

Gerenciamento de I/O e Interrupções

Parte 1 - Visão Geral e Mecanismos

Engenharia de Computação

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizagem

Ao final desta aula, você será capaz de:

- Explicar técnicas de gerenciamento de I/O.
 - Incluindo a sequência de eventos sob técnicas síncronas e assíncronas (polling vs interrupções)
- Discutir as vantagens e desvantagens das técnicas de gerenciamento de I/O
- Explicar (interrupção) livelock e mitigação via técnica híbrida.

Conceitos Fundamentais

Questões principais

- Como podemos gerenciar I/O?
 - Solução simples: Loop para aguardar I/O → Pooling
 - Por que manter a CPU ocupada em um loop quando nada acontece?
 - Com multitasking, esse tempo é desperdiçado, pois, outras tarefas poderiam utilizar melhor a CPU.
- Como podemos melhorar o processamento de I/O em relação ao polling?
 - Adicionar interrupções como mecanismo de notificação de I/O.
 - Como organizar o I/O então?
 - Qual o *overhead* (sobrecarga) que surge?

Visão Geral

- I/O é um dos aspectos mais importantes na operação de um computador
- Desafios:
 - Dispositivos de I/O variam muito
 - Métodos diversos para controlá-los
 - Gerenciamento de desempenho
 - Novos dispositivos surgem frequentemente

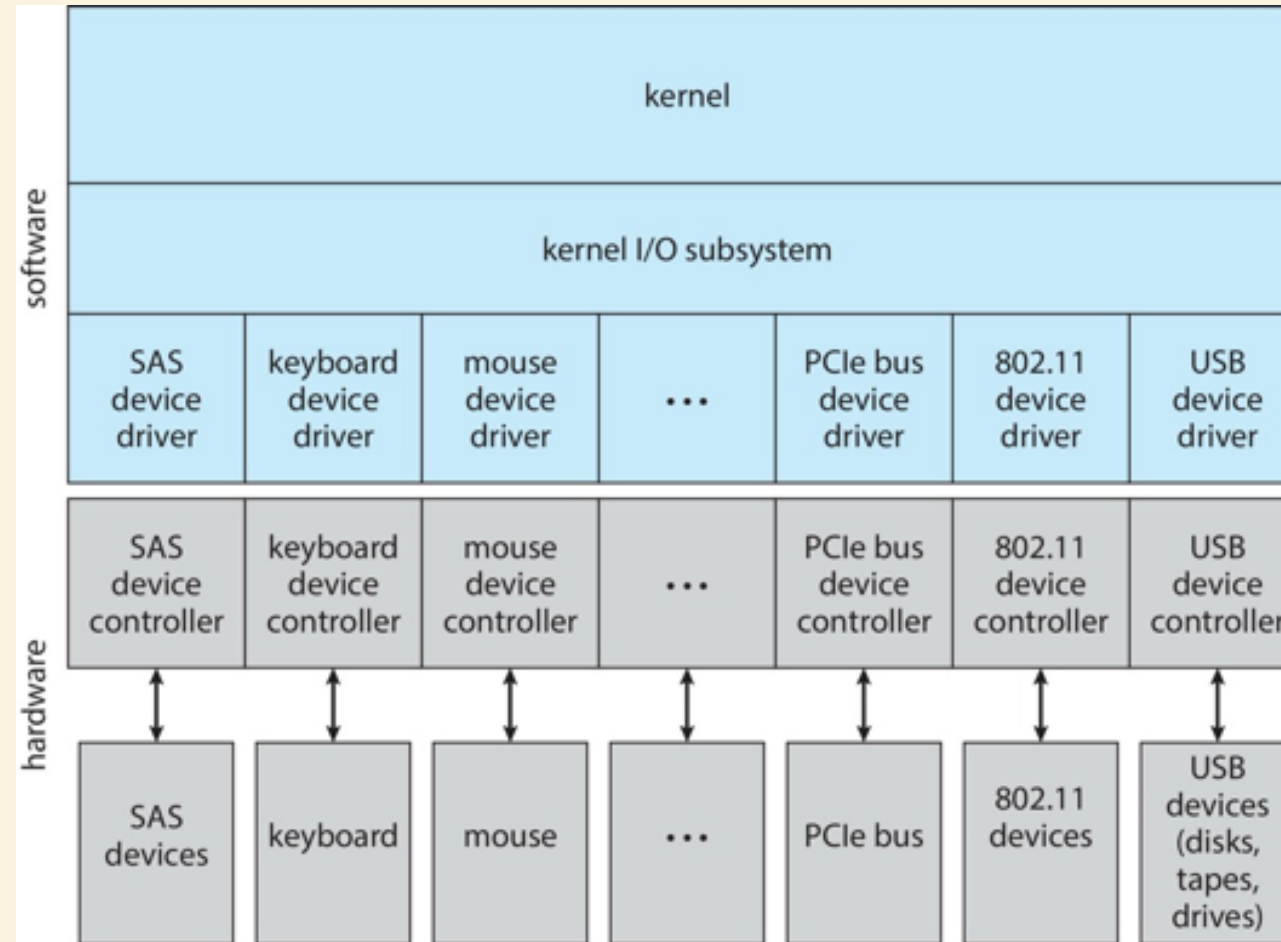
Visão Geral (cont.)

- Os dispositivos de I/O são componentes que interagem com o SO (Sistema Operacional).
- Alguns recebem solicitações e entregam resultados
 - Ex.: disco, impressora, placa de rede.
- Outros geram eventos por conta própria
 - Ex.: temporizador/relógio, teclado, placa de rede.

Visão Geral (cont.)

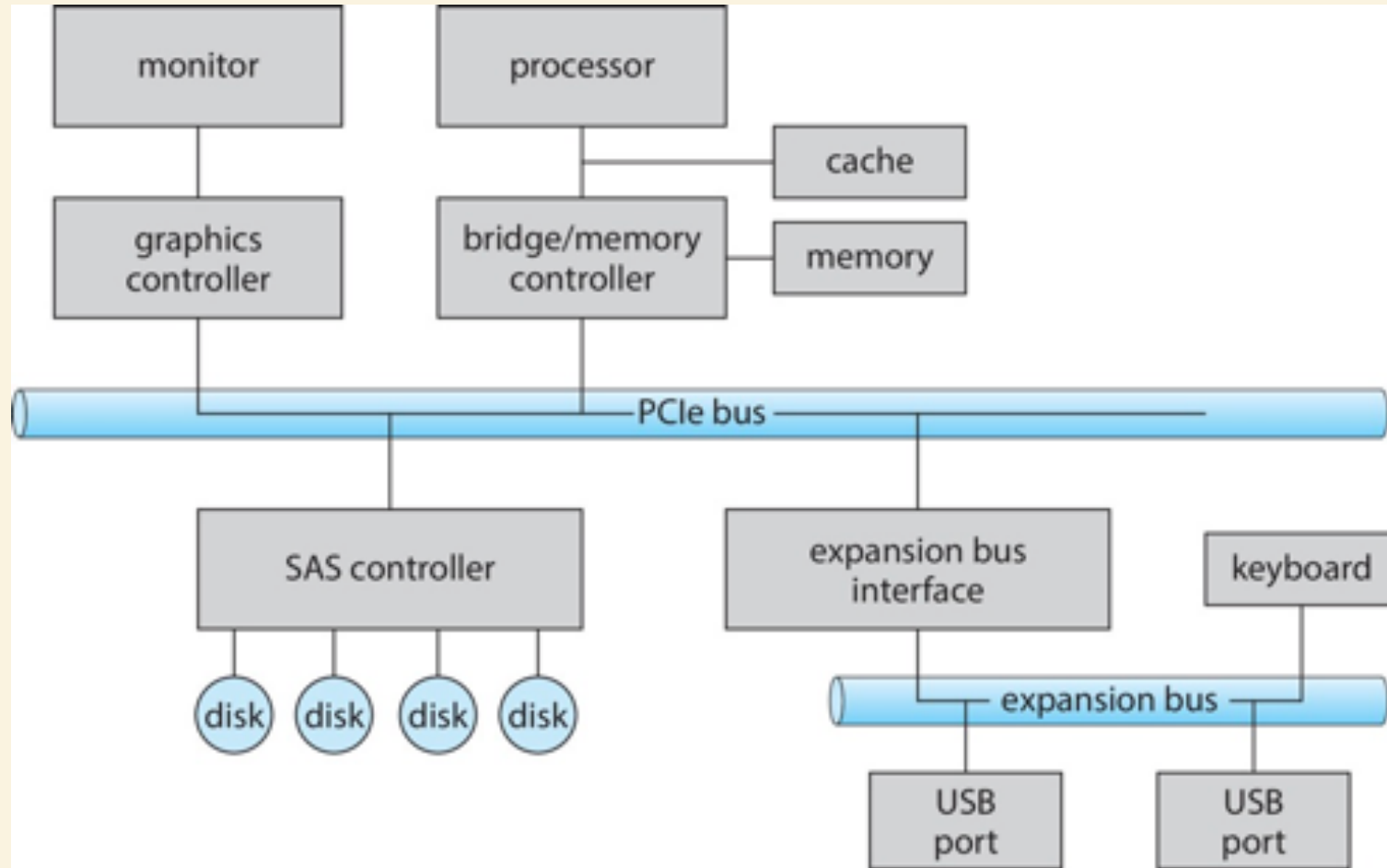
- **Função do SO:** gerenciar e controlar operações e dispositivos de I/O
- Driver de dispositivo: Interface uniforme para o subsistema de I/O
 - Encapsula detalhes do hardware do dispositivo
- Kernel do SO estruturado para usar módulos de **driver de dispositivo**.

Estrutura do Kernel para gerenciamento de I/O



Fonte da Imagem: A. Silberschatz *et. al*, **Operating Systems Concepts**, capítulo 13.

Estrutura Típica de Barramento em PCs



Fonte da Imagem: A. Silberschatz *et. al*, **Operating Systems Concepts**, capítulo 13.

Hardware de I/O

- Grande variedade de dispositivos de I/O:
 - Armazenamento
 - Interface humana
- Conceitos comuns:
 - Porta: ponto de conexão do dispositivo
 - Barramento: compartilhamento ou conexão em cadeia
- Exemplos:
 - Barramento PCI comum em PCs
 - PCI Express (PCIe) em servidores modernos
 - SAS (Serial Attached SCSI) para discos

Hardware de I/O (cont.)

- Controlador (adaptador de host):
 - Opera porta, barramento e dispositivo
 - Pode ser integrado ou em placa separada
 - Contém processador, microcódigo, memória privada, etc.
- Alguns controladores por dispositivo também possuem microcódigo próprio

Hardware de I/O (cont.)

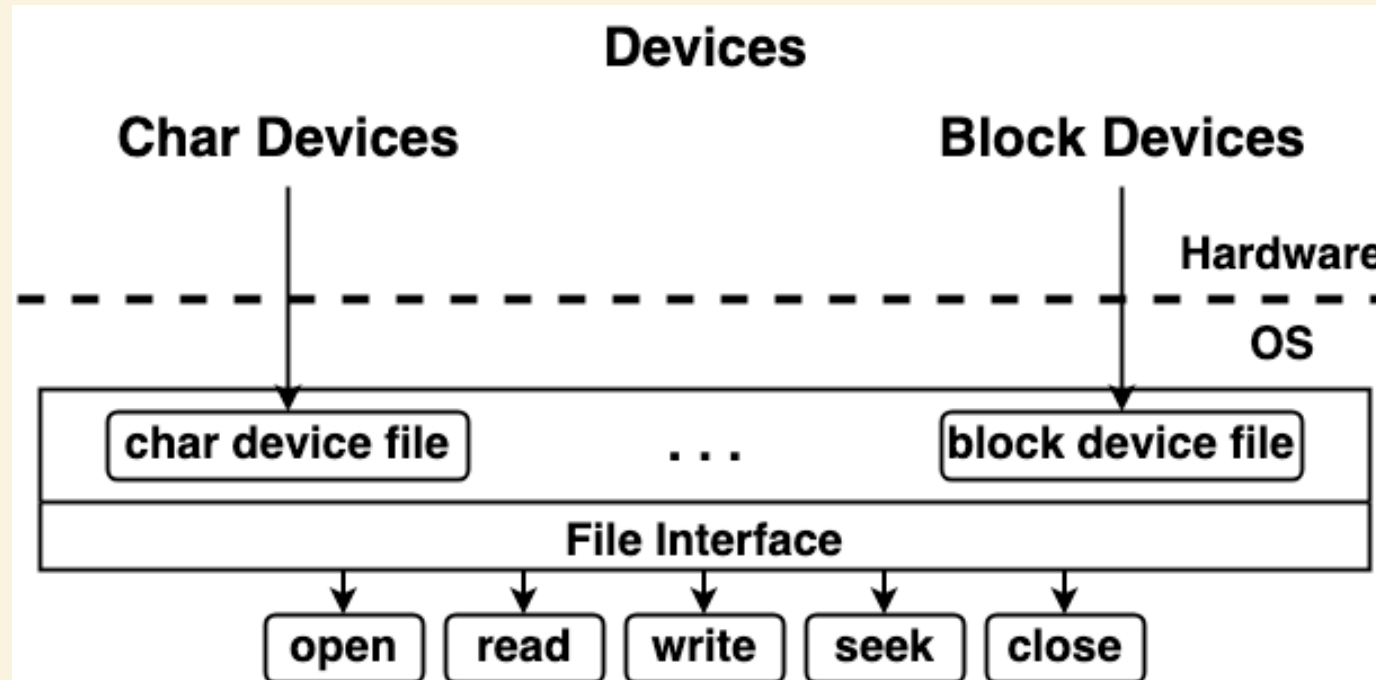
- Fibre Channel (FC): controlador complexo, geralmente em adaptador separado (host-bus adapter, HBA)
- Instruções de I/O controlam dispositivos
- Dispositivos possuem registradores para comandos, endereços e dados:
 - Data-in, Data-out, Status e Controle
 - Registradores geralmente entre 1-4 bytes ou FIFO

Endereçamento de Dispositivos

- Dispositivos têm endereços utilizados por:
 - Instruções diretas de I/O
 - I/O mapeado em memória
- **Memory mapping:** Dados e comandos dos dispositivos são mapeados no espaço de endereçamento do processador
 - Útil para dispositivos de grande capacidade como placas gráficas

Representação de Dispositivos de I/O no Linux

Linux abstrai um dispositivo de I/O como um arquivo especial (diretório `/dev`)

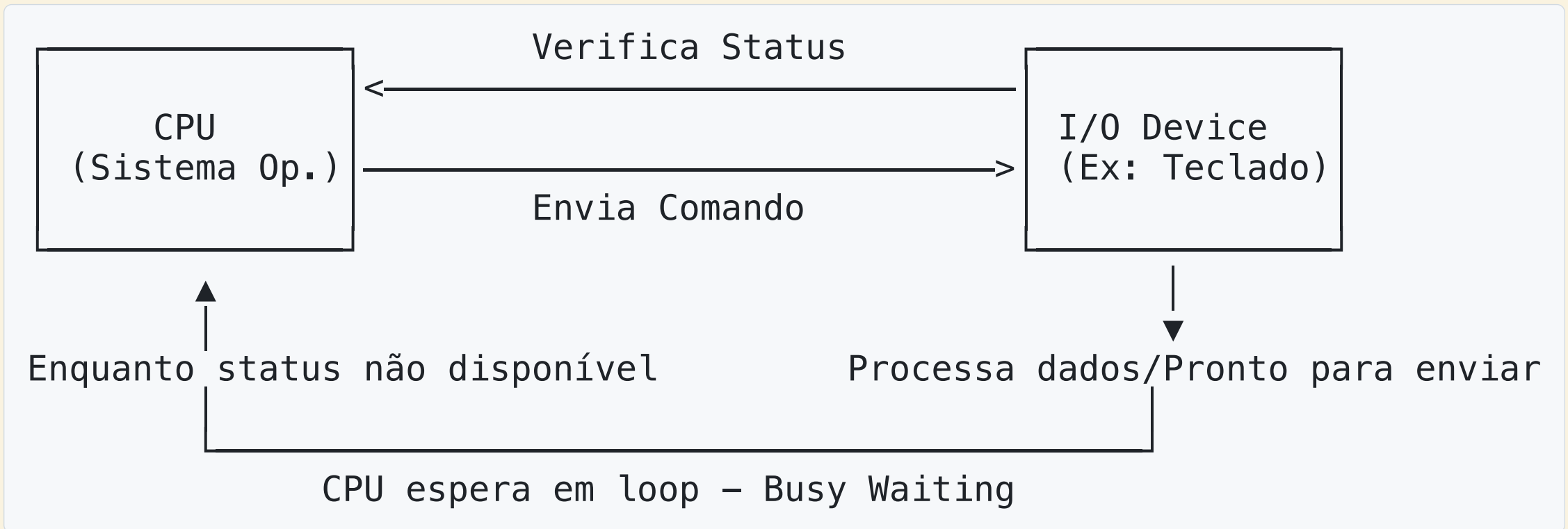


Fonte da Imagem: [OS Team - OS OER](#)

Mecanismos de gerenciamento de I/O

- **Polling:** SO "pergunta" continuamente por eventos ou resultados
 - Com o polling, a I/O é chamada de **síncrona**.
 - SO monitora a operação de I/O para verificação de conclusão.
- **Interrupções:** o dispositivo alerta quando ocorreu um evento ou resultado
 - Com interrupções, a I/O é chamada de **assíncrona**.
 - O SO inicia a operação de I/O.
 - I/O prossegue de forma assíncrona, liberando a CPU para realizar outras tarefas.
 - O dispositivo aciona uma interrupção quando a operação de I/O é concluída.
 - CPU é interrompida e o resultado da I/O é tratado pelo SO.

Polling



Polling (cont.)

Passos para cada byte de I/O:

1. Ler bit de ocupado do registrador de status até ser 0
2. Configurar bits de leitura/escrita
3. Copiar dados para registrador se for escrita
4. Ativar bit de comando-pronto
5. Controlador configure bit de ocupado e executa a transferência
6. Controlador limpa bits ao finalizar

Polling (cont.)

Passos para cada byte de I/O:

1. Ler bit de ocupado do registrador de status até ser 0
2. Configurar bits de leitura/escrita
3. Copiar dados para registrador se for escrita
4. Ativar bit de comando-pronto
5. Controlador configure bit de ocupado e execute a transferência
6. Controlador limpa bits ao finalizar

Passo 1 é um ciclo **busy-wait** para aguardar I/O do dispositivo.

- Razoável se o dispositivo for rápido. Mas ineficiente se o dispositivo for lento.
- A CPU muda para outras tarefas? Os dados pode ser sobrescritos/perdidos.

Polling (cont.)

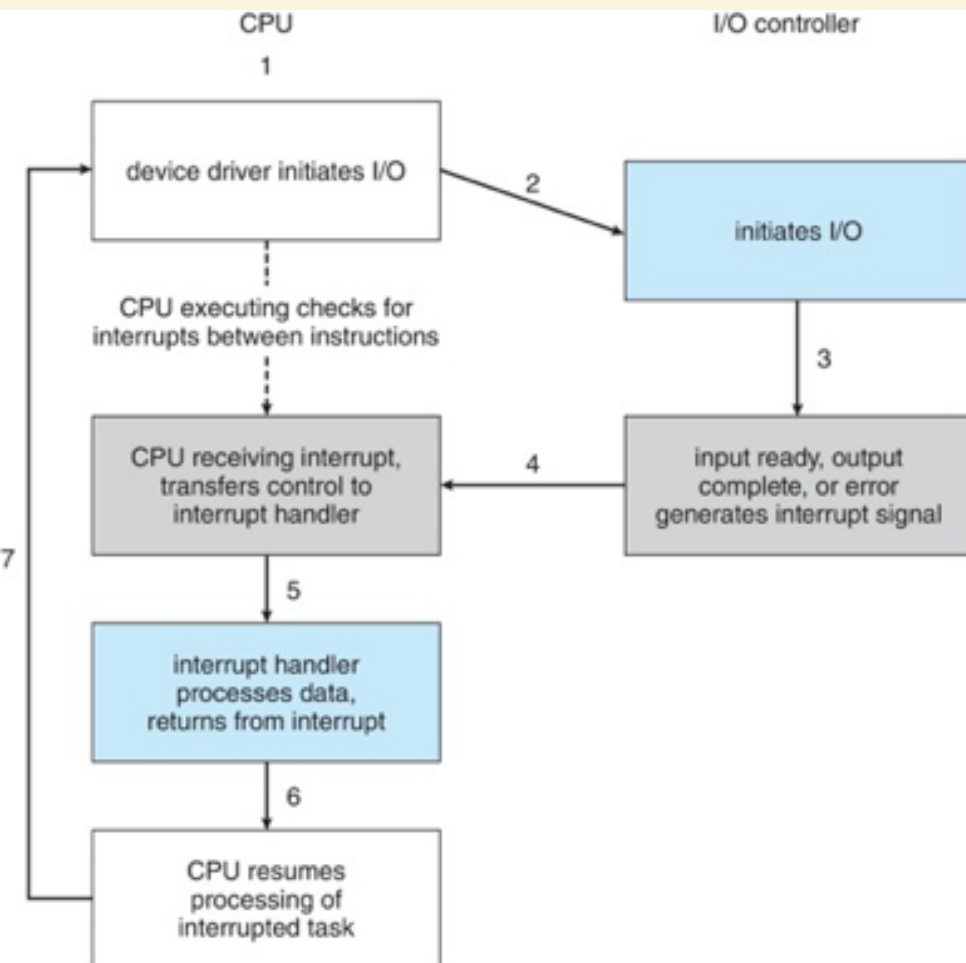
- **Vantagens:**
 - Fácil de programar
 - Rápido: resultado processado assim que estiver disponível
 - Sem overhead
- **Desvantagem:**
 - *Busy waiting*: CPU espera a ocorrência de um evento
 - Desperdício de CPU
 - Ruim se o período de espera é muito longo

Interrupções

- Alternativa ao polling para eficiência
- Gerada externamente à CPU:
 - SO inicia a operação de I/O no dispositivo.
 - CPU fica livre para fazer outra coisa **assincronamente** durante a execução da I/O.
 - Em momento posterior, a operação de I/O é concluída e o dispositivo aciona uma interrupção.
 - O manipulador de interrupções do SO age de acordo.
 - CPU é interrompida e salta (muda o registrador PC) para o *interrupt handler*
- CPU não precisa esperar I/O ser completada: sem *busy waiting*
 - Interrupt handler (tratador de interrupções) manipula evento

Ciclo de I/O Baseado em Interrupção

- Dispositivo de I/O aciona interrupção da CPU.
- O gerenciador (handler) de interrupções recebe a interrupção.
- Um vetor de interrupção é usado para despachar a interrupção para o gerenciador (handler) correto.
 - Troca de contexto no início e no fim, baseada na prioridade.
 - Encadeamento de interrupções se mais de um dispositivo estiver na mesma interrupção.
- Fonte da Imagem: A. Silberschatz *et. al*, **Operating Systems Concepts**, capítulo 13.



Interrupções (cont.)

- O SO especifica um manipulador para cada tipo de interrupção e exceção.
 - *Handler* = função
 - O tipo de interrupção é determinado pelo número.
- Exemplo em processadores x86:
 - Os endereços dos *handlers* são armazenados pelo SO em uma tabela na memória, a *Interrupt Descriptor Table* (IDT).
 - Vetor de Interrupção
 - Cada entrada da tabela aponta para um *handler*/função.
- Núcleo de CPU contém o *Interrupt Descriptor Table Register* (IDTR).
 - SO inicializa o IDTR com o endereço de início da IDT

Interrupções (cont.)

- Ao receber uma interrupção do tipo n:
 - Ocorre uma **mudança de contexto**, e (no modo kernel) a CPU:
 - Salva o estado da execução atual,
 - Usa o IDTR para acessar a IDT,
 - Consulta a entrada n na IDT e invoca o *handler*/função correspondente.
 - Posteriormente, o estado é restaurado e a execução anterior continua.
- A troca de contexto acarreta uma sobrecarga (overhead).

Interrupções (Cont.)

- Também usadas para exceções e chamadas de sistema
- Em sistemas multi-CPU, interrupções podem ser processadas simultaneamente
- Usadas para processamento sensível a tempo

Leitura adicional

- Capítulo 5 do livro **Sistemas Operacionais Modernos**, de A. TANENBAUM
- Capítulo 13 do livro: **Operating Systems Concepts**, de A. Silberchatz *et. al.*

Dúvidas e Discussão
