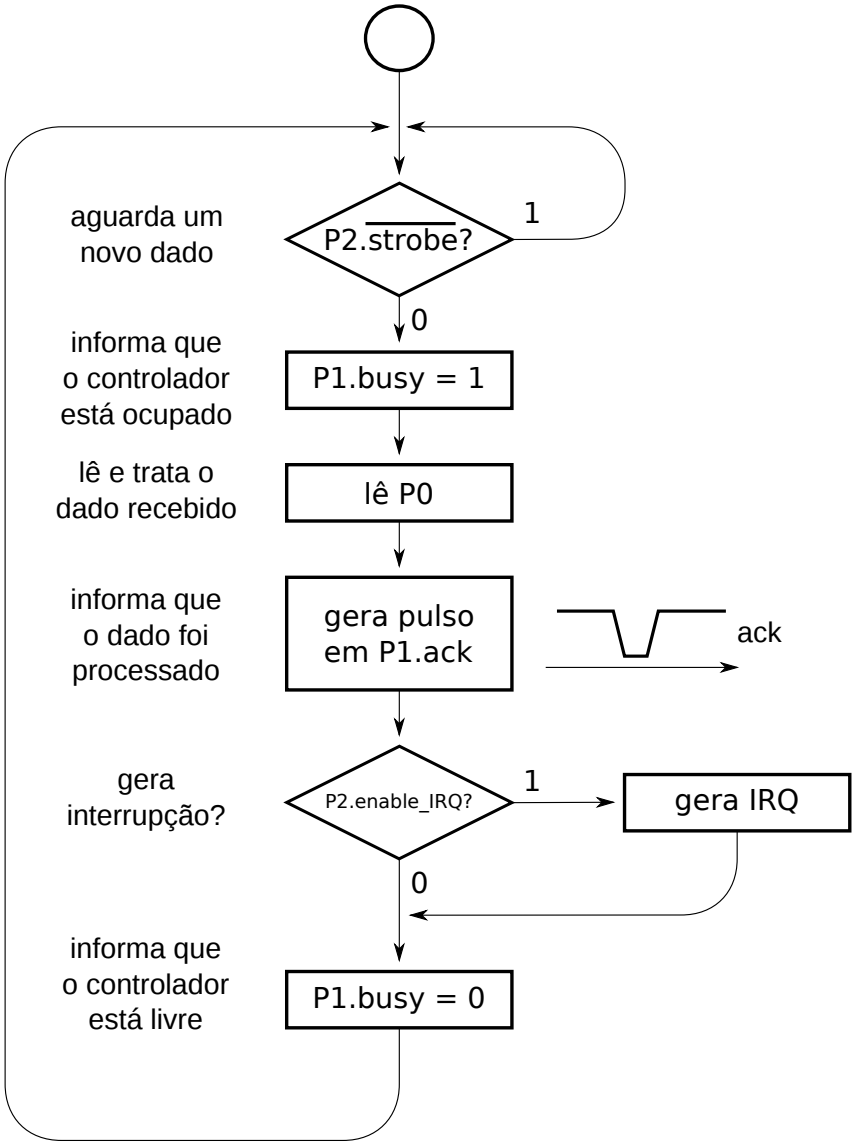

Figura 5: Comportamento básico do controlador da porta paralela.

2.4 Endereçamento

A forma de acesso aos registradores que compõem a interface de um dispositivo varia de acordo com a arquitetura do computador. Alguns sistemas utilizam **entrada/saída mapeada em portas** (*port-mapped I/O*), onde as portas que compõem a interface são acessadas pelo processador através de instruções específicas para operações de entrada/saída. Por exemplo, os processadores da família Intel usam a instrução "`IN` *reg port*" para ler o valor presente na porta "*port*" do dispositivo e depositá-lo no registrador "*reg*" do processador, enquanto a instrução "`OUT` *port reg*" é usada para escrever na porta "*port*" o valor contido no registrador "*reg*".

Na entrada/saída mapeada em portas, é definido um **espaço de endereços de entrada/saída** (*I/O address space*) separado da memória principal e normalmente compreendido entre 0 e 64K. Para distinguir entre endereços de memória e de portas de entrada/saída,

o barramento de controle do processador possui uma linha $IO/\overline{M}$, que indica se o endereço presente no barramento de endereços se refere a uma posição de memória (se $IO/\overline{M} = 0$) ou a uma porta de entrada/saída (se $IO/\overline{M} = 1$). A Tabela 2 apresenta os endereços típicos de algumas portas de entrada/saída de dispositivos em computadores pessoais que seguem o padrão IBM-PC.

Dispositivo	**Endereços das portas**
teclado e mouse PS/2	0060h e 0064h
barramento IDE primário	0170h a 0177h
barramento IDE secundário	01F0h a 01F7h
relógio de tempo real	0070h e 0071h
porta serial COM1	02F8h a 02FFh
porta serial COM2	03F8h a 03FFh
porta paralela LPT1	0378h a 037Fh

Tabela 2: Endereços de portas de E/S de alguns dispositivos.

Uma outra forma de acesso aos dispositivos de entrada/saída, usada frequentemente em interfaces gráficas e de rede, é a **entrada/saída mapeada em memória** (*memory-mapped I/O*). Nesta abordagem, uma parte não ocupada do espaço de endereços de memória é reservado para mapear as portas de acesso aos dispositivos. Dessa forma, as portas são vistas como se fossem parte da memória principal e podem ser lidas e escritas através das mesmas instruções usadas para acessar o restante da memória, sem a necessidade de instruções especiais como `IN` e `OUT`. Algumas arquiteturas de computadores, como é caso do IBM-PC padrão, usam uma abordagem híbrida para certos dispositivos como interfaces de rede e de áudio: as portas de controle e status são mapeadas no espaço de endereços de entrada/saída, sendo acessadas através de instruções específicas, enquanto as portas de entrada e saída de dados são mapeadas em memória (normalmente na faixa de endereços entre 640 KB e 1MB) [Patterson and Henessy, 2005].

Finalmente, uma abordagem mais sofisticada para o controle de dispositivos de entrada/saída é o uso de um hardware independente, com processador dedicado, que comunica com o processador principal através de algum tipo de barramento. Em sistemas de grande porte (*mainframes*) essa abordagem é denominada **canais de entrada/saída** (*IO channels*); em computadores pessoais, essa abordagem costuma ser usada em interfaces para vídeo ou áudio de alto desempenho, como é o caso das placas gráficas com aceleração, nas quais um processador gráfico (GPU – *Graphics Processing Unit*) realiza a parte mais pesada do processamento da saída de vídeo, como a renderização de imagens em 3 dimensões e texturas, deixando o processador principal livre para outras tarefas.

2.5 Interrupções

O acesso aos controladores de dispositivos através de seus registradores é conveniente para a comunicação no sentido *processador* → *controlador*, ou seja, para as interações iniciadas pelo processador. Entretanto, pode ser problemática no sentido *controlador* →

processador, caso o controlador precise informar algo ao processador de forma assíncrona, sem que o processador esteja esperando. Nesse caso, o controlador pode utilizar uma **requisição de interrupção** (IRQ - *Interrupt Request*) para notificar o processador sobre algum evento importante, como a conclusão de uma operação solicitada, a disponibilidade de um novo dado ou a ocorrência de algum problema no dispositivo.

As requisições de interrupção são sinais elétricos veiculados através do barramento de controle do computador. Cada interrupção está associada a um número inteiro, geralmente na faixa 0–31 ou 0–63, o que permite identificar o dispositivo que a solicitou. A Tabela 3 informa os números de interrupção associados a alguns dispositivos periféricos típicos.

Dispositivo	**Interrupção**
teclado	1
mouse PS/2	12
barramento IDE primário	14
barramento IDE secundário	15
relógio de tempo real	8
porta serial COM1	4
porta serial COM2	3
porta paralela LPT1	7

Tabela 3: Interrupções geradas por alguns dispositivos.

Ao receber uma requisição de interrupção, o processador suspende seu fluxo de instruções corrente e desvia a execução para um endereço pré-definido, onde se encontra uma **rotina de tratamento de interrupção** (*interrupt handler*). Essa rotina é responsável por tratar a interrupção, ou seja, executar as ações necessárias para identificar e atender o dispositivo que gerou a requisição. Ao final da rotina de tratamento da interrupção, o processador retoma o código que estava executando quando foi interrompido. A Figura 6 representa os principais passos associados ao tratamento de uma interrupção envolvendo o controlador de teclado, detalhados a seguir:

1. O processador está executando um programa qualquer;
2. O usuário pressiona uma tecla no teclado;
3. O controlador do teclado identifica a tecla pressionada, armazena seu código em um *buffer* interno e envia uma solicitação de interrupção (IRQ) ao processador;
4. O processador recebe a interrupção, salva na pilha seu estado atual (o conteúdo de seus registradores) e desvia sua execução para uma rotina de tratamento da interrupção;
5. Ao executar, essa rotina acessa os registradores do controlador de teclado para transferir o conteúdo de seu *buffer* para uma área de memória do núcleo. Depois disso, ela pode executar outras ações, como acordar algum processo ou *thread* que esteja esperando por entradas do teclado;

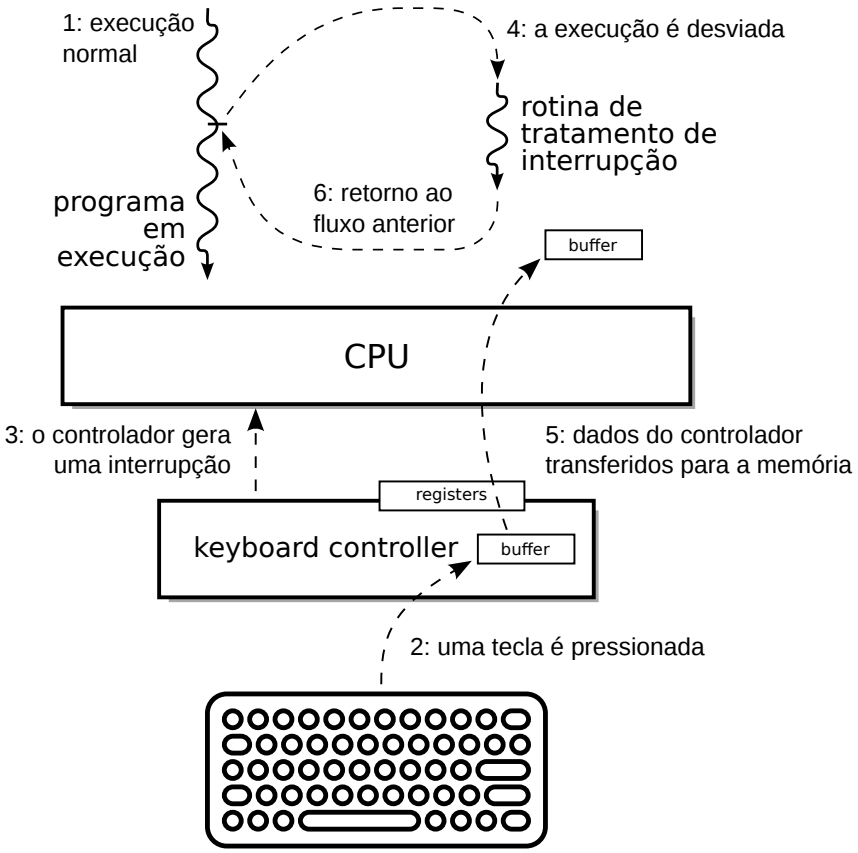

Figura 6: Roteiro típico de um tratamento de interrupção

6. Ao concluir a execução da rotina de tratamento da interrupção, o processador retorna à execução do fluxo de instruções que havia sido interrompido, usando a informação de estado salva no passo 4.

Essa sequência de ações ocorre a cada requisição de interrupção recebida pelo processador. Como cada interrupção corresponde a um evento ocorrido em um dispositivo periférico (chegada de um pacote de rede, movimento do mouse, conclusão de operação do disco, etc.), podem ocorrer centenas ou milhares de interrupções por segundo, dependendo da carga de trabalho e da configuração do sistema (número e natureza dos periféricos). Por isso, as rotinas de tratamento de interrupção devem realizar suas tarefas rapidamente, para não prejudicar o funcionamento do restante do sistema.

Como cada tipo de interrupção pode exigir um tipo de tratamento diferente (pois os dispositivos são diferentes), cada requisição de interrupção deve disparar uma rotina de tratamento específica. A maioria das arquiteturas atuais define um vetor de endereços de funções denominado *Vetor de Interrupções* (IV - *Interrupt Vector*); cada entrada desse vetor aponta para a rotina de tratamento da interrupção correspondente. Por exemplo, se a entrada 5 do vetor contém o valor `3C20h`, então a rotina de tratamento da IRQ 5 iniciará na posição `3C20h` da memória RAM. Dependendo do hardware, o vetor de interrupções pode residir em uma posição fixa da memória RAM, definida pelo

fabricante do processador, ou ter sua posição indicada pelo conteúdo de um registrador da CPU específico para esse fim.

As interrupções recebidas pelo processador têm como origem eventos externos a ele, ocorridos nos dispositivos periféricos e reportados por seus controladores. Entretanto, eventos gerados pelo próprio processador podem ocasionar o desvio da execução usando o mesmo mecanismo de interrupção: são as **exceções**. Eventos como instruções ilegais (inexistentes ou com operandos inválidos), divisão por zero ou outros erros de software disparam exceções internas no processador, que resultam na ativação de rotinas de tratamento de exceção registradas no vetor de interrupções. A Tabela 4 apresenta algumas exceções previstas pelo processador Intel Pentium (extraída de [Patterson and Henessy, 2005]).

Tabela 4: Algumas exceções do processador Pentium.

Exceção	**Descrição**
0	divide error
3	breakpoint
5	bound range exception
6	invalid opcode
9	coprocessor segment overrun
11	segment not present
12	stack fault
13	general protection
14	page fault
16	floating point error

Nas arquiteturas de hardware atuais, as interrupções geradas pelos dispositivos de entrada/saída não são transmitidas diretamente ao processador, mas a um **controlador de interrupções programável** (PIC - *Programmable Interrupt Controller*, ou APIC - *Advanced Programmable Interrupt Controller*), que faz parte do *chipset* do computador. As linhas de interrupção dos controladores de periféricos são conectadas aos pinos desse controlador de interrupções, enquanto suas saídas são conectadas às entradas de interrupção do processador.

O controlador de interrupções recebe as interrupções dos dispositivos e as encaminha ao processador em sequência, uma a uma. Ao receber uma interrupção, o processador deve acessar a interface do PIC para identificar a origem da interrupção e depois "reconhecê-la", ou seja, indicar ao PIC que aquela interrupção foi tratada e pode ser descartada pelo controlador. Interrupções já ocorridas mas ainda não reconhecidas pelo processador são chamadas de **interrupções pendentes**.

O PIC pode ser programado para bloquear ou ignorar algumas das interrupções recebidas, impedindo que cheguem ao processador. Além disso, ele permite definir prioridades entre as interrupções. A Figura 7 mostra a operação básica de um controlador de interrupções; essa representação é simplificada, pois as arquiteturas de computadores mais recentes podem contar com vários controladores de interrupções interligados.

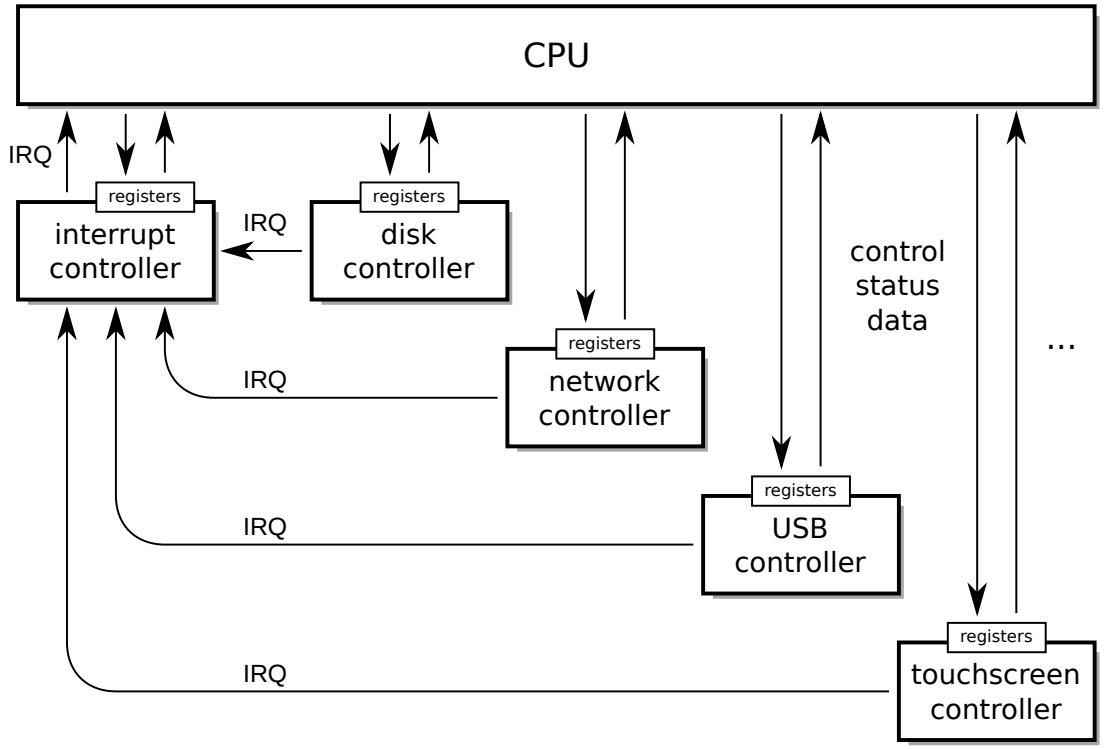

Figura 7: Uso de um controlador de interrupções.

O mecanismo de interrupção torna eficiente a interação do processador com os dispositivos periféricos. Se não existissem interrupções, o processador perderia muito tempo consultando todos os dispositivos do sistema para verificar se há eventos a serem tratados. Além disso, as interrupções permitem construir funções de entrada/saída assíncronas: o processador não precisa esperar a conclusão de cada operação solicitada, pois o dispositivo emitirá uma interrupção para "avisar" o processador quando a operação for concluída.

3 Software de entrada/saída

O sistema operacional é responsável por oferecer acesso aos dispositivos de entrada/saída às aplicações e, em última instância, aos usuários do sistema. Prover acesso eficiente, rápido e confiável a um conjunto de periféricos com características diversas de comportamento, velocidade de transferência, volume de dados produzidos/consumidos e diferentes interfaces de hardware é um imenso desafio. Além disso, como cada dispositivo define sua própria interface e modo de operação, o núcleo do sistema operacional deve implementar código de baixo nível para interagir com milhares de tipos de dispositivos distintos. Como exemplo, cerca de 60% das 12 milhões de linhas de código do núcleo Linux 2.6.31 pertencem a código de *drivers* de dispositivos de entrada/saída.

Para simplificar o acesso e a gerência dos dispositivos de entrada/saída, o código do sistema operacional é estruturado em camadas, que levam das portas de entrada/saída,

interrupções de operações de DMA a interfaces de acesso abstratas, como arquivos e *sockets* de rede. Uma visão conceitual dessa estrutura em camadas pode ser vista na Figura 8. Nessa figura, a camada inferior corresponde aos dispositivos periféricos propriamente ditos, como discos rígidos, teclados, etc. A camada logo acima, implementada em hardware, corresponde ao controlador específico de cada dispositivo (controlador IDE, SCSI, SATA, etc.) e aos controladores de DMA e de interrupções, pertencentes ao *chipset* do computador.

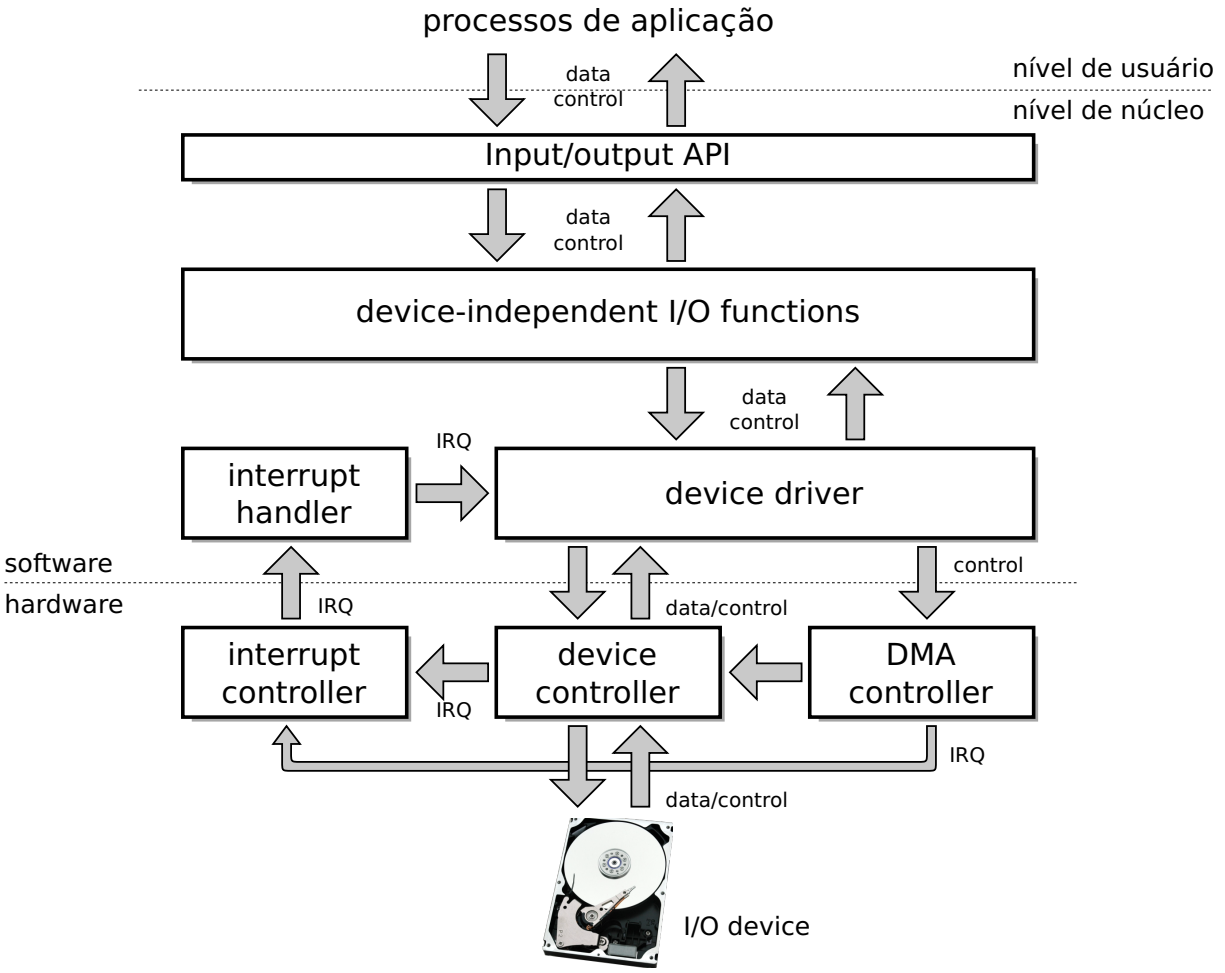

Figura 8: Estrutura em camadas do software de entrada/saída.

A primeira camada de software corresponde às rotinas de tratamento de interrupções (*interrupt handles*)e aos *drivers* de entrada/saída. As rotinas de tratamento de interrupção são acionadas pelo mecanismo de interrupção do processador a cada interrupção provinda do controlador de interrupções e servem basicamente para registrar sua ocorrência. A execução dessas rotinas deve ser muito breve, pois durante o tratamento de uma interrupção o processador desabilita a ocorrência de novas interrupções, conforme discutido na Seção 2.5. Assim, quando uma rotina é acionada, ela apenas reconhece a interrupção ocorrida junto ao controlador, cria um **descritor de evento** (*event handler*) contendo os dados da interrupção, o insere em uma fila de eventos pendentes mantida pelo *driver* do dispositivo, notifica o *driver* e conclui.

Os eventos da fila de eventos pendentes mantida por cada driver são tratados posterior, quando o processador estiver livre. A separação do tratamento de interrupções em dois níveis de urgência levar a estruturar o código de tratamento de cada interrupção também em dois níveis: o **bottom-half**, que compreende as ações imediatas a executar quando a interrupção ocorre, e o o **top-half**, que compreende o restante das ações de tratamento da interrupção.

3.1 Classes de dispositivos

Para simplificar a construção de aplicações (e do próprio sistema operacional), os dispositivos de entrada/saída são agrupados em classes ...

Os **dispositivos orientados a blocos** são aqueles em que as operações de entrada ou saída de dados são feitas usando blocos de bytes e nunca bytes isolados. Discos rígidos, fitas magnéticas e outros dispositivos de armazenamento são exemplos típicos desta categoria.

Os **dispositivos orientados a caracteres** são aqueles cujas transferências de dados são sempre feitas byte por byte, ou usando blocos de bytes de tamanho variável, cujo tamanho mínimo seja um byte. Dispositivos ligados às interfaces paralelas e seriais do computador, como *mouse* e teclado, são os exemplos mais clássicos deste tipo de dispositivo. Os terminais de texto e modems de transmissão de dados por linhas seriais (como as linhas telefônicas) também são vistos como dispositivos orientados a caracteres.

As **interfaces de rede** são colocadas em uma classe particular, pois são vistos como dispositivos orientados a blocos (os pacotes de rede são blocos), esses blocos são endereçáveis (os endereços dos destinos dos pacotes), mas a saída é feita de forma sequencial, bloco após bloco, e normalmente não é possível resgatar ou apagar um bloco enviado ao dispositivo.

sentido dos fluxos:	**entrada**	**saída**
granularidade	**caractere**: os dados são enviados ou recebidos byte por byte	**bloco**: os dados são enviados/-recebidos em blocos de tamanho fixo
exemplos:	terminais, portas paralelas e seriais, mouses, teclados	discos rígidos, interfaces de rede (pacotes), fitas magnéticas
acesso	**sequencial**: os dados são enviados ou recebidos em sequência, um após o outro	**direto**: a cada dado enviado ou recebido é associado um endereço (respectivamente de destino ou de origem)
exemplos:	porta paralela/serial, mouse, teclado, fita magnética	disco rígido, interface de rede
persistência:	**persistente**, se o dado enviado pode ser resgatado diretamente (ou, em outras palavras, se os dados lidos do dispositivo foram anteriormente escritos nele por aquele computador ou algum outro)	**volátil**, se o dado enviado é "consumido" pelo dispositivo, ou se o dado recebido do dispositivo foi "produzido" por ele e não anteriormente depositado nele.
exemplos:	fita magnética, disco rígido	interface de rede, porta serial/-paralela

- granularidade da informação: byte, bloco, stream
- tipos de dispositivos: a blocos, a caracteres, de rede, blocos sequenciais? (fita, rede)
- tipos de interface: bloqueante, não bloqueante, assíncrona
- arquitetura de E/S do kernel
  - estrutura de E/S do kernel: de devices genéricos a drivers específicos
  - interfaces, drivers, irq handlers, controllers
- Drivers
  - arquitetura geral
  - a estrutura de um driver
  - fluxograma de execução
  - top e bottom half
  - rotinas oferecidas aos processos
  - acesso via /dev
  - acesso ao hardware
  - integração ao kernel (recompilação ou módulos dinâmicos)

3.2 Estratégias de interação

O sistema operacional deve interagir com cada dispositivo de entrada/saída para realizar as operações desejadas, através das portas de seu controlador. Esta seção aborda as três estratégias de interação mais frequentemente usadas pelo sistema operacional, que são a entrada/saída controlada por programa, a controlada por eventos e o acesso direto à memória, detalhados a seguir.

3.2.1 Interação controlada por programa

A estratégia de entrada/saída mais simples, usada com alguns tipos de dispositivos, é a interação controlada por programa, também chamada **varredura** ou **polling**. Nesta abordagem, o sistema operacional solicita uma operação ao controlador do dispositivo, usando as portas *control* e *data-out* (ou *data-in*) de sua interface, e aguarda a conclusão da operação solicitada, monitorando continuamente os bits da respectiva porta de status. Considerando as portas da interface paralela descrita na Seção 2.3, o comportamento do processador em uma operação de saída na porta paralela usando essa abordagem seria descrito pelo seguinte pseudocódigo, no qual as leituras e escritas nas portas são representadas respectivamente pelas funções `in(port)` e `out(port,value)`[3]:

```
1  // portas da interface paralela LPT1 (endereço inicial em 0378h)
2  #define P0    0x0378        # porta de dados
3  #define P1    0x0379        # porta de status
4  #define P2    0x037A        # porta de controle
5
6  // máscaras para alguns bits de controle e status
7  #define BUSY  0x80          # 1000 0000 (bit 7)
8  #define ACK   0x40          # 0100 0000 (bit 6)
9  #define STROBE 0x01         # 0000 0001 (bit 0)
10
11 // buffer de bytes a enviar
12 unsigned char buffer[BUFSIZE] ;
13
14 polling_output ()
15 {
16   for (i = 0 ; i < BUFSIZE ; i++)
17   {
18     // espera o controlador ficar livre (bit BUSY deve ser zero)
19     while (in (P1) & BUSY) ;
20
21     // escreve o byte a enviar na porta P0
22     out (P0, buffer[i]) ;
23
24     // gera pulso em 0 no bit STROBE de P2, para indicar ao controlador
25     // que há um novo dado na porta P0
26     out (P2, in (P2) & ! STROBE) ; // poe bit STROBE de P2 em 0
27     usleep (1) ;                   // aguarda 1 us
28     out (P2, in (P2) | STROBE) ;   // poe bit STROBE de P2 em 1
29
30     // espera a entrada terminar de ser tratada (pulso "zero" em ACK)
31     while (in (P1) & ACK) ;
32   }
33 }
```

Em conjunto, processador e controlador executam ações coordenadas e complementares: o processador espera que o controlador esteja livre antes de enviar um novo dado; por sua vez, o controlador espera que o processador lhe envie um novo dado para processar. Essa interação é ilustrada na Figura 9. O controlador pode ficar esperando por novos dados, pois só precisa trabalhar quando há dados a processar, ou seja, quando o bit *strobe* de sua porta P_2 indicar que há um novo dado em sua porta P_0. Entretanto, manter o processador esperando até que a operação seja concluída é indesejável, sobretudo se a operação solicitada for demorada. Além de constituir uma situação clássica de desperdício de recursos por **espera ocupada**, manter o processador esperando pela resposta do controlador pode prejudicar o andamento de outras atividades importantes do sistema, como a interação com o usuário.

O problema da espera ocupada torna a estratégia de entrada/saída por programa pouco eficiente, sobretudo se o tempo de resposta do dispositivo for longo, sendo por isso pouco usada em sistemas operacionais de propósito geral. Seu uso se concentra

[3]Como o bit BUSY da porta P_1 deve retornar ao valor zero (0) após o pulso no bit ACK, o pseudocódigo poderia ser simplificado, eliminando o laço de espera sobre ACK; contudo, esse laço foi mantido para maior clareza didática.

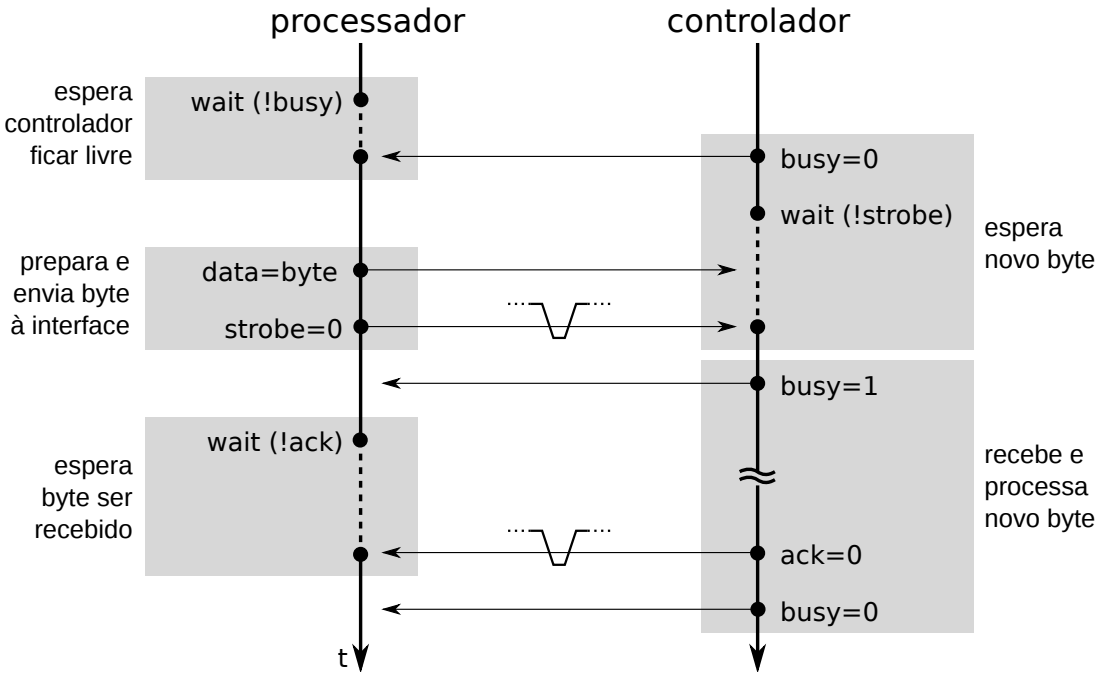

Figura 9: Entrada/saída controlada por programa.

sobretudo em sistemas embarcados dedicados, nos quais o processador só tem uma atividade (ou poucas) a realizar. A estratégia básica de varredura pode ser modificada, substituindo o teste contínuo do status do dispositivo por um teste periódico (por exemplo, a cada 10*ms*), e permitindo ao processador executar outras tarefas enquanto o dispositivo estiver ocupado. Todavia, essa abordagem implica em uma menor taxa de transferência de dados para o dispositivo e, por essa razão, só é usada em dispositivos com baixa vazão de dados.

3.2.2 Interação controlada por eventos

Uma forma mais eficiente de interagir com dispositivos de entrada/saída consiste em efetuar a requisição da operação desejada e suspender o fluxo de execução corrente, desviando o processador para tratar outro processo ou *thread*. Quando o dispositivo tiver terminado de processar a requisição, seu controlador irá gerar uma interrupção (IRQ) para notificar o processador, que poderá então retomar a execução daquele fluxo quando for conveniente. Essa estratégia de ação é denominada **interação controlada por eventos** ou por interrupções, pois as interrupções têm um papel fundamental em sua implementação.

Na estratégia de entrada/saída por eventos, uma operação de entrada ou saída é dividida em dois blocos de instruções: um bloco que inicia a operação, ativado pelo sistema operacional a pedido de um processo ou *thread*, e uma **rotina de tratamento de interrupção** (*interrupt handler*), ativado a cada interrupção, para prosseguir a transferência de dados ou para informar sobre sua conclusão. Considerando novamente a saída de um buffer de N bytes em uma interface paralela (descrita na Seção 2.3), o pseudocódigo a seguir representa o lançamento da operação de E/S pelo processador e a

rotina de tratamento de interrupção (subentende-se as constantes e variáveis definidas na listagem anterior):

```
1  // lançamento da operação de saída de dados
2  interrupt_driven_output ()
3  {
4    i = 0 ;
5
6    // espera o controlador ficar livre (bit BUSY de P1 deve ser zero).
7    while (in (P1) & BUSY) ;
8
9    // escreve o byte a enviar na porta P0.
10   out (P0, buffer[i]) ;
11
12   // gera pulso em 0 no bit STROBE de P2
13   out (P2, in (P2) & ! STROBE) ;
14   usleep (1) ;
15   out (P2, in (P2) | STROBE) ;
16
17   // suspende o processo solicitante, liberando o processador.
18   schedule () ;
19 }
20
21 // rotina de tratamento de interrupções da interface paralela
22 interrupt_handle ()
23 {
24   i++ ;
25   if (i >= BUFSIZE)
26     // a saída terminou, acordar o processo solicitante.
27     awake_process () ;
28   else
29   {
30     // escreve o byte a enviar na porta P0.
31     out (P0, buffer[i]) ;
32
33     // gera pulso em 0 no bit STROBE de P2
34     out (P2, in (P2) & ! STROBE) ;
35     usleep (1) ;
36     out (P2, in (P2) | STROBE) ;
37   }
38
39   // informa o controlador de interrupções que a IRQ foi tratada.
40   acknowledge_irq () ;
41 }
```

Nesse pseudocódigo, percebe-se que o processador inicia a transferência de dados para a interface paralela e suspende o processo solicitante (chamada *schedule*), liberando o processador para outras atividades. A cada interrupção, a rotina de tratamento é ativada para enviar um novo byte à interface paralela, até que todos os bytes do *buffer* tenham sido enviados, quando então sinaliza ao escalonador que o processo solicitante pode retomar sua execução (chamada *resume*). Conforme visto anteriormente, o controlador da interface paralela pode ser configurado para gerar uma interrupção através do flag

Enable_IRQ de sua porta de controle (porta P_2, Seção 2.3). O diagrama da Figura 10 ilustra de forma simplificada a estratégia de entrada/saída usando interrupções.

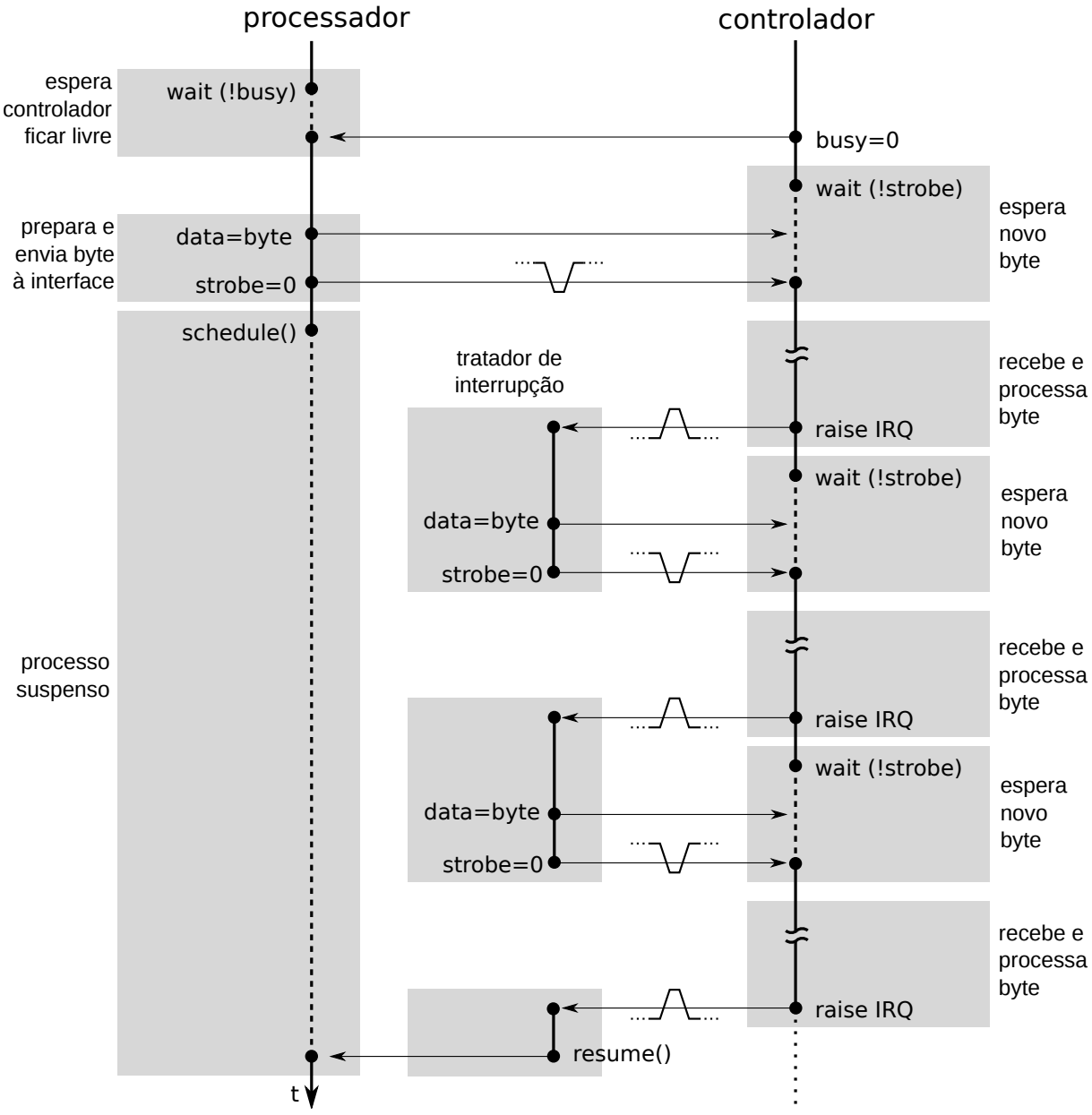

Figura 10: Entrada/saída controlada por eventos (interrupções).

Durante a execução da rotina de tratamento de uma interrupção, é usual inibir novas interrupções, para evitar a execução aninhada de tratadores de interrupção, o que tornaria o código dos *drivers* (e do núcleo) mais complexo e suscetível a erros. Entretanto, manter interrupções inibidas durante muito tempo pode ocasionar perdas de dados ou outros problemas. Por exemplo, uma interface de rede gera uma interrupção quando recebe um pacote vindo da rede; esse pacote fica em seu buffer interno e deve ser transferido dali para a memória principal antes que outros pacotes cheguem, pois esse

buffer tem uma capacidade limitada. Por essa razão, o tratamento das interrupções deve ser feito de forma muito rápida.

A maioria dos sistemas operacionais implementa o tratamento de interrupções em dois níveis distintos: um **tratador primário** (FLIH - *First-Level Interrupt Handler*) e um **tratador secundário** (SLIH - *Second-Level Interrupt Handler*). O tratador primário, também chamado *hard/fast interrupt handler* ou ainda *top-half handler*, é lançado quando a interrupção ocorre e deve executar rapidamente, pois as demais interrupções estão inibidas. Sua atividade se restringe a ações essenciais, como reconhecer a ocorrência da interrupção junto ao controlador e registrar dados sobre a mesma em uma fila de eventos pendentes, para tratamento posterior.

Por sua vez, o tratador secundário, também conhecido como *soft/slow interrupt handler* ou ainda *bottom-half handler*, tem por objetivo tratar a fila de eventos pendentes registrados pelo tratador primário, quando o processador estiver disponível. Ele pode ser escalonado e suspenso da mesma forma que um processo ou *thread*, embora normalmente execute com maior prioridade. Embora mais complexa, esta estrutura em dois nível traz algumas vantagens: ao permitir um tratamento mais rápido de cada interrupção, minimiza o risco de perder interrupções concomitantes; além disso, a fila de eventos pendentes pode ser analisada para remover eventos redundantes (como atualizações consecutivas de posição do *mouse*).

No Linux, cada interrupção possui sua própria fila de eventos pendentes e seus próprios *top-half* e *bottom-half*. Os tratadores secundários são lançados pelos respectivos tratadores primários, sob a forma de *threads* de núcleo especiais (denominadas *tasklets* ou *workqueues*) [Bovet and Cesati, 2005]. O núcleo Windows NT e seus sucessores implementam o tratamento primário através de rotinas de serviço de interrupção (ISR - *Interrupt Service Routine*). Ao final de sua execução, cada ISR agenda o tratamento secundário da interrupção através de um procedimento postergado (DPC - *Deferred Procedure Call*) [Russinovich and Solomon, 2004]. O sistema *Symbian* usa uma abordagem similar a esta.

Por outro lado, os sistemas Solaris, FreeBSD e MacOS X usam uma abordagem denominada *interrupt threads* [Mauro and McDougall, 2006]. Cada interrupção provoca o lançamento de uma *thread* de núcleo, que é escalonada e compete pelo uso do processador de acordo com sua prioridade. As interrupções têm prioridades que podem estar acima da prioridade do escalonador ou abaixo dela. Como o próprio escalonador também é uma *thread*, interrupções de baixa prioridade podem ser interrompidas pelo escalonador ou por outras interrupções. Por outro lado, interrupções de alta prioridade não são interrompidas pelo escalonador, por isso devem executar rapidamente.

3.2.3 Acesso direto à memória

Na maioria das vezes, o tratamento de operações de entrada/saída é uma operação lenta, pois os dispositivos são mais lentos que o processador. Além disso, o uso do processador principal para intermediar essas operações é ineficiente, pois implica em transferências adicionais (e desnecessárias) de dados: por exemplo, para transportar um byte de um *buffer* da memória para a interface paralela, esse byte precisa antes ser carregado em um registrador do processador, para em seguida ser enviado ao

controlador da interface. Para resolver esse problema, a maioria dos computadores atuais, com exceção de pequenos sistemas embarcados dedicados, oferece mecanismos de **acesso direto à memória** (DMA - *Direct Memory Access*), que permitem transferências diretas entre a memória principal e os controladores de entrada/saída.

O funcionamento do mecanismo de acesso direto à memória em si é relativamente simples. Como exemplo, a seguinte sequência de passos seria executada para a escrita de dados de um *buffer* em memória RAM para o controlador de um disco rígido:

1. o processador acessa os registradores do canal DMA associado ao dispositivo desejado, para informar o endereço inicial e o tamanho da área de memória RAM contendo os dados a serem escritos no disco. O tamanho da área de memória deve ser um múltiplo de 512 bytes, que é o tamanho padrão dos setores do disco rígido;
2. o controlador de DMA solicita ao controlador do disco a transferência de dados da RAM para o disco e aguarda a conclusão da operação;
3. o controlador do disco recebe os dados da memória;
4. a operação anterior pode ser repetida mais de uma vez, caso a quantidade de dados a transferir seja maior que o tamanho máximo de cada transferência feita pelo controlador de disco;
5. a final da transferência de dados, o controlador de DMA notifica o processador sobre a conclusão da operação, através de uma interrupção (IRQ).

A dinâmica dessas operações é ilustrada de forma simplificada na Figura 11. Uma vez efetuado o passo 1, o processador fica livre para outras atividades[4], enquanto o controlador de DMA e o controlador do disco se encarregam da transferência de dados propriamente dita. O código a seguir representa uma implementação hipotética de rotinas para executar uma operação de entrada/saída através de DMA.

[4]Obviamente pode existir uma contenção (disputa) entre o processador e os controladores no acesso aos barramentos e à memória principal, mas esse problema é atenuado pelo fato do processador poder acessar o conteúdo das memórias *cache* enquanto a transferência DMA é executada. Além disso, circuitos de *arbitragem* intermedeiam o acesso à memória para evitar conflitos. Mais detalhes sobre contenção de acesso à memória durante operações de DMA podem ser obtidos em [Patterson and Henessy, 2005].

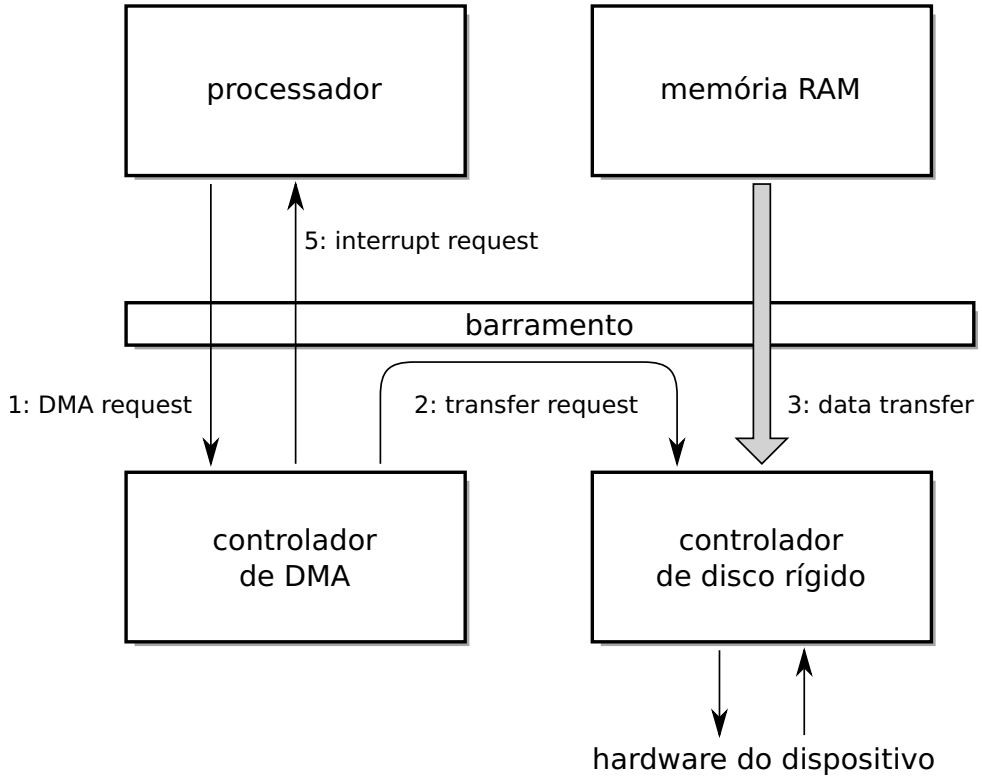

Figura 11: Funcionamento do acesso direto à memória.

```
1  // requisição da operação de saída através de DMA
2  dma_driven_output ()
3  {
4    // solicita uma operação DMA, informando os dados da transferência
5    // através da estrutura "dma_data".
6    request_dma (dma_data) ;
7
8    // suspende o processo solicitante, liberando o processador.
9    schedule () ;
10 }
11
12 // rotina de tratamento da interrupção do controlador de DMA
13 interrupt_handle ()
14 {
15   // informa o controlador de interrupções que a IRQ foi tratada.
16   acknowledge_irq () ;
17
18   // saída terminou, acordar o processo solicitante.
19   resume (...) ;
20 }
```

A implementação dos mecanismos de DMA depende da arquitetura e do barramento considerado. Computadores mais antigos dispunham de um controlador de DMA central, que oferecia vários **canais de DMA** distintos, permitindo a realização de transferências de dados por DMA simultâneas. Já os computadores pessoais usando

barramento PCI não possuem um controlador DMA central; ao invés disso, cada controlador de dispositivo conectado ao barramento pode assumir o controle do mesmo para efetuar transferências de dados de/para a memória sem depender do processador principal [Corbet et al., 2005], gerenciando assim seu próprio canal DMA.

No exemplo anterior, a ativação do mecanismo de DMA é dita **síncrona**, pois é feita explicitamente pelo processador, provavelmente em decorrência de uma chamada de sistema. Contudo, a ativação também pode ser **assíncrona**, quando ativada por um dispositivo de entrada/saída que dispõe de dados a serem transferidos para a memória, como ocorre com uma interface de rede ao receber dados provindos de outro computador através da rede.

O mecanismo de DMA é utilizado para transferir grandes blocos de dados diretamente entre a memória RAM e as portas dos dispositivos de entrada/saída, liberando o processador para outras atividades. Todavia, como a configuração de cada operação de DMA é complexa, para pequenas transferências de dados acaba sendo mais rápido e simples usar o processador principal [Bovet and Cesati, 2005]. Por essa razão, o mecanismo de DMA é usado preponderantemente nas operações de entrada/saída envolvendo dispositivos que produzem ou consomem grandes volumes de dados, como interfaces de rede, entradas e saídas de áudio, interfaces gráficas e discos.

Nas próximas seções serão discutidos alguns aspectos relevantes dos dispositivos de entrada/saída mais usados e da forma como estes são acessados e gerenciados nos sistemas de computação pessoal. Sempre que possível, buscar-se-á adotar uma abordagem independente de sistema operacional, concentrando a atenção sobre os aspectos comuns que devem ser considerados por todos os sistemas.

4 Discos rígidos

Discos rígidos estão presentes na grande maioria dos computadores pessoais e servidores. Um disco rígido permite o armazenamento persistente (não-volátil) de grandes volumes de dados com baixo custo e tempos de acesso razoáveis. Além disso, a leitura e escrita de dados em um disco rígido é mais simples e flexível que em outros meios, como fitas magnéticas ou discos óticos (CDs, DVDs). Por essas razões, eles são intensivamente utilizados em computadores para o armazenamento de arquivos do sistema operacional, das aplicações e dos dados dos usuários. Os discos rígidos também são frequentemente usados como área de armazenamento de páginas em sistemas de memória virtual (Seção **??**).

Esta seção inicialmente discute alguns aspectos de hardware relacionados aos discos rígidos, como sua estrutura física e os principais padrões de interface entre o disco e sua controladora no computador. Em seguida, apresenta aspectos de software que estão sob a responsabilidade direta do sistema operacional, como o *caching* de blocos e o escalonamento de operações de leitura/escrita no disco. Por fim, apresenta as técnicas RAID para a composição de discos rígidos, que visam melhorar seu desempenho e/ou confiabilidade.

4.1 Estrutura física

Um disco rígido é composto por um ou mais discos metálicos que giram juntos em alta velocidade (entre 4.200 e 15.000 RPM), acionados por um motor elétrico. Para cada face de cada disco há uma cabeça de leitura, responsável por ler e escrever dados através da magnetização de pequenas áreas da superfície metálica. Cada face é dividida logicamente em **trilhas** e **setores**; a interseção de uma trilha e um setor em uma face define um **bloco físico**, que é a unidade básica de armazenamento e transferência de dados no disco. Os discos rígidos atuais (até 2010) usam blocos de 512 bytes, mas o padrão da indústria está migrando para blocos físicos de 4.096 bytes. A Figura 12 apresenta os principais elementos que compõem a estrutura de um disco rígido. Um disco típico comporta milhares de trilhas e centenas de setores por face de disco [Patterson and Henessy, 2005].

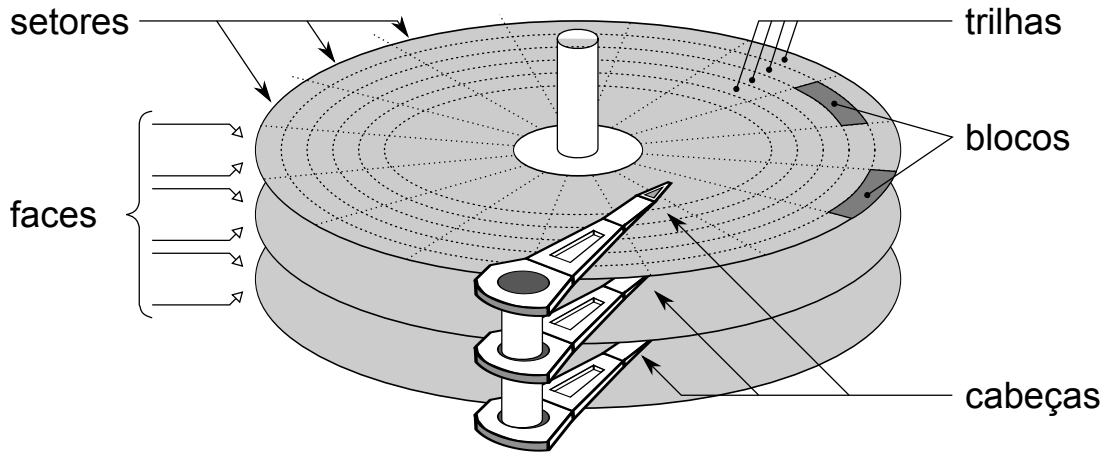

Figura 12: Elementos da estrutura de um disco rígido.

Por serem dispositivos eletromecânicos, os discos rígidos são extremamente lentos, se comparados à velocidade da memória ou do processador. Para cada bloco a ser lido/escrito, a cabeça de leitura deve se posicionar na trilha desejada e aguardar o disco girar até encontrar o setor desejado. Esses dois passos definem o *tempo de busca* (t_s – *seek time*), que é o tempo necessário para a cabeça de leitura se posicionar sobre uma determinada trilha, e a *latência rotacional* (t_r – *rotation latency*), que é o tempo necessário para o disco girar até que o setor desejado esteja sob a cabeça de leitura. Valores médios típicos desses atrasos para discos de uso doméstico são $t_s \approx 10ms$ e $t_r \approx 5ms$. Juntos, esses dois atrasos podem ter um forte impacto no desempenho do acesso a disco.

4.2 Interface de hardware

Conforme estudado anteriormente (Seções **??** e 2), o acesso do processador ao discos é feito através de uma *controladora* em hardware, ligada ao barramento do computador. Por sua vez, o disco é ligado à controladora de disco através de um barramento de interconexão, que pode usar diversas tecnologias. As mais comuns estão descritas a seguir:

- **IDE**: *Integrated Drive Electronics*, padrão também conhecido como PATA (*Parallell ATA - Advanced Technology Attachment*); surgiu nos anos 1980 e durante muito tempo foi o padrão de interface de discos mais usado em computadores pessoais. Suporta velocidades de até 133 MB/s, através de cabos paralelos de 40 ou 80 vias. Cada barramento IDE suporta até dois dispositivos, em uma configuração mestre/escravo.

- **SATA**: *Serial ATA*, é o padrão de interface de discos em *desktops* e *notebooks* atuais. A transmissão dos dados entre o disco e a controladora é serial, atingindo taxas entre 150 MB/s e 300 MB/s através de cabos com 7 vias.

- **SCSI**: *Small Computer System Interface*, padrão de interface desenvolvida nos anos 1980, foi muito usada em servidores e estações de trabalho de alto desempenho. Um barramento SCSI suporta até 16 dispositivos e atinge taxas de transferência de até 640 MB/s (divididos entre os dispositivos conectados no mesmo barramento).

- **SAS**: *Serial Attached SCSI*, é uma evolução do padrão SCSI, permitindo atingir taxas de transferência de até 600 MB/s em cada dispositivo conectado à controladora. É usado em equipamentos de alto desempenho.

É importante observar que esses padrões de interface não são de uso exclusivo em discos rígidos, muito pelo contrário. Há vários tipos de dispositivos que se conectam ao computador através dessas interfaces, como discos de estado sólido (SSD), leitores óticos (CD, DVD), unidades de fita magnética, *scanners*, etc.

4.3 Escalonamento de acessos

Em um sistema multitarefas, várias aplicações e processos do sistema podem solicitar acessos concorrentes ao disco, para escrita e leitura de dados. Por sua estrutura física, um disco rígido só pode atender a uma requisição de acesso por vez, o que torna necessário criar uma fila de acessos pendentes. Cada nova requisição de acesso ao disco é colocada nessa fila e o processo solicitante é suspenso até seu pedido ser atendido. Sempre que o disco concluir um acesso, ele informa o sistema operacional, que deve buscar nessa fila a próxima requisição de acesso a ser atendida. A ordem de atendimento das requisições pendentes é denominada **escalonamento de disco** e pode ter um grande impacto no desempenho do sistema operacional.

Na sequência do texto serão apresentados alguns algoritmos de escalonamento de disco clássicos. Para exemplificar seu funcionamento, será considerado um disco hipotético com 1.024 blocos, cuja cabeça de leitura se encontra inicialmente sobre o bloco 500. A fila de pedidos de acesso pendentes contém pedidos de acesso aos seguintes blocos do disco, em sequência: $\{278, 914, 71, 447, 161, 659, 335, 524\}$. Para simplificar, considera-se que nenhum novo pedido de acesso chegará à fila durante a execução dos algoritmos.

FCFS (*First Come, First Served*): esta abordagem consiste em atender as requisições de acesso na ordem em que elas foram pedidas pelos processos. É a estratégia mais simples de implementar, mas raramente oferece um bom desempenho. Se os

pedidos de acesso estiverem espalhados ao longo do disco, este irá perder muito tempo movendo a cabeça de leitura de um lado para o outro. A Figura 13 mostra os deslocamentos da cabeça de leitura para atender os pedidos de acesso da fila de exemplo. Pode-se perceber que a cabeça de leitura teve de percorrer 3374 blocos do disco para atender todos pedidos da fila.

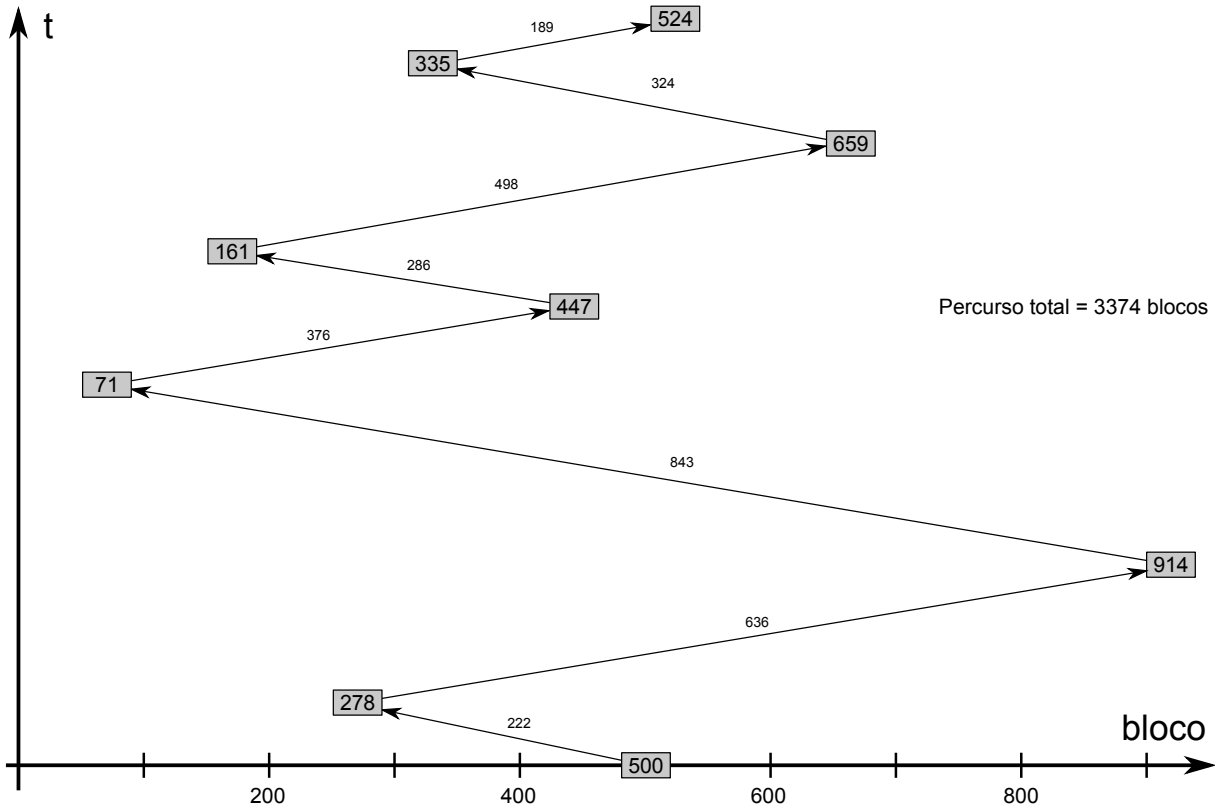

Figura 13: Escalonamento de disco FCFS.

SSTF (*Shortest Seek Time First – Menor Tempo de Busca Primeiro*): esta estratégia de escalonamento de disco consiste em sempre atender o pedido que está mais próximo da posição atual da cabeça de leitura (que é geralmente a posição do pedido que acabou de ser atendido). Dessa forma, ela busca reduzir os movimentos da cabeça de leitura, e com isso o tempo perdido entre os acessos atendidos. A estratégia SSTF está ilustrada na Figura 14. Pode-se observar uma forte redução da movimentação da cabeça de leitura em relação à estratégia FCFS, que passou de 3374 para 1320 blocos percorridos. Contudo, essa estratégia não garante um percurso mínimo. Por exemplo, o percurso 500 → 524 → 659 → 914 → 447 → 335 → 278 → 161 → 71 percorreria apenas 1257 blocos.

Apesar de oferecer um bom desempenho, esta estratégia pode levar à inanição (*starvation*) de requisições de acesso: caso existam muitas requisições em uma determinada região do disco, pedidos de acesso a blocos longe dessa região podem ficar muito tempo esperando. Para resolver esse problema, torna-se necessário implementar uma estratégia de *envelhecimento* dos pedidos pendentes.

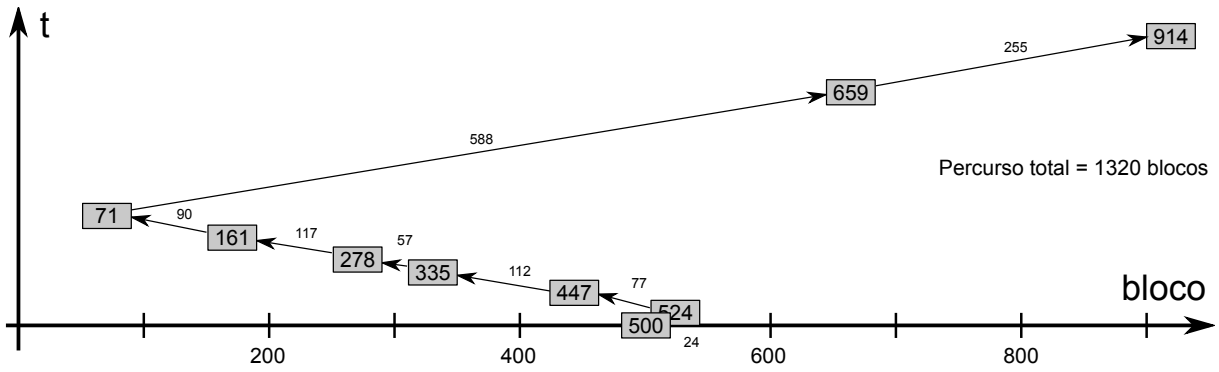

Figura 14: Escalonamento de disco SSTF.

Elevador : este algoritmo reproduz o comportamento dos elevadores em edifícios: a cabeça de leitura se move em uma direção e atende os pedidos que encontra pela frente; após o último pedido, ela inverte seu sentido de movimento e atende os próximos pedidos. Esse movimento também é análogo ao dos limpadores de para-brisas de um automóvel. A Figura 15 apresenta o comportamento deste algoritmo para a sequência de requisições de exemplo, considerando que a cabeça de leitura estava inicialmente se movendo para o começo do disco. Pode-se observar que o desempenho deste algoritmo foi melhor que os algoritmos anteriores, mas isso não ocorre sempre. A grande vantagem do algoritmo do elevador é atender os pedidos de forma mais uniforme ao longo do disco, eliminando a possibilidade de inanição de pedidos e mantendo um bom desempenho. Ele é adequado para sistemas com muitos pedidos concorrentes de acesso a disco em paralelo, como servidores de arquivos. Este algoritmo também é conhecido la literatura como SCAN ou LOOK [Silberschatz et al., 2001].

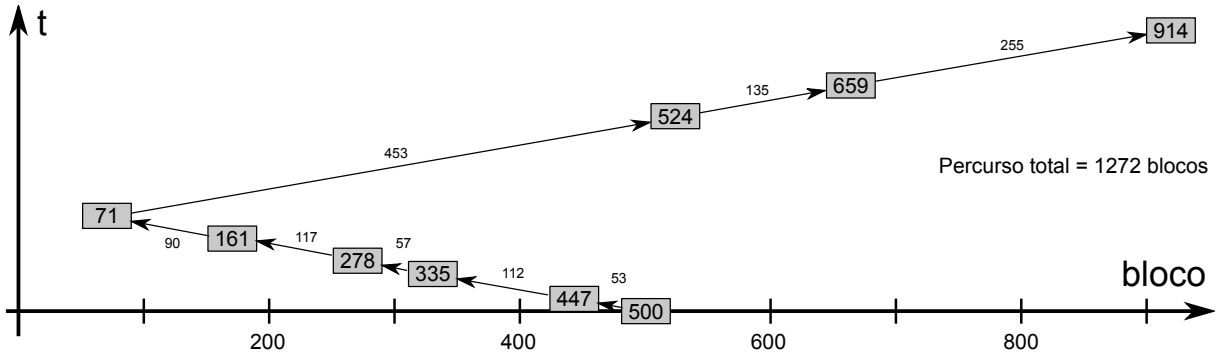

Figura 15: Escalonamento de disco com o algoritmo do elevador.

Elevador Circular : esta é uma variante do algoritmo do elevador, na qual a cabeça de leitura varre o disco em uma direção, atendendo os pedidos que encontrar. Ao atender o último pedido em um extremo do disco, ela retorna diretamente ao primeiro pedido no outro extremo, sem atender os pedidos intermediários, e recomeça. O nome "circular" é devido ao disco ser visto como uma lista circular de blocos. A Figura 16 apresenta um exemplo deste algoritmo. Apesar de seu

desempenho ser pior que o do algoritmo do elevador clássico, sua maior vantagem é prover um tempo de espera mais homogêneo aos pedidos pendentes, o que é importante em servidores. Este algoritmo também é conhecido como C-SCAN ou C-LOOK.

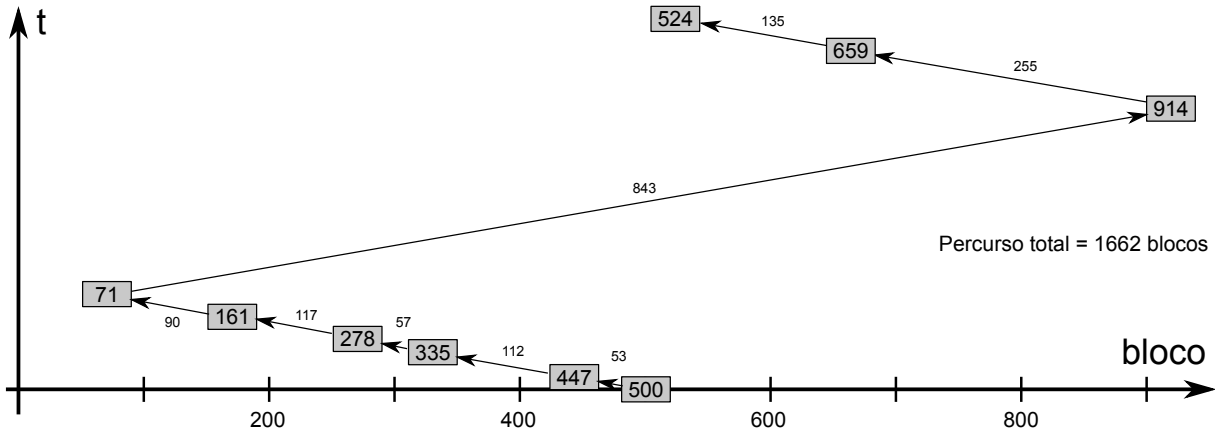

Figura 16: Escalonamento de disco com o algoritmo do elevador circular.

Sistemas reais mais complexos, como Solaris, Windows e Linux, utilizam escalonadores de disco geralmente mais sofisticados. No caso do Linux, os seguintes escalonadores de disco estão presentes no núcleo, podendo ser selecionados pelo administrador do sistema [Love, 2004, Bovet and Cesati, 2005]:

Noop (*No-Operation*): é o escalonador mais simples, baseado em FCFS, que não reordena os pedidos de acesso, apenas agrupa os pedidos direcionados ao mesmo bloco ou a blocos adjacentes. Este escalonador é voltado para discos de estado sólido (baseados em memória *flash*) ou sistemas de armazenamento que façam seu próprio escalonamento, como sistemas RAID (vide Seção 4.5).

Deadline : este escalonador é baseado no algoritmo do elevador circular, mas associa um prazo (*deadline*) a cada requisição, para evitar problemas de inanição. Como os pedidos de leitura implicam no bloqueio dos processos solicitantes, eles recebem um prazo de 500 *ms*; pedidos de escrita podem ser executados de forma assíncrona, por isso recebem um prazo maior, de 5 segundos. O escalonador processa os pedidos usando o algoritmo do elevador, mas prioriza os pedidos cujo prazo esteja esgotando.

Anticipatory : este algoritmo é baseado no anterior (*deadline*), mas busca se antecipar às operações de leitura de dados feitas pelos processos. Como as operações de leitura são geralmente feitas de forma sequencial (em blocos contíguos ou próximos), a cada operação de leitura realizada o escalonador aguarda um certo tempo (por default 6 *ms*) por um novo pedido de leitura naquela mesma região do disco, que é imediatamente atendido. Caso não surja nenhum pedido, o escalonador volta a tratar a fila de pedidos pendentes normalmente. Essa espera por pedidos adjacentes melhora o desempenho das operações de leitura.

CFQ (*Completely Fair Queuing*): os pedidos dos processos são divididos em várias filas (64 filas por default); cada fila recebe uma fatia de tempo para acesso ao disco, que varia de acordo com a prioridade de entrada/saída dos processos contidos na mesma. Este é o escalonador default do Linux na maioria das distribuições atuais.

4.4 *Caching* de blocos

Como o disco rígido pode apresentar latências elevadas, a funcionalidade de *caching* é muito importante para o bom desempenho dos acessos ao disco. É possível fazer *caching* de leitura e de escrita. No *caching* de leitura (*read caching*), blocos de dados lidos do disco são mantidos em memória, para acelerar leituras posteriores dos mesmos. No *caching* de escrita (*write caching*, também chamado *buffering*), dados a escrever no disco são mantidos em memória para leituras posteriores, ou para concentrar várias escritas pequenas em poucas escritas maiores (e mais eficientes). Quatro estratégias de *caching* são usuais:

- *Read-behind*: esta é a política mais simples, na qual somente os dados já lidos em requisições anteriores são mantidos em cache; outros acessos aos mesmos dados serão beneficiados pelo cache;
- *Read-ahead*: nesta política, ao atender uma requisição de leitura, são trazidos para o cache mais dados que os solicitados pela requisição; além disso, leituras de dados ainda não solicitados podem ser agendadas em momentos de ociosidade dos discos. Dessa forma, futuras requisições podem ser beneficiadas pela leitura antecipada dos dados. Essa política pode melhorar muito o desempenho de acesso sequencial a arquivos;
- *Write-through*: nesta política, ao atender uma requisição de escrita, uma cópia dos dados a escrever no disco é mantida em cache, para beneficiar possíveis leituras futuras desses dados;
- *Write-back*: nesta política, além de copiar os dados em cache, sua escrita efetiva no disco é adiada; esta estratégia melhora o desempenho de escrita de duas formas: por liberar mais cedo os processos que solicitam escritas (eles não precisam esperar pela escrita real no disco) e por concentrar as operações de escrita, gerando menos acessos a disco. Todavia, pode ocasionar perda de dados, caso ocorram erros de hardware ou falta de energia antes que os dados sejam efetivamente escritos no disco.

4.5 Sistemas RAID

Apesar dos avanços dos sistemas de armazenamento em estado sólido (como os dispositivos baseados em memórias *flash*), os discos rígidos continuam a ser o principal meio de armazenamento não-volátil de grandes volumes de dados. Os discos atuais têm capacidades de armazenamento impressionantes: encontram-se facilmente no mercado discos rígidos com capacidade da ordem de terabytes para computadores domésticos.

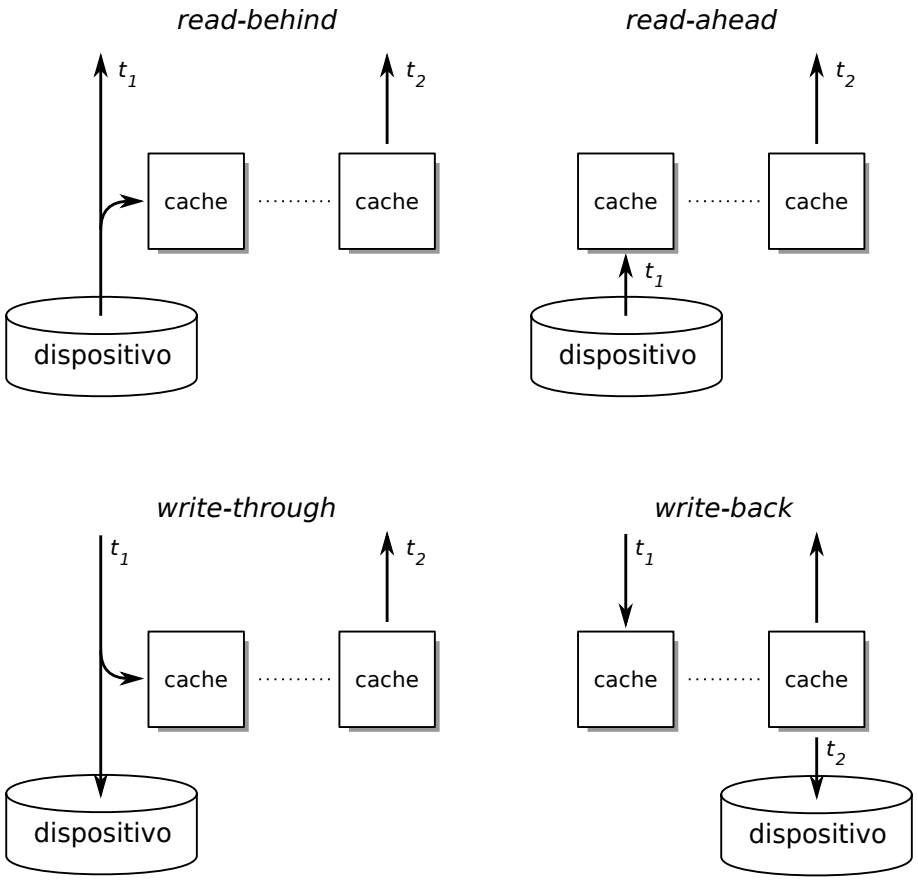

Figura 17: Estratégias de *caching* de blocos (t_1 e t_2 indicam dois instantes de tempo).

Entretanto, o desempenho dos discos rígidos evolui a uma velocidade muito menor que a observada nos demais componentes dos computadores, como processadores, memórias e barramentos. Com isso, o acesso aos discos constitui um dos maiores gargalos de desempenhos nos sistemas de computação. Boa parte do baixo desempenho no acesso aos discos é devida aos aspectos mecânicos do disco, como a latência rotacional e o tempo de posicionamento da cabeça de leitura do disco (vide Seção 4.3) [Chen et al., 1994].

Outro problema relevante associado aos discos rígidos diz respeito à sua confiabilidade. Os componentes internos do disco podem falhar, levando à perda de dados. Essas falhas podem estar localizadas no meio magnético, ficando restritas a alguns setores, ou podem estar nos componentes mecânicos/eletrônicos do disco, levando à corrupção ou mesmo à perda total dos dados armazenados.

Buscando soluções eficientes para os problemas de desempenho e confiabilidade dos discos rígidos, pesquisadores da Universidade de Berkeley, na Califórnia, propuseram em 1988 a construção de discos virtuais compostos por conjuntos de discos físicos, que eles denominaram RAID – *Redundant Array of Inexpensive Disks*[5] [Patterson et al., 1988], que em português pode ser traduzido como *Conjunto Redundante de Discos Econômicos*.

[5]Mais recentemente alguns autores adotaram a expressão *Redundant Array of Independent Disks* para a sigla RAID, buscando evitar a subjetividade da palavra *Inexpensive* (econômico).

Um sistema RAID é constituído de dois ou mais discos rígidos que são vistos pelo sistema operacional e pelas aplicações como um único disco lógico, ou seja, um grande espaço contíguo de armazenamento de dados. O objetivo central de um sistema RAID é proporcionar mais desempenho nas operações de transferência de dados, através do paralelismo no acesso aos vários discos, e também mais confiabilidade no armazenamento, usando mecanismos de redundância dos dados armazenados nos discos, como cópias de dados ou códigos corretores de erros.

Um sistema RAID pode ser construído "por hardware", usando uma placa controladora dedicada a esse fim, à qual estão conectados os discos rígidos. Essa placa controladora oferece a visão de um disco lógico único ao restante do computador. Também pode ser usada uma abordagem "por software", na qual são usados *drivers* apropriados dentro do sistema operacional para combinar os discos rígidos conectados ao computador em um único disco lógico. Obviamente, a solução por software é mais flexível e econômica, por não exigir uma placa controladora dedicada, enquanto a solução por hardware é mais robusta e tem um desempenho melhor. É importante observar que os sistemas RAID operam abaixo dos sistemas de arquivos, ou seja, eles se preocupam apenas com o armazenamento e recuperação de blocos de dados.

Há várias formas de se organizar um conjunto de discos rígidos em RAID, cada uma com suas próprias características de desempenho e confiabilidade. Essas formas de organização são usualmente chamadas *Níveis RAID*. Os níveis RAID padronizados pela *Storage Networking Industry Association* são [SNIA, 2009]:

RAID 0 (linear) : neste nível os discos físicos (ou partições) são simplesmente concatenados em sequência para construir um disco lógico. Essa abordagem, ilustrada na Figura 18, é denominada por alguns autores de *RAID 0 linear*, enquanto outros a denominam JBoD (*Just a Bunch of Disks* – apenas um punhado de discos). Como os blocos do disco lógico estão menos uniformemente espalhados sobre os discos físicos, os acessos podem se concentrar mais em um disco a cada instante, levando a um menor desempenho em leituras e escritas.

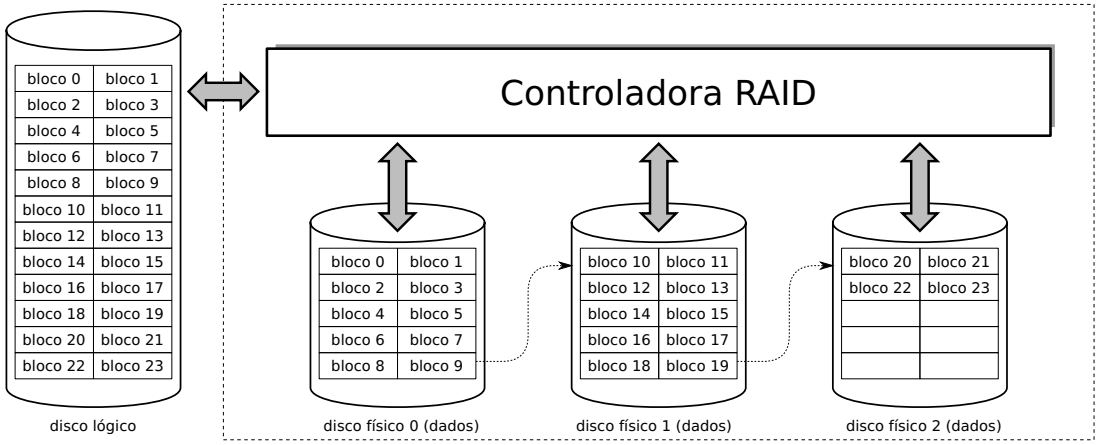

Figura 18: RAID nível 0 (*linear*).

RAID 0 (stripping) : neste nível os discos físicos são divididos em áreas de tamanhos fixo chamadas *fatias* ou *faixas* (*stripes*). Cada fatia de disco físico armazena um ou

mais blocos do disco lógico. As fatias são concatenadas usando uma estratégia *round-robin* para construir o disco lógico, como mostra a Figura 19.

Esta abordagem oferece um ganho de desempenho em operações de leitura e escrita: usando N discos físicos, até N operações podem ser efetuadas em paralelo. Entretanto, não há nenhuma estratégia de redundância de dados, o que torna este nível mais suscetível a erros de disco: caso um disco falhe, todos os blocos armazenados nele serão perdidos. Como a probabilidade de falhas aumenta com o número de discos, esta abordagem acaba por reduzir a confiabilidade do sistema de discos.

Suas características de grande volume de dados e alto desempenho em leitura/escrita tornam esta abordagem adequada para ambientes que geram e precisam processar grandes volumes de dados temporários, como os sistemas de computação científica [Chen et al., 1994].

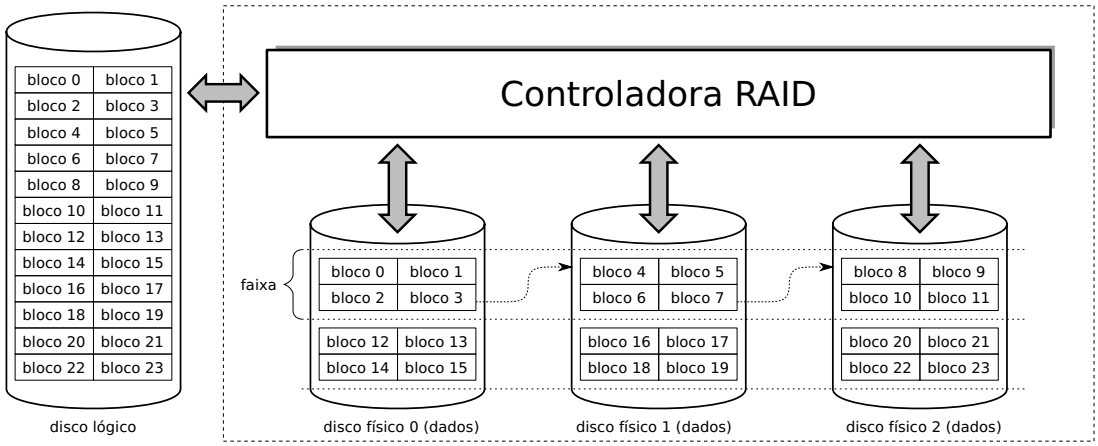

Figura 19: RAID nível 0 (*striping*).

RAID 1 : neste nível, cada disco físico possui um "espelho", ou seja, outro disco com a cópia de seu conteúdo, sendo por isso comumente chamado de *espelhamento de discos*. A Figura 20 mostra uma configuração simples deste nível, com dois discos físicos. Caso hajam mais de dois discos, devem ser incorporadas técnicas de RAID 0 para organizar a distribuição dos dados sobre eles (o que leva a configurações denominadas RAID 0+1, RAID 1+0 ou RAID 1E).

Esta abordagem oferece uma excelente confiabilidade, pois cada bloco lógico está escrito em dois discos distintos; caso um deles falhe, o outro continua acessível. O desempenho em leituras também é beneficiado, pois a controladora pode distribuir as leituras entre as cópias. Contudo, não há ganho de desempenho em escrita, pois cada operação de escrita deve ser replicada em todos os discos. Além disso, seu custo de implantação é elevado, pois são necessários dois discos físicos para cada disco lógico.

RAID 2 : este nível fatia os dados em bits individuais que são escritos nos discos físicos em sequência; discos adicionais são usados para armazenar códigos corretores de

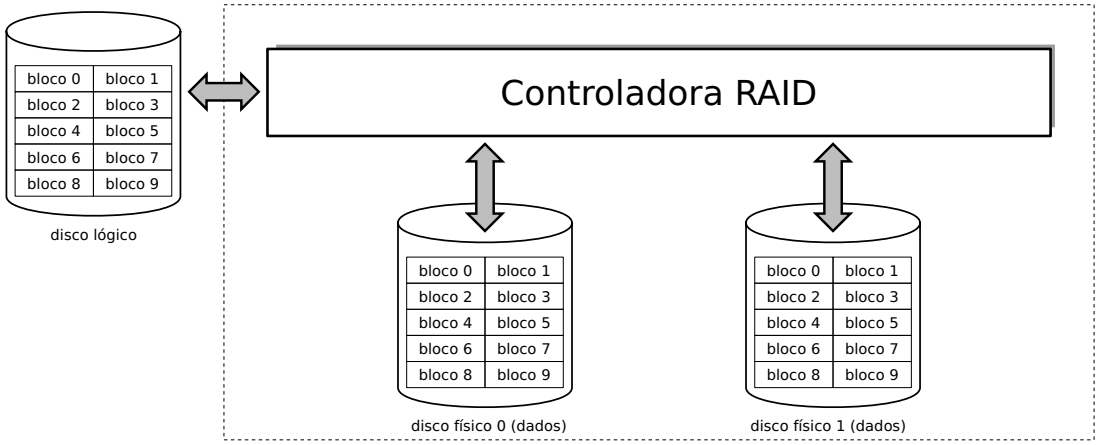

Figura 20: RAID nível 1 (*mirroring*).

erros (*Hamming Codes*), em um arranjo similar ao usado nas memórias RAM. Este nível não é usado na prática.

RAID 3 : este nível fatia os dados em bytes, que são escritos nos discos em sequência. Um disco separado contém dados de paridade, usados para a recuperação de erros. A cada leitura ou escrita, os dados do disco de paridade devem ser atualizados, o que implica na serialização dos acessos e a consequente queda de desempenho. Por esta razão, esta abordagem é raramente usada.

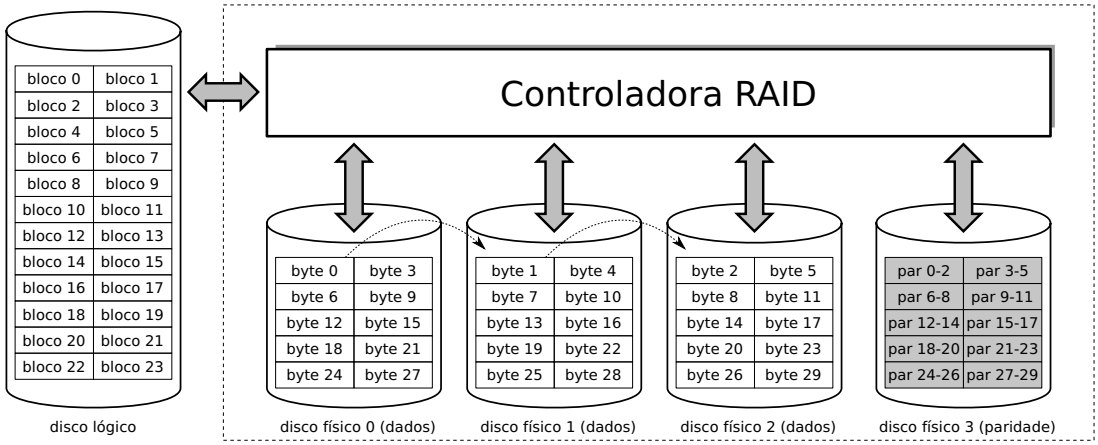

Figura 21: RAID nível 3.

RAID 4 : esta abordagem é similar ao RAID 3, com a diferença de que o fatiamento é feito por blocos ao invés de bytes (Figura 22). Ela sofre dos mesmos problemas de desempenho que o RAID 3, sendo por isso pouco usada. Todavia, ela serve como base conceitual para o RAID 5.

RAID 5 : assim como a anterior, esta abordagem também armazena informações de paridade para tolerar falhas em discos. Todavia, essas informações não ficam concentradas em um único disco físico, mas são distribuídas uniformemente entre

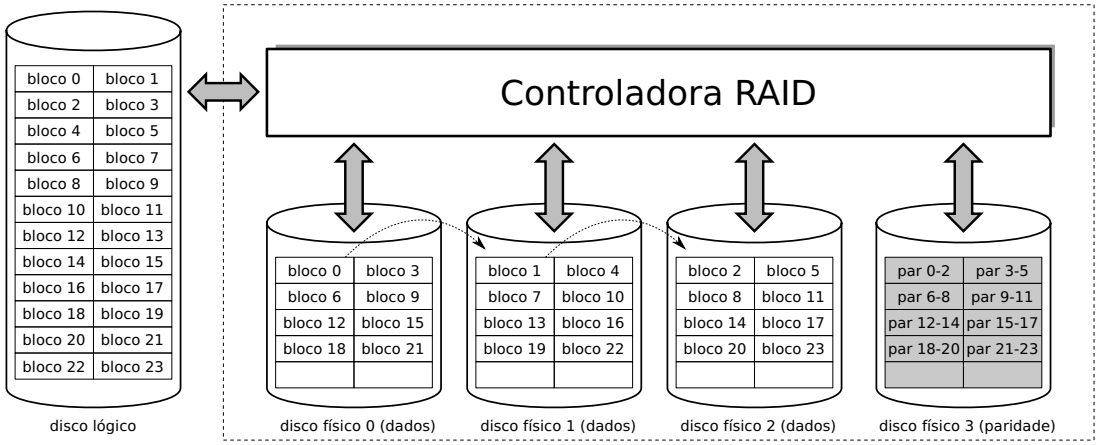

Figura 22: RAID nível 4.

todos eles. A Figura 23 ilustra uma possibilidade de distribuição das informações de paridade. Essa estratégia elimina o gargalo de desempenho no acesso aos dados de paridade. Esta é sem dúvida a abordagem de RAID mais popular, por oferecer um bom desempenho e redundância de dados com um custo menor que o espelhamento (RAID 1).

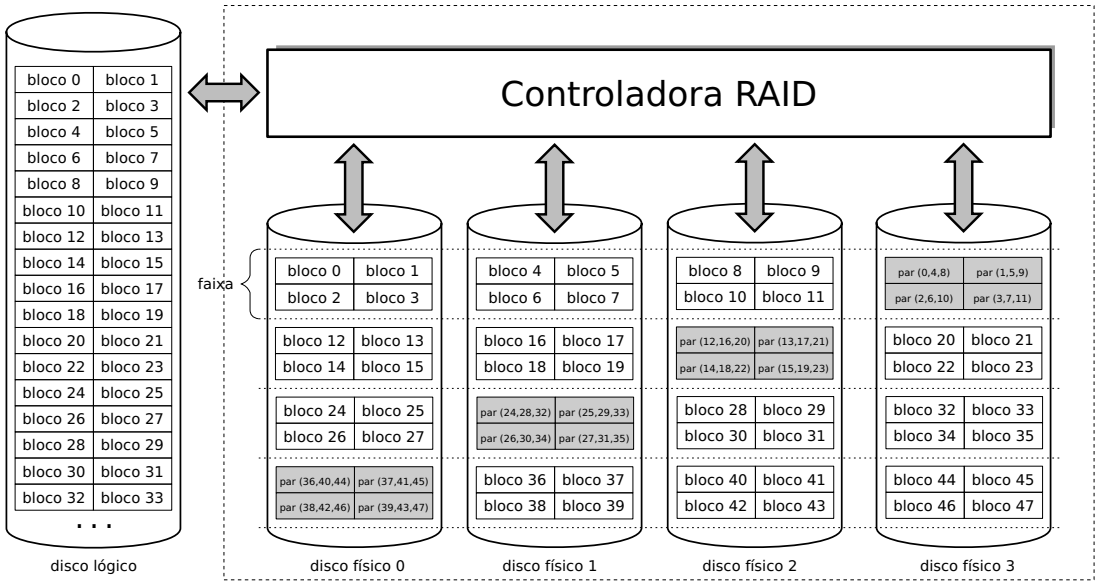

Figura 23: RAID nível 5.

RAID 6 : é uma extensão do nível RAID 5 que utiliza blocos com códigos corretores de erros de Reed-Solomon, além dos blocos de paridade. Esta redundância adicional permite tolerar falhas simultâneas de até dois discos.

Além dos níveis padronizados, no mercado podem ser encontrados produtos oferecendo outros níveis RAID, como 1+0, 0+1, 50, 100, etc., que muitas vezes implementam combinações dos níveis básicos ou soluções proprietárias. Outra observação importante

é que os vários níveis de RAID não têm necessariamente uma relação hierárquica entre si, ou seja, um sistema RAID 5 não é necessariamente melhor que um sistema RAID 1. Uma descrição mais aprofundada dos vários níveis RAID, de suas variantes e características pode ser encontrada em [Chen et al., 1994] e [SNIA, 2009].

5 Interfaces de rede

network I/O (modelo em camadas)

6 Dispositivos USB

7 Interfaces de áudio

capítulo separado sobre E/S de multimídia (áudio e vídeo, codecs)?

8 Interface gráfica

9 Mouse e teclado

falar de terminal e e/s serial?

10 Outros tópicos

- como deve ser tratada a gerência de energia?
- tratar relógio em separado?
- ioctl
- sistemas de arquivos /dev, /proc e /sys
- major/minor numbers
- gestão de módulos: insmod, lsmod, modprobe
- acesso a barramentos: lsusb, lspci, ...

Referências

[Bovet and Cesati, 2005] Bovet, D. and Cesati, M. (2005). *Understanding the Linux Kernel, 3rd edition*. O'Reilly Media, Inc.

[Chen et al., 1994] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. (1994). RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26:145–185.

[Corbet et al., 2005] Corbet, J., Rubini, A., and Kroah-Hartman, G. (2005). *Linux Device Drivers, 3rd Edition*. O'Reilly Media, Inc.

[Love, 2004] Love, R. (2004). *Linux Kernel Development*. Sams Publishing Developer's Library.

[Mauro and McDougall, 2006] Mauro, J. and McDougall, R. (2006). *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture*. Prentice-Hall PTR.

[Patterson and Henessy, 2005] Patterson, D. and Henessy, J. (2005). *Organização e Projeto de Computadores*. Campus.

[Patterson et al., 1988] Patterson, D. A., Gibson, G., and Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD International Conference on Management of Data*, pages 109–116. ACM.

[Russinovich and Solomon, 2004] Russinovich, M. and Solomon, D. (2004). *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press.

[Silberschatz et al., 2001] Silberschatz, A., Galvin, P., and Gagne, G. (2001). *Sistemas Operacionais – Conceitos e Aplicações*. Campus.

[SNIA, 2009] SNIA (2009). *Common RAID Disk Data Format Specification*. SNIA – Storage Networking Industry Association. Version 2.0 Revision 19.