

Dispositivos de E/S

- *Lembrete: o ponto de vista é o da programação do dispositivo, não da sua construção física*
- Existem duas grandes classes de dispositivos
- **Orientados a blocos:** transferem dados em unidades de tamanho fixo (blocos)
 - blocos variam de 512 bytes a 32 KB (tipicamente)
 - endereçamento também é por bloco (ler o bloco n)
- **Orientados a caracteres:** transferem dados em sequências (fluxos) de caracteres
 - não podem ser endereçados
- Nem todos os dispositivos se encaixam nessas classes
 - um relógio só gera interrupções a intervalos periódicos

Controladores de dispositivos

- Dispositivos tipicamente têm dois grandes componentes
 - eletrônico: programação
 - mecânico: dispositivo propriamente dito
- O componente eletrônico é o **controlador**
 - o mesmo controlador pode tratar vários dispositivos idênticos
- Tarefas do controlador
 - leitura e escrita serial de bits
 - conversão entre fluxos de bits e blocos fixos
 - verificação e correção de erros
 - tornar um bloco ou caracter disponível para ser copiado para a memória

Velocidade dos dispositivos

- As velocidades dos dispositivos variam bastante

Dispositivo	Taxa de dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem 56 K	7 KB/s
Canal telefônico	8 KB/s
Linhas ISDN dual	16 KB/s
Impressora a laser	100 KB/s
Scanner	400 KB/s
Ethernet clássica	1,25 MB/s
USB (<i>universal serial bus</i> — barramento serial universal)	1,5 MB/s
Câmara de vídeo digital	4 MB/s
Disco IDE	5 MB/s
CD-ROM 40x	6 MB/s
Ethernet rápida	12,5 MB/s
Barramento ISA	16,7 MB/s
Disco EIDE (ATA-2)	16,7 MB/s
FireWire (IEEE 1394)	50 MB/s
Monitor XGA	60 MB/s
Rede SONET OC-12	78 MB/s
Disco SCSI Ultra 2	80 MB/s
Ethernet Gigabit	125 MB/s
Dispositivo de Fita Ultrium	320 MB/s
Barramento PCI	528 MB/s
Barramento da Sun Gigaplane XB	20 GB/s

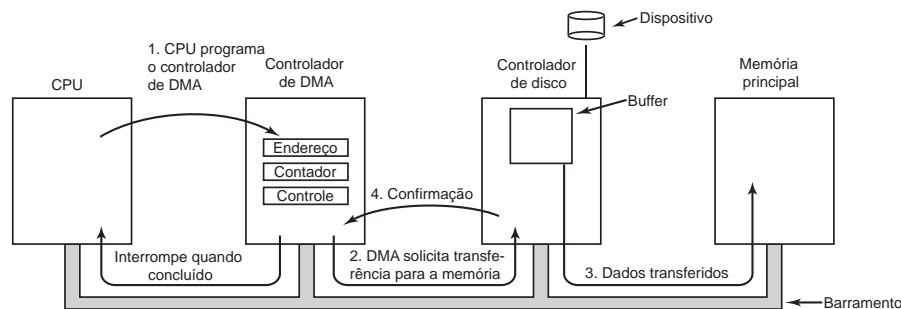
E/S mapeada em memória (1)

- Os controladores de dispositivos precisam ser endereçados de alguma maneira
 - registradores de dados, status e controle
- Existem duas formas básicas de endereçamento de E/S
- **Portas de E/S:** endereços separados para E/S
 - exigem instruções específicas de máquina (IN, OUT)
- **E/S mapeada em memória:** endereços de E/S são os mesmos endereços de memória
 - ocupam faixas distintas no espaço de endereçamento
 - utilizam as mesmas instruções de máquina que referenciam a memória

Leitura de dados sem DMA

1. O controlador efetua a leitura do bloco de dados (serialmente, bit a bit)
2. O controlador faz a verificação e correção de erros
3. O controlador causa uma interrupção
4. O SO lê o bloco do buffer do controlador, um byte ou bloco por vez
5. O SO armazena os dados lidos na memória
6. Repete até que a leitura esteja concluída

Transferência de dados com DMA



Leitura de dados com DMA

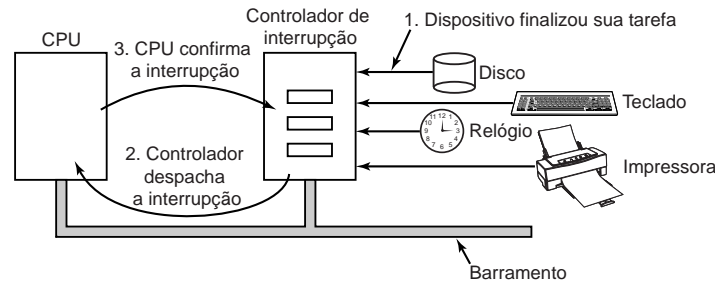
1. A CPU programa o controlador de DMA, indicando
 - quais os blocos que devem ser lidos
 - o endereço inicial na memória
 - o total de bytes/blocos a transferir
2. O controlador de DMA solicita a leitura do próximo bloco ao controlador de disco
3. O controlador de disco interrompe o controlador de DMA quando tiver lido um bloco
4. O controlador de DMA comanda a transferência dos dados do buffer do controlador de disco para a memória
5. Se houver mais dados a ler, volta ao passo 2; senão, interrompe a CPU sinalizando o final da transferência

Interrupções revisitadas (1)

Funcionamento de interrupções:

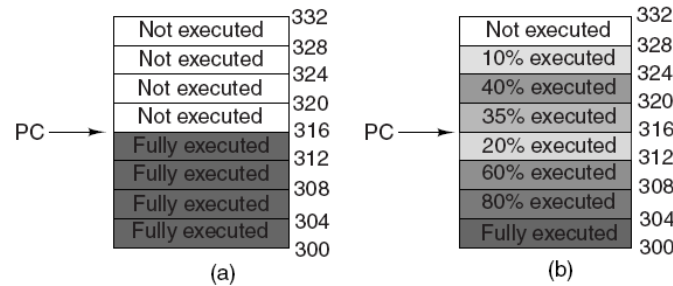
- Quando um dispositivo de E/S finaliza seu trabalho, ele gera uma interrupção
- Isso é feito através do envio de um sinal pela linha de barramento à qual o dispositivo está associado
- O sinal é detectado pelo chip controlador de interrupções localizado na placa-mãe
- O sinal de interrupção faz com que a CPU pare de executar a tarefa atual e vá tratar a interrupção

Interrupções revisitadas (2)



- A confirmação (3) sinaliza ao controlador que a interrupção foi atendida
 - o controlador de interrupção avisa ao dispositivo e pode liberar outra interrupção pendente

Interrupções precisas e imprecisas (2)



- (a) interrupção precisa
(b) interrupção imprecisa

Interrupções precisas e imprecisas (1)

- Em arquiteturas pipelined e superescalares, diversas instruções podem estar sendo executadas quando uma interrupção é gerada
 - qual o estado da CPU quando ocorre a interrupção?
- **Interrupções precisas** deixam o sistema em um estado bem definido
 1. O PC é salvo em um lugar conhecido
 2. Todas as instruções antes do PC foram completamente executadas
 3. Nenhuma instrução depois do PC foi executada
 4. O estado de execução da instrução apontada pelo PC é conhecido
- **Interrupções imprecisas** não atendem a esses requisitos
- Interrupções precisas complicam o hardware e simplificam o SO
 - abordagem adotada pela maioria das arquiteturas atuais

Sumário

- 1 Introdução
- 2 Princípios de hardware de E/S
- 3 Princípios do software de E/S
- 4 Camadas do software de E/S
- 5 Discos
- 6 E/S no Linux

Objetivos do software de E/S (1)

- **Independência do dispositivo:** programas de usuário devem funcionar com quaisquer dispositivos, sem se preocupar com as diferenças entre eles
 - o SO é responsável por tratar as diferenças
- **Nomeação uniforme:** todos os dispositivos devem ser nomeados da mesma maneira, sem distinções
 - exemplo: montagem de sistemas de arquivos no UNIX
- **Tratamento de erros:** muitos erros de E/S podem ser tratados em baixo nível, sem chegar até o usuário
 - erros transientes
 - ideal é tratar o mais próximo possível do hardware

Técnicas para realizar E/S

- E/S programada
- E/S orientada a interrupções
- E/S usando DMA

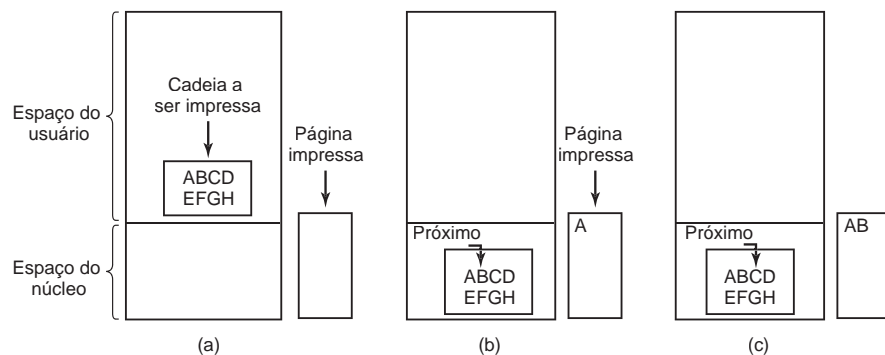
Objetivos do software de E/S (2)

- **Transferência síncrona vs assíncrona:** SO precisa implementar chamadas de sistema síncronas (bloqueantes) usando operações de E/S assíncronas
 - operações assíncronas: o chamador não espera pelo resultado, ele solicita a operação e mais tarde recupera o resultado
- **Buffers:** operações de E/S exigem que o SO gerencie buffers de armazenamento temporário de dados
 - transferências de dados entre buffers podem ter um impacto significativo no desempenho

E/S programada (1)

- A CPU faz todo o trabalho
 - forma mais simples
- Exemplo: impressão de uma string
 - SO copia string para um buffer
 - dados do buffer são enviados, caractere a caractere, para a impressora (usando E/S mapeada em memória, p.ex.)
- Espera ocupada ou *polling*
- CPU fica muito tempo ociosa
 - solução adequada para sistemas embarcados ou se E/S é muito rápida (e ociosidade é pequena)

E/S programada (2)



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

```
/* p é o buffer do núcleo */
/* executa o laço para cada caractere */
/* executa o laço até PRONTO */
/* envia um caractere para a saída */
```

E/S orientada a interrupções (2)

- Mais eficiente que E/S programada
- CPU ainda é interrompida para cada caractere impresso

E/S orientada a interrupções (1)

- Em vez da CPU esperar pelo dispositivo, ela inicia a operação de E/S e espera por uma interrupção para continuar
 - enquanto isso, outros processos podem ser executados

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

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

(a)

(b)

(a) início da operação

(b) tratamento da interrupção

E/S usando DMA

- CPU programa o controlador de DMA e é interrompida somente no final da operação
 - por exemplo, depois de todo o buffer impresso
- Em geral é mais eficiente
 - CPU é menos interrompida e fica livre para outros processos
 - controlador de DMA pode ser mais lento que a CPU

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

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

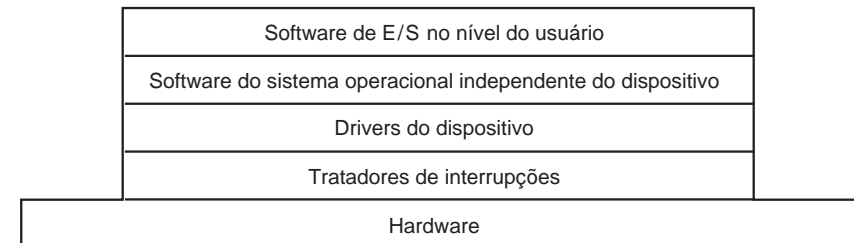
(a)

(b)

Camadas do software de E/S

- 1 Introdução
- 2 Princípios de hardware de E/S
- 3 Princípios do software de E/S
- 4 Camadas do software de E/S
- 5 Discos
- 6 E/S no Linux

- Software de E/S é organizado em quatro camadas
 - cada camada tem uma função bem definida



Tratadores de interrupções

- As interrupções devem ser escondidas o máximo possível
 - uma forma de fazer isso é bloqueando o driver que iniciou uma operação de E/S até que uma interrupção notifique que a E/S foi completada
- Rotina de tratamento de interrupção cumpre sua tarefa
 - e então desbloqueia o driver que a chamou
- Tratamento de interrupções em software envolve
 - trocar de contexto (processo corrente → tratador)
 - tratar a interrupção
 - escolher um novo processo para executar (escalonador)
 - trocar de contexto (tratador → novo processo)

- Cada controlador possui registradores para comandos e status
- O número de registradores varia de dispositivo para dispositivo
- Cada dispositivo de E/S precisa de um código específico de tratamento (**driver do dispositivo**)
- Em geral, os drivers são escritos pelo fabricante
 - questões de confiabilidade/segurança
- É desejável que um driver trate de uma classe de dispositivos

Drivers de dispositivos (2)

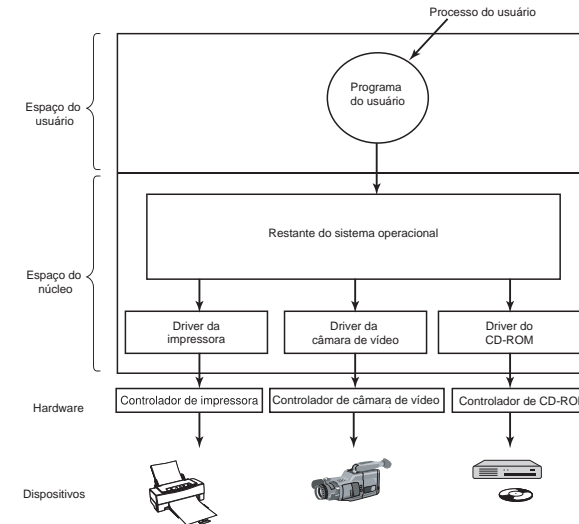
- Duas grandes categorias:
 - dispositivos de bloco: disco, fita, ...
 - dispositivos de caractere: mouse, teclado, impressora, ...
- Em geral, os sistemas operacionais definem uma interface padrão para cada categoria
- Algumas funções de um driver:
 - tratar requisições abstratas de leitura ou gravação independente do dispositivo
 - inicializar o dispositivo
 - tratar necessidade de energia
 - tratar eventos

E/S independente de dispositivo

Funções do software de E/S independente de dispositivo:

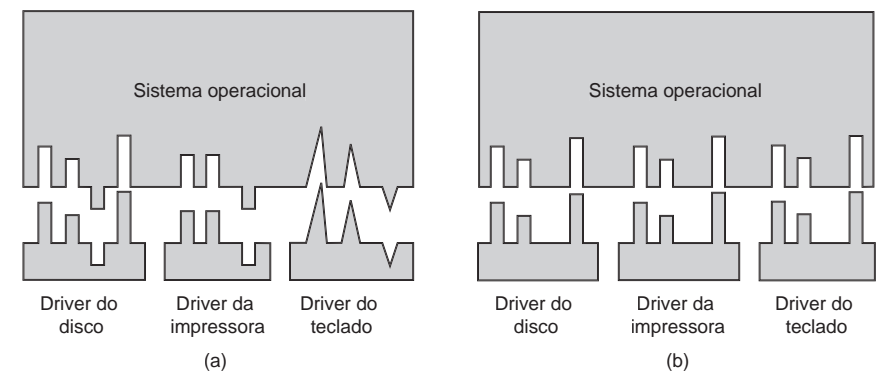
1. Interface uniforme para os drivers de dispositivos
2. Uso de buffers de E/S
3. Relatório de erros
4. Alocação e liberação de dispositivos dedicados
5. Tamanho de bloco independente de dispositivo

Drivers de dispositivos (3)



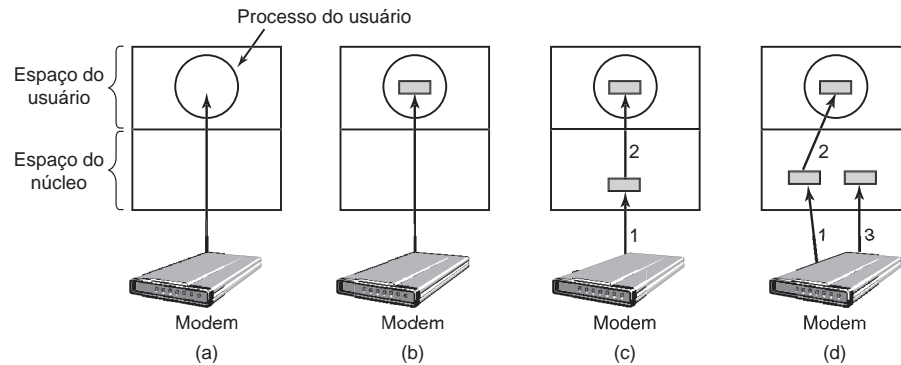
comunicação entre drivers e controladores é feita pelo barramento

Interface uniforme para os drivers de dispositivos



- (a) Sem uma interface padrão do driver
- (b) Com uma interface padrão do driver

Uso de buffers de E/S



- (a) Entrada sem uso de buffer
- (b) Uso de buffer no espaço do usuário
- (c) Uso de buffer no núcleo seguido de cópia para o espaço do usuário
- (d) Uso de buffer duplo no núcleo

Alocação/liberação de dispositivos

- Alguns dispositivos, tais como gravadores de CD/DVD, podem ser usados por apenas um único processo por vez
- Associar chamadas de sistema diretamente ao dispositivo (`open`)
 - caso o dispositivo não possa ser alocado, a chamada pode falhar ou o chamador ser bloqueado
 - decisão de projeto

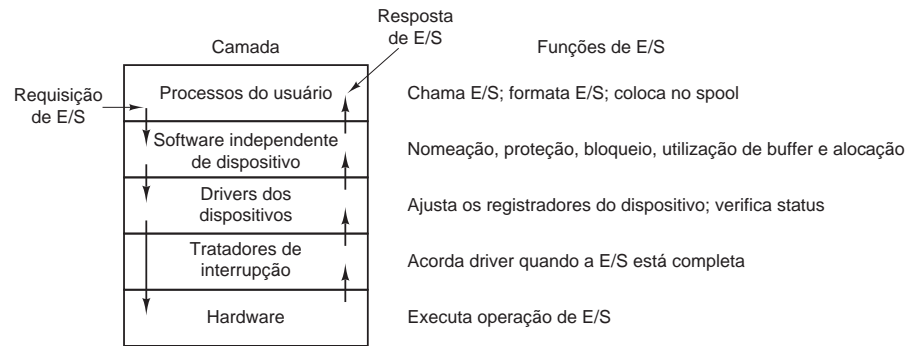
Relatórios de erros

- Erros de programação
 - escrita em um dispositivo de entrada (teclado)
 - fornecimento de endereços inválidos
 - atitude a ser tomada: retornar o código (tipo) de erro ocorrido ao processo envolvido
- Erros reais de E/S
 - tentativa de escrita em um bloco danificado
 - podem ser irreversíveis
 - leitura do MBR do disco de boot, por exemplo
 - atitude deve ser decidida pelo driver

Software de E/S no espaço de usuário

- Chamadas de sistema e funções de biblioteca que permitem ao usuário fazer E/S
 - `read`, `write`, `printf`, `scanf`, ...
- Spooling

Camadas do sistema de E/S: resumo



Discos magnéticos

- O disco é um dos dispositivos de E/S mais importantes
- Serve não apenas como memória secundária (de massa) como também oferece suporte a outras funções do SO
 - swapping
 - memória virtual
- Consideraremos aqui discos magnéticos (*hard disks*)
 - princípios de funcionamento similares aos de outros dispositivos como CD e DVD

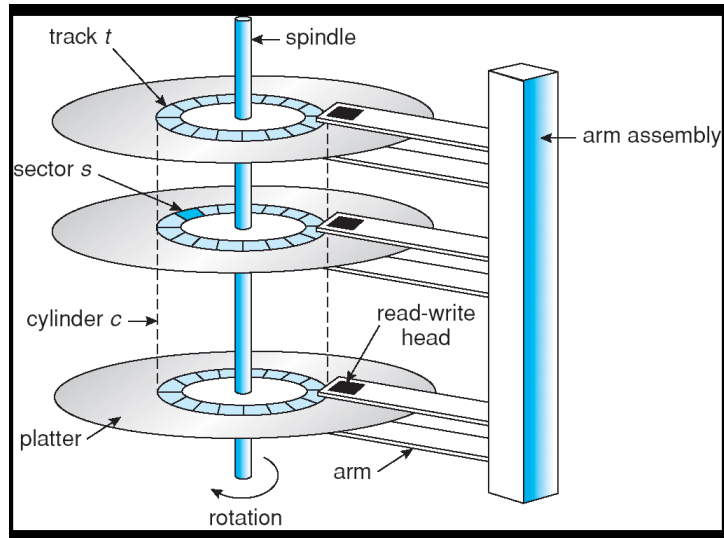
Sumário

- 1 Introdução
- 2 Princípios de hardware de E/S
- 3 Princípios do software de E/S
- 4 Camadas do software de E/S
- 5 Discos
- 6 E/S no Linux

Organização dos discos (1)

- Um disco é formado por um conjunto de pratos
 - superfícies cobertas por material ferromagnético
- A superfície do disco é magnetizada para armazenar os bits desejados
 - cabeça (cabeçote) de leitura e escrita converte bits de/para padrões magnéticos registrados na superfície do disco
 - há uma ou duas cabeças por superfície
- Cada disco é organizado em **trilhas** concêntricas
- Cada trilha é dividida em **setores** de tamanho fixo
 - trilhas e setores são definidos por formatação física pelo fabricante
- Um **cilindro** representa uma trilha em todas as superfícies

Organização dos discos (2)



45/72

Exemplos de parâmetros de disco

Parâmetro	Disco flexível IBM 360 KB	Disco rígido WD 18300
Número de cilindros	40	10 601
Trilhas por cilindro	2	12
Setores por trilha	9	281 (avg)
Setores por disco	720	35 742 000
Bytes por setor	512	512
Capacidade do disco	360 KB	18,3 GB
Tempo de posicionamento (cilindros adjacentes)	6 ms	0,8 ms
Tempo de posicionamento (caso médio)	77 ms	6,9 ms
Tempo de rotação	200 ms	8,33 ms
Tempo de pára/inicia do motor	250 ms	20 s
Tempo de transferência para um setor	22 ms	17 μs

- tempo de posicionamento diminuiu 7 vezes
- taxa de transferência aumentou 1300 vezes
- capacidade aumentou 50 mil vezes

47/72

Formato dos setores

- Cada setor possui o seguinte formato
 - preâmbulo: marca o início do setor
 - dados: geralmente em blocos de 512 bytes
 - ECC: um código de correção de erros (Reed-Solomon)
 - permite corrigir erros de leitura em alguns bits de dados



46/72

Acesso a dados

- Para acessar dados, é preciso localizar o bloco desejado
 - superfície, trilha, setor
- Existem dois métodos básicos de endereçamento
 - CHS (*Cylinder, Head, Sector*): usando as componentes
 - LBA (*Linear Block Addressing*): usando um no. de bloco
- O método mais usado atualmente é LBA
 - controlador é responsável por converter um número de bloco lógico nos parâmetros físicos adequados
 - permite mascarar detalhes como
 - variações no número de setores por trilha: zonas
 - setores defeituosos

48/72

Como determinar os componentes? (1)

- Para ler ou escrever dados é necessário posicionar corretamente a cabeça de leitura e gravação
- O tempo da operação é dado por três componentes
 - **tempo de posicionamento (seek)**: tempo necessário para levar o braço até a trilha desejada
 - **latência rotacional**: tempo necessário para que o setor desejado passe sob o cabeçote
 - **tempo de transferência**: tempo necessário para transferir efetivamente os dados
- Tempo de posicionamento domina

$$t_{acesso} = t_{seek} + t_{lat} + t_{transf}$$

Como determinar os componentes? (2)

- O tempo de transferência depende da velocidade de rotação e do tamanho e densidade dos setores

$$t_{transf} = \frac{b}{r \cdot N}$$

- b é a quantidade de bytes em um setor
- r é a velocidade de rotação (rps)
- N é a quantidade de bytes em uma trilha

- O tempo de seek é uma característica do disco, e tipicamente é fornecido pelo fabricante
 - muitas vezes se trabalha com um valor médio
- A latência rotacional depende da velocidade de rotação e de qual setor está passando embaixo do cabeçote quando o braço chega à trilha desejada
 - no melhor caso o setor desejado é o próximo
 - no pior caso, o setor desejado acabou de passar
 - na média, a latência é dada pelo tempo de meia rotação

Tempo médio de acesso

- O tempo médio de acesso é dado então por

$$t_{\text{acesso}} = t_{\text{seek}} + \frac{1}{2r} + \frac{b}{r \cdot N}$$

Cálculo do tempo médio (1)

- Qual o tempo médio necessário para ler um setor de um disco com os seguintes parâmetros?
 - tempo de seek médio: 10 ms
 - velocidade de rotação: 10.000 rpm
 - bytes/setor: 512
 - bytes/trilha: 32.768

$$t_{\text{acesso}} = 0,01 + \frac{1}{2 \times \frac{10000}{60}} + \frac{512}{\frac{10000}{60} \times 32768} = 13,09 \text{ ms}$$

Entrelaçamento (1)

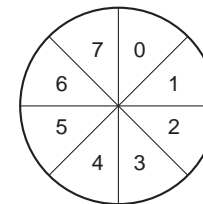
- Alguns discos não conseguem ler setores adjacentes de forma consecutiva
 - devido à transferência do bloco de dados do buffer do controlador para a memória, o próximo setor passa sob o cabeçote antes que a leitura possa iniciar
 - isso implica uma rotação adicional para cada setor
- Uma solução é introduzir setores entre setores logicamente adjacentes
 - dá tempo para que o próximo setor seja lido

Cálculo do tempo médio (2)

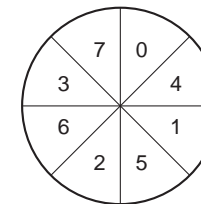
- Qual o tempo médio necessário para ler cinco setores consecutivos do disco usado como exemplo? (slide 47)
- Parâmetros:
 - tempo de seek médio: 6,9 ms
 - tempo de rotação: 8,33 ms
 - bytes/setor: 512
 - setores/trilha: 281

$$t_{\text{acesso}} = 6,9 + \frac{8,33}{2} + 8,33 \times \frac{5 \times 512}{281 \times 512} = 11,21 \text{ ms}$$

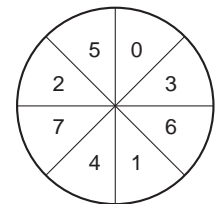
Entrelaçamento (2)



(a)



(b)



(c)

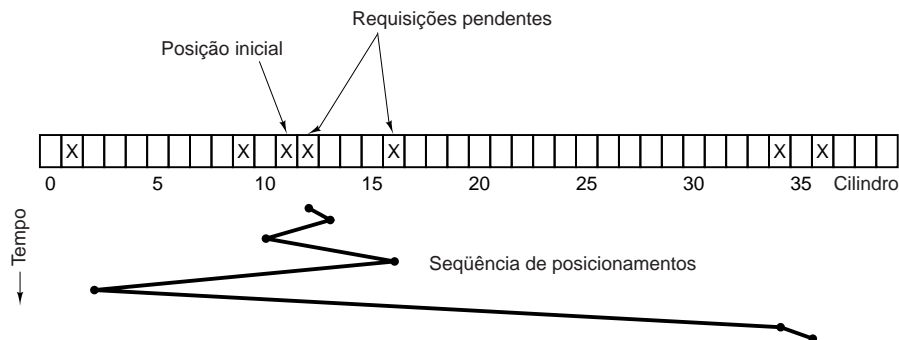
- (a) sem entrelaçamento
- (b) entrelaçamento simples
- (c) entrelaçamento duplo

FCFS (*First Come, First Served*)

- Como o tempo de posicionamento domina o tempo de acesso, reduções nesse tempo melhoram o desempenho dos acessos a disco
- Embora o tempo de posicionamento seja uma característica do disco, é possível mudar a ordem em que as requisições de leitura e escrita são atendidas para minimizar os deslocamentos do braço
- Essa é a função dos **algoritmos de escalonamento de disco**
 - FCFS
 - SSF
 - elevador

SSF (*Shortest Seek First*)

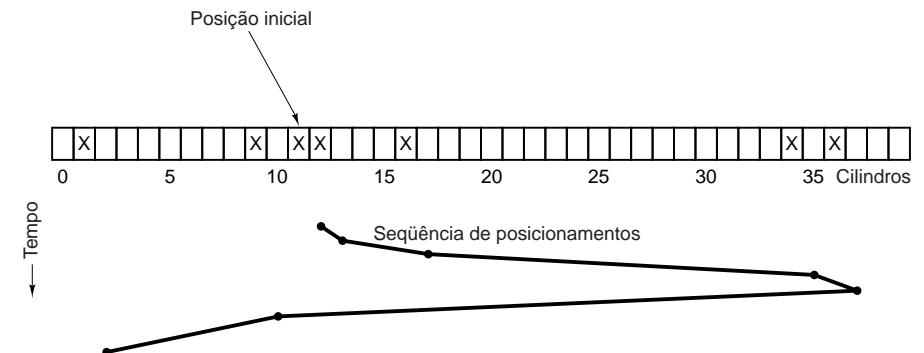
- A requisição seguinte a ser atendida é aquela que estiver mais próxima do cilindro atual
- Algumas requisições podem levar muito tempo para serem atendidas
 - concentração de requisições em uma região do disco
- Exemplo



- As requisições são atendidas na ordem de chegada
- Consideremos um exemplo
 - disco tem 40 cilindros
 - está sendo atendida uma requisição no cilindro 11
 - chegam requisições para os cilindros 1, 36, 16, 34, 9 e 12 (nessa ordem)
 - a distância total percorrida para atender o conjunto de requisições é de 111 cilindros

Algoritmo do elevador

- Atende a todas as requisições em uma direção
- Quando chega ao final, muda a direção e atende às demais requisições
- Exemplo (inicialmente subindo)



```
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
# echo deadline >/sys/block/sda/queue/scheduler
```


O elevador do Linus

- Usado no kernel 2.4
- Realiza ordenação e fusão de requisições, da seguinte forma:
 1. Se uma requisição para um setor adjacente já se encontra na fila, a requisição existente e a nova são fundidas
 - fusão pode ser recursiva, até um limite de blocos por requisição
 2. Se uma requisição na fila for suficientemente antiga, a nova requisição entra no final da fila
 - tenta evitar inanição de requisições, mas não funciona muito bem
 3. Caso contrário, insere a requisição na fila seguindo o algoritmo do elevador

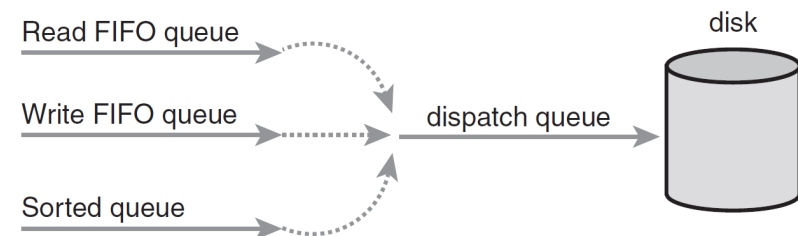
O escalonador com deadlines

- Introduzido no kernel 2.6
- Objetivo é reduzir os problemas de inanição do elevador do Linus
- Cada requisição possui um tempo de expiração
 - default: 500 ms para leituras, 5 s para escritas
- Mantém três filas de requisições
 - fila ordenada (como no elevador)
 - fila FIFO de leituras
 - fila FIFO de escritas
- Uma requisição é inserida na fila ordenada e no final da fila FIFO correspondente
- Geralmente, requisições são passadas da fila ordenada para a fila de despacho
 - se a requisição no início de uma das filas FIFO tiver expirado, ela é colocada na fila de despacho

Inanição de requisições

- O elevador do Linus está sujeito a inanição
- Concentração de requisições em uma região do disco
- Escritas causando inanição de leituras
 - operações de escrita são geralmente assíncronas
 - dados são copiados para um buffer no kernel antes de serem gravados no disco
 - a aplicação não fica esperando o resultado
 - leituras podem usar o buffer
 - operações de leitura são geralmente síncronas
 - aplicação fica bloqueada esperando os dados do disco
 - requisições de leitura tendem a ser encadeadas
 - uma requisição precisa esperar a anterior
 - latência de leitura afeta mais perceptivelmente o desempenho do que latência de escrita

O escalonador com deadlines



O escalonador antecipatório

- O escalonador com deadlines pode incorrer em seeks longos devido à expiração de requisições
 - se o sistema está num ciclo de escritas em uma região do disco e uma requisição de leitura expira, é necessário levar o braço até a trilha da leitura e depois retornar para as escritas
 - se as requisições de leitura forem encadeadas, isso pode ocorrer diversas vezes
 - vazão global do sistema é prejudicada
- O escalonador antecipatório adiciona uma heurística para minimizar o problema
 - quando uma requisição de leitura é atendida, o escalonador espera um tempo (default: 6 ms) antes de retornar à ordem normal de atendimento
 - se outra requisição de leitura para uma região adjacente chegar durante o tempo de espera, ela é atendida imediatamente
- Requer estatísticas por processo para antecipação funcionar bem



Escalonador noop

- Noop \rightarrow *No operation*
- Realiza apenas fusão de requisições adjacentes
- Usado em dispositivos de acesso realmente direto ($t_{seek} \rightarrow 0$)
 - pendrives e outras memórias de estado sólido

Escalonador com enfileiramento completamente justo

- *Completely fair queueing (CFQ)*
- Cada processo tem uma fila de requisições pendentes
 - uma nova requisição é ordenada e fundida com as requisições pendentes do mesmo processo
- O escalonador atende as filas em round-robin
 - um número de requisições de um processo (default: 4) são atendidas, e depois o escalonador escolhe a fila de outro processo
- Cada processo obtém uma fatia justa do disco
- Escalonador default no kernel 2.6

Bibliografía básica

-  **Andrew S. Tanenbaum.**
Sistemas Operacionais Modernos, 3ª Edição. Capítulo 5.
Pearson Prentice-Hall, 2010.
-  **Abraham Silberchatz, Greg Gagne e Peter Baer Galvin.**
Fundamentos de Sistemas Operacionais, 6ª Edição.
LTC - Livros Técnicos e Científicos Editora S.A., 2004.