

AOC

Arquitetura e Organizações de Computadores

Ronierison Maciel

Julho 2024

Quem sou eu?



Nome: Ronierison Maciel / Roni

Formação: Mestre em Ciência da Computação

Ocupação: Pesquisador, Professor e Desenvolvedor de Software

Hobbies: Jogar cartas, ficar com a família no final de semana conversando sobre diversos temas

Interesses: Carros, aprimoramento na área educacional, desenvolvimento de software, data science e machine learning

Email: ronierison.maciel@unirios.edu.br

GitHub: <https://github.com/ronierisonmaciel>

- 1 Introdução
- 2 Arquitetura de computadores
- 3 Sistema de numeração e aritmética binária
- 4 Representação de dados em computadores
- 5 Arquitetura de processadores

O que é um computador?

Um computador digital é uma máquina que resolve problemas executando instruções. Uma sequência de instruções é chamada de programa. Os circuitos eletrônicos de cada computador podem reconhecer e executar diretamente um conjunto limitado de instruções simples, para as quais todos os programas devem ser convertidos. Essas instruções básicas incluem:

- Somar dois números
- Verificar se um número é zero
- Copiar dados de uma parte da memória para outra

Introdução

Juntas, as instruções primitivas de um computador formam uma linguagem chamada **linguagem de máquina**, que permite a comunicação com o computador. Os projetistas de um novo computador devem decidir quais instruções incluir em sua linguagem de máquina.

Introdução

Os projetistas procuram manter as instruções primitivas simples, alinhadas com os requisitos de uso e desempenho do computador, para reduzir a complexidade e o custo dos circuitos. Como a maioria das **linguagens de máquina** é muito simples, seu uso direto pelas pessoas é difícil e tedioso.

Problema

Existe uma lacuna entre o que é conveniente para as pessoas e o que é conveniente para computadores. As pessoas querem fazer **X**, mas os computadores só podem fazer **Y**, o que cria um problema.

Existem duas formas de resolver o problema, criando um novo conjunto de instruções mais fácil para as pessoas usarem, chamado **L1**, enquanto o conjunto original é chamado de **L0**. Essas técnicas se diferenciam na maneira como os programas em **L1** são executados pelo computador, que só entende a linguagem de máquina **L0**.

Linguagens, níveis e máquinas virtuais

Um método de executar um programa em **L1** é substituir cada instrução por uma sequência equivalente de instruções em **L0**. O programa resultante é inteiramente em **L0** e o computador o executa no lugar do programa original em **L1**. Essa técnica é chamada de **tradução**.

A outra técnica é criar um programa em **L0** que trate os programas em **L1** como entrada e os execute diretamente, instrução por instrução, sem gerar um novo programa em **L0**. Essa técnica é chamada de interpretação, e o programa que a executa é um **interpretador**.

Tradução e **interpretação** são semelhantes, pois o computador executa instruções em **L1** usando sequências equivalentes em **L0**. Na tradução, todo o programa **L1** é convertido para **L0** antes de ser executado, e apenas o programa **L0** resultante é carregado e controlado pelo computador durante a execução.

Na interpretação, cada instrução **L1** é examinada e executada imediatamente, sem gerar um programa traduzido. Aqui, o interpretador controla o computador, e o programa **L1** é apenas dados. Ambos os métodos, e uma combinação dos dois, são amplamente utilizados. Em vez de pensar em tradução ou interpretação, imagine uma máquina virtual chamada M1, cuja linguagem seja **L1**. Se essa máquina virtual fosse barata, não precisaríamos da linguagem L0. Programas escritos em **L1** seriam executados diretamente.

Linguagens, níveis e máquinas virtuais

Mesmo que construir M1 seja caro ou complicado, programas ainda podem ser escritos para ela e depois interpretados ou traduzidos por um programa em **L0**. Assim, podemos escrever programas para máquinas virtuais como se elas existissem. Para tradução ou interpretação práticas, as linguagens **L0** e **L1** não devem ser muito diferentes. Isso significa que **L1**, embora melhor que **L0**, ainda não é ideal para todas as aplicações. A solução é criar uma linguagem **L2**, mais fácil para as pessoas, que também pode ser traduzida para **L1** ou executada por um interpretador escrito em **L1**.

Video Example

The Programming Language Guide: **Clique aqui**

A criação de uma série de linguagens, cada uma mais conveniente que a anterior, pode continuar indefinidamente até encontrarmos uma adequada. Cada linguagem se baseia na anterior, formando um computador com várias camadas ou níveis. De acordo com o Figura 1 a linguagem mais simples está na base, enquanto a mais sofisticada está no topo.

Linguagens, níveis e máquinas virtuais

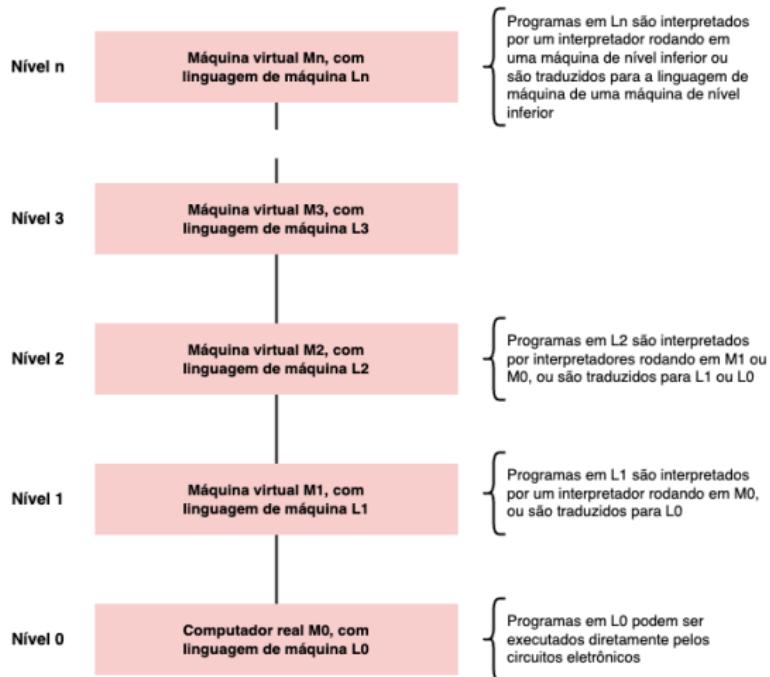


Figure: Máquina multinível.

Linguagens, níveis e máquinas virtuais

Existe uma relação importante entre uma linguagem e uma máquina virtual. Cada máquina tem uma linguagem de máquina, com todas as instruções que pode executar. Uma máquina define uma linguagem, e uma linguagem define uma máquina – aquela que pode executar seus programas. Construir diretamente uma máquina para linguagens como C, C++ ou Java seria muito complexo e caro, mesmo que seja possível com a tecnologia atual. No entanto, tal computador não seria econômico comparado a outras técnicas. Um projeto prático precisa ser não apenas viável, mas também econômico.

Linguagens, níveis e máquinas virtuais

Um computador com vários níveis pode ser visto como várias máquinas virtuais, cada uma com uma linguagem de máquina diferente. Usaremos "**nível**" e "**máquina virtual**" como sinônimos. Apenas programas em L0 podem ser executados diretamente pelo hardware. Programas em **L1, L2, ..., Ln** precisam ser interpretados por um nível inferior ou traduzidos para uma linguagem de nível mais baixo.

Máquinas multiníveis contemporâneas

A maioria dos computadores modernos tem dois ou mais níveis, com alguns chegando a ter até seis níveis, como mostrado na Figura 2. O nível 0, na base, é o hardware real da máquina, cujos circuitos executam programas na linguagem de máquina do nível 1. Abaixo do nível 0, existe o nível de dispositivo, onde o projetista lida com transistores individuais. Este nível, relacionado à engenharia elétrica, não abordaremos na disciplina, porque se quisermos entender o funcionamento dos transistores, entramos no campo da física do estado sólido.

Máquinas multiníveis contemporâneas

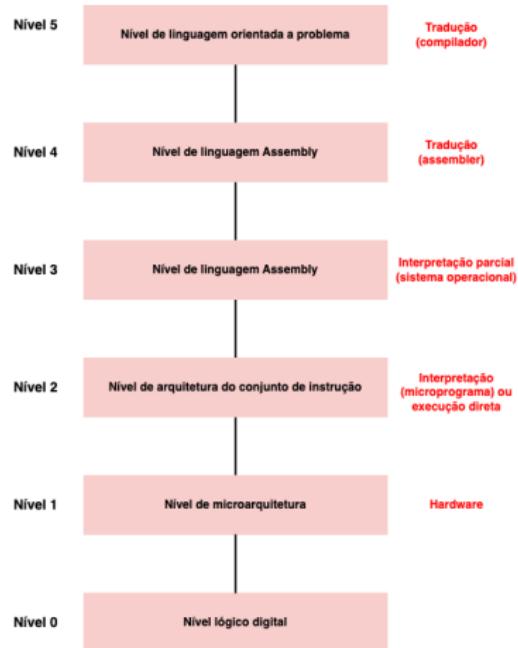


Figure: Um computador com seis níveis

Máquinas multiníveis contemporâneas

No nível lógico digital, estudaremos **portas** (ou gates). Embora feitas de componentes analógicos, como transistores, elas são modeladas como dispositivos digitais. Cada porta tem uma ou mais entradas digitais (0 ou 1) e calcula uma função simples dessas entradas, como **AND** (E) ou **OR** (OU). Algumas portas juntas podem formar uma memória de 1 bit, que armazena **0** ou **1**. Memórias de 1 bit podem ser agrupadas em 16, 32 ou 64 para formar registradores, que armazenam números binários. As portas também formam o mecanismo principal de computação. Estudaremos portas e o nível lógico digital em detalhes mais adiante.

O nível acima é o de **microarquitetura**. Aqui, temos de 8 a 32 registradores formando uma memória local e um circuito chamado **ULA** (Unidade Lógica e Aritmética), que realiza operações aritméticas simples. Os registradores estão conectados à ULA formando um caminho de dados. A operação básica envolve selecionar registradores, a ULA operar sobre eles (por exemplo, somando-os), e armazenar o resultado em um registrador.

Máquinas multiníveis contemporâneas

Chamaremos o nível 2 de nível de arquitetura do conjunto de instrução, ou nível **ISA** (Instruction Set Architecture). Os fabricantes publicam manuais para cada computador, como “Manual de Referência da Linguagem de Máquina” ou “Princípios de Operação do Computador Modelo X”. Esses manuais referem-se ao nível ISA, explicando o conjunto de instruções da máquina. Na verdade, eles descrevem as instruções executadas interpretativamente pelo microprograma ou circuitos de hardware. Se um fabricante tiver dois interpretadores para uma máquina, cada um com um nível ISA diferente, ele precisará fornecer dois manuais de referência, um para cada interpretador.

Máquinas multiníveis contemporâneas

O próximo nível é geralmente híbrido. Muitas instruções estão no nível ISA, mas há também novas instruções, uma organização de memória diferente, a capacidade de executar vários programas simultaneamente e outros recursos. Há mais variação entre os projetos do nível 3 do que entre os dos níveis 1 ou 2.

Máquinas multiníveis contemporâneas

As facilidades do nível 3 são executadas por um interpretador no nível 2, historicamente chamado de sistema operacional. Instruções de nível 3 que são iguais às do nível 2 são executadas diretamente pelo microprograma ou pelo controle de hardware, não pelo sistema operacional. Portanto, algumas instruções de nível 3 são interpretadas pelo sistema operacional e outras diretamente pelo microprograma. Chamamos isso de nível "híbrido". Neste livro, chamaremos esse nível de nível de máquina do sistema operacional.

Máquinas multiníveis contemporâneas

Há uma diferença fundamental entre os níveis 3 e 4. Os três níveis mais baixos não são destinados ao uso do programador comum, mas sim para a execução de interpretadores e tradutores que suportam os níveis superiores. Esses interpretadores e tradutores são criados por programadores de sistemas, especialistas em novas máquinas virtuais. Os níveis 4 e superiores são destinados ao programador de aplicações, que precisa resolver problemas específicos.

Máquinas multiníveis contemporâneas

No nível 4, muda o método de suporte aos níveis superiores. Os níveis 2 e 3 são sempre interpretados, enquanto os níveis 4, 5 e acima geralmente são suportados por tradução. Outra diferença é a natureza das linguagens. Nos níveis 1, 2 e 3, as linguagens de máquina são numéricas, compostas por longas séries de números, boas para máquinas, mas ruins para pessoas. A partir do nível 4, as linguagens usam palavras e abreviações que as pessoas entendem.

O nível 4, a linguagem de montagem (assembly), é uma forma simbólica de uma das linguagens subjacentes. Ele permite que as pessoas escrevam programas para os níveis 1, 2 e 3 de uma maneira mais amigável. Programas em assembly são traduzidos para as linguagens de nível 1, 2 ou 3 e depois interpretados pela máquina virtual ou real adequada. O programa que realiza essa tradução é chamado de assembler.

Máquinas multiníveis contemporâneas

O nível 5 é composto por linguagens projetadas para programadores de aplicações, conhecidas como linguagens de **alto nível**. Exemplos incluem C, C++, Java, Perl, Python e PHP. Esses programas são geralmente traduzidos para níveis 3 ou 4 por compiladores, embora possam ser interpretados. Por exemplo, programas em Java são traduzidos para bytecode Java, que é então interpretado.

Máquinas multiníveis contemporâneas

Alguns níveis 5 consistem em interpretadores para domínios específicos, como matemática simbólica, facilitando a resolução de problemas nesse campo. Em resumo, computadores são projetados em vários níveis, cada um construído sobre o anterior, com diferentes objetos e operações. Isso simplifica o entendimento de um sistema complexo.

Máquinas multiníveis contemporâneas

Cada nível possui uma arquitetura, visível ao usuário, como a quantidade de memória disponível. A tecnologia usada para implementar a memória não faz parte da arquitetura. O estudo da arquitetura de computadores foca nas partes visíveis para os programadores. Na prática, arquitetura de computadores e organização de computadores significam praticamente a mesma coisa.

Evolução de máquinas multiníveis

Para compreender as máquinas multiníveis, é útil examinar sua evolução histórica. Programas escritos na linguagem de máquina real (nível 1) são executados diretamente pelos circuitos eletrônicos do computador (nível 0), sem a necessidade de interpretadores ou tradutores. Esses circuitos, junto com a memória e dispositivos de entrada/saída, formam o hardware do computador. O hardware consiste em objetos tangíveis, como circuitos integrados, placas de circuito, cabos, fontes de alimentação, memórias e impressoras, ao invés de ideias abstratas, algoritmos ou instruções.

Evolução de máquinas multiníveis

Por outro lado, o software consiste em algoritmos, que são instruções detalhadas sobre como realizar tarefas, e suas representações no computador, conhecidas como programas. Esses programas podem ser armazenados em discos rígidos, CD-ROMs ou outros meios, mas a essência do software está no conjunto de instruções que compõem os programas, não no meio físico onde estão gravados.

Evolução de máquinas multiníveis

Nos primeiros computadores, a distinção entre hardware e software era clara. Com o tempo, essa fronteira tornou-se menos definida devido à adição, remoção e fusão de níveis conforme os computadores evoluíam. Hoje, é frequentemente difícil diferenciá-los (Vahid, 2003). Um tema central deste curso é que

hardware e software são logicamente equivalentes.

Evolução de máquinas multiníveis

Qualquer operação executada por software pode ser embutida diretamente no hardware, uma vez que tenha sido suficientemente compreendida. Como observou Karen Panetta: “Hardware é apenas software petrificado”.

O contrário também é verdadeiro: qualquer instrução executada em hardware pode ser simulada em software. A decisão de implementar certas funções em hardware ou software depende de fatores como custo, velocidade, confiabilidade e a frequência de mudanças esperadas. Existem poucas regras fixas para decidir se uma função deve ser implementada em hardware ou software. Essas decisões variam com as tendências econômicas, a demanda e a utilização dos computadores.

A invenção da microprogramação

Os primeiros computadores digitais, na década de 1940, tinham apenas dois níveis: o nível **ISA** (Instruction Set Architecture), onde toda a programação era feita, e o nível lógico digital, que executava esses programas. Os circuitos do nível lógico digital eram complicados, difíceis de entender e montar, e não confiáveis.

A invenção da microprogramação

Poucas máquinas de três níveis foram construídas na década de 1950, mas muitas foram feitas na década de 1960. Por volta de 1970, a interpretação do nível ISA por microprograma, em vez de meios eletrônicos diretos, tornou-se dominante e foi adotada em todas as principais máquinas da época.



A invenção da microprogramação

Em 1951, Maurice Wilkes, da Universidade de Cambridge, sugeriu um computador de três níveis para simplificar o hardware e reduzir o número de válvulas pouco confiáveis. Essa máquina teria um interpretador embutido (microprograma) para executar programas de nível ISA, necessitando de menos circuitos eletrônicos. Isso aumentaria a confiabilidade, reduzindo as falhas.

O que vimos até aqui?

- Linguagens, níveis e máquinas virtuais
- Máquinas multiníveis contemporâneas
- Evolução de máquina multiníveis

O que veremos agora?

- A geração zero - computadores mecânicos (1642-1945)
- A primeira geração - válvulas (1945-1955)
- A segunda geração - transistores (1955-1965)
- A terceira geração - circuitos integrados (1965-1980)
- A quarta geração - integração em escala (1980-?)
- A quinta geração - computadores de baixa potência e invisíveis

Introdução

Durante a evolução dos computadores digitais, foram projetados e construídos centenas de tipos de computadores. Muitos foram esquecidos, mas alguns impactaram significativamente as ideias modernas. Vamos apresentar um breve esboço dos principais desenvolvimentos históricos para entender como chegamos até aqui. Esta seção destaca alguns pontos principais, deixando muitos de fora. A Figura 1.4 mostra máquinas marcantes que discutiremos. Para mais informações históricas, consulte Slater (1987). Biografias e fotos de fundadores da era do computador estão no livro de arte de Morgan (1997).

Computadores mecânicos (1642-1945)

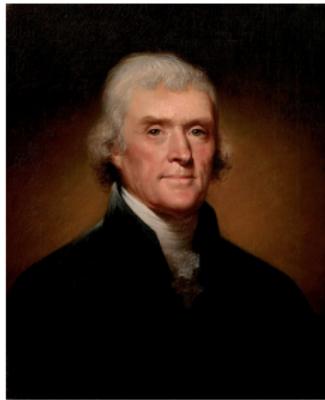


Blaise Pascal (1623–1662), um cientista francês, foi a primeira pessoa a construir uma máquina de calcular operacional. Esse dispositivo, criado em 1642, quando Pascal tinha apenas 19 anos, foi projetado para ajudar seu pai, um coletor de impostos. A máquina era totalmente mecânica, usava engrenagens e funcionava com uma manivela manual. A linguagem de programação Pascal foi batizada em sua homenagem.



Computadores mecânicos (1642-1945)

A máquina de Pascal podia realizar apenas adições e subtrações. Trinta anos depois, o matemático alemão Gottfried Wilhelm von Leibniz (1646–1716) construiu uma máquina mecânica que também podia multiplicar e dividir. Leibniz criou o equivalente a uma calculadora de bolso de quatro operações três séculos atrás.



Válvulas (1955-1965)

O estímulo para o desenvolvimento do computador eletrônico veio da Segunda Guerra Mundial. No início do conflito, submarinos alemães causavam grandes danos aos navios britânicos. As ordens dos almirantes em Berlim eram enviadas por rádio e interceptadas pelos britânicos, mas as mensagens eram codificadas usando o dispositivo ENIGMA. O antecessor do ENIGMA foi projetado pelo inventor amador e ex-presidente dos Estados Unidos, Thomas Jefferson.

Válvulas (1955-1965)



No início da guerra, a inteligência britânica adquiriu uma máquina ENIGMA da inteligência polonesa, que a havia roubado dos alemães. Decifrar uma mensagem codificada exigia muitos cálculos e precisava ser rápido para ser útil. Para isso, o governo britânico criou um laboratório ultrassecreto e construiu o computador eletrônico COLOSSUS, com a ajuda do matemático Alan Turing. Operando desde 1943, o COLOSSUS foi mantido em segredo por 30 anos, tornando-se um beco sem saída. É notável por ser o primeiro computador digital eletrônico do mundo.

Enigma e Colossus

Enigma



Colossus



Figure: Enigma and Colossus Computer

Máquina de Von Neumann

Imagine um mundo onde os computadores eram do tamanho de uma sala inteira, feitos de componentes enormes e cheios de fios. Foi nesse cenário que John von Neumann, um brilhante matemático e cientista da computação, apresentou um projeto revolucionário que mudaria para sempre a forma como os computadores funcionam. Este projeto básico, agora conhecido como máquina de von Neumann, se tornou a base para quase todos os computadores digitais modernos.

A revolução começa

No início dos anos 1940, a computação estava em seus primórdios. Computadores eram grandes, complexos e limitados. Foi então que von Neumann, junto com Herman Goldstine e outros colaboradores, propôs um novo tipo de arquitetura de computador. Essa inovação foi aplicada pela primeira vez no EDSAC, o primeiro computador de programa armazenado. Mais de meio século depois, a arquitetura de von Neumann ainda é fundamental para a maioria dos computadores digitais.

A revolução começa

Imagine um grande armazém cheio de caixas. Cada caixa representa uma palavra de memória. No total, há 4.096 caixas, e cada uma pode guardar até 40 bits. Um bit é como uma pequena luz que pode estar apagada (0) ou acesa (1).

- 1 Memória
 - 2 Unidade de Lógica e Aritmética (ULA)
 - 3 Unidade de Controle
 - 4 Equipamento de Entrada e Saída

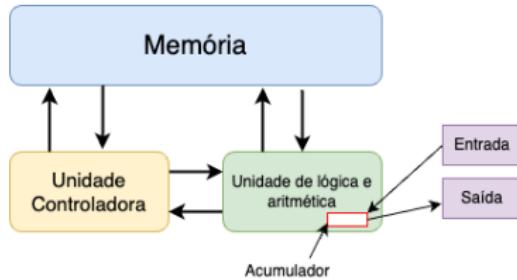


Figure: Máquina original de Von Neumann

A revolução começa

Na máquina de von Neumann original, a ULA e a unidade de controle funcionavam juntas como o "cérebro" do computador. Em computadores modernos, essas duas unidades foram combinadas em um único chip, conhecido como CPU (Central Processing Unit) ou unidade central de processamento. A CPU é o núcleo de qualquer computador, responsável por executar instruções e processar dados.

A segunda geração



Transistores (1955-1965)

Vamos voltar no tempo para o ano de 1948, quando três cientistas brilhantes, John Bardeen, Walter Brattain e William Shockley, fizeram uma descoberta que mudaria o mundo da computação para sempre. Trabalhando no Bell Labs, eles inventaram o transistor, uma pequena peça eletrônica que desempenharia um papel crucial na evolução dos computadores. Por essa invenção, eles foram agraciados com o Prêmio Nobel de Física em 1956.

A era do transistor começa

Imagine um mundo onde os computadores eram gigantescos, cheios de válvulas que funcionavam como interruptores eletrônicos, mas eram grandes, frágeis e consumiam muita energia. A invenção do transistor trouxe uma mudança radical. Esses pequenos dispositivos podiam fazer o mesmo trabalho das válvulas, mas eram muito menores, mais confiáveis e consumiam muito menos energia.

Em apenas dez anos, os transistores revolucionaram a computação. No final da década de 1950, os computadores que usavam válvulas se tornaram obsoletos, dando lugar a máquinas muito mais eficientes e compactas.

O primeiro computador transistorizado

Agora, vamos para o Lincoln Laboratory do MIT, onde um novo tipo de computador estava sendo criado. Imagine um grande laboratório cheio de engenheiros e cientistas trabalhando em uma máquina inovadora. Esse computador de 16 bits seguia a mesma linha do Whirlwind I, um dos primeiros computadores de propósito geral.

TX-0

Este novo computador foi chamado de TX-0 (Transistorized eXperimental computer 0). Pense no TX-0 como um protótipo, um campo de testes para novas ideias e tecnologias. Embora fosse uma máquina experimental, seu design e funcionamento abriram caminho para o desenvolvimento de computadores mais avançados.



Figure: TX-0

A importância do TX-0

O TX-0 foi uma plataforma de teste para o muito mais sofisticado TX-2. Os engenheiros do MIT usaram o TX-0 para experimentar e aperfeiçoar técnicas que seriam utilizadas em computadores futuros. Imagine o TX-0 como um laboratório de inovação, onde cada sucesso e cada falha ajudaram a moldar o futuro da computação.



Figure: TX-2

A história do PDP-8

Imagine a década de 1960, uma época em que os computadores ainda eram gigantescos e caros, acessíveis apenas para grandes empresas e universidades. Foi então que a Digital Equipment Corporation (DEC), uma empresa inovadora no campo da computação, lançou uma máquina que mudaria esse cenário: o PDP-8.



O nascimento de PDP-8: Um computador acessível

Alguns anos após o sucesso do PDP-1, a DEC apresentou ao mundo o PDP-8, um computador de 12 bits. O que tornava o PDP-8 especial não era apenas sua capacidade de processamento, mas sim o fato de ser significativamente mais barato do que o PDP-1, custando cerca de 16 mil dólares. Este preço, embora ainda elevado para os padrões de hoje, era revolucionário para a época e permitiu que pequenas empresas e laboratórios pudessem adquirir sua própria máquina.

• A inovação do barramento Omnibus

Agora, imagine um grande sistema de trilhos, como o que encontramos em uma ferrovia. Esses trilhos permitem que diferentes trens (ou neste caso, diferentes sinais) viajem de um ponto a outro. O PDP-8 trouxe uma inovação similar para o mundo dos computadores: o barramento Omnibus.

A inovação do barramento Omnibus

Um barramento é, essencialmente, um conjunto de fios paralelos que conecta os componentes internos de um computador, permitindo que eles "conversem" entre si. O Omnibus do PDP-8 foi um avanço significativo porque simplificou a maneira como os componentes do computador estavam interconectados, tornando a máquina mais eficiente e fácil de fabricar. Antes dessa inovação, a maioria dos computadores, como a máquina IAS, possuía uma arquitetura centrada na memória, o que complicava a comunicação entre seus componentes.

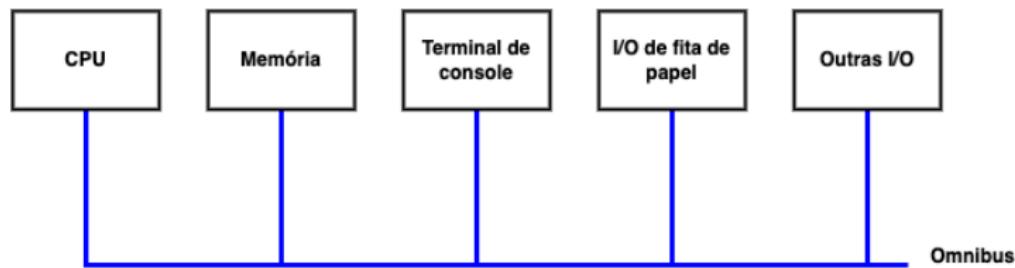


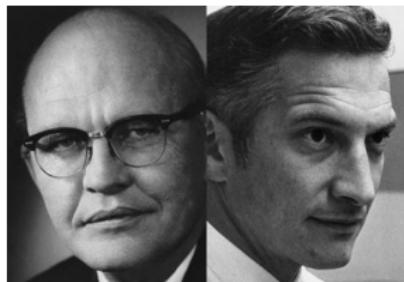
Figure: Barramento omnibus do PDP-8

O impacto do PDP-8: A era dos minicomputadores

O PDP-8 não foi apenas uma máquina de sucesso, mas uma verdadeira revolução. Sua arquitetura, centrada no barramento Omnimbus, foi adotada por quase todos os computadores de pequeno porte que se seguiram. Isso marcou uma ruptura com as arquiteturas anteriores, tornando os computadores mais acessíveis, compactos e práticos.

A DEC alcançou um marco impressionante ao vender 50 mil unidades do PDP-8, consolidando-se como a líder no mercado de minicomputadores. Esta conquista não só democratizou o acesso à computação, mas também lançou as bases para o desenvolvimento de computadores pessoais nas décadas seguintes.

Circuitos integrados (1965-1980)



Hoje vamos conhecer dois pioneiros que mudaram para sempre o mundo da tecnologia: **Jack Kilby** e **Robert Noyce**. Esses dois cientistas, trabalhando de forma independente, fizeram descobertas que se tornaram a base para a transformação radical na computação.

Os visionários da revolução tecnológica

Jack Kilby era um engenheiro eletricista que trabalhava na Texas Instruments. Em 1958, durante um período em que muitos de seus colegas estavam de férias, Kilby continuou trabalhando em uma ideia que poderia resolver um problema crítico na eletrônica: como reduzir o tamanho e o custo dos componentes eletrônicos. Foi nesse contexto que ele inventou o circuito integrado. Kilby conseguiu colocar vários componentes eletrônicos em um único pedaço de silício, criando assim o primeiro chip de circuito integrado da história. Essa invenção rendeu a ele o Prêmio Nobel de Física em 2000.

Os visionários da revolução tecnológica

Enquanto isso, em outra parte do país, **Robert Noyce**, que trabalhava na Fairchild Semiconductor, estava enfrentando desafios semelhantes. Noyce, conhecido por sua habilidade em resolver problemas práticos com soluções inovadoras, também desenvolveu um circuito integrado, mas com uma abordagem ligeiramente diferente. Ele aprimorou o design de Kilby, permitindo a fabricação em massa de chips, o que tornou a tecnologia acessível para a produção em larga escala. Noyce não só foi fundamental na criação do circuito integrado, mas também cofundou a Intel Corporation, uma das empresas mais influentes na história da computação.

A terceira geração de computadores: A revolução dos circuitos integrados

Com a invenção do circuito integrado por Kilby e Noyce, o mundo da computação entrou em uma nova era: a terceira geração de computadores. Antes dessa inovação, os computadores dependiam de transistores, que já eram uma grande melhoria em relação às válvulas, mas ainda apresentavam limitações em termos de tamanho, velocidade e custo.

Os circuitos integrados permitiram que dezenas de transistores fossem colocados em um único chip de silício, o que revolucionou o design dos computadores. Agora, as máquinas podiam ser menores, mais rápidas e mais baratas do que nunca. Essa tecnologia não só aumentou a eficiência dos computadores, mas também abriu caminho para o desenvolvimento de dispositivos que eventualmente se tornariam portáteis e acessíveis ao público em geral.

A terceira geração de computadores: A revolução dos circuitos integrados

Um dos primeiros e mais significativos exemplos dessa nova geração foi o lançamento, em 1964, da série IBM System/360, que utilizava circuitos integrados para oferecer um desempenho sem precedentes. A IBM, já uma gigante na indústria de computadores, consolidou ainda mais sua posição de liderança com essa série de máquinas, que se tornaram um padrão na indústria por muitos anos.

A terceira geração de computadores, impulsionada pelas inovações de Kilby e Noyce, não foi apenas uma evolução tecnológica, mas uma revolução que continua a impactar a maneira como vivemos e trabalhamos até hoje. É graças a esses visionários e suas invenções que podemos desfrutar da tecnologia avançada e acessível que temos em nosso cotidiano.

IBM System/360



Figure: IBM System/360

- **Democratização da computação:** Computadores se tornaram mais acessíveis para empresas menores e universidades.
- **Início da era da informação:** A automação de processos e o uso mais difundido de computadores contribuíram para o aumento da produtividade em diversos setores.
- **Desenvolvimento de softwares mais avançados:** Surgimento de linguagens de programação de alto nível como BASIC e PASCAL.

- **Complexidade de fabricação:** A miniaturização dos componentes aumentou a complexidade na fabricação e manutenção.
- **Restrições de memória:** Apesar dos avanços, a memória ainda era limitada em comparação com as gerações futuras.
- **Problemas de confiabilidade inicial:** Os primeiros ICs ainda apresentavam problemas de confiabilidade, que foram gradualmente resolvidos.

- **Legado:** A terceira geração estabeleceu as bases para o design modular e a arquitetura de computadores que ainda são usados hoje.
- **Transição para a quarta geração:** Marcada pelo uso de microprocessadores (início na década de 1970), que integravam todos os componentes principais de um computador em um único chip.

A quarta geração

- **Período:** Início em 1971 até os dias atuais.
- **Tecnologia principal:** Microprocessadores.
- **Avanço significativo:** Integração de todos os componentes principais de um computador (CPU, memória, interfaces) em um único chip de silício.

O Microprocessador: O coração da quarta geração

- **Intel 4004 (1971):** Primeiro microprocessador comercialmente disponível, desenvolvido pela Intel.
 - **Evolução rápida:** Seguido por processadores como o Intel 8080 e o Motorola 68000, que aumentaram drasticamente a capacidade de processamento.
 - **Revolução no design:** A introdução dos microprocessadores permitiu a criação de computadores pessoais.

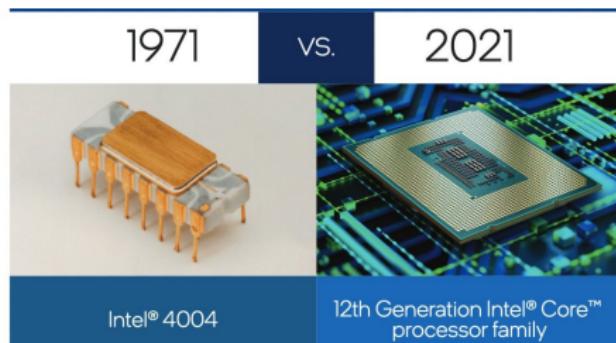


Figure: Intel 4004 vs 12th Intel Core

- **Computadores pessoais (PCs):** Surgimento de PCs como o IBM PC, Apple II, e o Commodore 64.
- **Desempenho e acessibilidade:** Aumento da capacidade de processamento e redução de custos, tornando os computadores acessíveis a indivíduos e pequenas empresas.
- **Sistemas operacionais:** Desenvolvimento e popularização de sistemas operacionais como MS-DOS, Windows, e macOS.

Computadores



Figure: Apple II



Figure: Commodore 64

Uma analogia entre os PCs da quarta geração e os smartphones modernos, destacando a evolução em termos de poder de processamento e acessibilidade.

Pergunta!

Qual linguagem de programação vocês estão aprendendo e como ela pode ter evoluído a partir das linguagens dos anos 70 e 80?

- **Integração em diversas áreas:** Computadores se tornaram essenciais em áreas como medicina, engenharia, educação e entretenimento.
- **Redes e internet:** Expansão das redes de computadores, culminando na popularização da Internet na década de 1990.
- **Desenvolvimento de linguagens de programação:** Crescimento de linguagens como C, C++, e Java, que ainda são amplamente usadas hoje.

Exemplos de computadores da quarta geração

- **IBM PC (1981):** Um dos primeiros computadores pessoais bem-sucedidos, que definiu um padrão para a arquitetura de PCs.
- **Apple Macintosh (1984):** Popularizou a interface gráfica (GUI) e o uso do mouse.
- **Consoles de videogame:** Consoles como o Nintendo Entertainment System (NES) e o Sega Genesis levaram os microprocessadores para o entretenimento doméstico.

Computadores



Figure: IBM PC



Figure: Apple Macintosh

Alguém de vocês ou seus familiares ainda têm algum desses dispositivos ou se conhecem alguém que tenha?

O Papel do software na quarta geração

- **Desenvolvimento de software de escritório:** Pacotes como Microsoft Office se tornaram ferramentas essenciais no ambiente de trabalho.
- **Jogos e entretenimento:** A evolução dos jogos para computadores e consoles, e o surgimento dos primeiros jogos de estratégia, simulação e RPG.
- **Softwares de design e desenvolvimento:** A popularização de softwares de CAD (Design Assistido por Computador) e IDEs (Ambientes de Desenvolvimento Integrado) para programadores.

Softwares



Figure: Windows 1.x



Figure: Paint

Observem como a interface e a usabilidade evoluíram ao longo do tempo 0_0.

Impacto da quarta geração na computação moderna

- Computadores portáteis: Surgimento de laptops e, mais tarde, notebooks e ultrabooks, levando a computação móvel a outro nível.
- **Início da era da internet:** A quarta geração foi crucial para o desenvolvimento da Internet, redes sociais e a computação em nuvem.
- **Perspectivas futuras:** A tecnologia de microprocessadores continua a evoluir, com a Intel, AMD, e ARM liderando o desenvolvimento de processadores mais rápidos e eficientes.

Pergunta!

Como vocês imaginam que a computação continuará evoluindo e quais novas tecnologias podem definir a próxima geração?

A quinta geração de computadores

Contexto histórico:

- Década de 1980
- Iniciativa do Japão com o projeto "Fifth Generation Computer Systems (FGCS)"

Objetivo:

- Criar máquinas com capacidade de raciocínio e processamento de linguagem natural.

Inteligência Artificial (IA):

- Foco em simular a inteligência humana.
- Desenvolvimento de sistemas capazes de aprender e tomar decisões.

Exemplo:

- Sistemas especialistas, como o MYCIN, utilizados para diagnósticos médicos.

Componentes principais de um sistema computacional

- ① CPU (Unidade Central de Processamento): Responsável pelo processamento das instruções e execução de tarefas.
- ② Memória: Armazena dados e instruções temporariamente (RAM) e permanentemente (ROM, discos rígidos).
- ③ Dispositivos de Entrada/Saída (I/O): Permitem a interação com o sistema (teclado, mouse, impressoras, etc.).
- ④ Barramentos (Buses): Estruturas que transportam dados entre a CPU, memória e dispositivos de I/O.

Arquitetura RISC (Reduced Instruction Set Computing):

- Instruções simples e de execução rápida.
- Uso eficiente de pipeline.

Arquitetura CISC (Complex Instruction Set Computing):

- Instruções mais complexas, permitindo operações avançadas.
- Maior número de instruções por ciclo.

Resumo da semana

- Arquitetura refere-se ao design conceitual; organização é sobre implementação.
- A evolução dos computadores passou por diversas gerações, cada uma com avanços significativos.
- Componentes principais: CPU, memória, dispositivos de I/O e barramentos.
- A arquitetura de Von Neumann é um modelo base para muitos sistemas computacionais.

Próximos passos

- **Estudo:** Leia sobre as diferentes arquiteturas (RISC vs. CISC) para a próxima aula.
- **Próxima aula:** Sistemas de numeração e aritmética binária.

Atividade

- **Tarefa:** Identifique e rotule os componentes principais em um diagrama de computador moderno.

Objetivo da aula

- Compreender os diferentes sistemas de numeração.
- Realizar operações básicas de aritmética no sistema binário.
- Aplicar conceitos de aritmética binária na prática.

Introdução aos sistemas de numeração

Sistema de numeração: Conjunto de símbolos e regras usados para representar números.

Principais sistemas:

- **Decimal** (base 10): Utiliza dígitos de 0 a 9 (ex.: 345).
- **Binário** (base 2): Utiliza dígitos 0 e 1 (ex.: 1011).
- **Octal** (base 8): Utiliza dígitos de 0 a 7 (ex.: 645).
- **Hexadecimal** (base 16): Utiliza dígitos de 0 a 9 e letras A a F (ex.: 1A3).

- **Definição:** Sistema de numeração em base 2, usando apenas os dígitos 0 e 1.
- **Importância:** Fundamental para o funcionamento dos computadores, pois os sistemas digitais operam em dois estados: ligado (1) e desligado (0).

Exemplo: Número binário 1010 representa o decimal 10

Conversão entre sistemas de numeração

- Decimal para binário: Divisão sucessiva por 2, anotando os restos.
 - Exemplo: 13 (decimal) → 1101 (binário).
- Binário para decimal: Soma dos produtos de cada dígito binário pelo seu valor posicional (potência de 2).
 - Exemplo: 1101 (binário) = $1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$ (decimal).

Aplicações da aritmética binária

- **Circuitos lógicos:** Operações aritméticas são implementadas em hardware usando portas lógicas (AND, OR, NOT, XOR).
- **Processamento de dados:** Operações de adição, subtração e multiplicação são fundamentais em unidades de processamento.
- **Codificação de dados:** Representação eficiente e compacta de dados em binário facilita a transmissão e armazenamento.

O que são circuitos lógicos?

Pense nos circuitos lógicos como os "cérebros" dos computadores. Eles são como interruptores que podem estar ligados (1) ou desligados (0). Esses interruptores fazem operações básicas de cálculo e decisão.

Como a aritmética binária se aplica?

Usamos operações lógicas para combinar esses interruptores de diferentes maneiras:

- AND (E): Só dá 1 se **ambos** os interruptores estiverem ligados (1).
- OR (OU): Dá 1 se pelo menos **um** dos interruptores estiver ligado (1).
- NOT (NÃO): **Inverte** o estado: se for 1, vira 0, e vice-versa.
- XOR (OU Exclusivo): Dá 1 se apenas **um** dos interruptores estiver ligado, mas não ambos.

O que é processamento de dados?

Quando você usa seu computador para qualquer coisa, como jogar, enviar mensagens, ou navegar na internet, o computador está processando dados. Isso significa pegar informações, fazer cálculos, e tomar decisões com base nesses cálculos.

Como a aritmética binária se aplica?

As operações de adição, subtração, e multiplicação são todas feitas em binário dentro do processador (CPU) do computador.

Por exemplo:

- Somar dois números binários pode dizer ao computador quantas operações ele deve fazer ou qual o resultado de uma ação específica.
- Subtrações e multiplicações ajudam a calcular coisas como gráficos de jogos ou ajustar o volume de som.

O que é codificação de dados?

Codificar dados significa transformar informações em um formato que possa ser facilmente armazenado ou transmitido.

Como a aritmética binária se aplica?

Os computadores usam binário porque é uma maneira eficiente de representar dados. Cada bit (0 ou 1) é uma unidade básica de informação. Isso facilita o armazenamento, porque os dados podem ser compactados e guardados de forma organizada.

- **Transmissão:** Em redes de internet, os dados são enviados em pacotes de bits. Usar binário facilita a conversão dessas informações para que possam ser transmitidas rapidamente.
- **Armazenamento:** Arquivos como fotos, músicas e vídeos são todos armazenados como sequências de 0s e 1s em dispositivos de armazenamento (como pendrives, discos rígidos).

Objetivo da aula

- Compreender como diferentes tipos de dados são representados em sistemas computacionais.
- Aprender sobre a representação de números inteiros e de ponto flutuante.
- Explorar a representação de caracteres usando padrões de codificação.
- A importância da precisão e os limites da representação de dados.

Introdução à representação de dados

Definição:

- Representação de dados refere-se ao formato utilizado para armazenar, processar e transmitir informações em sistemas de computador.

Importância:

- Eficiência na armazenagem e processamento.
- Precisão e integridade dos dados.
- Compatibilidade entre diferentes sistemas e dispositivos.

Representação de números inteiros

Inteiros sem sinal:

- Representam apenas números não negativos.
- Exemplo: 8 bits → varia de 0 a 255.

Inteiros com sinal:

- Podem representar valores negativos e positivos.
- Complemento de Dois: Método comum para representação de números negativos.
 - Exemplo: 8 bits → varia de -128 a 127.

Overflow:

- Ocorre quando o valor excede o limite superior que pode ser representado.
- Exemplo: 8 bits, adicionando $127 + 1$ resulta em -128 (complemento de dois).

Underflow:

- Ocorre quando o valor cai abaixo do limite inferior que pode ser representado.
- Exemplo: 8 bits, subtraindo $-128 - 1$ resulta em 127 .

Processamento de texto:

- Editores de texto e sistemas de comunicação utilizam codificações como ASCII e Unicode para armazenar e manipular texto.

Cálculos científicos:

- Aritmética de ponto flutuante é crucial para precisão em cálculos complexos e simulações.

Sistemas de banco de dados:

- Uso de representações de inteiros e ponto flutuante para armazenamento eficiente e consulta de dados.

Objetivos da aula

- Compreender a estrutura e função de um processador.
- Explorar os principais componentes da CPU: Unidade de Controle, Unidade Lógica e Aritmética (ALU) e Registradores.
- Entender o ciclo de instrução: Fetch, Decode, Execute.
- Discutir o papel dos processadores em arquiteturas modernas.

Introdução à arquitetura de processadores

Definição:

- A arquitetura de processadores refere-se ao design e organização de uma Unidade Central de Processamento (CPU), que é o "cérebro" do computador.

Importância:

- Determina a eficiência, capacidade de processamento e o desempenho geral de um sistema computacional.

Evolução:

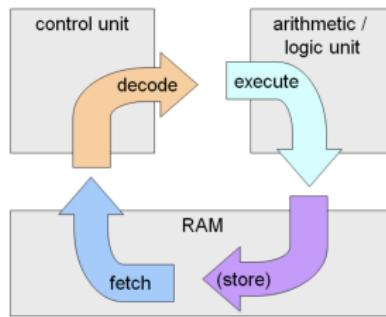
- Processadores evoluíram de simples circuitos para unidades complexas com múltiplos núcleos e capacidades avançadas de processamento paralelo.

Componentes principais da CPU

- ① Unidade de Controle (Control Unit - CU):
 - Gerencia e coordena as operações da CPU.
 - Direciona o fluxo de dados entre a CPU e outros componentes.
- ② Unidade Lógica e Aritmética (Arithmetic Logic Unit - ALU):
 - Executa operações aritméticas (adição, subtração) e lógicas (AND, OR, NOT).
- ③ Registradores:
 - Memórias pequenas e rápidas dentro da CPU.
 - Armazenam dados temporários e instruções durante o processamento.

Ciclo de instrução da CPU

- ① **Fetch** (Buscar): A CPU busca a próxima instrução da memória principal e a armazena em um registrador de instrução.
- ② **Decode** (Decodificar): A Unidade de Controle interpreta a instrução e identifica quais operações precisam ser realizadas.
- ③ **Execute** (Executar): A ALU executa as operações aritméticas ou lógicas especificadas pela instrução.
- ④ **Write-back** (Escrever de volta): O resultado é armazenado em um registrador ou na memória principal.



Processadores de núcleo único:

- Execução de uma única instrução por vez.
- Uso comum em dispositivos mais simples e aplicações específicas.

Processadores multinúcleo:

- Contêm múltiplos núcleos em um único chip, permitindo a execução simultânea de múltiplas instruções.
- Melhoraram a eficiência e desempenho para aplicações multitarefa.

Processadores com Hyper-Threading:

- Permitem que um único núcleo físico execute múltiplas threads, simulando núcleos adicionais.
- Aumentam a capacidade de processamento sem adicionar mais núcleos físicos.

Arquitetura RISC vs. CISC

- RISC (Reduced Instruction Set Computing):
- Conjunto reduzido de instruções simples e rápidas.
- Eficiência em execução e uso de pipeline.
- **Exemplo:** ARM.

CISC (Complex Instruction Set Computing):

- Conjunto mais complexo de instruções, permitindo operações mais avançadas em menos linhas de código.
- Mais flexível, mas potencialmente menos eficiente em comparação a RISC.
- **Exemplo:** x86.

RISC vs. CISC

RISC

- É uma ferramenta simples, mas muito eficientes. Cada ferramenta faz uma coisa só, mas faz rápido. Precisamos de mais passos para fazer uma atividade, mas cada passo é bem rápido e organizado.

CISC

- É como usar um canivete suíço com várias funções. Você pode fazer quase tudo com ele, mas às vezes é complicado e demora para encontrar a função certa. É ótimo para tarefas complexas pois temos uma ferramenta para cada coisa, mas pode ser mais lento para usar.

NOTA:

- ① **RISC:** Poucas ferramentas, passos rápidos, tudo bem organizado.
- ② **CISC:** Uma ferramenta multifuncional, faz muitas coisas de uma vez, mas pode ser mais lento.

Pipeline de processadores

Conceito:

- Técnica para melhorar o desempenho do processador permitindo que múltiplas instruções sejam sobrepostas durante a execução.

Estágios do pipeline:

- Instruções divididas em estágios, como Fetch, Decode, Execute.

Vantagens:

- Aumenta a taxa de instruções executadas por ciclo de clock.

Cache de memória na CPU

Definição:

- Memória de alta velocidade localizada na CPU para armazenar instruções e dados frequentemente usados.

Níveis de cache:

- L1 (mais rápida e menor), L2, L3 (maior e mais lenta).

Importância:

- Reduz a latência e aumenta o desempenho, minimizando a necessidade de acessar a memória principal.

Desempenho em jogos e aplicações de realidade virtual:

- Uso de processadores multinúcleo e otimizações de cache para renderizar gráficos complexos.

Processamento de inteligência artificial e aprendizado de máquina:

- Necessidade de alta capacidade de processamento paralelo e eficiência energética.

Servidores e computação em nuvem:

- Utilização de processadores de alto desempenho com suporte para virtualização e múltiplas threads.

Resumo da aula

- Componentes da CPU: Unidade de Controle, ALU, Registradores.
- Ciclo de Instrução: Processos de buscar, decodificar, executar e armazenar instruções.
- Arquiteturas RISC e CISC: Diferenças e exemplos de uso.
- Pipeline e Cache: Técnicas para melhorar o desempenho e eficiência dos processadores.

Exemplo prático

Imagine que o computador está lento enquanto você usa várias abas no navegador e ouve música ao mesmo tempo. A ferramenta a seguir pode te mostrar que a CPU está sobrecarregada, a memória está quase esgotada ou que o disco está sendo muito utilizado, ajudando a tomar ações corretivas, como fechar programas desnecessários.

Perfmon (Monitor de Desempenho)

O que é?

- O Perfmon, também conhecido como Monitor de Desempenho, é uma ferramenta nativa do Windows que permite acompanhar, em tempo real, o desempenho de um computador. Ele é uma evolução do System Monitor (sysmon.exe).

Para que serve?

- Essa ferramenta ajuda a identificar gargalos e falhas em componentes como CPU, memória, disco e rede, ajudando administradores e usuários a melhorarem o desempenho de seus sistemas.

Como funciona?

- O Perfmon coleta dados de diversas fontes, como contadores de desempenho, rastreamento de eventos e informações de configuração do sistema.
- Você pode utilizá-lo em tempo real ou agendar a coleta de dados para análise posterior, o que é útil para detectar problemas intermitentes.

Perfmon: Como utilizar?

① Passo a passo simples:

- ① Abra o Perfmon (busque por "perfmon.exe" no seu Windows).
- ② Escolha quais parâmetros deseja monitorar (ex.: uso de CPU, rede, memória).
- ③ Veja em tempo real os gráficos e os valores sendo exibidos.
- ④ Utilize as informações para decidir se precisa fechar aplicativos ou realizar upgrades.

Executando o Perfmon

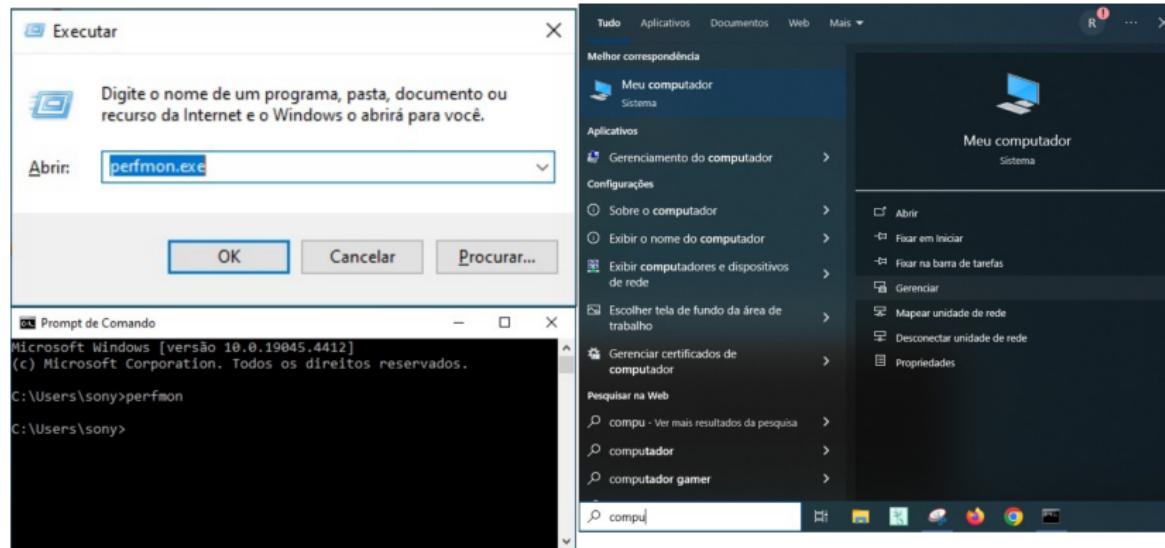


Figure: Executando o Perfmon

Monitor de confiabilidade

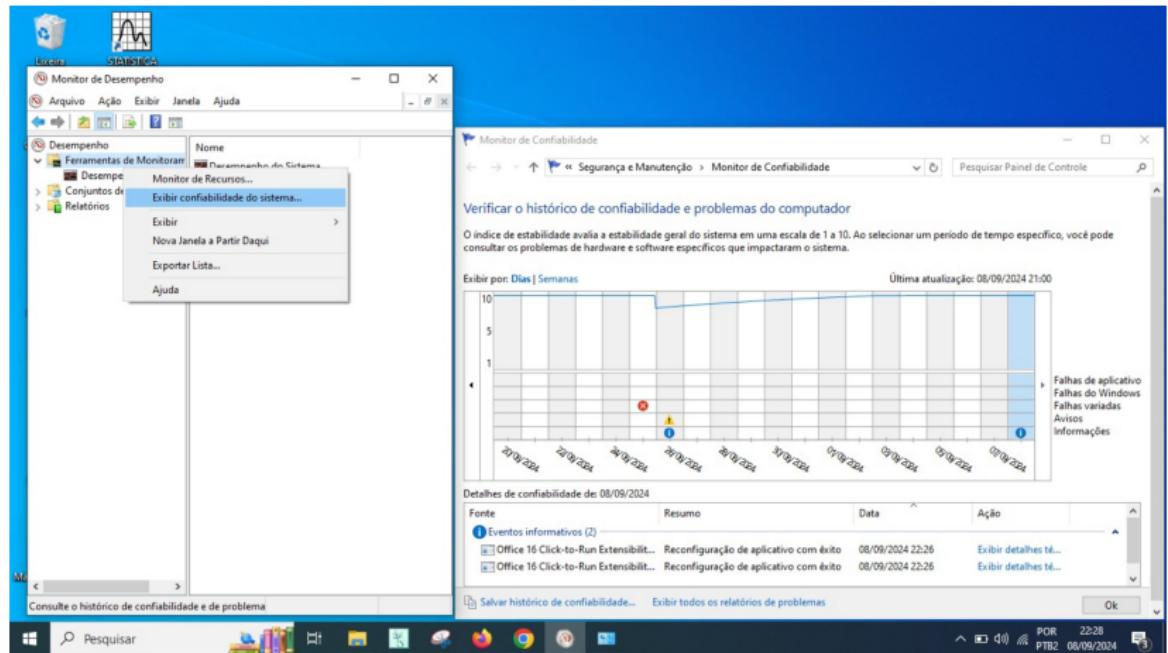


Figure: Monitor de confiabilidade

O Monitor de desempenho coleta dados a partir de três fontes:

- ① **Contador de desempenho**: refletem parte do estado do sistema ou atividade.
- ② **Rastreamento de eventos**: permitem escutar determinados eventos de um sistema ou aplicação.
- ③ **Informação de configuração**: coletado a partir de informações do registro do Windows.

O Monitor de Desempenho agrupa várias métricas coletadas a partir das fontes acima em uma unidade chamada conjunto de coletores de dados

Contadores de desempenho

Contadores de desempenho do processador:

- Processador % tempo do processador
 - Intervalo aceitável: 0 - 85%
- Processador % tempo de usuário
- Processador % tempo de interrupção
 - Intervalo aceitável: 0-15%

Soluções:

- Otimizar aplicativo
- Upgrade da CPU

Contadores de desempenho de memória:

- Memória % Bytes confirmados em uso
 - Intervalo aceitável: 0-80%
- Memória % Mbytes disponíveis
 - Intervalo aceitável: 5% do total da Ram - 100%

Contadores de desempenho

Contadores de desempenho do disco:

- LogicalDisk % espaço livre
 - Intervalo aceitável: 15% - 100%
- PhysicalDisk % tempo ocioso
 - Intervalo aceitável: 20% - 100%
- PhysicalDisk Média de disco Leitura
 - Intervalo aceitável: 0 - 20ms
- PhysicalDisk Média de disco Gravação
 - Intervalo aceitável: 0 - 25ms

Monitorando em tempo real

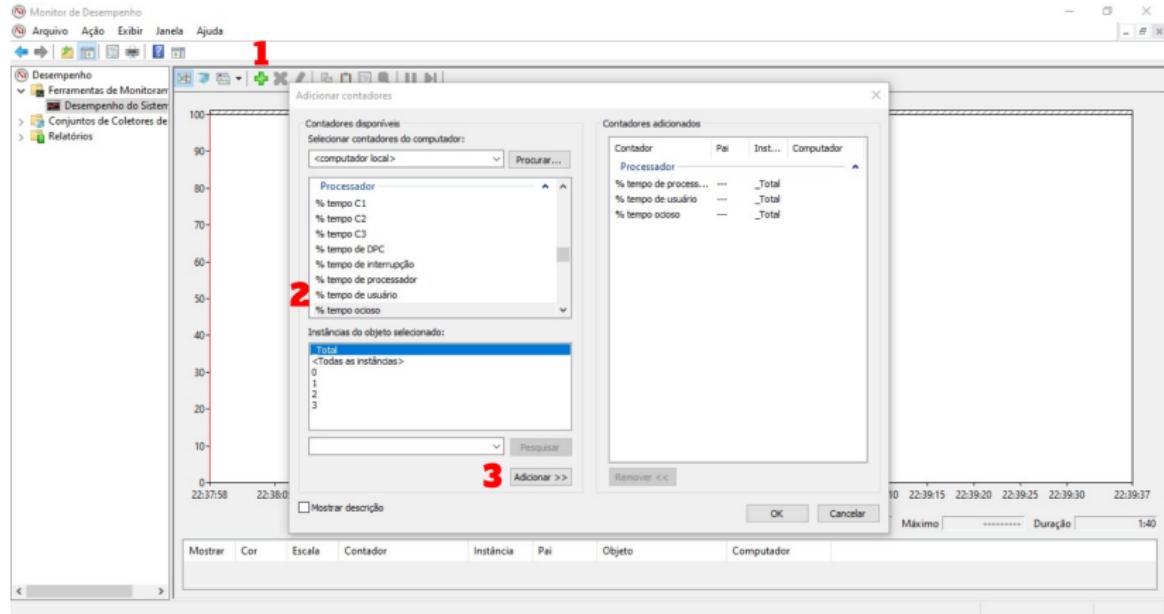


Figure: Monitoramento em tempo real

Monitorando em tempo real

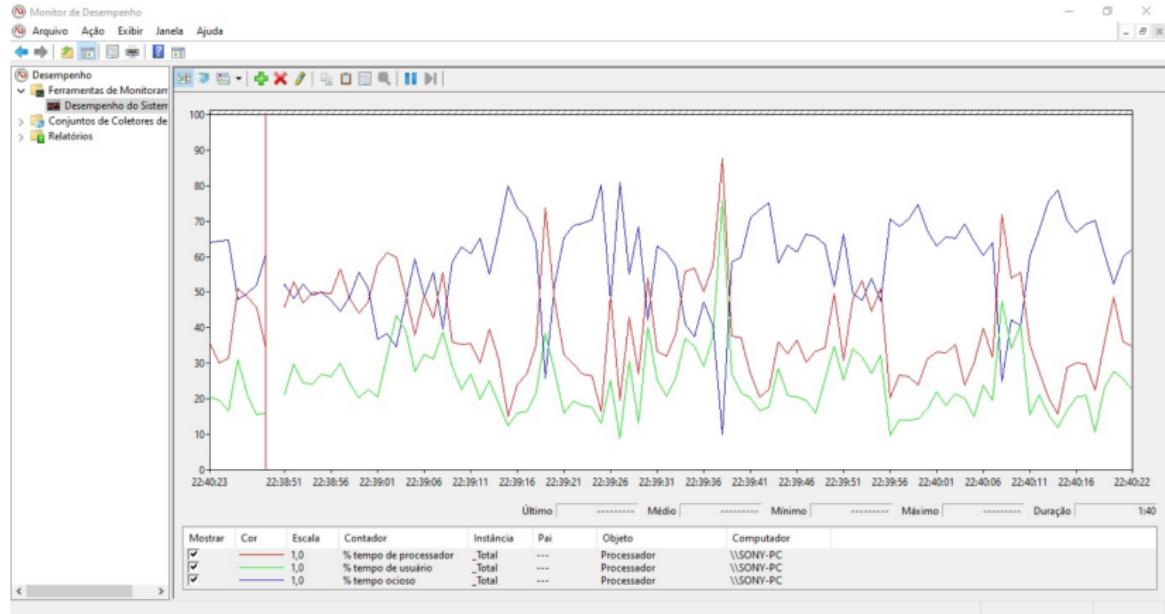


Figure: Monitoramento em tempo real

Criando conjunto de coletor de dados

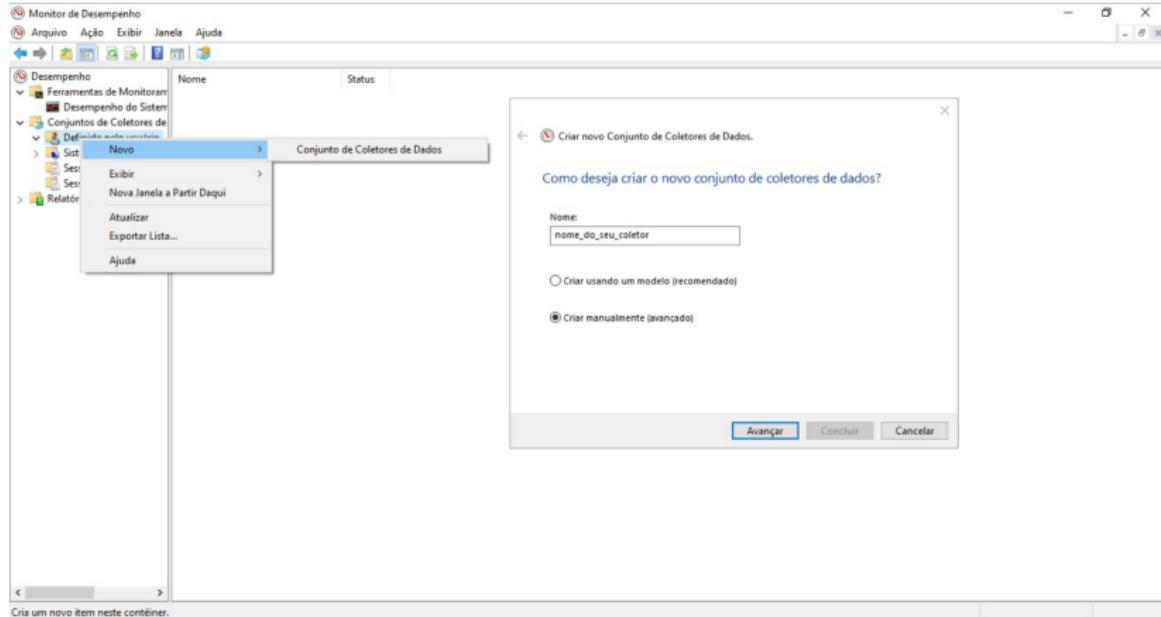


Figure: Criando conjunto de coletor de dados

Criando conjunto de coletor de dados

X

← Criar novo Conjunto de Coletores de Dados.

Que tipo de dados deseja incluir?

Criar logs de dados

Contador de desempenho

Dados do rastreamento de eventos

Informações de configuração do sistema

Alerta do Contador de Desempenho

Avançar Concluir Cancelar

Figure: Criando conjunto de coletor de dados

Criando conjunto de coletor de dados

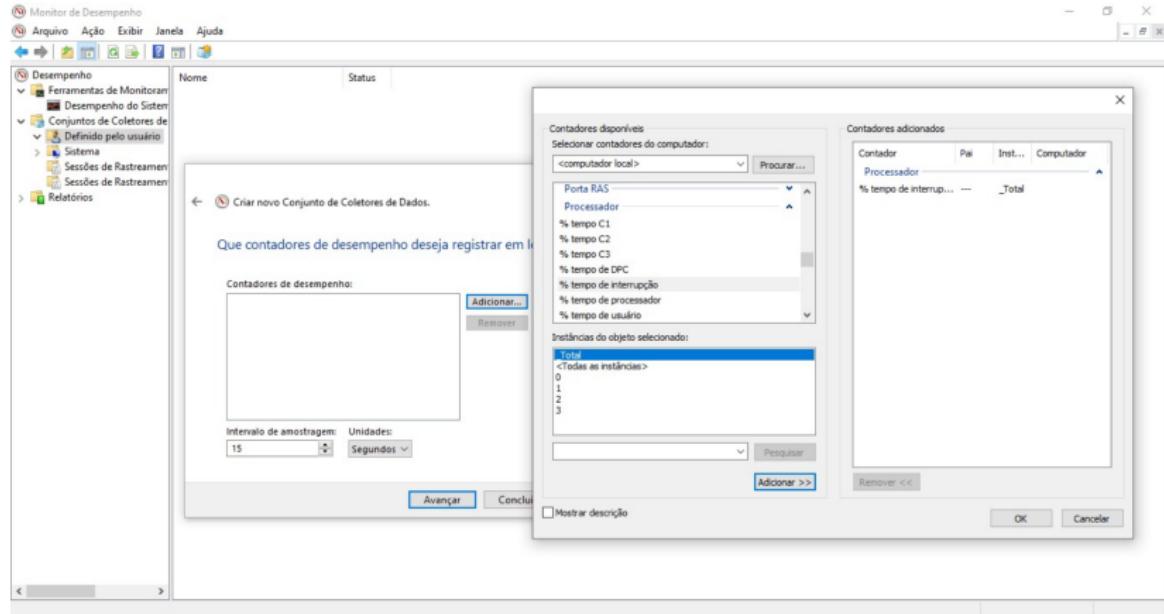


Figure: Criando conjunto de coletor de dados

Visualizando o arquivo de log

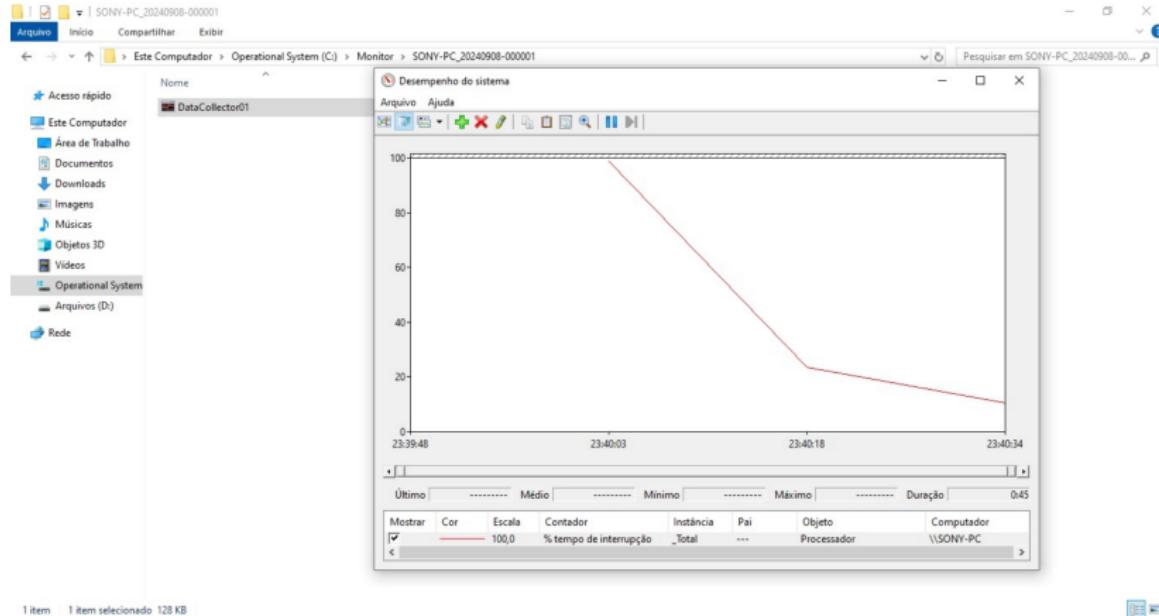


Figure: Arquivo de log

Propriedade do arquivo e adicionando contadores

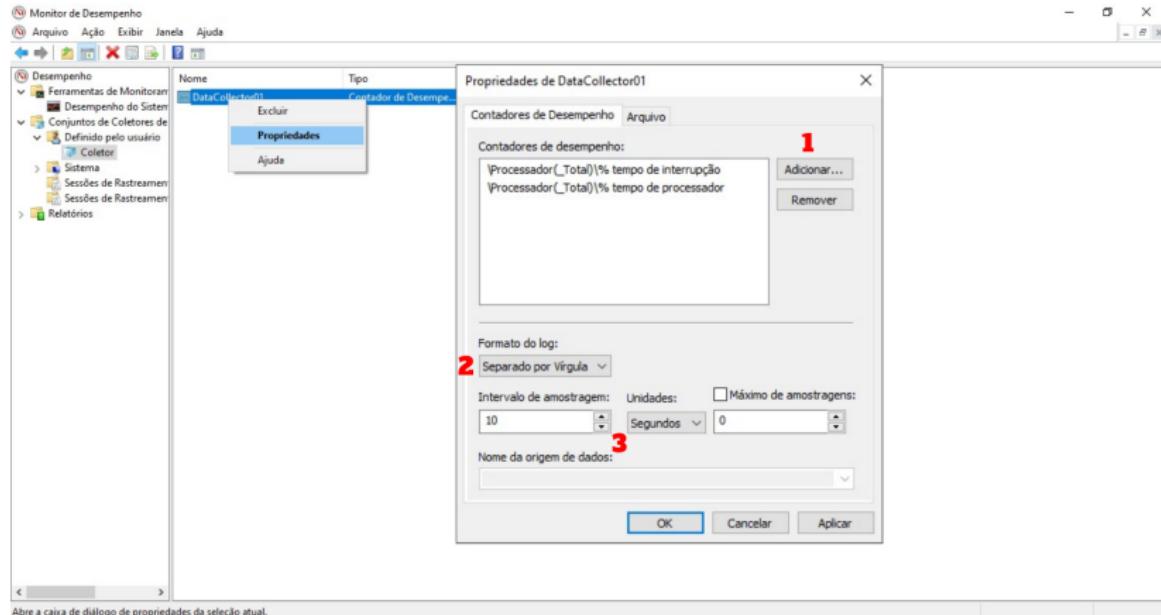


Figure: Propriedade do arquivo e adicionando contadores

- **1º ponto**
 - O Monitor de Performance é uma poderosa ferramenta de monitoramento e traz como vantagem o fato de já vir incluída no Windows por padrão.
- **2º ponto**
 - Tem como principal função ajudar administradores a descobrir gargalos em servidores Windows.
- **3º ponto**
 - É necessário cuidado no dimensionamento da janela de monitoramento e intervalo de amostragem.

Exemplo prático

Imagine que você precisa monitorar quais programas estão utilizando muita memória no seu computador. Com o comando Get-Process, você pode ver a lista de processos em execução e criar um script que te avisa quando o uso de memória ultrapassar um limite definido.

O que é?

- O PowerShell é uma interface de linha de comando poderosa, que permite controlar e automatizar tarefas no sistema operacional Windows, seja via interface gráfica (GUI) ou texto (CLI).

Para que serve?

- O PowerShell é usado para criar scripts que automatizam tarefas, como gerenciamento de usuários, configuração de sistemas, ou execução de comandos para obter informações detalhadas de processos e serviços rodando no Windows.

Por que usar PowerShell?

- Com ele, é possível automatizar tarefas repetitivas, como copiar arquivos, monitorar serviços ou até mesmo criar backups de forma automatizada.

Interface do PowerShell



Figure: Prompt comando do PowerShell

Sobre o PowerShell

Calculadora

- 5-4
- $(5 + 9) * 4$
- 5GB / 120MB
- O PowerShell suporta valores de armazenamento computacional como:
 - Kilobytes (KB)
 - Megabytes (MB)
 - Gigabytes (GB)
 - Terabytes (TB)
 - Petabytes (PB)

Comandos externos

O PowerShell pode executar comandos do prompt de comando Microsoft

- ipconfig
- cls
- ls
- clear
- ping
- dir

Comandos do PowerShell

Os comandos do PowerShell são cmdlets. Os nomes dos comandos são compostos por um verbo seguido de um hífen (-) e uma ação.

- Digue no terminal:
 - **Get-Command**: Lista todos os comandos disponíveis no PowerShell, incluindo cmdlets, funções, aliases e scripts. Útil para descobrir quais comandos estão disponíveis.
 - **Get-Help**: Exibe informações de ajuda sobre cmdlets e comandos do PowerShell. Você pode usar esse comando para aprender como usar um comando específico ou entender sua sintaxe.
 - **Get-Location**: Mostra o diretório atual em que você está no PowerShell. É útil para navegar pelo sistema de arquivos e verificar onde você está trabalhando.
 - **Get-History**: Mostra o histórico dos comandos que você já executou na sessão atual do PowerShell, facilitando o reuso de comandos anteriores.

Command-lets(CMDLETS)	Descrição
Add	Adiciona um recurso ou anexa um item em outro item. Exemplo: Add-Computer Assim como tem o Add existe o Remove
Clear	Remove um recurso. Exemplo: Clear-Content
Close	Altera o estado de um recurso. Assim como existe Close existe o Open
Format	Formata (arruma) objetos ou saídas em determinados layouts.
Get	Ação que recupera informações, por exemplo, uma lista de objetos. Exemplo: Get-Command
Move	Move recursos de uma localização para outra.
Show	Exibe informações relacionadas ao “substantivo”

Figure: CMDLETS do PowerShell

PowerShell ISE

PowerShell ISE (Integrated Scripting Environment), um ambiente de programação do PowerShell que facilita o desenvolvimento de scripts, você pode executar comandos, gravar, testar e depurar scripts em uma interface de usuário gráfica baseada no Windows.

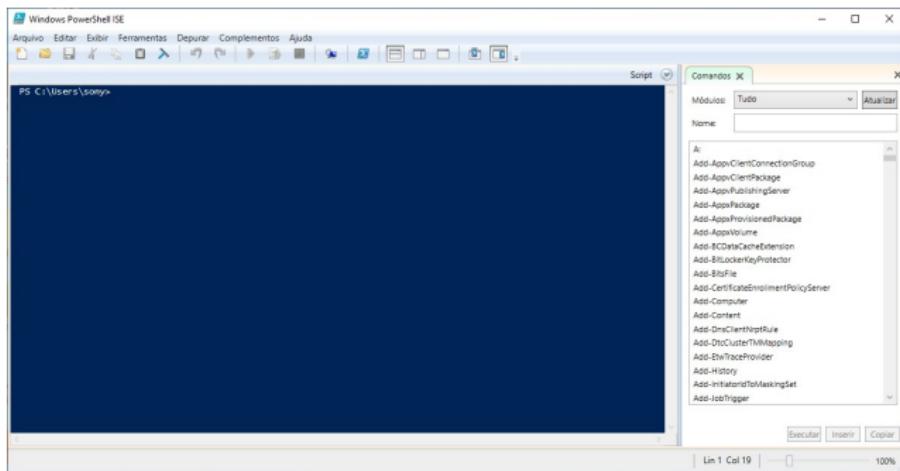


Figure: PowerShell ISE

Um fator muito importante no uso de um programa ou linguagem de programação é ter uma base de conhecimento completa e atualizada. O PowerShell tem um help atualizável e fácil de usar. Para atualizar o Help do PowerShell usamos o comando `update-help`

- **Get-Help <cmdlet>** - Exibe o help no console
- **Get-Help <cmdlet> -Online** - Exibe o help online, digamos que eu queira saber mais sobre “Compare-Object” .
- **Get-Help Compare-Object -Online** – Acessa os recursos online
- **Get-Help Compare-Object -Examples** – Exibe exemplos do comando
- **Get-Help Compare-Object -Detailed** – Exibe um help detalhado
- **Get-Help Compare-Object -ShowWindow** - Exibe uma janela

Exibição

As informações que você pode coletar através do PowerShell pode ser formatada de modo que facilite a visualização das informações. Um dos cmdlets que nós administradores sempre precisamos executar é o **Get-Process**, com ele, listamos os processos em execução em nosso servidor ou estação.

- **Get-Process**

Conhecendo o Get-Process

Usado o pipe (—) podemos passar a saída do comando para diversas opções. O pipe é um operador. Cada comando após o pipe recebe um objeto do comando anterior, realiza alguma operação no objeto, e depois passa adiante para o próximo comando no pipeline.

- Get-Process — more
- Get-Process — Format-List
- Get-Process — Format-List — more
- Get-Process — ConvertTo-HTML — Out-File "Processos.html"
- Get-Process — Export-CSV "Processos.csv"

cmdlets out

Alguns cmdlets existentes criam saídas legais, são os casos do cmdlets out.
Para listar os cmdlets “out”: Get-command out

- Out-Default - Envie a saída para o formatador padrão e o cmdlet de saída padrão.
- Out-File - Envia a saída para um arquivo.
- Out-GridView - Envia a saída para uma tabela interativa em uma janela separada.
 - Get-Process — Out-GridView
 - Get-Process — Out-File -FilePath C:\Monitor\processos.txt

Filtrar resultados

O cmdlet Where-Object fornece a capacidade de criarmos filtros específicos no retorno de outros cmdlets. Como você já deve ter percebido, alguns cmdlets exibem na tela todos os dados de determinado objeto ou recurso.

Por exemplo o cmdlet Get-Service trará na tela todos os serviços estando iniciados e parados. Com o Where-Object você pode criar um filtro e trazer apenas os serviços em execução.

- `get-service — where-object $_.Status -eq "Running"`

Filtrar resultados

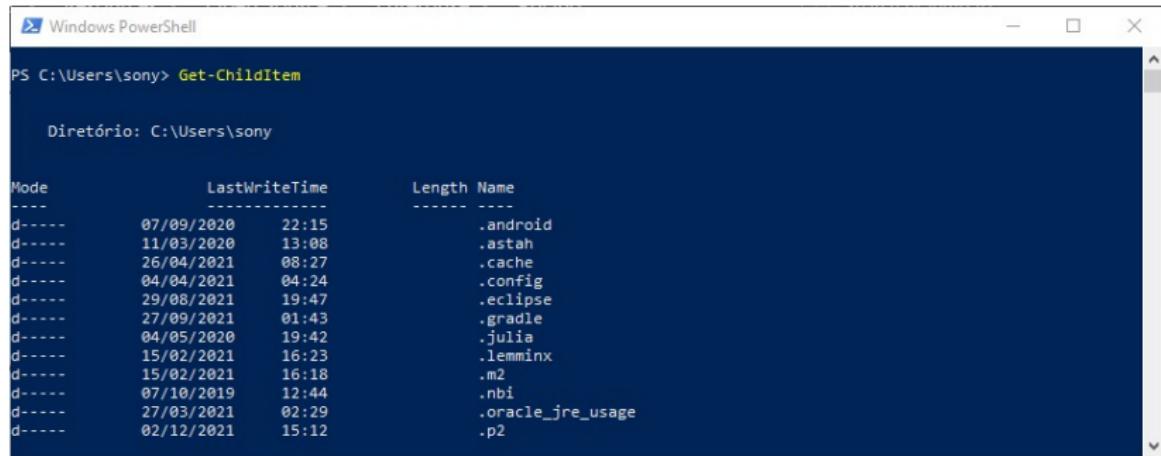
Operador	Descrição
-lt	Menor que
-le	Menor ou igual
-gt	Maior que
-ge	Maior ou igual
-eq	Igual
-ne	Não igual
-like	Usa wildcards para comparar padrões

Figure: Operadores para filtrar resultados

Filtrar resultados

Cada cmdlet exibe na tela diferentes resultados, portanto no momento de usar **Where-Object** você deve conhecer o resultado padrão e analisar quais são os nomes dos campos que deseja utilizar como campo. No exemplo abaixo foi executado o cmdlet **Get-ChildItem** e podemos notar que existem 15 campos.

Filtrando resultados



```
Windows PowerShell

PS C:\Users\sony> Get-ChildItem

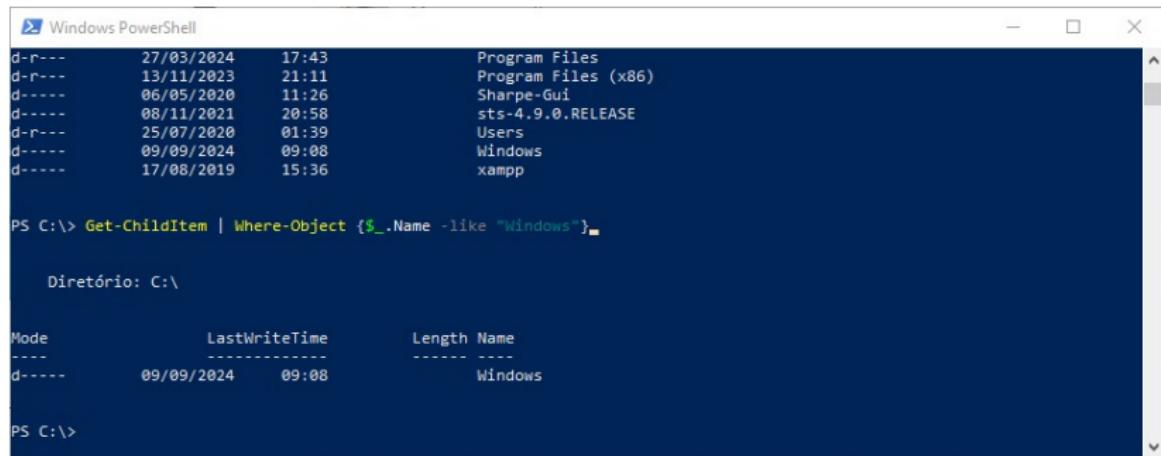
Diretório: C:\Users\sony

Mode          LastWriteTime      Length Name
----          -----      ----- 
d---- 07/09/2020 22:15           .android
d---- 11/03/2020 13:08           .astah
d---- 26/04/2021 08:27           .cache
d---- 04/04/2021 04:24           .config
d---- 29/08/2021 19:47           .eclipse
d---- 27/09/2021 01:43           .gradle
d---- 04/05/2020 19:42           .julia
d---- 15/02/2021 16:23           .lemminx
d---- 15/02/2021 16:18           .m2
d---- 07/10/2019 12:44           .nbi
d---- 27/03/2021 02:29           .oracle_jre_usage
d---- 02/12/2021 15:12           .p2
```

Figure: Get-ChildItem

Filtrar resultados

Podemos então fazer um filtro com where-object `$_.Name -like "Windows"`



The screenshot shows a Windows PowerShell window. At the top, there's a list of files in the current directory:

Mode	LastWriteTime	Name
d-r---	27/03/2024 17:43	Program Files
d-r---	13/11/2023 21:11	Program Files (x86)
d----	06/05/2020 11:26	Sharpe-Gui
d----	08/11/2021 20:58	sts-4.9.0.RELEASE
d-r---	25/07/2020 01:39	Users
d----	09/09/2024 09:08	Windows
d----	17/08/2019 15:36	xampp

Below this, a command is shown:

```
PS C:\> Get-ChildItem | Where-Object {$_.Name -like "Windows"}
```

Then, the output of the filtered command is displayed:

Mode	LastWriteTime	Name
d----	09/09/2024 09:08	Windows

```
PS C:\>
```

Figure: Where-Object filter

Permissões cmdlet

O cmdlet `Set-ExecutionPolicy` permite determinar como os scripts serão permitidos para execução. Windows PowerShell tem quatro diferentes políticas de execução:

- **Restricted** – Nenhum script pode ser executado. Windows PowerShell pode ser usado apenas no modo interativo.
- **AllSigned** - Somente scripts assinados por um fornecedor confiável pode ser executado.
- **RemoteSigned** - os scripts baixados devem ser assinados por um fornecedor confiável antes que eles possam ser executados.
- **Unrestricted** - Sem restrições, todos os scripts do Windows PowerShell pode ser executado.

Permissões cmdlet

Alguns ambientes podem não permitir a execução de scripts por motivos de segurança. Para habilitar a execução de scripts você deve definir uma política de execução com o comando:

- Dar permissão
 - Set-ExecutionPolicy RemoteSigned
 - Set-ExecutionPolicy Unrestricted -Force

Criando nosso primeiro script!

- É possível criar scripts PowerShell e sempre que necessário executa-lo como se fosse o velho batizinho (Batch Files).
- A extensão para execução de scripts no PowerShell é **.PS1**. Usando o editor de textos basta criar um arquivo e salvar como `nome_desejado.ps1`
- A vantagem de fazer uso de scripts PowerShell é criar ferramentas poderosas de administração ou de automação de tarefas cotidianas.

Criando um script

Criar um arquivo texto simples na raiz com o nome qualquer (ex: test.ps1) e edite o script abaixo:

```
1 # Comentário no PowerShell  
2 "Massa D+"
```

Executar: .\test.ps1

Get-Process

```
PS C:\Users\sony\Downloads> Get-Process
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
157	9	2892	7980		5948	0	AdminService
1093	44	9784	23128		5808	0	afwServ
135	7	2980	8692		8260	0	AggregatorHost
324	19	8444	25684	0,17	11876	1	ApplicationFrameHost
133	8	1604	6196		5988	0	armsvc
611	33	103852	149972		8168	0	aswEngSrv
988	29	35956	54928		8592	0	aswidagent
1464	35	37220	46980		4028	0	aswToolsSvc
246	14	11872	19228	2,05	14224	0	audiodg
5435	124	125644	174248		3960	0	AvastSvc
752	32	21720	37400	0,41	4792	1	AvastUI
2458	56	44916	55488	35,31	5472	1	AvastUI
631	31	16300	36536	0,34	7028	1	AvastUI
568	28	14948	30824	0,19	9656	1	AvastUI
166	12	2436	9040	0,09	11832	1	BleServicesCtrl
347	26	58416	108416	3,52	4860	1	chrome
376	24	20564	44164	20,20	4876	1	chrome

Figure: Get-Process

Monitorando processos

Get-Process

- **Handles** - O número de manipulações abertas pelo processo.
- **NPM(K)** - A quantidade de memória não paginada usada pelo processo, em kilobytes.
- **PM(K)** - A quantidade de memória paginada usada pelo processo, em kilobytes.
- **WS(K)** - O tamanho do conjunto de trabalho do processo, em kilobytes. O conjunto de trabalho consiste nas páginas de memória recentemente referenciadas pelo processo.
- **VM(M)** - A quantidade de memória virtual usada pelo processo, em megabytes. A memória virtual inclui o armazenamento em disco dos arquivos de paginação.
- **CPU(s)** - O tempo do processador que o processo usou em todos os processadores, em segundos.
- **ID** - O ID de processo (PID) do processo.
- **ProcessName** - O nome do processo.

Tipos de dados e definição de variável

- \$num1=10
- \$num2="20"
- \$num1+\$num2
- \$num2+\$num1
- [int]\$num2+\$num1
- \$num1.GetType().Name
- \$processos = get-process

Manipulação de arquivos

- **Criar**

- New-Item -Path 'C:\temp\New Folder' -ItemType "directory"

- **Editar**

- Add-Content .\arquivo.txt "texto q"
 - Add-Content -Path "c:\sample.txt" -Value "r\nThis is the last line"

- **Recuperar**

- Get-Content .\arquivo.txt

Recuperação de informações

- Get-Process — where `{$__.ProcessName - like "chrome"}`
- Get-Process — where `{($__.ProcessName - like "chrome") -AND
($_.WS -gt 300000000)}`
- Get-Process — Get-Member

Recuperação de informações

```
1 for ($i=0; $i -le 10; $i++) {  
2     Get-Process | where {($_.ProcessName -like "chrome") -and  
3         ($_.WS -gt 300000000)}  
4     Start-Sleep 3  
5 }
```

Estudo de caso

Imagine que você precisa monitorar o desempenho de processos críticos, como navegadores ou serviços do sistema, em uma estação de trabalho ou servidor que está apresentando lentidão. Coletar dados como uso de CPU e memória pode ajudar a identificar gargalos e otimizar o sistema. O script a seguir realiza essa tarefa, salvando informações importantes em um arquivo de log para análise detalhada posterior.

Monitorar uso de CPU e memória

Um exemplo de script em PowerShell que demonstra o poder da linguagem ao automatizar uma tarefa simples e útil: monitorar o uso de CPU e memória dos processos em execução e salvar essas informações em um arquivo para análise.

```
1 $arquivoLog = "C:\Monitoramento\Relatorio_Desempenho.txt"
2 # Falta código
3 for ($i=0; $i -le 10; $i++) {
4 # Falta código
5     Add-Content $arquivoLog "Memória Usada: $memoriaMB
6     MB"
7 }
```

Clique aqui para acessar o repositório e executar o arquivo **processos.ps1**.

Tarefa para casa

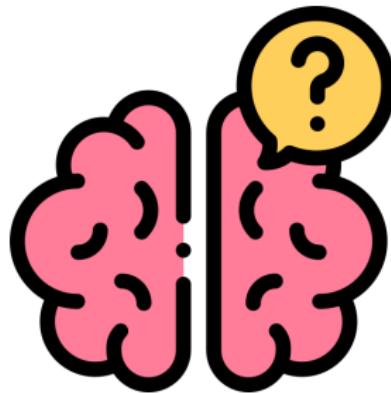
Crie um script em PowerShell que monitore o uso de CPU e memória de seus jogos ou programas favoritos enquanto estão sendo executados, e salve os resultados em um arquivo de log.

Passos:

- ① Escolha seu jogo favorito ou um programa que você usa com frequência (exemplo: chrome, spotify, discord).
- ② Crie um script PowerShell que monitore o uso de CPU e memória do programa por 10 iterações, com intervalos de 5 segundos.
- ③ Salve o resultado em um arquivo de log na pasta C:\Monitoramento.

Objetivos:

- Compreender a hierarquia de memória e suas camadas (Cache, RAM, ROM).
- Distinguir entre memória principal e secundária.
- Explorar técnicas de gerenciamento de memória, como paginação, segmentação e memória virtual.



Uma dúvida!



- Vocês já se perguntaram por que seu computador parece mais rápido quando abre um programa pela segunda vez?
- O que acontece quando falta espaço de memória?

Hierarquia de memória

- **Cache**: Memória ultrarrápida, mas com capacidade limitada. Armazena dados que a CPU precisa acessar rapidamente.
- **RAM** (Memória principal): Memória de acesso aleatório, onde os dados e instruções são temporariamente armazenados durante a execução de programas.
- **ROM** (Memória permanente): Armazena dados que não podem ser alterados facilmente, como o firmware do sistema.

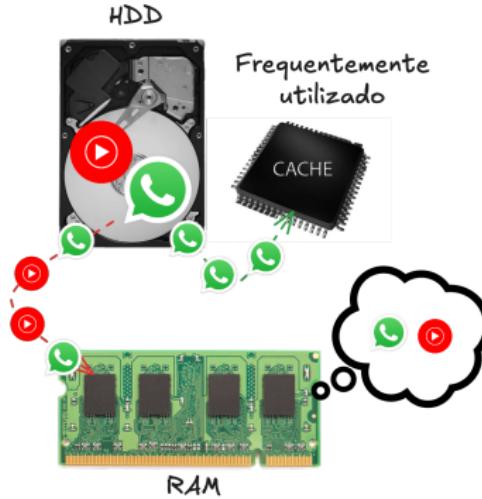
Hierarquia de memória



- **Cache** são as ferramentas mais importantes que ficam ao alcance imediato.
- **RAM** é uma "gaveta" na mesa.
- **ROM** é um cofre que você raramente abre, mas armazena documentos importantes e imutáveis.

- **Memória principal (RAM):**
 - A RAM armazena dados temporariamente, e os dados são perdidos quando o sistema é desligado.
- **Memória secundária (HDD/SSD):**
 - Armazena dados permanentemente, como discos rígidos (HDD) ou discos de estado sólido (SSD), mas é mais lenta que a RAM.
- **Diferenças e comparações:**
 - Memória principal é volátil (dados são perdidos quando o sistema é desligado).
 - Memória secundária é não volátil (os dados permanecem mesmo quando o sistema está desligado).

Memória principal e secundária



- Os dados são carregados do HDD/SSD para a RAM.
- Se frequentemente usados, são armazenados na Cache para acesso rápido.
- Ao desligar o sistema, os dados desaparecem da RAM, mas permanecem no HDD/SSD.

Uma dúvida!

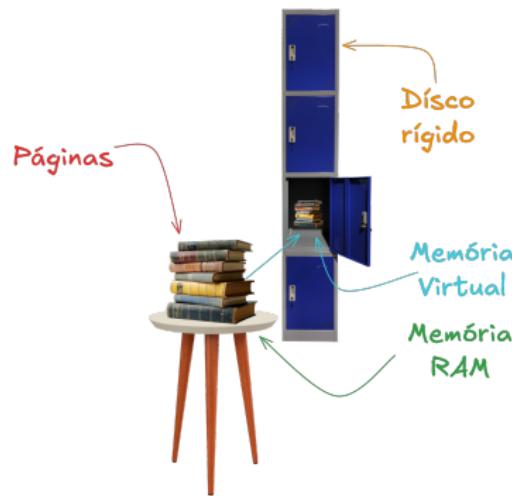


- Por que precisamos da memória secundária se já temos a RAM?
- O que acontece com o computador quando falta espaço de RAM e temos que usar a memória secundária?

Técnicas de gerenciamento de memória

- **Paginação:**

- Divisão da memória em páginas e o uso de páginas virtuais para otimizar o uso da RAM.
 - A RAM é uma mesa pequena e o disco rígido, um armário. Quando a mesa enche, você guarda um livro no armário (memória virtual) para abrir espaço para outro. Esse processo de troca é a paginação.



Técnicas de gerenciamento de memória

- Segmentação:

- Organização da memória em segmentos lógicos, como dados, código e pilha.
 - Pense na segmentação como organizar uma mala. Em vez de misturar tudo, você separa os itens em compartimentos: roupas, sapatos e produtos de higiene, facilitando o acesso a cada "segmento" de forma organizada.



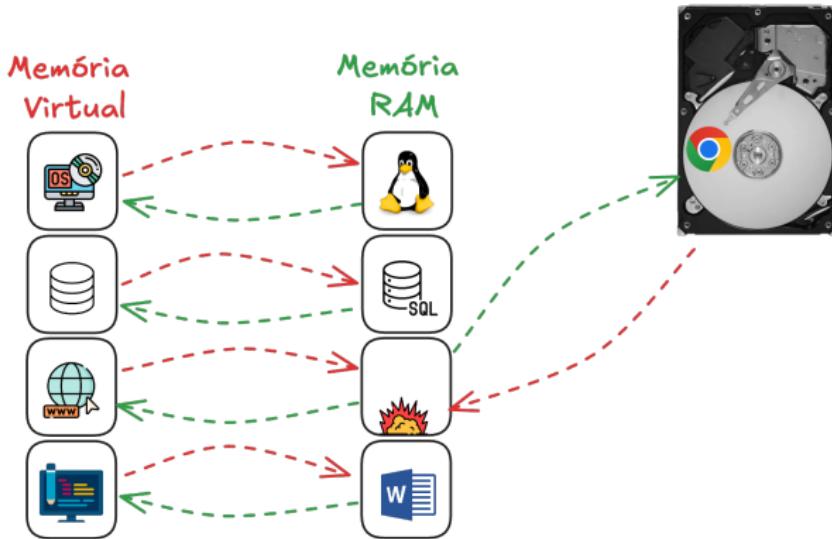
Técnicas de gerenciamento de memória

- Memória virtual:

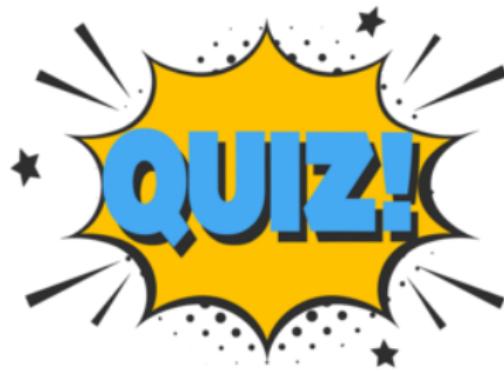
- Uso do disco rígido para expandir a memória disponível, simulando mais RAM.
 - A RAM é como sua mesa de trabalho, onde você mantém os itens mais importantes. Quando a mesa enche, você move coisas menos urgentes para a gaveta (memória virtual) para liberar espaço. Isso mantém seu trabalho fluindo, mas buscar algo na gaveta demora mais, pois ela (disco rígido) é mais lenta que a mesa (RAM).



Técnicas de gerenciamento de memória



Vamos fazer uma simulação, para isso visite a URL.



Clique aqui

Resumo da aula

- Relembre os principais tópicos abordados:
 - Hierarquia de Memória.
 - Diferenças entre memória principal e secundária.
 - Técnicas de gerenciamento de memória.

Sistema de entrada e saída

Objetivos:

- Compreender a função dos dispositivos de entrada e saída em um sistema de computação.
- Explorar os barramentos de entrada e saída e sua importância na comunicação entre dispositivos e o processador.
- Identificar o papel dos controladores e interfaces no gerenciamento de dispositivos de E/S.



- **Entrada:** Dispositivos que enviam dados para o sistema (ex.: teclado, mouse, scanner).
- **Saída:** Dispositivos que recebem dados do sistema (ex.: monitor, impressora, alto-falante).
- **Dispositivos mistos:** Dispositivos que podem funcionar como entrada e saída, como as telas touchscreen.

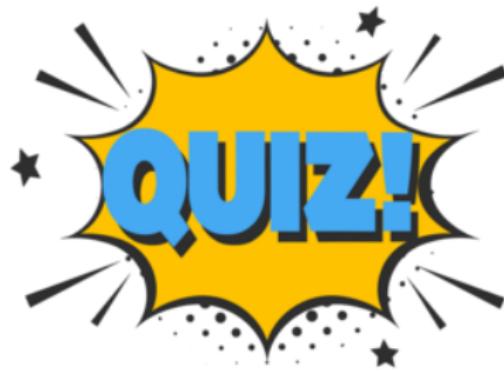
Barramentos de Entrada e Saída

- Barramentos de E/S são vias de comunicação que conectam dispositivos de entrada e saída ao processador.
- Os principais tipos de barramentos incluem PCIe, USB, SATA, etc.
- **Exemplo** - Clique aqui

Controladores e interfaces

- Os controladores são responsáveis por gerenciar a comunicação entre dispositivos de E/S e o processador. Eles garantem que os dados sejam transferidos corretamente.
 - As interfaces (ex.: USB, HDMI, VGA) são padrões que definem como os dados devem ser transmitidos entre o dispositivo e o sistema.





Clique aqui

Resumo da aula

- Relembrando principais conceitos abordados:
 - Dispositivos de Entrada e Saída.
 - Barramentos de E/S.
 - Controladores e Interfaces.

Objetivos:

- Compreender os conceitos de paralelismo e sua aplicação em sistemas computacionais.
- Diferenciar arquiteturas SIMD (Single Instruction, Multiple Data) e MIMD (Multiple Instruction, Multiple Data).
- Explorar os fundamentos do processamento distribuído e seus benefícios.

- **O que é paralelismo?**

- Paralelismo é a técnica de dividir uma tarefa em várias subtarefas que podem ser executadas simultaneamente, permitindo um ganho de desempenho significativo.

- **Por que usar arquiteturas paralelas?**

- Processamento de imagens e vídeos
- Simulações científicas (ex.: clima, dinâmica de fluidos)
- Servidores web que atendem a várias requisições de forma simultânea.

- 1 Quantos de vocês já perceberam que seus computadores têm múltiplos núcleos? O que isso significa para o desempenho?
- 2 Em que situações vocês acham que o paralelismo é mais importante?

Conceito de paralelismo - Atividade

Objetivo:

- Demonstrar a diferença entre execução sequencial e paralela;
- Grupo 1 (Execução Sequencial);
- Grupo 2 (Execução Paralela) e
- Qual o tempo de execução de ambos grupos?

- **O que é SIMD (Single Instruction, Multiple Data)?**
 - Uma única instrução é aplicada a múltiplos dados ao mesmo tempo.
 - Ideal para operações em paralelo que envolvem grandes volumes de dados similares.

Prós e Contra



- **O que é MIMD (Multiple Instruction, Multiple Data)?**
 - Diferentes instruções são aplicadas a diferentes conjuntos de dados simultaneamente.
 - Permite maior flexibilidade e complexidade nas operações.

Arquiteturas MIMD

Prós e Conta



Comparação entre SIMD e MIMD

Quando usar cada um?

- **SIMD:**

- Vantagens: Alta eficiência em tarefas repetitivas, como gráficos e processamento de imagens.
- Desvantagens: Menos flexível para tarefas variadas.

- **MIMD:**

- Vantagens: Flexibilidade para lidar com diversas tarefas simultaneamente.
- Desvantagens: Pode ser mais complexo de programar e coordenar.

Simulação de SIMD e MIMD

Código em Python para simular SIMD e MIMD:

```
1 import concurrent.futures  
2 import time  
3  
4 def processar(valor):  
5     return valor * 2  
6     ...
```

- **O que é processamento distribuído?**

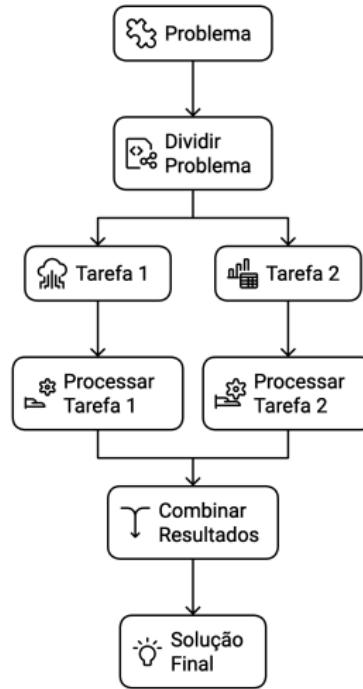
- Consiste em dividir um problema entre várias máquinas, cada uma processando uma parte do problema.
- Usado em sistemas de larga escala como Google, Amazon e em simulações científicas.

- **Benefícios e desafios:**

- Vantagens: Escalabilidade, alta disponibilidade, e processamento mais rápido.
- Desafios: Complexidade de implementação, sincronização de dados, e falhas de comunicação.

Processamento distribuído

Como funciona o processamento distribuído?



Processamento distribuído

Aplicações do Processamento Distribuído



Simulação de processamento

Simulação de processamento distribuído com Python

```
1 from multiprocessing.pool import Pool  
2 import time  
3  
4 def processar_dados(dado):  
5     time.sleep(1)  
6     return dado * 2  
7     ...
```

Resumo da aula

- Revisão dos principais conceitos:
 - Diferença entre paralelismo e processamento distribuído.
 - Arquiteturas SIMD e MIMD.
 - Benefícios e desafios de sistemas distribuídos.

Objetivos:

- Compreender os métodos de medição e avaliação de desempenho de sistemas computacionais.
- Explorar o uso de benchmarks para avaliação de hardware e software.
- Identificar e aplicar técnicas de otimização para melhorar o desempenho de sistemas.

- **Tempo de resposta:**

Atrasos nas ações do usuário e frustra a experiência geral.

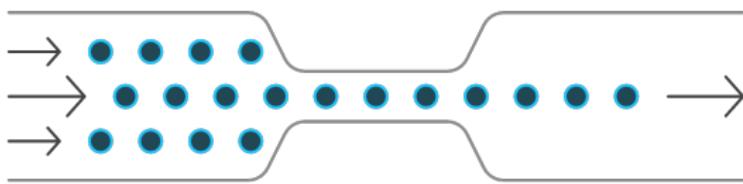


Figure: Tempo de resposta

Introdução ao desempenho de sistemas

- Exemplo:



Figure: Exemplo de tempo de resposta

Introdução ao desempenho de sistemas

- **Taxa de transferência de dados:** A quantidade de dados processados pelo sistema em um período de tempo.

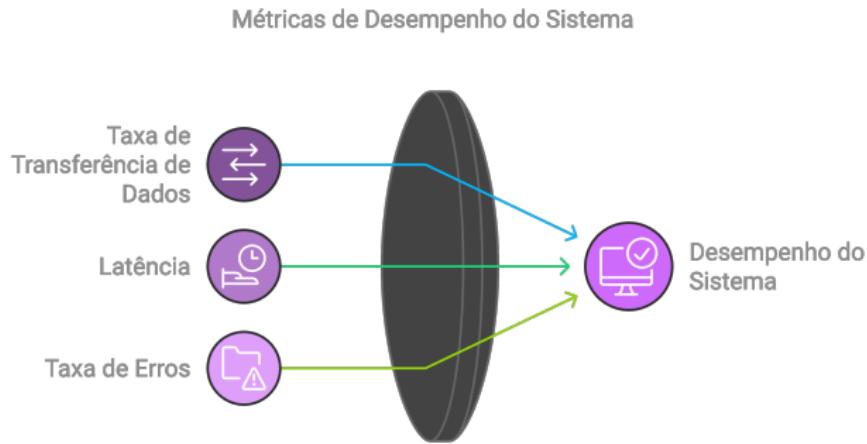
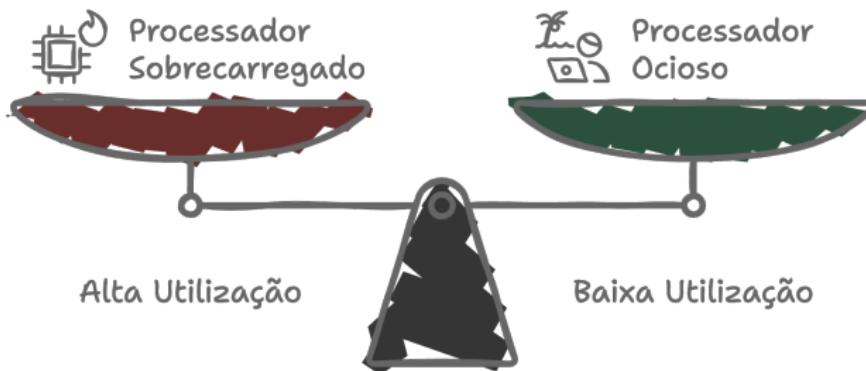


Figure: Throughput

Introdução ao desempenho de sistemas

- Utilização de CPU:



Equilibrando a utilização da CPU para desempenho ideal.

Figure: Utilização de CPU

Introdução ao desempenho de sistemas

- **Exemplo:**

Analisando Problemas de Utilização da CPU

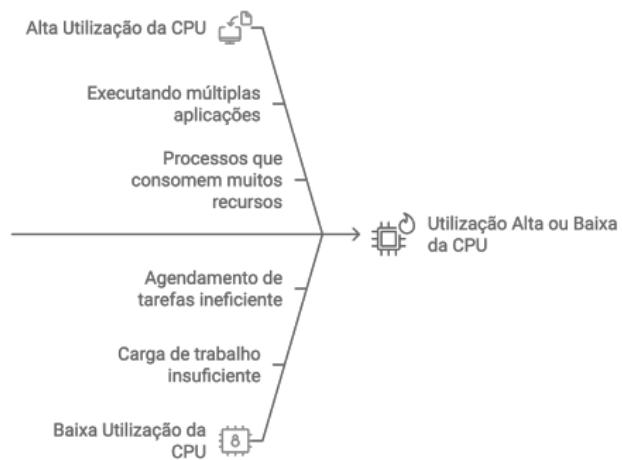


Figure: Utilização de CPU

Por que entender gargalos de desempenho?

- **Gargalos de hardware:**

- Incluem limitações de CPU, memória insuficiente ou lenta, e problemas em dispositivos de entrada e saída (como um disco rígido lento).

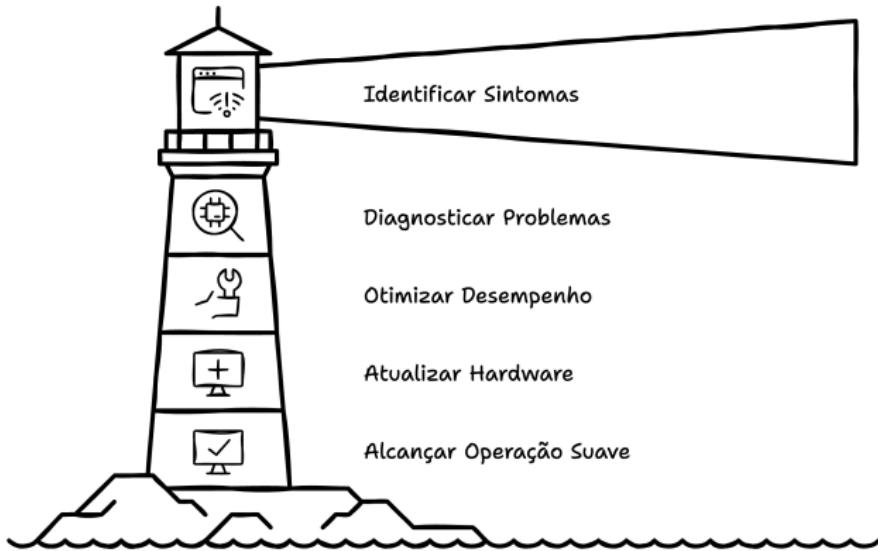
- **Gargalos de software:**

- Podem incluir código mal otimizado, processos que consomem muitos recursos ou má gestão de memória.

Identificar e eliminar os gargalos é essencial para melhorar a performance do sistema e proporcionar uma experiência mais satisfatória ao usuário.

Resolver lentidão no sistema

Passos para Resolver a Lentidão do Computador



Problema

Imagine que você acessa um site para verificar as notas de uma prova, mas ele demora 15 segundos para carregar cada página. Isso causa frustração e pode levar os usuários a desistirem de usar o serviço.

Otimização do problema

Técnicas para Melhorar o Desempenho do Site

Minificação de Arquivos

Diminuindo o tamanho dos arquivos para downloads mais rápidos



Otimização de Imagens

Reducir o tamanho dos arquivos de imagem para carregamento mais rápido

Uso de CDN

Distribuindo conteúdo entre servidores para reduzir a latência

Medição e avaliação de desempenho

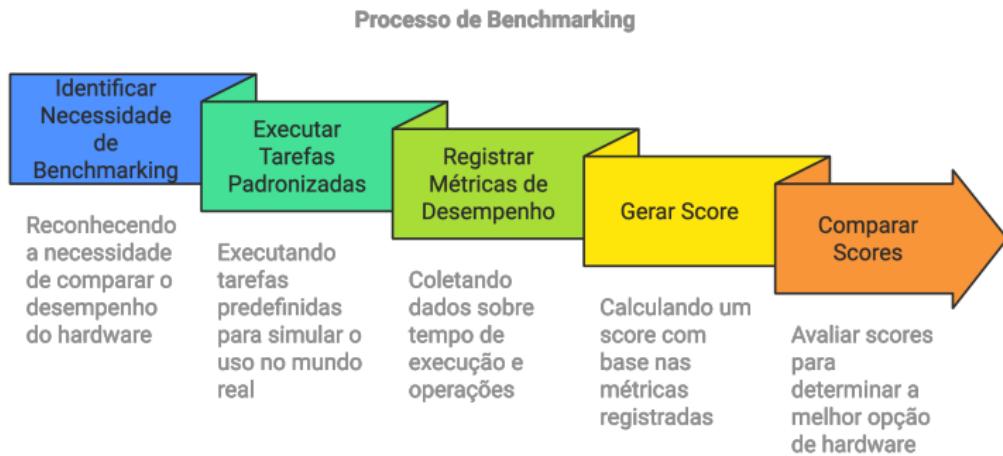
Vamos criar um script que mede o tempo de execução de um processo para que vocês compreendam a importância de otimizar algoritmos.

```
1 import time
2
3 def processo_pesado():
4     soma = 0
5     for i in range(1, 1000000):
6         soma += i
7     return soma
8
9 start_time = time.time()
10 resultado = processo_pesado()
11 end_time = time.time()
12
13 print(f"Resultado: {resultado}")
14 print(f"Tempo de execução: {end_time - start_time:.4f} segundos")
```

Benchmarks

- O que são Benchmarks?

- Benchmarks são testes padronizados que medem a performance de sistemas e componentes de hardware, como CPUs, memória, armazenamento e GPUs.



Pergunta

- Alguém já comprou um celular ou computador novo e viu as avaliações de desempenho antes? Isso é resultado de benchmarks!



Tipos de Benchmarks

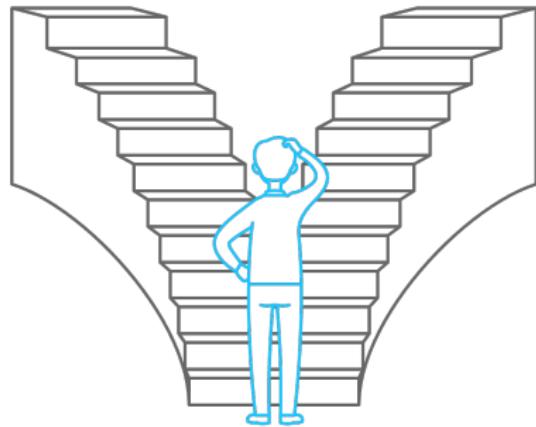
Qual tipo de benchmark utilizar?

Sintético

Simula cenários controlados para avaliar desempenho específico.

Real

Avalia desempenho em condições reais de uso.



Exemplos de Benchmarks

- **SPEC:**

- Avalia o desempenho de CPUs em tarefas de cálculo intensivo.
- Utilizado por profissionais para comparar processadores de servidores e estações de trabalho.

- **Geekbench:**

- Mede o desempenho de CPU e GPU em dispositivos como PCs, Macs e smartphones.
- Gera um score que permite comparar a performance de dispositivos diferentes, levando em conta múltiplos núcleos.

- **LINPACK:**

- Testa a capacidade do sistema em resolver equações matemáticas complexas.
- Muito usado em supercomputadores para medir a capacidade de processamento em operações científicas.

Atividades:

- Pesquisa - Coletar informações a respeito do tema;
- Estrutura - Organizar o conteúdo em tópicos:
 - Introdução;
 - Desenvolvimento;
 - Conclusão.
- Apresentação - Se organizar para que todos membros participem.
- Material - artigo e pptx;
- Cronograma - 33 min para cada equipe;