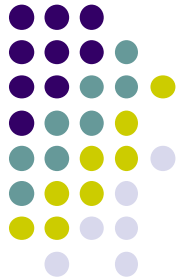
Sistemas de Computação

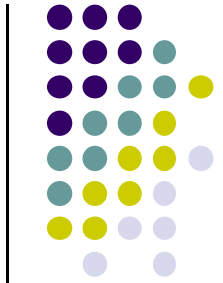
Entrada e Saída



Entrada/Saída

- Princípios do hardware de E/S
- Princípios do software de E/S
- Camadas do software de E/S
- Impasses
- Discos

Introdução



O controle da E/S é uma das tarefas centrais de um sistema operacional, que:

- emite comandos para os controladores de dispositivo, processa interrupções e trata erros
- esconde detalhes específicos dos diferentes dispositivos para o resto do S.O. através de uma interface uniforme
- tenta paralelizar E/S e processamento da CPU/Memória para minimizar latência e maximizar vazão do acesso aos dispositivos de E/S
- controla o acesso concorrente aos dispositivos

O sub-sistema de E/S consiste de duas camadas:

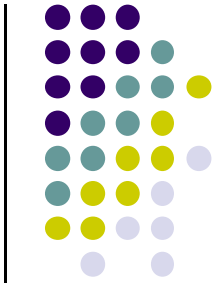
- Software independente dos dispositivos (para cada classe de dispositivo)
- Software dependente do dispositivo (drivers)

Princípios do Hardware de E/S

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 (*universal serial bus* — 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

Taxas de dados típicas de dispositivos, redes e barramentos

Tipos de Dispositivo



O Hardware de um dispositivo de E/S é definido por:

- Conjunto de comandos básicos disponibilizados
- As funções que executa
- As mensagens de status/erro que emite

Essencialmente, existem dois tipos de dispositivo:

Dispositivos de bloco = transferem ou armazenam dados em blocos de tamanho fixo, cada bloco possui um endereço e cada bloco pode ser acessado independentemente. Exemplos: HD, Floppy-disk, CD-ROM, DVD

Dispositivos de Caractere = envia ou aceita um fluxo de caracteres. Cada caractere não possui endereço e não pode ser acessado individualmente. Para cada caractere (ou conjunto pequeno deles), é gerada uma interrupção.

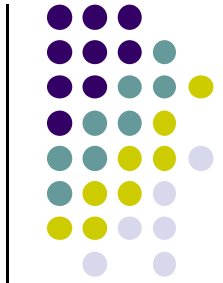
Exemplos: Teclado, Mouse, Interface de rede, etc.

Obs: Esta classificação é apenas geral: nem todo dispositivo de E/S se enquadra exatamente em uma das 2 categorias.

Ex: Fitas de backup, Vídeo mapeado em memória, etc.

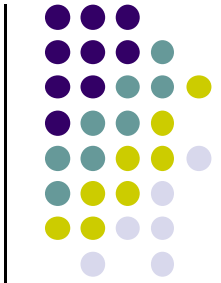
Categorias de Dispositivos

Outra classificação



- Para Interação Homem-Computador
  - Teclado, mouse, Display
  - Impressoras
- Para armazenamento e processamento interno
  - Discos
  - Pen-drives
- Para comunicação remota
  - Interfaces de rede (Ethernet, 802.11, BlueTooth, …)
  - Modems

Controladores de Dispositivos



- Componentes de dispositivos de E/S
  - Componente eletro-mecânico (motor, engrenagem, cabeçote, sensores, etc.)
  - Circuito integrado (= controlador do dispositivo)

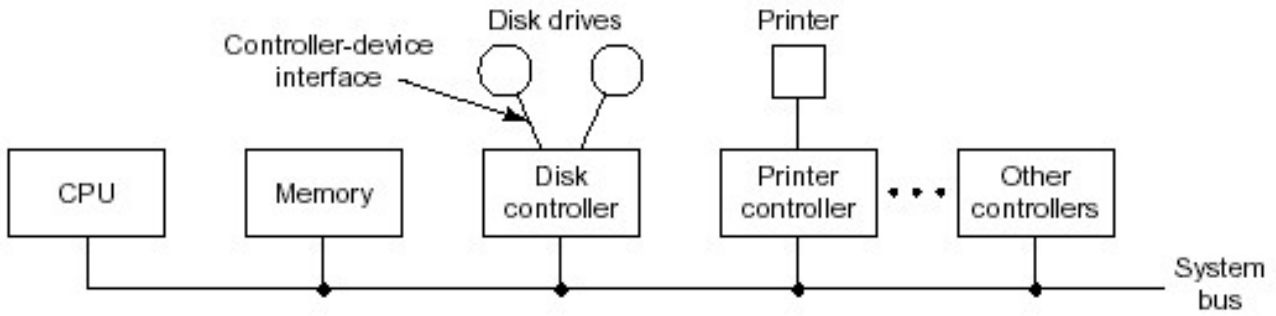
- A interface de hardware (pinos) entre o controlador e a componente mecânica é padronizada (ISO, SCSI, IDE)
- Um controlador pode fazer o controle agregado de vários dispositivos (componentes eletro-mecânicas)

- Tarefas do controlador
  - converter fluxo serial de bits em conjuntos de bytes
  - Verificação e correção de erros
  - Bufferização: agrupar bloco de bytes para transferência para a memória principal

- Do ponto de vista conceitual, o controlador é o dispositivo de E/S.

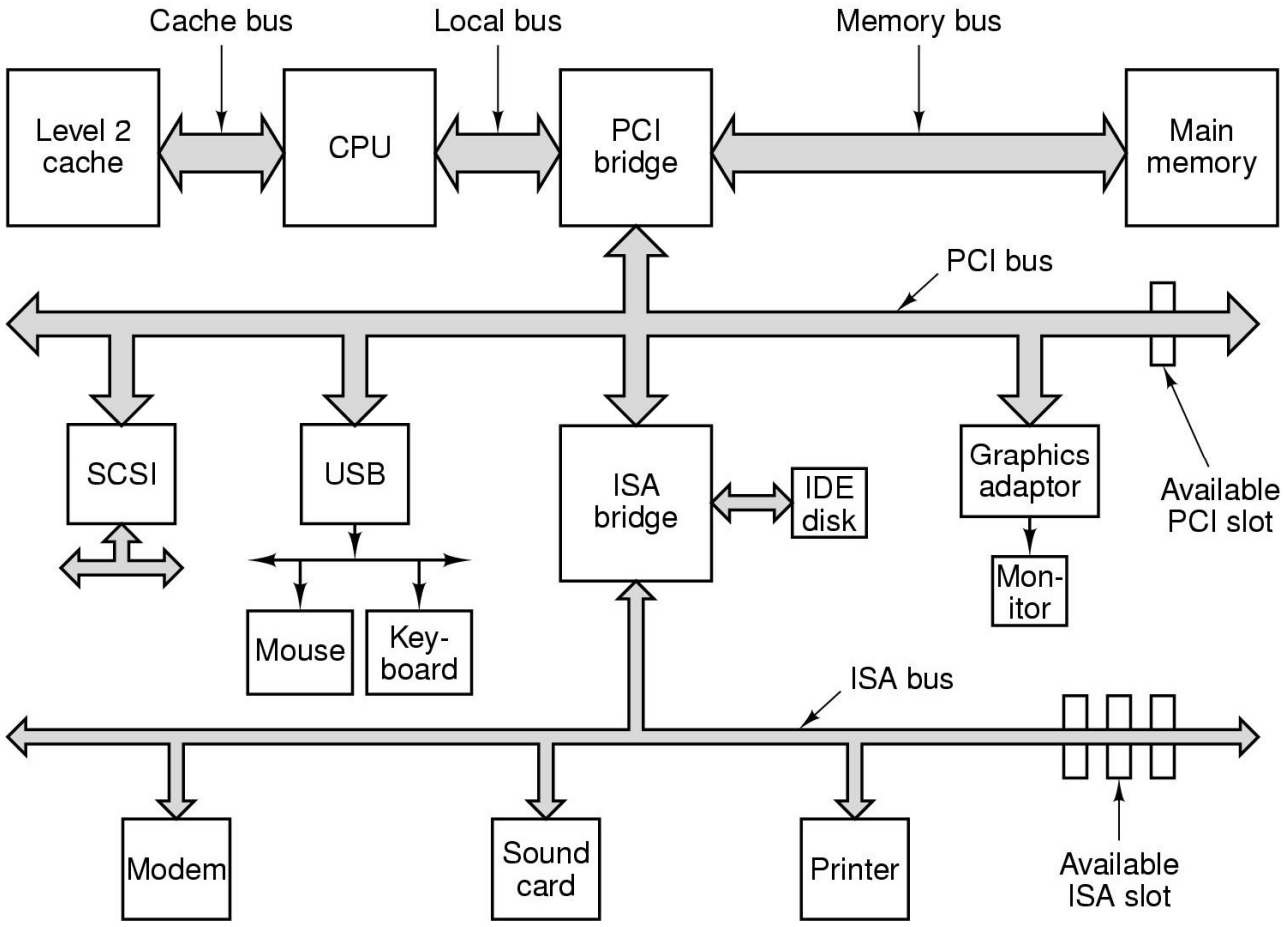
Arquitetura Conceitual

- **Conceitualmente, é como se houvesse conexão direta entre CPU/Memória e os controladores**



- **Em arquiteturas reais, existem vários barramentos e processadores dedicados para tirar a carga da CPU principal e servir de ponte entre os barramentos**

Arquitetura Real

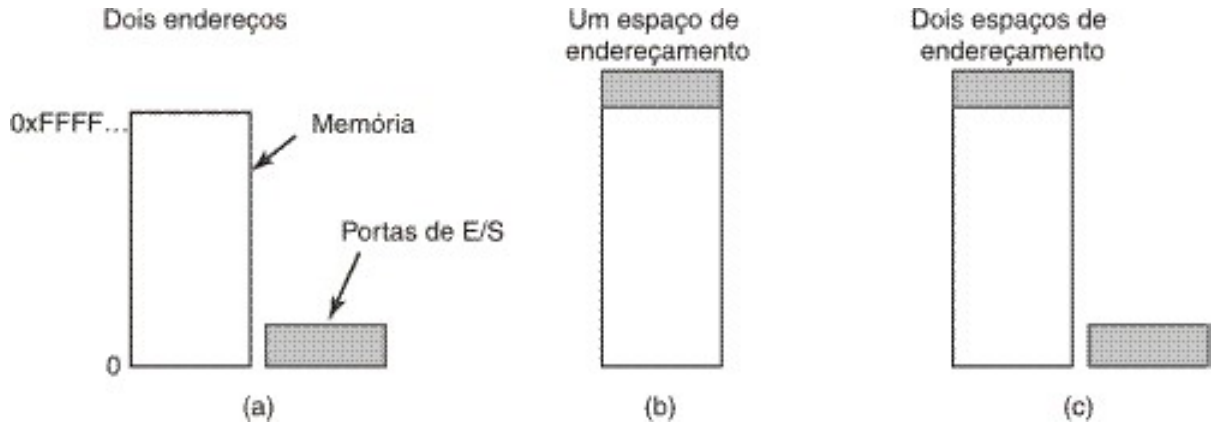


Arquitetura de um Pentium

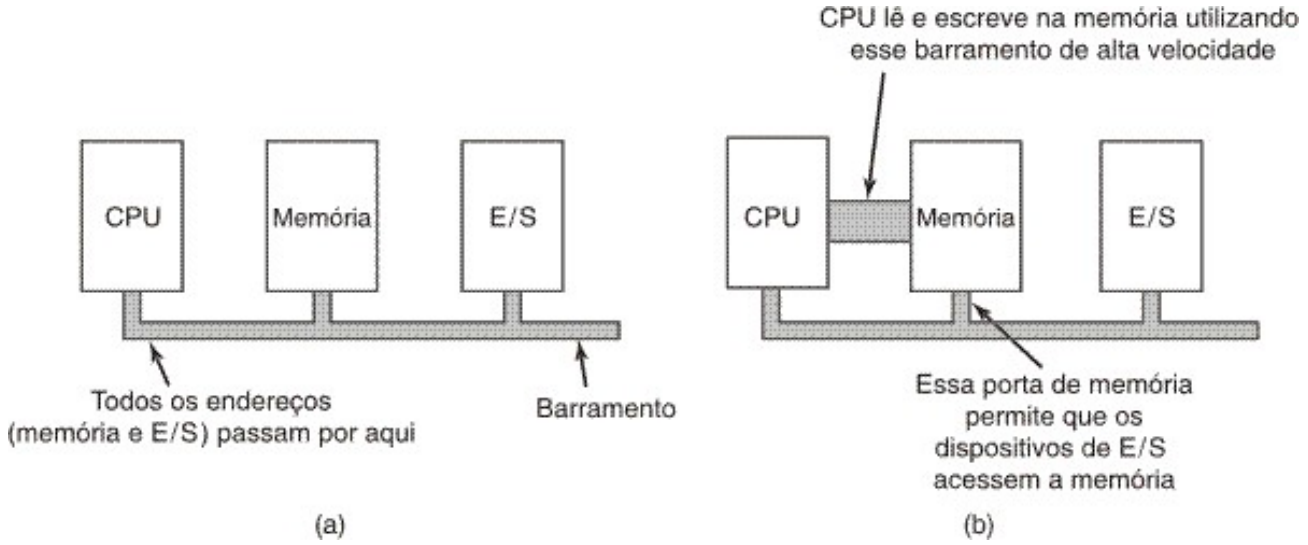
Controladores de Dispositivos

- Cada controlador possui alguns registradores que são usados para fazer o controle da E/S (da parte eletro-mecânica) e para emitir informações sobre status e condições de erro (Estes registradores são chamados portas de E/S)
- Em algumas arquiteturas, o espaço de endereçamento da memória RAM e desses registradores é único (E/S mapeada em memória) Exemplo: Motorola 680x0
- Em outras arquiteturas, existe um espaço de endereçamento específico para E/S (fora da RAM), usado apenas pelos controladores.
- A associação de um endereço de porta de E/S para um controlador é feita na interface componente-barramento pelo circuito de decodificação de endereços do barramento.

E/S mapeada na memória

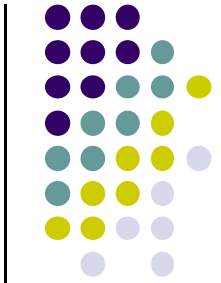


(a) Espaços de memória e E/S separados X (b) E/S mapeada na memória X (c) Híbrido



(a) Barramento único X (b) Barramento separado

Interação CPU-Controlador



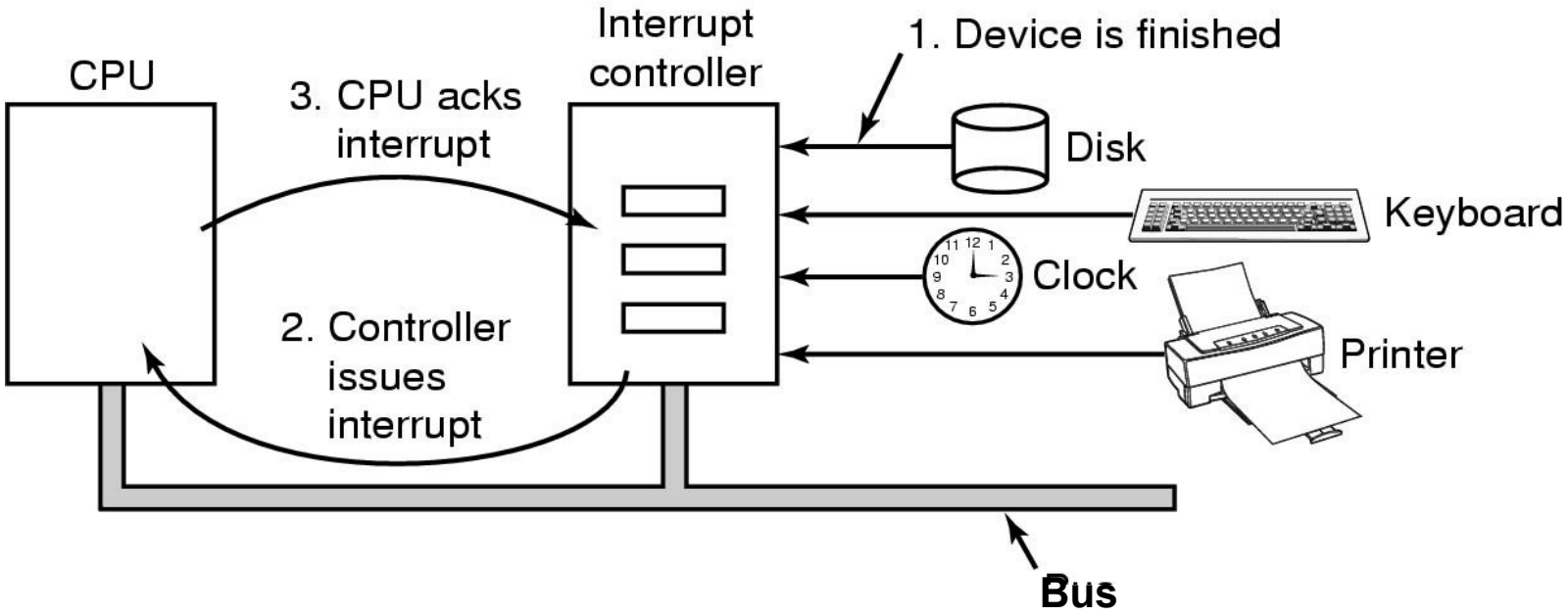
Funcionamento básico:

- **Driver escreve comandos e parâmetros nas portas de E/S (Por exemplo, instruções assembly**

  `IN Reg,Port` **e** `OUT Reg,Port`

- **Controladores ativam interrupções (e.g. IRQ = Interrupt ReQuest line), avisando quando seus registradores podem ser lidos/escritos ou quando a E/S foi finalizada**
- **O chip de controle de interrupções converte o IRQ para um índice do vetor de interrupção.**

Interrupções de Hardware



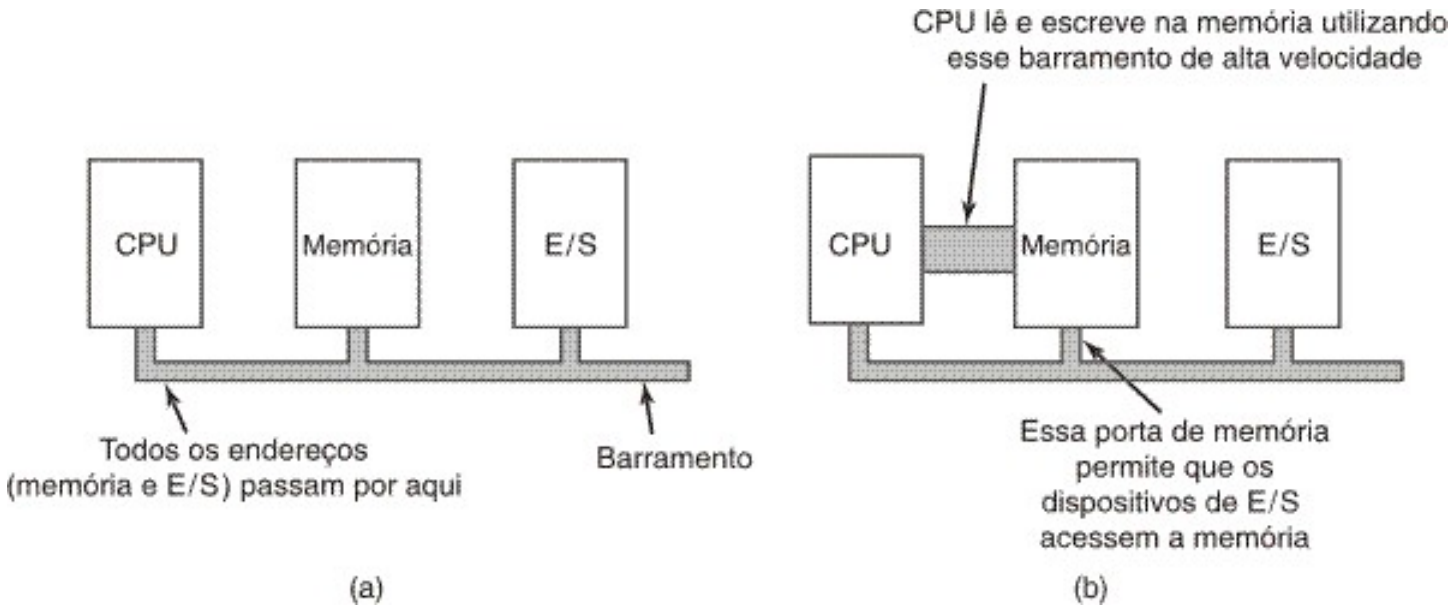
Obs: Conexão entre dispositivos e controlador de interrupções usam "linhas de interrupção" nos barramentos (em vez de cabos dedicados)

Tecnologia Plug'nPlay: Ao se conectar um dispositivo de E/S na placa mãe, a BIOS automaticamente atribui um IRQ livre para o dispositivo (Exemplo: Pentium PCs possuem 15 IRQs.)

Passos de uma E/S de bloco controlada por CPU

1. CPU escreve comando (p.ex., ler bloco de endereço *E*) em registrador do controlador de disco
2. Controlador aciona a componente eletro-mecânica, e transfere os dados bit-a-bit do disco para um buffer do controlador
3. Controlador calcula checksum para verificar se há dados corrompidos. Se houver, faz nova leitura.
4. Quando bloco de dados foi completamente lido, controlador levanta uma interrupção
5. Driver executa loop para copiar byte-by-byte o bloco do controlador para a memória principal

E/S mapeada na memória



(a) Arquitetura com barramento único
(b) Arquitetura com barramento dual

Direct Memory Access (DMA)

Em várias arquiteturas existe um componente de HW dedicado à transferência direta de blocos de dados para a memória - Direct Memory Access (DMA):

- Evita que a CPU tenha que executar o loop para transferência do bloco de/para a memória principal.
- CPU apenas inicia E/S informando: endereço inicial na memória, endereço do bloco no disco, número de bytes a serem transferidos.

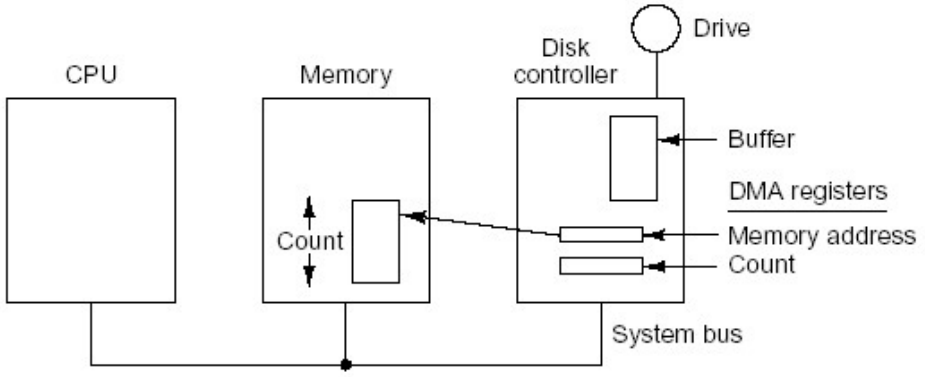
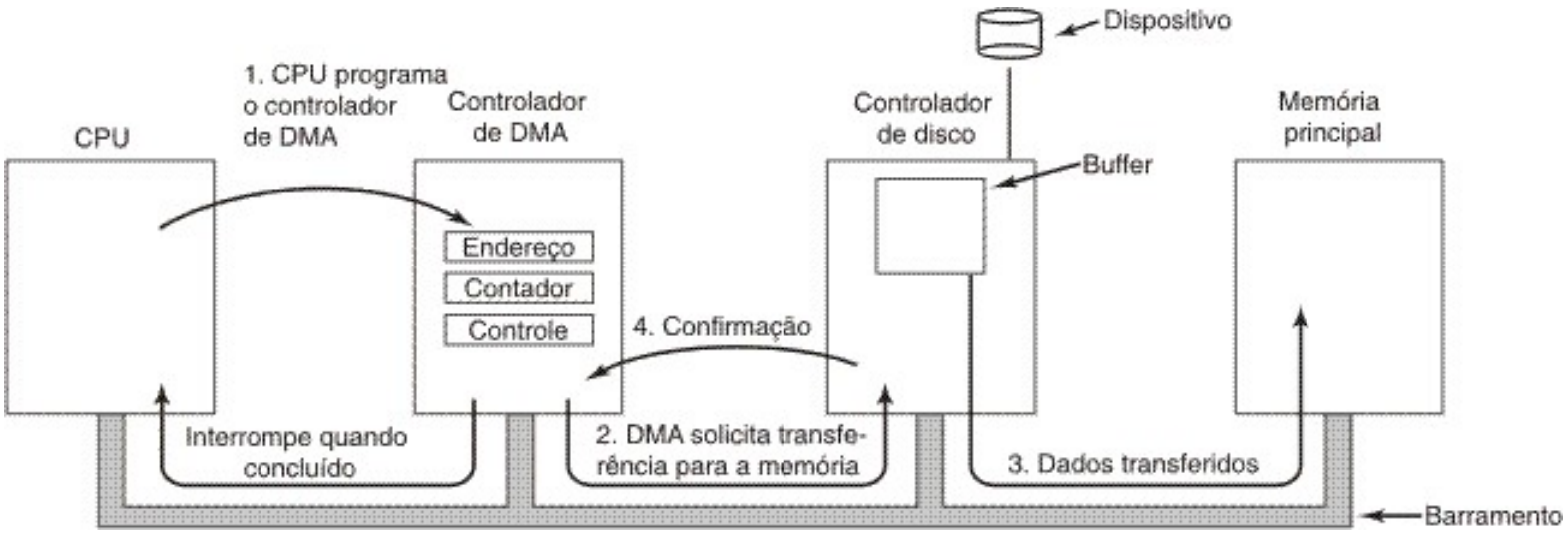


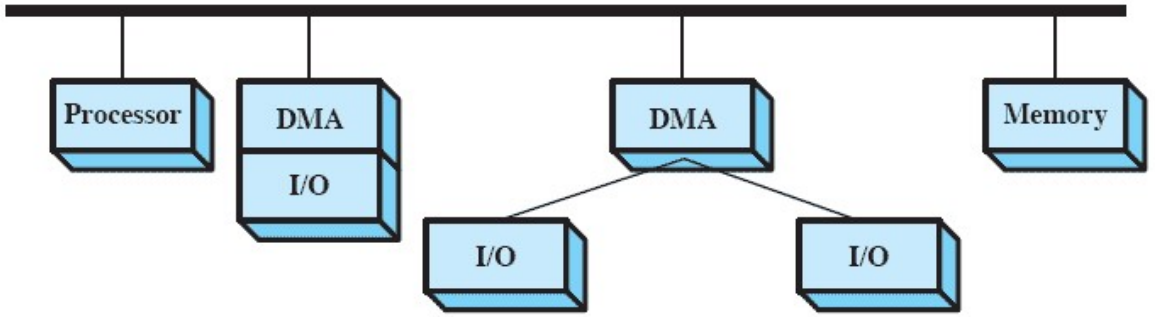
Figure 3-3. A DMA transfer is done entirely by the controller.

Acesso Direto à Memória (DMA)

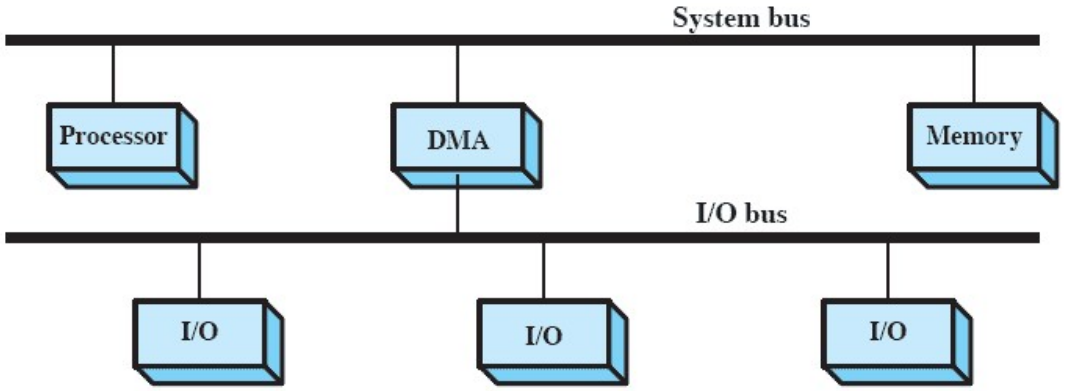


Operação de uma transferência com DMA

Configurações de uso de DMA

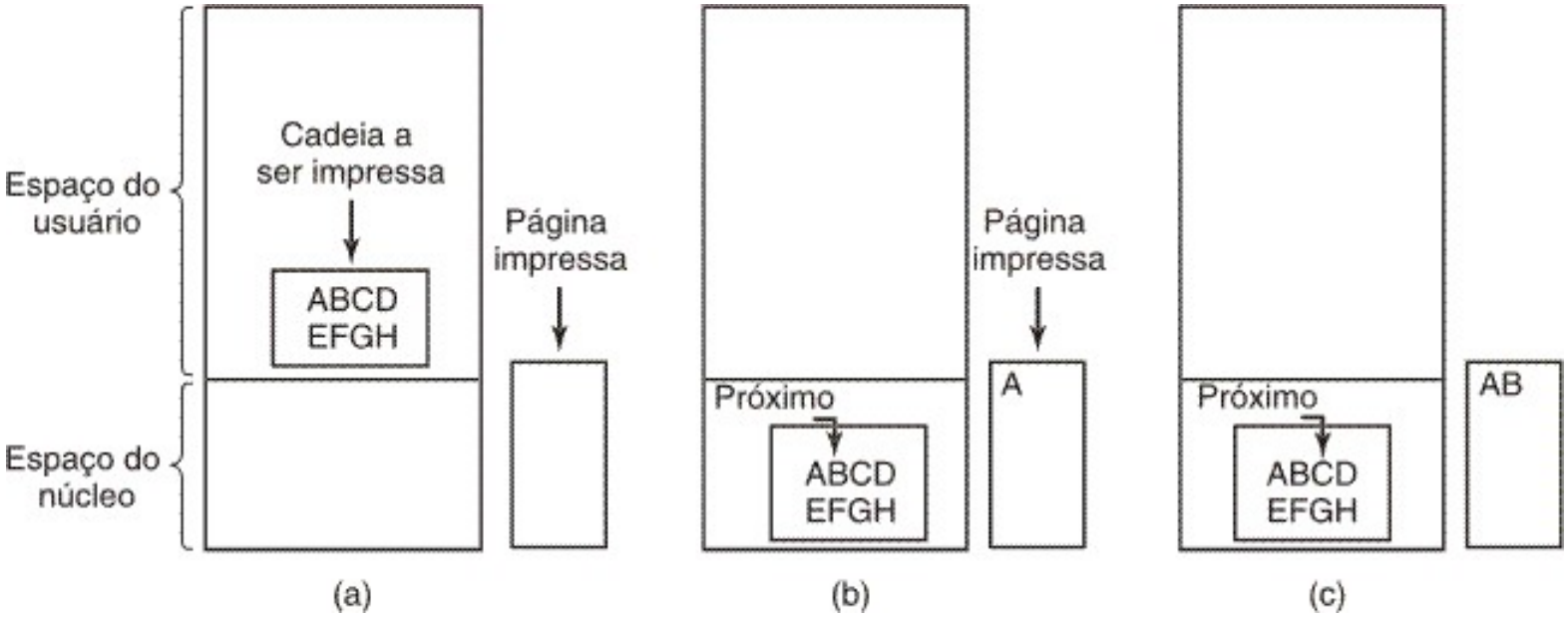


(b) Single-bus, Integrated DMA-I/O



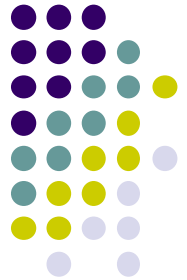
(c) I/O bus

Saída (Escrita em Impressora)



Passos da impressão de uma cadeia de caracteres

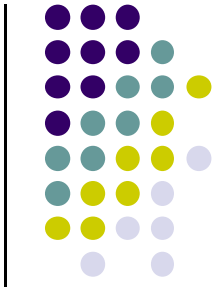
E/S Programada



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

Escrita de uma cadeia de caracteres para a impressora usando E/S programada

E/S Orientada à Interrupção



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

(a)

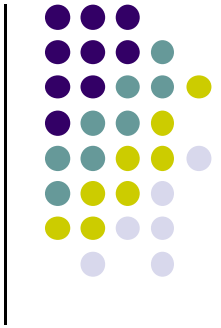
```
if (count == 0) {
    unblock_user( );
} else {
    *printer_data_register = p[i];
    count = count – 1;
     i = i + 1;
}
acknowledge_interrupt( );
return_from_interrupt( );
```

(b)

Escrita de uma cadeia de caracteres para a impressora usando E/S orientada à interrupção

1) Código executado quando é feita a chamada ao sistema para impressão
2) Rotina de tratamento de interrupção

E/S Usando DMA



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

(a)

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

(b)

Impressão de uma cadeia de caracteres usando DMA

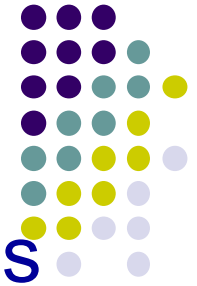
a) **Código executado quando quando é feita a chamada ao sistema para impressão**

b) **Rotina de tratamento de interrupção**

Princípios do Software de E/S
Objetivos do Software de E/S (1)

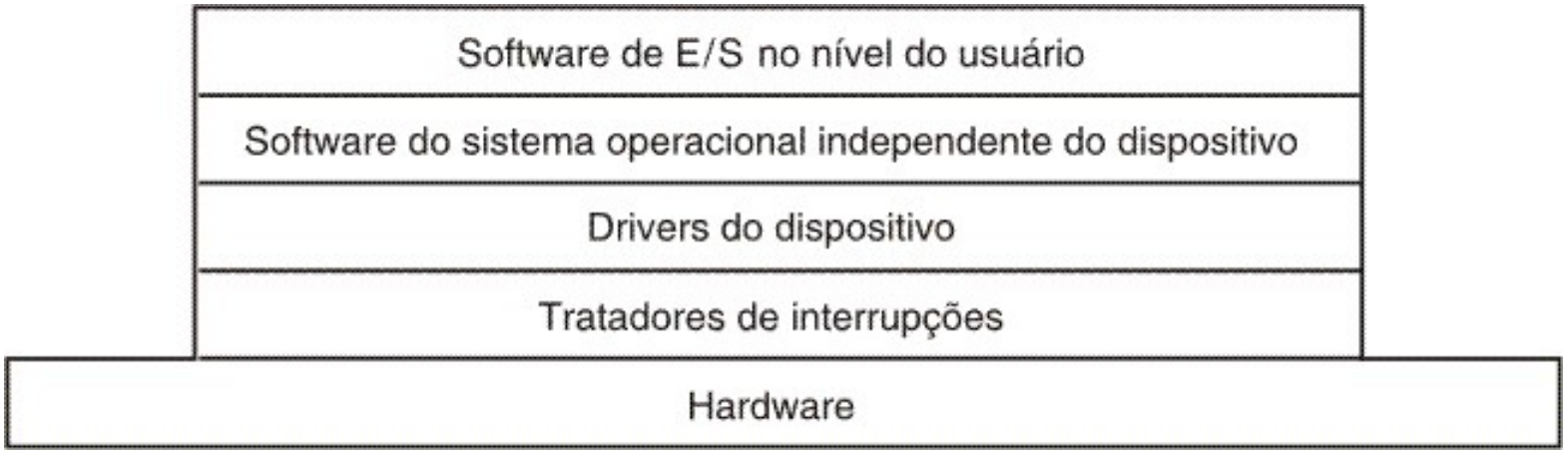
- Independência de dispositivo
  - Programas podem acessar qualquer dispositivo de E/S sem especificar previamente qual (disquete, disco rígido ou CD-ROM)
- Nomeação uniforme
  - Nome de um arquivo ou dispositivo pode ser uma cadeia de caracteres ou um número inteiro que é independente do dispositivo
- Tratamento de erro
  - Trata o mais próximo possível do hardware

Objetivos do Software de E/S (2)



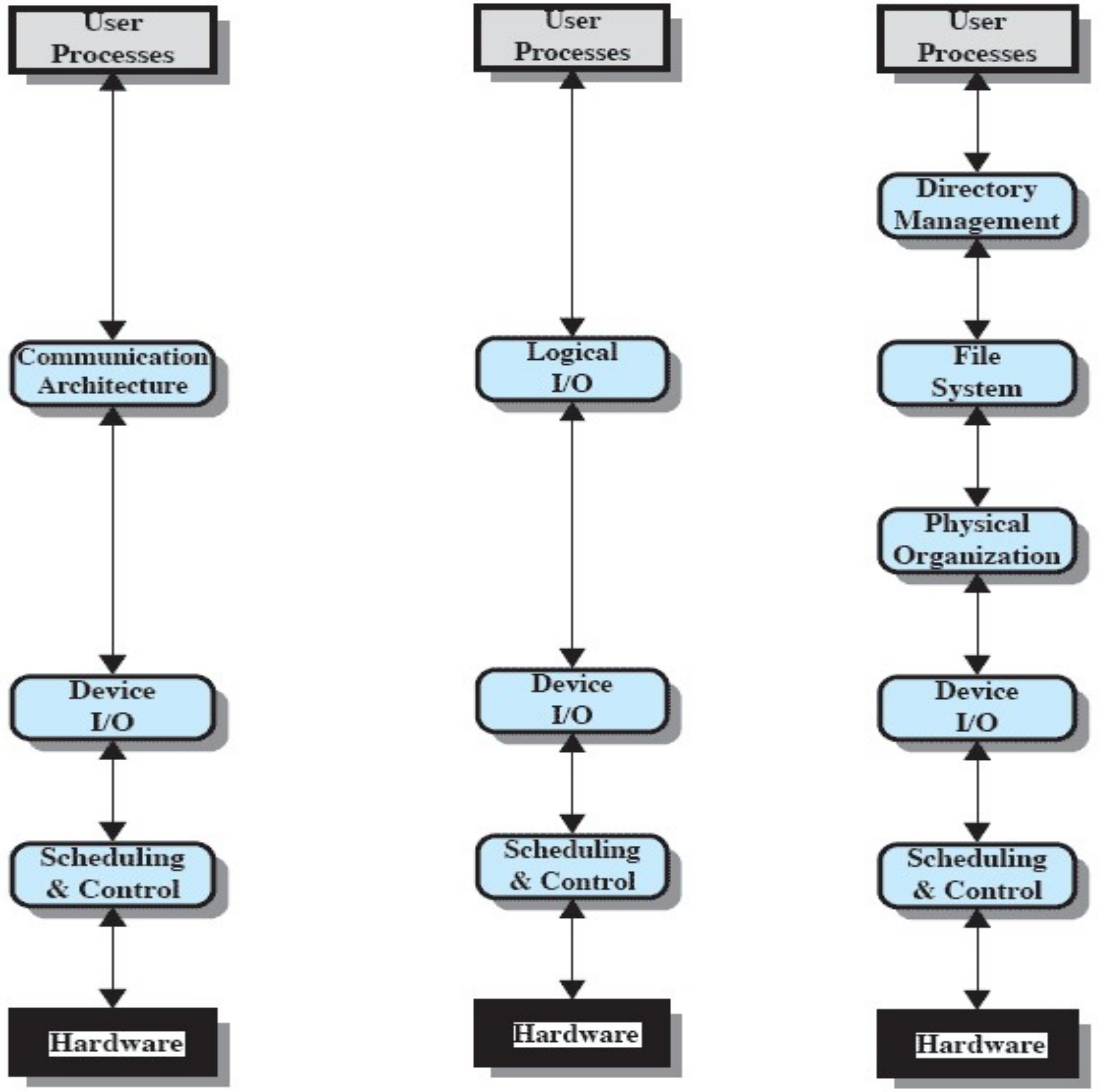
- Transferências Síncronas vs. Assíncronas
  - transferências bloqueantes vs. orientadas a interrupção
  - utilização de buffers para armazenamento temporário
  - dados provenientes de um dispositivo muitas vezes não podem ser armazenados diretamente em seu destino final
- Gerenciar o acesso concorrente (e otimizado) a dispositivos compartilhados
  - Disco, monitor, rede são disp. E/S compartilhados
  - unidades de fita não são

Camadas do Software de E/S



Camadas do sistema de software de E/S

Modelo de Operação de I/O



(b) Communications port

(a) Local peripheral device

(c) File system

Tratadores de Interrupção (1)

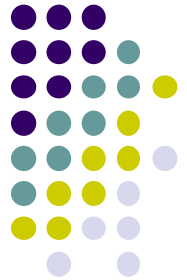
- As interrupções devem ser escondidas o máximo possível
  - uma forma de fazer isso é bloqueando o driver que iniciou uma operação de E/S até que uma interrupção notifique que a E/S foi completada
- Rotina de tratamento de interrupção cumpre sua tarefa
  - e então desbloqueia o driver que a chamou

Tratadores de Interrupção (2)

Passos que devem ser executados em software depois da interrupção ter sido concluída

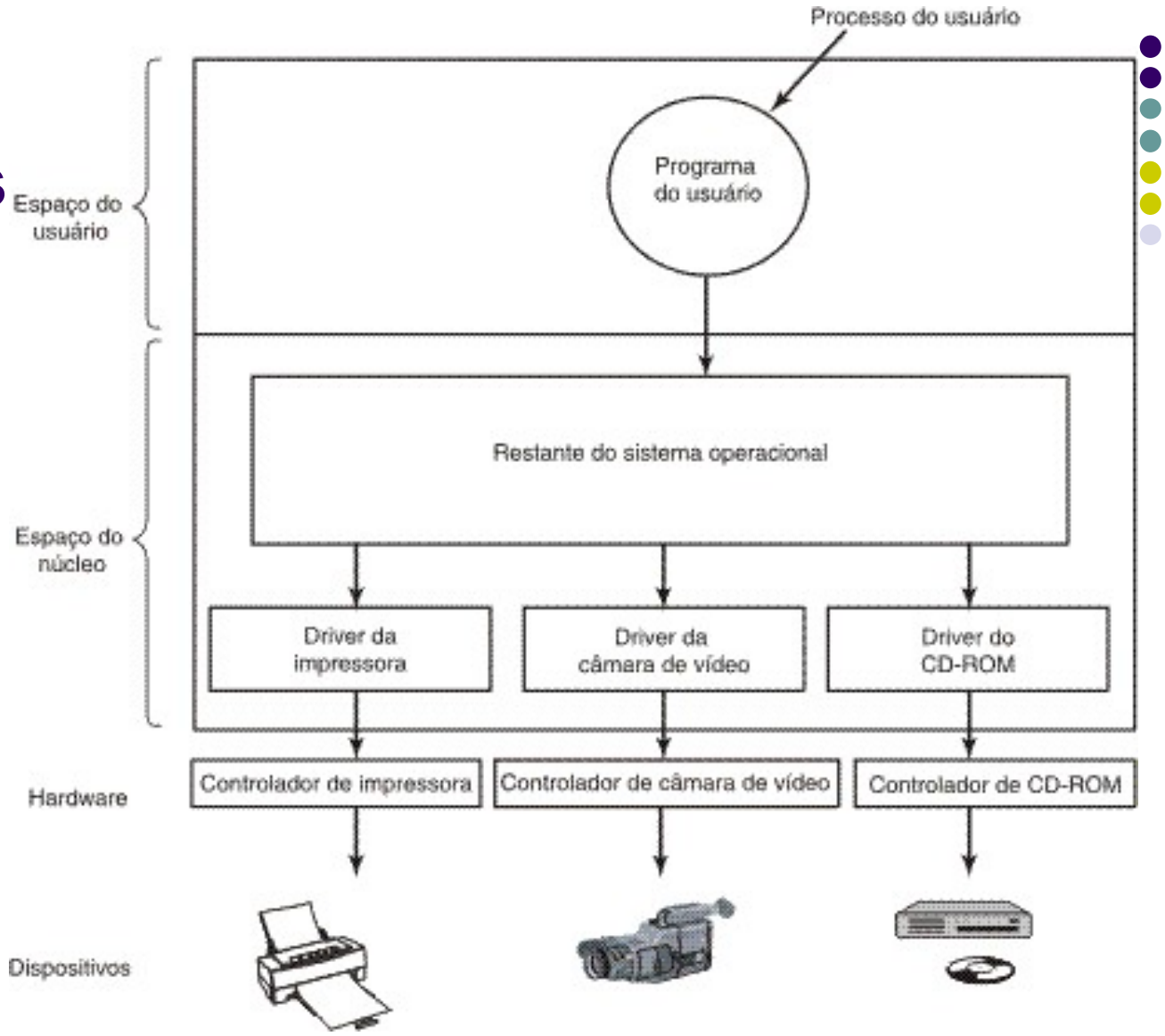
1. salva registradores que ainda não foram salvos pelo hardware de interrupção
2. estabelece contexto para rotina de tratamento de interrupção
3. estabelece uma pilha para a rotina de tratamento de interrupção
4. sinaliza o controlador de interrupção, reabilita as interrupções
5. copia os registradores de onde eles foram salvos
6. executa rotina de tratamento de interrupção
7. escolhe o próximo processo a executar
8. estabelece o contexto da MMU para o próximo processo a executar
9. carrega os registradores do novo processo
10. começa a executar o novo processo

Drivers de Dispositivo



- Driver de dispositivo esconde as especificidades dos controladores do restante do subsistema de E/S
- Muitas vezes, escritos em assembly
- Dispositivos de E/S podem variar em vários aspectos
  - Orientados a bloco X orientados a caracteres
  - Acesso Sequencial ou aleatório
  - Uso concorrente ou dedicado
  - Taxa de transferência de dados
  - Leitura-escrita, só leitura, só escrita
- Chamadas de sistema para E/S refletem o comportamento em categorias gerais
- Para dispositivo de bloco
  - `read(), write(), seek(),`
- Para dispositivos de caracteres
  - `get(), put()`

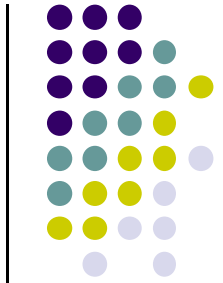
Drivers dos Dispositivos



A comunicação entre os drivers e os controladores de dispositivos é feita por meio do barramento

Drivers escrevem e lêm as portas de E/S (comandos específicos, variáveis de controle e estado do dispositivo)

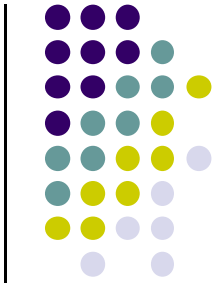
Drivers de Dispositivo



Características/Tarefas dos Drivers:

- Código para controle básico do controlador do dispositivo
- Para alguns dispositivos mantém uma fila de requisições pendentes
- Trata requisições concorrentes (por vários processos)
- Para dispositivos de bloco traduz requisições lógicas (*leia bloco x*) em comandos específicos (p.exemplo: *mova para cilindro 3; leia a transfira dados dos setores 34-35 em trilha 6*)
- Espera a chegada de interrupção, verifica integridade dos dados e transfere dados para uma região de memória

Software de E/S
Independente de Dispositivo



Implementa funções comuns a todos as formas de E/S e fornece uma interface uniforme para as aplicações.

Funções típicas do software de E/S independente de dipositivo

API uniforme para os drivers dos dispositivos
Armazenamento em buffer
Relatório dos erros
Alocação e liberação de dispositivos dedicados
Define tamanho de bloco único, independente de dispositivo

API e nomes uniformes

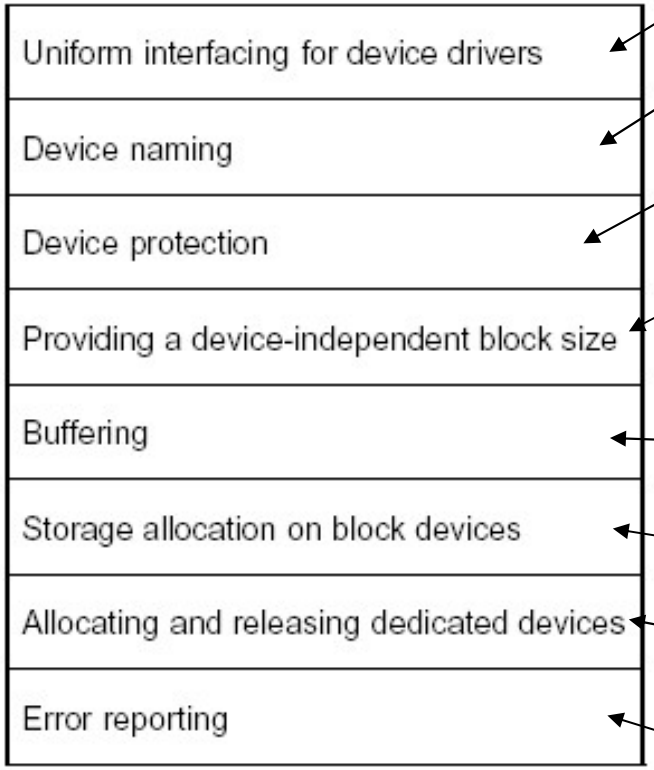
- Com uma API padrão para drivers é mais fácil instalar novos drivers
- Referência uniforme de arquivos e dispositivos facilita o uso:
  - Nomes simbólicos de dispositivos são mapeados para o driver correspondente

i-node de *dev/disk0* contém:
• Numero principal do dispositivo = localização do driver
• Numero secundário do disp. = a unidade de disco (parâmetro do driver)

Bits de controle de acesso (rwx) também definem as permissões de acesso ao dispositivo

Software Independente do Dispositivo

Principais Funções:



Exemplos:
`get(), put()`

Mapeamento de nomes simbólicos para os drivers correspondentes (/dev/tty00) .

Permissões para acesso aos dispositivos por programas em espaço usuário.

Mapeamento de tamanho de bloco único para tamanhos de setores variados em discos diferentes.

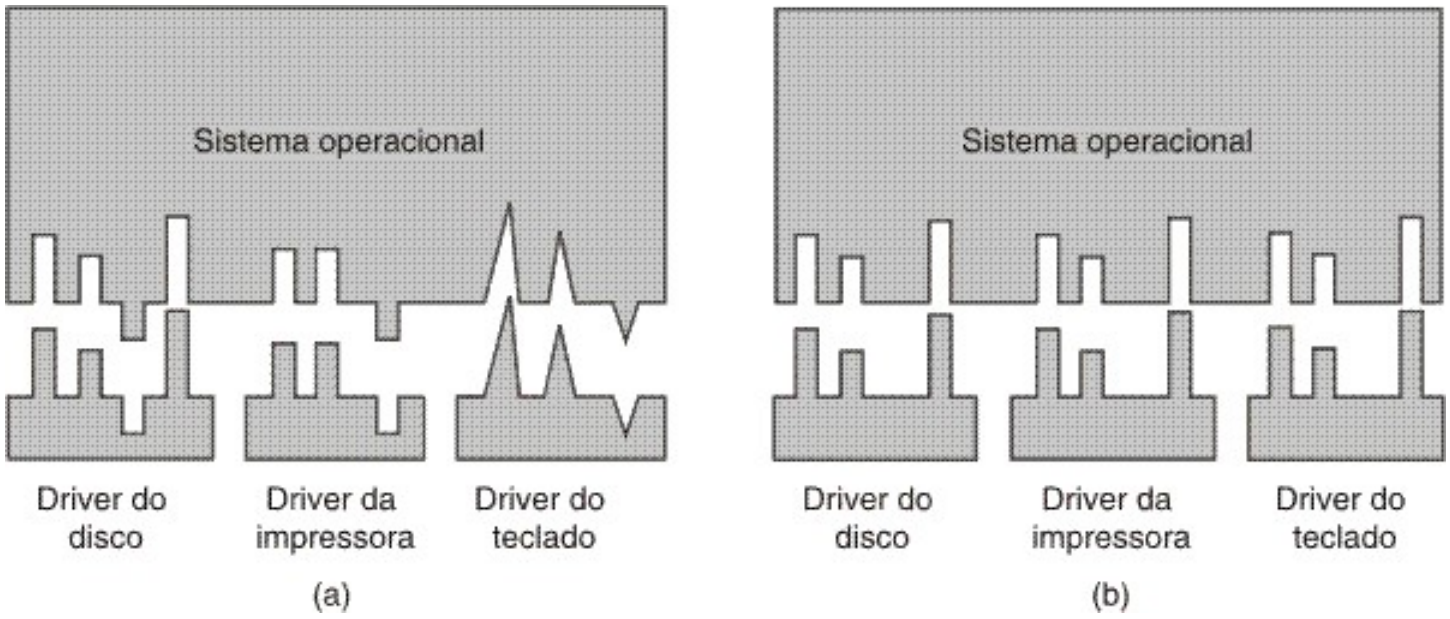
Transferência controlada de bytes do disco para interface de rede, compensando diferentes taxas de dados de cada dispositivo

Gerenciamento de blocos livres nos dispositivos de bloco (discos)

Funções para a alocação e liberação de certos dispositivos (p.ex. fita magnética)

Se erro não pode ser compensado/contornado pelo driver, deve informar ao programa do usuário e evitar a sua futura ocorrência (marcando "blocos com problema")

Software de E/S
Independente de Dispositivo (2)



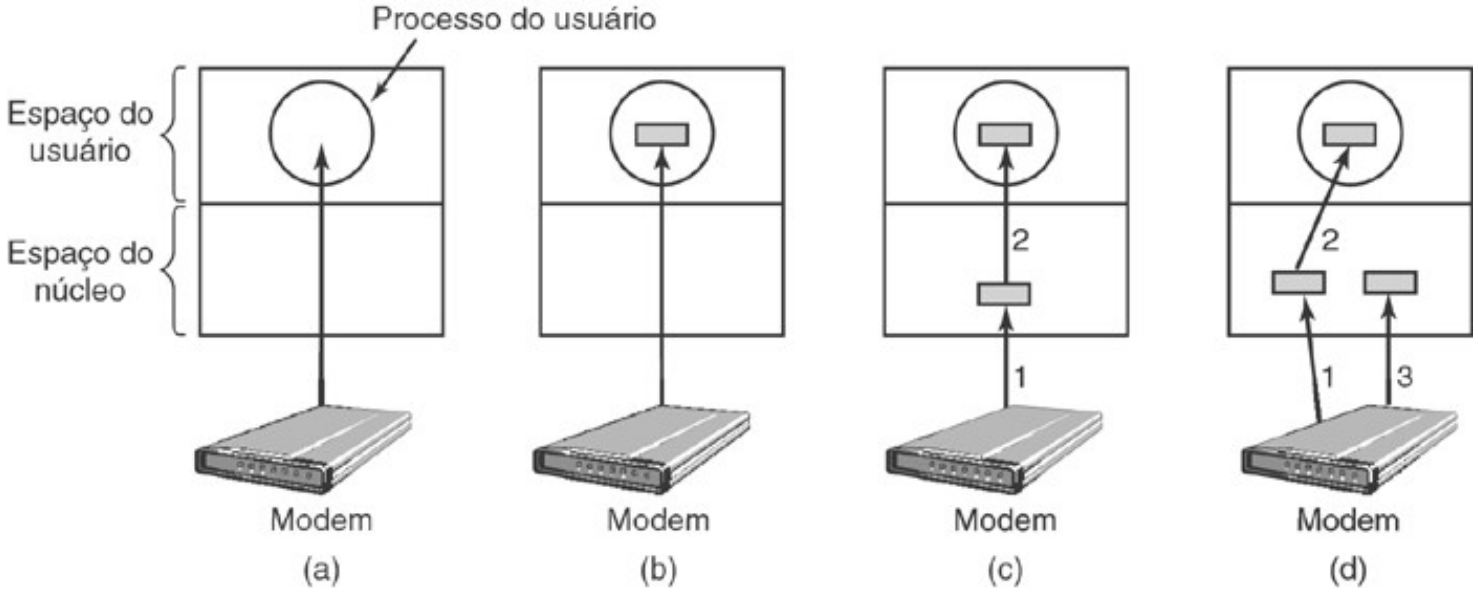
(a) Sem uma interface-padrão do driver

(b) Com uma interface-padrão do driver

Software de E/S
Independente de Dispositivo

Bufferização:

- **para compensar taxas de transferência**
- **para impedir acesso direto**

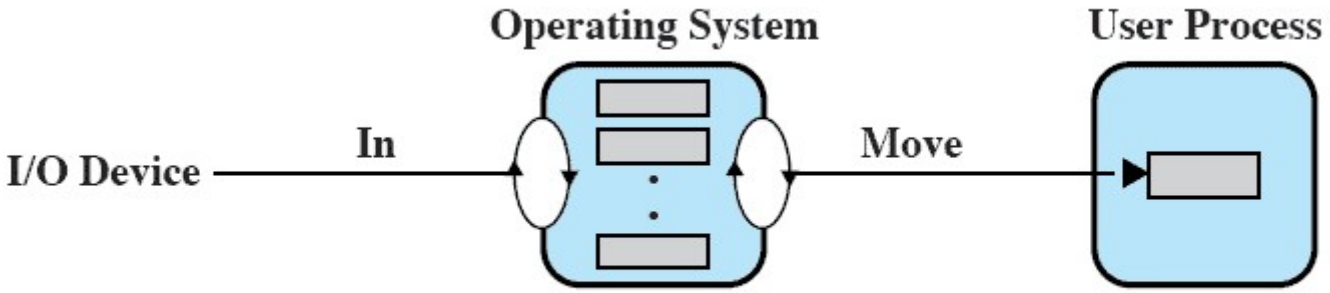


a) **Entrada sem utilização de buffer**
b) **Utilização de buffer no espaço do usuário**
c) **Utilização de buffer no núcleo seguido de cópia para o espaço do usuário**
d) **Utilização de buffer duplo no núcleo**

Obs: só c) e d) permitem multiprogramação!

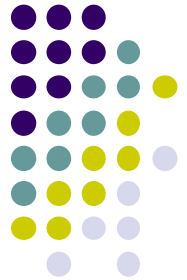
Buffer Circular

- É a generalização do buffer duplo
- Torna a taxa de E/S independente da taxa de consumo dos dados pelo processo destinatário
- Usado para fazer a busca antecipada de blocos (provável necessidade de blocos vizinhos)
- Permite atender requisições de leitura de vários processos



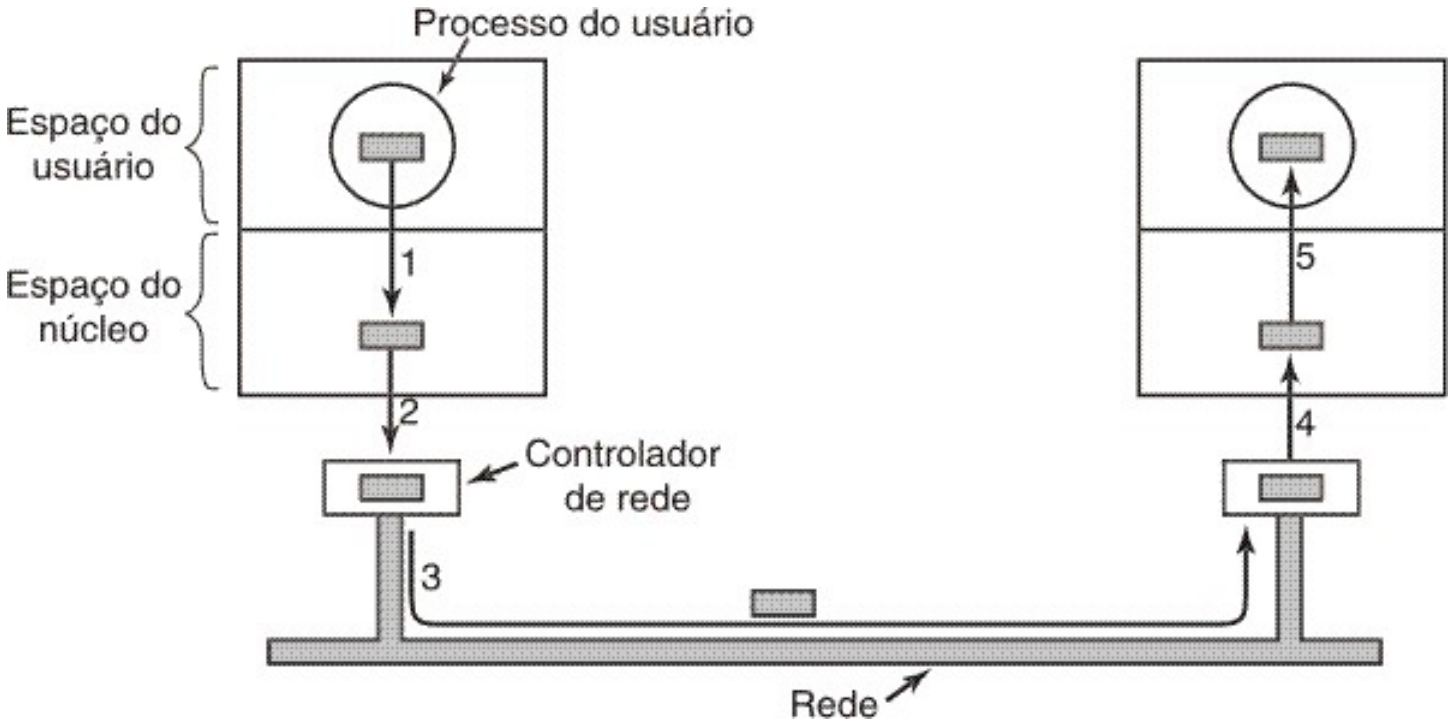
(d) Circular buffering

E/S com buffers



- **Block-oriented**
  - Processo pode processar um bloco enquanto o outro está sendo lido para memória
  - Buffers ficam residentes em RAM (sem swapping)
  - Subsistema de E/S mantém registro da associação entre buffers e processos do usuário envolvidos na E/S
- **Stream-oriented**
  - Uma linha é lida/escrita de cada vez (com CR-Carriage return) sinalizando fim da linha
  - Os caracteres no buffer podem ser pré processados antes de serem entregues para o processo

Software de E/S
Independente de Dispositivo



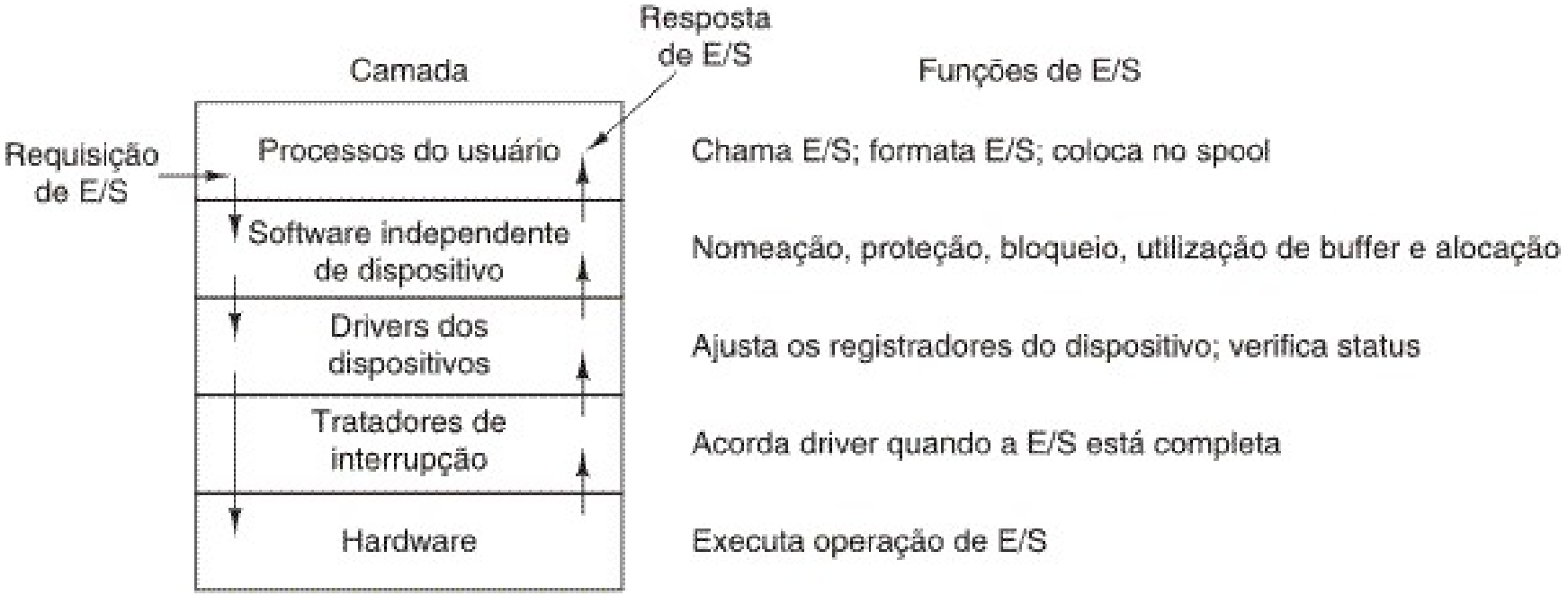
A operação em rede pode envolver muitas cópias de um pacote

Software de E/S no Espaço do Usuário

São processos (daemons) responsáveis por efetuar a E/S

Exemplos:

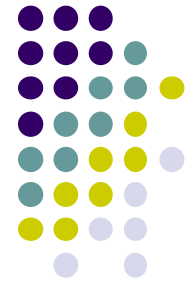
- lpd - spool de arquivos para a impressão
- inetd, ftpd, rshd, httpd, dhcpd - processos que tratam E/S com a rede



Algoritmos de Escalonamento de Acesso ao Disco

- O Tempo necessário para ler ou escrever um bloco de disco é dominado pelo tempo de posicionamento (seek time)
- A fim de minimizar o tempo de posicionamento para várias requisições, o driver deve escalonar as requisições de acordo com a trilha alvo
- A verificação do código de redundância (bits de paridade) é feita por controladores

Algoritmos de Escalonamento de Disco

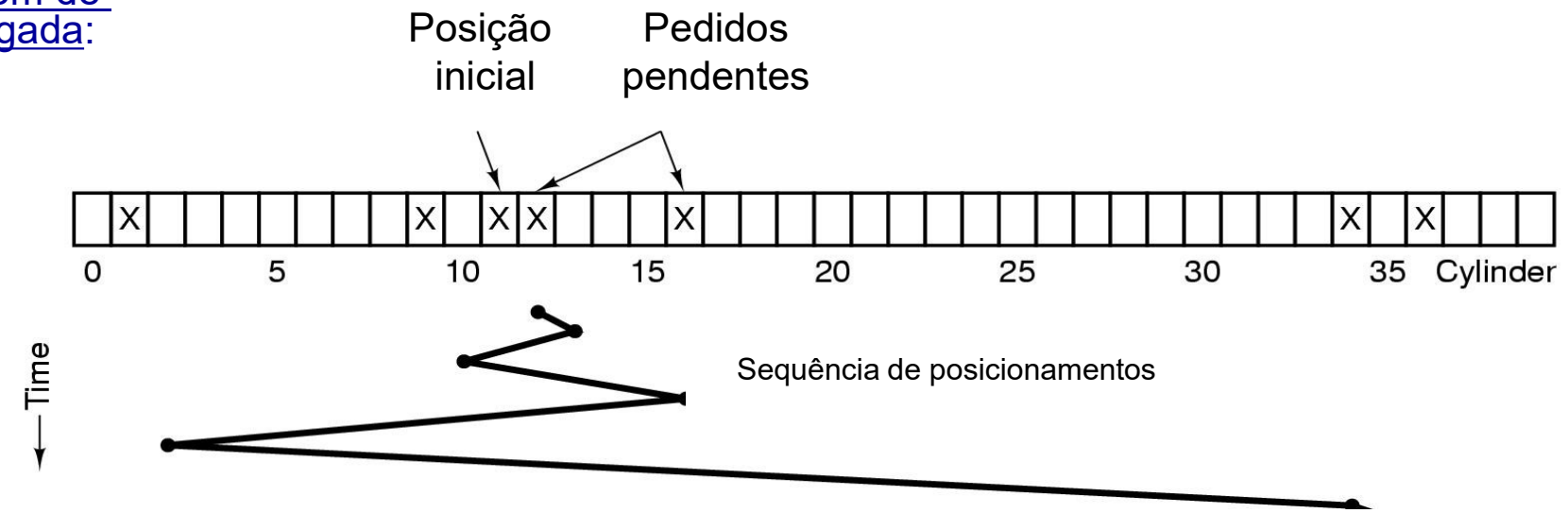


1) First-Come-First Served (FCFS): não há como otimizar tempo de posicionamento (seek time).

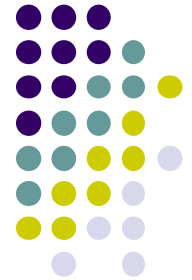
2) Algoritmo *Primeiro-Mais-Próximo* (Shortest Seek First) :

- **Usa tabela indexada por cilindro com lista de pedidos para cada cilindro**
- **Enquanto move o cabeçote para um cilindro, novos pedidos vão chegando**
- **Escolhe sempre o pedido para o cilindro mais próximo do corrente**
- **Atende todos os pedidos para o cilindro corrente**

Ordem de Chegada:
11
1
36
16
34
9
12

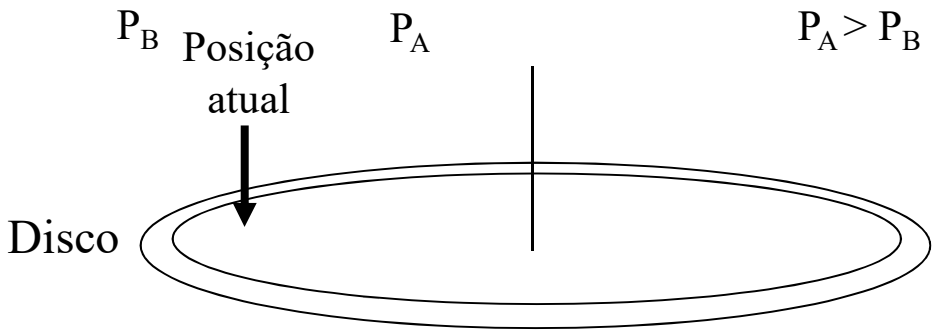


Algoritmos de Escalonamento de Disco

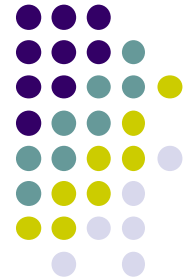


Principal problema do *Shortest Seek First*:

- **Assumindo uma distribuição uniforme dos pedidos em todas as trilhas, então:**
- **Para discos muito utilizados, o cabeçote tenderá a permanecer nos cilindros do meio, e mover-se com menor probabilidade para os cilindros nos extremos.**
- **Portanto, dados gravados em cilindros extremos levarão mais tempo para serem acessados do que dados em cilindros do meio.**



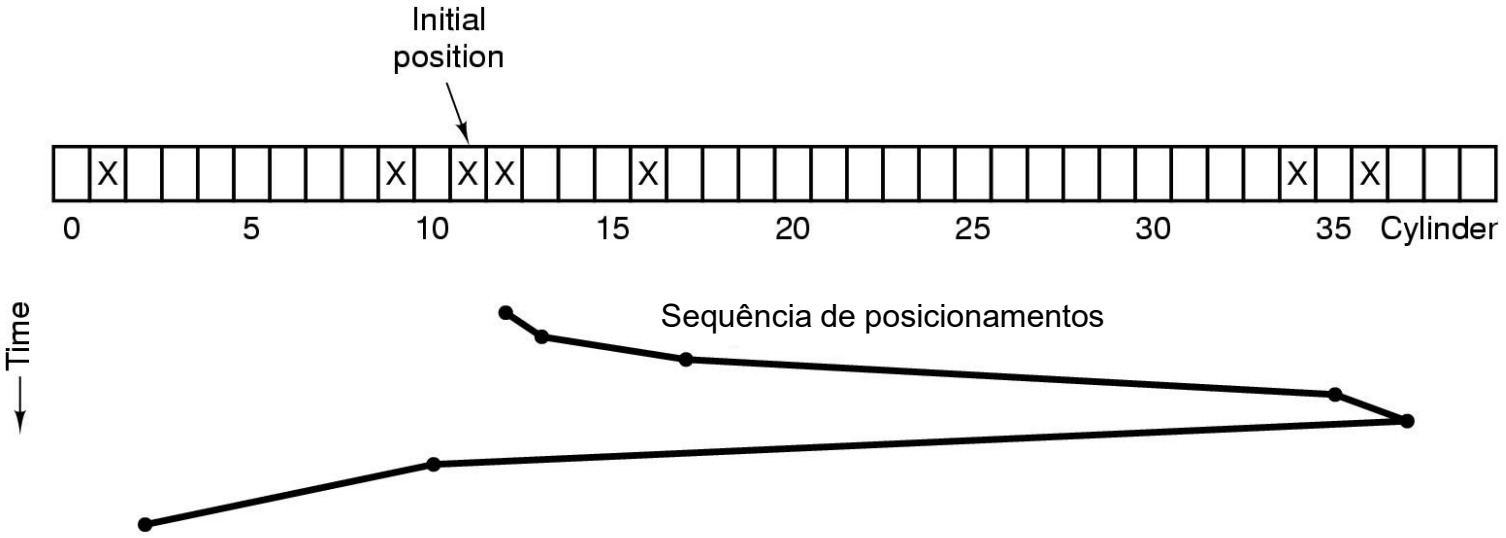
Algoritmos de Escalonamento de Disco



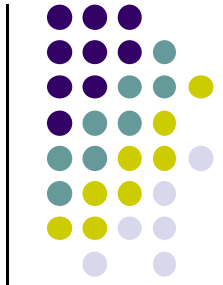
Algoritmo do elevador:

- Manter o sentido da movimentação até que não haja mais pedidos para acesso em "cilindros adiante"
- No sentido de movimentação, atende primeiro os cilindros mais próximos
- Um flag (UP/DOWN) registra o sentido de movimentação.

Ordem chegada:
11
1
36
16
34
9
12



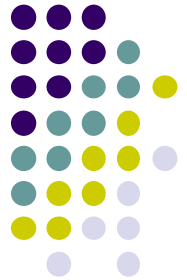
Algoritmo do Elevador



Variantes:

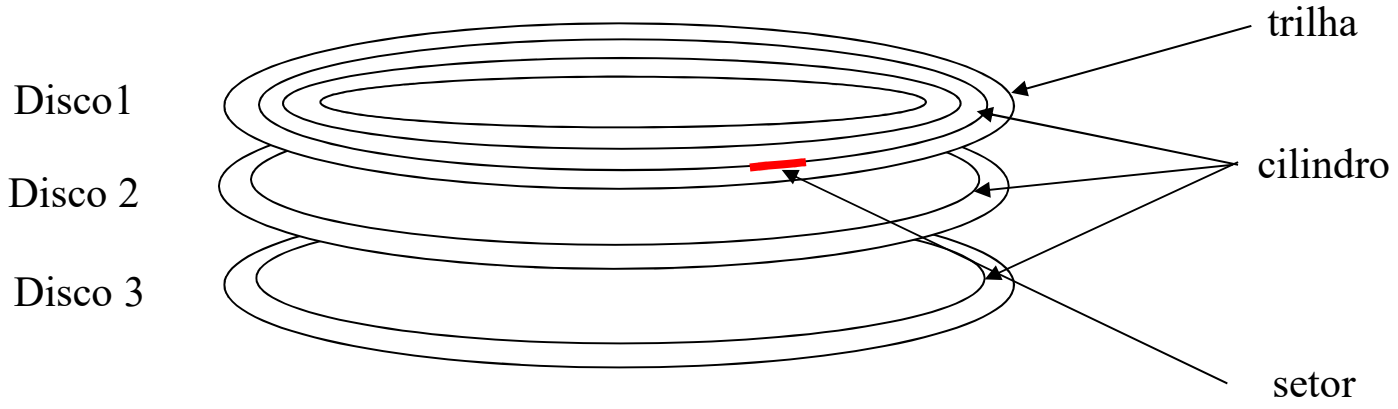
- Mover o cabeçote somente até o cilindro mais afastado para o qual exista uma requisição
  - Vantagem: ganha-se eficiência no atendimento global das requisições
- Mover o cabeçote até o cilindro mais afastado, independente de haver requisição
  - Vantagem: garante-se um tempo médio igual de atendimento para requisições nos cilindros centrais e extremos do disco
  - Desvantagem: se nunca houve escritas em cilindros além dos limites, é improvável que aconteçam requisições para lá.
- Guardar quais foram os cilindros mais extremos usados até então (cil_{min}, cil_{max}), e fazer a varredura dentro desse intervalo.

Estrutura do Disco



- **Cilindro: coleção de N trilhas (uma em cada superfície de disco)**
- **Trilha: uma circunferência de raio T completa em uma superfície de disco; contém número variável de setores (8-32 em floppies, milhares em HD)**
- **Setor: uma parte de uma trilha (com número fixo de bytes)**

O tempo de acesso a um setor depende essencialmente do tempo de posicionamento do pente de leitores até o cilindro correspondente e da velocidade de rotação do disco.

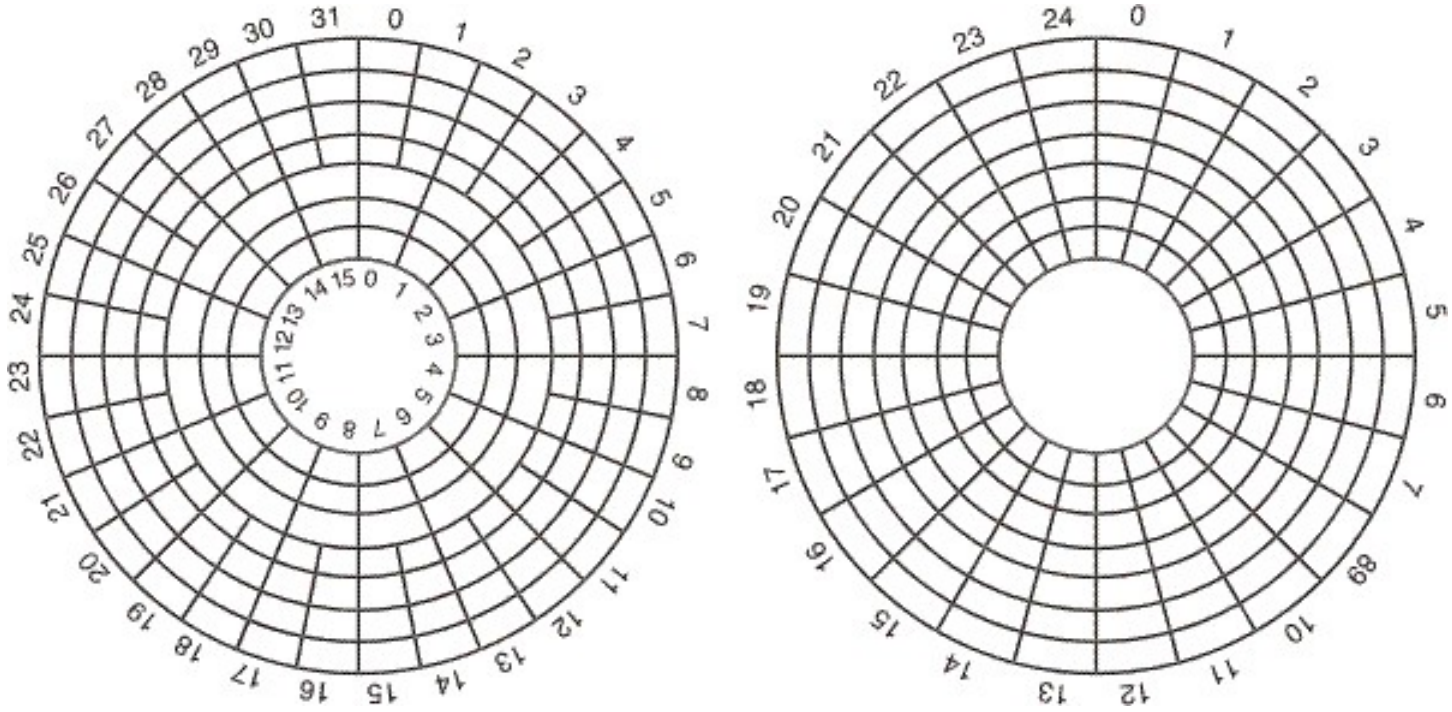


Discos
Hardware do Disco (1)

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

Parâmetros de disco para o disco flexível original do IBM PC e o disco rígido da Western Digital WD 18300

Hardware do Disco (2)



- Geometria física de um disco com duas regiões
- Uma possível geometria virtual para esse disco

Parametros de Desempenho de acesso ao Disco

O cabeçote precisa ser posicionado na trilha, no início do setor alvo. Isso envolve:

- Tempo de posicionamento (Seek time)
  - Tempo para posicionar o cabeçote na trilha
- Latência rotacional
  - Tempo necessário para que o início do setor apareça abaixo do cabeçote

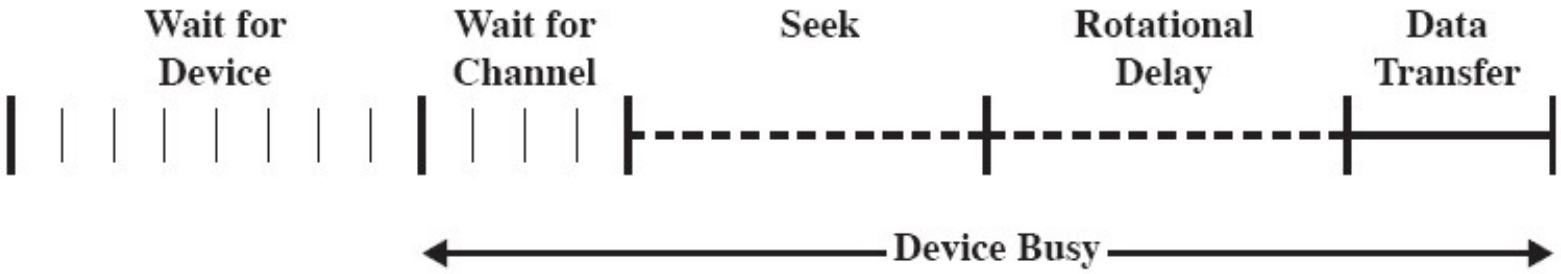
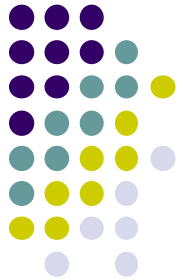


Figure 11.6 Timing of a Disk I/O Transfer

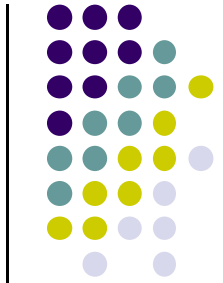
Parametros de Desempenho de acesso ao Disco

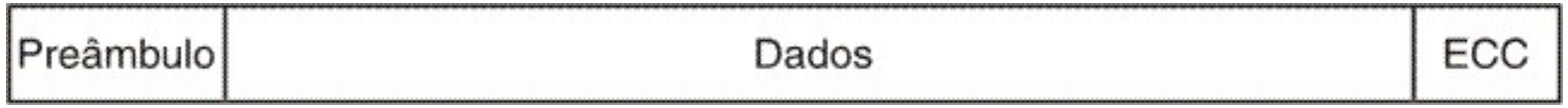


- Tempo de acesso
  - Seek time + latência rotacional

- De fato, o tempo de posicionamento é o que domina os demais
- Transferência dos dados ocorre enquanto o setor passa em baixo do cabeçote

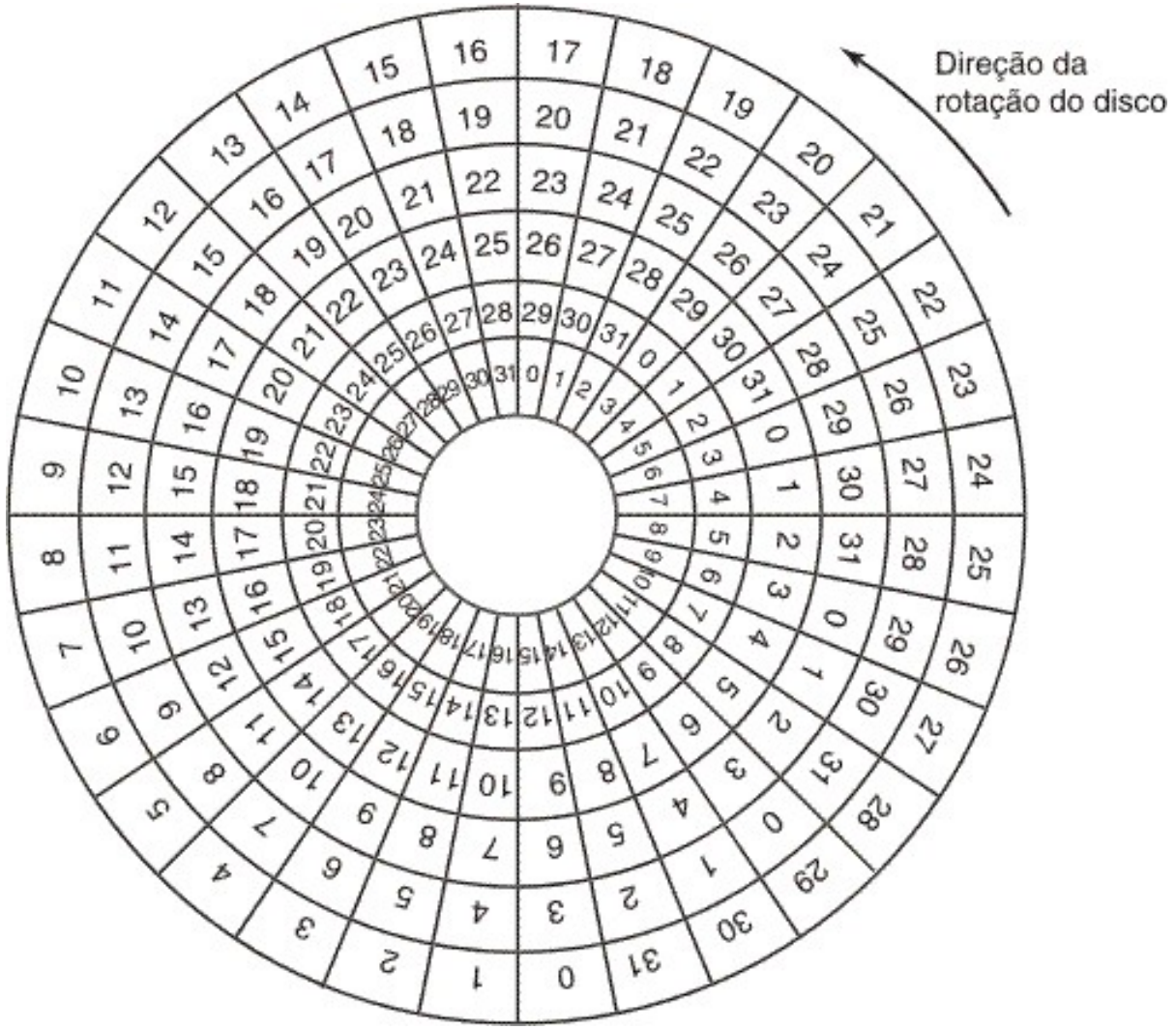
Formatação de Disco (1)





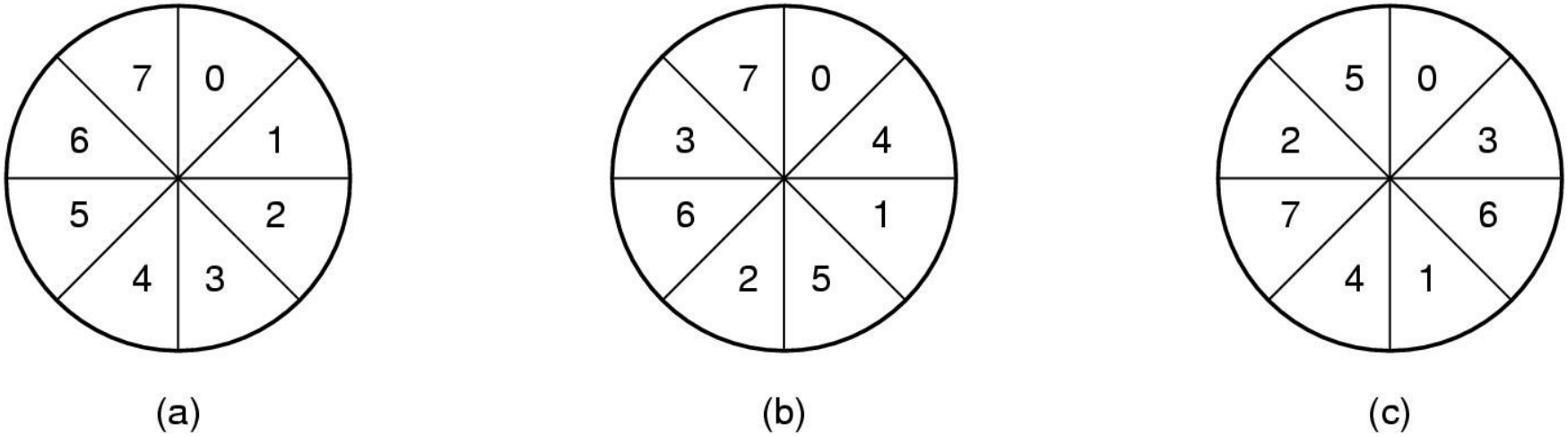
Um setor do disco

Formatação de Disco (2)



Uma ilustração da torção cilíndrica

Formatação de Disco



Disposição dos setores em uma trilha:

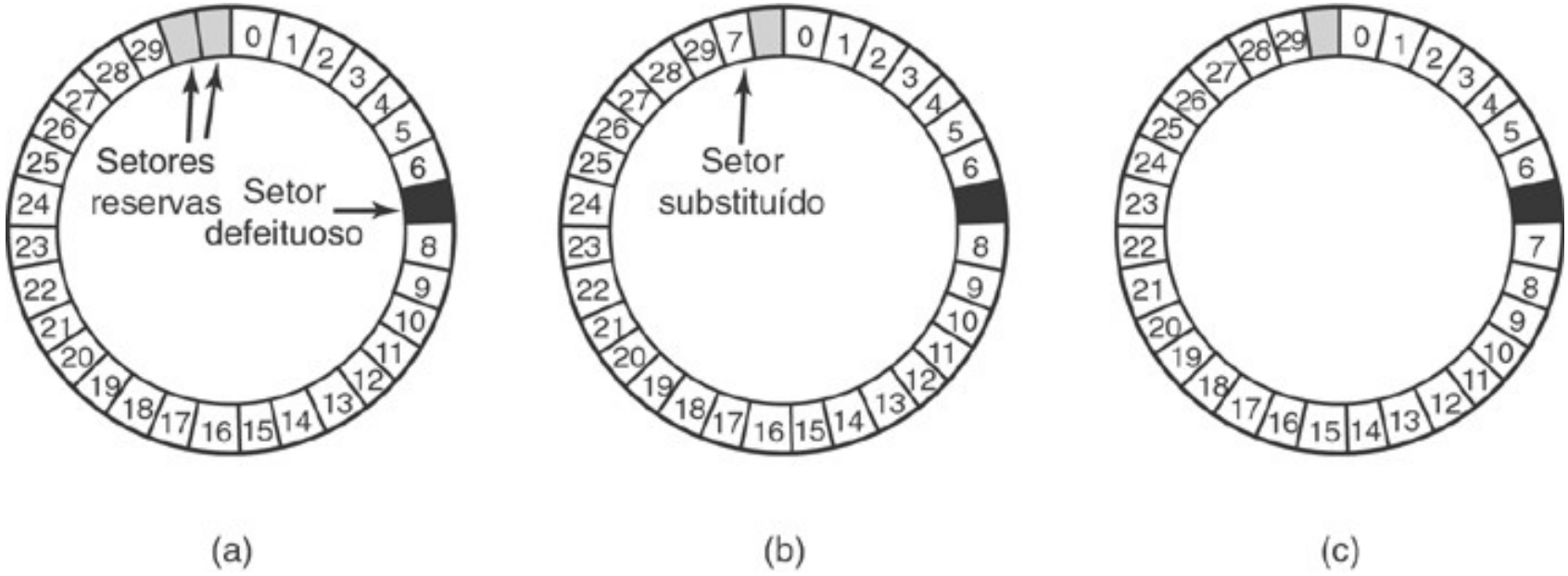
a) **Sem entrelaçamento**

b) **Entrelaçamento simples**

c) **Entrelaçamento duplo**

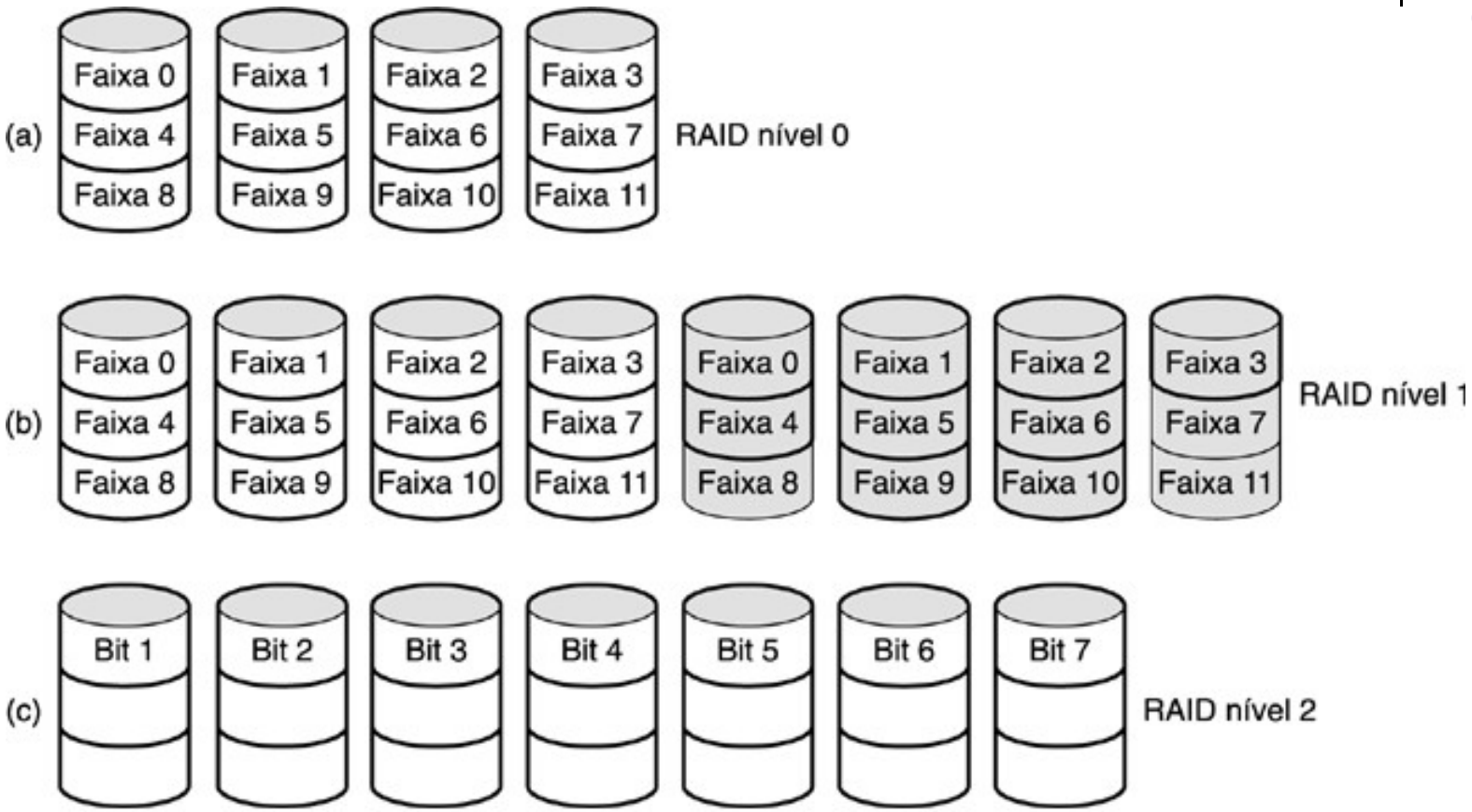
Vantagem: (b) e (c) evitam gargalos de buffers em acessos de setores consecutivos

Tratamento de Erro



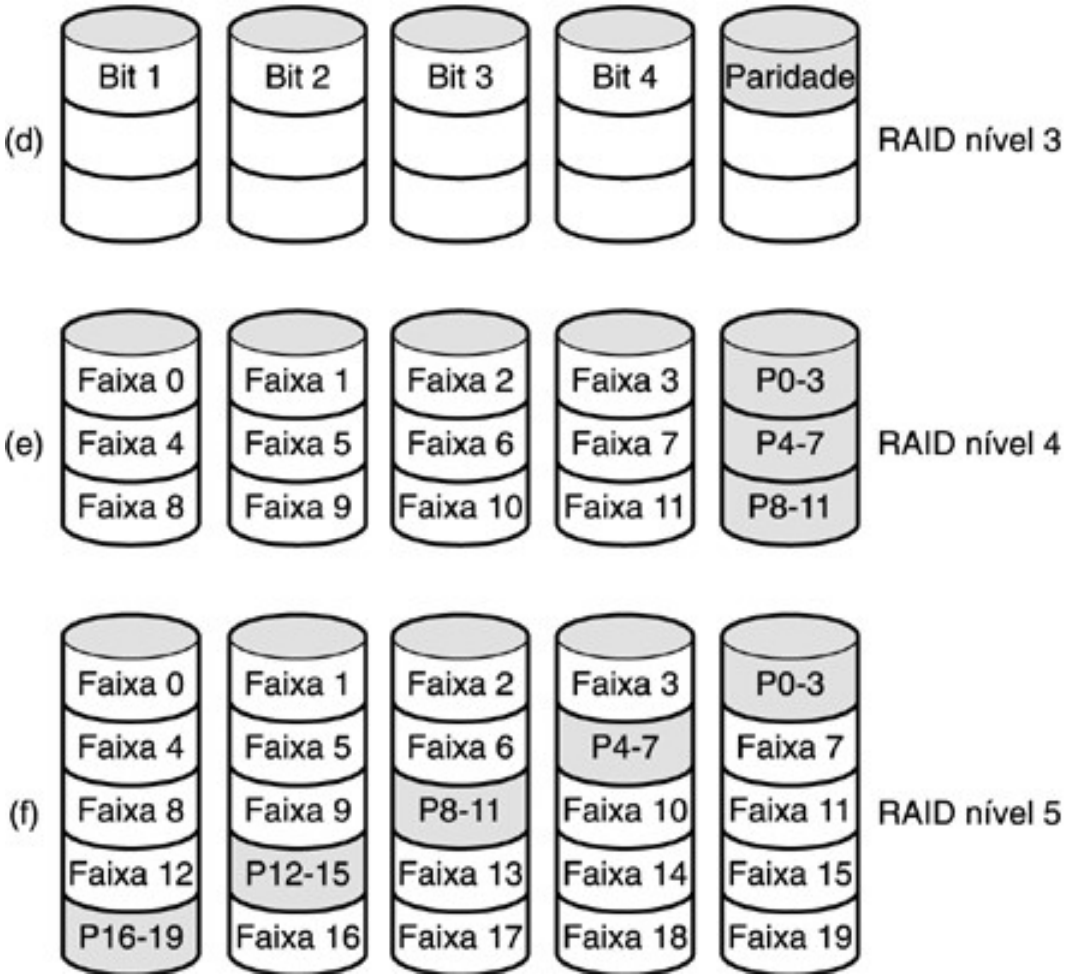
1. Uma trilha de disco com um setor defeituoso
2. Substituindo um setor reserva por um setor defeituoso
3. Deslocando todos os setores para pular o setor defeituoso

Redundância de Disco



- **RAIDs níveis 0 a 2**
- **Discos de segurança e de paridade são os sombreados**

Redundância de Disco



- RAIDs níveis 3 a 5
- Discos de segurança e de paridade são os sombreados

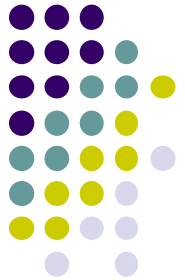
Redundância de Disco

Raid 0 - Os dados do computador são divididos entre dois ou mais discos rígidos, o que oferece uma alta performance de transferência de dados, porém não oferece segurança de dados, pois caso haja alguma pane em um disco rígido, todo o conteúdo gravado neles irá ser perdido. O RAID 0 pode ser usado para se ter uma alta performance, porém não é indicado para sistemas que necessitam de segurança de dados.

RAID1 - O RAID 1 também é conhecido como “espelhamento”, ou seja, os dados do computador são divididos e gravados em dois ou mais discos ao mesmo tempo, oferecendo, portanto, uma redundância dos dados com segurança contra falha em disco. Esse nível de RAID tende a ter uma demora maior na gravação de dados nos discos, pelo fato da replicação ocorrer entre os dois discos instalados, mas sua leitura será mais rápida, pois o sistema terá duas pontes de procura para achar os arquivos requeridos.

RAID 2 - Este nível de RAID é direcionado para uso em discos que não possuem detecção de erro de fábrica. O RAID 2 é muito pouco usado uma vez que os discos modernos já possuem de fábrica a detecção de erro no próprio disco.

Redundância de Disco



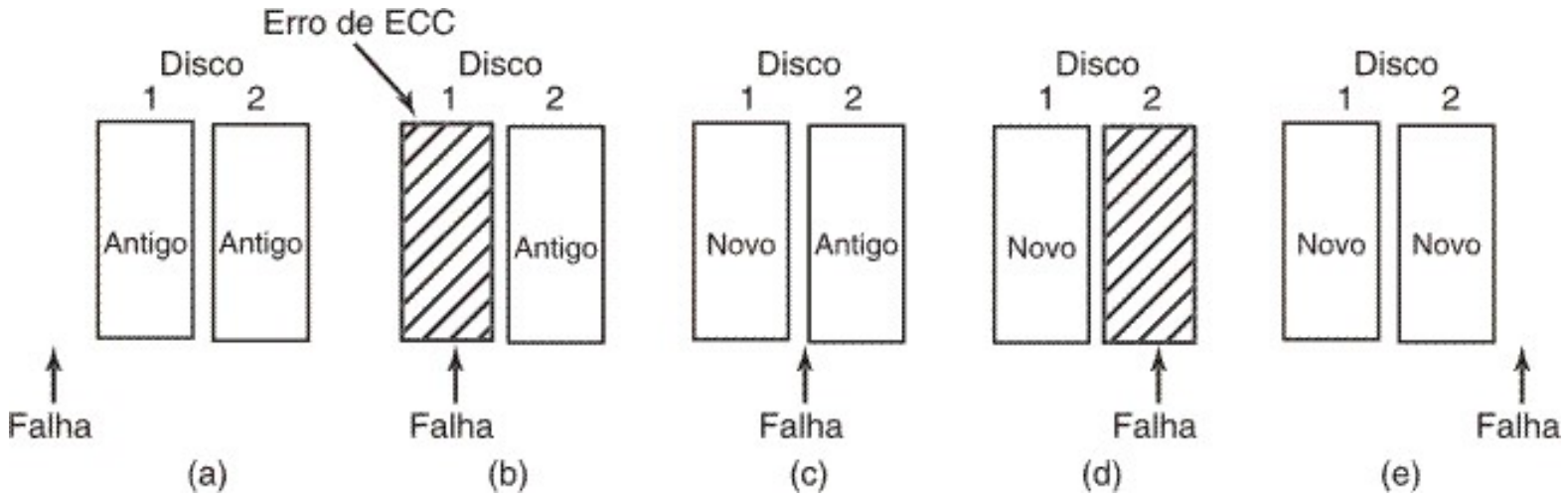
RAID 3 - O RAID 3 divide os dados, a nível de byte, entre vários discos. A paridade é gravada em um disco em separado. Para ser usado este nível, o hardware deverá possuir este tipo de suporte implementado. Ele é muito parecido com o RAID 4.

RAID 4 - O RAID 4 divide os dados, a nível de “blocos”, entre vários discos. A paridade é gravada em um disco separado.

RAID 5 - O RAID 5 é comparável ao RAID 4, mas ao invés de gravar a paridade em um disco separado, a gravação é distribuída entre os discos instalados.

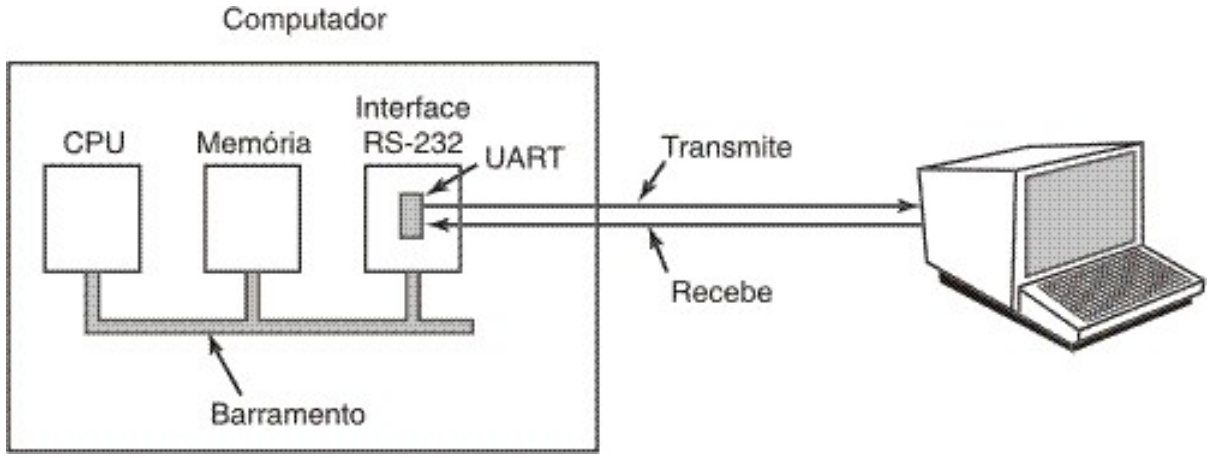
Existem outros RAID que são utilizados em menor escala e/ou são baseados naquele acima mencionados, por exemplo: RAID 6 (dupla paridade)
É essencialmente uma extensão do RAID 5 com dupla paridade, etc.

Armazenamento estável



Análise da influência das falhas nas escritas estáveis

E/S baseada em caracteres
Hardware do Terminal RS-232

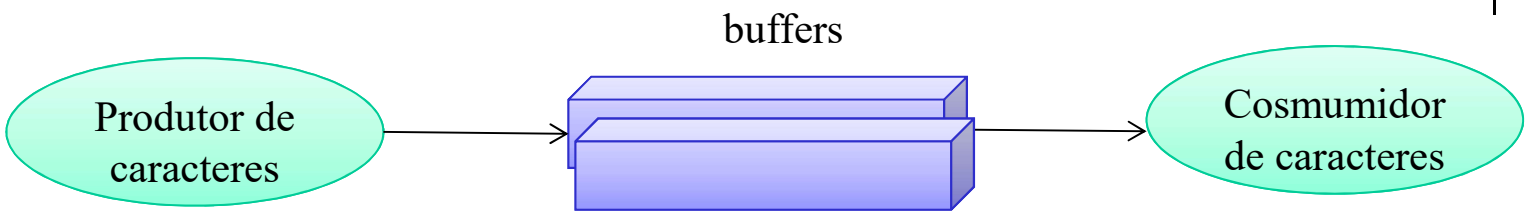


- **RS-232 / interface serial / Modem / terminais / tty / COM1/COM2 (Windows)**
- **Bit serial:**
  - **(conectores de 9- a 25-pinos), apenas 3 linhas**
  - **os bits são transferidos em sequência, 1 bit por vez**
- **Computador e terminal são completamente independentes**

E/S Orientada a caracteres

- As requisições são para obtenção (ou escrita) caracter a caracter (e.g. put(c) e get(c))
- Para compensar diferentes taxas de produção e consumo de caracteres, utiliza-se buffers
- Vários caracteres possuem um significado especial (caracteres de controle), que precisam ser interpretados ou gerados
- Para alguns dispositivos, esses caracteres devem ser interpretados/ gerados nos buffers, antes de serem enviados serem consumidos.

E/S Orientada a caracteres



Caracter = byte codificando um evento/comando de E/S

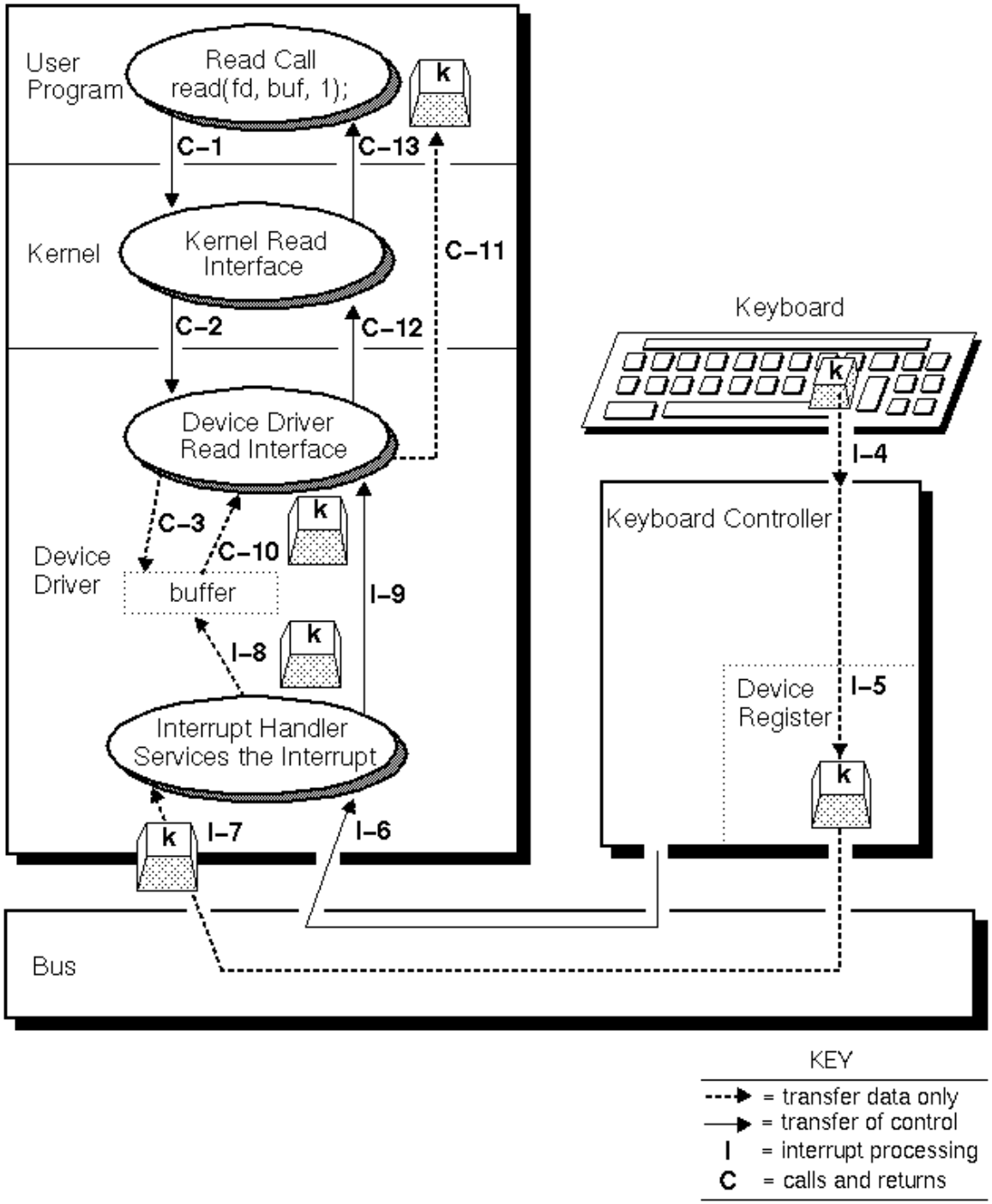
Exemplos de pares Produtor/Consumidor:

- Entrada:
  - (teclado/ programa_usuário),
  - (interface_rede/ programa_usuário)
- Saída:
  - (Programa_usuário, monitor)
  - (Programa _usuário, interface_rede)

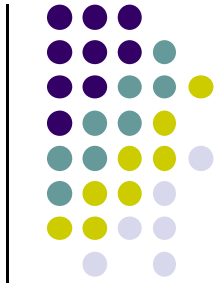
Funcionamento de uma entrada de dados de um teclado

- Controlador gera uma interrupção para avisar que tecla foi lida
- Driver copia um número (código de tecla pressionada/solta) da controladora para buffer interno
  - driver converte caracteres para código ASCII
  - muitos SOs fornecem mapas de teclas ou páginas de códigos carregáveis
- Núcleo recebe uma requisição de leitura de um processo, e repassa para driver

Etapas da Leitura de uma tecla

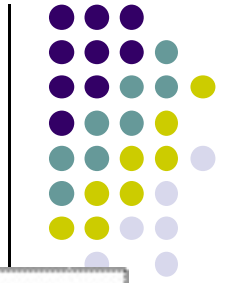


Software de Entrada



- Processamento de caracteres
  - Usuário digita "hella←o"
  - Teclado gera: "hella←_←o"
  - Processo recebe: "hello"
- Modo crú (raw mode)
  - Driver entrega todos os caracteres para o processo (incl. teclas ctrl, alt, f1-f10, alt, shift,…)
  - Sem modificações, sem eco (=mostrar o caracter digitado)
  - Alguns programas usuários exigem: vi, emacs, password entry
- Modo cozido (cooked mode)
  - Driver faz o eco e o processamento de caracteres especiais.
  - Posix padroniza o efeito de teclas especiais
  - É o modo canônico (default)

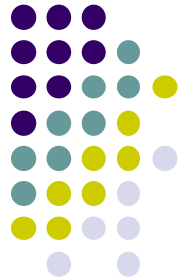
Software de Entrada



Caractere	Nome POSIX	Comentário
CTRL-H	ERASE	Retrocede um caractere
CTRL-U	KILL	Apaga a linha toda que está sendo digitada
CTRL-V	LNEXT	Interpreta literalmente o próximo caractere
CTRL-S	STOP	Pára a saída
CTRL-Q	START	Inicia a saída
DEL	INTR	Interrompe o processo (SIGINT)
CTRL-\	QUIT	Força a gravação da imagem da memória (SIGQUIT)
CTRL-D	EOF	Finaliza o arquivo
CTRL-M	CR	Retorno do carro (inalterável)
CTRL-J	LF	Próxima linha (inalterável)

Caracteres tratados de forma especial no modo canônico

Modo Cozido



- Driver precisa...
  - Bufferizar uma linha inteira antes de passa-la para o processo
  - Processar caracteres de controle especiais
    - Control-C, Backspace, Del, line-erase, Tab, shift
  - Ecoar o caracter digitado
  - Nova linha pode ser digitada em paralelo com o processamento de linha anterior
    - Para isso, precisa de buffers internos
- Abordagem 1 (para computadores com muitos terminais)
  - Manter um pool de buffers a serem usados por demanda
- Abordagem 2 (para computadores com 1 usuário)
  - Manter um buffer por terminal ((e.g., 500 bytes)

Saida para um terminal

- O terminal aceita sequencias de escape (escape sequence), que embutem controles especiais

ESCAPE: 0x1B

Exemplo:

esc [3 ; 1 H esc [0 K esc [1 M

Vá p/ posição (3,1) Da tela

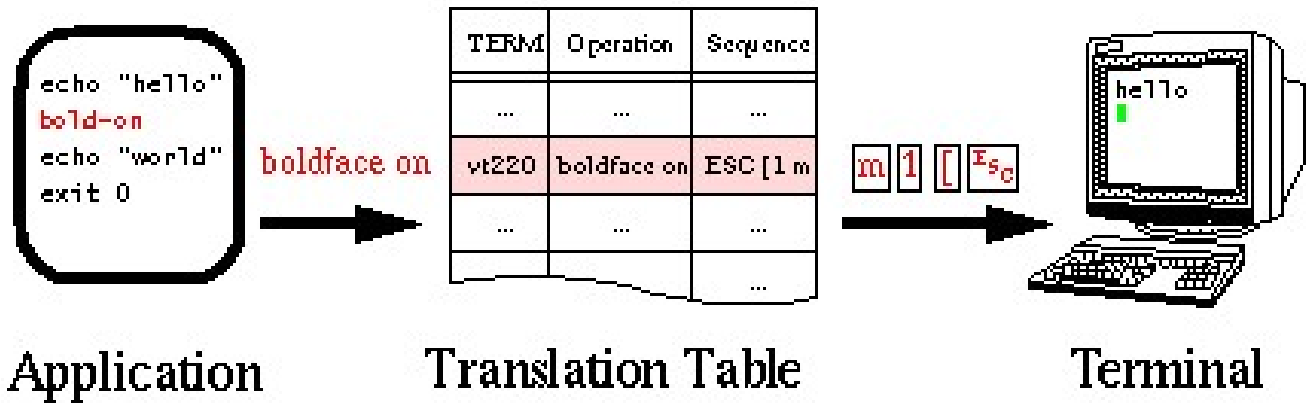
Apague a linha

Suba linhas seguintes em 1 linha

- Cada fabricante de terminal define sequencias ligeiramente diferentes
  - *Dificulta fazer software independente do dispositivo*
  - Unix usa arquivo “termcap”
    - É uma base de dados que gera as sequencias de escape para cada fabricante.

Papel do termcap

Fazer a tradução de comandos genéricos de saída para sequências de escape específicas para o tipo de terminal sendo usado.



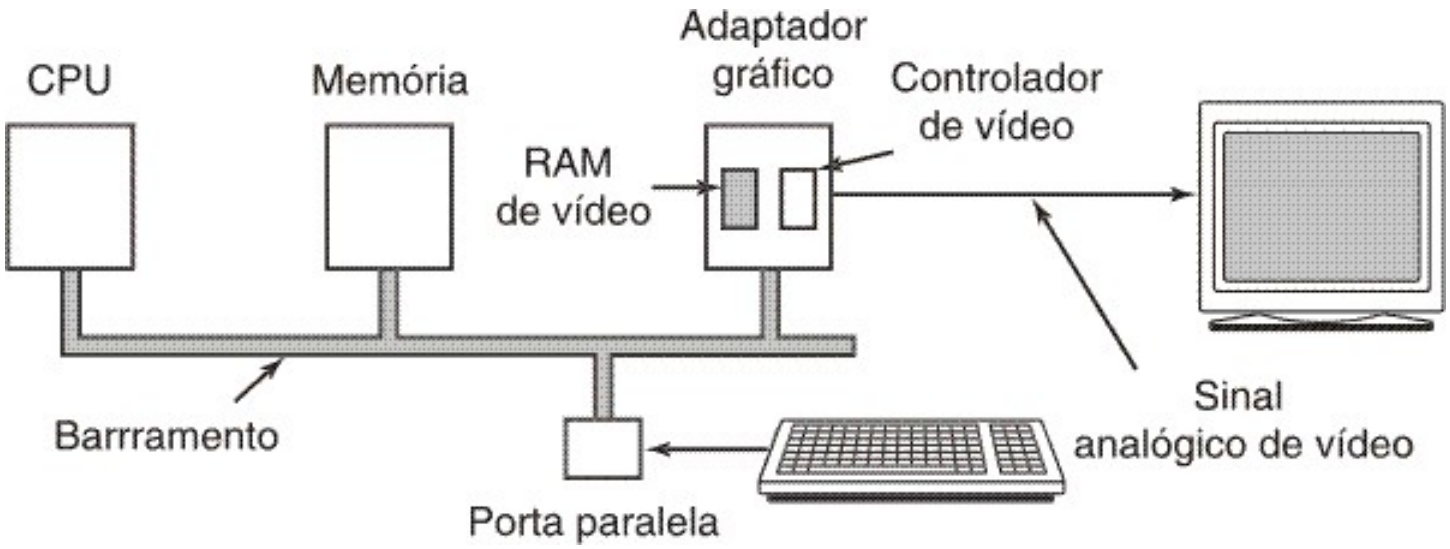
Software de Saída

Seqüência de escapes	Significado
ESC [*nA*	Move *n* linhas acima
ESC [*nB*	Move *n* linhas abaixo
ESC [*nC*	Move *n* espaços à direita
ESC [*nD*	Move *n* espaços à esquerda
ESC [*m*; *nH*	Move o cursor para (*m*, *n*)
ESC [*s*J	Limpa a tela a partir do cursor (0 até o final, 1 desde o início, 2 tudo)
ESC [*s*K	Limpa a linha a partir do cursor (0 até o final, 1 desde o início, 2 tudo)
ESC [*n*L	Insere *n* linhas a partir da posição do cursor
ESC [*n*M	Remove *n* linhas a partir da posição do cursor
ESC [*n*P	Remove *n* caracteres a partir da posição
ESC [*n* @	Insere *n* caracteres a partir da posição do cursor
ESC [*n m*	Habilita substituição do tipo *n* (0=normal, 4=negrito, 5=piscante, 7=invertido)
ESC M	Rola a tela para trás se o cursor está na linha do topo

Seqüências de escapes ANSI são usadas para controlar navegação e edição em editores de texto

- **reconhecidas pelo driver do terminal**
- **ESC é o caractere de escape ASCII (0x1B)**
- **n,m, e s são parâmetros numéricos opcionais**

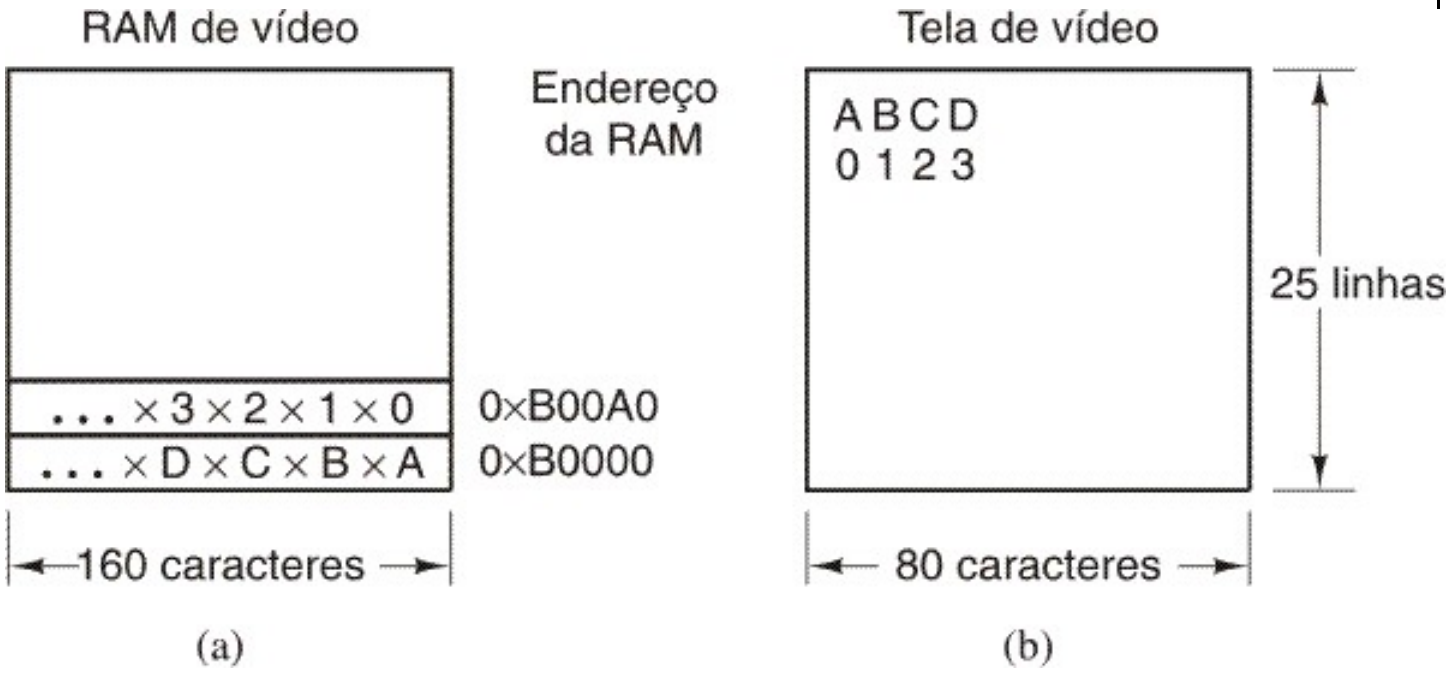
Hardware de Vídeo (1)



Vídeos mapeados na memória

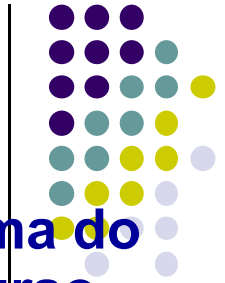
- driver escreve diretamente na RAM de vídeo do monitor

Hardware de Vídeo (2)



- Uma imagem da RAM de vídeo
  - tela monocromática simples
  - modo caractere
- Tela correspondente
  - os x´s são bytes de atributos

Impasses (*Deadlocks*)



Em várias situações o sistema operacional (ou um programa do usuário) precisa ter acesso exclusivo a mais de um recurso ou dispositivo. Por exemplo: cópia direta de dados entre dois dispositivos

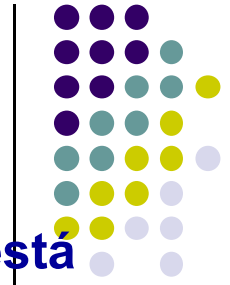
Impasses podem ocorrer para qualquer tipo de recurso: dispositivo de E/S (impressora, região em disco, interface de rede) ou uma bases de dados

- Exemplo em Banco de Dados:
  - Processo A faz lock em registro R1, e B faz lock em registro R2. Quando A tenta adquirir R2, é bloqueado, e não libera R1. Assim, B pode ficar bloqueado também

Recurso:: qualquer coisa que pode ser usada por um único processo a cada vez.

Um sistema geralmente tem vários tipos de recurso, e talvez várias instâncias de cada tipo. Cada instância poderá ser usada por um único processo.

Impasses



Recurso preemptivo:: que pode ser tirado do processo que o está usando sem causar problemas

Exemplo de recurso: memória principal

- Processo pode ser swapped out

Recurso não-preemptivo:: não pode ser tirado do processo que o está usando sem causar uma falha no processamento e inconsistência no estado do recurso

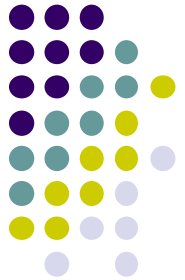
Exemplo de recursos: impressora ou fita

- Impasses só ocorrem com recursos não-preemptivos!
- Potenciais impasses com recursos preemtiveis podem ser resolvidos realocando o recurso de um processo para outro.
- Sequência de ações executadas por cada processo:

  (Requisita recurso; Usa recurso; Libera recurso)

A requisição bloqueia enquanto o recurso está em uso por outro processo.

Impasses



Quatro condições são necessárias para existência de um impasse:

1. Exclusão Mútua: cada recurso é atribuído a um único processo, ou então está disponível;

2. Condição segura&pede: O processo que é detentor de um recurso pode solicitar novos recursos.

3. Impossibilidade de preempção: somente o processo de posse do recurso pode liberá-lo;

4. Espera circular: deve existir uma cadeia circular de dois ou mais processos, cada um esperando por recursos sendo mantidos por outro processo;

Modelando Impasses

Essas 4 condições podem ser modeladas usando um grafo direcionado de recursos (e processos)

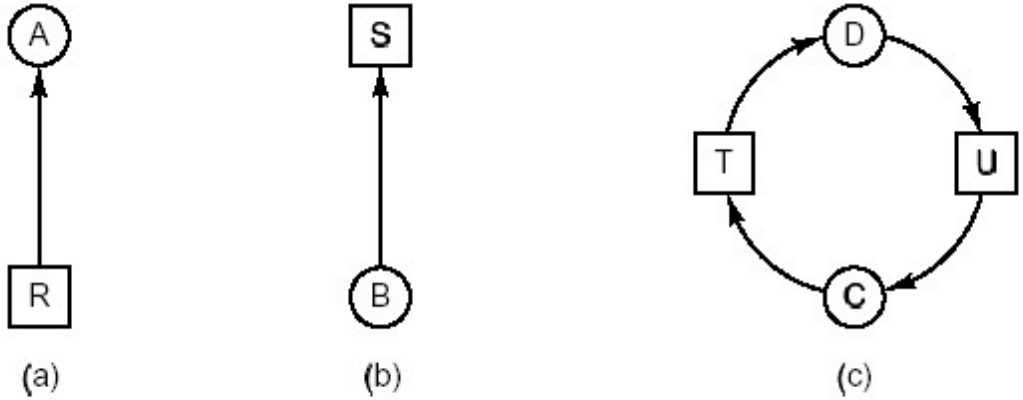


Figure 3-7. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

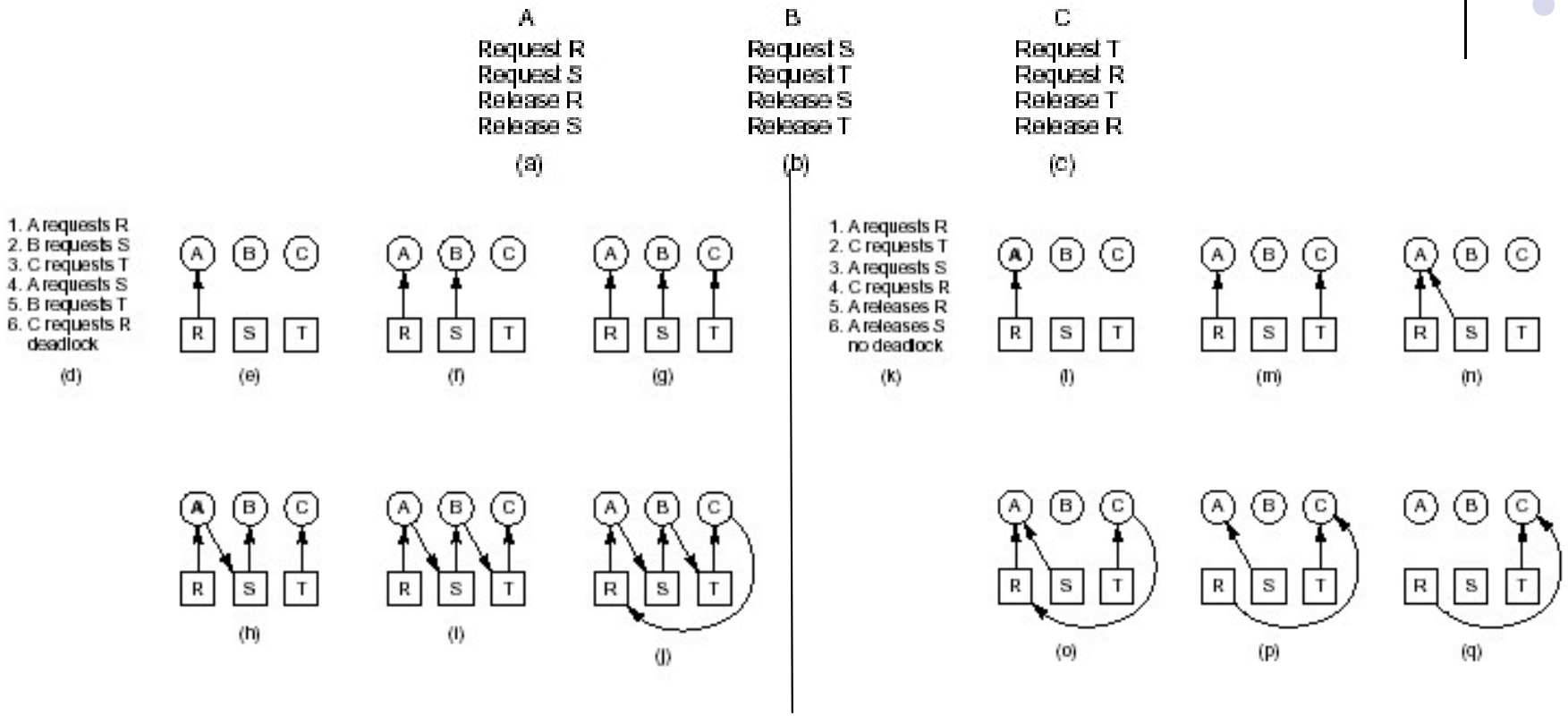
Impasse ⇔ se houver um ciclo no grafo!

Grafos de Recursos podem seu usados para verificar se uma certa sequência de requisições de recursos leva a um impasse:

➔ **Execute as requisições passo-a-passo e verifique se em algum momento forma-se um ciclo.**

Exemplo de Impasse

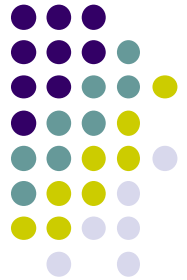
A ordem de alocação de recursos faz toda a diferença!



Se existe possibilidade de ocorrência de impasse, o S.O. precisa criar um *escalonamento seguro* de processos, i.e. suspender temporariamente um processo.

➔ No exemplo: executar primeiro A e C, depois B.

Estratégias para lidar com impasses



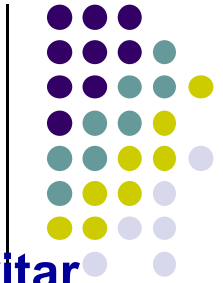
1. Ignore o problema e torça para que não ocorra
   - Abordagem prática de engenharia
   - Avalie bem: probabilidade vs impacto negativo vs custo de implementação
   - A maioria dos Sistemas Operacionais (incl. Unix/Minix) seguem essa abordagem

2. Detecção e Recuperação
   - A cada alocação de recurso (ou periodicamente) verifique o Grafo de Recursos; se houver um ciclo, termine um dos processos (e desfaça seus efeitos colaterais)

3. Prevenção
   - evite qualquer uma das 4 condições necessárias.

4. Alocação segura de recursos

Prevenção de Impasses



Idéia: impor restrições adequadas sobre os processos para evitar ocorrência de qualquer uma das 4 condições necessárias:

1. Exclusão Mútua: cada recurso é atribuído a um único processo, ou então está disponível;

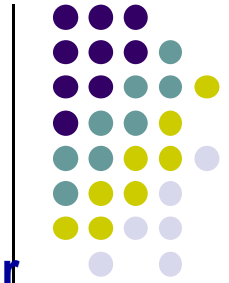
2. Condição segura&pede: O processo que é detentor de um recurso pode solicitar novos recursos.

3. Impossibilidade de preempção: somente o processo de posse do recurso pode liberá-lo;

4. Espera circular: deve existir uma cadeia circular de dois ou mais processos, cada um esperando por recursos sendo mantidos por outro processo;

Vejamos métodos para evitar cada condição...

Prevenção de Impasses



1. **Evitar exclusão mútua:**

   Concentre alocação em único processo (coordenador), e.g. spooler

   ...mas nem todos os recursos podem ser gerenciados dessa forma

   Exemplo: se spooler também usa espaço em disco: se começa a imprimir job1 antes de ter todos os dados em disco, um job de impressão (job2) pode ocupar todo o espaço restante do disco e evitar que job1 termine

2. **Evitar condição Segura&Pede:**

   Alternativa 1: Faça com que cada processo requisite todos os recursos antes de começar. Mas isso gera problemas:

   - Pode ser impossível conhecer de antemão todos os processos que serão usados.
   - Alocação de recursos não será eficiente: recursos serão bloqueados (lock) mesmo enquanto o processo estiver acessando outros recursos

   Alternativa 2: Se um processo quer requisitar um novo recurso, precisa antes liberar temporariamente (e re-adquirir posse) de todos os recursos que detém

   - Problema: É muito provável que todos os processos sejam sempre interrompidos (➔aumenta a sobrecarga)

Prevenção de Impasses

3. **Evite nâo-preempção:**

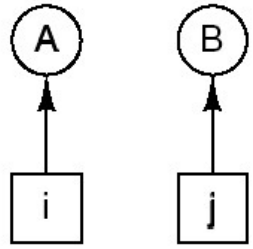
Todos os recursos teriam que ser preemptivos, mas isso é impossível (p.ex. Impressora, fita backup)

4. **Eliminar condição de espera circular – de alguma forma:**

Alternativa 1: garanta que cada processo mantenha um único recurso a cada momento (geralmente, inaceitável)

Alternativa 2: defina uma ordem global dos recursos e garanta que todas as requisições são sempre feitas seguindo essa ordem. Ciclo é evitado pois se $i \neq j$ então: se A mantém R_i não irá requisitar R_j, ou se B mantém R_j não irá requisitar P_i.

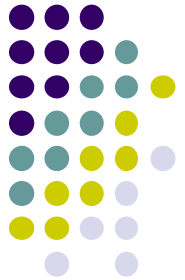
1. CD-ROM
2. Printer
3. Plotter
4. Tape drive
5. Robot arm



Variante: Em vez de impor uma ordem estrita de requisições garanta que nenhum processo requisite um recurso com no. maior do que um recurso que já possui.

Problema: O número de recursos potencialmente requisitados pode não ser conhecido de antemão, impossibilitando uma ordenação

Prevenção de Impasses



Resumo das abordagens:

Condition	Abordagem
Mutual exclusion	Coordene alocação
Hold and wait	Requisite todos Rs incialmente
No preemption	Desaloque recursos
Circular wait	Alocação ordenada de R's

Alocação segura de recursos (Deadlock Avoidance)

O Algoritmo do Banqueiro (Dijkstra,65) determina uma alocação segura de recursos de forma que impasses nunca irão ocorrer.

Idéia princial:

- Mantenha sempre suficientes recursos (estado seguro) de maneira que sempre exista um processo que possa alocar todos os recursos que precisa e assim possa terminar

- Estado seguro:: é um estado de alocação de recursos tal que exista uma sequência de estados futuros de alocação dos recursos (e finalização dos processos) que garanta que todos os processos em algum momento obterão todos os recursos necessários e terminarão.
- Premissa:

  Existe conhecimento prévio do conjunto máximo de recursos necessários para cada processo

Algoritmo do Banqueiro

Caso para 1 tipo de recurso (por exemplo, $$):

Um banco de uma pequena cidade dá crédito a fazendeiros até um limite máximo; clientes sacam o dinheiro à medida que precisam dele; o banco precisa garantir que sempre há suficiente dinheiro disponível face os créditos concedidos.

Exemplo: Banco tem um total de 10 unidades (p.ex. R$ 10 milhões) para todos os seus clientes.

Name	Used	Maximum
Andy	0	6
Barbara	0	5
Marvin	0	4
Suzanne	0	7

Available: 10

(a)

Name	Used	Maximum
Andy	1	6
Barbara	1	5
Marvin	2	4
Suzanne	4	7

Available: 2

(b)

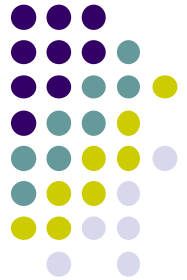
Name	Used	Maximum
Andy	1	6
Barbara	2	5
Marvin	2	4
Suzanne	4	7

Available: 1

(c)

Figure 3-11. Three resource allocation states: (a) Safe. (b) Safe. (c) Unsafe.

Algoritmo do Banqueiro
Recursos com vários tipos



- Usa duas matrizes e 3 vetores:
  - E: total de instâncias de cada tipo de recurso
  - P: quantidades alocadas de cada tipo de recurso
  - A: quantidades disponíveis de cada tipo de recurso

Process	Tape drives	Plotters	Printers	CD-ROMS
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD-ROMS
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

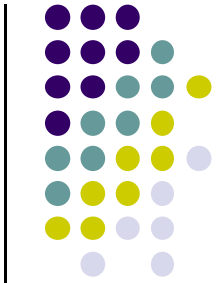
E = (6342)
P = (5322)
A = (1020)

Algoritmo do Banqueiro

Verificação se uma nova requisição de recursos (equivale a pedido de empréstimo) vai levar a um estado seguro:

1. Procure por processo P (linha da matriz à direita) cuja demanda restante de recursos é menor do que A. Se não existe tal processo, então sistema **pode** gerar em impasse;
2. Senão, assuma que P terminou (marque-o), e adicione o número máximos de recursos de P ao vetor A.
3. Repita passos 1 e 2 até que todos os processos foram marcados como terminados, ou então verificou-se que estado não é seguro ➔ requisição não deve ser atendida.

Algoritmo do Banqueiro



Trata-se de um trabalho teórico interessante ... Mas raramente aplicado, de fato na prática.

Principais problemas:

- Impossibilidade de conhecer de antemão as quantidades máximas de recursos necessários para cada processo;
- Conjunto dos processos não é fixo
- O conjunto de recursos é dinâmico (incl. falhas)

Existem abordagens mais pragmáticas para a Prevenção de Impasses (p.ex. Bloqueio em 2 fases, principalmente para bancos de dados)

- Processo tenta bloquear todos os registros que precisa (fase 1) de uma vez. Se um dos registros já está bloqueado, processo aborta tentativa e tenta novamente mais tarde
- Se tiver sucesso, faz os acessos e libera todos os registros simultaneamente (phase 2).

Perguntas?