Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
Bacharelado em Tecnologia da Informação

IMD0041 - INTRODUÇÃO A ORGANIZAÇÃO E ARQUITETURA DE COMPUTADORES

DISPOSITIVOS DE E/S, BARRAMENTO, MULTI-CORE/MULTITHREADING E SISTEMA OPERACIONAL

Natal/RN
15 de junho de 2016

Sumário

O que é esse documento?

Esse é um documento que visa resumir o conteúdo apresentado em sala de aula na disciplina **Introdução a Organização e Arquitetura de Computadores** da **Universidade Federal do Rio Grande do Norte**, durante o descorrer da terceira unidade, matéria essa da grade curricular do curso de **Bacharelado em Tecnologia da Informação**.

Tenha em vista que esse documento não server como base concreta/completa de estudo. Ele, na verdade, visa resumir e direcionar o estudo da disciplina. Dessa forma, cabe ao aluno buscar formas complementares de entender o que está sendo dito aqui, como, por exemplo, material referência utilizado.

Sobre as figuras

Todas as imagens (figuras) desse documento foram retiradas dos materiais apresentados em sala de aula na disciplina de Introdução a Organização e Arquitetura de Computadores feitos pela professora Dra. Monica Magalhães Pereira, do Departamento de Informática e Matemática Aplicada (DIMAp), da Universidade Federal do Rio Grande do Norte (UFRN).

Entretando, saiba também que as imagens presentes no material da professora podem ter sido retirados de outros materiais fontes. A maioria deverá estar listado nas referências desse documento. Portanto, ao utilizar as imagens aqui presentes, tome cuidado e cite as suas devidas autorias.

Capítulo 1

Introdução

Uma rápida lembrança de como é a **Arquitetura de von Neumann**, exemplificada na figura 1.1:

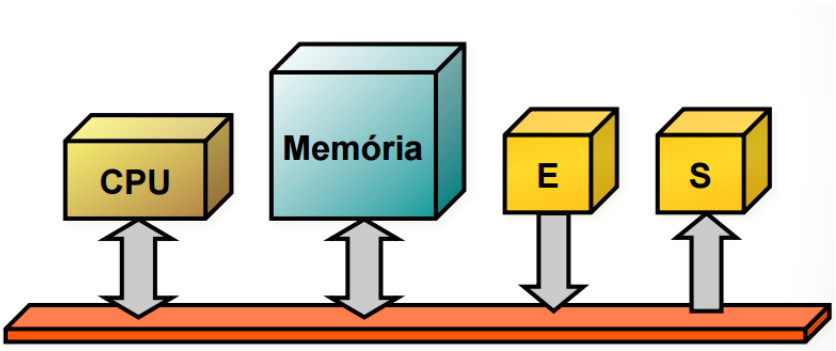


Figura 1.1: Arquitetura de von Neumann. O retângulo laranja representa um barramento.

Aqui vamos buscar entender como funciona a organização dos **dispositivos de entrada e saída** (**E/S** ou I/O), como eles se comunicam com os demais dispositivos do computador e entender como funciona um **barramento**.

Capítulo 2

Entrada / Saída

Dispositivos de Entrada e Saída (E/S ou I/O – Input/ Output) oferecem um meio para **troca de dados** entre **ambiente** externo e o processador. Na figura 2.1 temos alguns exemplos de E/S.

Figura 2.1: Exemplos de dispositivos de entrada (teclado, mouse) e saída (impressora), os chamados **Periféricos**.

2.1 Introdução

A comunicação entre os dispositivos perífericos e os dispositivos internos é feita por meio de um barramento, como exemplificado na figura 2.2.

2.2 Controlador E/S

Note que entre os dispositivos perífericos (**Disco é considerado periférico**) e o barramento existe um **Controlador de Entrada e Saída**.

Mas por que periférico não pode ser conectado direto no barramento do sistema?

Isso acontece devido a grande variedade de periféricos (o que implica em várias lógicas diferentes); a taxa de transferência dos periféricos, que é muito mais lenta e maior do que memória ou processador (figura 2.3); e ao formato

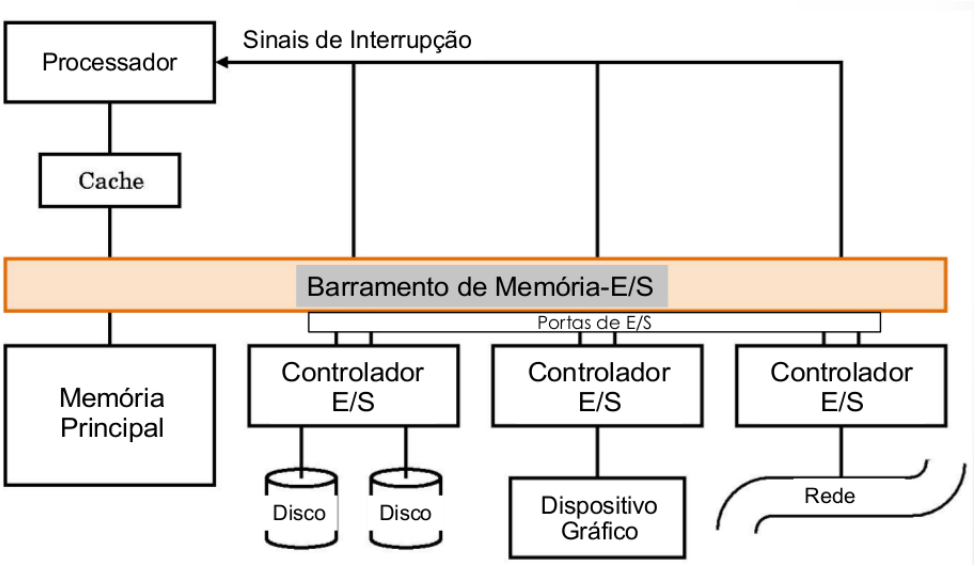


Figura 2.2: Básica organização interna de um computador. Barramento manuseia dados e comunicação entre CPU, memória e E/S.

de dados e tamanhos de palavras diferentes. Por isso é necessário o controlador (ou módulo) de E/S.

CD-ROM/CD-RW 8x	1200 KB/s
CD-ROM/CD-RW 56x	8400 KB/s
USB 1.1	1.5MB/s
USB 2.0	60 MB/s
USB 3.0	600 MB/s
COM (mouse, teclado)	14.4 KB/s
LPT (impressora, scaner)	1.2 MB/s

Figura 2.3: Exemplos de taxas de tranferências de alguns dispositivos de E/S.

2.3 Portas de E/S

Portas de E/S funcionam como uma **interface** entre o barramento de sistemas e os dispositivos de E/S. Elas permitem **troca**, **retirada** e **acréscimo** de dispositivos de E/S **sem alteração** do restante da arquitetura. Isso implica que o processador fica livre dos detalhes de baixo-nível.

2.4 Características

Os dispositos E/S possuem algumas características básicas, como mostrado na figura 2.4.

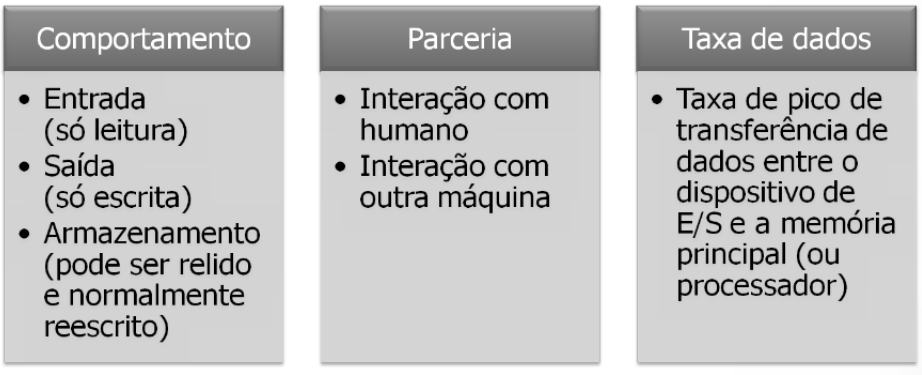


Figura 2.4: Características básicas na organização de dispositivos de E/S.

Exemplo: teclado é um dispositivo de **entrada** utilizado por um **humano** com uma taxa de transferência em torno de **15 KB/s**.

2.5 Esquemas de acesso

Dependendo da forma como são acessadas pelo processador, as portas de E/S podem ser:

Dedicadas	- Possuem um espaço de endereço único, diferente do espaço de endereçamento da memória; - Comandos especiais de E/S são utilizados para comunicação.
Mapeadas em memória	- Compartilham o mesmo espaço de endereçamentoda memória; - Por isso, são normalmente vistas pelo processador como locais da memória.

2.6 Modos de transmissão

Há duas maneiras básicas de se realizar transmissão/recepção de dados entre os periféricos e memória principal (**MP**), bem como entre dispositivos interconectados entre si, local ou remotamente, que são a **transmissão serial** ou **transmissão paralela.**

2.6.1 Trasmissão paralela

Utiliza uma linha por bit, e todos os bits são transmitidos simultaneamente.

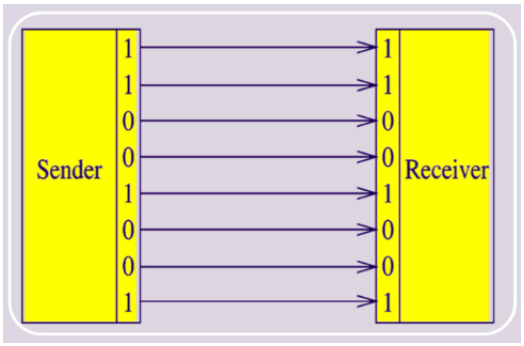


Figura 2.5: Transmissão Paralela. Indicada para transmissões internas no sistema de computação (barramentos) e para ligações de periféricos a curta distância (impressoras, discos rígidos, etc).

2.6.2 Transmissão serial

Única linha de transmissão de dados, e cada bit é transmitido serialmente, um por vez.

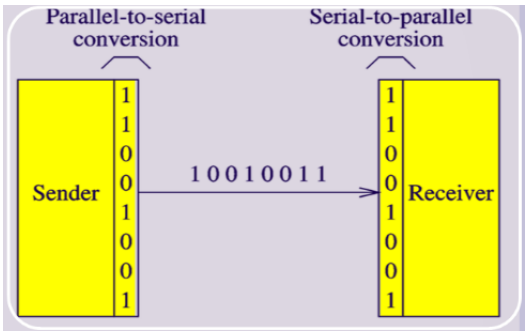


Figura 2.6: Trasmissão serial. Pode ser divido em dois tipos: **Síncrona** e **Assíncrona**.

Assíncrona

Consiste em um processo de sincronização do receptor a cada novo caractere transmitido.

Antes de se iniciar a transmissão, cada caractere é acrescido de 2 pulsos, um no início do caractere, denominado START, e o outro, denominado STOP, além do bit de paridade

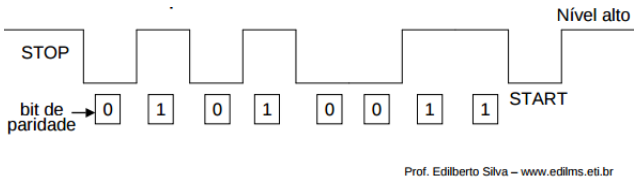


Figura 2.7: Transmissão Serial Assíncrona

Síncrona

Ela é mais eficiente que a assíncrona. O transmissor monta um bloco (128 a 256 carac) -> sem intervalo entre o primeiro e o último bit;

2.7 Transferência de dados

A transferência de dados entre memória e dispositivos de E/S ocorre por meio de **protocolos**.

Três técnicas básicas são utilizadas (figura 2.8: (i) E/S Programada (Polling); (ii) Tranferência Orientada a Interrupções e (iii) Direct Memory Access (DMA). É importante deixar claro que o (i) possui um maior **grau de envolvimento** com o processador, o que vai diminuindo conforme vamos para o (ii) e mais ainda para o (iii).

2.7.1 Polling

O processador testa **repetida** e **sequencialmente** cada periférico, a fim de verificar se eles se encontram prontos para receber ou enviar dados

Para atender às demandas de todos os periféricos, a CPU deve **consultar** todos os dispositivos com a frequência do dispositivo que exige a **maior taxa**.

Ações da CPU ao consultar um dispositivo E/S:

- **Interrompe** a execução do programa.
- Realiza a **sequência** de consulta.
- **Provê** o serviço, se necessário.
- **Retoma** a execução do programa.

Exemplificando... "Imagine um professor dando aula.Ele aguarda que um documento venha da direção até ele e, por causa disso, a cada x minutos para a aula e verifica na porta se alguém está vindo deixar o papel. Depois de verificar, continua a aula."

2.7.2 Interrupções

Por este método, a CPU responde por uma requisição de serviço **somente quando requisitada** pelo periférico.

Dessa forma, a CPU pode se concentrar na execução de um programa sem ter de interromper desnecessariamente para verificar se um periférico precisa ou não ser atendido.

Exemplificando... "Seguindo os mesmos passos do exemplo anterior. Agora, a pessoa responsável por trazer o documento até o professor bate na porta, e ai sim o professor para a aula e vai até a porta buscar o documento."

2.7.3 DMA

Nos casos de transferência de dados entre dispositivos de E/S, o uso de técnicas de polling ou de interrupção **elevaria a carga** de trabalho do processador.

O mecanismo de Acesso Direto à Memória (DMA – Direct Memory Access) possibilita transferir dados diretamente da/para memória sem envolver o **processador**.

O mecanismo de interrupções é utilizado apenas para:

- Informar a CPU de que a **transferência terminou**, ou
- Notificar a ocorrência de **erros**.

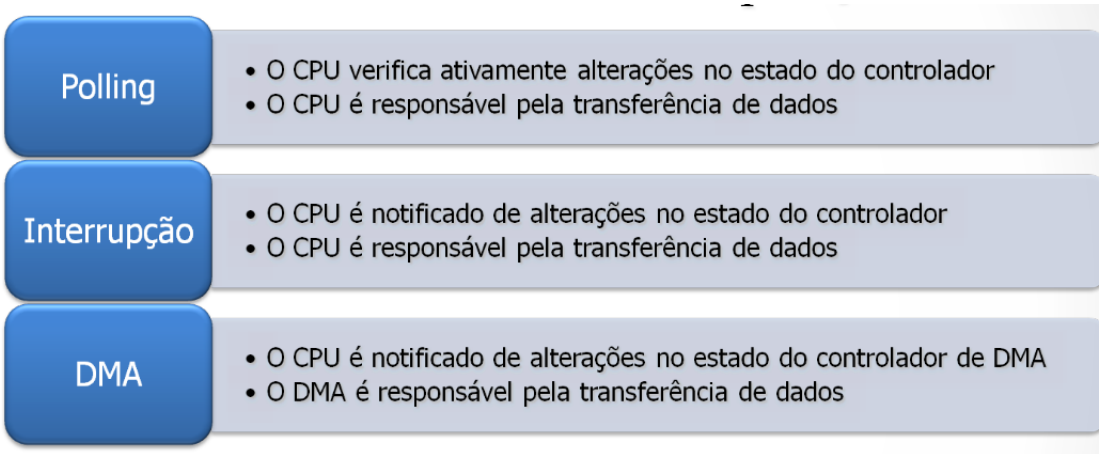


Figura 2.8: Transferência de Dados - Comparação

Capítulo 3

Barramentos

Barramento é o meio de comunicação compartilhado por vários dispositivos, constituído por sinais de dados, endereços e controle.

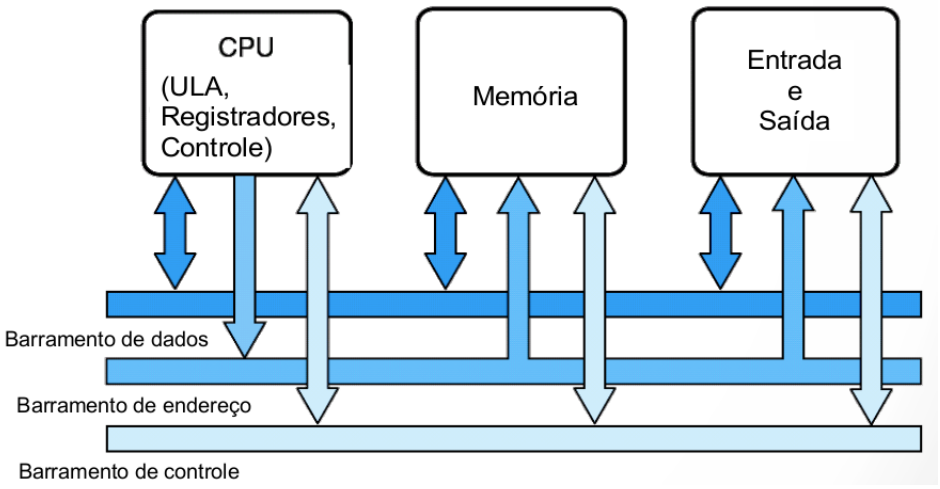


Figura 3.1: Barramento

3.1 Estrutura do barramento

Um barramento do sistema, possui várias linhas de transmissão.

- Cada linha transmite sinais representando o valor binário 1 ou 0;
- Uma sequência de dígitos podem ser transmitidos ou várias linhas com transmissão em paralelo e simultânea podem formar códigos binários

Cada linha possui uma função e um significado. Essas linhas podem ser classificadas em 3 grupos funcionais (como pode ser visto na imagem 3.1):

- Linhas de Dados
- Linhas de Endereço
- Linhas de Controle

3.1.1 Barramento de Dados

Transporta informações (dados ou instruções) e é **bidirecional**[1].

3.1.2 Barramento de Endereços

Utilizado pelo processador para enviar endereços de memória ou dos dispostivos de E/S. A largura depende do número de locais de memória que se pretende acessar e é **unidirecional**[2].

3.1.3 Barramento de Controle

Utilizado para sinalizar solicitações e confirmações. É **bidirecional**.
Exemplo:

- clock
- reset
- memory read/write
- bus request

3.2 Características dos Barramentos

3.2.1 Largura

A largura de barramento é o **número de caminhos elétricos (linhas)** que o compõem. Quanto mais linhas um barramento possui:

- Barramento de endereços: mais endereços de memória
- Barramento de dados: mais bits podem (Exemplo: 32 bits -> 64 bits)
- Mais hardware -> maior custo

[1] Pode transferir dados nos dois sentidos, mas não em ambos simultaneamente (eles são tipicamente utilizados quando qualquer um dos dispositivos pode ser a origem e qualquer um outro pode ser o destino).

[2] Só pode transferir dados em um sentido (tipicamente usado para interligar dois dispositivos um dos quais é sempre a origem e o outro é sempre o destino).

3.2.2 Tipo

Barramento Dedicado

Barramentos distintos carregam informações de dados e de endereços; Alto desempenho; Alto custo

Barramento Multiplexado

Informações de dados e de endereços são multiplexadas em um barramento compartilhado; Redução de custos; Sistema mais lento

3.2.3 Temporização

Temporização Síncrona

São barramentos que incluem um **sinal de clock** nas **linhas de controle** e um protocolo fixo para comunicação, que é relativo ao clock.

Aplicação: barramentos processador-memória.

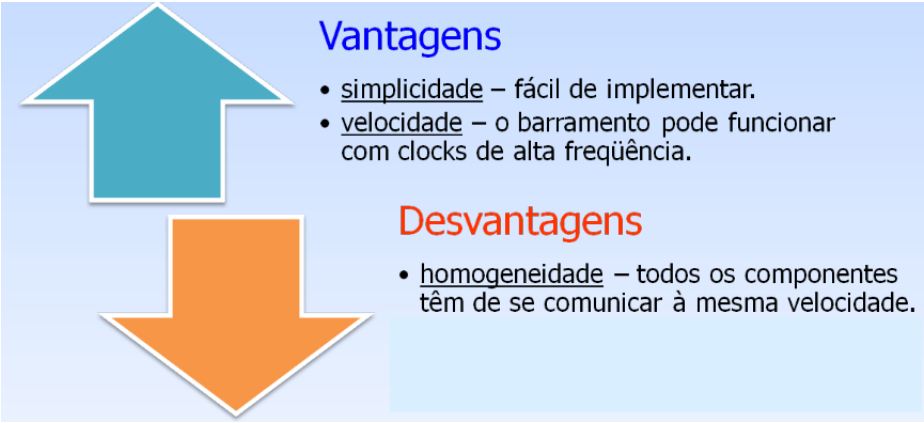


Figura 3.2: Vantagens e Desvantagens de **Barramentos Síncronos**.

Temporização Assíncrona

Não há sinal de clock; Utilizam um protocolo de **handshaking** (exemplo na figura 3.3) para coordenar o uso do barramento.

3.2.4 Arbitragem

Uma disputa acontece quando dois ou mais dispositivos E/S tentam acessar ao mesmo tempo o barramento comum, como por exemplo na figura 3.5

Para evitar disputa e disciplinar o acesso dos dispositivos E/ S, utiliza-se um arranjo **master-slave**:

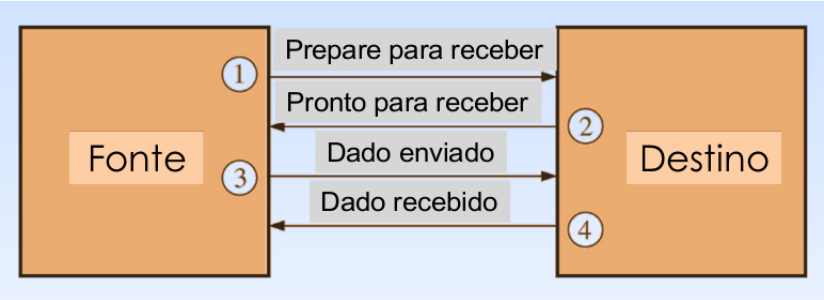


Figura 3.3: Exemplo básico de uma sequência de handshaking.

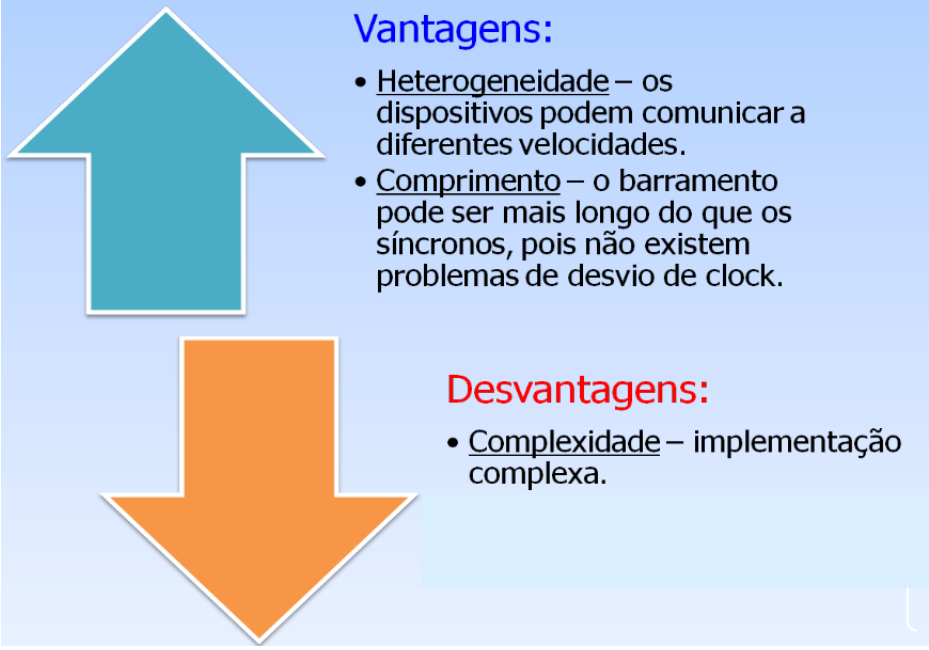


Figura 3.4: Vantagens e Desvantagens de **Barramentos Assíncronos**.

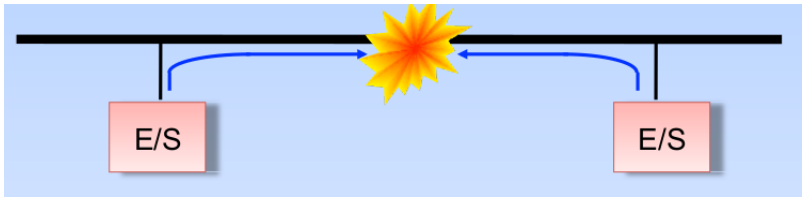


Figura 3.5: Dois dispositivos E/S solicitam o barramento ao mesmo tempo.

- Somente o mestre do barramento (**bus master**) pode controlar o acesso ao barramento.
- Ele inicia e controla todas as requisições do barramento.
- Um **bus slave** responde às requisições.

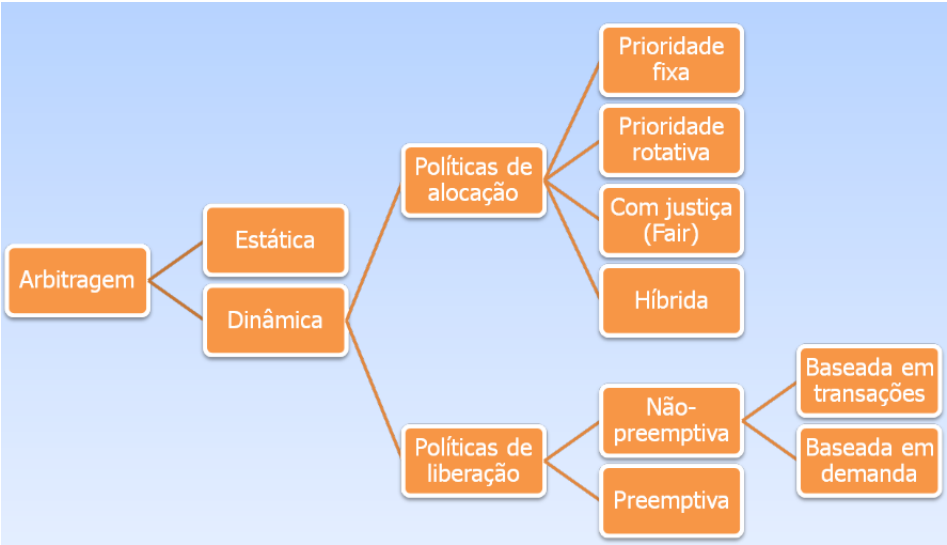


Figura 3.6: Arbitragem - Taxonomia.

Arbitragem Estática

- Barramento somente é alocado em **resposta a uma aquisição**
- Duas linhas: O mestre usa a linha **bus request** para solicitar uso do barramento. Antes de usar o barramento, o mestre deve receber permissão pela linha **bus grant**.

Arbitragem Dinâmica

- O controle do barramento é compartilhado de forma **pré-determinada**.
- Implementação **fácil**.
- Não leva em consideração **necessidades** dos dispositivos.
- Utilização ineficiente: o barramente é alocado mesmo quando não é preciso.

Focando na **arbitragem estática** e, como visto na figura 3.6, veremos as políticas de alocação.

Políticas de alocação

- **Prioridade**:
  - O árbitro decide qual dispositivo terá uso do barramento a partir da sua prioridade.
  - Exemplo:
    - 0 - Sinal de clock da placa mãe;
    - 1 - Teclado
    - 2 - Livre
    - 3 - COM 2
    - 4 - COM 1
    - 5 - Disco Rígido
    - 6 - Drive de disquetes
    - 7 - Porta paralela
- **Justiça**:
  - O árbitro cede o barramento por um tempo determinado e igual para todos os dispositivos, sem prioridade.
- **Híbrida**:
  - Justiça + Prioridade

-> Arbitragem - Daisy Chaining Quando o árbitro vê uma requisição de barramento, envia uma confirmação através da linha **bus grant**.

O dispositivo mais próximo ao árbitro, verifica se foi ele quem pediu acesso. Caso **positivo**, ele toma controle do barramento, sem propagar o sinal de **grant**. Caso **negativo**, **repassa o sinal de grant** para frente.

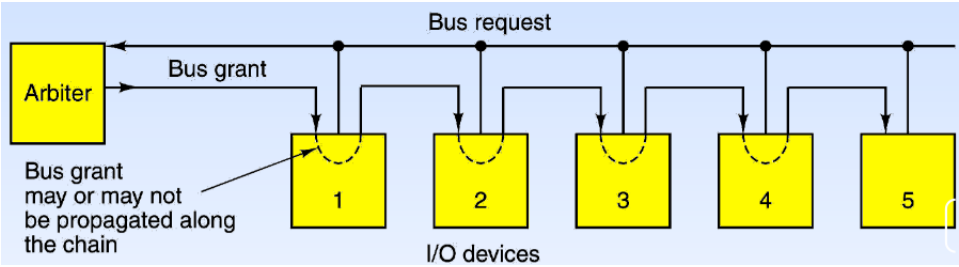


Figura 3.7: Arbitragem - Daisy Chaining

-> Arbitragem - Requisições independentes Cada dispositivo mestre é conectado ao árbitro central por linhas de **grant** e **request** separadas.

Árbitro central pode utilizar diversas políticas de alocação para definir quem deve ter acesso ao barramento. **Desvantagem**: implementação complexa.

-> Arbitragem - Híbrida Dispositivos mestre com prioridades semelhantes são divididos em **classes**. Cada classe possui suas próprias linhas de **request** e **grant**.

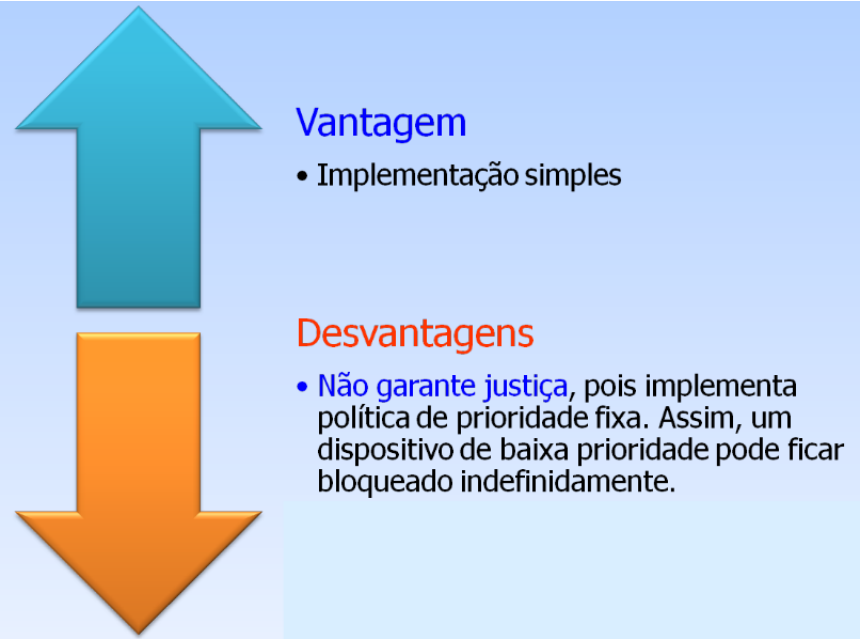


Figura 3.8: Arbitragem - Daisy Chaining - Vantagens e Desvantagens

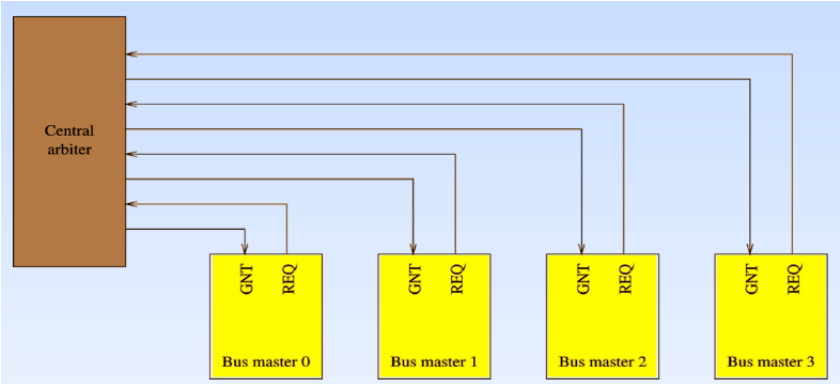


Figura 3.9: Arbitragem - Requisições independentes

Dentro de cada classe, o barramento é conectado usando **daisy-chaining**.

-> Arbitragem Distribuída Os próprios dispositivos mestres determinam quem deve acessar o barramento no próximo ciclo de transações.

Podemos ter **versões distribuídas** dos esquemas de **daisy-chaining** e de **requisições independentes**.

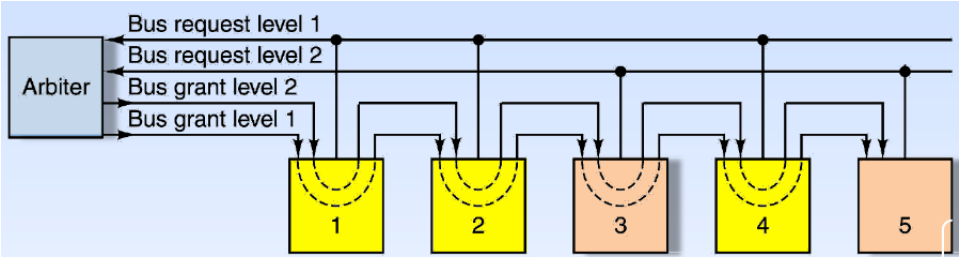


Figura 3.10: Arbitragem - Híbrida

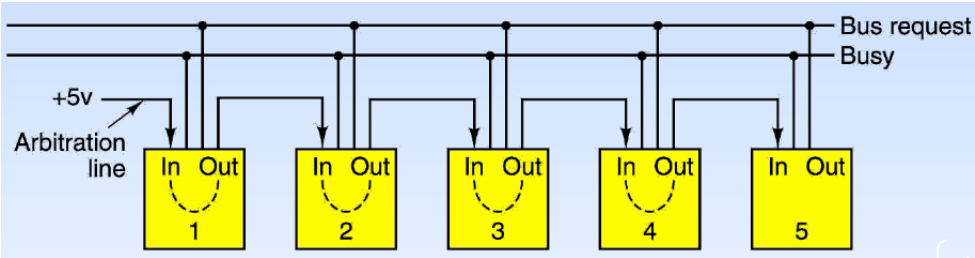


Figura 3.11: Arbitragem Distribuída

3.2.5 Hierarquia

Um grande número de dispositivos conectados a um barramento pode prejudicar o desempenho do sistema.

As principais causas são:

- Quanto maior o número de dispositivos conectados, maior é o comprimento de um barramento, e assim, maior o atraso na propagação de sinais.
- O barramento pode se tornar um gargalo do sistema quando a demanda agregada por transferência de dados se aproxima da capacidade do barramento.

Solução: Utilizar múltiplos barramentos, geralmente dispostos em uma hierárquia.

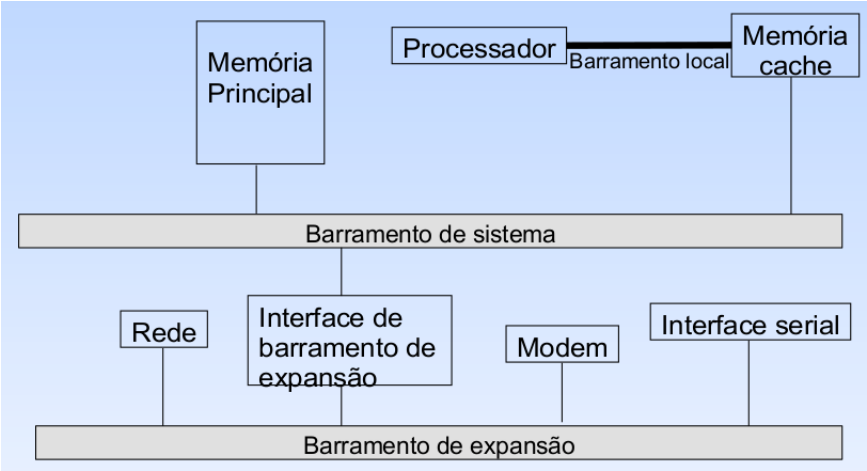


Figura 3.12: Hierarquia de Barramentos

Capítulo 4

Sistema Operacional

Com o passar do tempo, um computador foi capaz de aceitar muitos programas e muitos periféricos. Com isso, surgiu-se também a **necessecidade de gerenciamento** disso tudo. Dessa forma, foram desenvolvidos os **Sistemas Operacionais** (SO).

Um SO é responsável por gerenciar recursos, controlar a execução dos programas pela CPU e fazer a interface usuário-hardware. Veja na figura 4.1 uma exemplificação de uma pirâmade de componenetes de um computador.

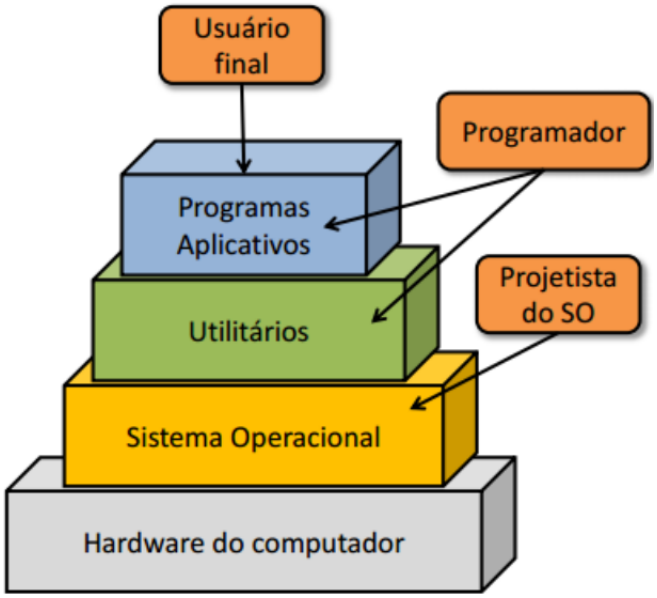


Figura 4.1: Sistema Operacional é uma das bases para um computador.

4.1 Introdrução

Seviços fornecidos pelo SO:

- Execução de programas
- Acesso aos dispositivos de E/S
- Acesso controlado aos arquivos
- Acesso ao sistema
- Gerenciamento de memória virtual
- Detecção e tratamento de erros
- Monitoramento do sistema

4.2 Execução

A figura 4.2 exemplifica como ocorre a execução de um SO.

4.3 Características

Interrupções SO pode ganhar o processador mesmo quando estiver executando programas dos usuários

Proteção de memória Impede que um programa do usuário altere a área de memória que contém o SO, ou outro programa de usuário

Temporização Evita monopólio de um programa

Instruções privilegiadas Algumas instruções só podem ser executadas pelo SO (instruções de E/S, por exemplo)

4.4 Tipos de Sistema Operacional

Os SO podem ser classificados em vários tipos. Vamos ver apenas um deles, que é apenas **quanto à execução simultânea**:

- **Monoprogramação**: executa totalmente 1 programa/ vez. Ex.: MS-DOS
- **Multiprogramação**: Diversos programas são carregados simultaneamente na Memória, e o tempo do processador é dividido entre eles. Ex.: Windows, Linux

4.4.1 Monoprogramação

Potencialmente lento. Nos sistemas monoprogramados o que temos é a existência de um único processo sendo executado de cada vez na memória.. Entenda melhor como funciona na figura 4.3

- **Problema**: Potencialmente lento
  - Lentidão dos dispositivos E/S
- **Solução**: Multiprogramação
- Requer hardware que suporte:
  - Interrupções de E/S
  - DMA
  - Gerenciamento de memória
  - Algoritmo de escalonamento
    - Controla a execução dos programas

4.4.2 Múltiplos programas

Com a multiprogramação existem vários processos na memória aptos à executar e um em execução. Sem dúvida, o conceito de multiprogramação é um dos mais importantes nos sistemas operacionais modernos. Se existirem vários programas carregados na memória ao mesmo tempo, a CPU pode ser compartilhada entre eles, aumentando a eficiência da máquina e produzindo mais resultados em menos tempo. A idéia por detrás da multiprogramação é bastante simples.

Quando um programa libera a CPU, seja para realizar alguma operação de E/S ou por outro motivo, ela fica parada. Enquanto espera que o programa volte para executar, a CPU não realiza nenhum trabalho útil. Para acabar com a ociosidade deste tempo vários programas são mantidos ao mesmo tempo na memória e o sistema operacional se encarrega de escolher um deles para executar.

Assim, sempre que um programa é interrompido, um outro é escolhido para ser executado em seu lugar. Com isso, a CPU estará durante grande parte do tempo ocupada processando instruções de programas. Entenda melhor analisando a figura 4.4.

Os benefícios da multiprogramação são vários: aumento da utilização da CPU e da taxa de saída do sistema computacional, isto é, da quantidade de trabalho realizada dentro de um intervalo de tempo (**throughput**).

4.4.3 Sistemas de Tempo Compartilhado

Consiste no **compartilhamento do tempo do processador**. Possibilita que vários usuários usem o sistema simultaneamente (multiusuário):

- Multiprogramação permite a execução de várias tarefas interativas
- SO é responsável por intercalar a execução dos programas de usuário

Visa **minimizar o tempo de resposta**.

4.5 Complexidade do SO

SO multiprogramado é mais complexo que o monoprogramado.

Ele Requer carregamento dos programas na memória (gerência de memória) e Escolher qual programa deve ser executado (**escalonamento [??]**)

4.6 Escalonamento

Consite na escolha entre **processos**. É a chave para a multiprogramação e transforma tarefas (no disco) em processos.

- Tipos de Escalonamento
  - **Longo Prazo**: Devo executar um novo processo?
  - **Médio Prazo**: Devo acrescentar este processo na memória principal?
  - **Curto Prazo**: Qual destes processos na memória deverá ser executado?
  - **E/S**: Qual E/S pendente deve ser atendida?

4.6.1 Escalonamento de Longo Prazo

Converte (ou não) uma tarefa em um novo processo, que é colocado na fila do escalonador de médio prazo. Executado com frequência relativamente baixa. Observe a figura 4.5.

4.6.2 Escalonamento de Médio Prazo

Acrescenta na MP um novo processo ao conjunto de processos a serem executados (ver figura 4.6). Parte da função de troca de processos (**swapping**).

4.6.3 Escalonamento de Curto Prazo

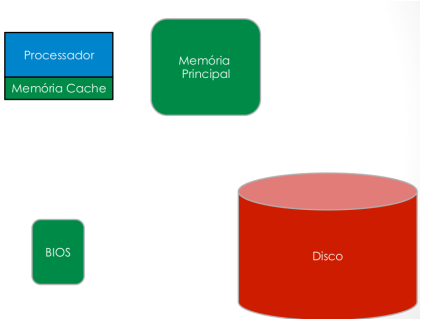
Conhecido como despachante (dispatcher), decide qual dos processos residentes na MP deve ser executado (figura 4.7). É executado com frequência.

4.6.4 Escalonamento de E/S

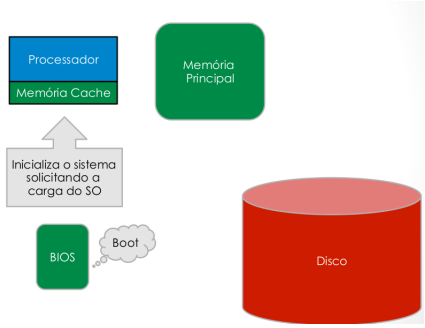
Decide qual processo deve ser atendido por um dado dispositivo de E/S (4.8).

4.6.5 Resumindo...

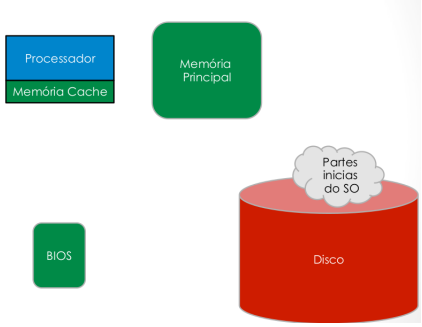
Como síntese, tente analisar as iamgens 4.9 e 4.10.



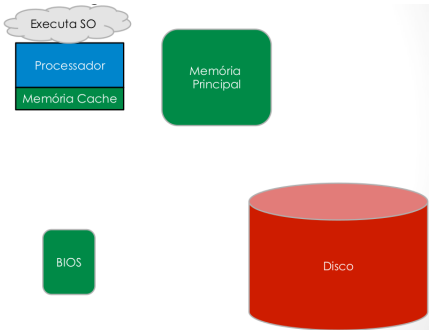
(a) Estado inicial de um computador



(b) Após o boot, o SO é carregado na memória



(c) Partes do SO estão no disco



(d) O processador executa o SO

Figura 4.2: Execução de um Sistema Operacional.

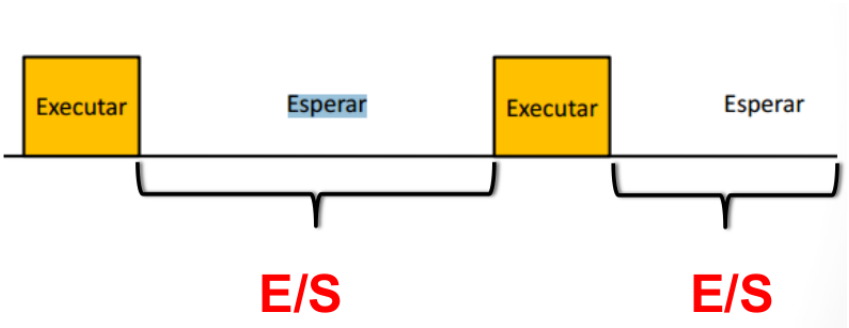


Figura 4.3: SO do tipo Monoprogramável.

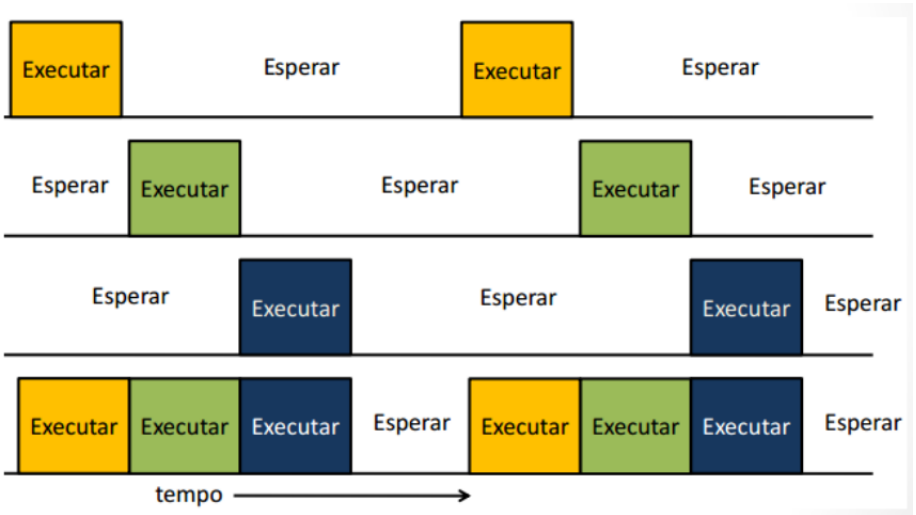


Figura 4.4: SO do tipo multiporgramável.

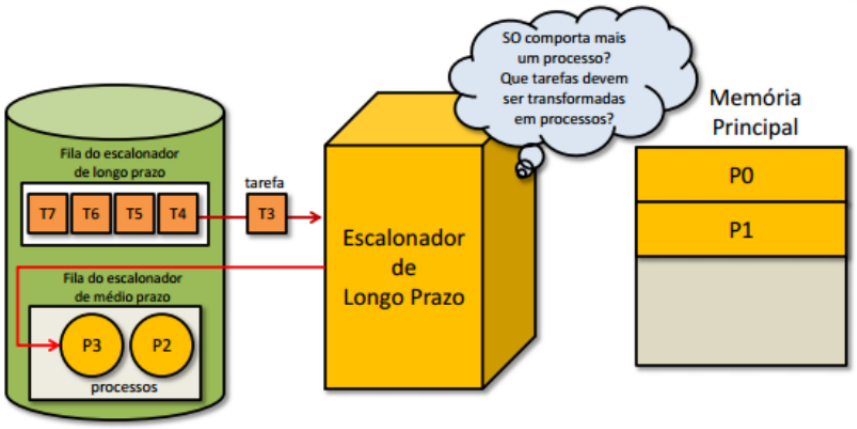


Figura 4.5: Escalonamento de longo prazo.

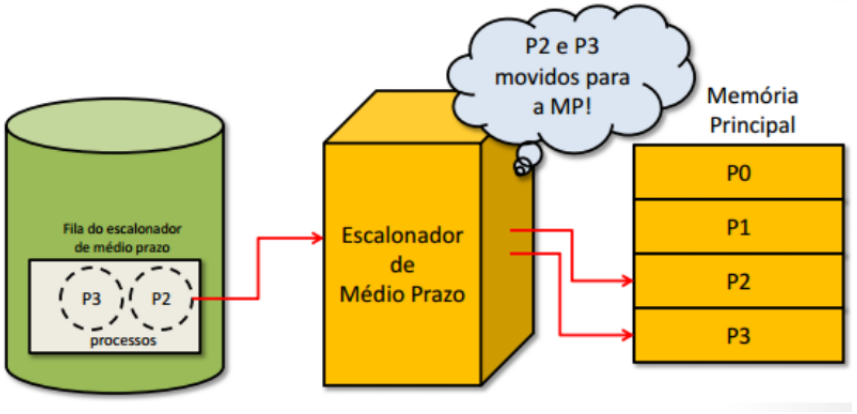


Figura 4.6: Escalonamento de médio prazo.

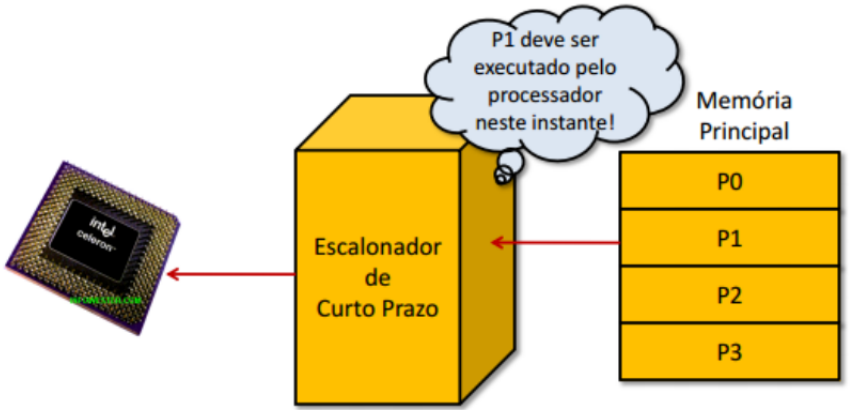


Figura 4.7: Escalonamento de curto prazo.

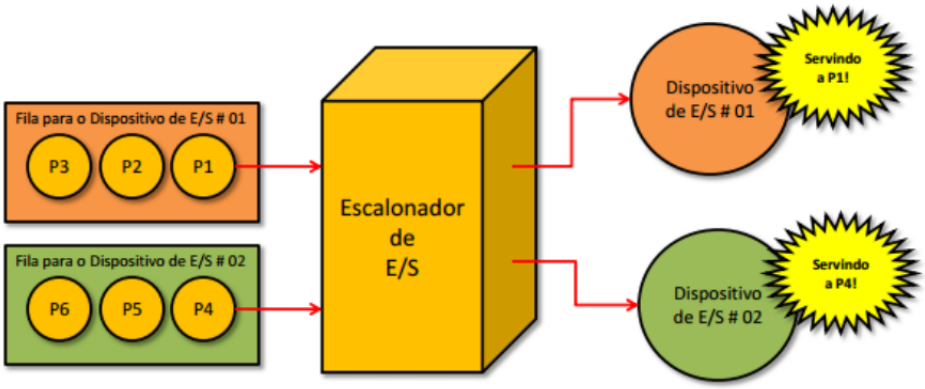


Figura 4.8: Escalonamento de E/S.

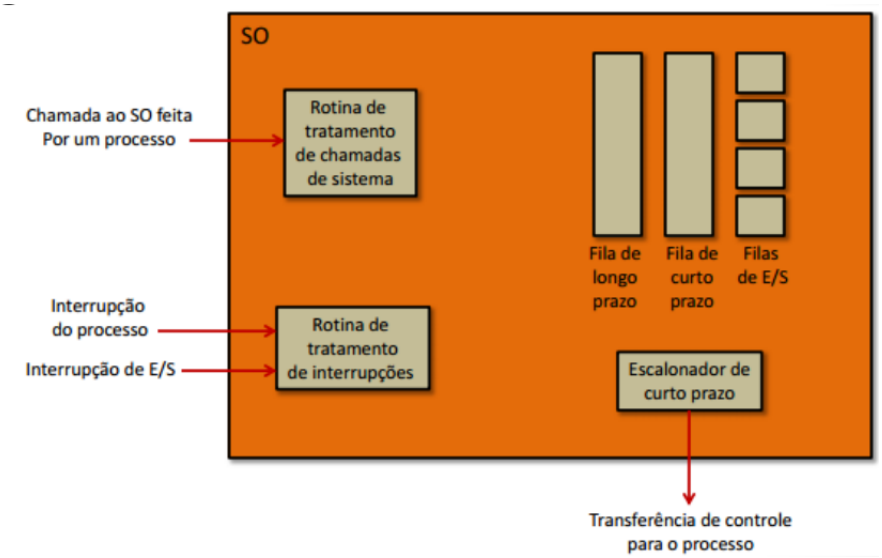


Figura 4.9: Principais Elementos de SO no gerenciamento de processos.

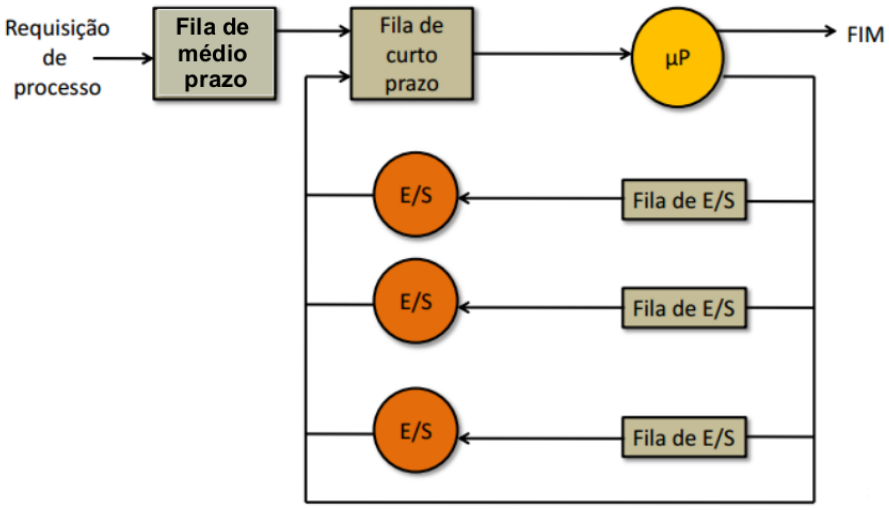


Figura 4.10: Escalonamento de Tarefas

O Sistema Operacional em síntese

Com o objetivo de executar programas, é necessário um SO para (i) Gerenciar recursos (ii) Proteger informações (iii) Escolar os processos.

Para o funcionamento do SO, faz-se utilização de diversos componentes de HW, como registradores, memórias, buffer e controladores.

Capítulo 5

Multithreading e Multicore

Mesmo processadores superescalares não conseguem obter desempenho máximo. Isso porquê ainda assim irão existir dependências de dados, acesso a memória, latência de E/S e etc. Isso gera um desperdício horizontal e vertical (figura 5.1).

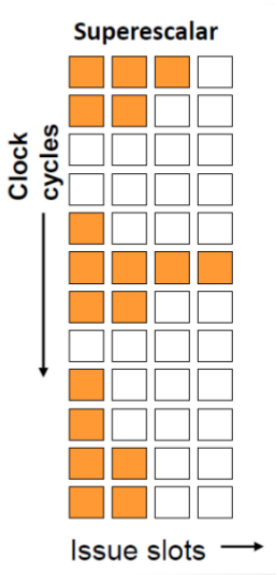


Figura 5.1: Superscalar não garante desempenho máximo.

5.1 Multithreading em hardware

Thread é um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas. É o termo em

inglês para Linha ou Encadeamento de Execução. Essas tarefas múltiplas podem ser executadas simultaneamente para rodar mais rápido do que um programa em um único bloco ou praticamente juntas, mas que são tão rápidas que parecem estar trabalhando em conjunto ao mesmo tempo.

É preciso desenvolver uma técnica de suporte em hardware a múltiplas threads. Entretanto, a mudança de contexto de uma thread para outra leva muito tempo.

Para suportar este cenário, precisamos de mais contadores de programa (PC) e registradores de contexto para evitar register spilling (i.e. guardar registradores na memória).

Dessa forma, várias threads estarão em execução na CPU ao mesmo tempo e a latência da mudança diminui consideravelmente.

5.2 Multithreading

- Duas ou mais threads podem executar virtualmente de forma simultânea no mesmo processador.
- **Replicação** do estado que mantém as threads, porém com **compartilhamento** da maior parte dos recursos de hardware.
- Hierarquia de memória compartilhada
- Objetivos:
  - Melhor utilização de recursos (cache e unidades de execução são compartilhadas entre threads)
  - Baixo consumo de área ($< 5\%$ da área do chip)
  - Redução do tempo de execução da aplicação

Praticamente todos os processadores modernos de propósito geral utilizam multithreading. Na descrição nominal do processador destaca-se o número de threads suportadas.

HyperThreading é um caso de Multithreading proposto pela Intel. Disponível em processadores como Xeon, Pentium 4, Atom e benchmarks da Intel apontam ganho médio de 30de desempenho.

- Vantagens da abordagem:
  - Quando uma thread gera muitos misses (requerendo dados da memória principal), outras threads podem usar os recursos do processador. Reduzindo a ociosidade do processador
  - Quando mais de uma thread executa sobre a mesma área de dados, elas podem compartilhar a cache, reduzindo o tempo de execução global do sistema
- Desvantagens da abordagem
  - As threads podem interferir umas com as outras no compartilhamento de recursos de hardware como cache

5.3 Tipos de Multithreading

Existem três tipos de multithreading.

- Granularidade grossa (coarse grained) -> Block multithreading
- Granularidade fina (fine grained) -> Interleaved multithreading
- Simultaneous multithreading (SMT) -> Multithreading simultâneo

5.3.1 Coarse-grain (granularidade grossa)

- Processador troca de threads somente em paradas (stalls) longas
- Exemplo: quando dados não estão na cache ou E/S
- Vantagens:
  - Solução simples para melhorar a vazão do pipeline
  - Pode ser utilizada em pipelines simples (não- superescalares)
- Desvantagens:
  - No caso de várias paradas curtas, não existe troca de threads, portanto, existe perda de tempo

5.3.2 Fine-grain (granularidade fina)

- Processador troca de thread a cada instrução
- Usa escalonamento Round-Robin[1]
- Pula threads paradas (stalled)
- Processador deve trocar threads a cada ciclo de clock
- Vantagens:
  - Reduz/evita a necessidade de paradas (bolhas) devido a dependência de dados, devido ao entrelaçamento das threads
  - Aumenta a percepção de paralelismo espacial entre as threads. Escalonamento mais "justo" entre as threads
- Desvantagens:
  - Muito custo para trocar de threads a cada ciclo

[1] A idéia do algoritmo é a seguinte. Uma pequena unidade de tempo, denominada timeslice ou quantum, é definida. Todos os processos são armazenados em uma fila circular. O escalonador da CPU percorre a fila, alocando a CPU para cada processo durante um quantum. Mais precisamente, o escalonador retira o primeiro processo da fila e procede à sua execução. Se o processo não termina após um quantum, ocorre uma preempção, e o processo é inserido no fim da fila. Se o processo termina antes de um quantum, a CPU é liberada para a execução de novos processos. Em ambos os casos, após a liberação da CPU, um novo processo é escolhido na fila. Novos processos são inseridos no fim da fila.
Quando um processo é retirado da fila para a CPU, ocorre uma troca de contexto, o que resulta em um tempo adicional na execução do processo.

5.3.3 Simultaneous Multithreading

- É uma variação de multithreading para processadores
  - **Superescalares** (ou seja, várias unidades de processamento) com escalonamento dinâmico de threads e instruções
  - Escalonamento de instruções de threads diferentes
  - Instruções de threads diferentes podem executar paralelamente desde que haja unidades funcionais livres
  - Explora paralelismo ao nível de instrução (Instruction Level Parallelism - ILP)

Observe, na figura 5.2, um parelelo entre os três tipos vistos.

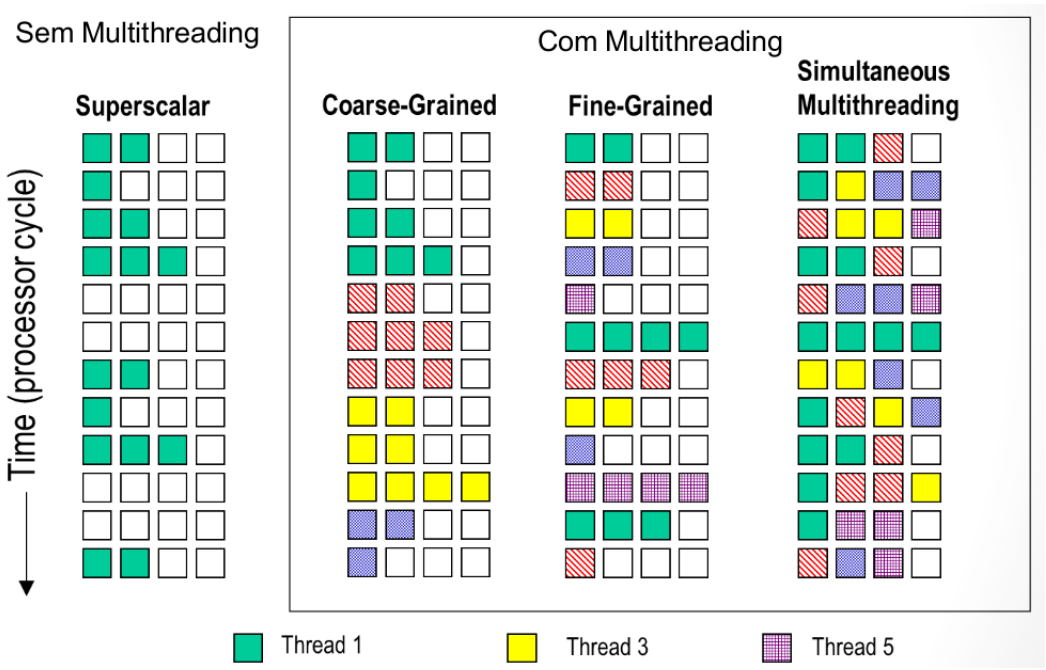


Figura 5.2: Tipos de Multithreading.

- Algumas mudanças são necessárias para o suporte a SMT
  - Múltiplos registradores PCs
  - Pilhas de retornos separadas para cada thread
  - Identificação da thread em cada entrada da tabela de desvios
  - Um banco de registradores grande, com registradores para as threads e registradores adicionais para renomeação
  - Dois estágios no pipeline para acesso aos registradores

5.4 Multicore

CPU com um único processador que contém dois ou mais núcleos. Compartilham recursos computacionais, como barramentos, cache L2/L3.

Alcançam alto desempenho por dois motivos: (i) Vários núcleos executando em paralelo e (ii) tempo curto de comunicação, já que estão em uma mesma CPU.

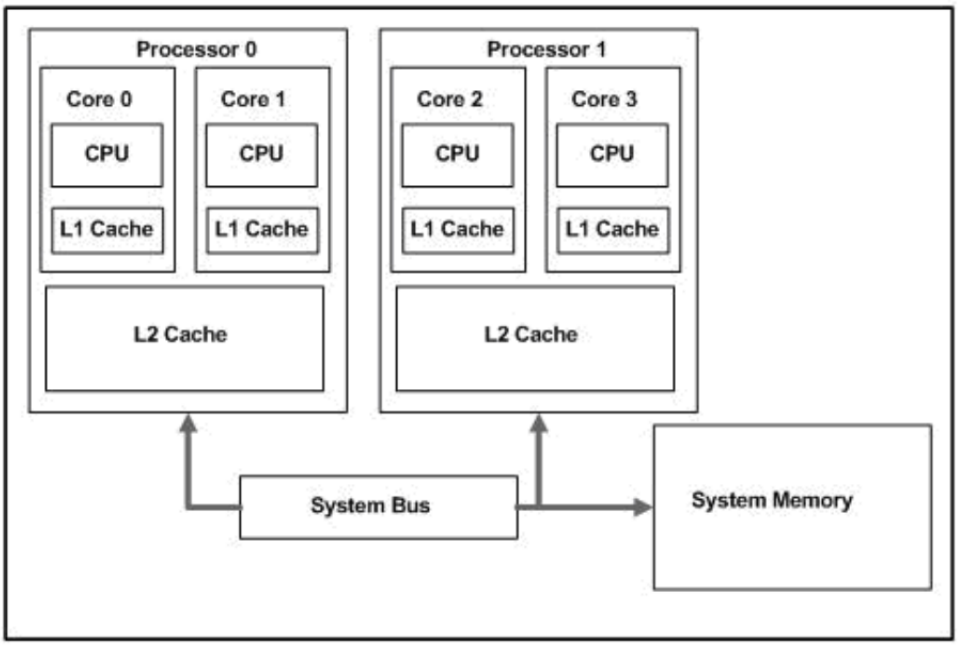


Figura 5.3: Sistema Multicore.

5.5 Tipos de Multicores

5.5.1 Multicores homogêneos

Todos os processadores são iguais. Em algum aspecto seja ele organizacional (monociclo, multiciclo, pipeline) ou arquitetural (ISA). São mais simples de desenvolver e testar (arquitetura regular).

5.5.2 Multicores heterogêneos

Pelo menos UM processador é diferente. Em algum aspecto seja ele organizacional (monociclo, multiciclo, pipeline) ou arquitetural (ISA).

Possui implementação complexa pois testes de comunicação são necessários. Entretanto dão mais opções de execução: Processadores mais rápidos porém maiores vs. Processadores mais menores porém mais lentos.

5.5.3 Comunicação

- Redes-em-Chip (NoC) vs Barramentos
  - Mais recentes pesquisas em MPSoCs tem feito uso de NoCs
    - Solução de interconexão com maior escalabilidade do que barramentos
  - Alto grau de paralelismo
    - Reflete em organizações de memória e modelos de programação

5.6 Organizações de Memória

5.6.1 Memória Distribuída

Organização onde a memória está associada a um único processador
$\cdots$ MUTICORE TEM MAIS COISA, MAS... $\cdots$

5.7 Programação Paralela

No que diz respeito à programação em sistemas multiprocessados, o modelo está intimamente ligado à organização de memória.

De modo a criar uma abstração para o programador, estes modelos de programação são implementados na forma de bibliotecas.

As mais comuns são: (i) Message Passing Interface (MPI) para Troca de Mensagens (ii) OpenMP para Variáveis Compartilhadas

5.7.1 Message Passing Interface

É uma especificação com múltiplas implementações. Provê várias primitivas.

Inicialmente criada para o uso em computação de alto desempenho, entretanto recentes trabalhos na área de sistemas embarcados têm sido desenvolvidos.

5.7.2 OpenMP

É uma API de programação paralela baseada no uso de threads. Através de diretivas de compilação, o programador define trechos de código a serem executados paralelamente em threads (Típicamente laços).

Também é possível para o programador definir o escopo das variáveis no trecho (Privadas, públicas)

Esconde do programador a necessidade de estruturas de sincronização

5.7.3 Alocação de Tarefas

Múltiplas aplicações e múltiplas threads: Gerência de recursos computacionais de processamento.

Alocação de tarefas: Que processadores executarão que tarefas? Que tarefas se comunicam?

Melhor manter próximas tarefas comunicantes e espalhar tarefas para aumentar paralelismo.

5.7.4 Migração de Tarefas

Será que seria melhor que outro processador executasse esta tarefa? (Escalonador global vs. Local)

Migrar significa copiar dados e instruções para que se tornem disponíveis para execução em outro processador; salvar e copiar o estado do processador e as referências a memória podem mudar (Possível ação do linker e depende da organização da memória).

5.8 Conclusões

- Diversas limitações físicas comprometem o aumento da capacidade de processamento dos sistemas computacionais atuais
- Uma das soluções mais utilizadas nos dias atuais são os MPSoCs
  - Mesmo neste contexto, a hierarquia de memória ainda é um gargalo.
  - Como prover dados de maneira eficiente?
  - Como diminuir a latência na comunicação?
  - Como dar suporte a diferentes modelos de programação?
- Dada a crescente demanda por poder de processamento, o número de processadores de um único chip tende a crescer
  - Uso de GPU para processamento geral
- De modo a gerenciar da melhor maneira, soluções de escalonamento em vários níveis podem surgir
  - Clusterização dos processadores
  - Organizações de memórias heterogêneas entre os clusters
  - Alocação e migração de tarefas considerando diversos aspectos

Referências Bibliográficas

[1] David A Patterson and John L Hennessy. Organização e projeto de computadores: interface hardware/software. 2014.

[2] William Stallings. Arquitetura e organização de computadores 8a edição, 2010.

Lista de Figuras