



O Guia Definitivo para Sobreviver a Sistemas Operacionais

Este PDF é estruturado para proporcionar uma compreensão acessível e prática dos sistemas operacionais, abrangendo desde a teoria básica até exercícios práticos que reforçam o aprendizado.

Table of Contents

Sistemas Operacionais	2
Capítulo 1	3
1.1 O que os Sistemas Operacionais fazem	4
1.1.1 Hardware	6
1.1.2 Software	8
1.1.3 Visões do Sistema	10
1.2 Operação do Computador	13
1.3 Estrutura de Armazenamento	18
1.4 Estrutura de Entrada e Saída	25
1.5 Arquitetura do Sistema	31
1.6 Estrutura do sistema operacional	37
1.7 Operações do Sistema Operacional	38
1.8 Gerência de processos	40
1.9 Gerência de memória	42
1.10 Gerência de armazenamento	44
1.11 Proteção e segurança	51
1.12 Sistemas de uso específico	53
1.13 Ambientes de Computação	56
1.14 Sistemas Operacionais de Código Aberto	59
Exercícios Práticos Resolvidos	62
Capítulo 2	66
2.2 Interface usuário-sistema operacional	68
2.3 Chamadas de sistema	71
2.4 Tipos de chamadas de sistema	75
2.5 Programas do sistema	78
2.6 Projeto e implementação do sistema operacional	79
2.7 Máquinas virtuais	82
2.8 Geração do sistema operacional	85
2.9 Boot do sistema	88
Exercícios Práticos Resolvidos	91
Questões 1	96
Prática 1	102
Respostas - Questões 1	168
Bibliografia	171

Sistemas Operacionais

Bem-vindo ao nosso guia sobre Sistemas Operacionais! Nesta obra, exploraremos os conceitos fundamentais que regem o funcionamento dos sistemas que permitem que nossos dispositivos funcionem de maneira eficaz. Os sistemas operacionais são uma peça crucial da tecnologia moderna, servindo como intermediários entre o hardware e o software, gerenciando recursos, permitindo a execução de aplicativos e garantindo uma experiência de usuário fluida.

Este PDF é estruturado para proporcionar uma compreensão acessível e prática dos sistemas operacionais, abrangendo desde a teoria básica até exercícios práticos que reforçam o aprendizado.

Como utilizar este material

Para aproveitar ao máximo este guia, recomendamos a seguinte abordagem:

- 1. Estude os conceitos:** Leia atentamente cada seção, focando na compreensão dos conceitos fundamentais. Não se preocupe se não entender tudo de imediato; os sistemas operacionais são um tema complexo que se torna mais claro com o tempo e a prática.
- 2. Pratique regularmente:** Utilize os exercícios práticos fornecidos para reforçar seu aprendizado. A experiência prática é essencial para solidificar o conhecimento teórico.
- 3. Resolva as questões:** Tente responder às questões propostas ao final de cada seção. Isso ajudará a avaliar sua compreensão e identificar áreas que podem precisar de revisão.
- 4. Explore além do material:** Encorajamos você a pesquisar tópicos adicionais que despertem seu interesse. A área de sistemas operacionais é vasta e está em constante evolução.
- 5. Aplique o conhecimento:** Sempre que possível, relate o que você aprendeu com situações do dia a dia ou problemas reais de computação. Isso ajudará a contextualizar o conhecimento adquirido.

Capítulo 1

1.1 Introdução

Um **Sistema Operacional** é uma interface que interliga o **hardware** e o **software**, fazendo o gerenciamento dos dois mundos do computador.

- A inexistência, de um **SO** resulta em um impedimento de uma interação natural com o computador.

Como todo grande projeto complexo caímos em uma operação em que deve ser bem definida em como o **SO** irá trabalhar, quais são seus requisitos.

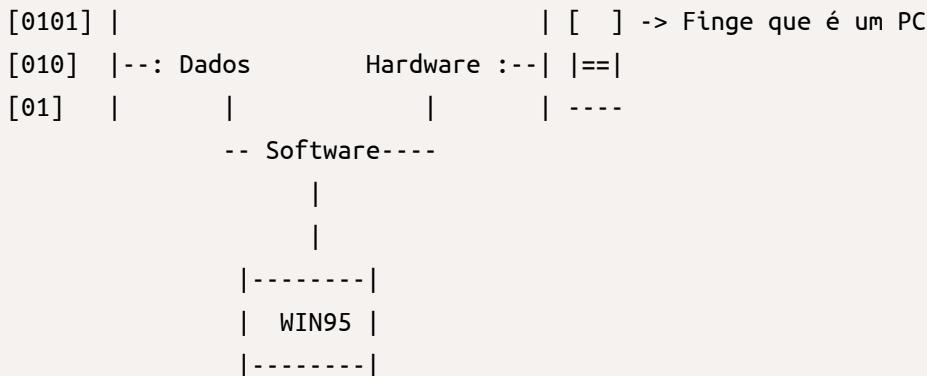
E o **SO** tem algumas funções básicas que é:

- dar partida no sistema
- gerenciar memoria
- gerenciar dispositivos E/S

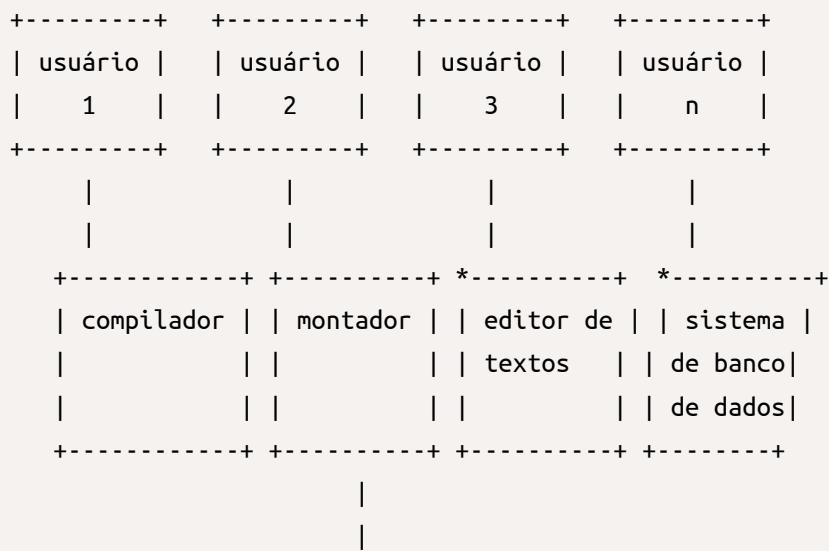
1.1 O que os Sistemas Operacionais fazem

Um **sistema computadorizado** ou só computador, pode ser *dividido em quatro partes*:

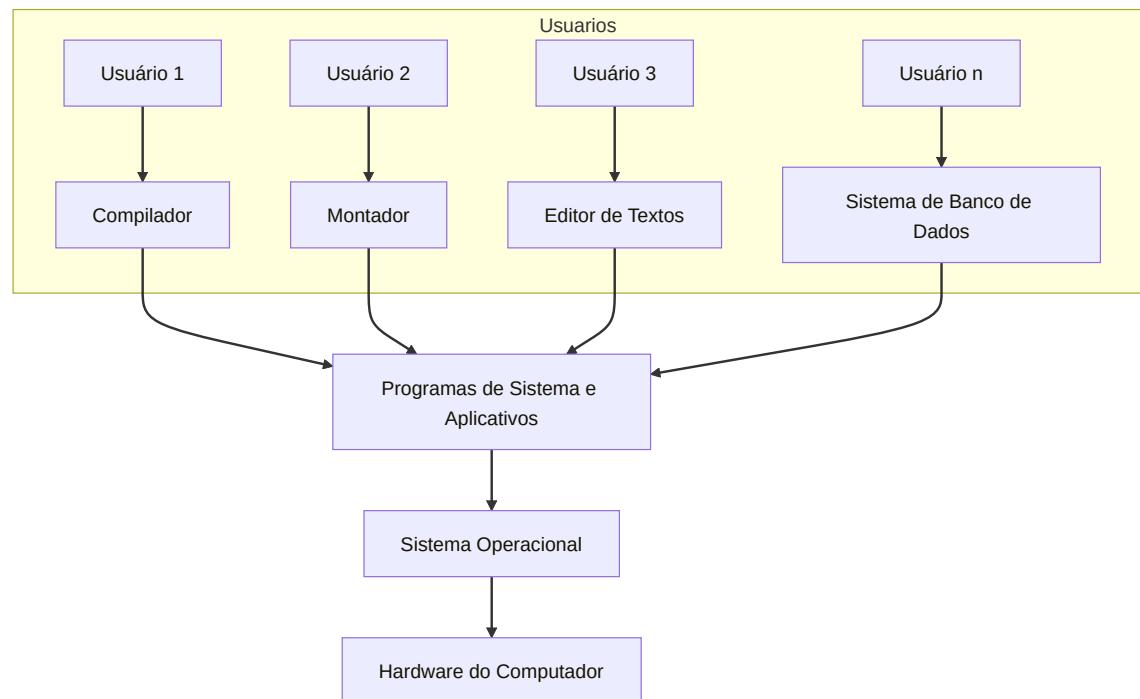
- Hardware
 - Sistema Operacional
 - Software
 - Usuários
- Também podemos considerar que um sistema computadorizado é composto por:



- Exemplo de Funcionamento de um Sistema Operacional



```
+-----+
| programas|
| de sistema|
| e aplicat-|
| ivos       |
+-----+
|
|
+-----+
| sistema   |
| operacional|
+-----+
|
|
+-----+
| hardware do|
| computador  |
+-----+
```



1.1.1 Hardware

O hardware de um computador é como os blocos fundamentais de Minecraft que compõem o mundo do seu computador. Assim como você precisa de diferentes tipos de blocos para construir estruturas complexas em Minecraft, um computador precisa de vários componentes de hardware para funcionar.

Componentes Principais

Processador (CPU)

Pense no processador como o jogador em Minecraft. Assim como o jogador executa ações e toma decisões, a CPU processa instruções e realiza cálculos. É o cérebro do computador.

Memória RAM

A RAM é como o inventário do jogador em Minecraft. Ela armazena temporariamente informações que o processador precisa acessar rapidamente, assim como você mantém itens importantes no seu inventário para uso imediato.

Armazenamento (HDD/SSD)

O armazenamento é semelhante aos baús em Minecraft. HDDs e SSDs guardam dados a longo prazo, como programas e arquivos, assim como os baús armazenam itens que você não precisa carregar o tempo todo.

Placa-mãe

A placa-mãe é como o terreno em Minecraft onde você constrói. Ela conecta todos os outros componentes, permitindo que eles se comuniquem entre si.

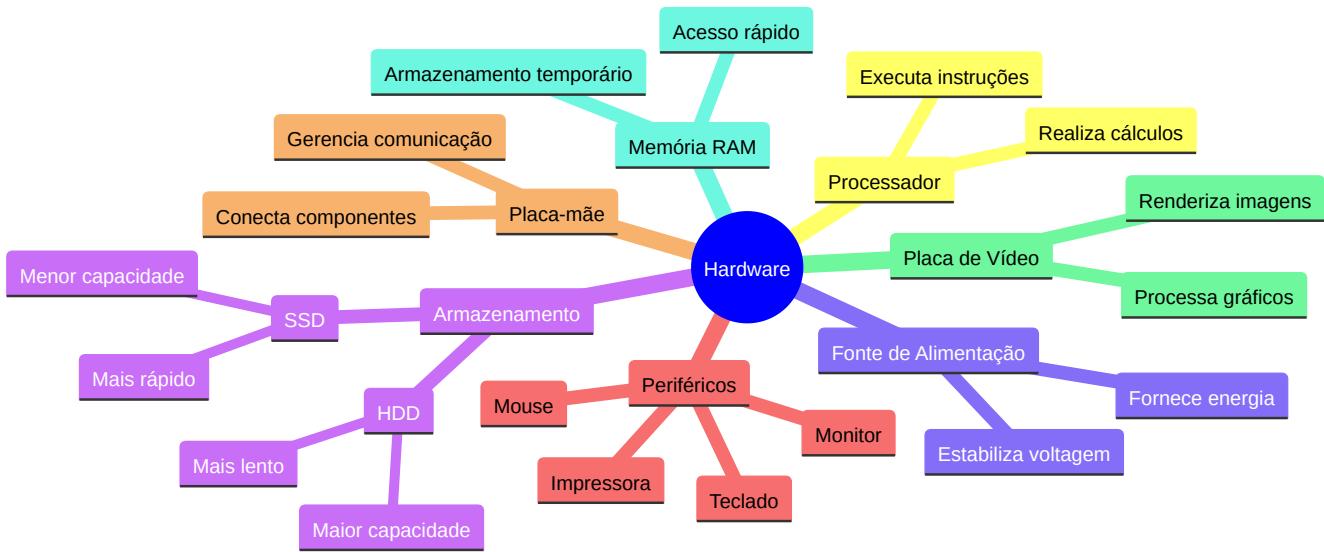
Placa de Vídeo (GPU)

A GPU é como o mecanismo de renderização em Minecraft. Ela processa gráficos e imagens, tornando possível ver o mundo digital na sua tela.

Fonte de Alimentação

A fonte de alimentação é como a energia redstone em Minecraft. Ela fornece energia para todos os componentes, mantendo tudo funcionando.

Mindmap do Hardware



Este mindmap ilustra os principais componentes de hardware de um computador, mostrando como eles se relacionam entre si, assim como diferentes estruturas em Minecraft se conectam para formar um mundo funcional.

Entender o hardware é essencial para compreender como os sistemas operacionais interagem com os componentes físicos do computador, gerenciando recursos e otimizando o desempenho, assim como um bom jogador de Minecraft gerencia seus recursos para construir e explorar eficientemente.

1.1.2 Software

Software é como o conjunto de regras e mecânicas que fazem o mundo de Minecraft funcionar. Assim como Minecraft tem diferentes tipos de mecânicas (como física, geração de mundo, interações de itens), um computador tem diferentes tipos de software que trabalham juntos para criar uma experiência funcional e interativa.

Tipos de Software

Sistema Operacional

O sistema operacional é como o modo de jogo em Minecraft (Sobrevivência, Criativo, etc.). Ele define as regras básicas de como o computador funciona e como os outros programas podem interagir com o hardware.

Aplicativos

Aplicativos são como os mods em Minecraft. Eles adicionam funcionalidades específicas ao sistema, permitindo que você realize tarefas como escrever documentos, navegar na internet ou editar imagens.

Drivers

Drivers são semelhantes aos comandos de bloco em Minecraft. Eles permitem que o sistema operacional se comunique com o hardware específico, assim como os comandos de bloco permitem interações complexas com o mundo do jogo.

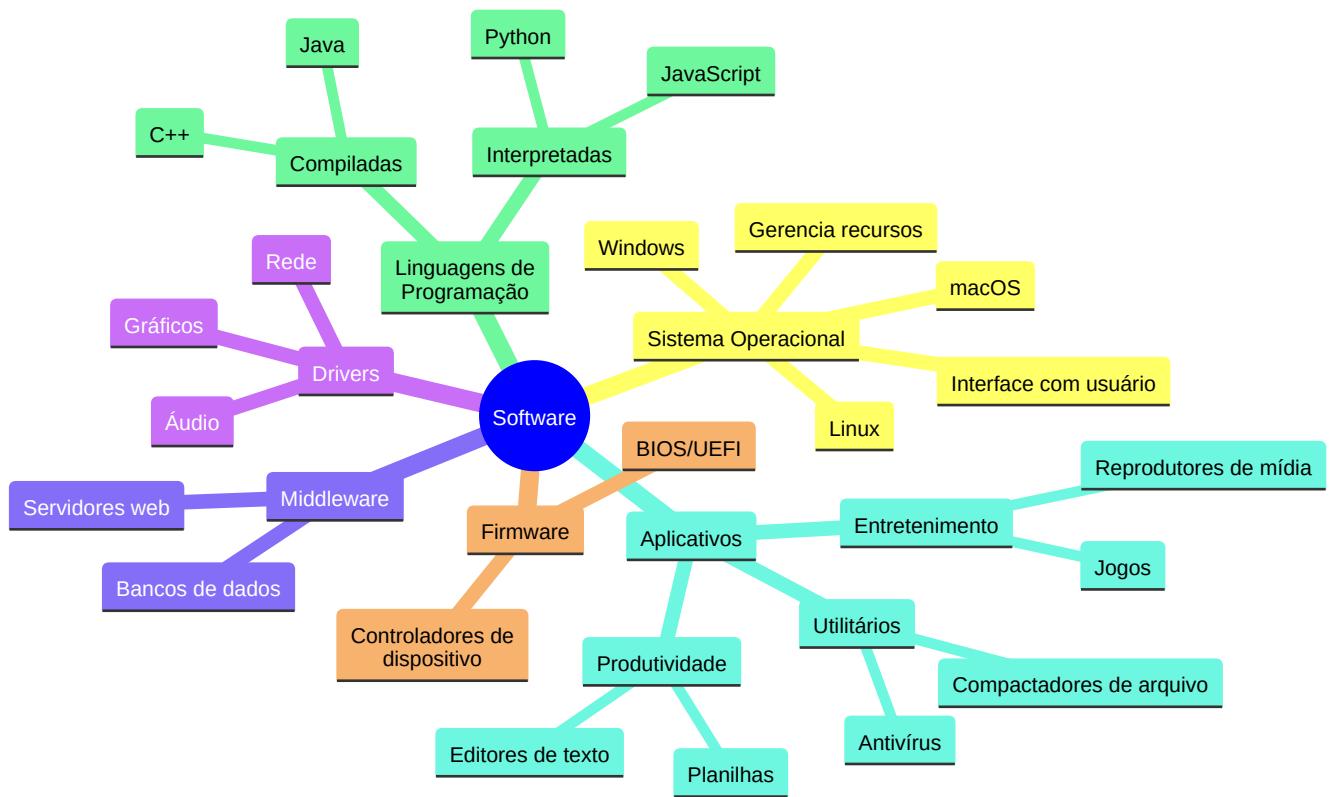
Firmware

O firmware é como as configurações internas dos blocos em Minecraft. É um software embutido no hardware que fornece instruções básicas para o funcionamento do dispositivo.

Linguagens de Programação

As linguagens de programação são como a linguagem de comandos em Minecraft. Elas permitem que os desenvolvedores criem software, assim como os comandos permitem aos jogadores criar comportamentos complexos no jogo.

Mindmap do Software



Este mindmap ilustra os principais tipos e categorias de software, mostrando como eles se relacionam e se organizam no ecossistema digital, assim como diferentes elementos se combinam para criar a experiência completa de Minecraft.

1.1.3 Visões do Sistema

Visão do Sistema: O Administrador do Servidor

Do ponto de vista do computador, o sistema operacional é como o administrador de um servidor Minecraft. Assim como um admin controla todos os aspectos do jogo, o sistema operacional gerencia intimamente o hardware do computador.

O Sistema Operacional como Alocador de Recursos

Imagine o sistema operacional como o sistema de plugins de um servidor Minecraft, responsável por gerenciar:

- 1. Tempo de CPU:** Como o dia e a noite no Minecraft, distribuindo tempo para cada processo.
- 2. Espaço de Memória:** Similar ao inventário dos jogadores, alocando espaço para programas.
- 3. Armazenamento de Arquivos:** Como baús no Minecraft, organizando e armazenando dados.
- 4. Dispositivos de E/S:** Portais para outros mundos, gerenciando a comunicação com dispositivos externos.

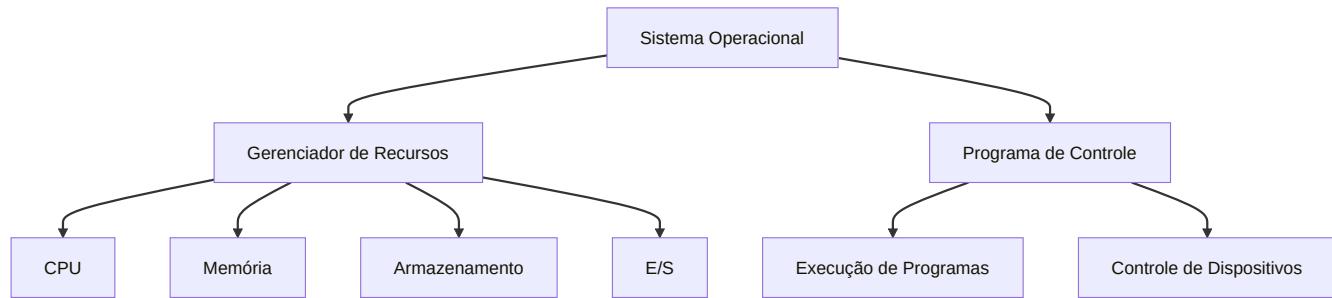
O sistema operacional deve alocar esses recursos de forma eficiente e justa, assim como um bom admin de Minecraft garante que todos os jogadores tenham acesso justo aos recursos do servidor.

Unidades de Armazenamento: Os Blocos do Mundo Digital

- **Bit:** O bloco mais básico, como um grão de areia no Minecraft.
- **Byte:** 8 bits, como um bloco completo no Minecraft.
- **Word:** A unidade nativa do computador, como um chunk no Minecraft.
- **Kilobyte (KB):** 1.024 bytes, como uma pequena construção.
- **Megabyte (MB):** 1.024^2 bytes, como uma vila inteira.
- **Gigabyte (GB):** 1.024^3 bytes, como um reino completo no Minecraft.

O Sistema Operacional como Programa de Controle

Assim como as regras e configurações de um servidor Minecraft, o sistema operacional controla a execução de programas e o uso de dispositivos para prevenir erros e uso indevido.



Este diagrama mostra como o Sistema Operacional, assim como o core de um servidor Minecraft, gerencia recursos e controla a execução de programas e dispositivos, mantendo todo o sistema funcionando harmoniosamente.

Visão do Usuário

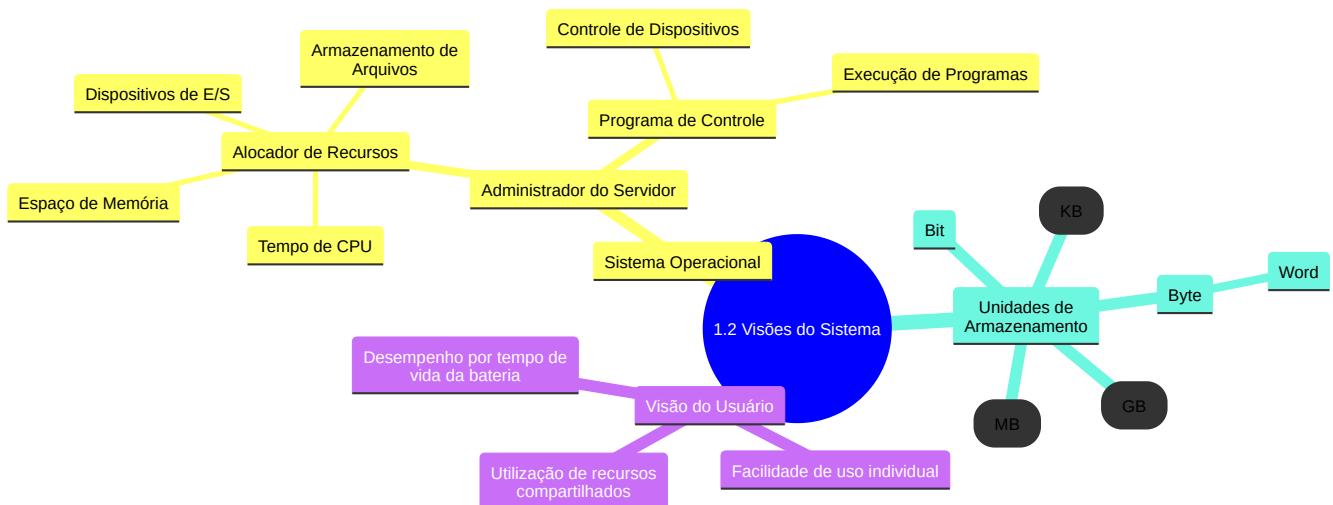
A visão do computador pelo usuário varia de acordo com a interface utilizada. Na maioria dos casos, os jogadores de Minecraft se sentam à frente de um computador, com um monitor, teclado, mouse e processador. Esse sistema foi projetado para que o jogador monopolize os recursos do computador.

O objetivo é proporcionar uma experiência mais rápida e imersiva no jogo. Nesse caso, o sistema operacional foi projetado principalmente para a facilidade de uso, com alguma atenção ao desempenho e pouca consideração à utilização de recursos – como a competição por espaço de memória e processamento.

É natural que o desempenho seja importante para o jogador; mas esses sistemas são otimizados para a experiência individual do jogador.

Em alguns casos, o jogador pode se conectar a um servidor remoto, permitindo que vários jogadores accessem o mesmo computador. Nesse caso, o sistema operacional foi projetado para um equilíbrio entre a facilidade de uso individual e a utilização de recursos compartilhados.

Alguns computadores podem ter pouca ou nenhuma visão do usuário. Por exemplo, os computadores embutidos nos consoles de jogos podem ter teclados numéricos e botões de luz indicadora, para mostrar o status do jogo, mas, em sua maioria, eles e seus sistemas operacionais são projetados para serem executados sem a intervenção do jogador.



1.2 Operação do Computador

Ao desligar o computador e ligá-lo, o que acontece? Como ele "chama" o Sistema Operacional.

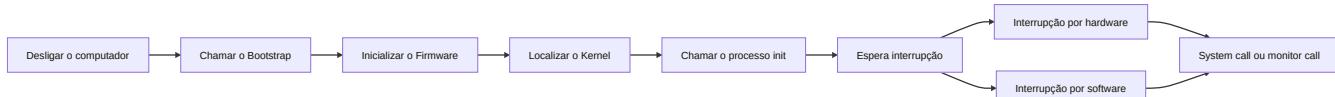
Para o computador começar a funcionar, ele chama um programa básico, chamado de **bootstrap**.

Normalmente, este programa está alocado na memória apenas de leitura (**ROM**) ou é salvo na memória de somente leitura apagável programavelmente (**EEPROM**).

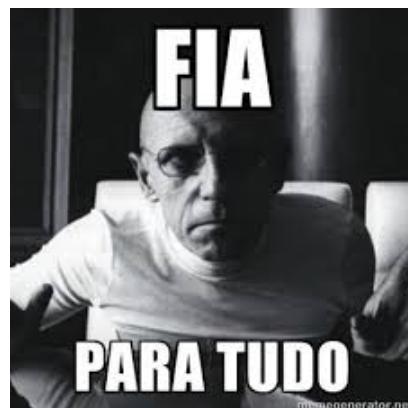
Este programa é conhecido como **Firmware**, pois está instalado diretamente no hardware, assim, ele inicializa todos os aspectos do sistema, desde os registradores da CPU até os dispositivos e o conteúdo na memória.

Para carregar o SO, ele precisa localizar o **Kernel**, que é o núcleo do sistema operacional. Assim que o Kernel é carregado na memória do computador, ele chama um processo chamado **init**, que espera uma interrupção do sistema ou do hardware. Os dois casos são:

- Se for pelo hardware, ele envia uma interrupção por sinal para a CPU, via normalmente o barramento do sistema;
- Se for por software, ele pode fazer de duas maneiras: chamando uma **system call** (chamada do sistema) ou usando um **monitor call** (monitor de chamada). Essas são operações especiais executadas para disparar uma interrupção, enviando um sinal para a CPU.



Quando a CPU recebe uma interrupção, ela para o que está fazendo e executa a rotina de tratamento correspondente:



Meme fia para tudo

A CPU então manda a execução para uma **localização fixa na memória**, onde essa localização contém o **endereço inicial** da rotina para **atender a essa interrupção**.

Essas **interrupções** podem ser tratadas de diferentes maneiras, e cada computador possui seu próprio mecanismo. Um método simples para isso é tratar a transferência chamando uma rotina genérica. Para dar mais enfoque em velocidade pode ser usada uma **tabela de ponteiros a pontando para as interrupções**, já que elas devem ser predefinidas. **Essa tabela é armazenada em memória baixa**, sendo ela a primeira parte ou locação da memória.

Esse **vetor de interrupção** vai ser indexado exclusivamente pelo número do dispositivo, fornecido com a requisição da interrupção para gerar o endereço do tratamento da interrupção:

Interrupção

CPU manda execução para local fixo na , com endereço da rotina de tratamento. 

Diferentes formas de tratar interrupções, cada  com seu próprio jeito.

Método simples:
Transfere para uma rotina genérica. 

Método rápido:
 Tabela de ponteiros para interrupções, em memória baixa. 

 Vetor usa dispositivo para gerar endereço do tratamento. 

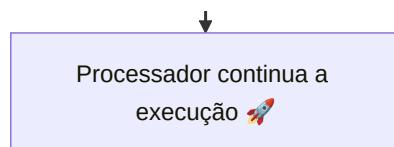
A arquitetura de interrupção **precisa salvar o endereço da instrução interrompida**, em projetos:

- Em alguns antigos armazenam o endereço da interrupção de maneira fixa ou local indexado por um numero do dispositivo;
- Em arquiteturas modernas, eles armazenam em **pilhas do sistema**;

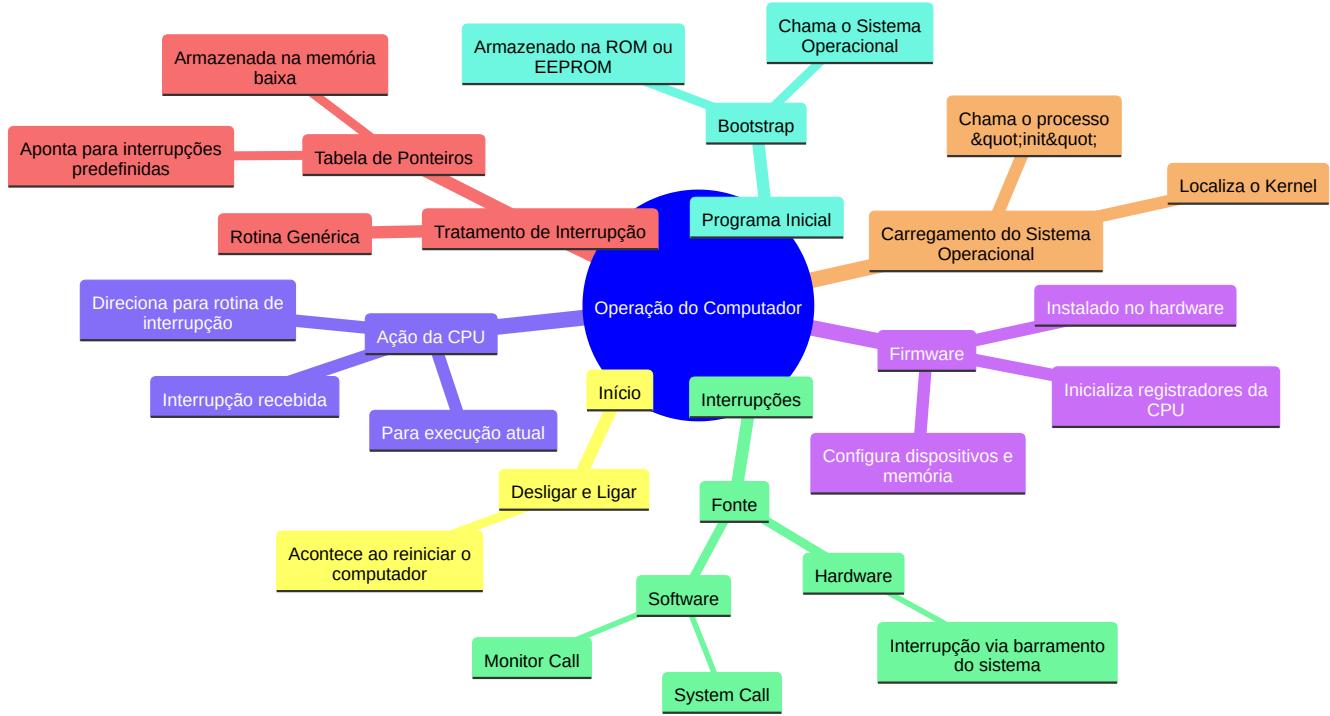
Se a rotina de interrupção precisar modificar algum estado do processador, por exemplo alterando os valores do **registrador**:

- Ela vai **salvar** o estado atual, explicitamente;
- Depois **carregar e restaurar** esse estado para depois **retornar**;
- Em seguida será carregado para o **contador de programa o endereço do retorno e o processador** que foi **interrompido** continua como se nada tivesse acontecido:





Diagrama

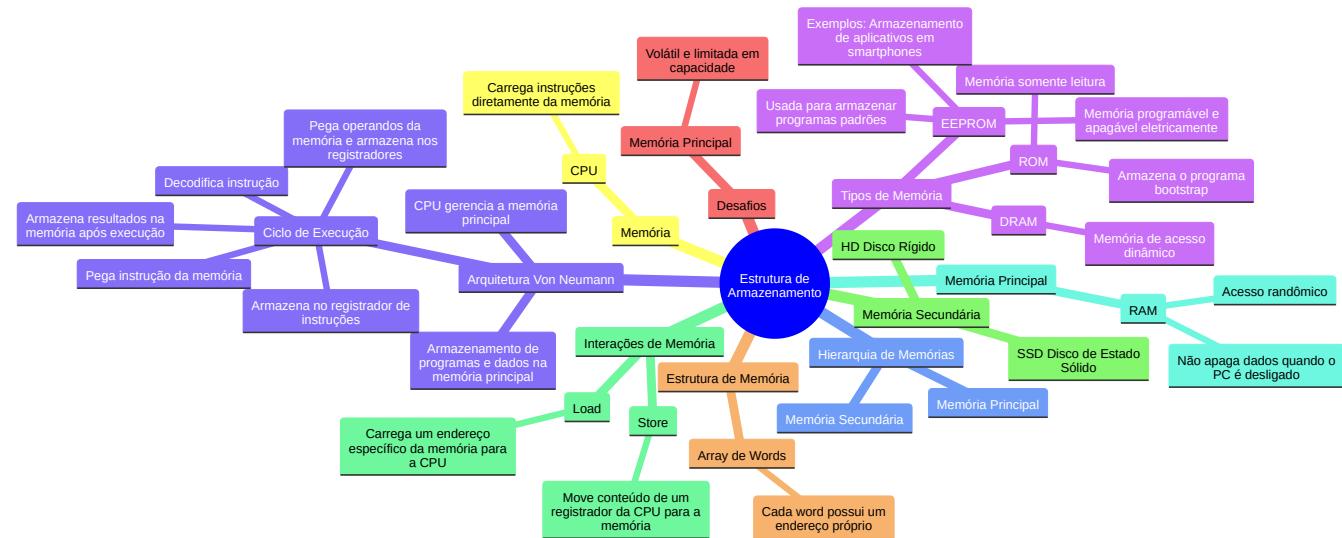


1.3 Estrutura de Armazenamento

Para os computadores que temos a **CPU** só consegue carregar instruções que vêm diretamente da memória.

- A memória não sendo nada, mas a **Memória Principal** – aquela cujo acesso é randômico, ou seja, desligar o PC não apaga os dados armazenados, que é a memória **RAM**.

Diagrama



Veja mais sobre tipos de memória em:

A memória RAM é comumente feita numa arquitetura de semicondutores chamada de **Dynamic Random Access Memory** (DRAM) ou, em português, **memória de acesso dinâmica**.

Um outro tipo de memória é aquela que só serve para leitura, assim como a mulher do seu amigo, apenas olhe. As conhecidas são:

- ROM (Read Only Memory)** ==> normalmente vem nos computadores e é usada para armazenar o programa bootstrap.
 - Além disso, é usada por empresas de jogos para guardar os jogos, já que ela possui essa natureza imutável.
- EEPROM (Electrically Erasable Programmable Read Only Memory)**
 - Por não ser modificada com frequência, essa memória costuma ser usada para armazenar

programas padrões de modo estático.

- Smartphones, por exemplo, utilizam a EEPROM de modo que as fabricantes armazenam nele os aplicativos de fábrica.

Quaisquer destas memórias utilizam **um array de words** ou uma unidade de armazenamento.

- Cada *word* possui seu próprio endereço.
- As interações se dão por instruções:
 - **load** - carrega um endereço específico da **memória principal** para um dos **registradores da CPU**.
 - **store** - move um conteúdo de um **registrador da CPU** para a **memória principal**.

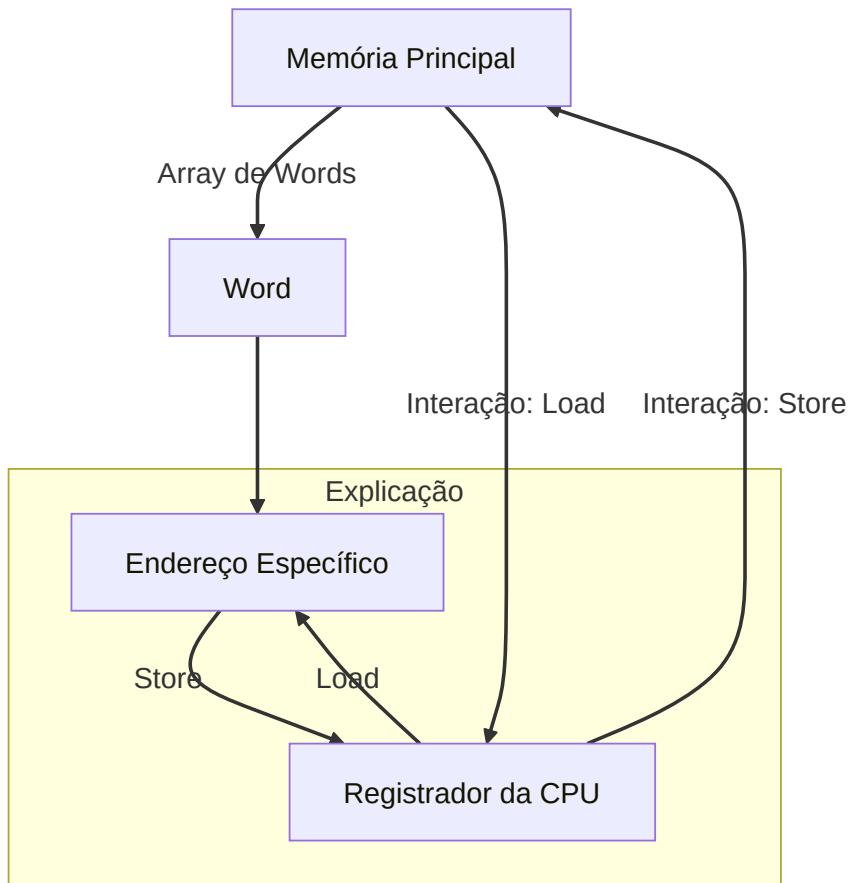


Ilustração de um esquema sobre instruções da CPU (*load* e *store*)

- i** A CPU carrega e armazena essas instruções tanto explicitamente (dizer para ela fazer) como de maneira automática - ela faz sozinha o carregamento da memória principal para serem executadas.

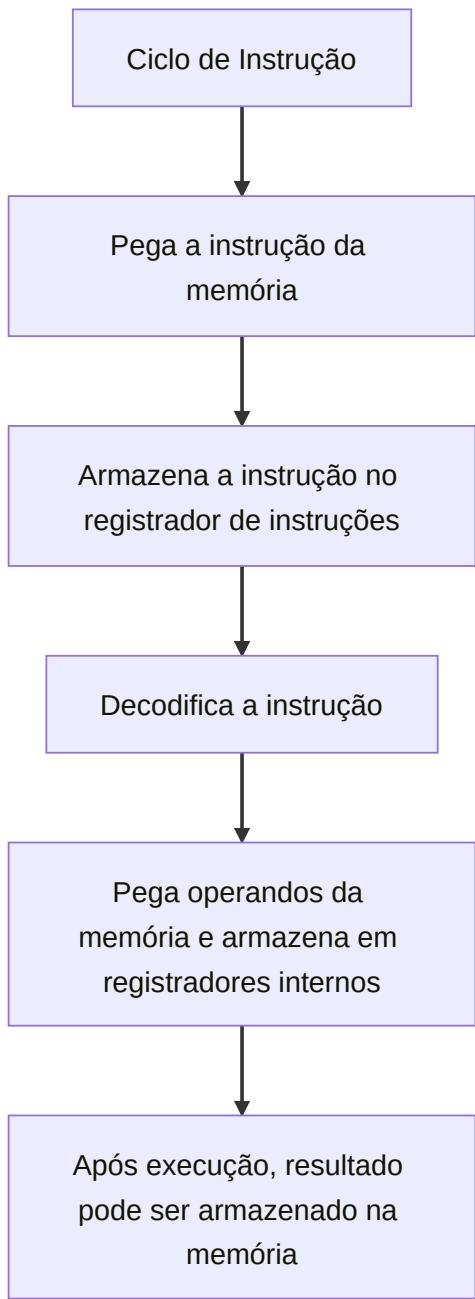
A arquitetura mais usada nos computadores modernos é a de **Von Neumann**. Essa arquitetura funciona da seguinte forma:

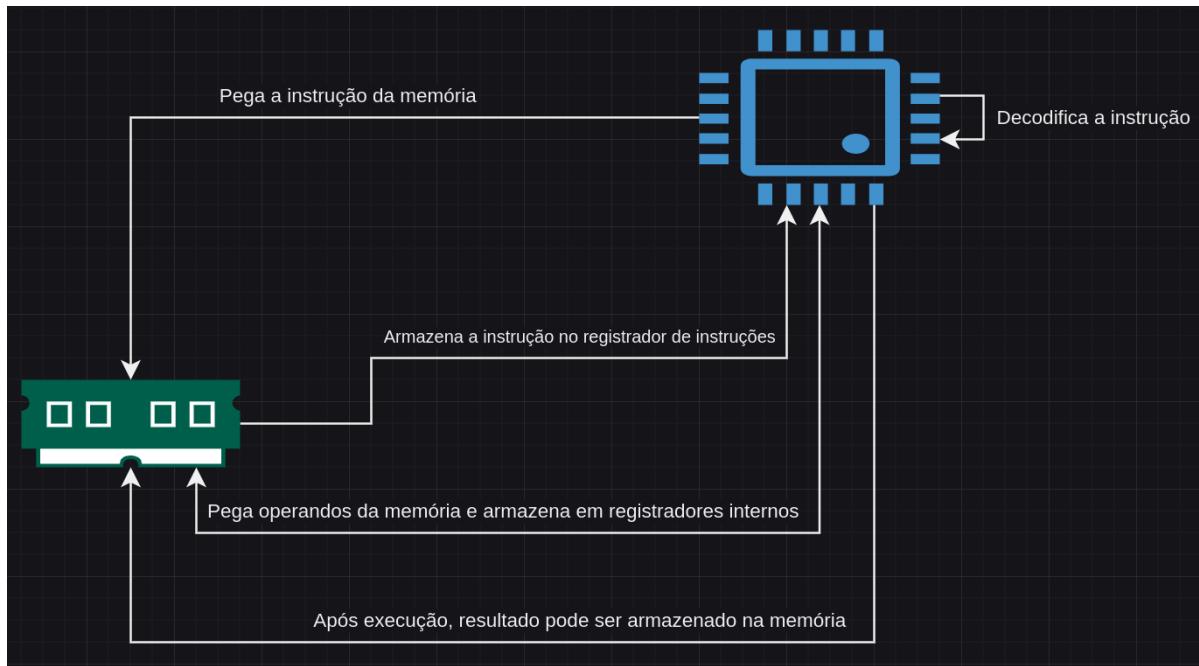
- Programas e dados são armazenados na memória principal.
- A CPU gerencia a memória principal.

Vamos para um ciclo de execução - quando uma instrução é dada:

1. Pega a instrução da memória.
2. Armazena essa instrução no **registraror de instruções**.
3. Essa instrução é então decodificada.
 1. Pode pegar operandos da memória e armazená-los em registradores internos.
4. Após a execução dos operandos, o resultado pode ser armazenado na memória.

Diagramas de Execução de Instrução





003 - Estrutura de Armazenamento

- i** A unidade de memória só consegue ver um fluxo de endereços de memória. Ela não sabe:
- Como são gerados (Gerados por contador de instruções, indexação, endereços literais e etc)
 - Para que servem
 - Se são instruções ou dados.

Seria bom, mas a vida não é um morango, a memória principal não consegue armazenar todos os dados e programas. Entretanto, não temos isso, já que:

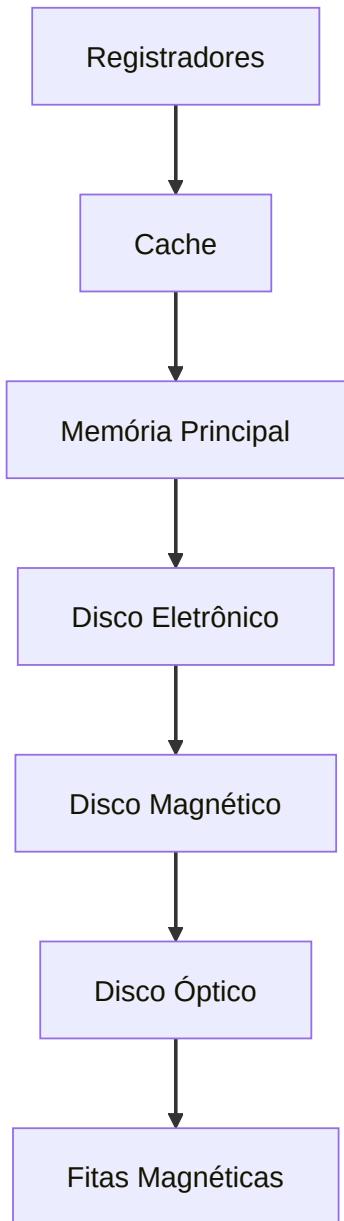
- A memória principal é volátil, ela perde os dados assim que a máquina é desligada.
- A memória principal possui um armazenamento irrisoriamente pequeno para armazenar todos os programas e dados.

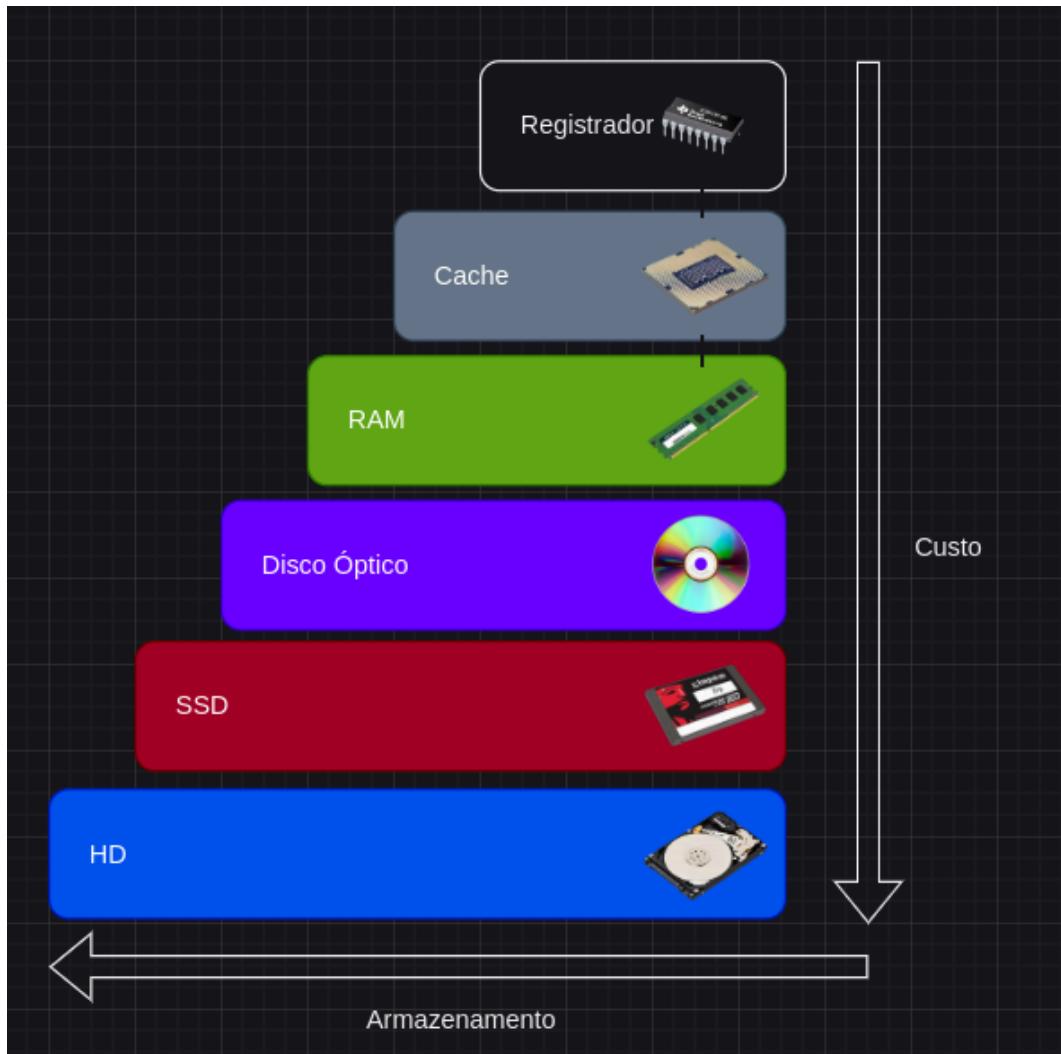
Assim, precisamos de outro tipo de memória chamado memória secundária, que tem o propósito de armazenar dados e programas de maneira permanente.

Um bom exemplo de memória secundária é o HD (Disco Rígido) e também temos outro tipo que está se tornando mais popular no mercado, o SSD (Disco de Estado Sólido).

No entanto, não há apenas dispositivos de armazenamento nessa hierarquia. Também podemos fazer uma hierarquia desses dispositivos, que é assim:

Diagramas de Dispositivos de Armazenamento:





003 - Estrutura de Armazenamento Hierarquia Dispositivos De Armazenamento

1.4 Estrutura de Entrada e Saída

Os dispositivos de Entrada e Saída (ou E/S), são um dos grandes pontos importantes para um Sistema Operacional, como podemos notar no armazenamento que possui grande importância para ser um dispositivo de E/S.

- Um outro ponto importante é que grande parte do código do SO é pensado para E/S;
 - Tanto por causa da **confiabilidade** como **desempenho**.

i Um sistema computadorizado para uso geral, consiste em:

- CPU
- Diversos tipos de controladores de dispositivos conectados por um barramento comum
- Cada controlador possui um tipo específico de dispositivo

Por exemplo, para o controlador SCSI (Small Computer-System Interface) podemos ter sete ou até mais dispositivos conectados ao mesmo controlador.

Cada controlador armazena **buffer local** e um **conjunto de registradores de uso especial**.

Os controladores tem duas funções básicas, que se baseiam:

- **Move** os dados para os dispositivos periféricos que controla.
- **Gerencia** o uso do buffer local.

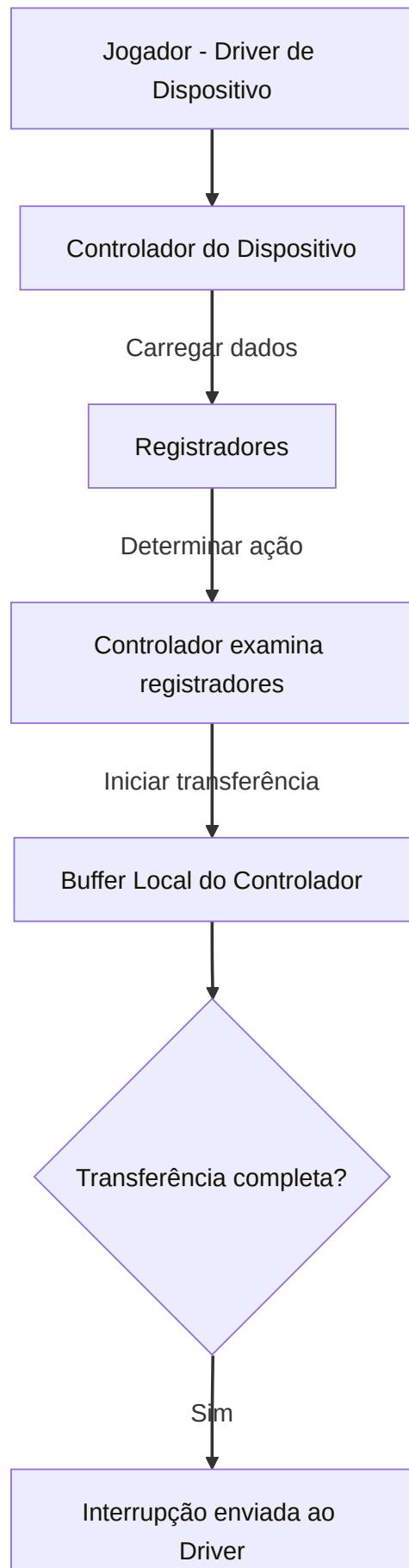
Tais sistemas possuem um **driver de dispositivo** (driver de dispositivo) que serve como ponte entre o dispositivo e o sistema, permitindo que a **entrada dos dispositivos** tenha uma **saída uniforme** para o restante do sistema.

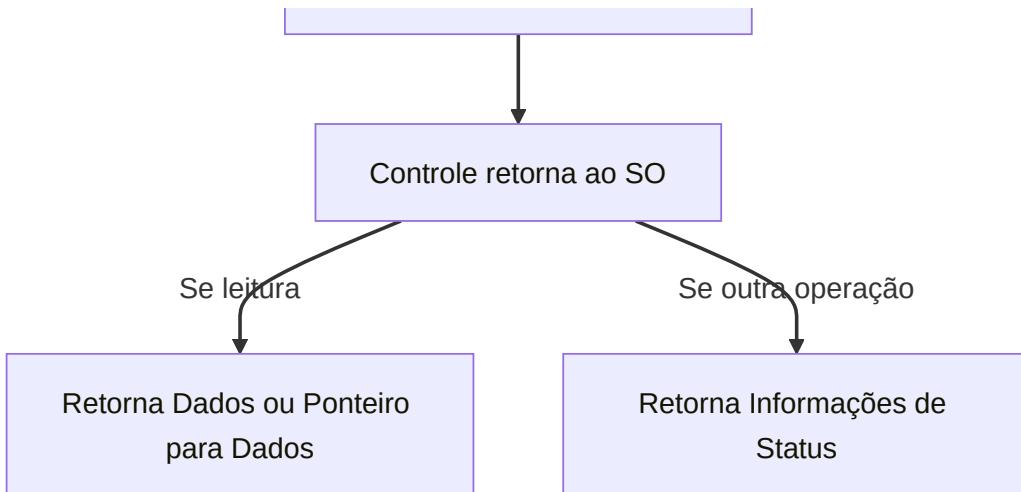
O funcionamento de uma operação de E/S:

- O **driver de dispositivo** carrega os **registradores** apropriados para dentro do **controlador do dispositivo**.
- O **controlador** examina o **conteúdo** que tem nos **registradores**, para determinar que ação deve ser tomada.

- O controlador começa a transferir os dados do dispositivo para o seu buffer local.
- Assim que a transferência está concluída, o **controlador de dispositivo** envia uma **interrupção** para o **driver de dispositivo** informando que a transferência foi concluída.
- O driver de dispositivo então retorna o controle diretamente para o SO, retornando os dados ou um ponteiro para esses dados, possivelmente, caso a operação seja de leitura.
 - Para outras operações, o driver retorna informações de status.

Representação:





- Para pequenas porções de dados, essa arquitetura de E/S por interrupção funciona bem, mas não funciona somente com isso há muito tempo, por isso, se usarmos essa forma para grandes volumes de dados como E/S de disco causa um **overhead** (que é uma sobrecarga).

Com esse grande problema, precisamos então de um outro dispositivo, um que armazene esses dados para que o acesso seja mais rápido, para isso usamos a **DAM** (Direct Access Memory ou Memória de Acesso Direto).

Logo o ciclo se torna assim:

- Depois de configurar buffers, ponteiros e contadores, o dispositivo de E/S, o controlador de dispositivo **move um bloco inteiro de dados** diretamente para ou do seu próprio buffer local para a memória.
 - Somente **uma interrupção é feita por bloco**, para que seja avisado ao driver de dispositivo que a **transferência foi concluída**.

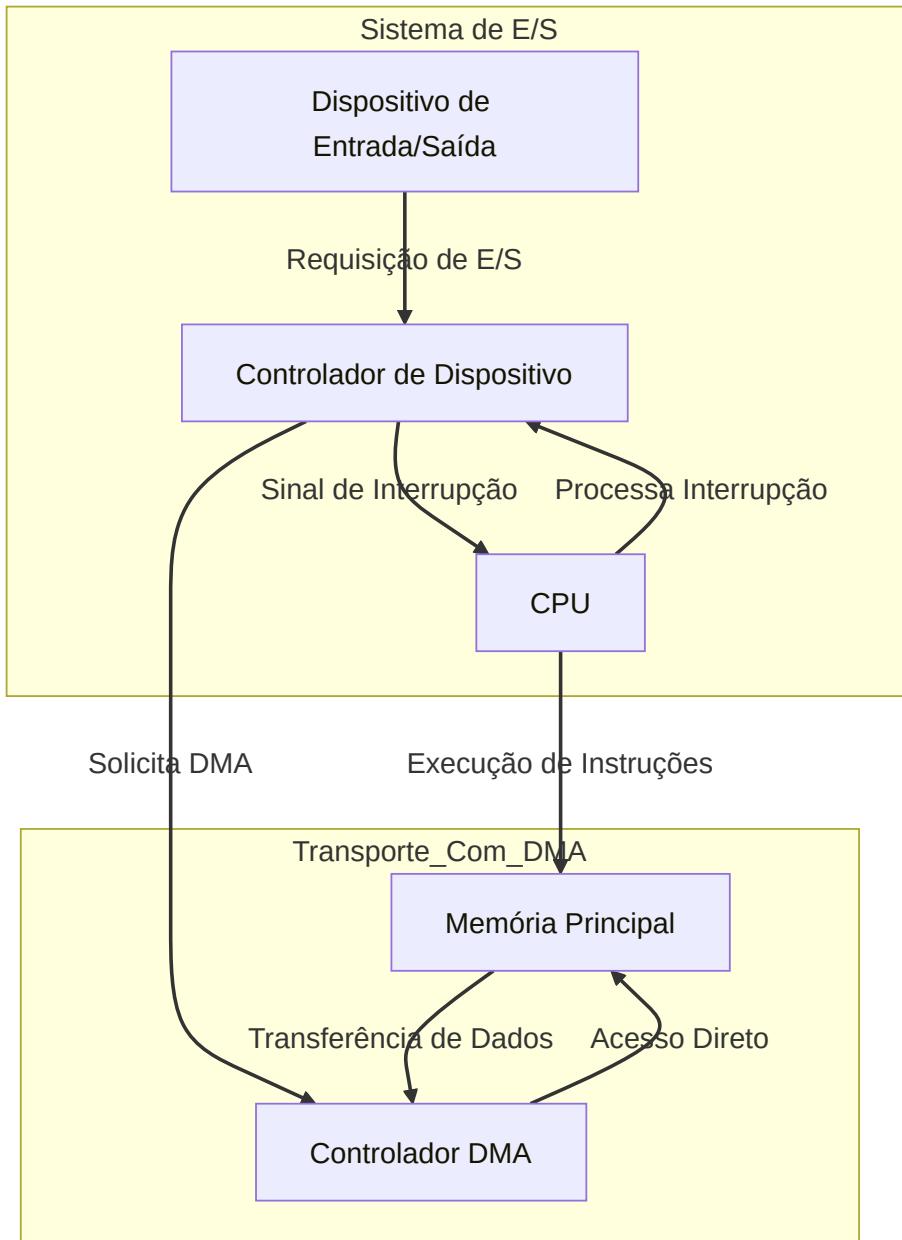
- Nesta etapa de transferência direta não ocorre intervenção da CPU, assim apenas o controlador de dispositivo cuida dessa tarefa.

Para alguns sistemas não é utilizado essa arquitetura de barramento e sim de switch:

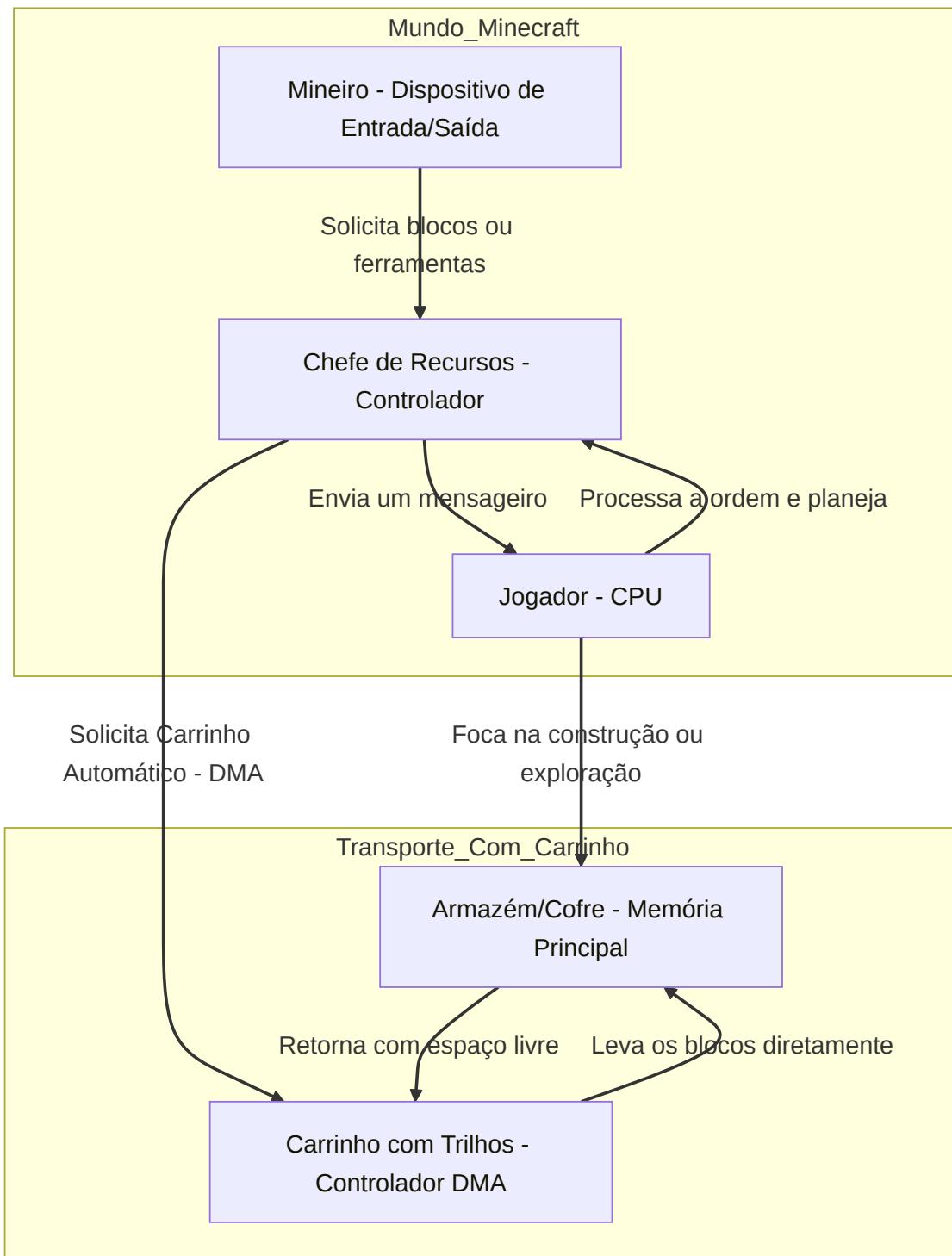
- Nesse tipo de sistema, os vários componentes do sistema podem interagir entre si ao mesmo tempo.
- Ao invés de competir por ciclos de um barramento compartilhado.

- Assim o DMA consegue ser ainda mais eficiente.

Representação da interação dos componentes num sistema:



- Com Mineiro:*



1.5 Arquitetura do Sistema

Agora falaremos sobre a categorização dos sistemas computadorizados, que é feita com base no número de processadores que ele possui, ou seja, estamos nos referindo a computadores de uso geral.

1.5.1 Sistema Monoprocessador

Esses sistemas, como o nome diz, possuem um único processador e foram muito utilizados, desde PDAs até mainframes. Assim, esses sistemas contêm uma única CPU que pode realizar diversas instruções de uso geral, assim como os processos do usuário.

- A maioria dos sistemas utiliza um processador de uso específico, como, por exemplo, para processamento gráfico, com os controladores gráficos, ou nos mainframes, com os processadores de E/S.

Esses processadores específicos não executam processos do usuário e somente realizam instruções limitadas e especializadas.

- Em alguns casos, o sistema operacional controla esse componente, pois o sistema envia informações sobre sua próxima tarefa e monitora seu status.

Exemplo:

- Um processador controlador de disco recebe uma sequência de requisições da CPU principal.
- Implementa sua própria fila de disco e algoritmo de escalonamento.

- Com isso, há um alívio na carga de processamento do escalonamento de disco, que, de outra forma, seria delegado à CPU principal.

O sistema operacional não pode se comunicar diretamente com esses processadores, pois eles operam em um nível mais baixo. Um exemplo disso são os teclados, que possuem um microprocessador responsável por converter os toques nas teclas em códigos que serão enviados para a CPU principal.

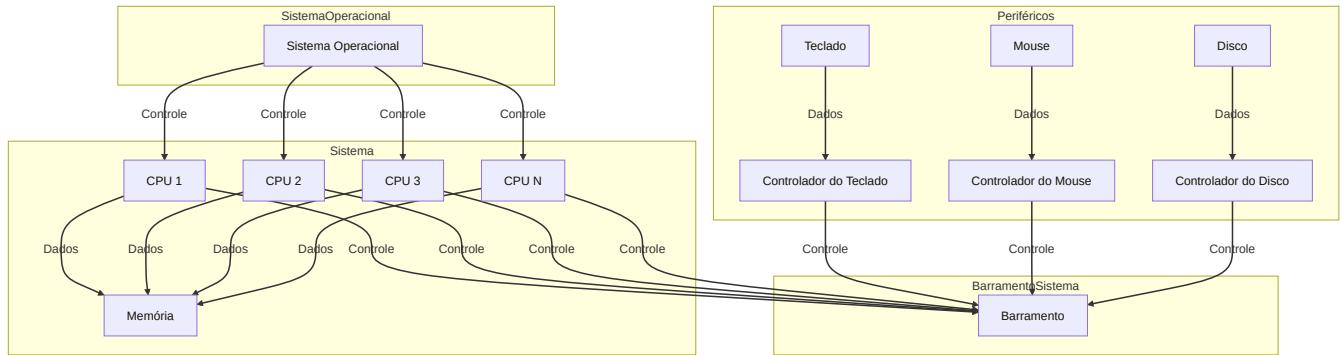
Assim, esses processadores realizam suas tarefas de forma anônima, pois não interagem diretamente com o sistema operacional.

Mesmo com o uso desses processadores específicos, o sistema ainda não é considerado

multiprocessado.

Para que um sistema seja classificado como monoprocessador, ele deve possuir uma única CPU de uso geral. Os processadores mencionados anteriormente são de uso específico.

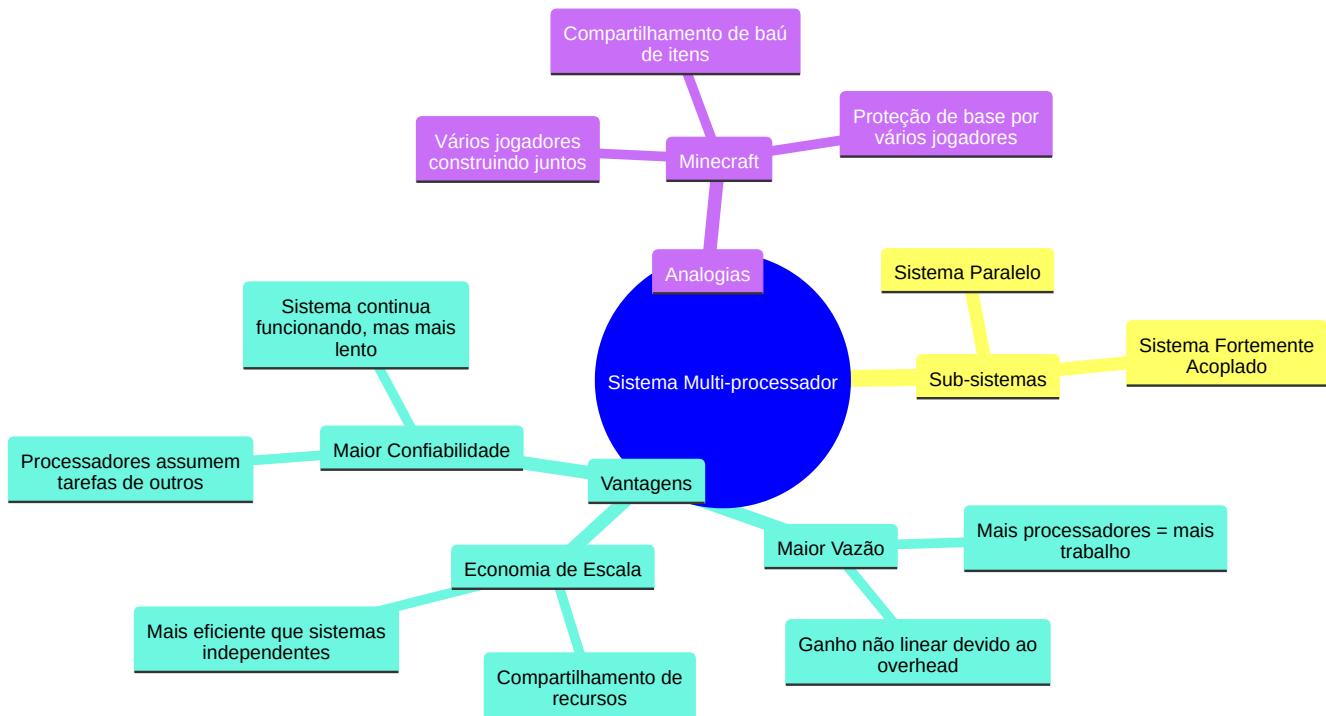
Diagrama



1.5.2 Sistema multi-processador

Esse tipo de sistema em que temos mais de um processador, de uso geral, dentro de um mesmo sistema computadorizado tem ganhado cada vez mais espaço por diversas razões o lugar dos sistemas mono processador.

Os sistemas multiprocessados, ou também conhecidos como: **sistemas paralelos** (parallel system) ou **sistema fortemente acoplado** (tightly coupled system) fazem um compartilhamento perfeito de periféricos, relógio do computador, barramento do computador para vários processadores de modo que a comunicação entre eles é perfeita.



Podemos escalar **três grandes vantagens** acerca desse tipo de arquitetura para sistemas:

1. Maior vazão:

- Como ter vários jogadores trabalhando juntos em uma construção no Minecraft.
- Mais processadores = mais trabalho realizado em menos tempo.
- Porém, o ganho não é linear devido ao overhead de coordenação.

2. Economia de escala:

- Semelhante a compartilhar um baú de itens entre vários jogadores no Minecraft.
- Sistemas multiprocessados compartilham recursos (periféricos, armazenamento, energia).
- Mais eficiente que ter vários sistemas independentes.

3. Maior confiabilidade:

- Como ter vários jogadores protegendo uma base no Minecraft.
- Se um processador falha, os outros podem assumir suas tarefas.
- O sistema continua funcionando, apenas mais lento, em vez de travar completamente.

Estas vantagens tornam os sistemas multiprocessados cada vez mais populares, assim como servidores de Minecraft com vários jogadores oferecem uma experiência mais robusta e dinâmica.

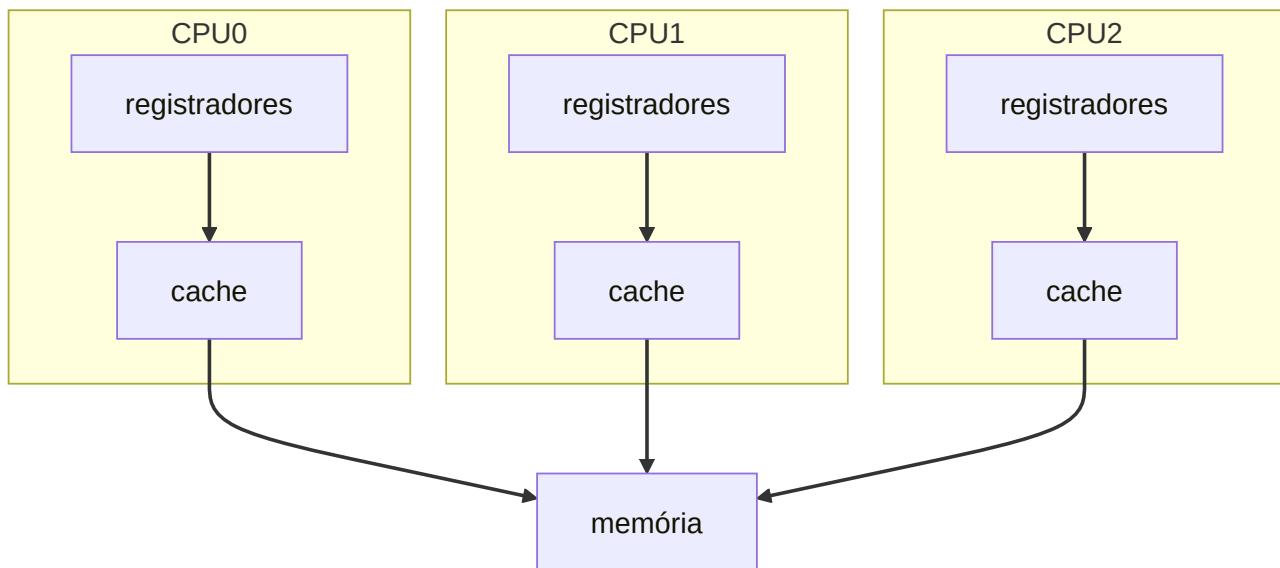
Imagine construir um mundo no Minecraft. A **confiabilidade** do sistema é como a estabilidade do mundo: se algo der errado (bloco sumir, mob bugar), ele continua funcionando, mesmo que limitado. Isso é **degradação controlada** — como minerar com uma ferramenta pior se a melhor quebrar.

Sistemas **tolerantes a falhas** vão além: mesmo com falhas, funcionam sem interrupções. No Minecraft, seria um backup automático que restaura blocos destruídos por creepers sem você sair do jogo.

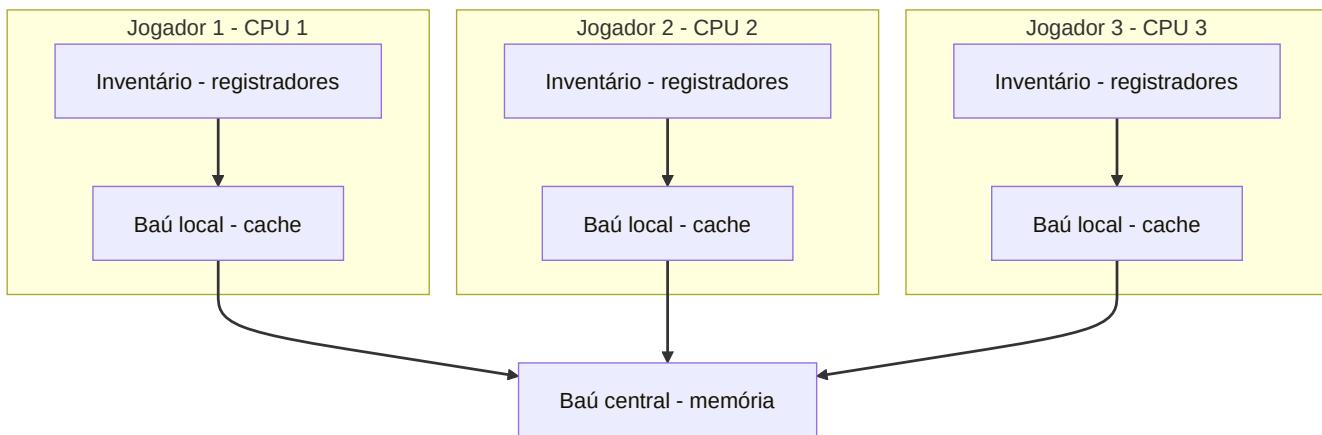
O **HP NonStop** é como um servidor com duplicação: dois jogadores (CPUs) constroem a mesma coisa ao mesmo tempo. Se um errar, o sistema corrige e transfere a tarefa para outro par, garantindo continuidade, mas com custo maior.

Já os **sistemas multiprocessados** são como vários jogadores trabalhando juntos:

1. **Assimétrico**: Um jogador mestre comanda os outros. Se ele sair, tudo pode parar.
2. **Simétrico (SMP)**: Todos são iguais, compartilham recursos (baú/memória) e trabalham juntos sem perder desempenho. Sistemas como **Solaris**, Windows e Linux usam isso.



- Com Minecraft:



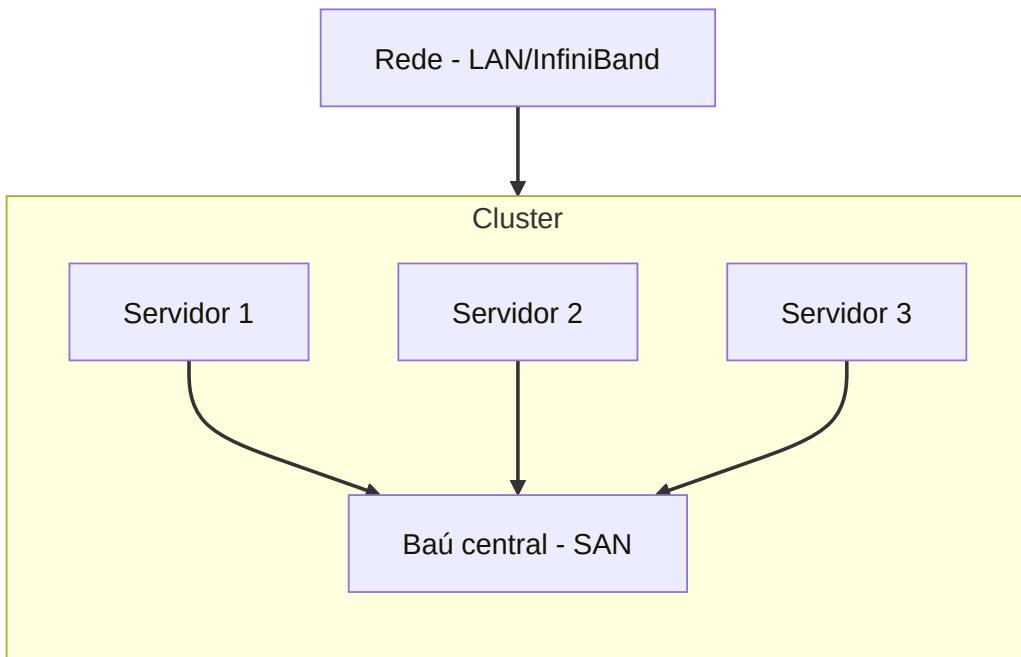
1.5.3 Sistemas em Clusters

Resumo com analogias ao Minecraft:

Um **sistema em cluster** é como um grupo de servidores de Minecraft trabalhando juntos. Cada servidor (nó) é independente, mas eles estão conectados por uma rede (LAN ou conexão rápida) e compartilham armazenamento (como um baú central). O objetivo é garantir **alta disponibilidade e alto desempenho**.

- **Alta disponibilidade:** Se um servidor falhar (explodir como um creeper), outro assume seu lugar, mantendo o mundo (serviço) funcionando com pouca interrupção.
- **Modo assimétrico:** Um servidor fica de olho (hot-standby) enquanto o outro roda o jogo. Se o ativo falhar, o standby assume.
- **Modo simétrico:** Vários servidores rodam o jogo e se monitoram, usando todo o hardware de forma eficiente.
- **Alto desempenho:** Vários servidores podem trabalhar juntos para resolver tarefas complexas, como gerar chunks ou processar comandos em paralelo. Isso exige que o jogo (aplicação) seja dividido em partes que rodam simultaneamente em diferentes servidores.
- **Clusters paralelos:** Vários servidores acessam os mesmos dados (como um banco de dados compartilhado). Para evitar conflitos, um sistema de "trava" (DLM) garante que apenas um servidor modifique os dados por vez.
- **SANs (Storage-Area Networks):** É como um baú gigante conectado a todos os servidores. Se um servidor cair, outro pode pegar os itens (dados) e continuar o jogo.

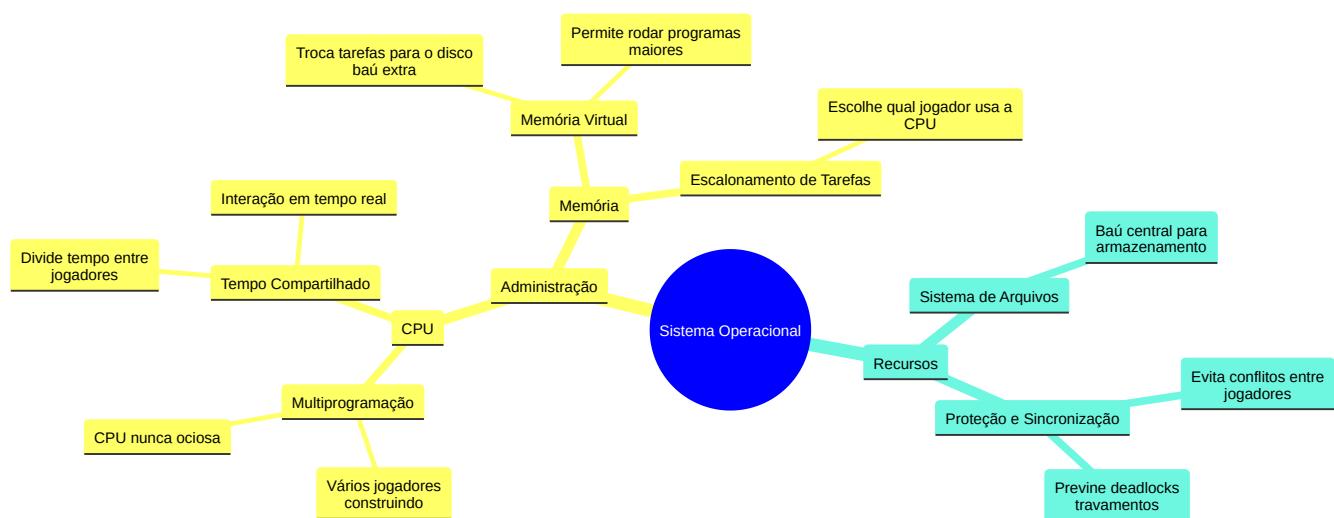
Resumo visual:



1.6 Estrutura do sistema operacional

Um **sistema operacional** é como o "administrador" de um servidor de Minecraft. Ele gerencia recursos (CPU, memória, dispositivos) e permite que vários programas (ou jogadores) funcionem ao mesmo tempo.

- **Multiprogramação:** É como ter vários jogadores construindo no mesmo mundo. Se um jogador precisa esperar (por exemplo, para minerar), o sistema passa para outro, mantendo a CPU sempre ocupada. Isso evita que o servidor fique ocioso.
- **Tempo compartilhado (time sharing):** É como dividir o tempo do servidor entre vários jogadores. Cada um recebe um pouco de atenção do servidor, mas tão rápido que parece que todos estão jogando ao mesmo tempo. Isso permite interação em tempo real, como digitar comandos e ver resultados imediatos.
- **Escalonamento de tarefas:** O sistema escolhe qual jogador (tarefa) deve usar o servidor (CPU) a seguir, garantindo que todos tenham uma chance justa.
- **Memória virtual:** Se o servidor não tem espaço para todos os jogadores (tarefas) na memória, ele "troca" alguns para o disco (como um baú extra) e os traz de volta quando necessário. Isso permite rodar programas maiores do que a memória física.
- **Sistema de arquivos:** É como o baú central do servidor, onde todos os itens (arquivos) são armazenados e organizados.
- **Proteção e sincronização:** O sistema garante que os jogadores (tarefas) não interfiram uns com os outros, evitando conflitos e travamentos (deadlocks).



1.7 Operações do Sistema Operacional

Resumo com analogias ao Minecraft:

O sistema operacional é como o "administrador" de um servidor de Minecraft, controlando tudo que acontece no mundo (sistema). Ele usa **interrupções e traps** para lidar com eventos, como um jogador tentando fazer algo que não deveria (erro) ou pedindo ajuda (chamada de sistema).

1. Modo Dual (Usuário e Kernel):

- **Modo Usuário:** Onde os jogadores (programas de usuário) operam. Eles têm permissão limitada, como construir ou minerar, mas não podem alterar o servidor diretamente.
- **Modo Kernel:** Onde o administrador (sistema operacional) opera. Ele tem controle total sobre o servidor, como gerenciar recursos, corrigir erros ou expulsar jogadores problemáticos.
- **Transição:** Quando um jogador precisa de algo que só o administrador pode fazer (como abrir um portal), ele faz uma **chamada de sistema**, e o servidor muda para o modo kernel temporariamente.

2. Proteção:

- O sistema operacional protege o servidor de jogadores mal-intencionados ou erros. Por exemplo, se um jogador tentar destruir o servidor (executar uma instrução privilegiada no modo usuário), o sistema bloqueia a ação e notifica o administrador.

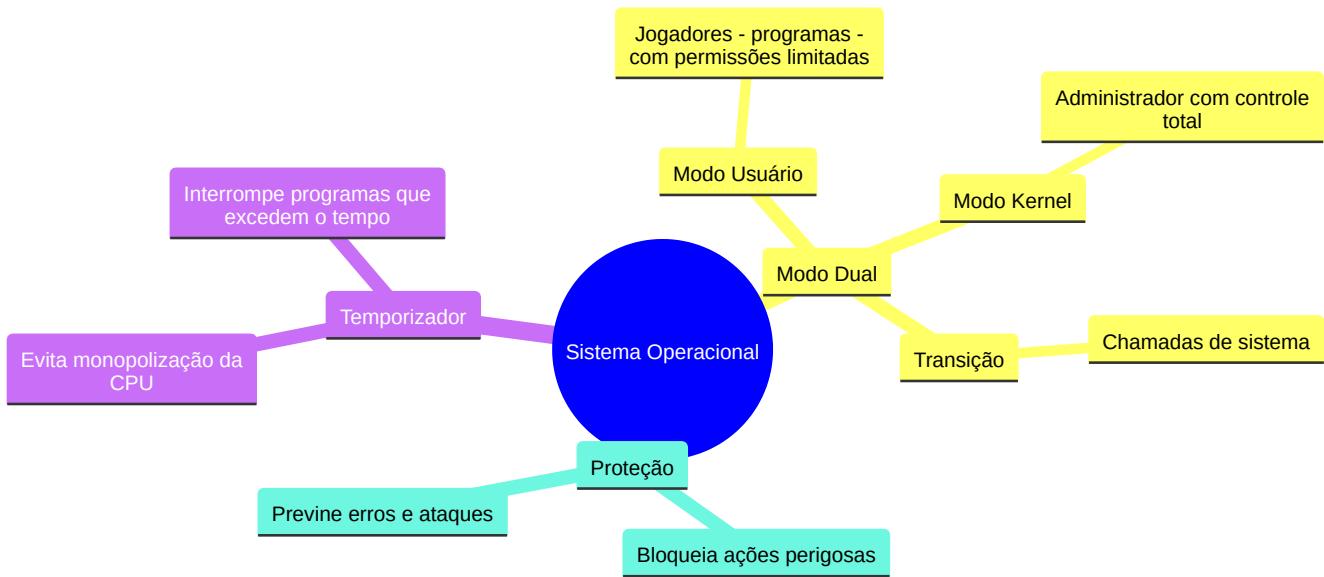
3. Temporizador:

- Para evitar que um jogador monopolize o servidor (loop infinito), o sistema usa um **temporizador**. Se um jogador ficar muito tempo sem ceder a vez, o sistema interrompe e passa o controle para outro jogador ou para o administrador.

4. Ciclo de Execução:

- O sistema operacional começa no modo kernel (administrador) ao ligar o servidor. Ele carrega os jogadores (programas) no modo usuário e alterna entre os modos conforme necessário, garantindo que tudo funcione sem problemas.

Resumo visual:



Em resumo, o sistema operacional é como um administrador de servidor de Minecraft, alternando entre modos para garantir que os jogadores (programas) possam jogar sem causar problemas, enquanto mantém o controle total sobre o sistema.

1.8 Gerência de processos

Um **processo** é como um jogador em um servidor de Minecraft. Um programa (arquivo no disco) é só um conjunto de instruções, mas quando ele é executado, vira um processo (jogador ativo). Cada processo precisa de recursos como tempo de CPU (atenção do servidor), memória (espaço no inventário), e dispositivos de E/S (ferramentas e blocos).

1. Processo vs. Programa:

- **Programa:** É como um livro de instruções para construir algo no Minecraft (passivo).
- **Processo:** É um jogador seguindo essas instruções e construindoativamente (ativo).

2. Recursos do Processo:

- Cada processo (jogador) recebe recursos do sistema operacional (administrador do servidor), como tempo de CPU, memória e acesso a arquivos ou dispositivos.
- Quando o processo termina (jogador sai), os recursos são devolvidos ao sistema.

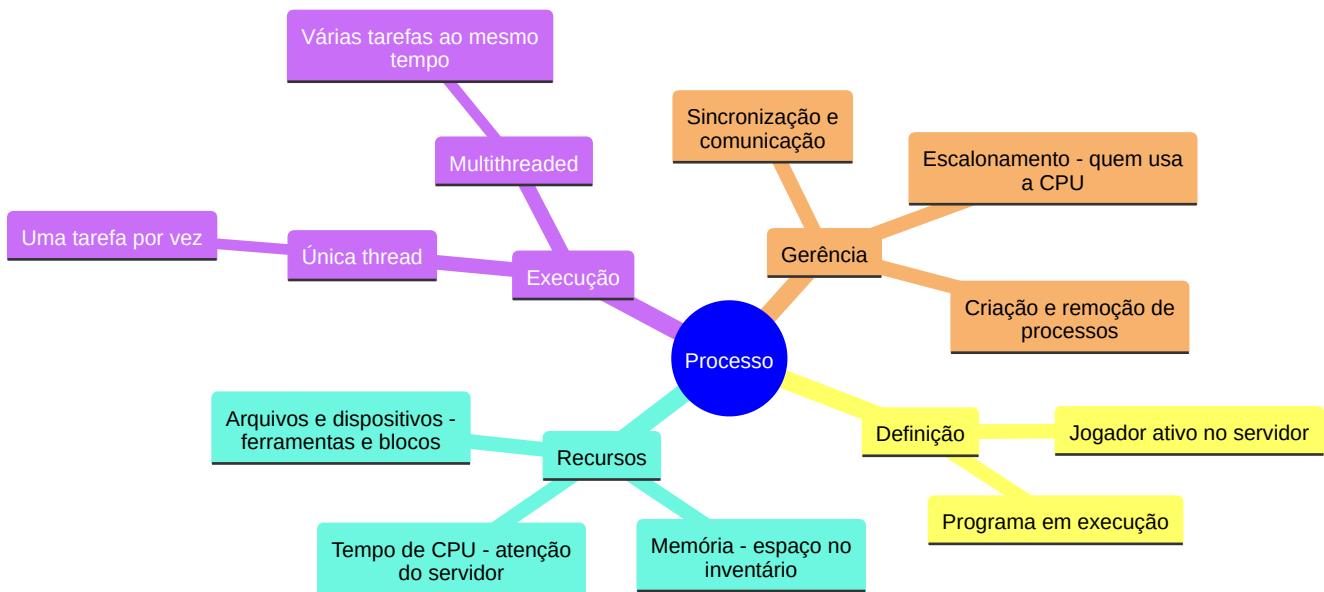
3. Execução de Processos:

- Um processo de **única thread** é como um jogador com uma única tarefa, seguindo uma sequência de instruções (contador de programa).
- Um processo **multithreaded** é como um jogador com várias tarefas ao mesmo tempo (vários contadores de programa).

4. Gerência de Processos:

- O sistema operacional (administrador) gerencia os processos (jogadores), decidindo quem usa a CPU (escalonamento), criando ou removendo processos, e garantindo que eles não interfiram uns com os outros (sincronização e comunicação).

Resumo visual:



Em resumo, um processo é como um jogador ativo no servidor de Minecraft, usando recursos e seguindo instruções. O sistema operacional é o administrador que gerencia todos os jogadores, garantindo que tudo funcione sem problemas.

1.9 Gerência de memória

Resumo com analogias ao Minecraft:

A **memória principal** é como o inventário do jogador no Minecraft. Ela armazena dados e instruções que a CPU (jogador) precisa para executar tarefas rapidamente. Assim como o inventário tem espaço limitado, a memória principal também tem um tamanho finito e precisa ser gerenciada com cuidado.

1. Função da Memória Principal:

- É o "inventário" do computador, onde a CPU busca instruções e dados para executar programas.
- Para que um programa rode, ele precisa ser carregado na memória, como colocar itens no inventário.

2. Acesso Direto:

- A CPU só pode acessar diretamente a memória principal. Dados de dispositivos como discos (baús externos) precisam ser transferidos para a memória antes de serem usados.

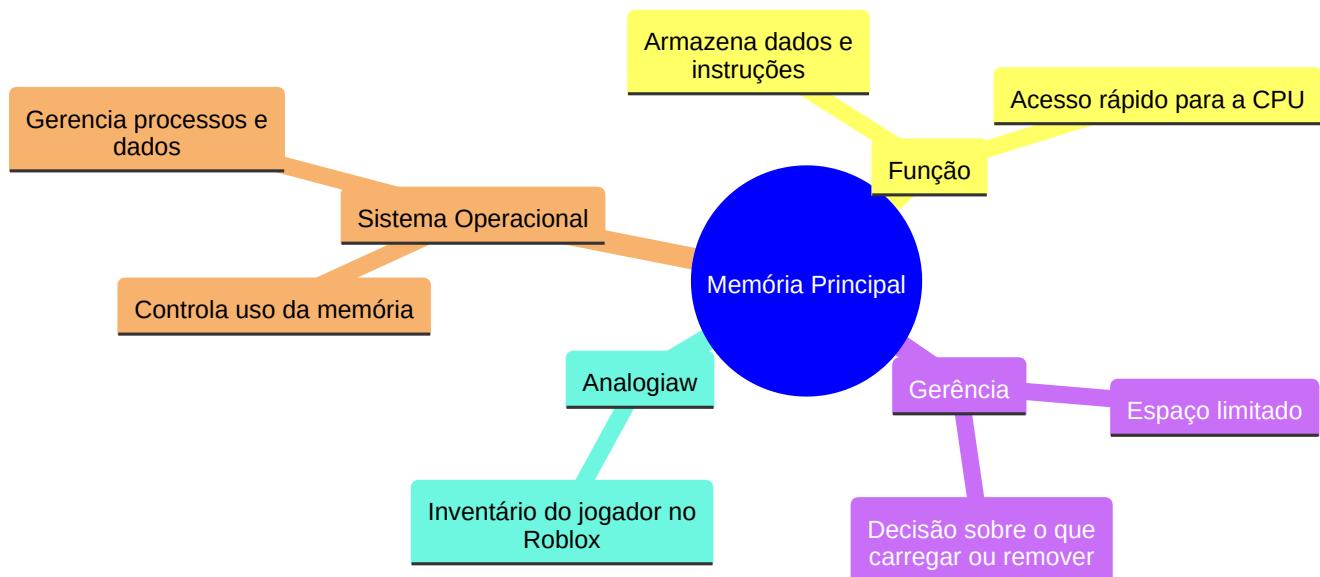
3. Gerência de Memória:

- O sistema operacional (administrador) gerencia o espaço na memória, decidindo quais programas (itens) ficam na memória e quais são removidos quando o espaço acaba.
- Isso é crucial para manter vários programas rodando ao mesmo tempo, como ter vários itens no inventário para diferentes tarefas.

4. Atividades do Sistema Operacional:

- Controlar quais partes da memória estão em uso e por quem.
- Decidir quais processos (tarefas) e dados devem ser carregados ou removidos da memória.

Resumo visual:



1.10 Gerência de armazenamento



O sistema operacional fornece uma visão lógica e uniforme do armazenamento de informações, abstraindo as propriedades físicas dos dispositivos de armazenamento. Ele define uma unidade de armazenamento lógica chamada **arquivo**, que é mapeada no meio físico e acessada por dispositivos de armazenamento.

Analogia com Roblox: Imagine o Roblox como um sistema operacional. Ele gerencia todos os itens, skins, mapas e scripts que você usa nos jogos. Esses itens são como "arquivos" que o Roblox organiza e torna acessíveis para você, independentemente de onde eles estejam armazenados fisicamente (servidores, nuvem, etc.).

1.10.1 Gerência de Sistema de Arquivos

A gerência de arquivos é uma parte visível do sistema operacional, responsável por organizar e controlar o acesso a arquivos e diretórios. Os arquivos podem ser de vários tipos (texto, binários, etc.) e são armazenados em diferentes mídias (discos magnéticos, ópticos, fitas). O sistema operacional gerencia a criação, remoção, organização e acesso a esses arquivos.

Analogia com Roblox: No Roblox, você tem uma "pasta" de inventário onde todos os seus itens (arquivos) são organizados. Alguns itens são raros (como arquivos importantes), outros são comuns (como arquivos de texto). O Roblox também controla quem pode acessar seus itens (leitura, escrita, remoção), assim como um sistema operacional faz com arquivos.

1.10.2 Gerência de Armazenamento em Massa

Como a memória principal é limitada e volátil, o armazenamento secundário (como discos) é essencial para guardar programas e dados. O sistema operacional gerencia o espaço livre, a

alocação de armazenamento e o escalonamento do disco para garantir eficiência. Além disso, há o armazenamento terciário (como fitas e CDs), usado para backups e dados raramente acessados.

Analogia com Roblox: Pense no armazenamento secundário como o seu "inventário principal" no Roblox, onde você guarda os itens que usa com frequência. Já o armazenamento terciário seria como um "baú de tesouro" onde você guarda itens raros ou que não usa muito (como skins antigas ou itens de eventos passados). O Roblox gerencia esses espaços para que você possa acessá-los quando precisar.

1.10.3 Caching

O **caching** é um conceito essencial para entender como os sistemas computadorizados otimizam o acesso a informações. Ele funciona como uma camada intermediária de armazenamento rápido, reduzindo o tempo de acesso a dados frequentemente utilizados.

Como funciona:

1. Armazenamento de Informações:

- As informações são armazenadas em dispositivos como a memória principal.
- Quando acessadas, são copiadas temporariamente para uma memória mais rápida, chamada **cache**.

2. Busca de Dados:

- Ao buscar uma informação, o sistema primeiro verifica se ela está no cache.
 - Se estiver (**cache hit**), os dados são usados diretamente do cache.
 - Se não estiver (**cache miss**), o sistema busca a informação na memória principal (ou secundária) e a copia para o cache, acelerando futuros acessos.

3. Registradores e Algoritmos:

- Registradores (como os de índice) são gerenciados por algoritmos que decidem quais dados manter no cache e quais enviar para a memória principal.
- Esses algoritmos são implementados por programadores, compiladores ou diretamente no hardware.

4. Cache de Instruções:

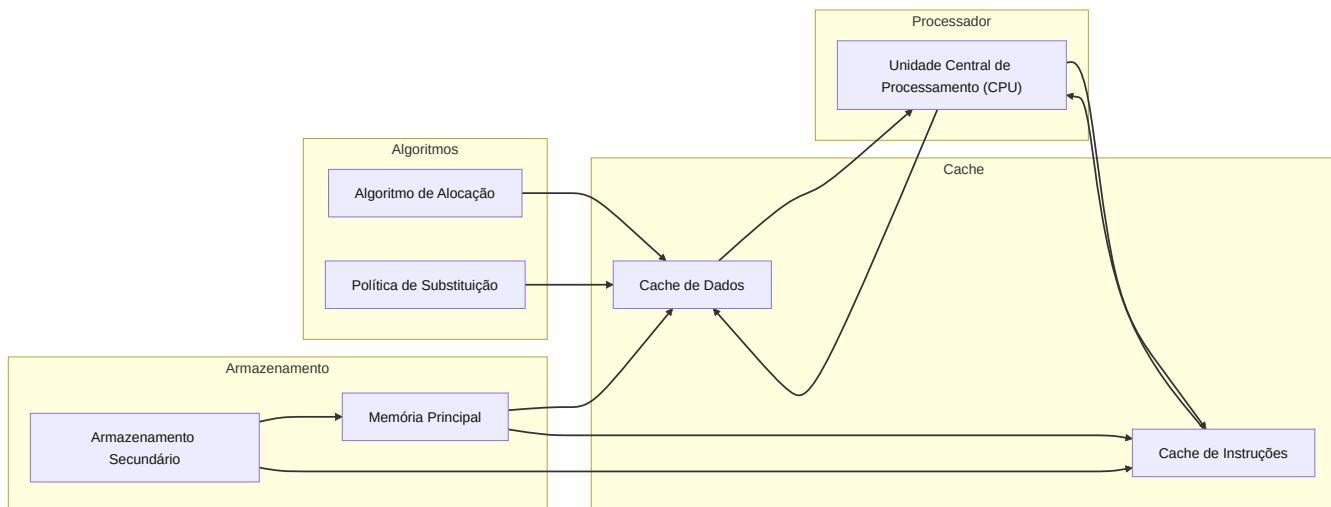
- Muitos sistemas possuem um **cache de instruções**, que armazena as próximas instruções a serem executadas pela CPU.
- Isso evita que a CPU perca ciclos buscando instruções na memória principal.

5. Hierarquia de Memórias:

- O cache está no topo da hierarquia de memórias, sendo a mais rápida, porém com capacidade limitada.
- Abaixo dele estão a memória principal e o armazenamento secundário (discos, SSDs).

6. Gerenciamento de Cache:

- Como o cache tem tamanho reduzido, seu gerenciamento é crucial. Isso inclui:
 - Definir o **tamanho do cache**.
 - Estabelecer a **política de substituição** (ex.: LRU - Least Recently Used) para decidir quais dados remover quando o cache estiver cheio.



Esses fatores podem melhorar o desempenho da memória cache.

A **memória principal** pode ser vista como um **cache rápido para o armazenamento secundário**, pois os dados precisam ser copiados da memória secundária para a principal antes de serem utilizados.

De forma recíproca, para serem **movidos para a memória secundária**, os dados **precisam estar primeiro na memória principal**, garantindo proteção e integridade.

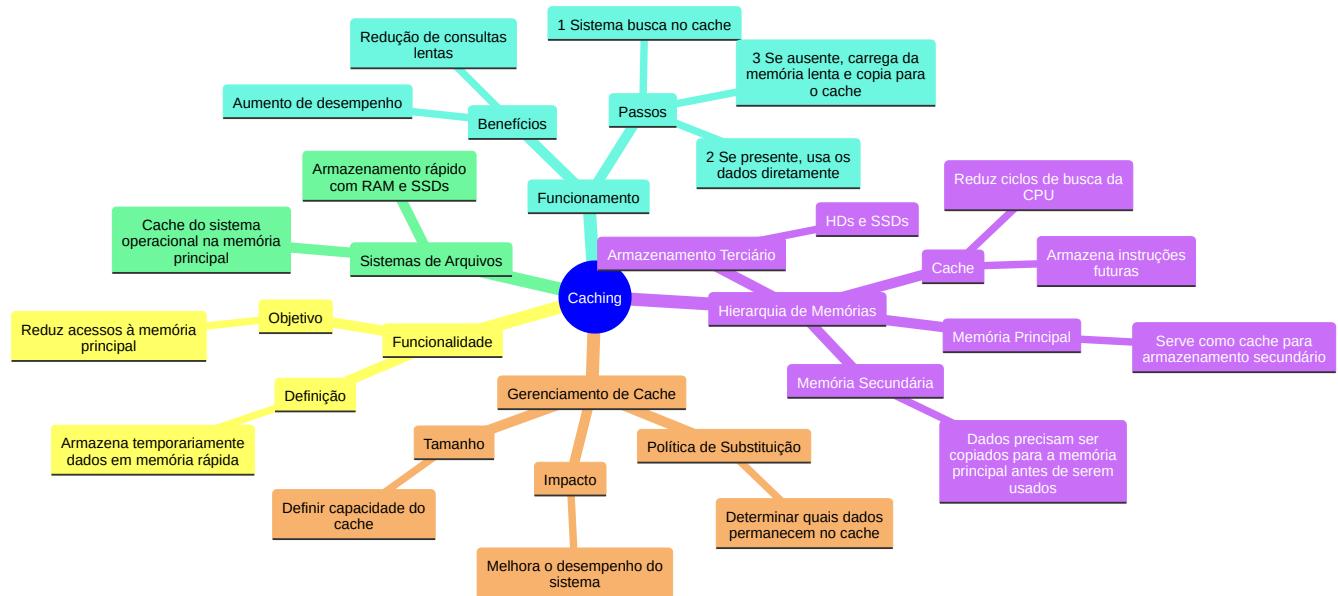
O sistema de arquivos vê os dados permanentemente gravados no armazenamento secundário de forma hierárquica, existindo *diversos níveis na hierarquia*:

- No nível mais alto -> o **sistema operacional** pode manter um **cache do sistema de arquivos na memória principal**.

Também é possível que memórias RAM, como discos de estado sólido (ou então discos eletrônicos de RAM), sejam usadas para **armazenamento de alta velocidade**, acessados pela **interface do**

sistema de arquivos. Isso significa que a comunicação deve ser feita diretamente com o sistema de arquivos.

Atualmente, a maior parte do **armazenamento terciário** consiste em **HDs ou SSDs**.



Níveis e o Cache

Os **movimentos** de informações entre os **níveis da hierarquia de memórias** podem ser de dois tipos: **explícitos** e **implícitos**. Isso depende da arquitetura do **hardware** e do **software** que controla o sistema operacional.

Podemos exemplificar essa questão:

- A transferência de dados entre a cache e a CPU e seus registradores-> ocorre diretamente no **hardware**, sem intervenção do sistema operacional.
- A transferência de dados do disco para a memória RAM-> normalmente é controlada pelo **sistema operacional**.

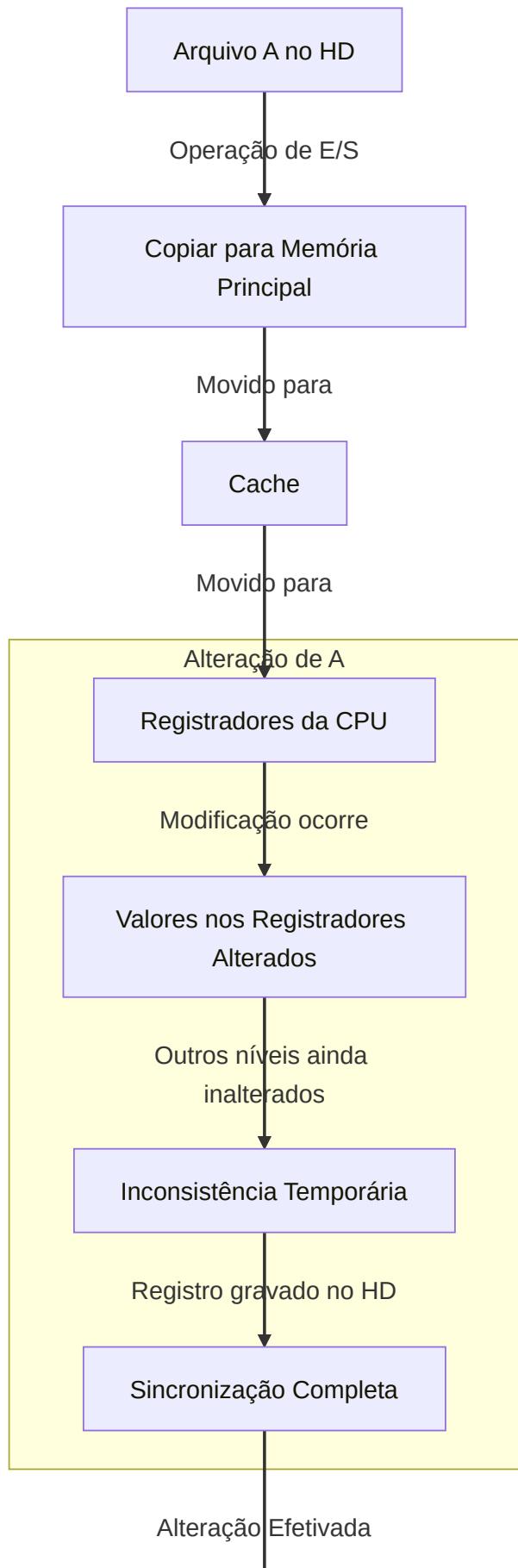
Como, nessa estrutura hierárquica, os mesmos dados podem aparecer em diferentes níveis de armazenamento, vejamos um exemplo:

- Suponha que um texto no arquivo A precise ser alterado para um outro valor no arquivo B, que reside no HD.
- Antes da alteração, o sistema precisa emitir uma operação de E/S para copiar o bloco de disco contendo A para a memória principal.
- Em seguida, o arquivo A será copiado para o cache e para os registradores internos da CPU.

- Assim, a **cópia de A** estará presente em vários níveis, conforme mostrado abaixo:

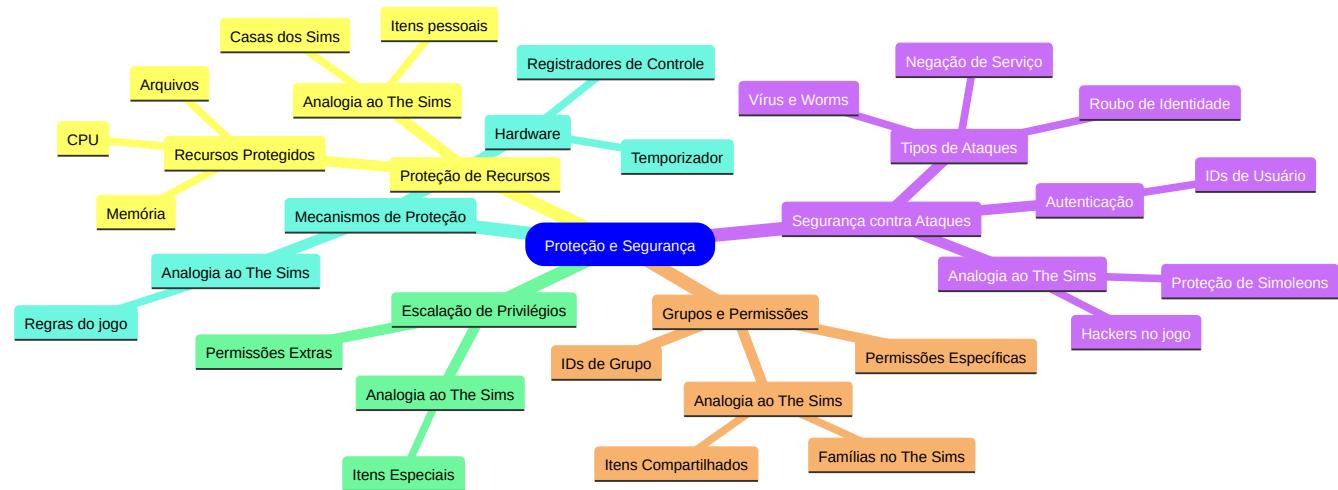


- Quando a alteração for feita nos registradores internos da CPU, os valores de **A** serão diferentes nos outros níveis de armazenamento, que permanecerão inalterados.
- Somente quando o **registraror gravar a mudança no disco rígido** (memória secundária), os valores nos diferentes níveis estarão **sincronizados**, tornando a alteração efetiva.



Valores Iguais em Todos os
Níveis

1.11 Proteção e segurança



1. Proteção de Recursos:

- Em um sistema com múltiplos usuários e processos, o acesso aos recursos (arquivos, memória, CPU) precisa ser controlado. O sistema operacional garante que apenas processos autorizados possam acessar esses recursos.
- Analogia ao The Sims:** Imagine que cada Sim (usuário) tem sua própria casa (espaço de memória) e itens (recursos). O jogo impede que um Sim entre na casa de outro ou use seus itens sem permissão.

2. Mecanismos de Proteção:

- O hardware e o sistema operacional trabalham juntos para proteger recursos. Por exemplo, o temporizador impede que um processo monopolize a CPU, e os registradores de controle de dispositivo protegem periféricos.
- Analogia ao The Sims:** No jogo, há regras que impedem que um Sim fique indefinidamente em uma atividade (como cozinhar ou dormir), garantindo que outros Sims também tenham acesso aos recursos.

3. Segurança contra Ataques:

- A segurança protege o sistema contra ataques externos e internos, como vírus, roubo de identidade e negação de serviço. A autenticação (IDs de usuário) é usada para garantir que apenas usuários autorizados acessem o sistema.
- Analogia ao The Sims:** Imagine que um "hacker" tenta invadir o jogo e roubar os Simoleons (moeda do jogo) de um Sim. O sistema de segurança do jogo (como senhas ou autenticação em dois fatores) impede isso.

4. Grupos e Permissões:

- Os sistemas operacionais usam IDs de usuário e grupo para controlar permissões. Um usuário pode pertencer a um ou mais grupos, e cada grupo tem permissões específicas.
- **Analogia ao The Sims:** No jogo, você pode criar famílias (grupos) e definir quais Sims têm permissão para usar certos itens ou áreas da casa.

5. Escalação de Privilégios:

- Às vezes, um usuário precisa de permissões extras para realizar tarefas específicas. O sistema operacional permite essa escalação de forma controlada.
- **Analogia ao The Sims:** Se um Sim precisa usar um item especial (como um objeto de magia), ele pode ganhar permissões temporárias para acessá-lo.

1.12 Sistemas de uso específico

1.11 Sistemas de Uso Específico

Os sistemas computadorizados de uso específico são projetados para tarefas especializadas e limitadas, diferindo dos sistemas de uso geral que estamos acostumados a utilizar. Eles são amplamente empregados em dispositivos embutidos, como eletrodomésticos inteligentes, carros autônomos, drones e dispositivos IoT (Internet das Coisas).

1.11.1 Sistemas de Tempo Real Embutidos

- **O que são?** Sistemas embutidos são computadores dedicados a tarefas específicas, como controlar motores de carros, robôs industriais, drones ou até mesmo dispositivos domésticos inteligentes, como assistentes virtuais (Alexa, Google Home) e termostatos (Nest). Eles operam em **tempo real**, o que significa que precisam responder a eventos dentro de um tempo definido, ou o sistema falha.
- **Analogia ao Minecraft:** Imagine um **redstone circuit** no Minecraft. Ele é projetado para realizar uma tarefa específica, como abrir uma porta automaticamente quando um jogador se aproxima. Se o circuito não responder imediatamente, a funcionalidade falha. Assim como um sistema de tempo real, o circuito de redstone tem "restrições de tempo" para funcionar corretamente.
- **Exemplos Modernos:**
 - Carros autônomos (como um **redstone contraption** que controla um veículo automático no Minecraft).
 - Drones (como um **dispenser** que lança foguetes no tempo exato).
 - Dispositivos IoT (como um **sensor de movimento** no Minecraft que acende luzes automaticamente).

1.11.2 Sistemas Multimídia

- **O que são?** Sistemas multimídia lidam com dados como áudio, vídeo e realidade aumentada (AR), que precisam ser entregues em "streaming" com restrições de tempo (ex.: 60 frames por segundo para jogos ou vídeos 4K). Eles são usados em aplicações como videoconferências (Zoom, Teams), streaming (Netflix, YouTube) e realidade virtual (VR).
- **Analogia ao Minecraft:** Pense em um **mapa de aventura** no Minecraft com cutscenes (cenas pré-gravadas). Para que a experiência seja imersiva, as cenas precisam ser exibidas sem atrasos,

assim como um vídeo precisa ser reproduzido sem travamentos. Se o sistema não conseguir entregar os frames no tempo certo, a experiência é prejudicada.

- **Exemplos Modernos:**

- Streaming de jogos (como o **Minecraft RTX**, que exige alta performance gráfica).
- Realidade virtual (como um **mundo VR** no Minecraft).
- Videoconferências (como um **evento ao vivo** no servidor de Minecraft com transmissão em tempo real).

1.11.3 Sistemas Portáteis

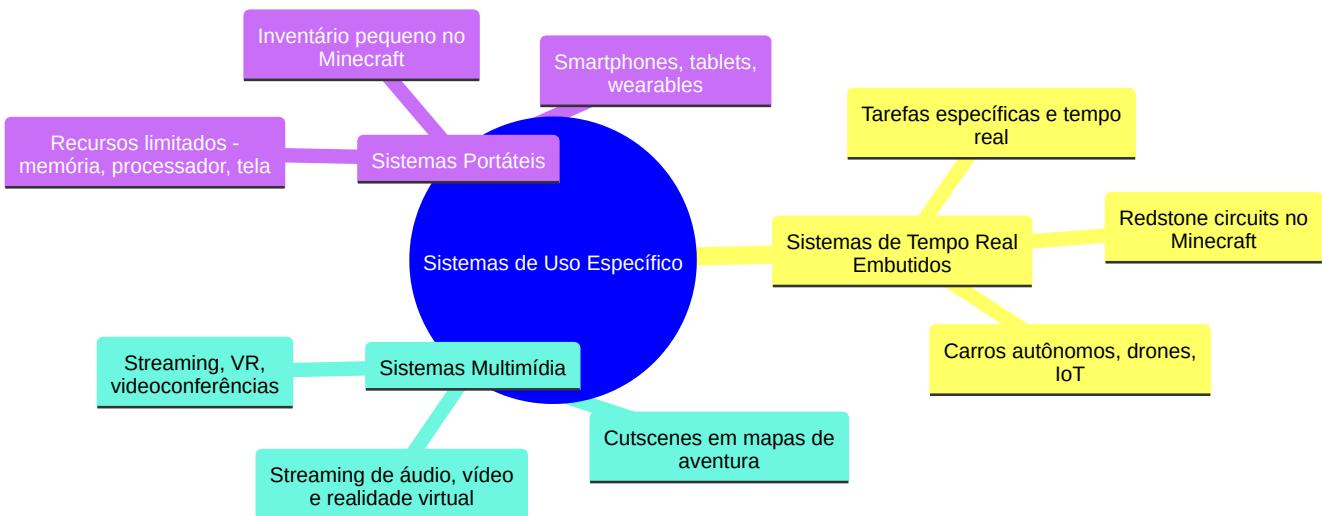
- **O que são?** Sistemas portáteis, como smartphones, tablets e wearables (smartwatches), têm recursos limitados devido ao seu tamanho reduzido. Eles possuem pouca memória, processadores eficientes (mas não tão potentes quanto desktops) e telas pequenas, mas são altamente convenientes e portáteis.
- **Analogia ao Minecraft:** Imagine um **inventário de Minecraft**. Ele tem espaço limitado, então você precisa gerenciar os itens com cuidado, priorizando o que é mais importante. Assim como um smartphone, o inventário é pequeno, mas essencial para a jogabilidade.

- **Desafios Modernos:**

- Memória limitada (como um **baú pequeno** no Minecraft, mas com otimizações para armazenar mais itens).
- Processadores eficientes (como jogar Minecraft no **celular** com gráficos reduzidos para evitar lag).
- Telas pequenas (como a interface compacta do **Minecraft Pocket Edition**).

- **Tecnologias Atuais:**

- Smartphones com 5G (como um **servidor de Minecraft** com conexão ultrarrápida).
- Wearables (como um **smartwatch** que monitora sua saúde enquanto você joga).
- Tablets (como jogar Minecraft em um **iPad** com tela maior e portabilidade).



1.13 Ambientes de Computação

1.12.1 Computação Tradicional

- **O que é?** A computação tradicional refere-se ao uso de PCs, servidores e mainframes em ambientes como escritórios e residências. Antigamente, os sistemas eram centralizados, com terminais conectados a mainframes ou PCs ligados a redes locais. Hoje, a computação tradicional se expandiu com o uso de tecnologias web, dispositivos portáteis e conexões de alta velocidade.
- **Evolução:**
 - **Antes:** Sistemas em lote (batch) e interativos, com tempo compartilhado para otimizar recursos.
 - **Hoje:** PCs potentes, laptops, tablets e smartphones com acesso remoto e portabilidade.
 - **Tendências:** Portais web, sincronização de dispositivos e redes domésticas inteligentes.
- **Exemplos Modernos:**
 - **Escritórios:** Uso de laptops, desktops e servidores em nuvem (como Google Workspace ou Microsoft 365).
 - **Residências:** Redes domésticas com dispositivos IoT (smart TVs, assistentes virtuais) e conexões de alta velocidade (fibra óptica, 5G).

1.12.2 Sistemas Cliente-Servidor

- **O que é?** Neste modelo, os sistemas são divididos em dois papéis:
 - **Cliente:** Solicita serviços (ex.: navegador web).
 - **Servidor:** Fornece serviços (ex.: servidor de arquivos ou banco de dados).
- **Tipos de Servidores:**
 - **Servidor de Processamento (Compute-Server):** Executa ações e retorna resultados (ex.: servidor de banco de dados).
 - **Servidor de Arquivos (File-Server):** Gerencia arquivos e os disponibiliza para clientes (ex.: servidor web).

- **Vantagens:**

- Centralização de recursos e dados.
- Facilidade de gerenciamento e segurança.

- **Exemplos Modernos:**

- Serviços em nuvem (AWS, Google Cloud).
- Aplicações web (Netflix, Spotify).

1.12.3 Sistemas Peer-to-Peer (P2P)

- **O que é?** No modelo P2P, todos os nós (dispositivos) na rede são iguais, podendo atuar como clientes e servidores. Não há centralização, e os serviços são distribuídos entre os nós.

- **Funcionamento:**

- **Descoberta de Serviços:**

- **Centralizada:** Um servidor central mantém um índice de serviços (ex.: Napster).
- **Descentralizada:** Os nós enviam requisições por broadcast (ex.: Gnutella).

- **Vantagens:**

- Eliminação de gargalos (não há um único servidor).
- Escalabilidade e resiliência.

- **Exemplos Modernos:**

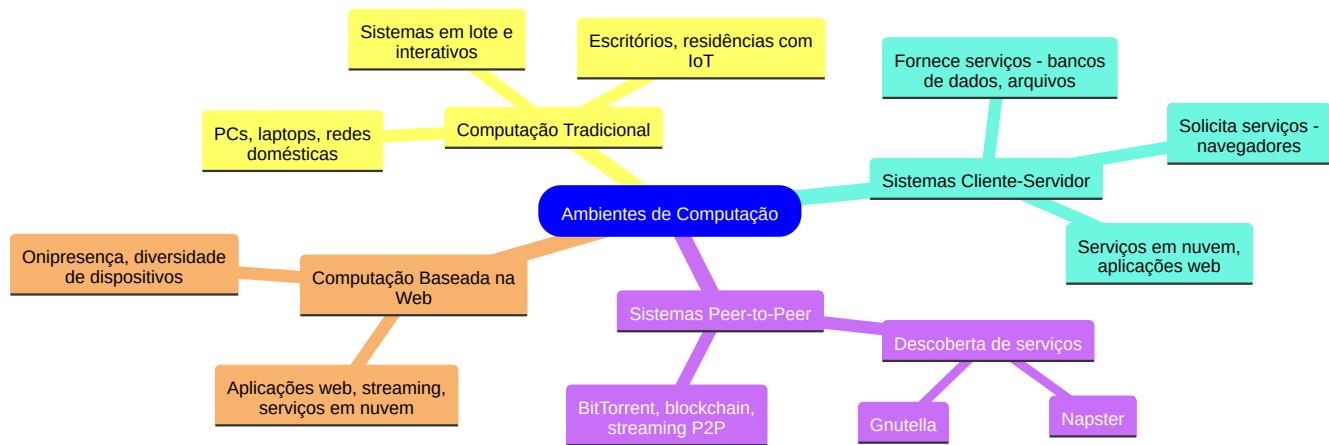
- Compartilhamento de arquivos (BitTorrent).
- Criptomoedas (blockchain, Bitcoin).
- Streaming P2P (ex.: plataformas de vídeo descentralizadas).

1.12.4 Computação Baseada na Web

- **O que é?** A computação baseada na web transformou a forma como acessamos e utilizamos recursos computacionais. Ela permite o acesso a serviços e dados por meio de navegadores e dispositivos conectados à internet.

- **Características:**

- **Onipresença:** Acesso de qualquer lugar, a qualquer hora.
- **Diversidade de Dispositivos:** PCs, smartphones, tablets, IoT.
- **Conectividade:** Redes sem fio (Wi-Fi, 5G) e balanceadores de carga para distribuição de tráfego.
- **Exemplos Modernos:**
 - Aplicações web (Google Docs, Figma).
 - Plataformas de streaming (YouTube, Twitch).
 - Serviços em nuvem (Dropbox, iCloud).



1.14 Sistemas Operacionais de Código Aberto

1.13.1 Benefícios dos Sistemas de Código Aberto

- **Transparência e Flexibilidade:** O acesso ao código-fonte permite que programadores e estudantes entendam como o sistema funciona, modifiquem o código e criem versões personalizadas.
- **Aprendizado Prático:** Estudantes podem modificar o código, compilar e testar suas alterações, o que é uma excelente ferramenta educacional.
- **Comunidade Ativa:** Uma grande comunidade de desenvolvedores contribui para o código, ajudando a identificar e corrigir bugs rapidamente. Isso torna o software mais seguro e confiável.
- **Modelos de Negócios:** Empresas como Red Hat e SUSE mostram que é possível gerar receita com software de código aberto, oferecendo suporte técnico, serviços personalizados e hardware compatível.

1.13.2 História dos Sistemas de Código Aberto

- **Origens:** Nos anos 1950 e 1960, o software era frequentemente compartilhado livremente entre entusiastas e grupos de usuários. A cultura de compartilhamento de código era comum.
- **Restrições Comerciais:** Com o crescimento da indústria de software, empresas começaram a proteger seus códigos-fonte, distribuindo apenas binários compilados para evitar cópias não autorizadas.
- **Movimento de Software Livre:** Em 1983, Richard Stallman iniciou o projeto **GNU** para criar um sistema operacional livre e compatível com UNIX. Ele fundou a **Free Software Foundation (FSF)** e criou a **Licença Pública Geral (GPL)**, que exige que o código-fonte seja compartilhado junto com qualquer distribuição do software.

1.13.3 Linux

- **Origem:** Criado em 1991 por Linus Torvalds, o Linux é um kernel de código aberto que, combinado com ferramentas GNU, forma o sistema operacional **GNU/Linux**.

- **Distribuições:** Existem centenas de distribuições Linux, como **Ubuntu**, **Fedora**, **Debian** e **Red Hat**, cada uma com foco em diferentes usuários (desktop, servidores, gamers, etc.).
- **Acesso ao Código-Fonte:** O código-fonte do Linux pode ser baixado e modificado por qualquer pessoa. Ferramentas como o **VMware Player** permitem testar distribuições Linux em máquinas virtuais.

1.13.4 BSD UNIX

- **História:** Derivado do UNIX da AT&T, o **BSD UNIX** foi desenvolvido na Universidade da Califórnia em Berkeley. Em 1994, uma versão totalmente funcional e de código aberto, a **4.4BSD-lite**, foi lançada.
- **Distribuições:** Incluem **FreeBSD**, **NetBSD**, **OpenBSD** e **DragonFly BSD**, cada uma com foco em diferentes aspectos, como segurança, portabilidade e desempenho.
- **Influência no macOS:** O kernel do macOS, chamado **Darwin**, é baseado no BSD e também é de código aberto.

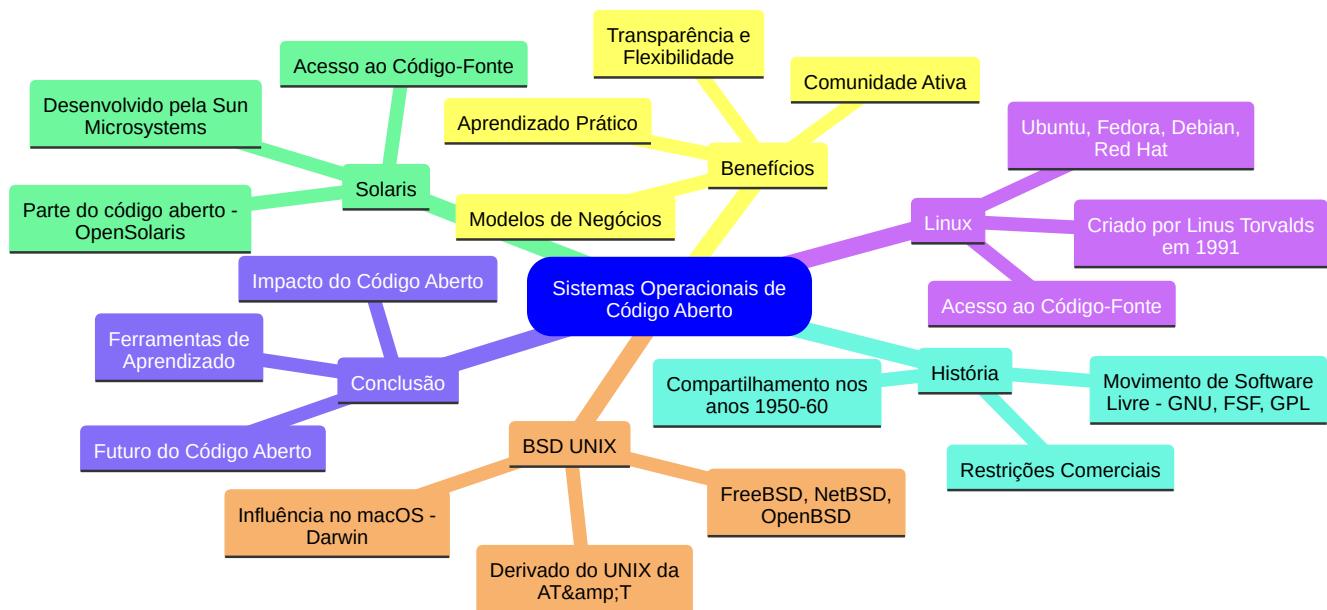
1.13.5 Solaris

- **Origem:** Desenvolvido pela Sun Microsystems, o **Solaris** é um sistema operacional baseado no UNIX. Em 2005, a Sun abriu parte do código-fonte do Solaris, criando o projeto **OpenSolaris**.
- **Características:** Embora nem todo o Solaris seja de código aberto (devido a componentes proprietários), grande parte do sistema pode ser explorada e modificada.
- **Acesso ao Código-Fonte:** O código-fonte está disponível no site opensolaris.org, onde também é possível explorar o código online.

1.13.6 Conclusão

- **Impacto do Código Aberto:** O movimento de software livre e código aberto tem impulsionado a inovação, permitindo que milhares de desenvolvedores colaborem em projetos como Linux, BSD e Solaris.
- **Ferramentas de Aprendizado:** O acesso ao código-fonte de sistemas operacionais maduros, como Linux e BSD, é uma ferramenta valiosa para estudantes e profissionais que desejam entender e contribuir para o desenvolvimento de software.
- **Futuro:** A tendência é que mais empresas e indivíduos adotem projetos de código aberto,

impulsionados pela transparência, segurança e colaboração que esse modelo oferece.



Exercícios Práticos Resolvidos

1.1. Quais são as três principais finalidades de um sistema operacional?

1. **Gerenciamento de recursos:** Controlar e alocar hardware (CPU, memória, dispositivos de E/S) para programas.
2. **Facilitar a execução de programas:** Fornecer um ambiente para que os programas sejam executados de forma eficiente.
3. **Proteger o sistema:** Garantir que programas e usuários não interfiram uns com os outros ou com o sistema.

1.2. Quais são as principais diferenças entre os sistemas operacionais para computadores mainframe e computadores pessoais?

- **Mainframe:**
 - Focado em alta confiabilidade, disponibilidade e processamento de grandes volumes de dados.
 - Suporta milhares de usuários simultaneamente.
 - Exemplos: IBM z/OS, Linux on IBM Z.
- **Computadores pessoais:**
 - Focado em interatividade e usabilidade para um único usuário.
 - Suporta aplicações como navegadores, editores de texto e jogos.
 - Exemplos: Windows, macOS, Linux.

1.3. Relacione as quatro etapas que são necessárias para executar um programa em uma máquina completamente dedicada – um computador que esteja executando apenas esse programa.

1. **Carregar o programa na memória:** Transferir o código do programa do disco para a memória

RAM.

2. **Configurar o contador de programa:** Definir o endereço inicial do programa para a CPU começar a executá-lo.
3. **Executar o programa:** A CPU executa as instruções do programa.
4. **Finalizar o programa:** Encerrar a execução e liberar os recursos usados.

1.4. Quando é apropriado que o sistema operacional abra mão da eficiência e “desperdice” recursos?

- **Resposta:** Em sistemas interativos ou de tempo real, onde a experiência do usuário é prioridade (ex.: animações suaves, respostas rápidas).
- **Por que não é desperdício?:** O "desperdício" de recursos pode melhorar a usabilidade e a satisfação do usuário, o que é valioso em muitos contextos.

1.5. Qual é a principal dificuldade que um programador deverá contornar na escrita de um sistema operacional para um ambiente de tempo real?

- **Resposta:** Garantir que o sistema atenda a prazos rígidos (deadlines) para execução de tarefas, sem atrasos.
- **Explicação:** Em sistemas de tempo real, a previsibilidade e a resposta rápida são essenciais, o que exige algoritmos de escalonamento e gerenciamento de recursos altamente otimizados.

1.6. O sistema operacional deverá incluir aplicações como navegadores Web e programas de e-mail?

- **Argumento a favor:**
 - Facilita a usabilidade, pois o usuário já tem ferramentas essenciais instaladas.
 - Integração mais profunda com o sistema operacional.
- **Argumento contra:**
 - Aumenta o tamanho e a complexidade do sistema operacional.
 - Limita a escolha do usuário, que pode preferir outras aplicações.

1.7. Como a distinção entre o modo kernel e o modo usuário pode funcionar como uma forma rudimentar de sistema de proteção (segurança)?

- **Resposta:** O modo kernel tem acesso total ao hardware, enquanto o modo usuário tem acesso restrito. Isso impede que programas de usuário realizem operações perigosas, como acessar diretamente o hardware ou modificar áreas críticas do sistema.

1.8. Quais das seguintes instruções deverão ser privilegiadas?

- Privilegiadas:
 - a. Definir o valor do temporizador.
 - c. Apagar a memória.
 - e. Desativar interrupções.
 - f. Modificar entradas na tabela de status de dispositivo.
 - g. Passar do modo usuário para o modo kernel.
 - h. Acessar dispositivo de E/S.
- Não privilegiadas:
 - b. Ler o valor do relógio.
 - d. Emitir uma instrução de trap.

1.9. Duas dificuldades de proteger o sistema operacional em uma partição de memória imutável

1. **Falta de flexibilidade:** Dificulta atualizações e correções no sistema operacional.
2. **Ineficiência:** Pode limitar o uso de técnicas avançadas de gerenciamento de memória, como memória virtual.

1.10. Dois usos possíveis para múltiplos modos de operação em CPUs

1. **Virtualização:** Um modo adicional para executar máquinas virtuais.
2. **Segurança:** Modos intermediários para controle de acesso a recursos específicos.

1.11. Como temporizadores poderiam ser usados para calcular a hora atual?

- **Resposta:** Um temporizador pode ser configurado para gerar interrupções em intervalos regulares (ex.: 1 segundo). Cada interrupção incrementa um contador que representa a hora atual.
- **Explicação:** O sistema operacional usa o contador para manter o relógio do sistema atualizado.

1.12. A Internet é uma LAN ou uma WAN?

- **Resposta:** A Internet é uma WAN (Wide Area Network), pois conecta redes e dispositivos em escala global, ao contrário de uma LAN (Local Area Network), que é limitada a uma área geográfica pequena, como uma casa ou escritório.

Capítulo 2

2.1 Serviços do sistema operacional

Os serviços do sistema operacional usando analogias simples do Minecraft:

1. Interface do Usuário (UI)

- No Minecraft, você pode jogar de várias formas: no modo criativo (GUI, com menus e cliques), no modo sobrevivência (linha de comando, digitando comandos) ou com mods pré-configurados (interface batch, arquivos de comandos).
- O sistema operacional também oferece diferentes interfaces para você interagir com ele, seja por cliques, comandos ou scripts.

2. Execução de Programas

- No Minecraft, você coloca blocos e cria estruturas (programas) para fazer coisas acontecerem. O sistema operacional é como o mundo do Minecraft: ele carrega e executa os programas, permitindo que eles funcionem e, se necessário, os interrompe se algo der errado.

3. Operações de Entrada/Saída (E/S)

- No Minecraft, você interage com o mundo usando ferramentas (teclado, mouse) e dispositivos como portais ou baús (arquivos e periféricos). O sistema operacional gerencia isso, garantindo que você não "quebre" o jogo ao tentar acessar algo diretamente.

4. Manipulação de Arquivos

- No Minecraft, você organiza seus itens em baús (arquivos) e pastas (diretórios). O sistema operacional faz o mesmo, permitindo criar, ler, escrever e excluir arquivos, além de controlar quem pode acessá-los.

5. Comunicação

- No Minecraft, você pode jogar com amigos no mesmo mundo (memória compartilhada) ou em servidores diferentes (rede). O sistema operacional facilita a comunicação entre programas, seja no mesmo computador ou em redes.

6. Detecção de Erros

- No Minecraft, se você tentar colocar um bloco onde não pode, o jogo avisa. O sistema operacional faz o mesmo, detectando erros de hardware, software ou permissões e corrigindo ou alertando sobre eles.

7. Alocação de Recursos

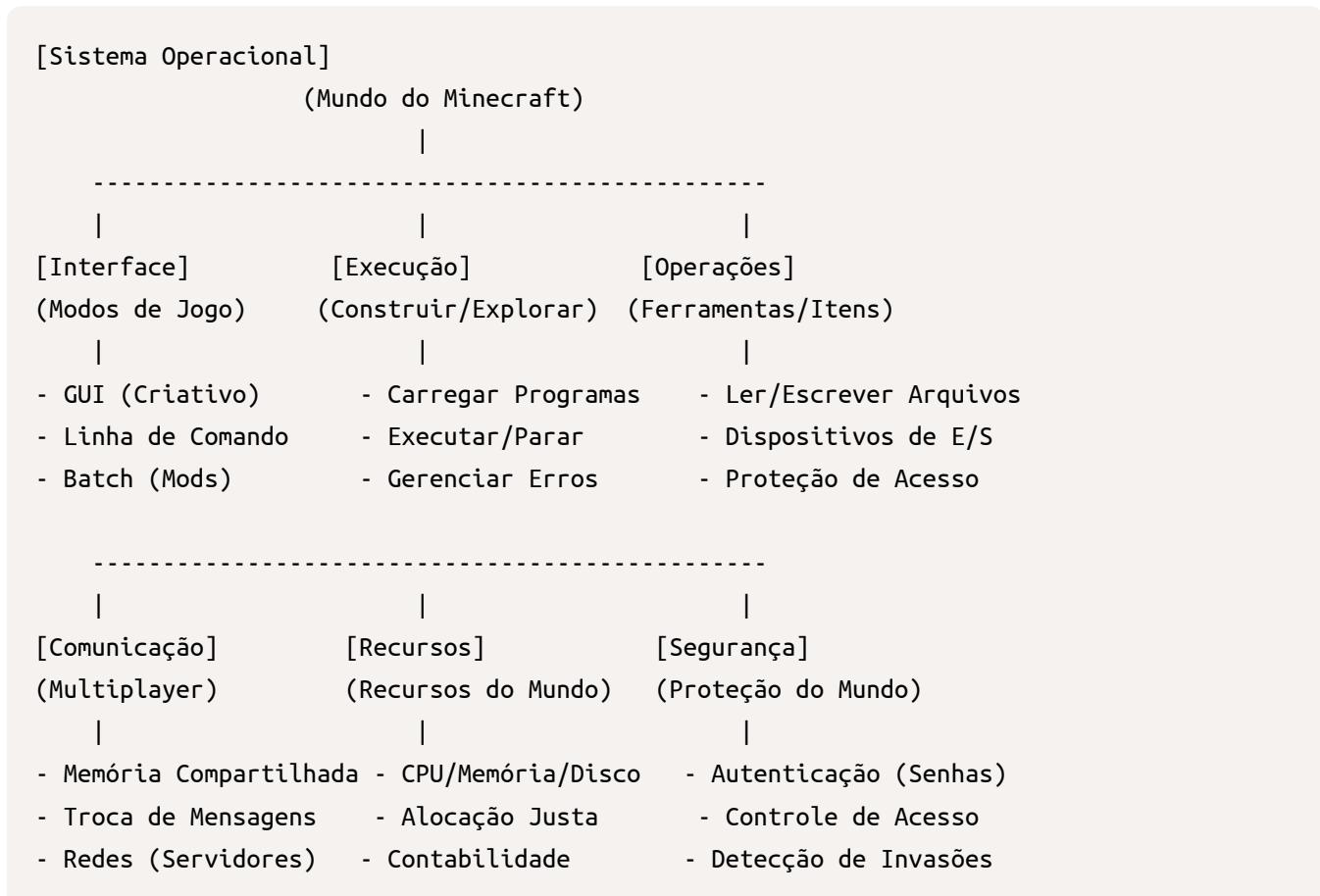
- No Minecraft, recursos como madeira, minérios e tempo são limitados. O sistema operacional gerencia recursos como memória, CPU e dispositivos, distribuindo-os de forma justa entre os programas.

8. Contabilidade

- No Minecraft, você pode ver quanto de cada recurso coletou. O sistema operacional registra o uso de recursos para cobrança ou análise, como um "log" de atividades.

9. Proteção e Segurança

- No Minecraft, você protege seu mundo com senhas ou modos de jogo. O sistema operacional faz o mesmo, garantindo que apenas usuários autorizados acessem recursos e protegendo o sistema contra invasões.



2.2 Interface usuário-sistema operacional

1. Interpretador de Comandos (CLI - Command Line Interface)

- **O que é:** Uma interface baseada em texto onde o usuário digita comandos diretamente.
- **Funcionamento:**
 - O interpretador (ou *shell*) captura e executa os comandos.
 - Exemplos: Bourne shell, C shell, Bash (Linux/UNIX), Prompt de Comando (Windows).
- **Implementação:**
 - **Método 1:** O próprio interpretador contém o código para executar os comandos (ex.: comandos internos).
 - **Método 2:** Comandos são programas externos (ex.: `rm` no UNIX), onde o interpretador apenas localiza e executa o arquivo correspondente.
- **Vantagens:**
 - Poderoso e flexível para tarefas avançadas.
 - Permite automação via scripts.

2. Interface Gráfica com o Usuário (GUI - Graphical User Interface)

- **O que é:** Uma interface visual com janelas, ícones, menus e mouse.
- **Funcionamento:**
 - O usuário interage clicando em ícones, arrastando arquivos ou selecionando opções em menus.
 - Exemplos: Windows Explorer, Aqua (Mac OS X), GNOME/KDE (Linux).
- **Histórico:**
 - Surgiu na década de 1970 (Xerox PARC).

- Popularizada pelo Macintosh (1980) e Windows (1990).

- **Vantagens:**

- Mais intuitiva e acessível para usuários comuns.
- Facilita a organização de arquivos e execução de programas.

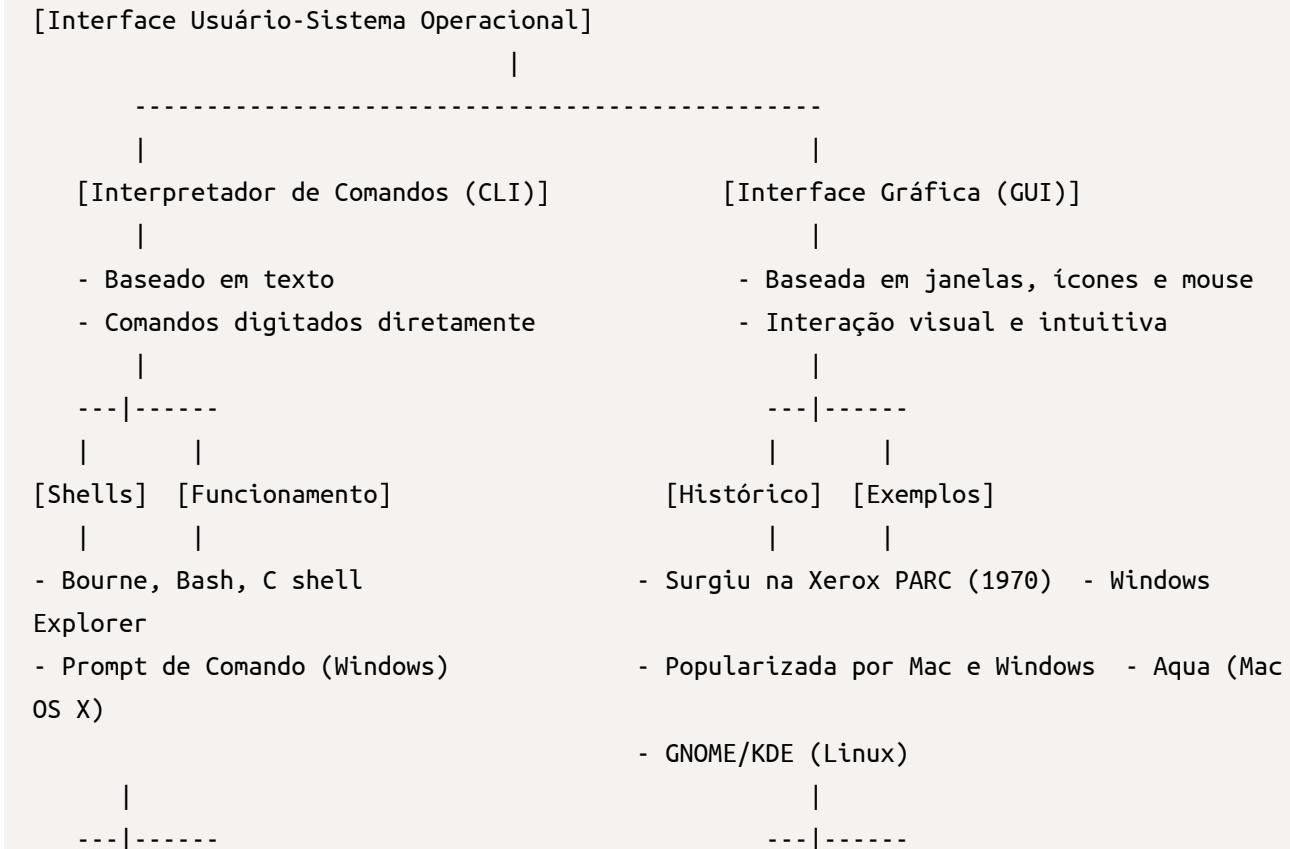
Comparação e Preferências

- **CLI vs GUI:**

- **CLI:** Preferido por usuários avançados (ex.: programadores, administradores de sistemas) por sua eficiência e controle.
- **GUI:** Preferido pela maioria dos usuários por ser mais amigável e visual.

- **Exemplos:**

- UNIX/Linux: Tradicionalmente CLI, mas oferece GUIs como GNOME e KDE.
- Windows e Mac: Focam em GUIs, mas também possuem CLIs (Prompt de Comando no Windows, Terminal no Mac).



[Vantagens]	[Implementação]	[Vantagens]	[Preferências]
- Poderoso e flexível		- Mais acessível e intuitiva	- Usuários comuns
- Permite automação (scripts)	técnica	- Facilita organização e execução	- Menos
- Ideal para tarefas avançadas		- Foco em usabilidade	

|
[Comparação CLI vs GUI]

- |
- CLI: Preferido por técnicos e programadores
- GUI: Preferido pela maioria dos usuários
- Ambos coexistem para atender diferentes necessidades

2.3 Chamadas de sistema

As **chamadas de sistema** são a interface entre os programas e os serviços oferecidos pelo sistema operacional. Elas permitem que programas solicitem operações como leitura/escrita de arquivos, gerenciamento de memória e comunicação com dispositivos. Aqui está um resumo organizado:

1. O que são Chamadas de Sistema?

- **Definição:** São rotinas que permitem que programas solicitem serviços do sistema operacional.
- **Implementação:** Escritas em linguagens como C/C++ ou assembly (para tarefas de baixo nível).
- **Exemplo:** Um programa que lê dados de um arquivo e os copia para outro usa várias chamadas de sistema:
 - Solicitar nomes dos arquivos (E/S).
 - Abrir arquivos.
 - Ler e escrever dados.
 - Tratar erros (arquivo inexistente, falta de espaço no disco, etc.).
 - Fechar arquivos e finalizar o programa.

2. Como Funcionam?

- **Sequência de Chamadas:**
 1. Solicitar nomes dos arquivos (E/S interativa ou via GUI).
 2. Abrir arquivo de entrada e criar arquivo de saída.
 3. Ler dados do arquivo de entrada e escrever no arquivo de saída.
 4. Tratar erros durante a leitura/escrita.
 5. Fechar arquivos e finalizar o programa.
- **Exemplo de Chamadas:**
 - `open()`: Abrir um arquivo.
 - `read()`: Ler dados de um arquivo.

- `write()`: Escrever dados em um arquivo.
- `close()`: Fechar um arquivo.

3. APIs e Chamadas de Sistema

- **API (Interface de Programação de Aplicação):**
 - Conjunto de funções que simplificam o uso de chamadas de sistema.
 - Exemplos: API Win32 (Windows), API POSIX (UNIX/Linux/Mac), API Java.
- **Vantagens:**
 - Portabilidade: Programas podem rodar em sistemas com a mesma API.
 - Facilidade: APIs são mais simples de usar do que chamadas de sistema diretas.
- **Relacionamento:**
 - Funções da API (ex.: `CreateProcess()` no Windows) chamam funções do sistema operacional (ex.: `NTCreateProcess()`).
 - O sistema operacional executa a operação e retorna o resultado.

4. Passagem de Parâmetros

- **Métodos:**
 1. **Registradores:** Parâmetros são passados diretamente nos registradores da CPU.
 2. **Bloco/Tabela:** Parâmetros são armazenados em memória, e o endereço do bloco é passado em um registrador.
 3. **Pilha:** Parâmetros são empilhados (push) e desempilhados (pop) pela CPU.
- **Exemplo:** No Linux, parâmetros são passados como uma tabela na memória.

5. Chamadas de Sistema em Java

- **Java Native Interface (JNI):**
 - Permite que métodos Java chamem funções nativas escritas em C/C++.
 - Essas funções podem invocar chamadas de sistema específicas do sistema operacional.

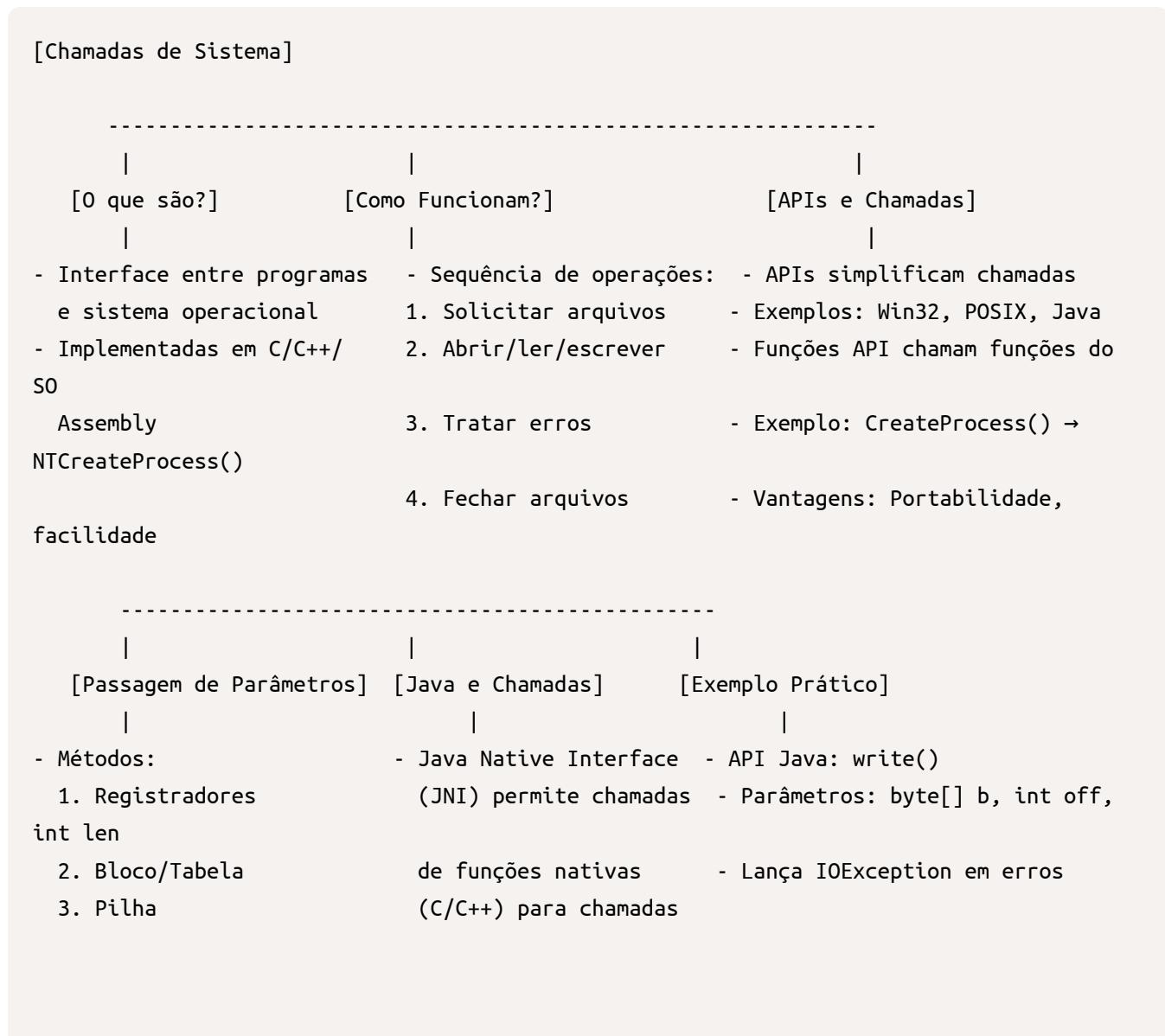
- **Limitação:** Programas que usam JNI perdem portabilidade entre plataformas.

6. Exemplo Prático

- API Java:

- Método `write()` da classe `java.io.OutputStream`:
 - Escreve dados em um arquivo ou conexão de rede.
 - Parâmetros: `byte[] b` (dados), `int off` (offset), `int len` (número de bytes).
 - Lança `IOException` em caso de erro.

Diagrama



de sistema

- Perde portabilidade

2.4 Tipos de chamadas de sistema

1. Controle de Processos

- **Função:** Gerenciar a execução de programas (processos).
- **Exemplos de Chamadas:**
 - Criação/Término: fork(), create process(), exit(), abort().
 - Controle: wait(), signal(), get/set process attributes().
 - Sincronização: acquire lock(), release lock().
- **Casos de Uso:**
 - Iniciar, pausar ou finalizar processos.
 - Esperar por eventos ou processos filhos.
 - Gerenciar concorrência e compartilhamento de recursos.

2. Manipulação de Arquivos

- **Função:** Criar, ler, escrever e gerenciar arquivos e diretórios.
- **Exemplos de Chamadas:**
 - Abertura/Fechamento: open(), close().
 - Leitura/Escrita: read(), write().
 - Atributos: get file attributes(), set file attributes().
- **Casos de Uso:**
 - Criar, excluir ou renomear arquivos.
 - Ler e escrever dados em arquivos.
 - Gerenciar permissões e atributos de arquivos.

3. Manipulação de Dispositivos

- **Função:** Gerenciar dispositivos de hardware (físicos ou virtuais).
- **Exemplos de Chamadas:**
 - **Acesso:** `read()`, `write()`, `ioctl()`.
 - **Alocação:** `request device()`, `release device()`.
- **Casos de Uso:**
 - Ler/escrever em dispositivos como impressoras ou discos.
 - Controlar dispositivos com operações específicas (ex.: ajustar resolução de tela).

4. Manutenção de Informações

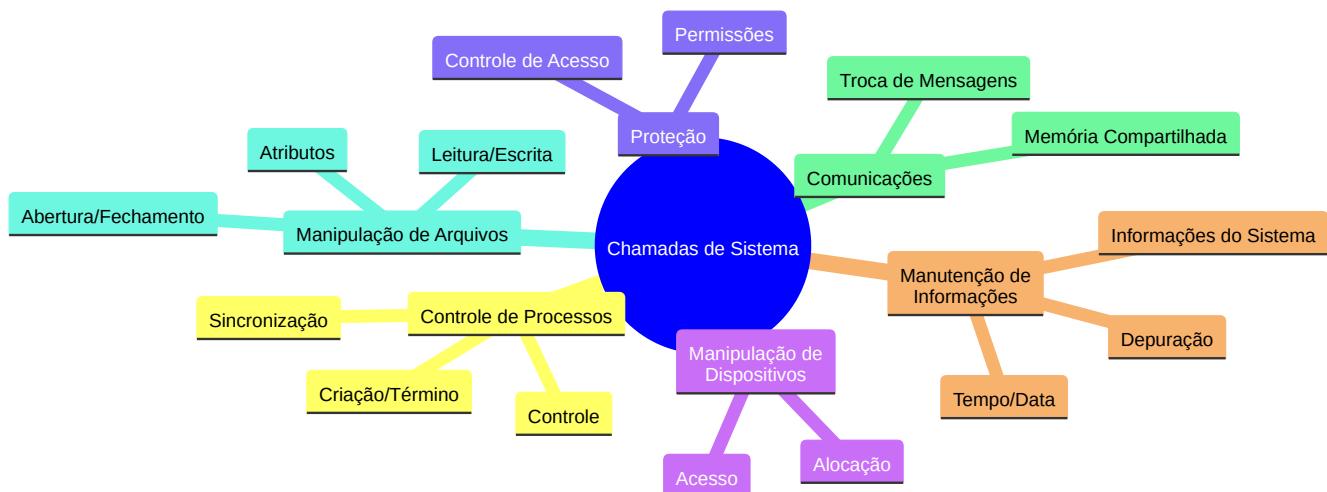
- **Função:** Obter e definir informações do sistema e do usuário.
- **Exemplos de Chamadas:**
 - **Tempo/Data:** `get time()`, `set time()`.
 - **Informações do Sistema:** `get system info()`, `get process info()`.
 - **Depuração:** `dump memory()`, `trace()`.
- **Casos de Uso:**
 - Obter informações como uso de memória, número de usuários ou versão do sistema.
 - Depurar programas com ferramentas como dump de memória ou perfil de tempo.

5. Comunicações

- **Função:** Facilitar a comunicação entre processos (no mesmo computador ou em rede).
- **Modelos:**
 - **Troca de Mensagens:** `send message()`, `receive message()`.
 - **Memória Compartilhada:** `shared memory create()`, `shared memory attach()`.
- **Casos de Uso:**
 - Trocar mensagens entre processos (ex.: cliente-servidor).
 - Compartilhar memória para comunicação rápida entre processos.

6. Proteção

- **Função:** Controlar o acesso a recursos do sistema.
- **Exemplos de Chamadas:**
 - **Permissões:** set permission(), get permission().
 - **Controle de Acesso:** allow user(), deny user().
- **Casos de Uso:**
 - Definir permissões de acesso a arquivos, dispositivos ou processos.
 - Proteger o sistema contra acessos não autorizados.



2.5 Programas do sistema

Resumo:

Os **programas do sistema** (ou utilitários) são ferramentas incluídas no sistema operacional para facilitar o desenvolvimento, execução e gerenciamento de programas. Eles se dividem em categorias como:

1. **Gerência de Arquivos:** Criar, remover, copiar, renomear e manipular arquivos/diretórios.
2. **Informações de Status:** Obter dados como hora, uso de memória, espaço em disco e logs de desempenho.
3. **Modificação de Arquivos:** Editores de texto e ferramentas para buscar/transformar conteúdo.
4. **Suporte para Linguagem de Programação:** Compiladores, interpretadores e depuradores.
5. **Carga e Execução de Programas:** Carregadores e sistemas de depuração para executar programas.
6. **Comunicações:** Ferramentas para conexões remotas, transferência de arquivos e mensagens.
7. **Programas de Aplicação:** Navegadores, editores de texto, planilhas, jogos, etc.

Além disso, a experiência do usuário é definida pelos programas de aplicação e interfaces (GUI ou CLI), que podem variar mesmo no mesmo hardware (ex.: dual-booting entre Mac OS X e Windows).



2.6 Projeto e implementação do sistema operacional

O **projeto e implementação de sistemas operacionais** envolvem desafios complexos, como definir objetivos, separar políticas de mecanismos, escolher linguagens de programação e estruturar o sistema de forma eficiente. Aqui estão os principais pontos:

1. Objetivos de Projeto

- **Objetivos do Usuário:** Conveniência, facilidade de uso, confiabilidade, segurança e velocidade.
- **Objetivos do Sistema:** Facilidade de projeto, implementação, manutenção, flexibilidade e eficiência.
- **Desafio:** Não há uma solução única; os requisitos variam conforme o tipo de sistema (batch, tempo real, multiusuário, etc.).

2. Mecanismos e Políticas

- **Mecanismo:** Como algo é feito (ex.: temporizador para proteção da CPU).
- **Política:** O que deve ser feito (ex.: tempo alocado para cada usuário).
- **Separação:** Mantém o sistema flexível, permitindo mudanças de políticas sem alterar mecanismos.

3. Implementação

- **Linguagens:** Sistemas operacionais modernos são escritos em linguagens de alto nível (ex.: C, C++), com trechos em assembly para otimização.
- **Vantagens:** Código mais rápido de escrever, compacto, portável e fácil de depurar.
- **Desvantagens:** Potencial redução de desempenho, mas compensada por otimizações de compiladores modernos.

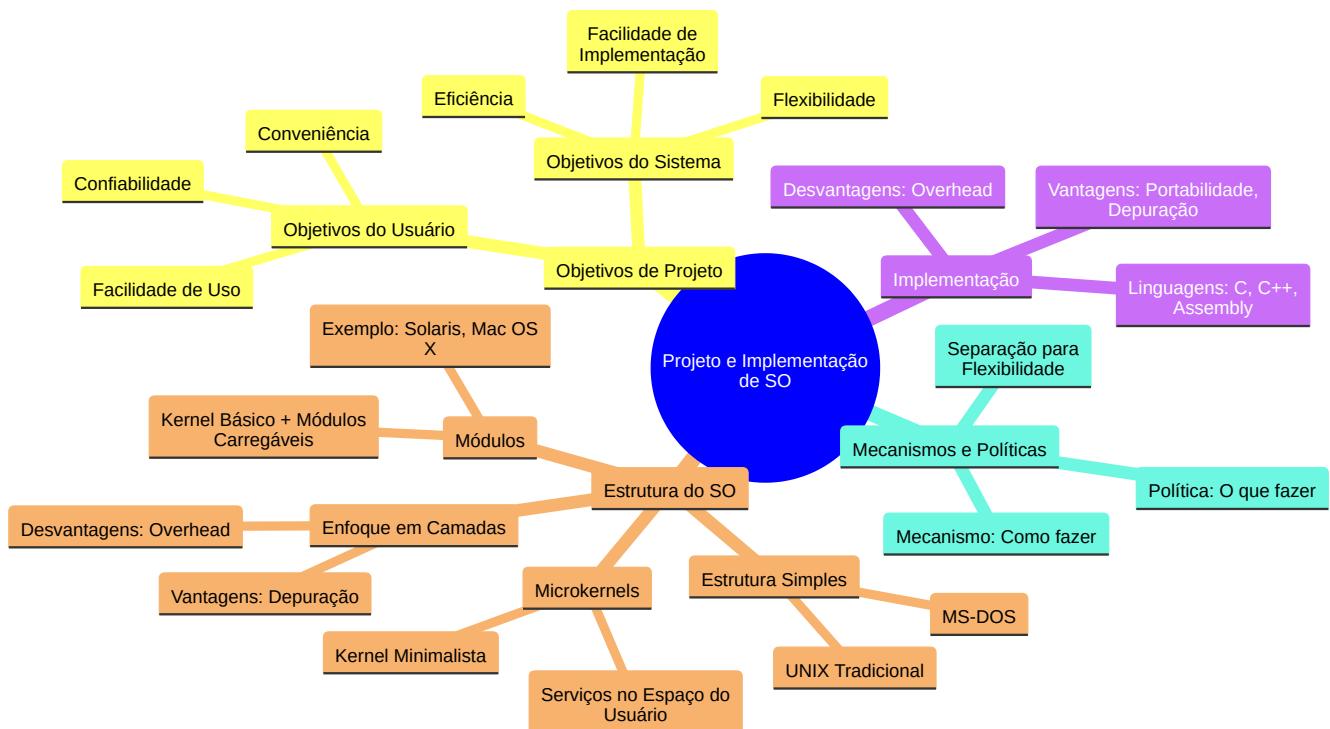
4. Estrutura do Sistema Operacional

- **Estrutura Simples:** Sistemas como MS-DOS e UNIX inicial tinham designs monolíticos, com pouca separação de componentes.
- **Enfoque em Camadas:** Divide o sistema em níveis, facilitando depuração e manutenção, mas pode adicionar overhead.
- **Microkernels:** Kernel minimalista, com serviços essenciais (gerência de processos, memória e comunicação). Serviços adicionais rodam no espaço do usuário, aumentando segurança e modularidade.
- **Módulos:** Combina vantagens de camadas e microkernels. O kernel básico carrega módulos dinamicamente (ex.: drivers, sistemas de arquivos), oferecendo flexibilidade e eficiência.

5. Exemplos de Estruturas

- **MS-DOS:** Monolítico, sem proteção de hardware.
- **UNIX Tradicional:** Kernel grande e monolítico, difícil de manter.
- **Solaris:** Usa módulos carregáveis para sistemas de arquivos, drivers e escalonamento.
- **Mac OS X:** Híbrido, com microkernel Mach e componentes BSD para redes, sistemas de arquivos e threads.

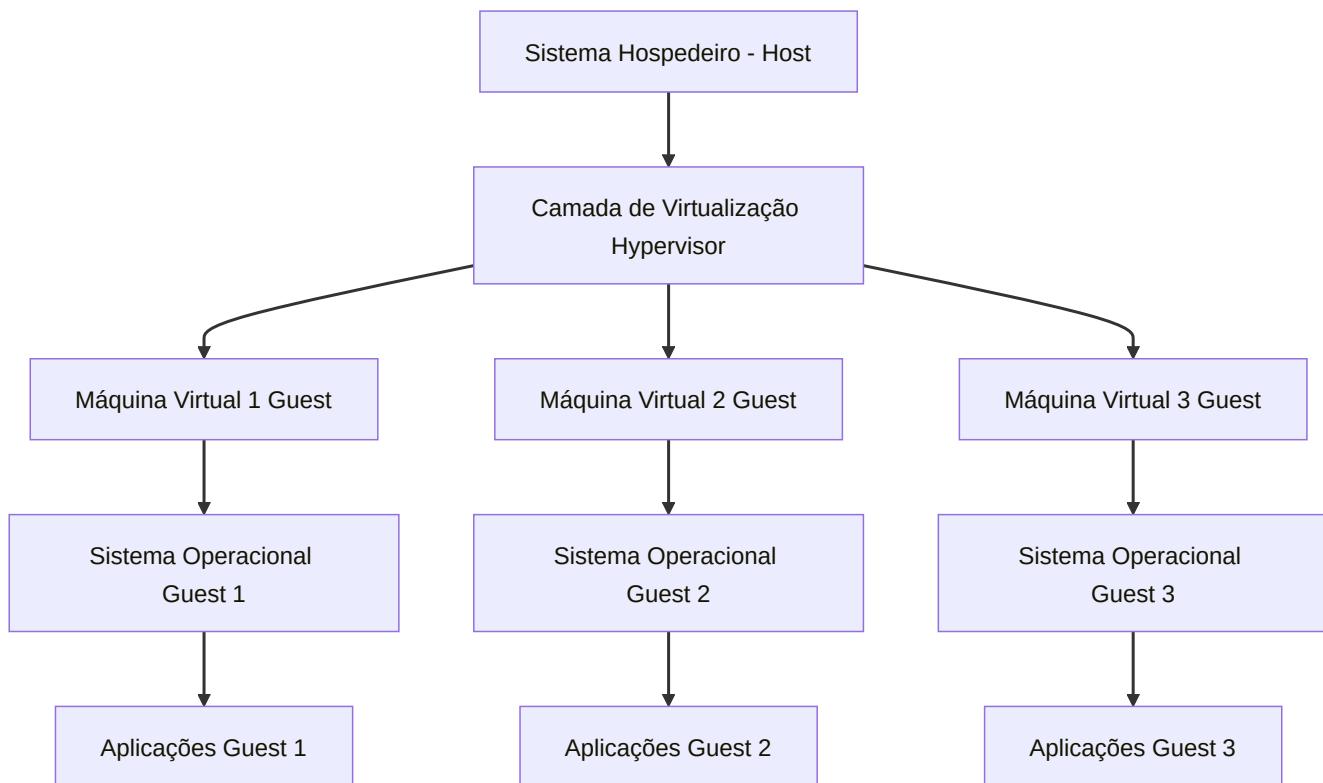
Mindmap em Mermaid:



2.7 Máquinas virtuais

Resumo:

As **máquinas virtuais (VMs)** são ambientes isolados que simulam um computador completo, permitindo a execução de múltiplos sistemas operacionais simultaneamente em um único hardware. Aqui estão os principais pontos:



1. Sistema Hospedeiro (Host):

- É o sistema físico que contém o hardware real (CPU, memória, disco, etc.).
- Roda o sistema operacional principal (ex.: Linux, Windows).

2. Camada de Virtualização (Hypervisor):

- É o software que gerencia as máquinas virtuais.
- Pode ser do Tipo 1 (executa diretamente no hardware) ou Tipo 2 (executa como uma aplicação no sistema hospedeiro).

3. Máquinas Virtuais (Guests):

- São ambientes isolados que simulam um computador completo.

- Cada máquina virtual tem seu próprio sistema operacional e aplicações.

4. Sistemas Operacionais Guests:

- Sistemas operacionais rodando dentro das máquinas virtuais (ex.: Windows, Linux, macOS).

5. Aplicações Guests:

- Programas que rodam dentro dos sistemas operacionais guests.

1. Conceito de Máquinas Virtuais

- **Definição:** Separação do hardware em múltiplos ambientes de execução, cada um com seu próprio sistema operacional.
- **Funcionamento:** Usa técnicas de escalonamento de CPU e memória virtual para criar a ilusão de um computador dedicado para cada VM.
- **Exemplo:** Um sistema físico pode rodar Windows, Linux e macOS simultaneamente como VMs.

2. Benefícios das Máquinas Virtuais

- **Isolamento:** Protege o sistema hospedeiro e outras VMs de falhas ou vírus.
- **Desenvolvimento e Testes:** Permite testar sistemas operacionais e aplicações em ambientes isolados sem afetar o sistema principal.
- **Consolidação de Sistemas:** Reduz custos ao executar múltiplos sistemas em um único hardware.
- **Portabilidade:** Facilita a migração de aplicações entre sistemas.

3. Implementação de Máquinas Virtuais

- **Desafios:** Simular o hardware completo, incluindo modos de operação (usuário e kernel).
- **Técnicas:**
 - **Modo Usuário Virtual:** Simula o modo usuário dentro do modo usuário físico.
 - **Modo Kernel Virtual:** Simula o modo kernel dentro do modo usuário físico.
- **Supporte de Hardware:** CPUs modernas (ex.: Intel VT-x, AMD-V) facilitam a virtualização com modos hospedeiro e guest.

4. VMware

- **Funcionamento:** Executa como uma aplicação no sistema hospedeiro, criando VMs independentes.
- **Exemplo:** Um sistema Linux pode rodar FreeBSD, Windows NT e Windows XP como VMs.
- **Vantagens:** Facilita a cópia, movimentação e gerenciamento de sistemas guest.

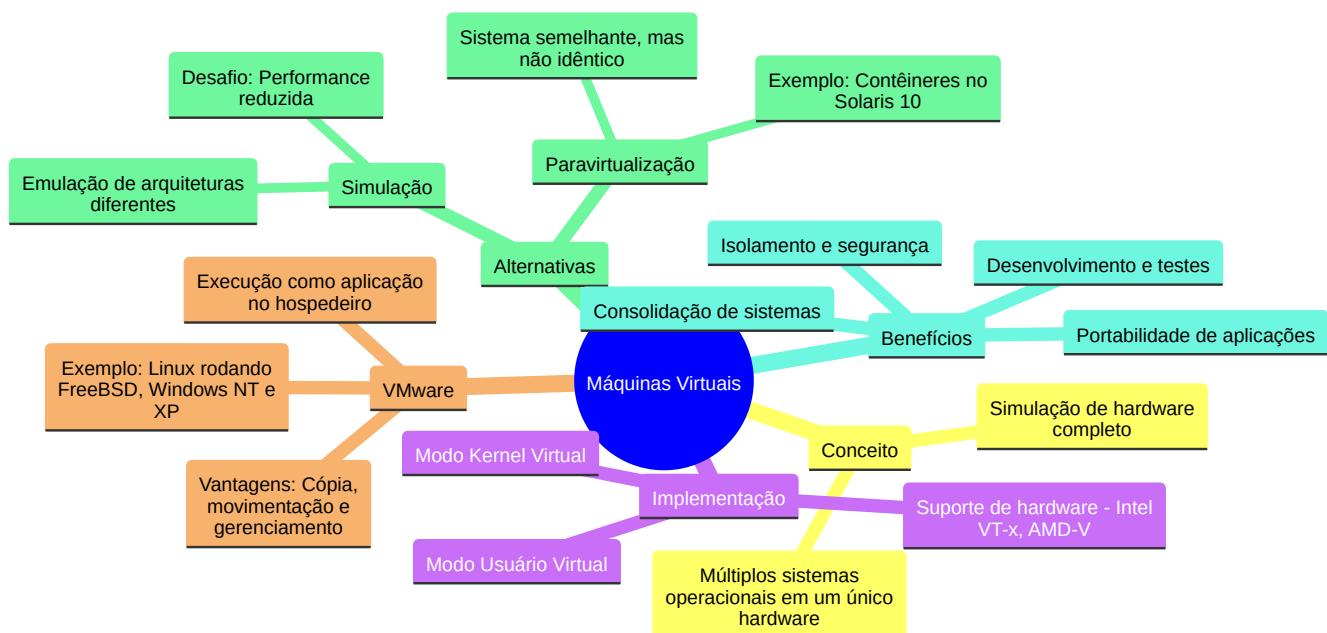
5. Alternativas à Virtualização

- **Simulação:**

- **Definição:** Emula uma arquitetura de hardware diferente da do sistema hospedeiro.
- **Uso:** Executar programas antigos em hardware moderno.
- **Desafio:** Performance reduzida, pois cada instrução é traduzida.

- **Paravirtualização:**

- **Definição:** Apresenta um sistema semelhante, mas não idêntico, ao hardware real.
- **Uso:** Requer modificações no sistema operacional guest, mas oferece melhor desempenho.
- **Exemplo:** Contêineres no Solaris 10, que virtualizam o sistema operacional, não o hardware.



2.8 Geração do sistema operacional

A **geração do sistema operacional (SYSGEN)** é o processo de configurar um sistema operacional para uma máquina específica, considerando seu hardware, periféricos e necessidades do usuário. Esse processo garante que o sistema operacional funcione de forma otimizada para a configuração do computador. Aqui estão os principais pontos:

1. Objetivo da Geração do Sistema

- **Personalização:** Adaptar o sistema operacional para uma máquina específica.
- **Configuração:** Definir parâmetros como CPU, memória, dispositivos de E/S e opções do sistema.

2. Informações Necessárias para SYSGEN

- **CPU:**
 - Tipo de processador e opções instaladas (ex.: aritmética de ponto flutuante).
 - Número de CPUs em sistemas multiprocessados.
- **Memória:**
 - Quantidade de memória RAM disponível.
- **Dispositivos de E/S:**
 - Tipos de dispositivos (ex.: discos, impressoras, placas de rede).
 - Endereços de hardware, interrupções e características específicas.
- **Opções do Sistema:**
 - Tamanho de buffers, algoritmo de escalonamento, número máximo de processos, etc.

3. Métodos de Geração do Sistema

1. Compilação Personalizada:

- Modifica o código-fonte do sistema operacional com base nas informações coletadas.
- Compila o sistema operacional para gerar uma versão específica para a máquina.

- **Vantagem:** Altamente personalizado.
- **Desvantagem:** Processo lento e complexo.

2. Seleção de Módulos Pré-Compilados:

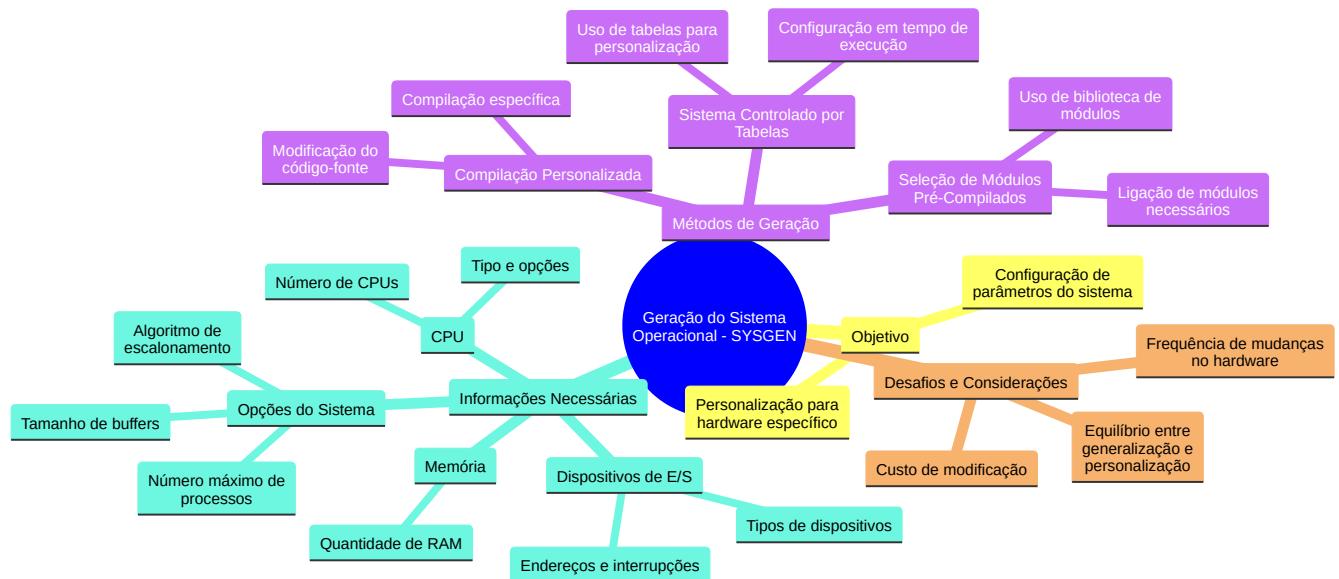
- Usa uma biblioteca de módulos pré-compilados.
- Seleciona e liga apenas os módulos necessários para a configuração.
- **Vantagem:** Mais rápido que a compilação personalizada.
- **Desvantagem:** Menos personalizado.

3. Sistema Controlado por Tabelas:

- Todo o código do sistema operacional está presente.
- A configuração é feita em tempo de execução, usando tabelas.
- **Vantagem:** Flexível e fácil de modificar.
- **Desvantagem:** Pode ser menos eficiente.

4. Desafios e Considerações

- **Frequência de Mudanças:**
 - A necessidade de reconfiguração depende da frequência com que o hardware muda.
- **Custo de Modificação:**
 - Alterar o sistema para suportar novos dispositivos pode ser caro e demorado.
- **Equilíbrio entre Generalização e Personalização:**
 - Sistemas muito genéricos podem ser menos eficientes.
 - Sistemas muito personalizados podem ser difíceis de manter.



2.9 Boot do sistema

O **boot do sistema** é o processo de inicialização do computador, que carrega o sistema operacional na memória e o prepara para execução. Com avanços tecnológicos, o processo de boot evoluiu, mas mantém os princípios básicos. Aqui estão os principais pontos atualizados:

1. Programa de Boot (Bootstrap Loader)

- **Função:** Localiza o kernel do sistema operacional, carrega-o na memória e inicia sua execução.
- **Localização:** Armazenado em firmware (UEFI/BIOS) ou em memória não volátil (como chips SPI Flash).
- **Processo:**
 - A CPU começa a execução em um endereço predefinido após o reset.
 - O programa de boot realiza diagnósticos (POST - Power-On Self-Test) e inicializa o hardware.
 - Carrega o kernel do sistema operacional na memória.

2. Tipos de Boot

- **Sistemas com Sistema Operacional em Memória Não Volátil:**
 - Usado em dispositivos embarcados, como smartphones, IoT e consoles modernos.
 - Vantagem: Simplicidade e operação reforçada.
 - Desvantagem: Dificuldade de atualização (requer reflash do firmware).
- **Sistemas com Sistema Operacional em Armazenamento (SSD/NVMe/HDD):**
 - Usado em PCs, servidores e dispositivos modernos.
 - O programa de boot (armazenado em firmware UEFI) carrega o sistema operacional do armazenamento para a memória.
 - Vantagem: Fácil atualização (basta modificar o sistema operacional no armazenamento).

3. Etapas do Boot Moderno

1. Reset da CPU: A CPU começa a execução em um endereço predefinido (definido pelo firmware UEFI/BIOS).

2. Execução do Firmware (UEFI/BIOS):

- Realiza diagnósticos do hardware (POST).
- Inicializa dispositivos básicos (memória, controladores de armazenamento, etc.).
- Localiza e executa o **bootloader** (ex.: GRUB, Windows Boot Manager).

3. Carregamento do Kernel:

- O bootloader carrega o kernel do sistema operacional na memória.
- Inicia a execução do kernel, que inicializa o sistema operacional.

4. Firmware Moderno (UEFI vs BIOS)

• **BIOS (Legacy):**

- Mais antigo, com limitações (ex.: suporte a discos de até 2 TB).
- Usa o MBR (Master Boot Record) para gerenciar o boot.

• **UEFI (Unified Extensible Firmware Interface):**

- Substituiu o BIOS na maioria dos sistemas modernos.
- Oferece suporte a discos maiores (GPT - GUID Partition Table).
- Permite boot mais rápido e seguro (Secure Boot).
- Suporta drivers e aplicativos UEFI.

5. Armazenamento de Boot

• **Disco de Boot (SSD/NVMe/HDD):**

- Contém o sistema operacional e o bootloader.
- Partição de boot (ex.: EFI System Partition no UEFI).

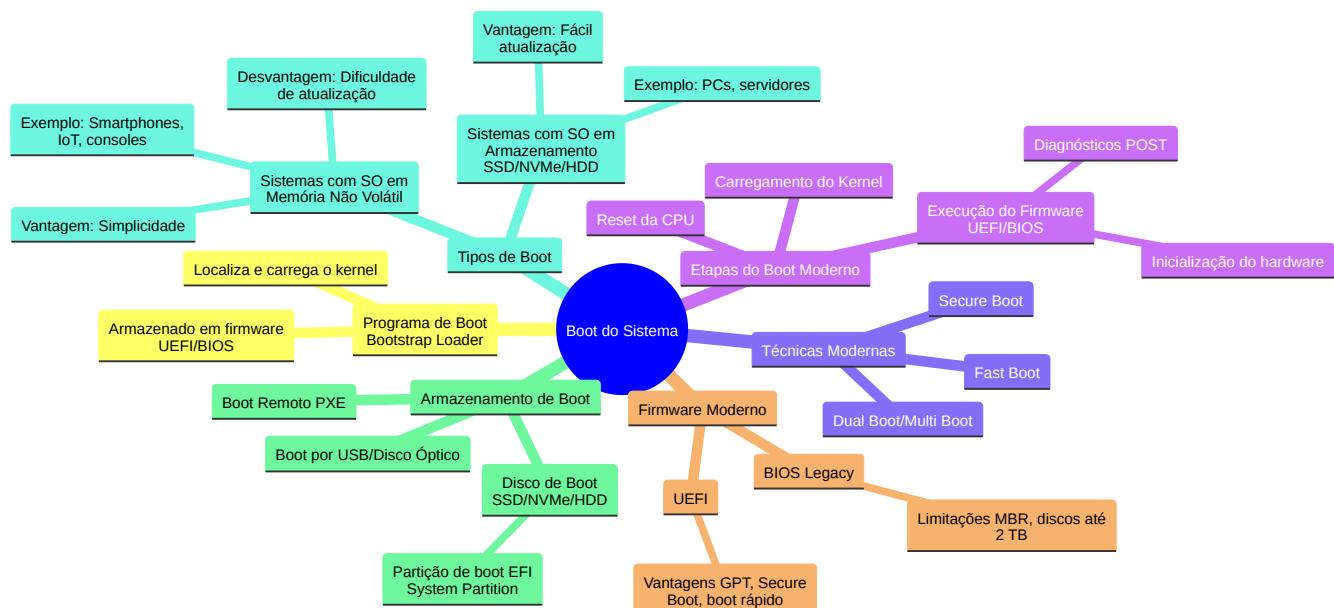
• **Boot Remoto (PXE):**

- Usado em servidores e sistemas corporativos.

- O sistema operacional é carregado pela rede.
- **Boot por USB/Disco Óptico:**
 - Usado para instalação ou recuperação de sistemas operacionais.

6. Técnicas Modernas de Boot

- **Fast Boot:**
 - Reduz o tempo de boot ao pular verificações desnecessárias.
- **Secure Boot:**
 - Verifica a integridade do bootloader e do kernel para evitar malware.
- **Dual Boot/Multi Boot:**
 - Permite a escolha entre múltiplos sistemas operacionais no boot.



Exercícios Práticos Resolvidos

2.1. Qual é o propósito das chamadas do sistema?

- **Resposta:** As chamadas do sistema (system calls) são interfaces que permitem que programas de usuário solicitem serviços ao sistema operacional. Elas atuam como uma ponte entre o software de aplicação e o hardware, permitindo que os programas realizem operações como leitura/escrita de arquivos, criação de processos, comunicação entre processos e acesso a dispositivos de hardware.
- **Explicação:** Imagine que você está escrevendo um programa e precisa ler um arquivo do disco. Em vez de acessar o disco diretamente (o que seria complexo e inseguro), você usa uma chamada de sistema como `read()`. O sistema operacional cuida de todos os detalhes de baixo nível, como acessar o hardware e garantir que o arquivo seja lido corretamente.

2.2. Quais são as cinco principais atividades de um sistema operacional em relação ao gerenciamento de processos?

- **Resposta:**
 1. **Criação e término de processos:** Criar novos processos (ex.: ao abrir um programa) e encerrá-los quando não são mais necessários.
 2. **Escalonamento de processos:** Decidir qual processo deve ser executado pela CPU em um determinado momento.
 3. **Sincronização de processos:** Garantir que processos que compartilham recursos não interfiram uns com os outros.
 4. **Comunicação entre processos:** Permitir que processos troquem informações (ex.: mensagens ou memória compartilhada).
 5. **Gerenciamento de deadlocks:** Evitar ou resolver situações em que processos ficam bloqueados esperando por recursos que nunca serão liberados.
- **Explicação:** O sistema operacional age como um "gerente" dos processos, garantindo que todos tenham acesso justo aos recursos e que o sistema funcione de forma eficiente e segura.

2.3. Quais são as três principais atividades de um sistema operacional em relação ao gerenciamento de memória?

- **Resposta:**

1. **Alocação de memória:** Distribuir a memória disponível para os processos que precisam dela.
 2. **Proteção de memória:** Garantir que um processo não acesse a memória de outro processo sem permissão.
 3. **Gerenciamento de memória virtual:** Usar técnicas como paginação e segmentação para expandir a memória disponível e otimizar o uso da memória física.
- **Explicação:** O sistema operacional gerencia a memória para evitar conflitos e garantir que cada processo tenha o espaço necessário para executar suas tarefas.

2.4. Quais são as três principais atividades de um sistema operacional em relação ao gerenciamento de armazenamento secundário?

- **Resposta:**

1. **Gerenciamento de espaço livre:** Controlar quais áreas do disco estão disponíveis para armazenar novos dados.
 2. **Alocação de espaço:** Atribuir espaço no disco para arquivos e diretórios.
 3. **Gerenciamento de disco:** Otimizar o acesso aos dados no disco (ex.: agendamento de operações de leitura/escrita).
- **Explicação:** O sistema operacional organiza o armazenamento secundário (como discos rígidos ou SSDs) para garantir que os dados sejam armazenados e recuperados de forma eficiente.

2.5. Qual é a finalidade do interpretador de comandos? Por que, normalmente, ele é separado do kernel?

- **Resposta:** O interpretador de comandos (ou shell) é um programa que permite aos usuários interagir com o sistema operacional, executando comandos e scripts. Ele é separado do kernel para:
 1. **Flexibilidade:** Diferentes interpretadores de comandos (ex.: Bash, PowerShell) podem ser usados sem modificar o kernel.
 2. **Segurança:** Se o interpretador de comandos falhar, o kernel não é afetado.
 3. **Facilidade de desenvolvimento:** Novos interpretadores podem ser criados sem alterar o

núcleo do sistema.

- **Explicação:** Imagine o shell como um "tradutor" entre o usuário e o sistema operacional. Ele recebe comandos do usuário, traduz para chamadas de sistema e envia ao kernel para execução.

2.6. Quais chamadas do sistema precisam ser executadas por um interpretador de comandos ou shell a fim de iniciar um novo processo?

- **Resposta:**
 1. **fork()**: Cria uma cópia do processo atual (o processo filho).
 2. **exec()**: Substitui o código do processo filho pelo código de um novo programa.
 3. **wait()**: Espera que o processo filho termine (opcional).
- **Explicação:** Quando você digita um comando no shell, ele usa `fork()` para criar um novo processo e `exec()` para carregar o programa que você quer executar. O `wait()` é usado se o shell precisar esperar o término do processo.

2.7. Qual é a finalidade dos programas do sistema?

- **Resposta:** Os programas do sistema (ou utilitários) fornecem ferramentas para gerenciar e interagir com o sistema operacional. Eles incluem editores de texto, compiladores, gerenciadores de arquivos e ferramentas de rede.
- **Explicação:** Esses programas facilitam tarefas como editar arquivos, compilar código, gerenciar arquivos e configurar redes, sem que o usuário precise escrever código complexo.

2.8. Qual é a principal vantagem da técnica de camadas para o projeto do sistema? Quais são as desvantagens do uso da técnica de camadas?

- **Resposta:**
 - **Vantagem:** Facilita a depuração e manutenção, pois cada camada pode ser testada e modificada independentemente.
 - **Desvantagens:**
 1. **Overhead:** A comunicação entre camadas pode adicionar custos de desempenho.

2. **Complexidade:** Definir as camadas de forma adequada pode ser difícil.
- **Explicação:** Imagine o sistema operacional como um prédio com vários andares (camadas). Cada andar tem uma função específica, mas subir e descer entre eles pode ser lento.

2.9. Relacione cinco serviços fornecidos por um sistema operacional e explique como cada um cria conveniência para os usuários. Em que casos seria impossível que os programas no nível do usuário proverem esses serviços?

- **Resposta:**
 1. **Gerenciamento de arquivos:** Permite criar, ler e organizar arquivos. Programas de usuário não poderiam acessar o disco diretamente sem o sistema operacional.
 2. **Gerenciamento de memória:** Aloca memória para programas. Sem o sistema operacional, os programas poderiam colidir e corromper a memória.
 3. **Escalonamento de processos:** Decide qual programa roda na CPU. Programas de usuário não têm visão global do sistema para tomar essa decisão.
 4. **Proteção e segurança:** Impede que programas maliciosos acessem recursos indevidos. Programas de usuário não têm controle sobre o hardware.
 5. **Comunicação entre processos:** Permite que programas troquem dados. Programas de usuário não poderiam coordenar isso sem o sistema operacional.
- **Explicação:** O sistema operacional age como um "guardião" que gerencia recursos e garante que tudo funcione de forma segura e eficiente.

2.10. Por que alguns sistemas armazenam o sistema operacional no firmware, enquanto outros o armazenam no disco?

- **Resposta:**
 - **Firmware:** Usado em dispositivos embarcados (ex.: smartphones, IoT) para simplicidade e operação reforçada. O sistema operacional é carregado diretamente da memória não volátil.
 - **Disco:** Usado em PCs e servidores para flexibilidade e facilidade de atualização. O sistema operacional é carregado do armazenamento secundário (SSD/HDD).
- **Explicação:** Dispositivos pequenos e especializados usam firmware para economizar espaço e garantir operação confiável, enquanto sistemas maiores usam disco para permitir atualizações e

personalização.

2.11. Como um sistema poderia ser projetado para permitir uma escolha de sistemas operacionais para o boot do sistema? O que o programa de boot precisaria fazer?

- **Resposta:**
 - **Dual Boot/Multi Boot:** O programa de boot (ex.: GRUB) permite escolher entre vários sistemas operacionais instalados no disco.
 - **Funcionamento:**
 1. O programa de boot carrega uma lista de sistemas operacionais disponíveis.
 2. O usuário seleciona o sistema desejado.
 3. O programa de boot carrega o kernel do sistema operacional escolhido na memória.
- **Explicação:** Imagine o programa de boot como um "menu" que permite escolher entre Windows, Linux ou outro sistema operacional instalado no computador.

Questões 1

Esta seção contém perguntas relacionadas ao assunto em questão, que podem ser usadas como referência para aprender ou revisar conhecimentos.

- i** Sugerimos que resolva estas questões para auxiliar em um melhor entendimento do conteúdo e assim melhor resolução dos quizzes

Pergunta 1

Dispositivos que utilizam Bluetooth e redes Wi-Fi se comunicam sem fio dentro de uma determinada área, formando uma:

- a) Rede de área metropolitana (MAN)
- b) Rede de área local (LAN)
- c) Rede de pequena área (SAN)
- d) Rede de longa distância (WAN)

Pergunta 2

O que fornece serviços adicionais para desenvolvedores ao intermediar a comunicação entre aplicativos e o sistema operacional?

- a) Virtualização
- b) Middleware
- c) Computação em nuvem
- d) Software de sistema

Pergunta 3

Os sistemas operacionais geralmente operam em dois modos distintos. Quais são eles?

- a) Modo físico e modo lógico
- b) Modo usuário e modo kernel
- c) Modo supervisor e modo secundário
- d) Modo protegido e modo comum

Pergunta 4

No contexto do Linux, uma versão personalizada do sistema operacional é conhecida como:

- a) Instalação (Installation)
- b) Distribuição (Distribution)
- c) LiveCD
- d) Virtual Machine

Pergunta 5

Qual das seguintes afirmações sobre dispositivos móveis é falsa?

- a) Eles geralmente possuem menos núcleos de processamento do que desktops.
- b) O consumo de energia é um fator crítico para dispositivos móveis.
- c) Dispositivos móveis sempre possuem mais capacidade de armazenamento que laptops.
- d) Algumas funcionalidades dos dispositivos móveis não estão presentes em desktops.

Pergunta 6

Sistemas embarcados geralmente executam um sistema operacional de:

- a) Tempo real
- b) Rede
- c) Clusterizado
- d) Multiprogramação

Pergunta 7

Quais são dois fatores importantes no design da memória cache?

- a) Consumo de energia e reutilização
- b) Política de tamanho e substituição
- c) Privilégios de acesso e controle
- d) Velocidade e volatilidade

Pergunta 8

De que forma um sistema operacional pode ser comparado a um governo?

- a) Ele executa todas as funções sozinho.
- b) Ele cria um ambiente para que outros programas possam operar.

- c) Ele prioriza as necessidades individuais dos usuários.
- d) Ele raramente funciona corretamente.

Pergunta 9

Sobre instruções privilegiadas, qual das afirmações a seguir é incorreta?

- a) Elas não podem ser executadas no modo usuário.
- b) Apenas podem ser executadas no modo kernel.
- c) São usadas para gerenciamento de interrupções.
- d) Nunca representam riscos ao sistema.

Pergunta 10

A menor unidade de execução dentro de um sistema operacional é chamada de:

- a) Sistema operacional
- b) Timer
- c) Bit de modo
- d) Processo

Pergunta 11

Qual das opções abaixo é um exemplo de um programa de sistema?

- a) Navegador Web
- b) Interpretador de comandos
- c) Planilha eletrônica
- d) Software de edição de imagem

Pergunta 12

Qual chamada de sistema do Windows é equivalente à `close()` do UNIX?

- a) CloseHandle()
- b) close()
- c) Exit()
- d) CloseFile()

Pergunta 13

As chamadas de sistema são responsáveis por:

- a) Fornecer uma interface para os serviços do sistema operacional
- b) Gerenciar apenas memória virtual
- c) Proteger exclusivamente processos críticos
- d) Impedir a execução de programas de terceiros

Pergunta 14

O que define o que será feito dentro de um sistema operacional?

- a) Política
- b) Mecanismo
- c) Estratégia
- d) Interface

Pergunta 15

No Windows, a chamada de sistema `CreateFile()` é utilizada para criar arquivos. Qual a chamada equivalente no UNIX?

- a) `fork()`
- b) `open()`
- c) `createfile()`
- d) `ioctl()`

Pergunta 16

Qual das opções **não** é uma categoria principal de chamadas de sistema?

- a) Segurança
- b) Proteção
- c) Controle de processos
- d) Comunicação

Pergunta 17

Microkernels utilizam ____ para comunicação interna.

- a) Chamadas de sistema
- b) Memória compartilhada
- c) Virtualização
- d) Passagem de mensagens

Pergunta 18

Um bloco de inicialização (bootstrap loader):

- a) É composto por vários blocos de disco
- b) Pode conter múltiplos cilindros de disco
- c) Normalmente é suficiente para carregar e iniciar o sistema operacional
- d) Apenas aponta para a localização do restante do sistema de boot

Pergunta 19

Qual é o sistema operacional utilizado em dispositivos iPhone e iPad?

- a) iOS
- b) UNIX
- c) Android
- d) Mac OS X

Pergunta 20

Para um programa SYSGEN de um sistema operacional, qual das informações abaixo é **menos útil**?

- a) Quantidade de memória disponível
- b) Configurações como tamanho de buffer e algoritmo de escalonamento de CPU
- c) Lista de aplicativos a serem instalados
- d) Arquitetura da CPU

Pergunta 21

Um sistema operacional pode ser classificado como um:

- a) Gerenciador de recursos
- b) Programa de usuário
- c) Firmware de inicialização

- d) Simulador de processos

Pergunta 22

O que é multitarefa em um sistema operacional?

- a) A execução de um único processo por vez
- b) A capacidade de executar múltiplos processos simultaneamente
- c) A execução de tarefas em tempo real sem atraso
- d) A execução de comandos administrativos pelo usuário

Pergunta 23

Qual das seguintes opções descreve um **hipervisor**?

- a) Um software responsável pela virtualização de hardware
- b) Um protocolo de comunicação em redes
- c) Um sistema operacional para servidores
- d) Um método de gerenciamento de arquivos

Pergunta 24

O que acontece quando um processo em execução tenta acessar uma página de memória que não está carregada?

- a) Ele é imediatamente encerrado pelo sistema operacional
- b) O sistema operacional gera uma interrupção e carrega a página necessária
- c) O sistema operacional ignora a solicitação
- d) O processo entra em um estado de loop infinito

Pergunta 25

O que é um deadlock em um sistema operacional?

- a) Um erro crítico no kernel
- b) Um estado onde dois ou mais processos ficam bloqueados indefinidamente
- c) Um método de gerenciamento de arquivos
- d) Uma forma de escalonamento de processos

Prática 1

Nesta prática, vamos criar e configurar máquinas virtuais.

Conhecendo ferramentas

Para fazer a virtualização de sistemas operacionais, temos alguns programas disponíveis:

- Windows:
 - VirtualBox
 - VMWare
 - Parallels Desktop
- Linux:
 - VirtualBox
 - GNOME Boxes
 - Virtual Machine Manager

i Há casos em que o VirtualBox falha por razões desconhecidas, então é melhor ter mais de uma opção disponível.

Vamos começar

Requisitos dos sistemas

Confira os requisitos dos sistemas operacionais que serão usados:

- Windows 10 (<https://support.microsoft.com/pt-br/windows/requisitos-do-sistema-do-windows-10-6d4e9a79-66bf-7950-467c-795cf0386715>)
- Ubuntu Desktop (<https://ubuntu.com/server/docs/system-requirements>)

i É importante verificar os requisitos mínimos para garantir que sua máquina virtual tenha recursos suficientes para executar o sistema operacional escolhido.

Instalando a ferramenta

Vamos usar o VirtualBox.

- Para Windows (<https://download.virtualbox.org/virtualbox/7.1.6/VirtualBox-7.1.6-167084-Win.exe>)
- Para Linux (https://www.virtualbox.org/wiki/Linux_Downloads)

i O VirtualBox é uma ferramenta de virtualização gratuita e de código aberto, adequada para usuários iniciantes e avançados.

Instalar ISOs (Windows e Ubuntu)

- Windows (<https://www.microsoft.com/pt-br/software-download/windows10ISO>)

i Na ISO oficial do Windows, o link acima, vêm todas as versões.

- Ubuntu Desktop (<https://ubuntu.com/download/desktop>)

i Certifique-se de baixar a versão mais recente e estável do Ubuntu Desktop para garantir compatibilidade e suporte.

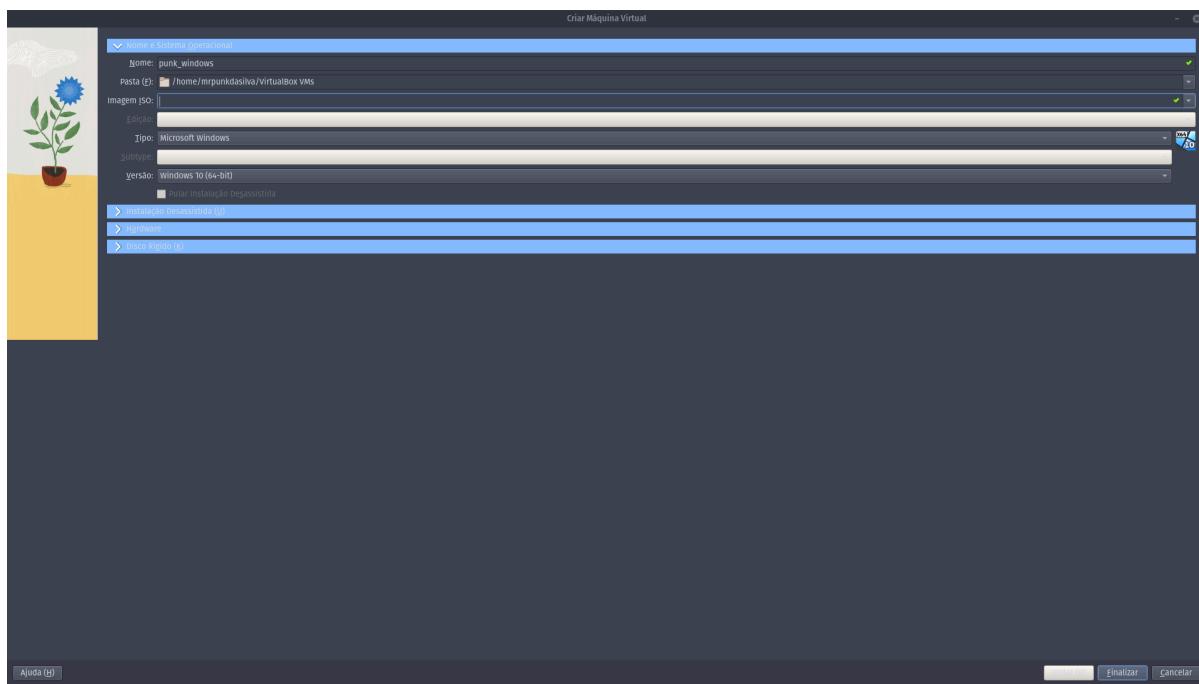
Criando Máquinas Virtuais

Windows

1. Com o VirtualBox aberto, pressione: **CTRL** + **N**

i Você pode acessar a opção também por: Machine > Add

2. Definir nome, sistema e versão:

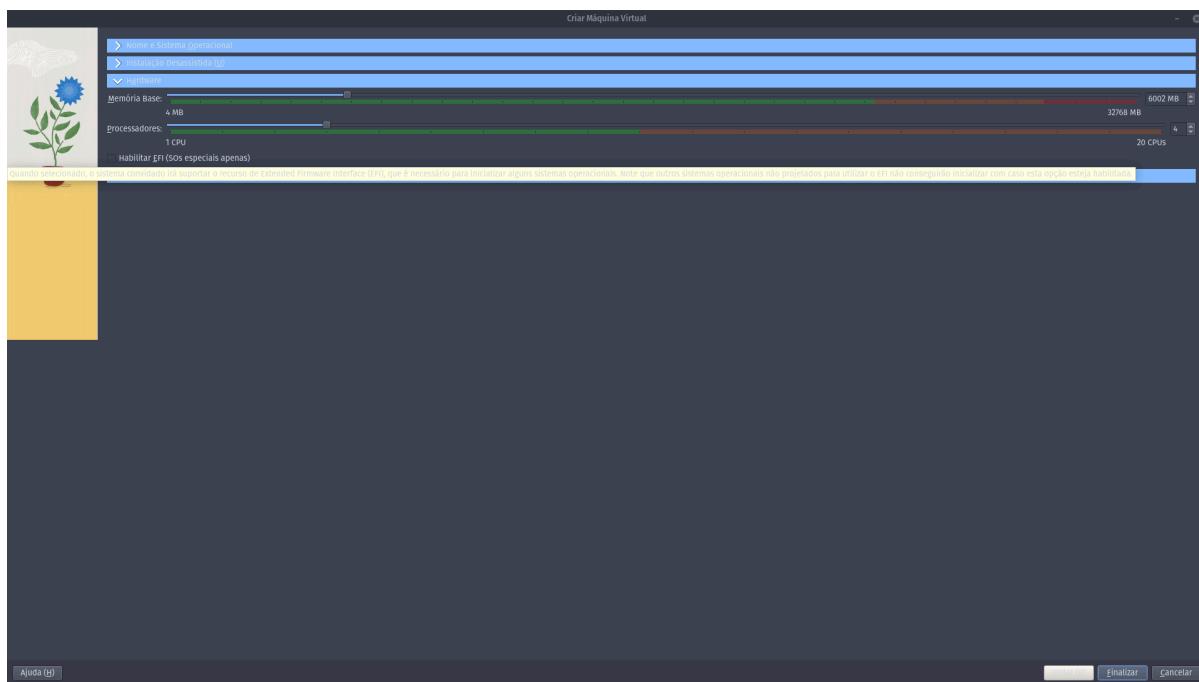


Tela de criação de máquina virtual no VirtualBox

- i** Escolha um nome descritivo para sua máquina virtual e selecione a versão correta do Windows que você planeja instalar.

3. Agora vamos definir a memória RAM. É bom deixarmos no mínimo 4GB

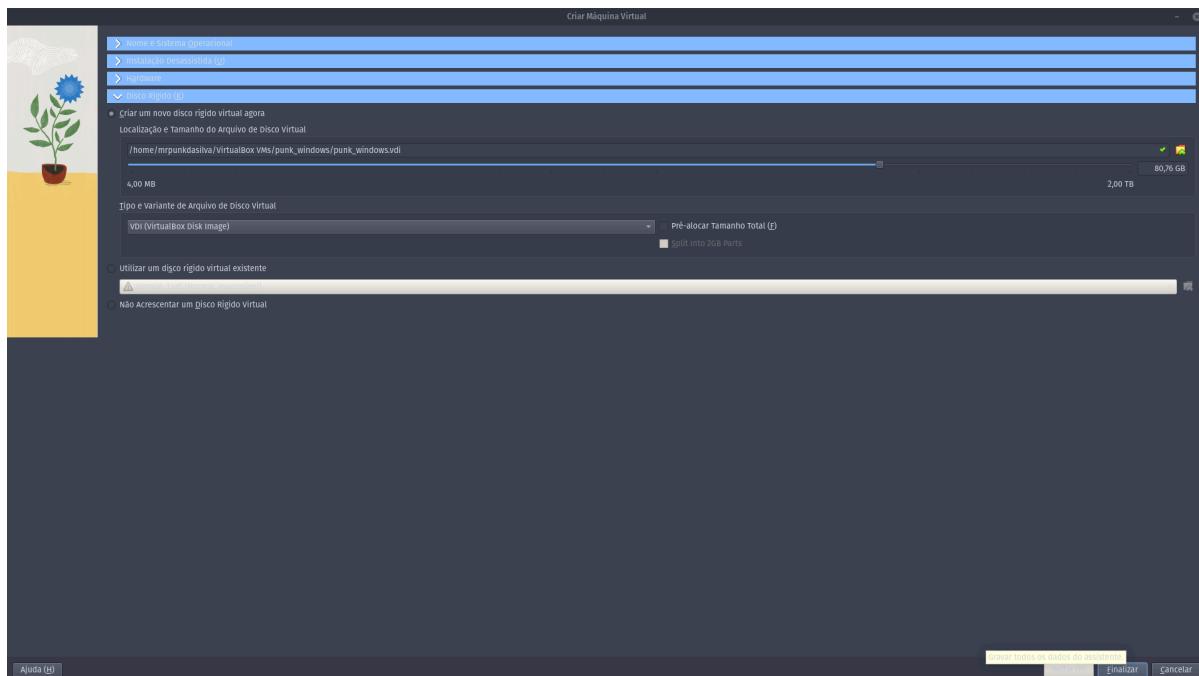
- i** Atente-se que computadores não lidam bem com números ímpares.



Configuração de memória RAM para a máquina virtual

- i** A quantidade de RAM alocada afetará diretamente o desempenho da sua máquina virtual. Certifique-se de deixar RAM suficiente para o seu sistema host também.

4. Agora definimos o espaço de armazenamento, disco rígido:

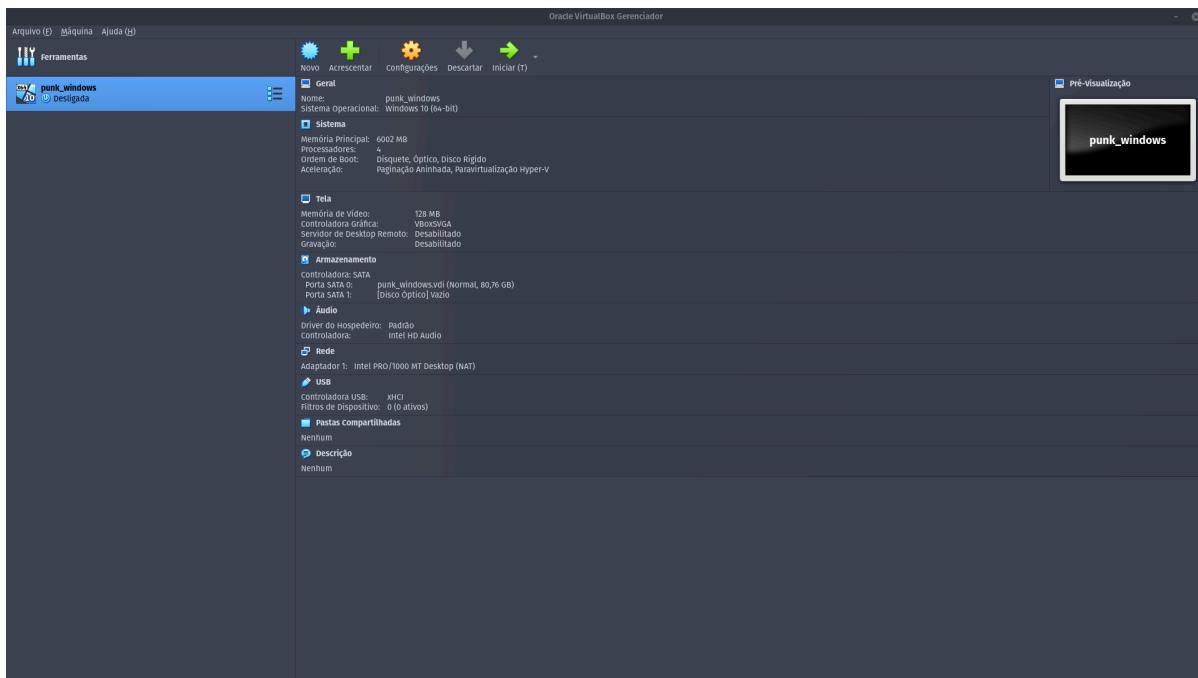


Configuração de disco rígido para a máquina virtual

- i** O tamanho do disco virtual deve ser suficiente para o sistema operacional e os programas que você planeja instalar. Você pode aumentar o tamanho posteriormente, mas é mais complicado diminuí-lo.

5. Com tudo criado, basta ir em **Finish**:

6. Temos então nossa primeira máquina virtual criada:



Máquina virtual Windows criada no VirtualBox

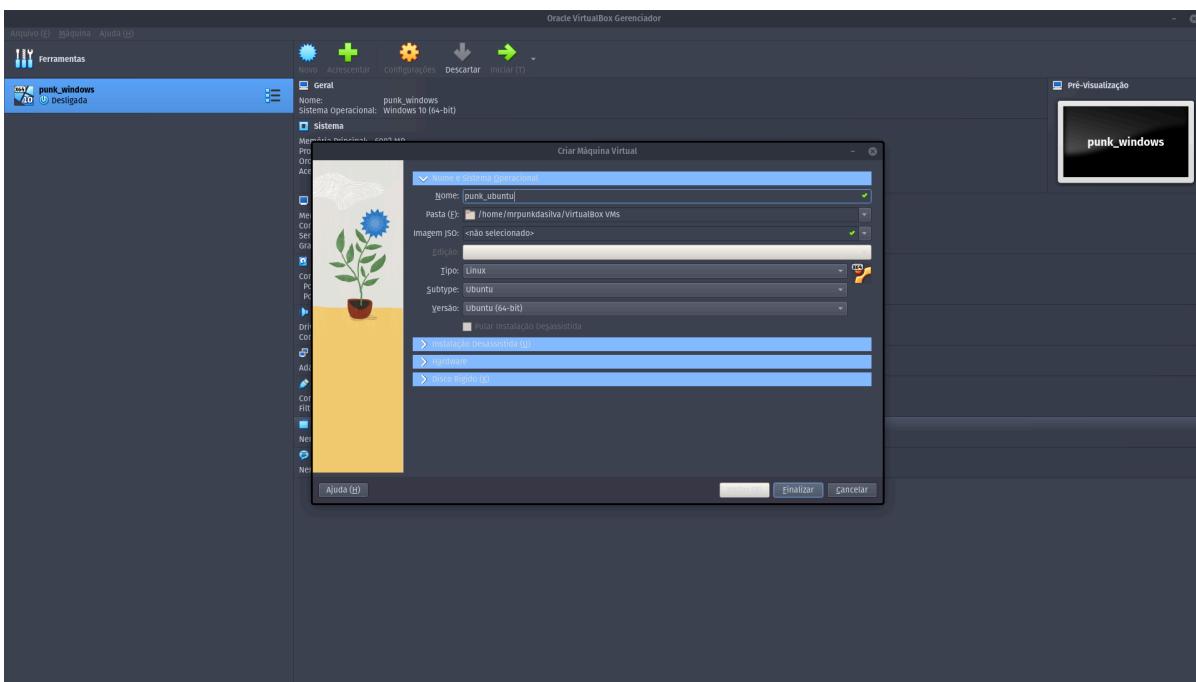
- i** Parabéns! Você criou sua primeira máquina virtual. Agora está pronta para receber o sistema operacional.

Ubuntu

1. Com o VirtualBox aberto, pressione: **CTRL** + **N**

- i** Você pode acessar a opção também por: Machine > Add

2. Definir nome, sistema e versão:

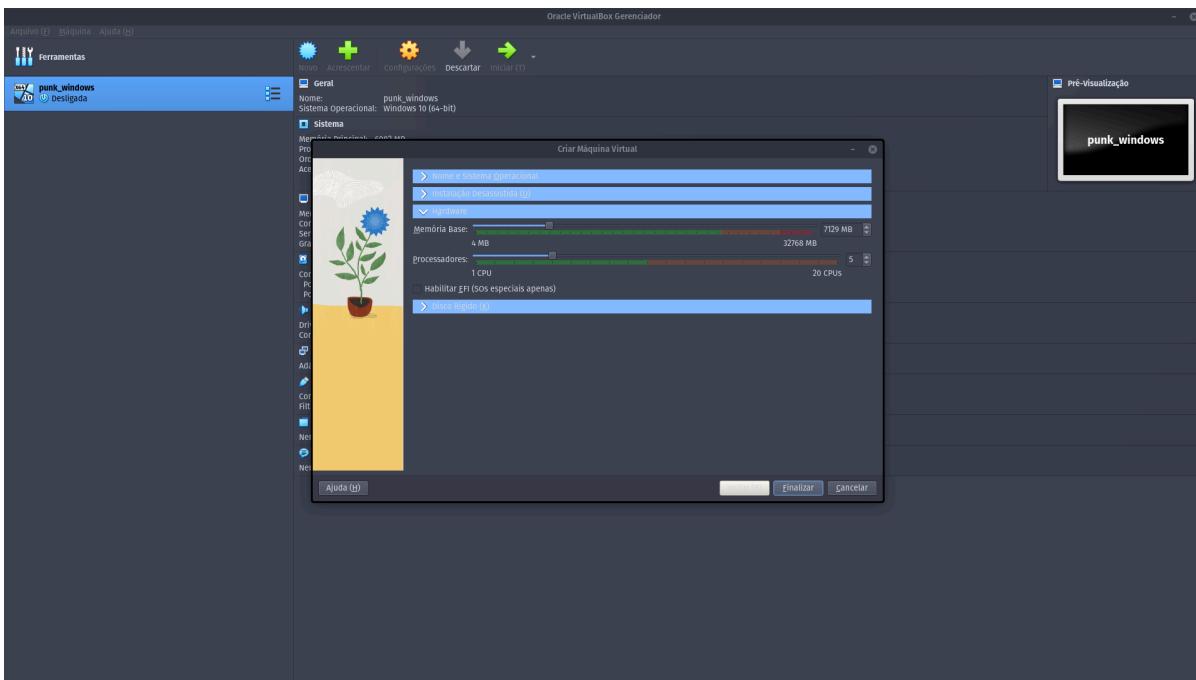


Tela de criação de máquina virtual Ubuntu no VirtualBox

- i** Certifique-se de selecionar "Ubuntu" como o tipo de sistema e escolha a versão correta que você planeja instalar.

3. Agora vamos definir a memória RAM. É bom deixarmos no mínimo 4GB

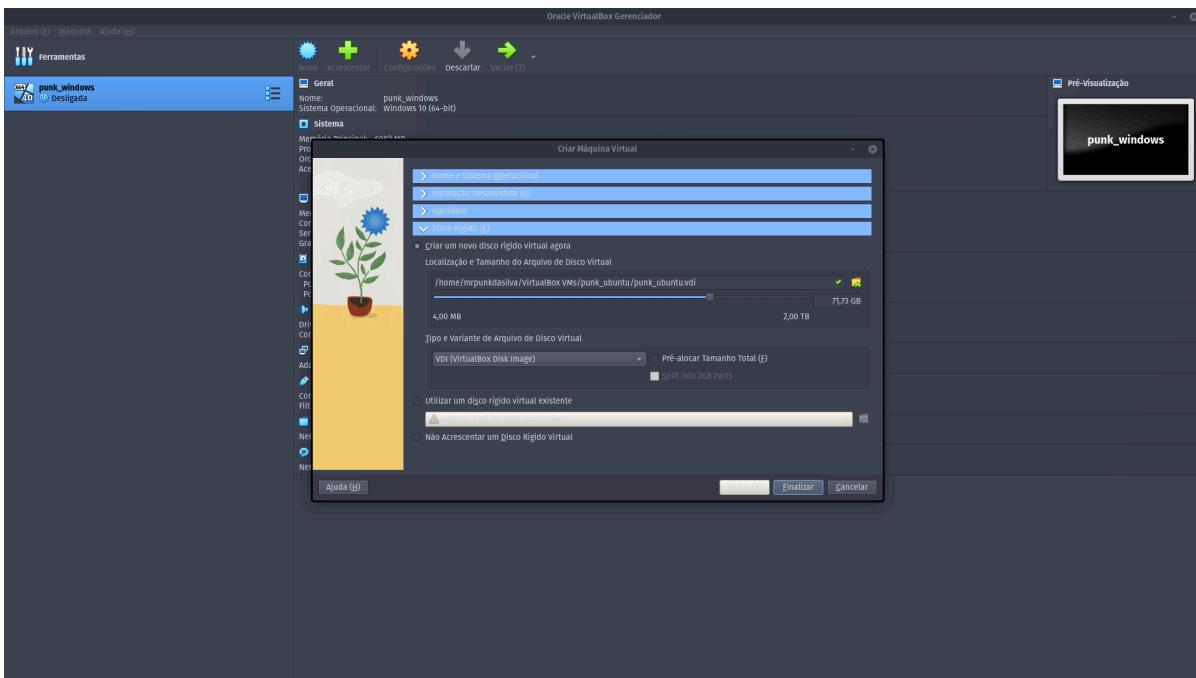
- i** Atente-se que computadores não lidam bem com números ímpares.



Configuração de memória RAM para a máquina virtual Ubuntu

- O Ubuntu geralmente requer menos RAM que o Windows, mas 4GB é uma boa quantidade para garantir um desempenho suave.

4. Agora definimos o espaço de armazenamento, disco rígido:

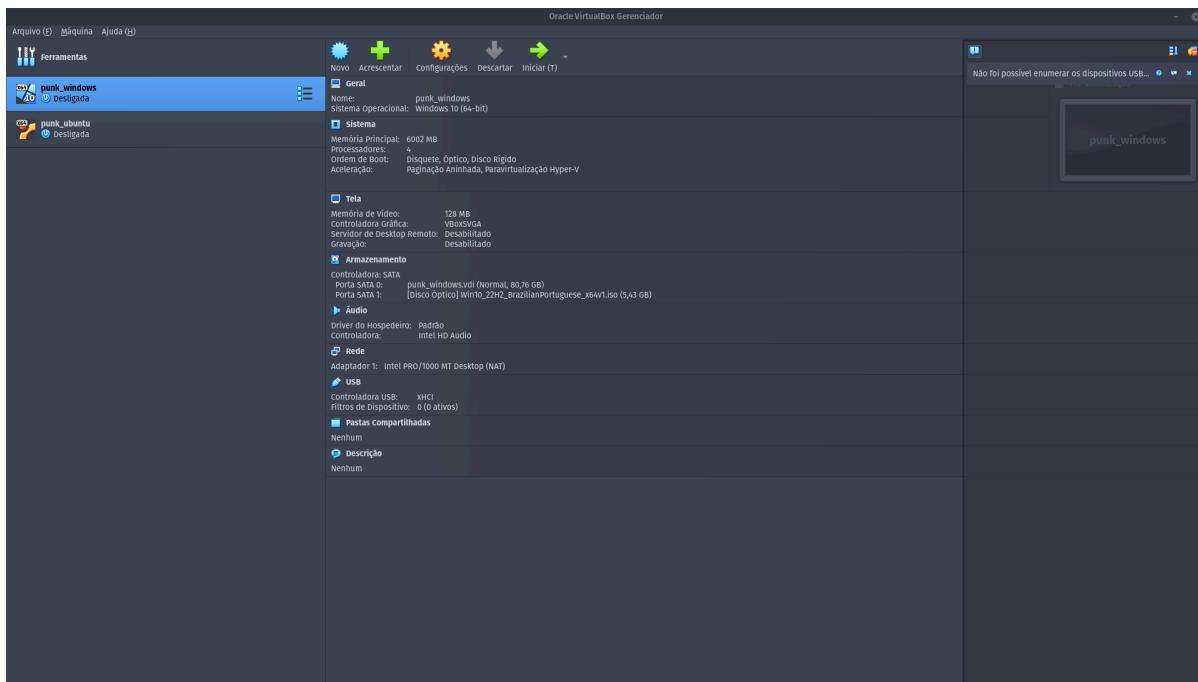


Configuração de disco rígido para a máquina virtual Ubuntu

- i** O Ubuntu geralmente requer menos espaço em disco que o Windows. 20GB é suficiente para uma instalação básica, mas considere alocar mais se planeja instalar muitos programas.

5. Com tudo criado, basta ir em **Finish**:

6. Temos então nossa primeira máquina virtual criada:



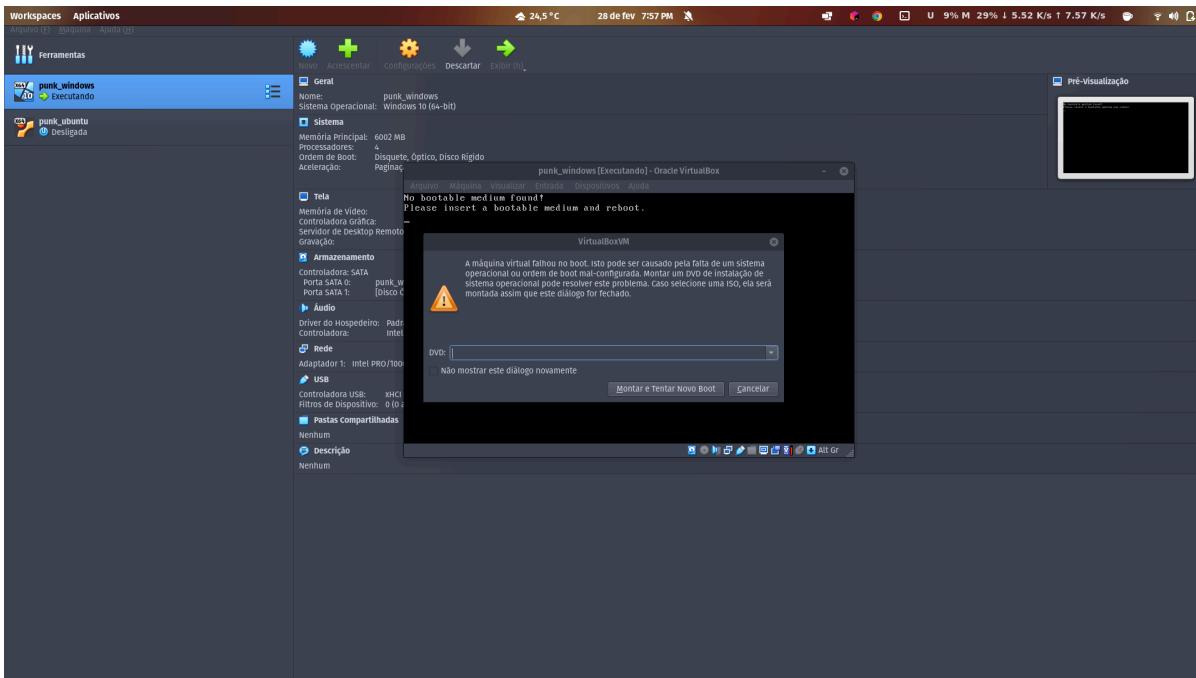
Máquina virtual Ubuntu criada no VirtualBox

- i** Sua máquina virtual Ubuntu está pronta para receber o sistema operacional. O próximo passo será iniciar a instalação do Ubuntu.

Logar nas VMs recém-criadas

Ao executar as máquinas, nenhum sistema será inicializado, já que não foi definida nenhuma ISO (Imagem de um Sistema Operacional). Assim, as máquinas ficam em seu estado puro, sem nenhum sistema operacional, e são inutilizáveis.

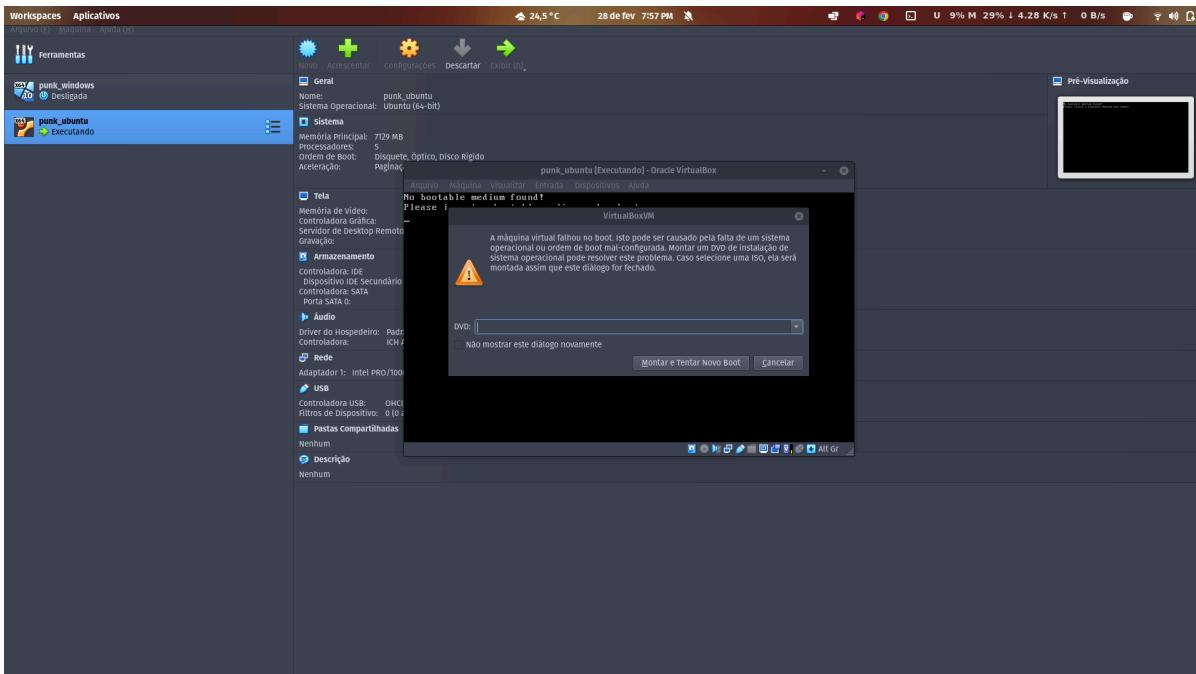
Windows



Tela inicial da máquina virtual Windows sem sistema operacional

- Esta tela indica que a máquina virtual está pronta, mas ainda não tem um sistema operacional instalado.

Ubuntu



Tela inicial da máquina virtual Ubuntu sem sistema operacional

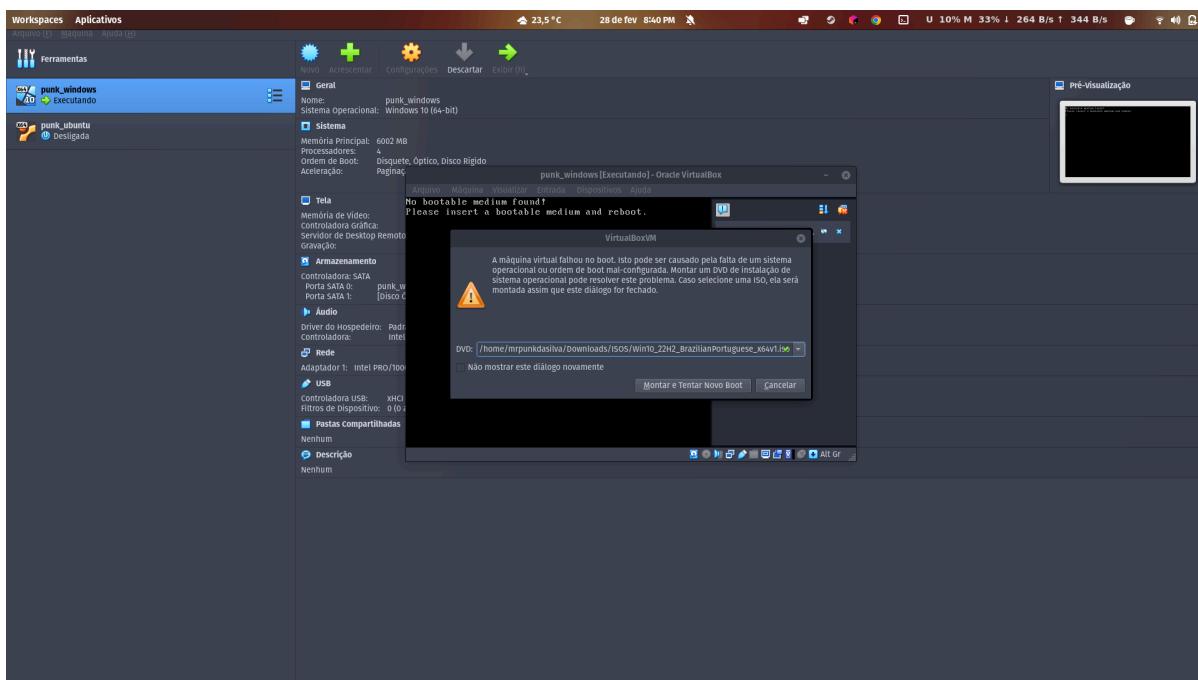
- i** Semelhante à máquina Windows, esta tela mostra que a máquina virtual Ubuntu está criada, mas ainda precisa de um sistema operacional.

Configurando VMs para os SOs

Para tornar as VMs utilizáveis, precisamos definir as ISOs que serão as imagens do sistema usadas para instalar o sistema operacional.

Windows

Com a máquina em execução e este pop-up aparecendo, selecionamos onde está a ISO do Windows que foi baixada nos passos anteriores:

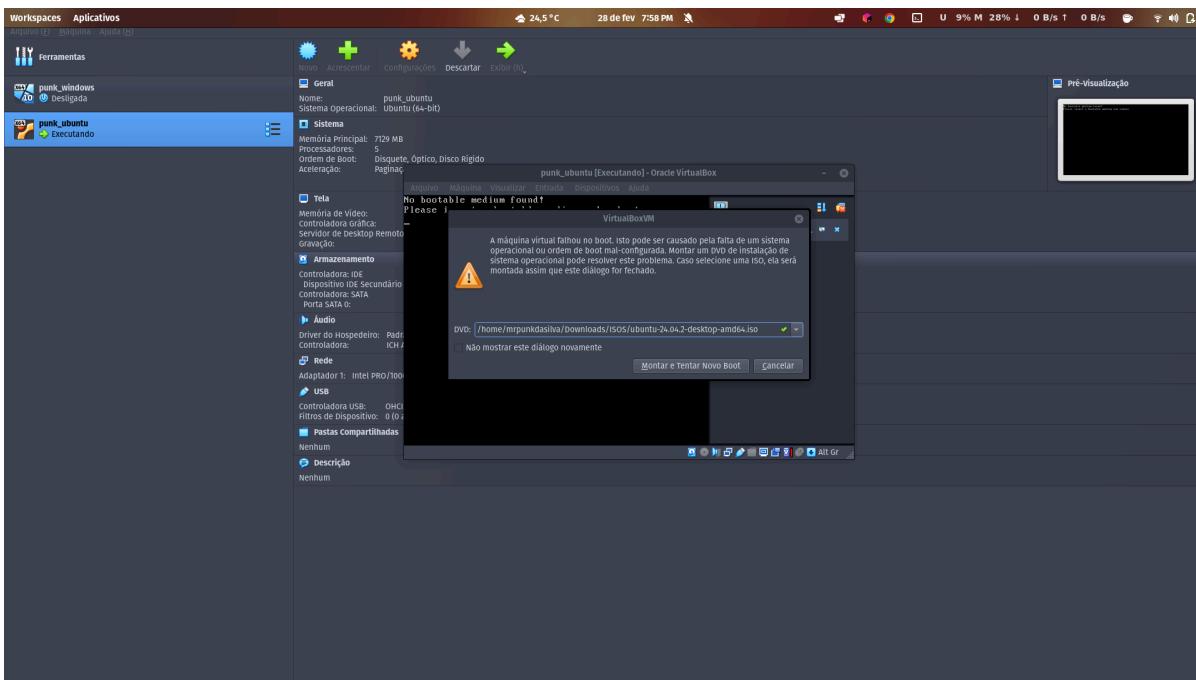


Seleção da ISO do Windows para instalação

- i** Certifique-se de selecionar a ISO correta do Windows que você baixou anteriormente. Isso iniciará o processo de instalação do Windows na sua máquina virtual.

Ubuntu

Com a máquina em execução e este pop-up aparecendo, selecionamos onde está a ISO do Ubuntu que foi baixada nos passos anteriores:



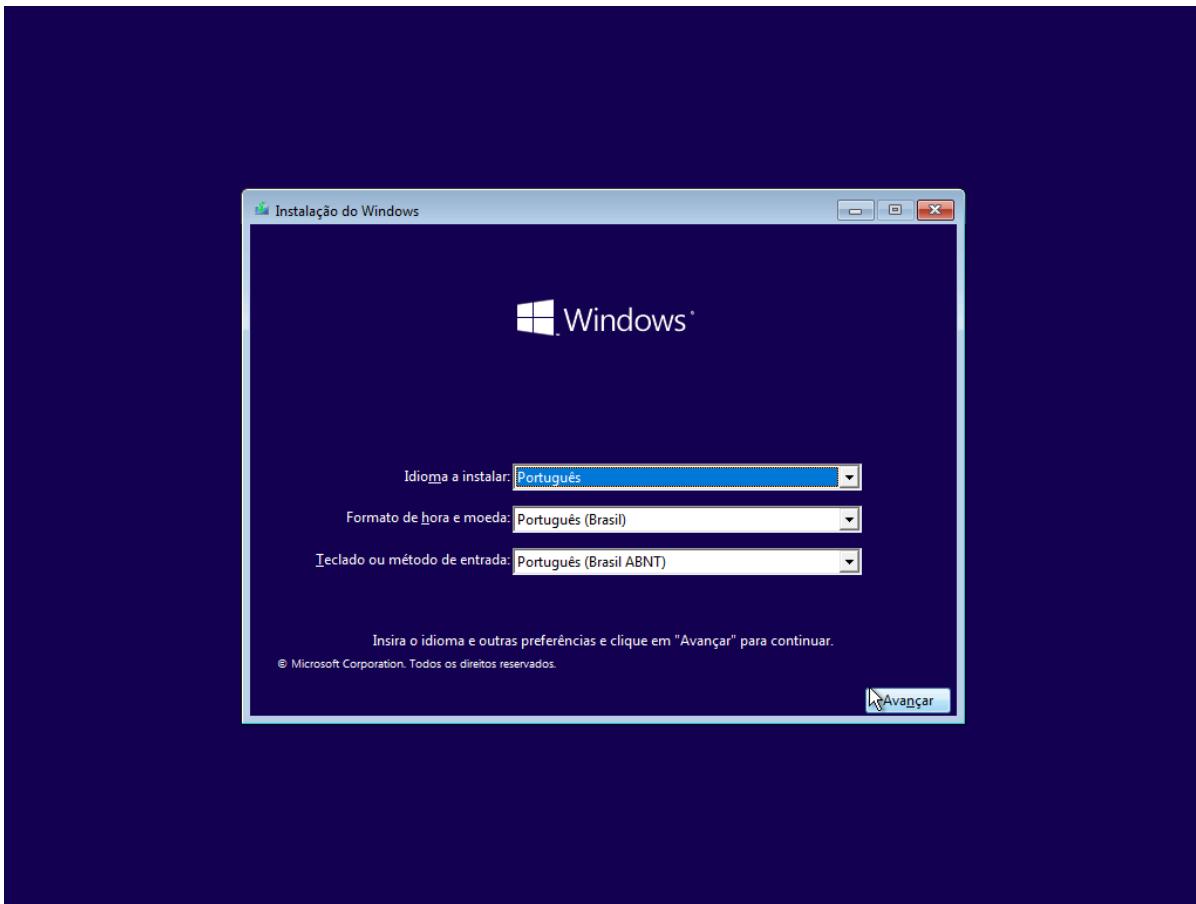
Seleção da ISO do Ubuntu para instalação

- i** Selecione a ISO do Ubuntu que você baixou. Isso iniciará o processo de instalação do Ubuntu na sua máquina virtual.

Logar nas VMs

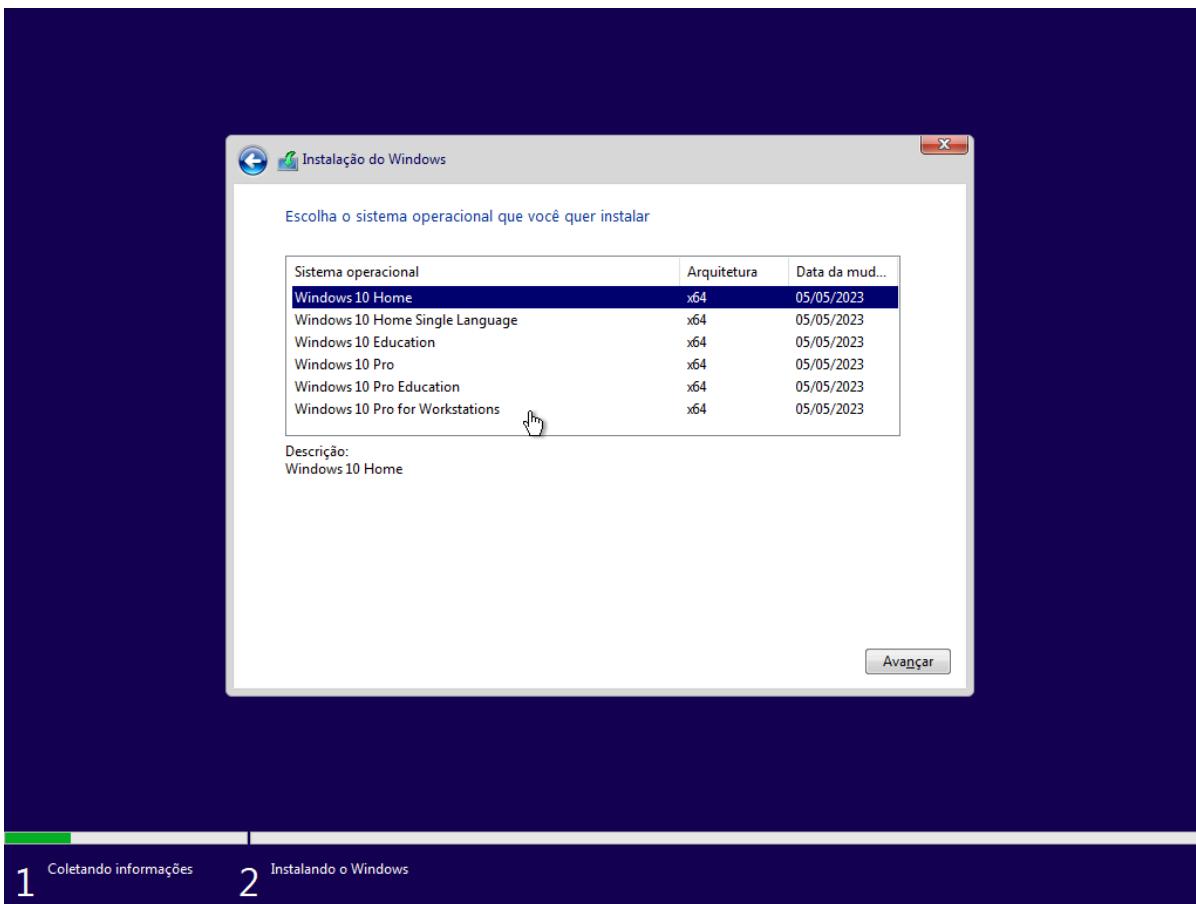
Agora, com tudo o que vimos, podemos fazer a instalação do sistema. Para isso, devemos logar ou entrar nas máquinas que criamos e então fazer as etapas de instalação do Windows e Ubuntu.

Windows



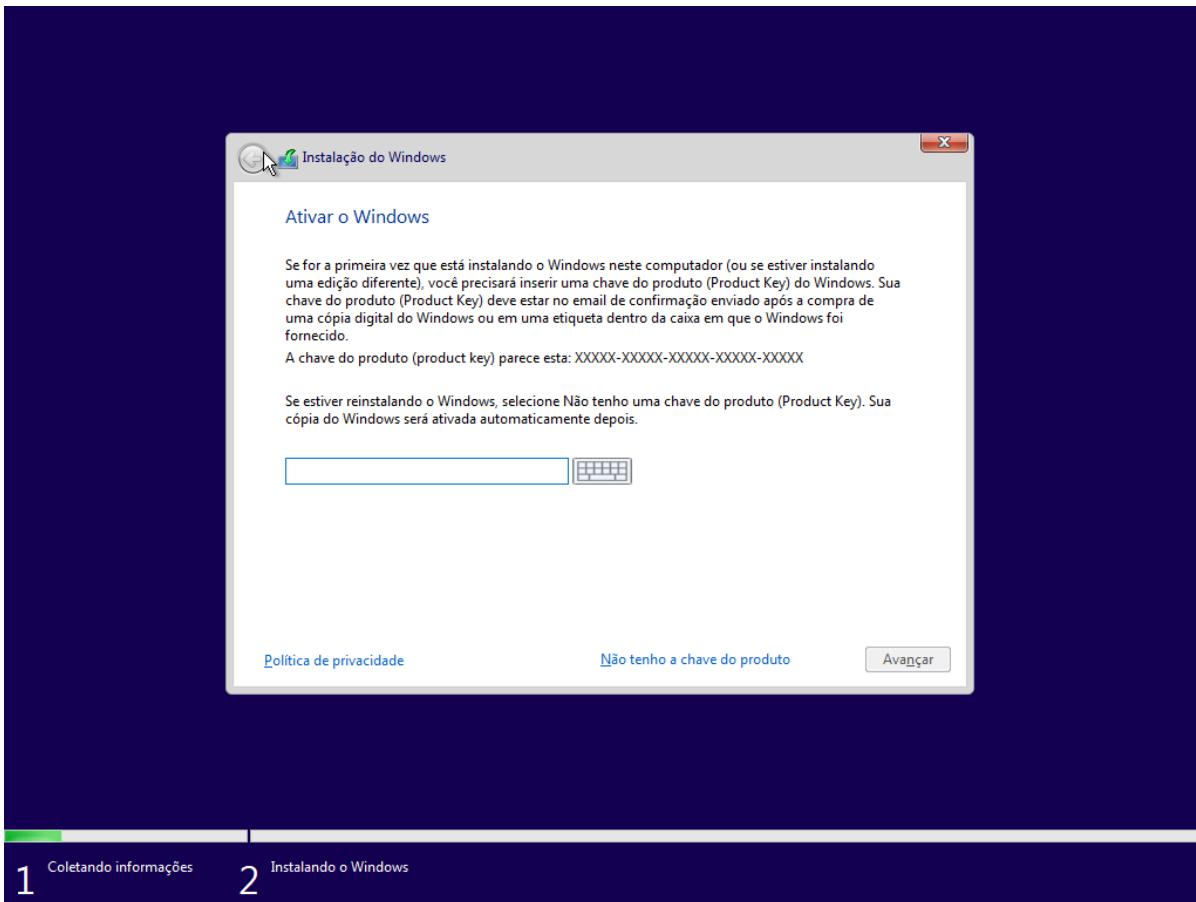
Tela inicial de instalação do Windows

- Esta é a tela inicial de instalação do Windows. A partir daqui, você seguirá as etapas para configurar seu sistema Windows.



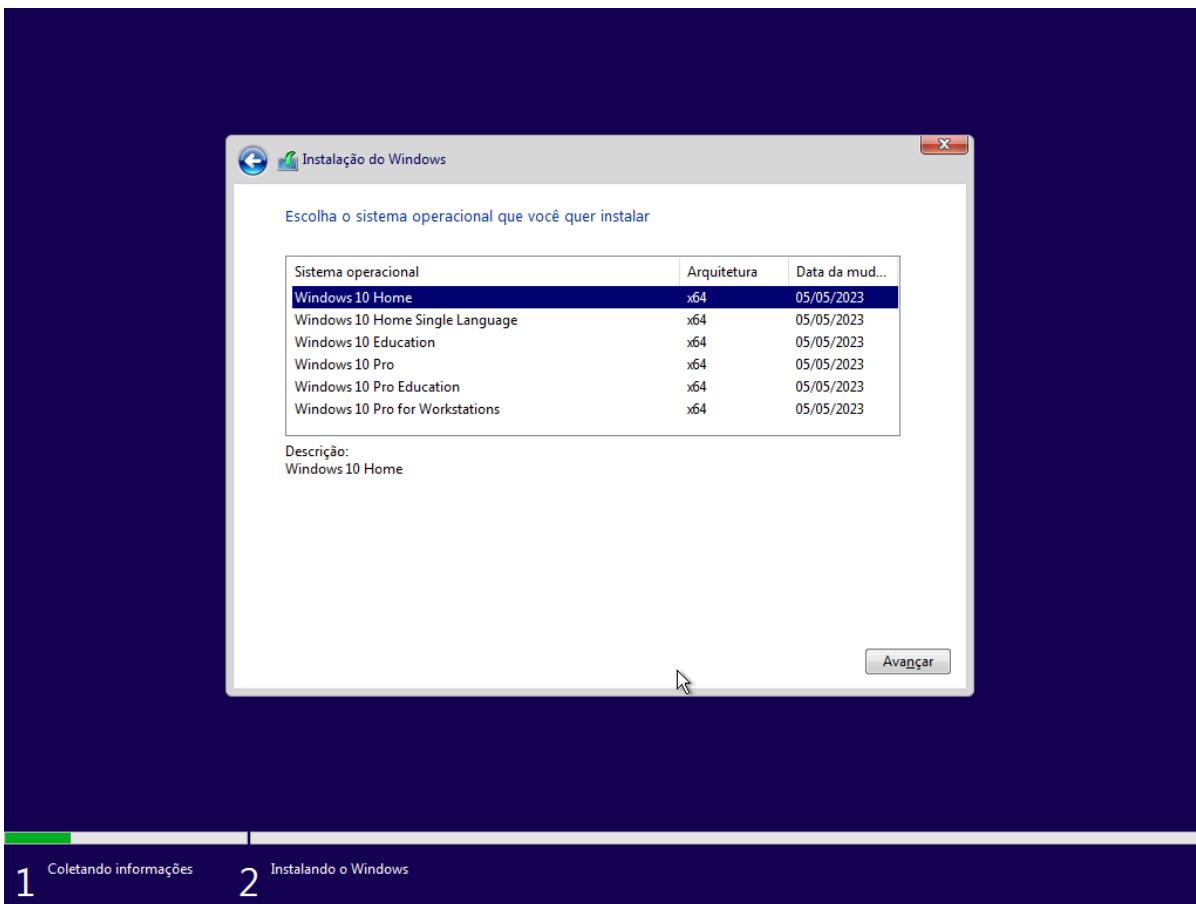
Seleção de idioma, formato de hora e moeda, e layout de teclado

- Escolha as opções que melhor se adequam à sua região e preferências de idioma.



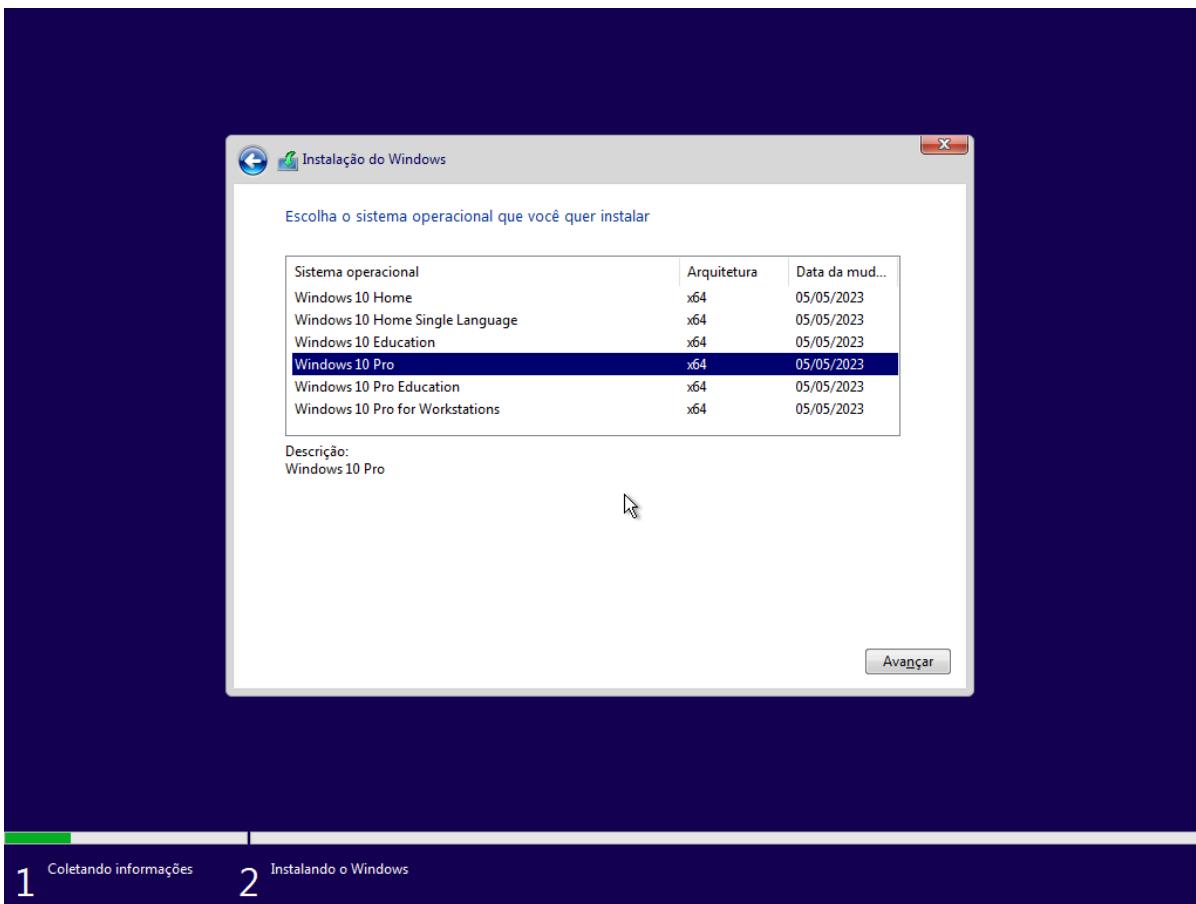
Botão "Instalar agora" para iniciar a instalação do Windows

- Clique em "Instalar agora" para começar o processo de instalação do Windows.



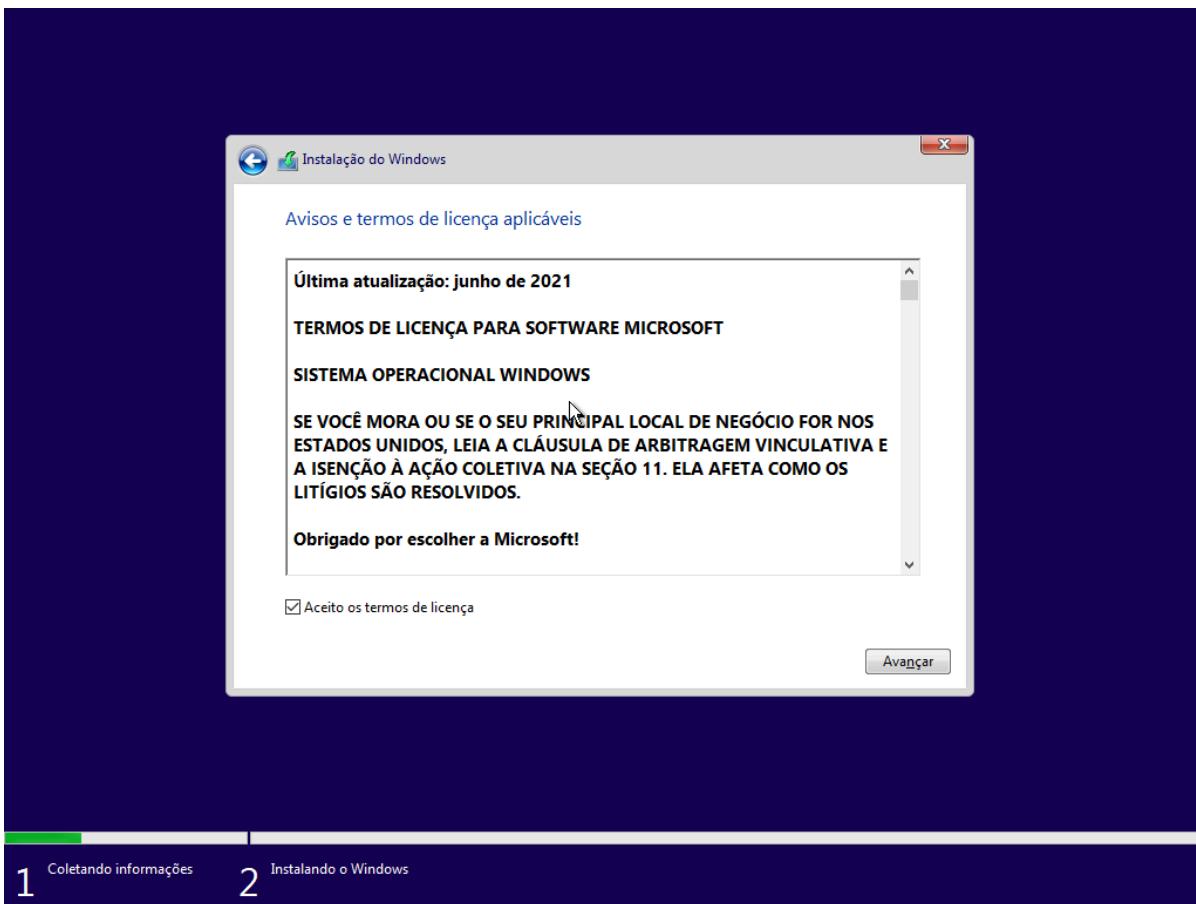
Inserção da chave do produto Windows

- Se você tiver uma chave do produto, insira-a aqui. Caso contrário, você pode pular esta etapa e ativar o Windows posteriormente.



Seleção da versão do Windows a ser instalada

- Escolha a versão do Windows que deseja instalar. Para uso pessoal, "Windows 10 Home" geralmente é suficiente.

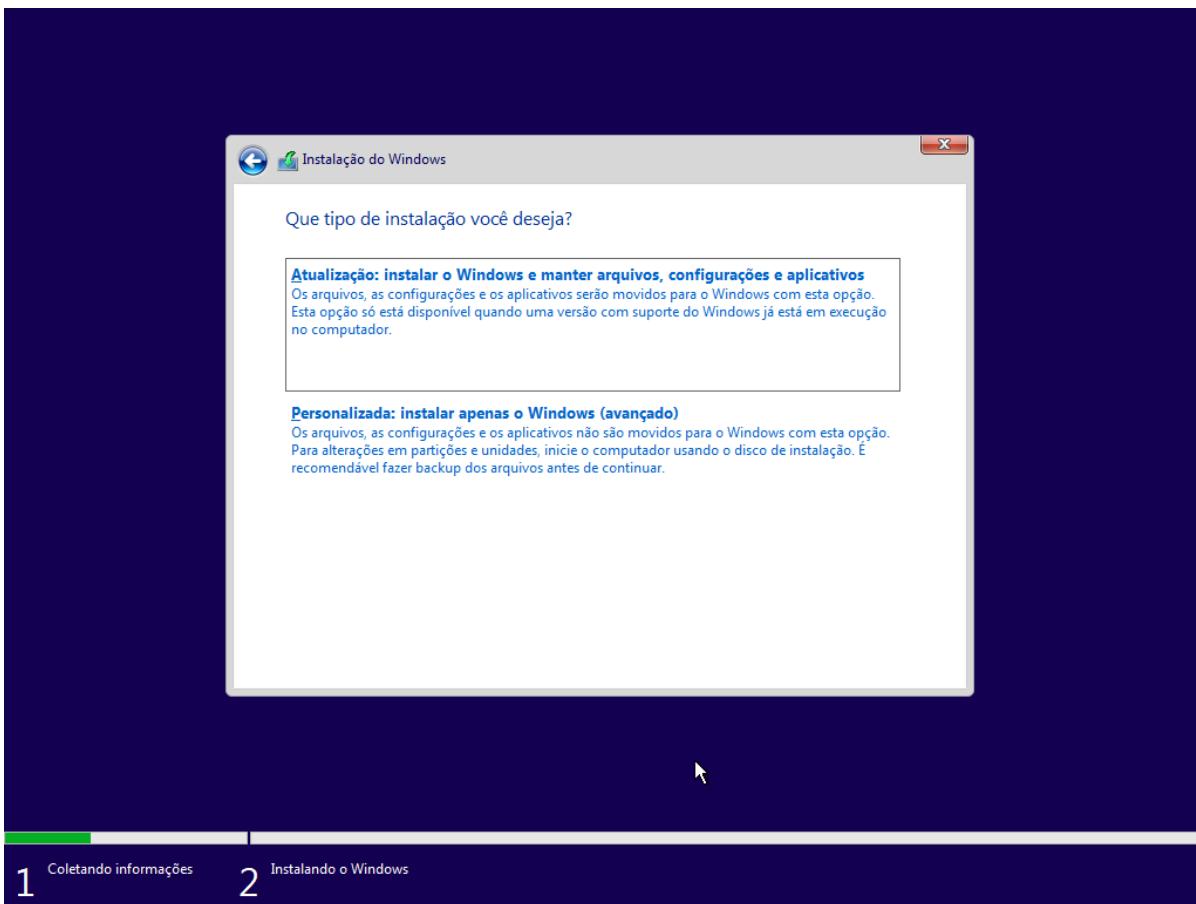


1 Coletando informações

2 Instalando o Windows

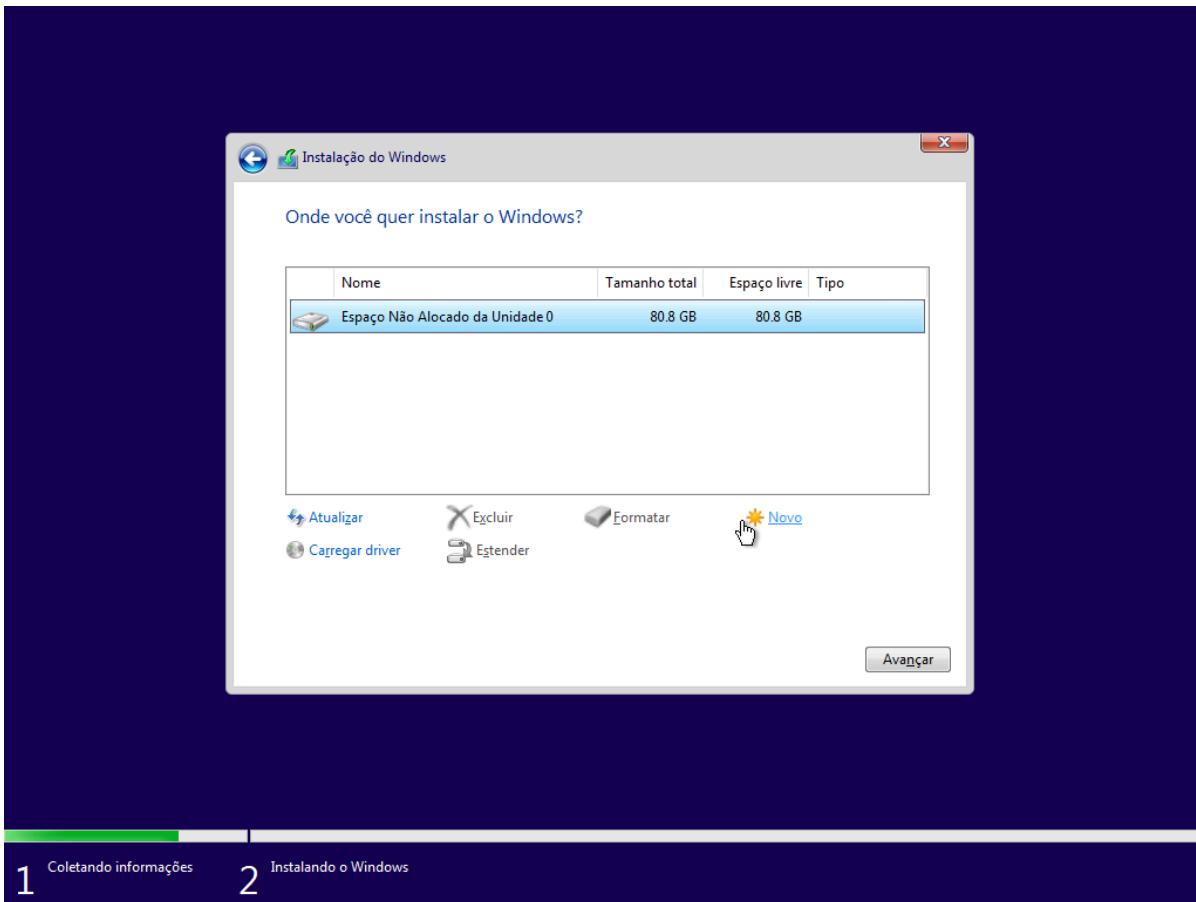
Aceitação dos termos de licença do Windows

- Leia e aceite os termos de licença para prosseguir com a instalação.



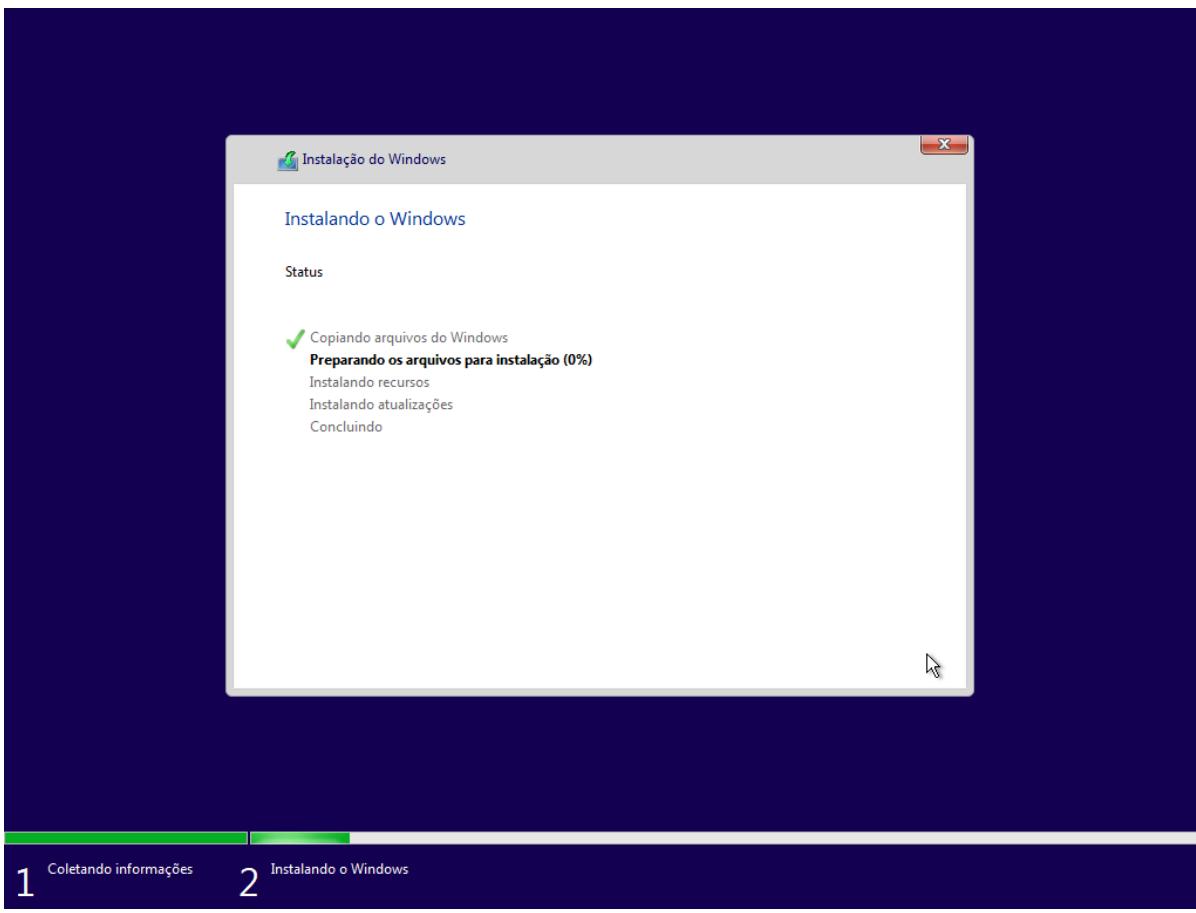
Escolha do tipo de instalação: Atualização ou Personalizada

- Para uma nova instalação em uma máquina virtual, escolha "Instalação Personalizada".



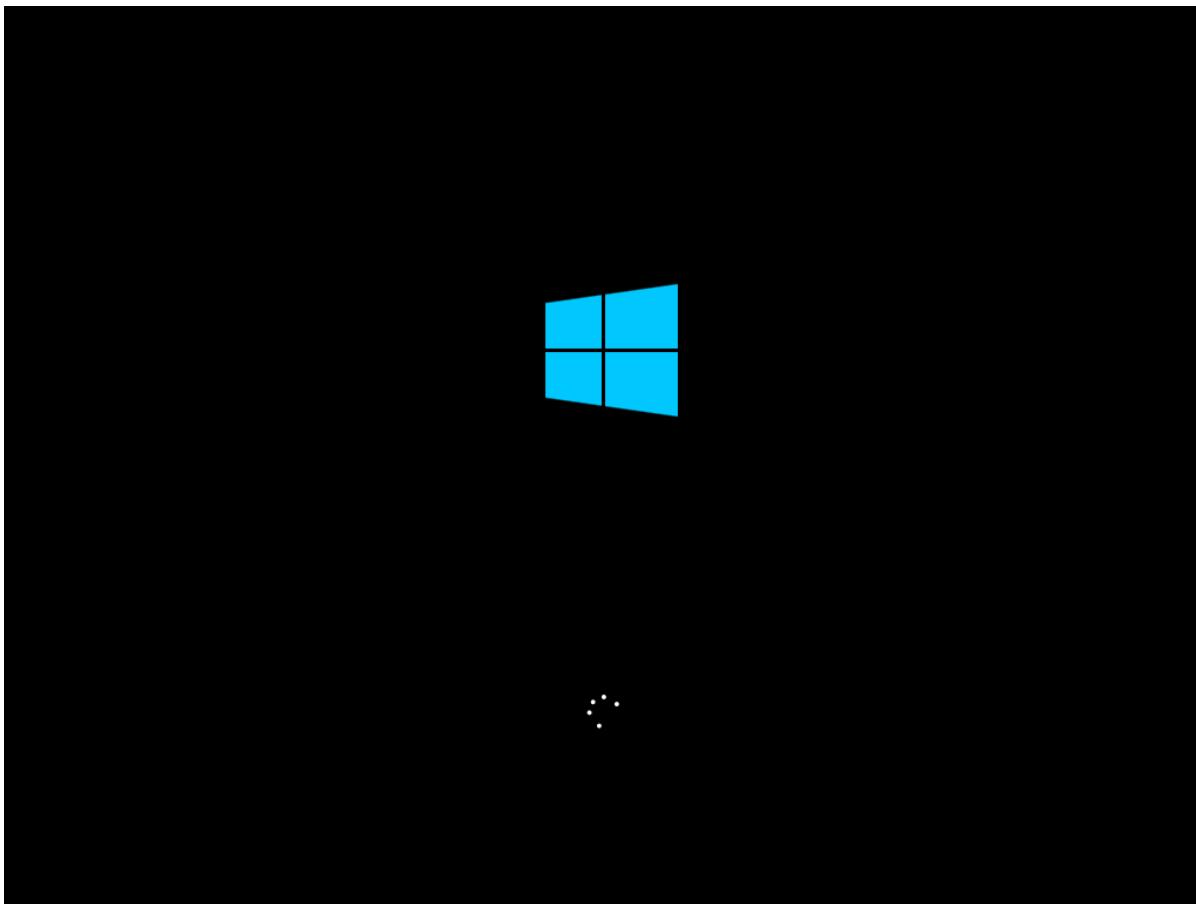
Seleção do disco onde o Windows será instalado

- Selecione o disco virtual que você criou anteriormente para instalar o Windows.



Progresso da instalação do Windows

- i** A instalação do Windows está em andamento. Isso pode levar alguns minutos.



Reinicialização do sistema após a instalação inicial

⚠ Após a instalação inicial, o sistema irá

Vamos começar

Requisitos dos sistemas

Confira os requisitos dos sistemas operacionais que serão usados:

- Windows 10 (<https://support.microsoft.com/pt-br/windows/requisitos-do-sistema-do-windows-10-6d4e9a79-66bf-7950-467c-795cf0386715>)
- Ubuntu Desktop (<https://ubuntu.com/server/docs/system-requirements>)

Instalando a ferramenta

Vamos usar o VirtualBox.

- Para Windows (<https://download.virtualbox.org/virtualbox/7.1.6/VirtualBox-7.1.6-167084-Win.exe>)

- Para Linux (https://www.virtualbox.org/wiki/Linux_Downloads)

Instalar ISOs (Windows e Ubuntu)

- Windows (<https://www.microsoft.com/pt-br/software-download/windows10ISO>)

i Na ISO oficial do Windows, o link acima, vêm todas as versões.

- Ubuntu Desktop (<https://ubuntu.com/download/desktop>)

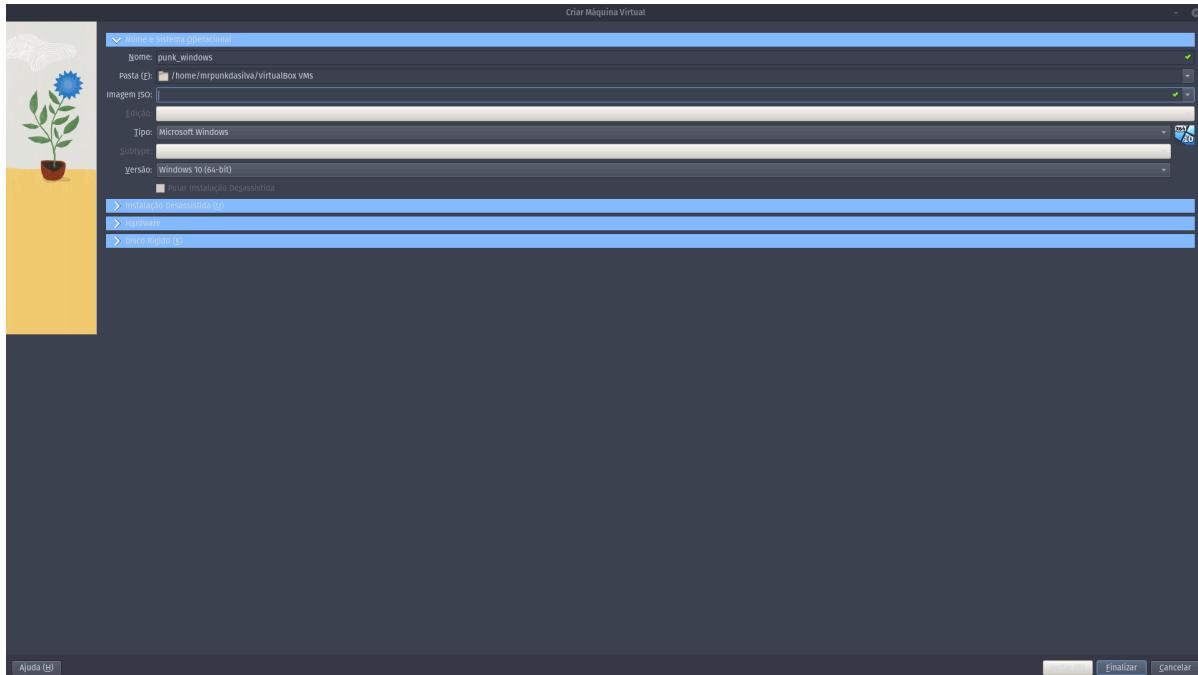
Criando Máquinas Virtuais

Windows

1. Com o VirtualBox aberto, pressione: **CTRL** + **N**

i Você pode acessar a opção também por: Machine > Add

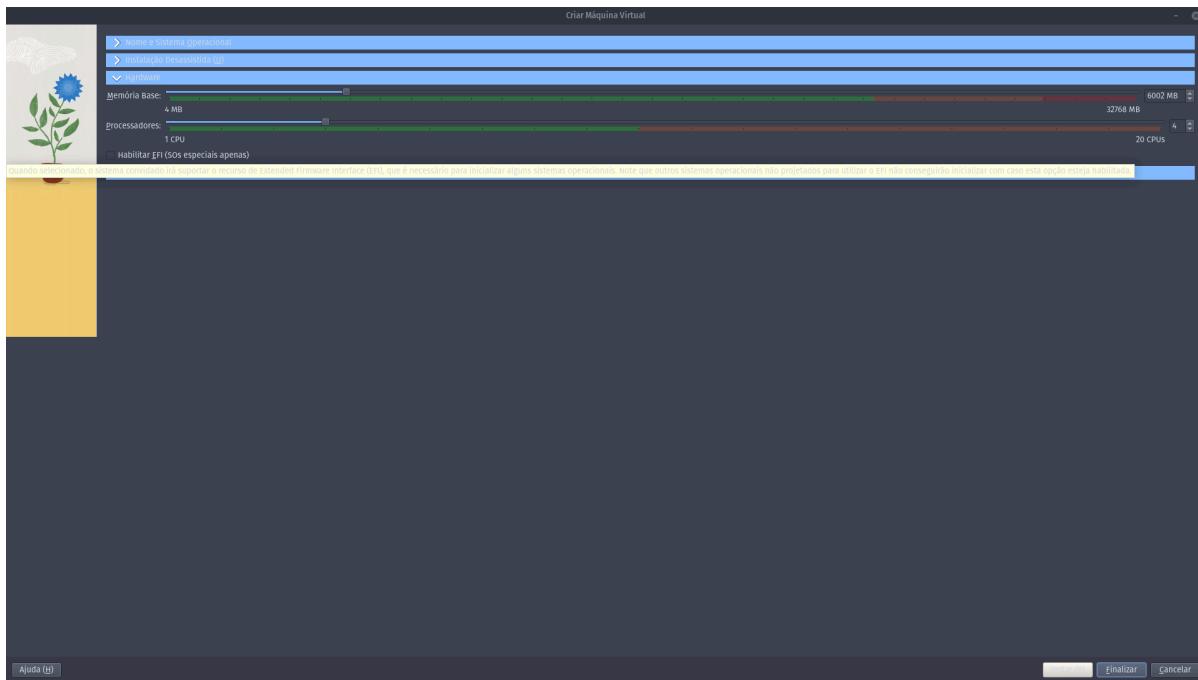
2. Definir nome, sistema e versão:



Tela de criação de máquina virtual no VirtualBox

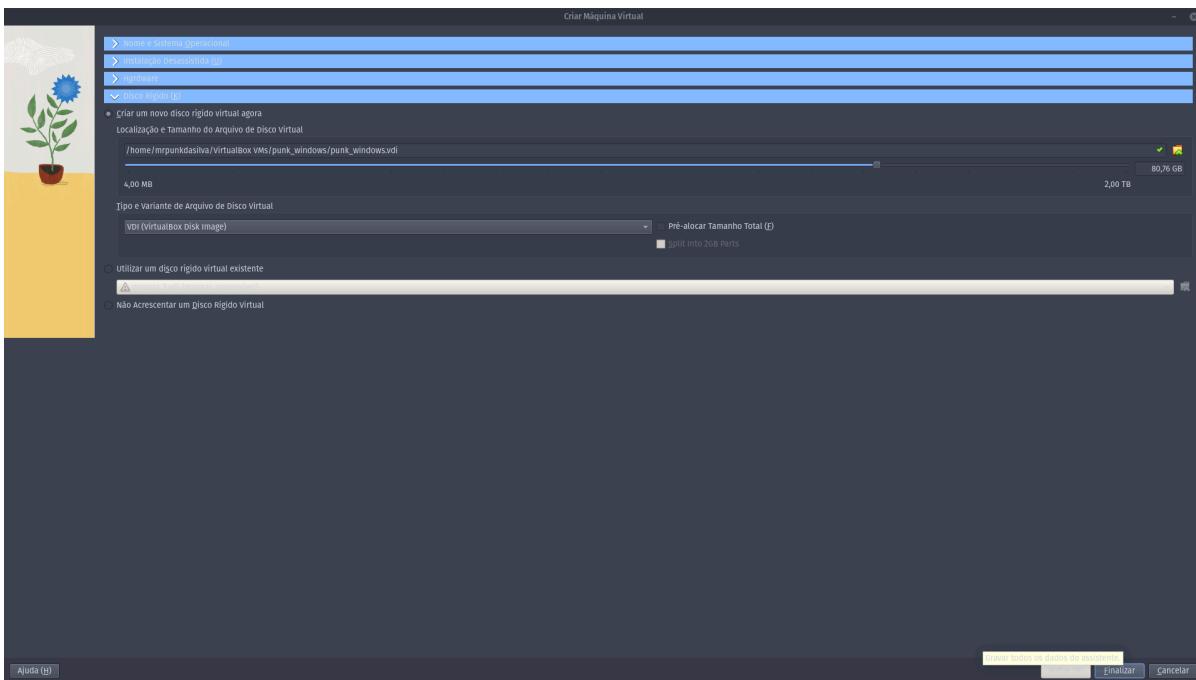
3. Agora vamos definir a memória RAM. É bom deixarmos no mínimo 4GB

i Atente-se que computadores não lidam bem com números ímpares.



Configuração de memória RAM para a máquina virtual

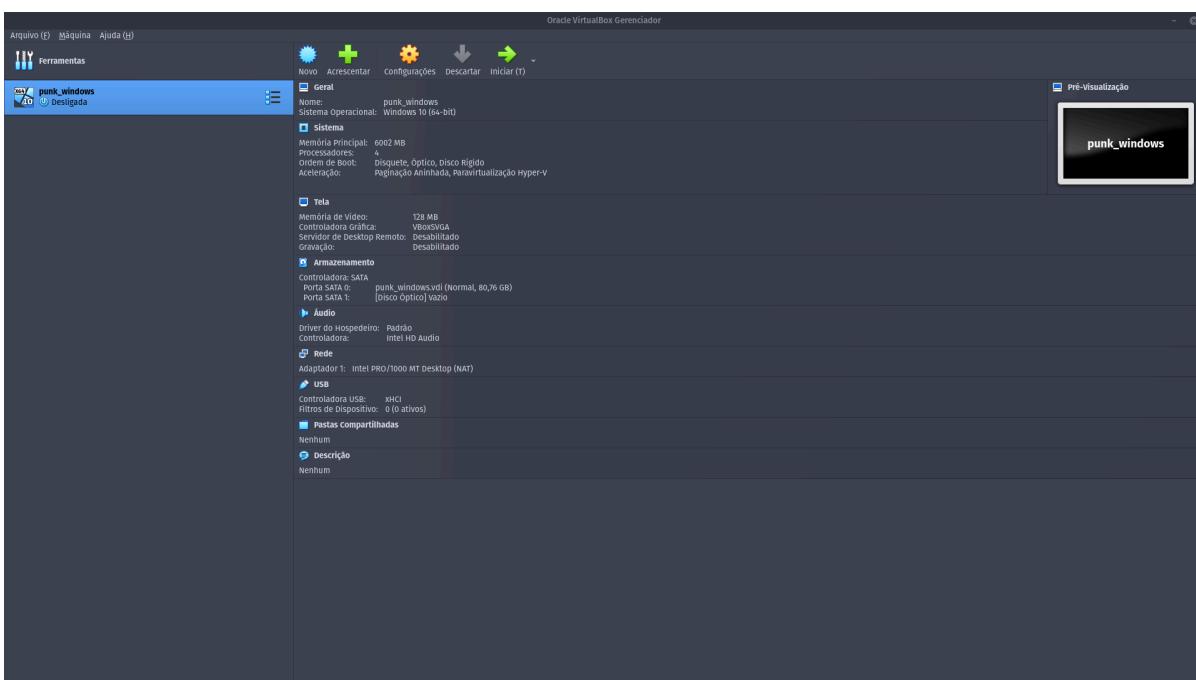
4. Agora definimos o espaço de armazenamento, disco rígido:



Configuração de disco rígido para a máquina virtual

5. Com tudo criado, basta ir em **Finish**:

6. Temos então nossa primeira máquina virtual criada:



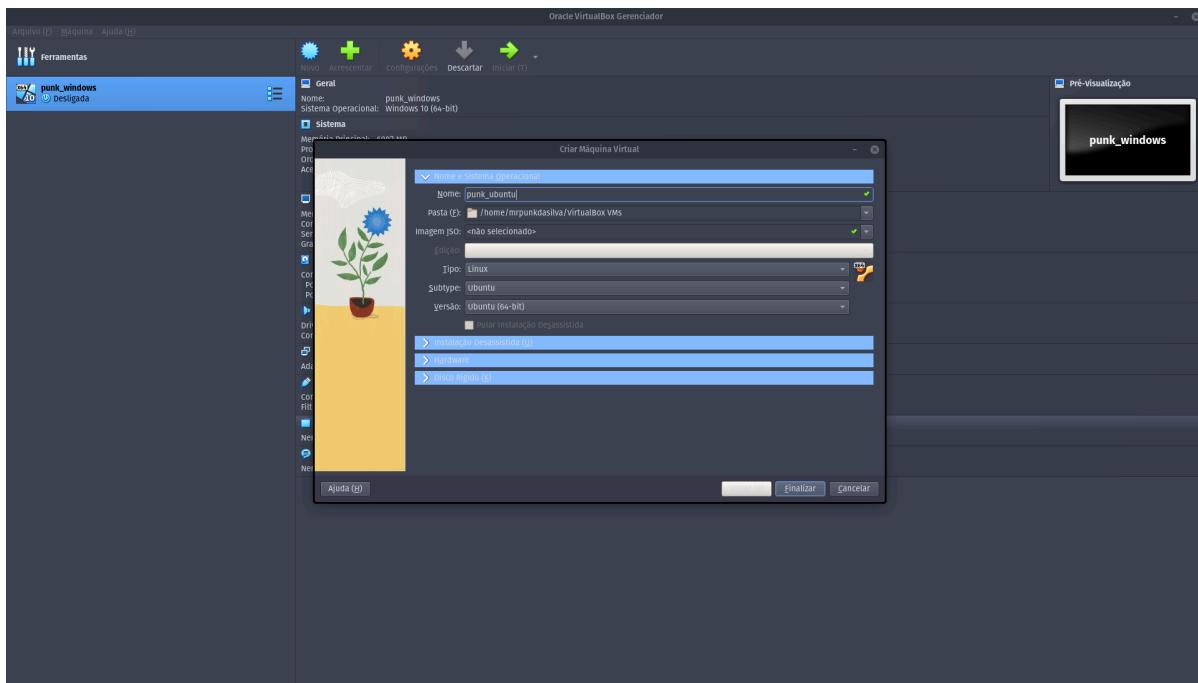
Máquina virtual Windows criada no VirtualBox

Ubuntu

1. Com o VirtualBox aberto, pressione: **CTRL** + **N**

i Você pode acessar a opção também por: Machine > Add

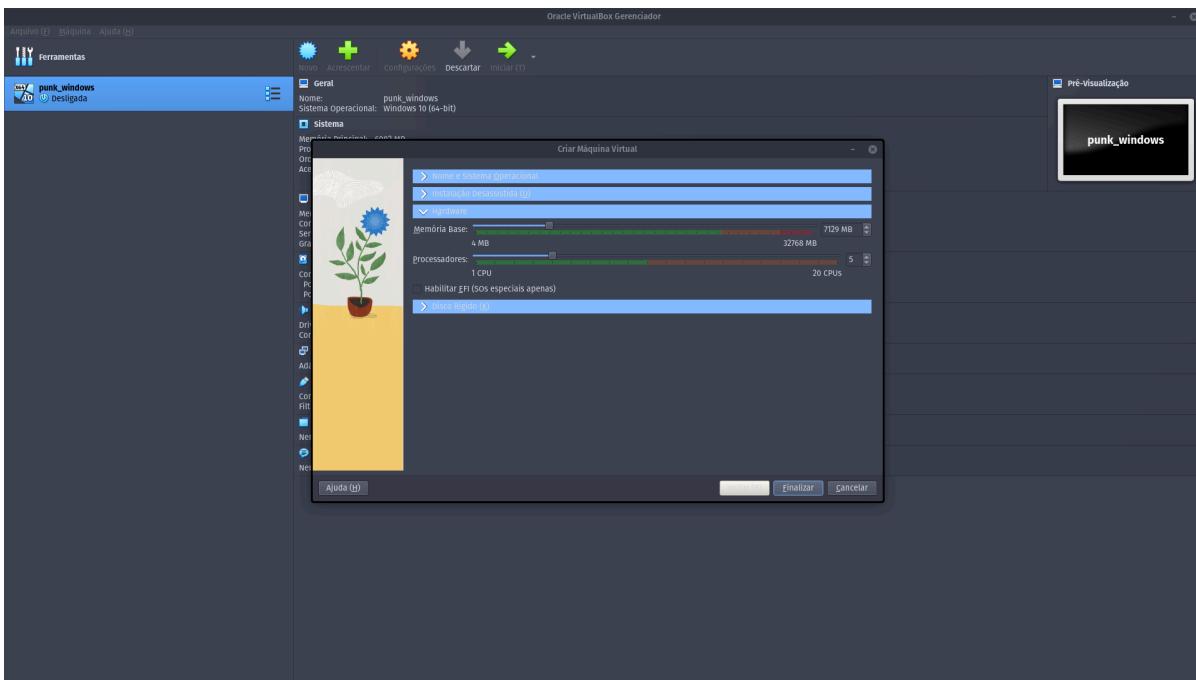
2. Definir nome, sistema e versão:



Tela de criação de máquina virtual Ubuntu no VirtualBox

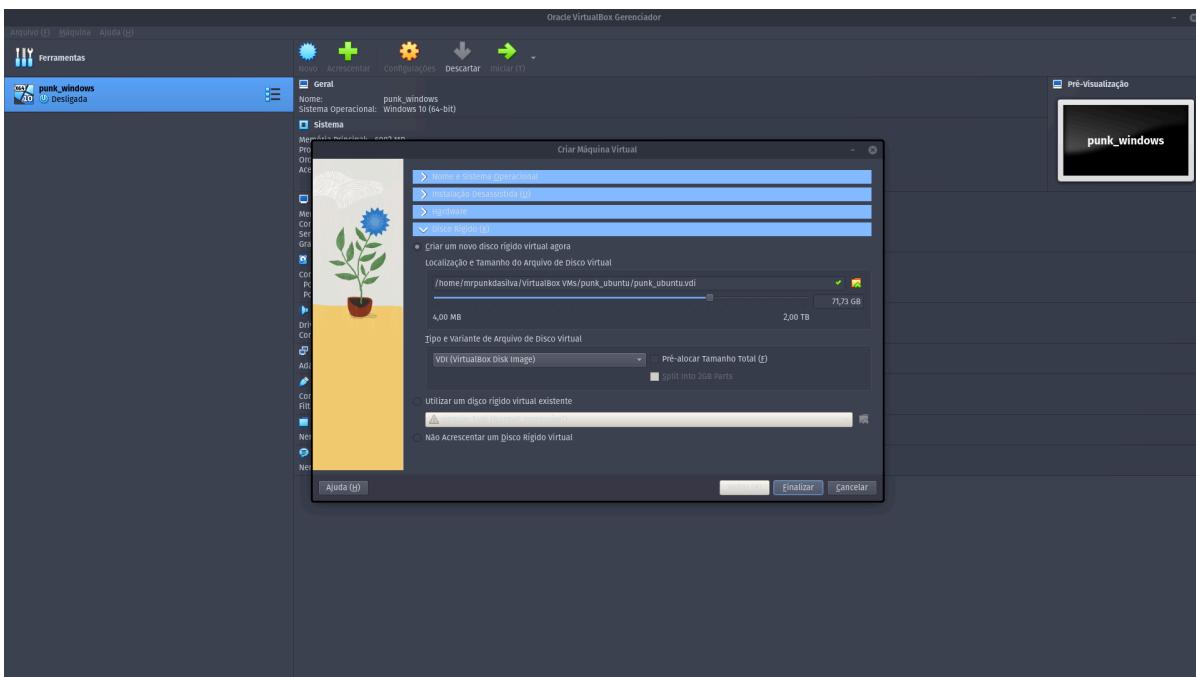
3. Agora vamos definir a memória RAM. É bom deixarmos no mínimo 4GB

i Atente-se que computadores não lidam bem com números ímpares.



Configuração de memória RAM para a máquina virtual Ubuntu

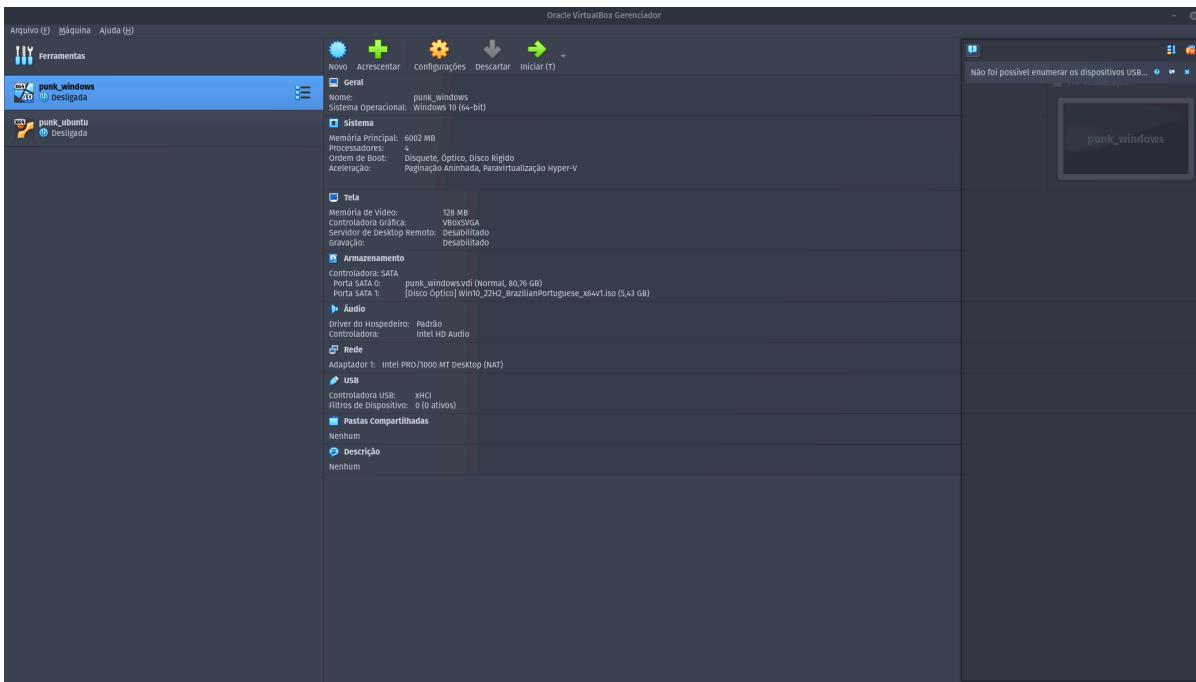
4. Agora definimos o espaço de armazenamento, disco rígido:



Configuração de disco rígido para a máquina virtual Ubuntu

5. Com tudo criado, basta ir em **Finish**:

6. Temos então nossa primeira máquina virtual criada:

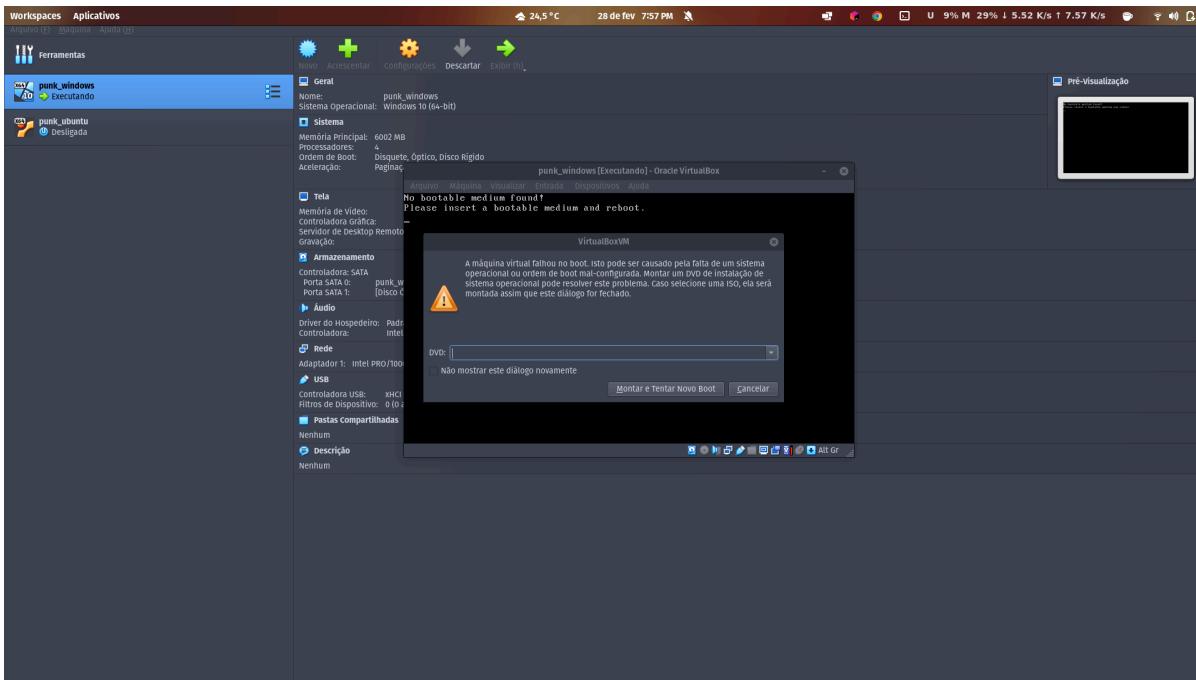


Máquina virtual Ubuntu criada no VirtualBox

Logar nas VMs recém-criadas

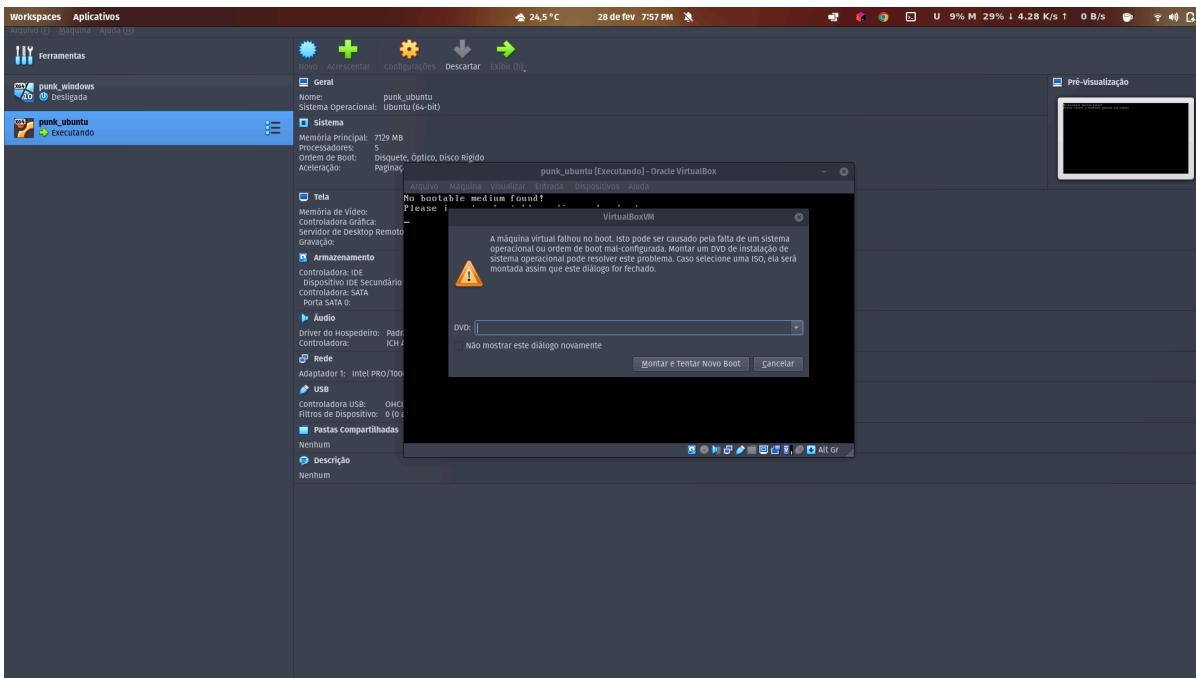
Ao executar as máquinas, nenhum sistema será inicializado, já que não foi definida nenhuma ISO (Imagem de um Sistema Operacional). Assim, as máquinas ficam em seu estado puro, sem nenhum sistema operacional, e são inutilizáveis.

Windows



Tela inicial da máquina virtual Windows sem sistema operacional

Ubuntu



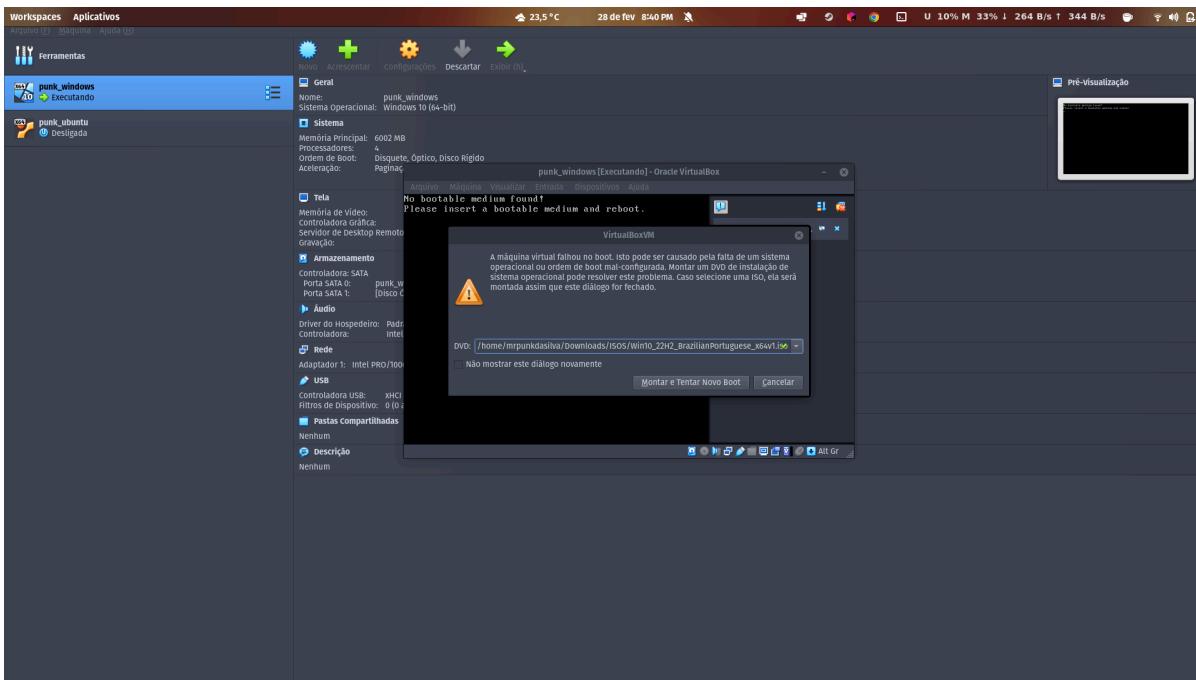
Tela inicial da máquina virtual Ubuntu sem sistema operacional

Configurando VMs para os SOs

Para tornar as VMs utilizáveis, precisamos definir as ISOs que serão as imagens do sistema usadas para instalar o sistema operacional.

Windows

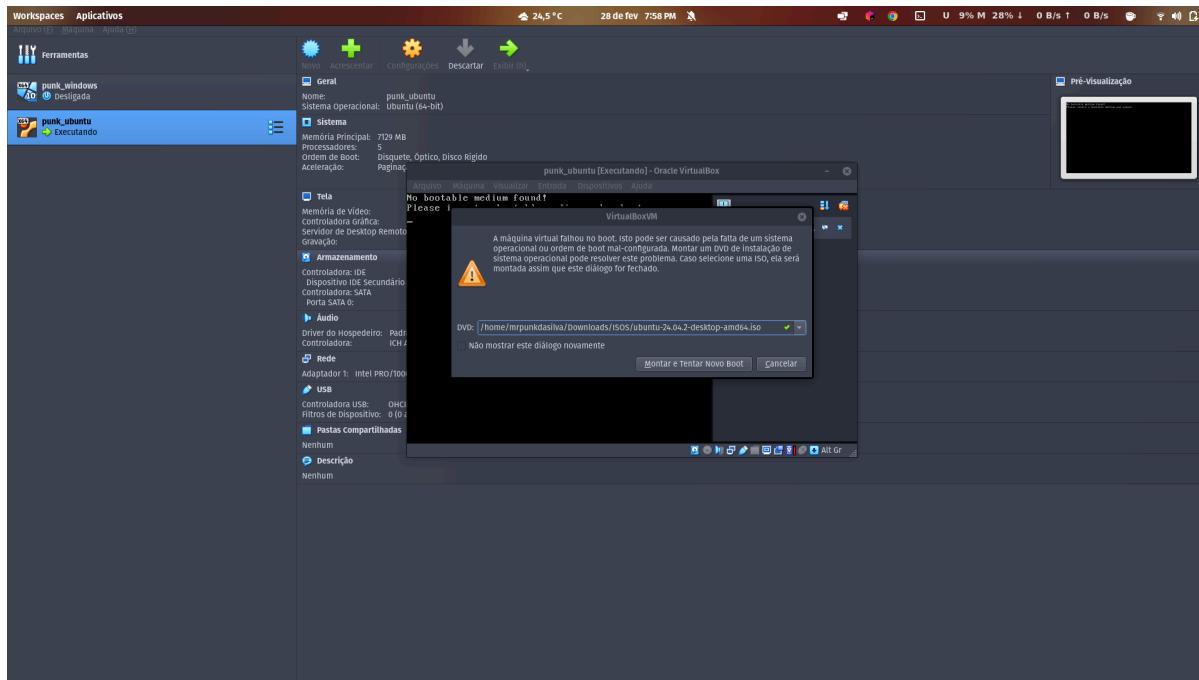
Com a máquina em execução e este pop-up aparecendo, selecionamos onde está a ISO do Windows que foi baixada nos passos anteriores:



Seleção da ISO do Windows para instalação

Ubuntu

Com a máquina em execução e este pop-up aparecendo, selecionamos onde está a ISO do Ubuntu que foi baixada nos passos anteriores:

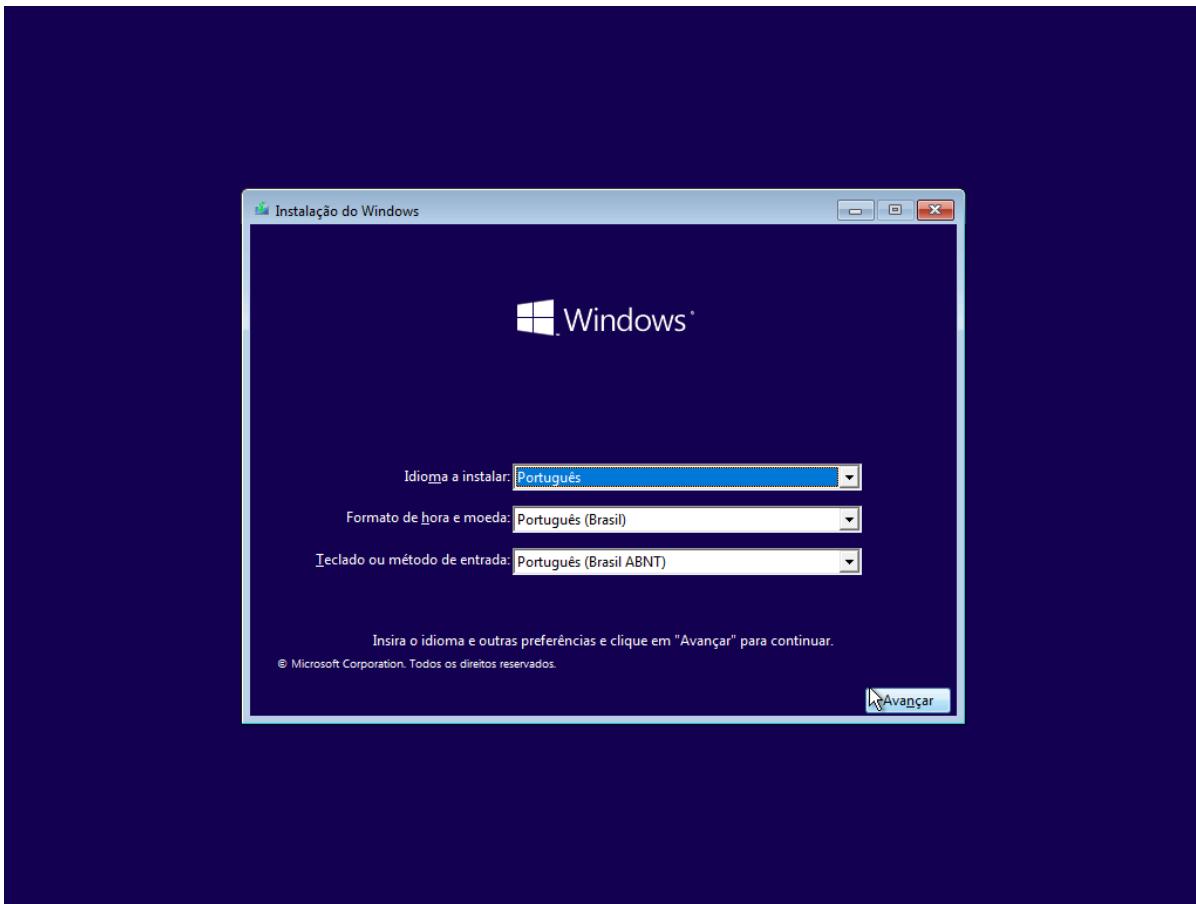


Seleção da ISO do Ubuntu para instalação

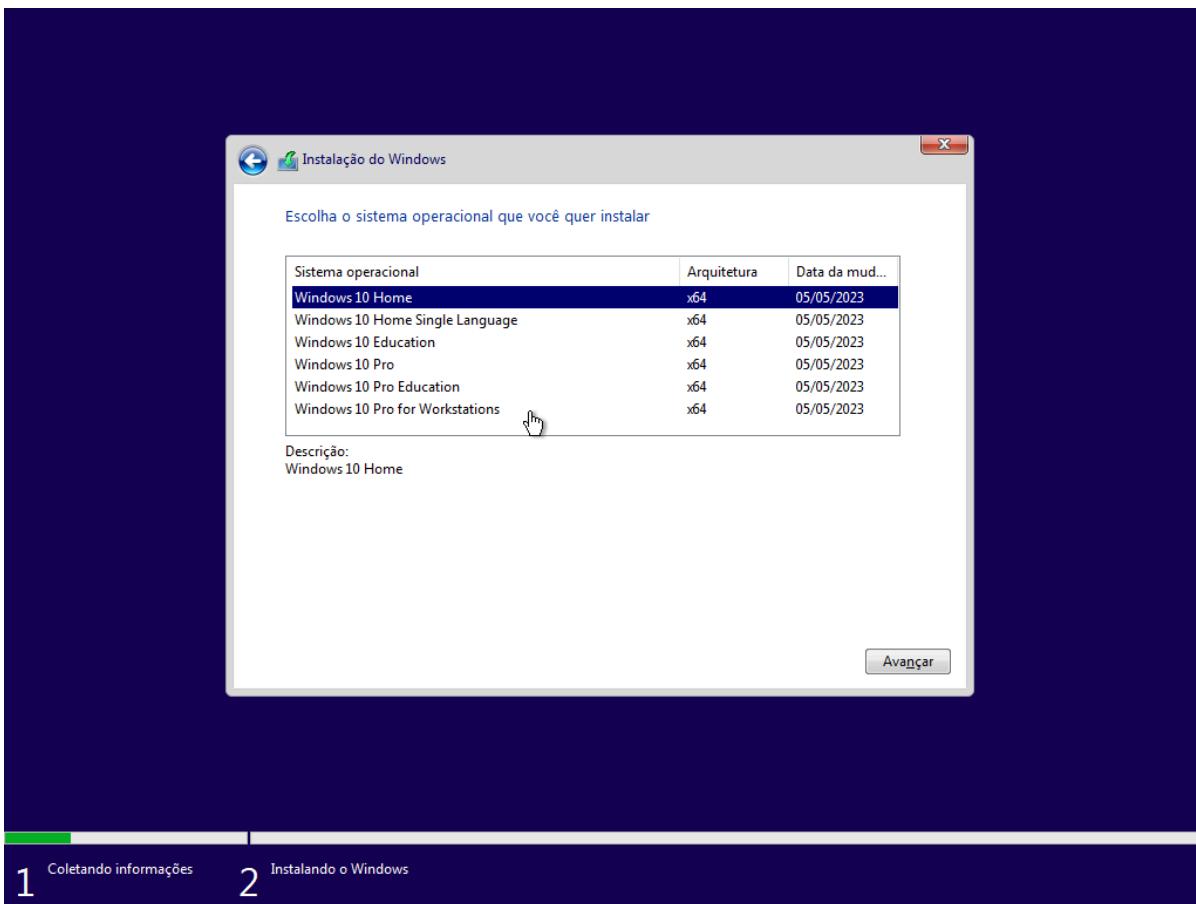
Logar nas VMs

Agora, com tudo o que vimos, podemos fazer a instalação do sistema. Para isso, devemos logar ou entrar nas máquinas que criamos e então fazer as etapas de instalação do Windows e Ubuntu.

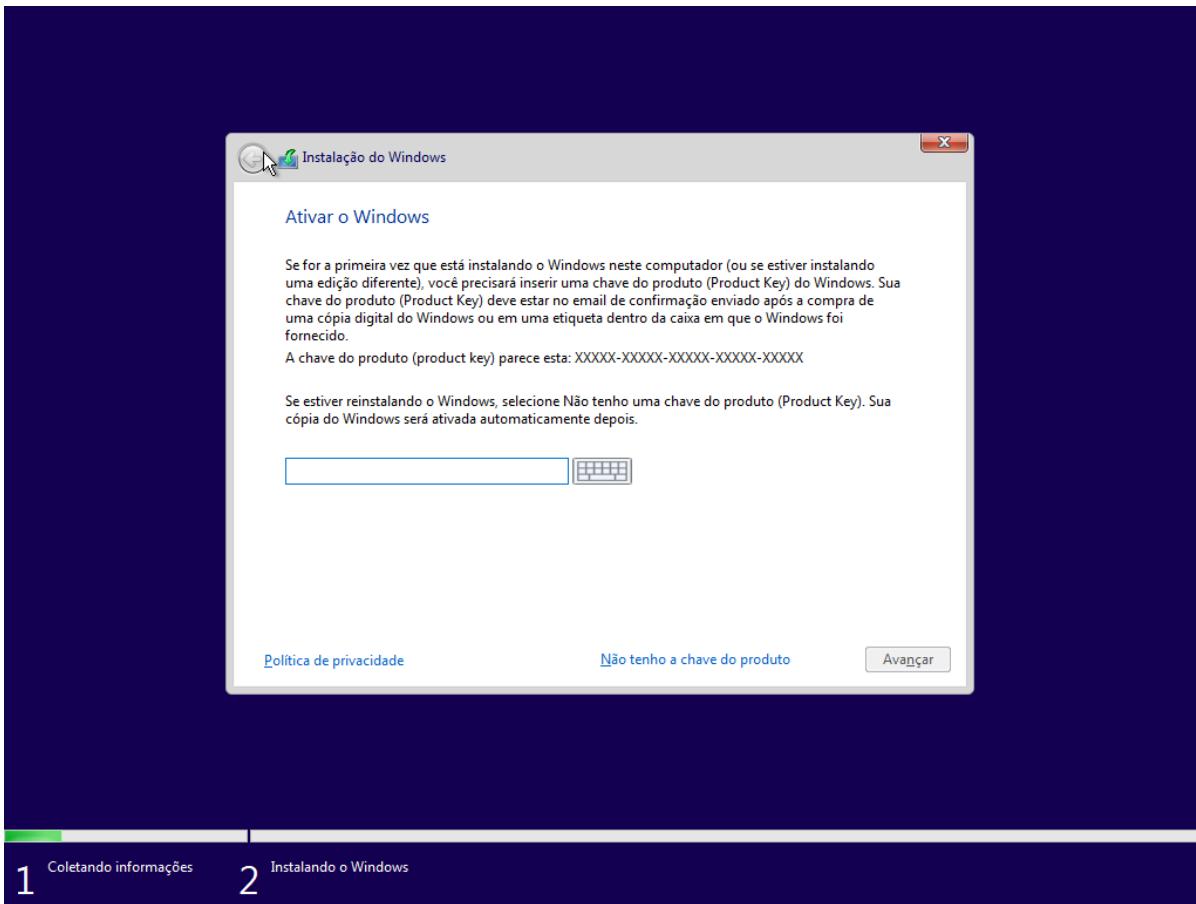
Windows



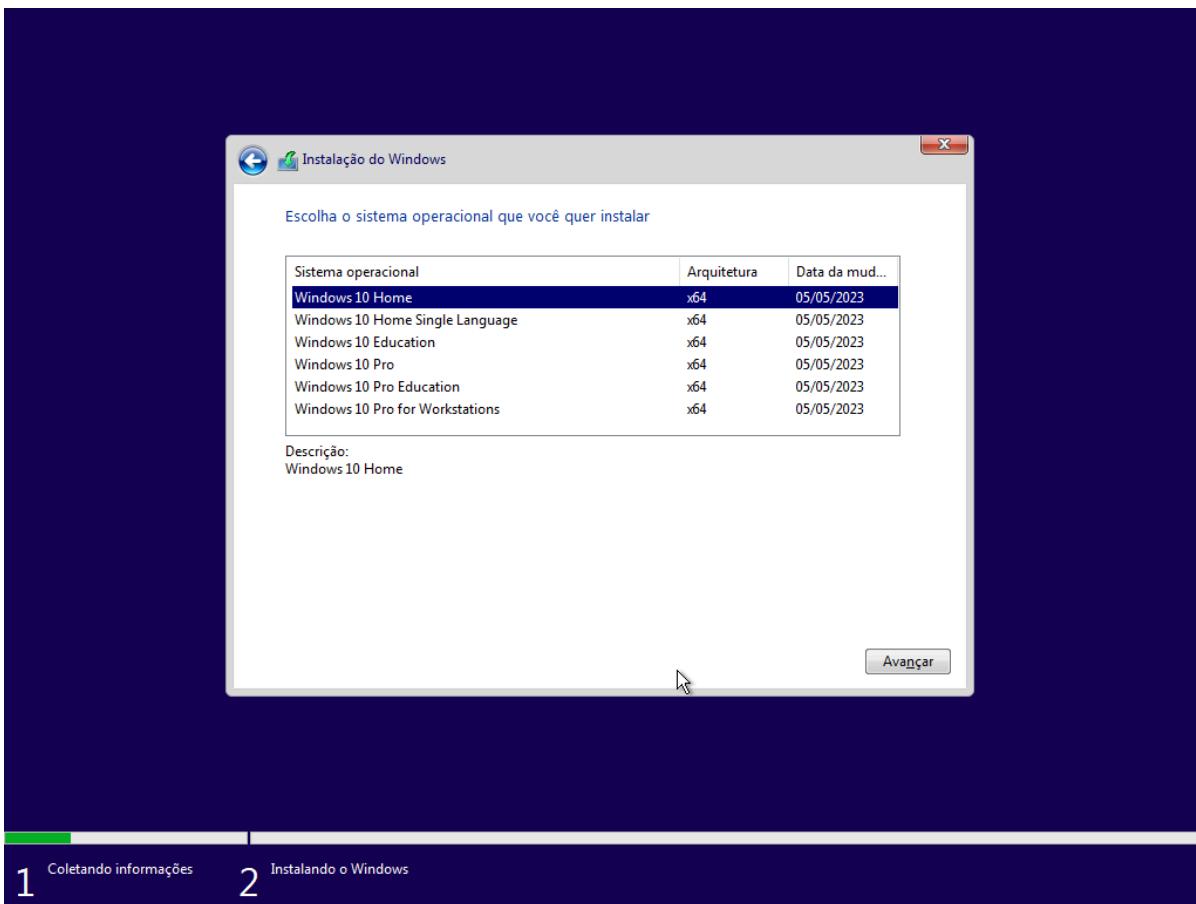
Tela inicial de instalação do Windows



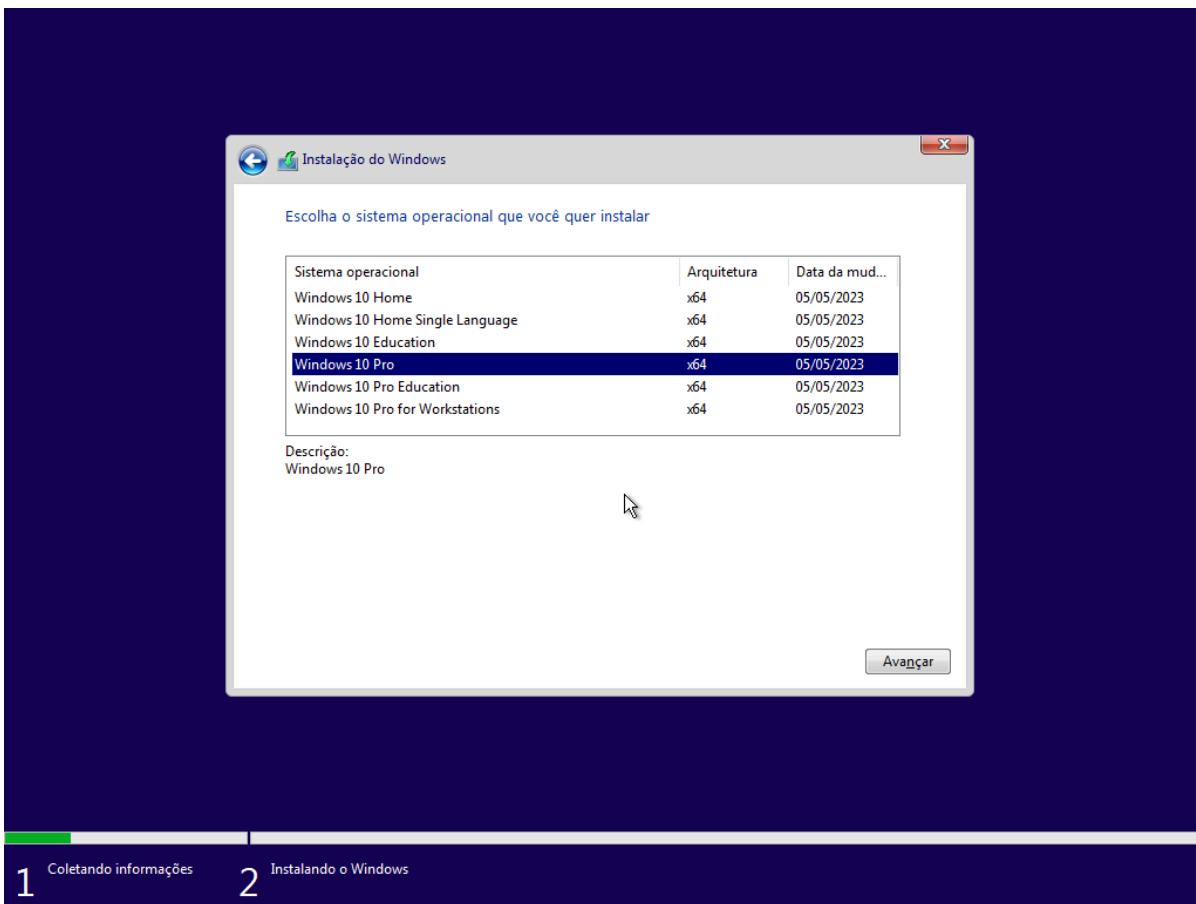
Seleção de idioma, formato de hora e moeda, e layout de teclado



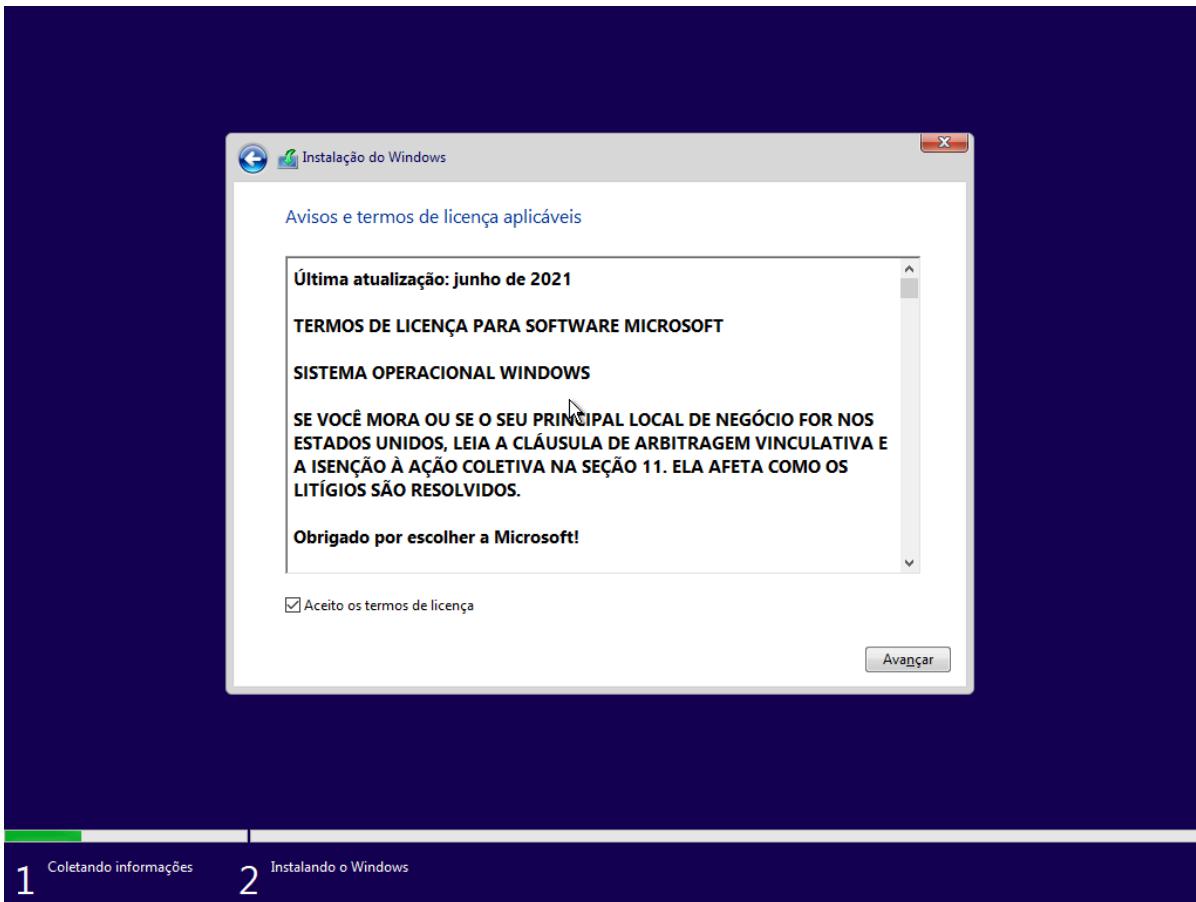
Botão "Instalar agora" para iniciar a instalação do Windows



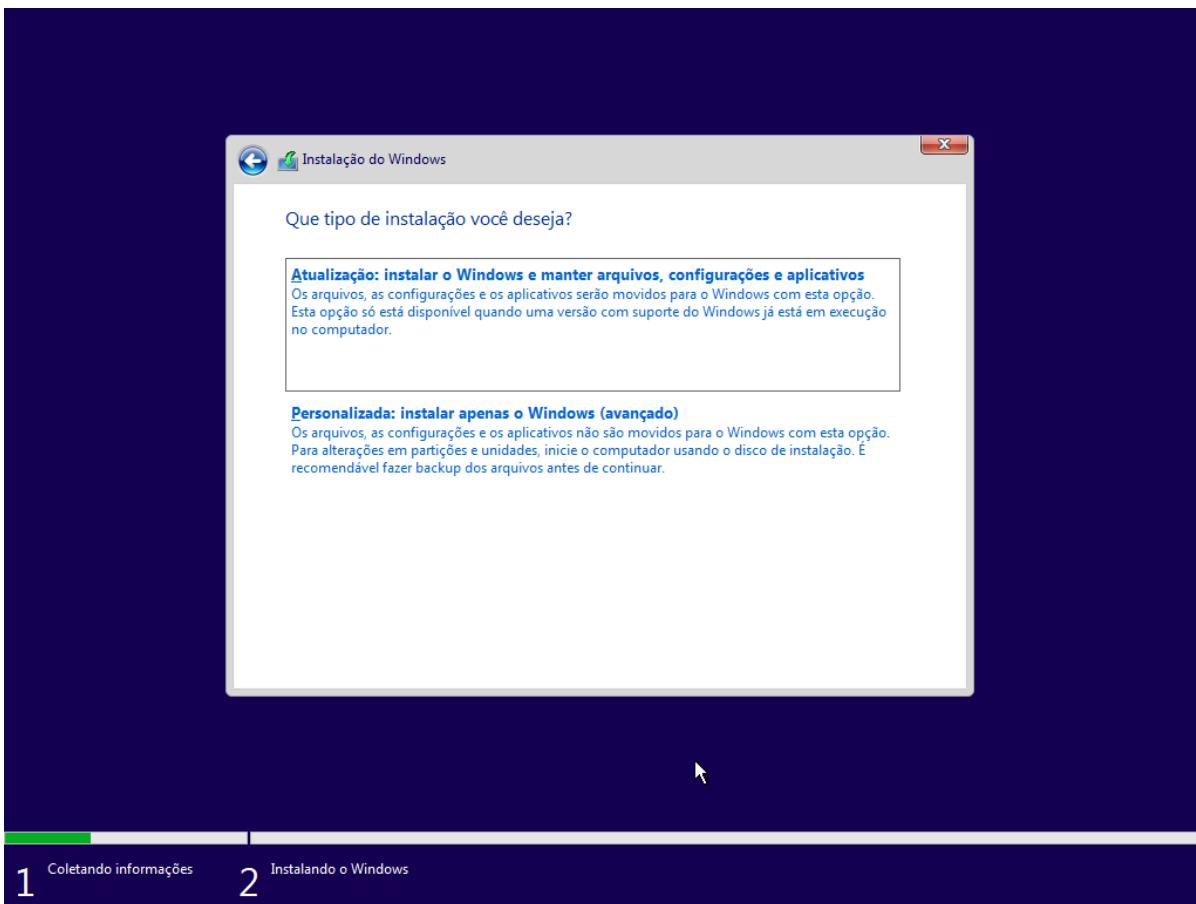
Inserção da chave do produto Windows



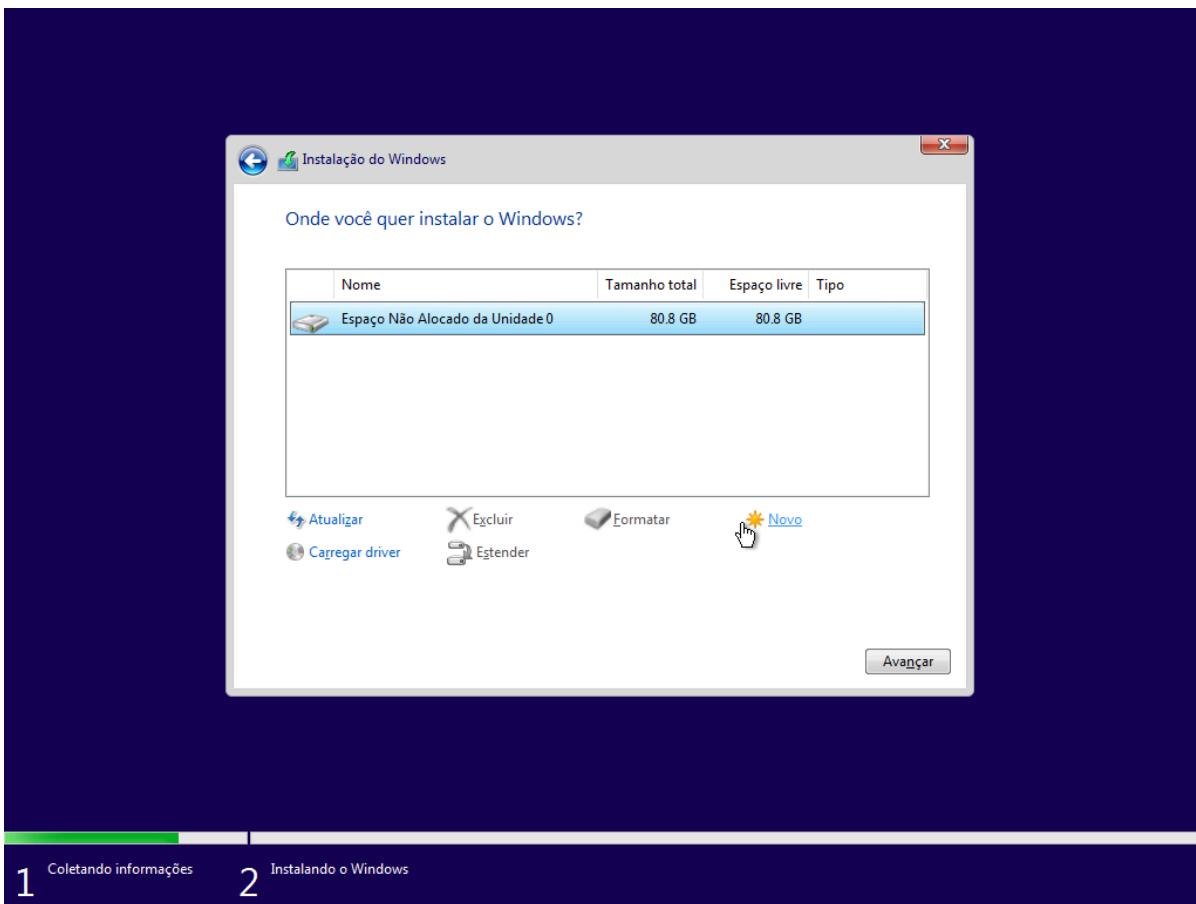
Seleção da versão do Windows a ser instalada



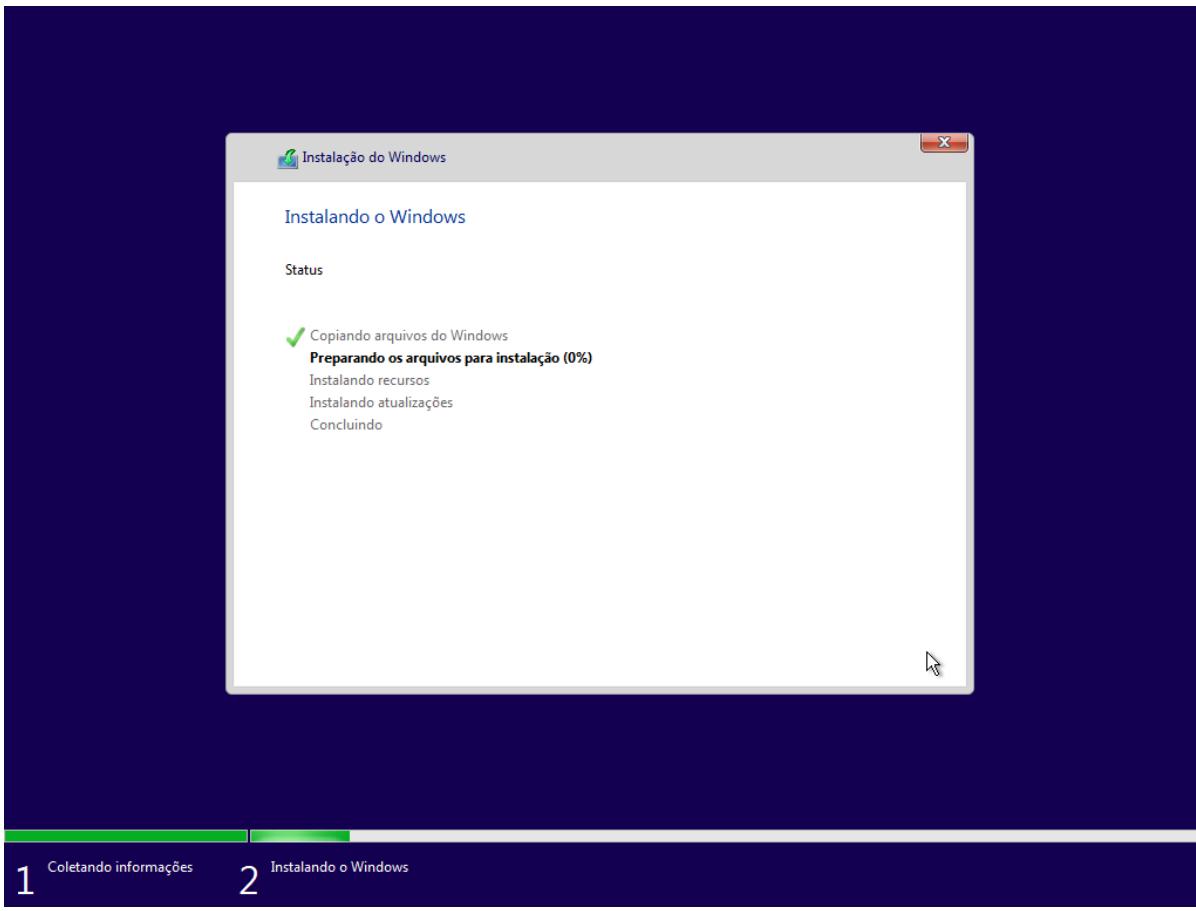
Aceitação dos termos de licença do Windows



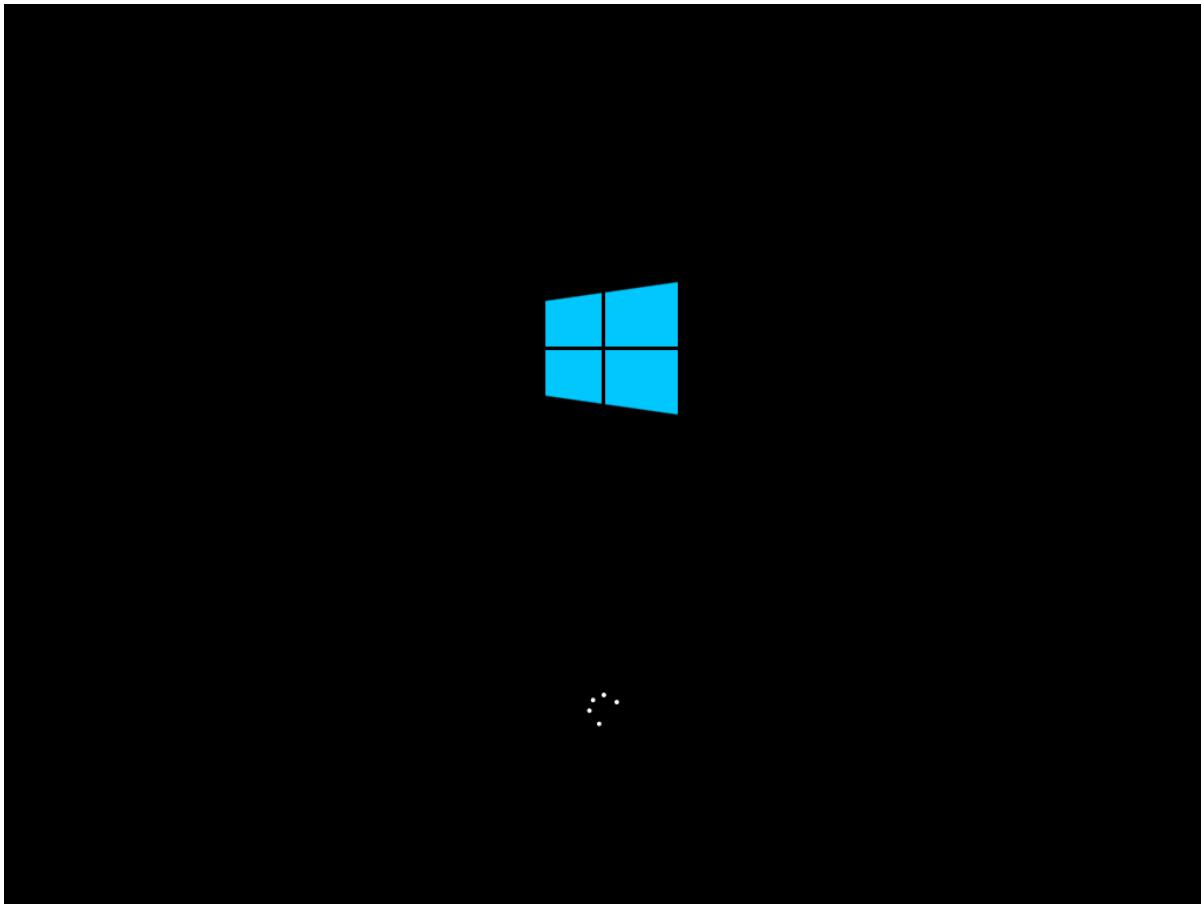
Escolha do tipo de instalação: Atualização ou Personalizada



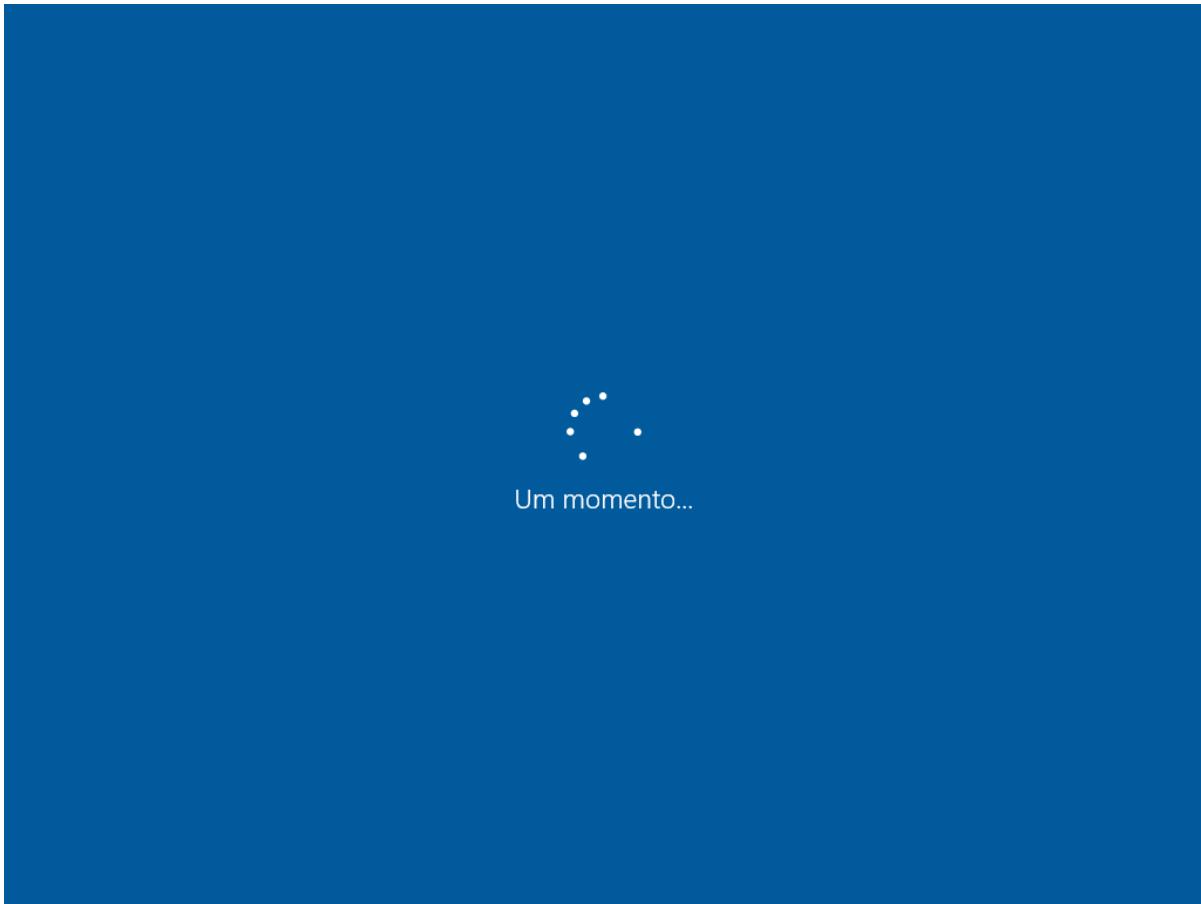
Seleção do disco onde o Windows será instalado



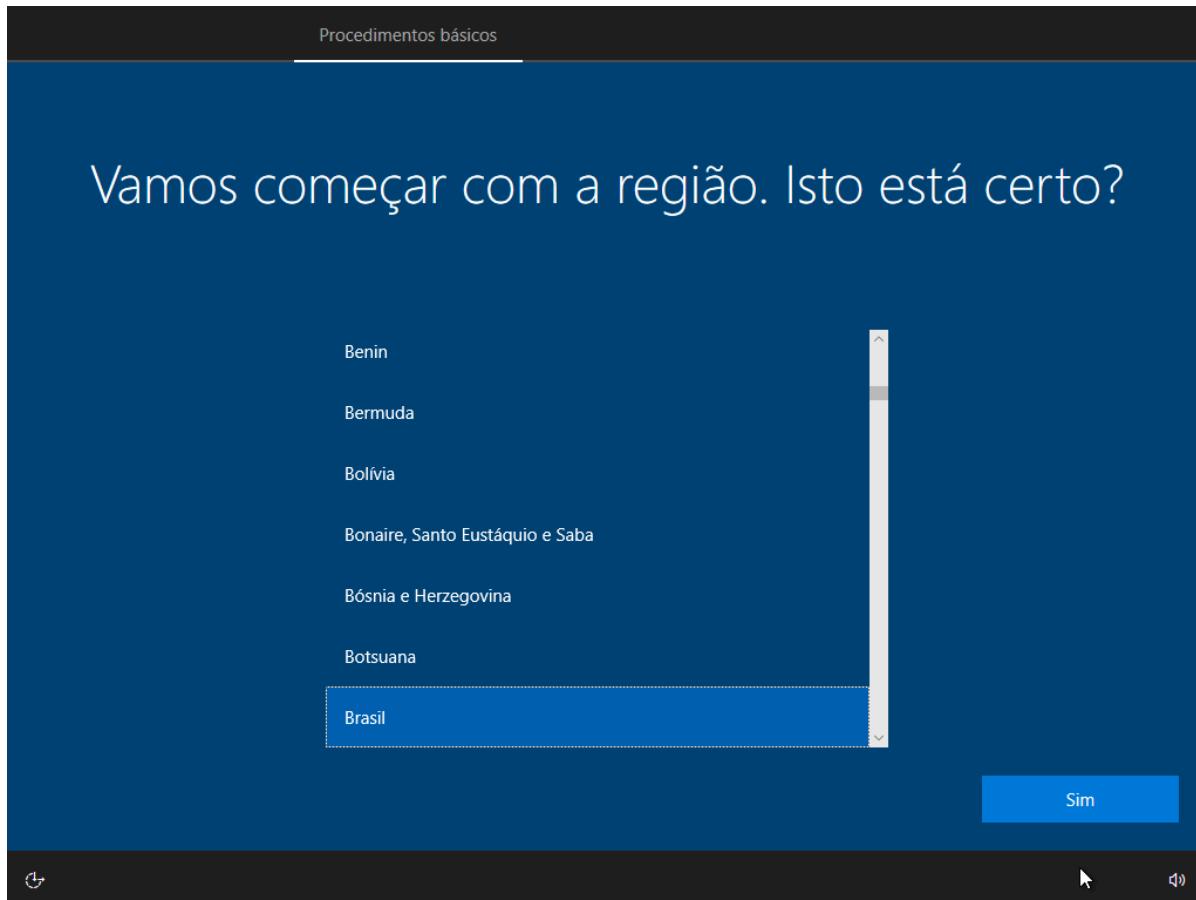
Progresso da instalação do Windows



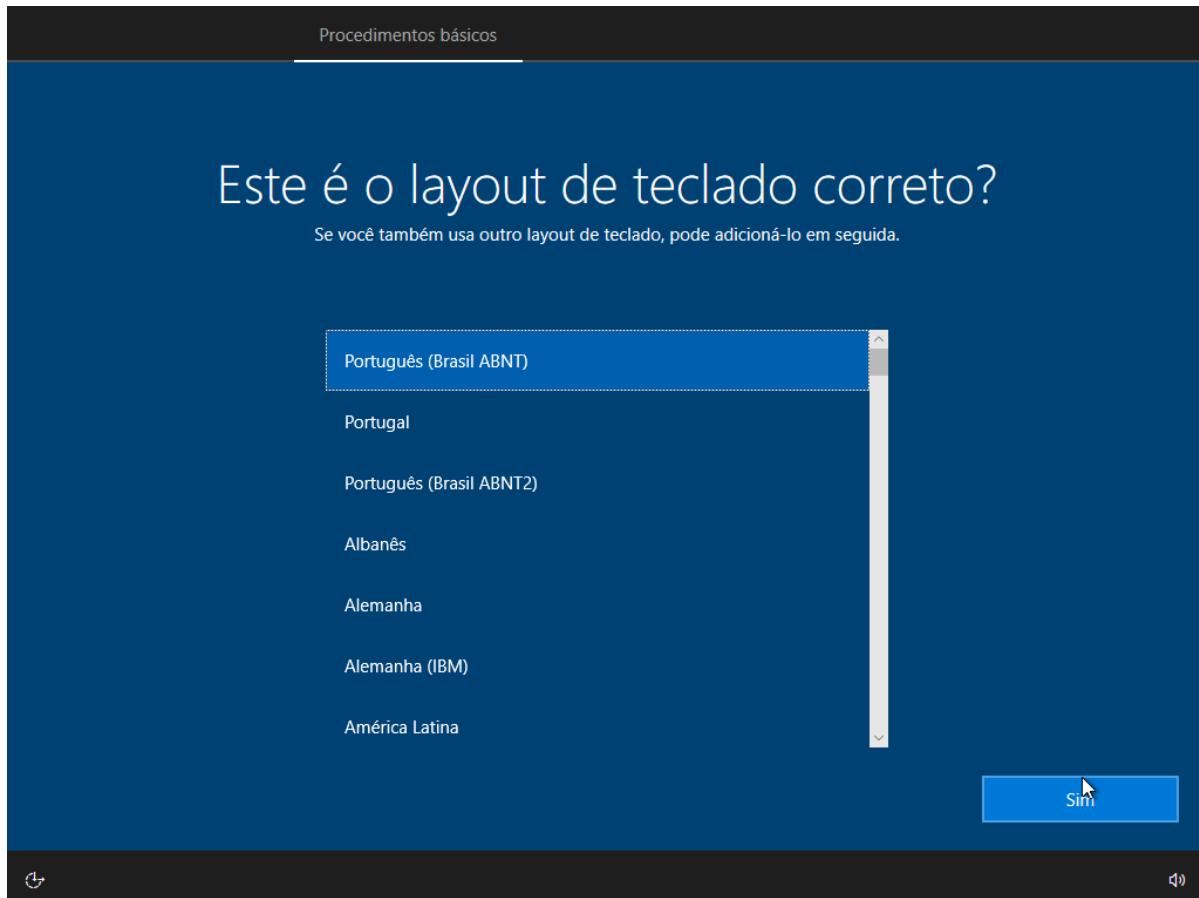
Reinicialização do sistema após a instalação inicial



Configuração inicial: Seleção de região



Configuração inicial: Confirmação do layout de teclado



Configuração inicial: Opção de adicionar um segundo layout de teclado

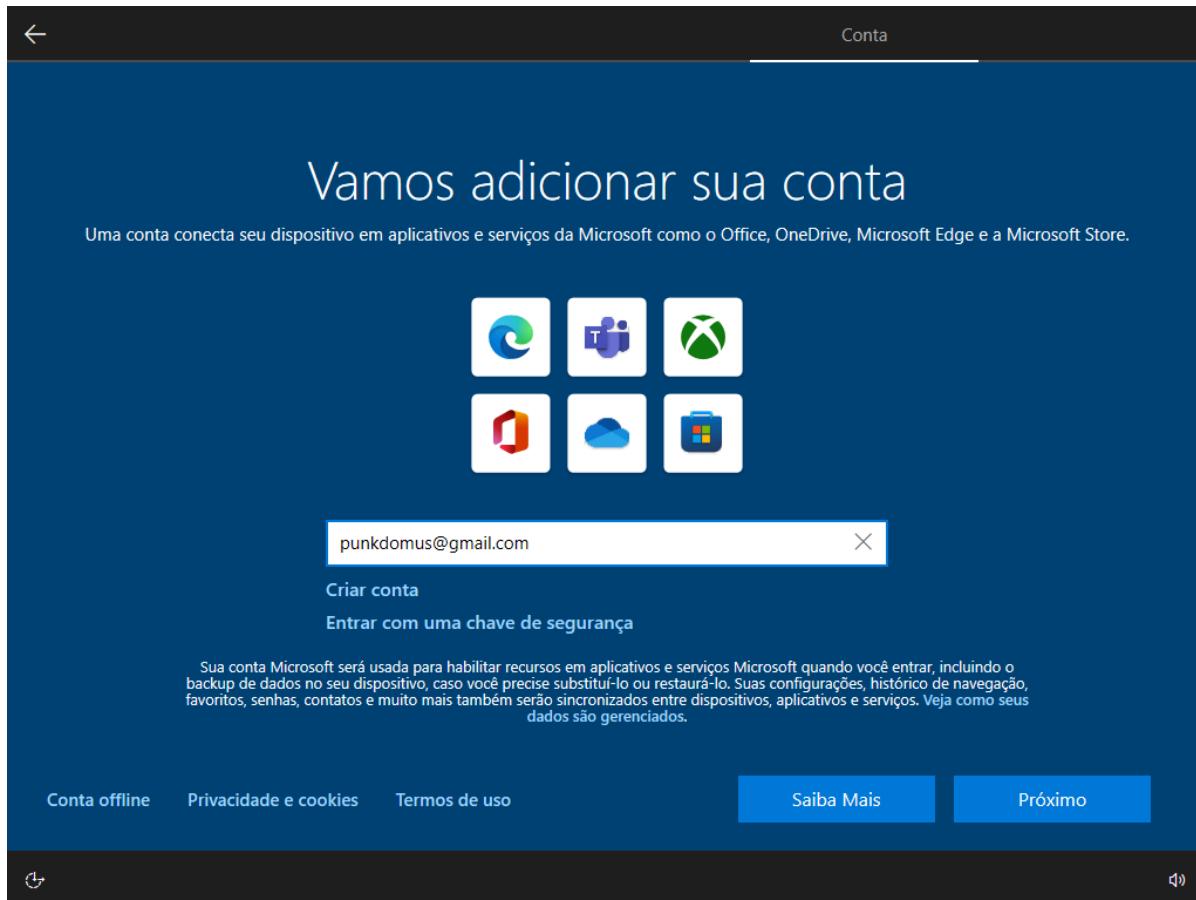
Como você gostaria de configurar?

Configurar para uso pessoal
Iremos ajudá-lo a configurar com uma conta pessoal da Microsoft. Você terá controle total sobre este dispositivo.

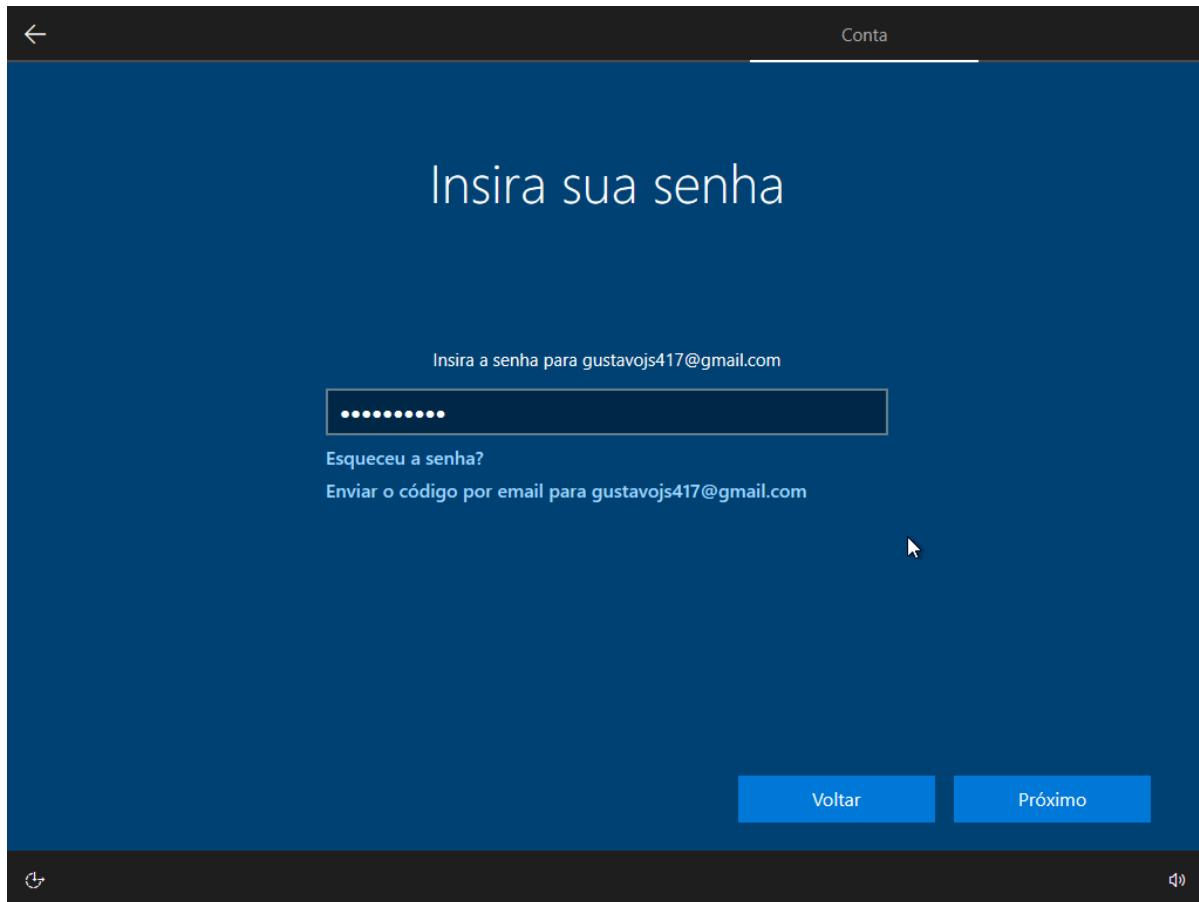
Configurar para uma organização
Você terá acesso aos recursos de sua organização como email, rede, aplicativos e serviços. Sua organização terá controle total sobre este dispositivo.

Próximo

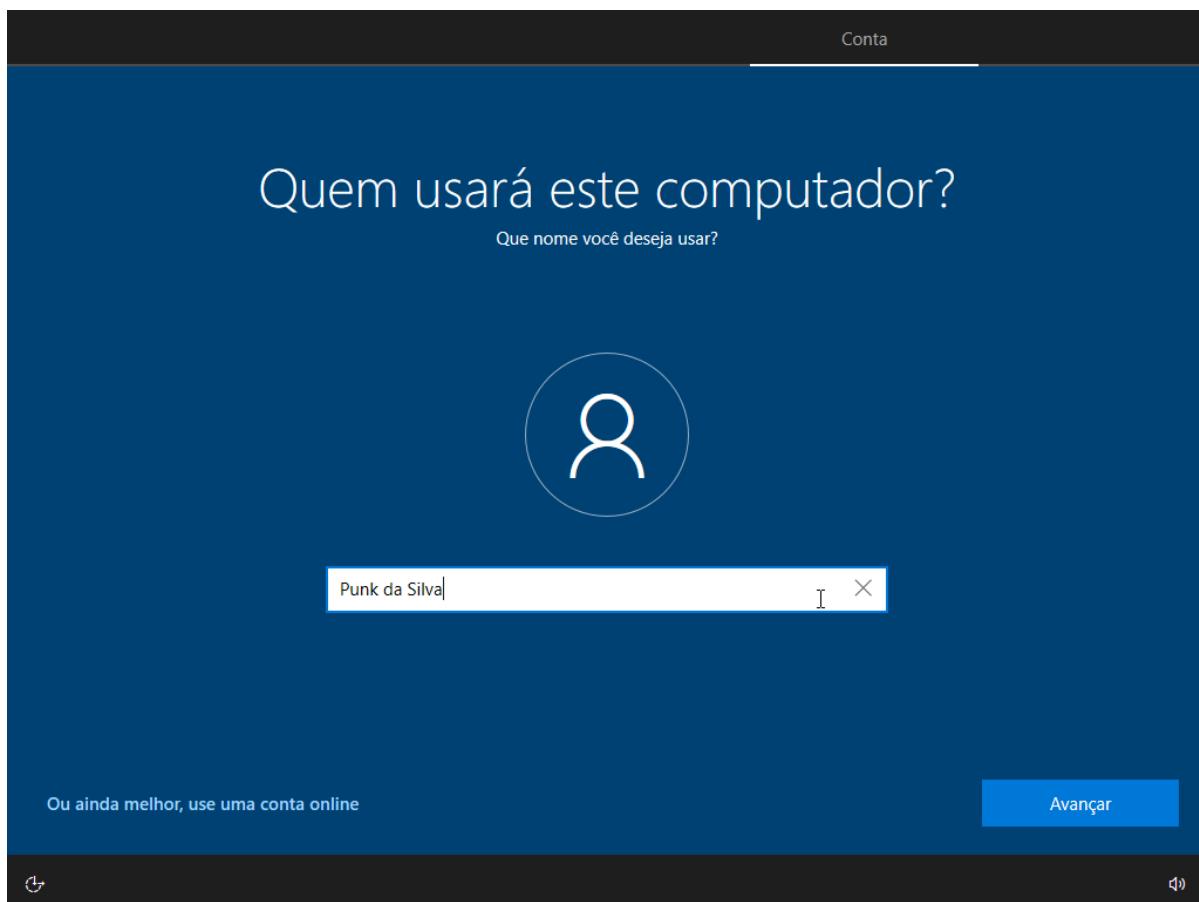
Configuração de rede: Conexão à internet



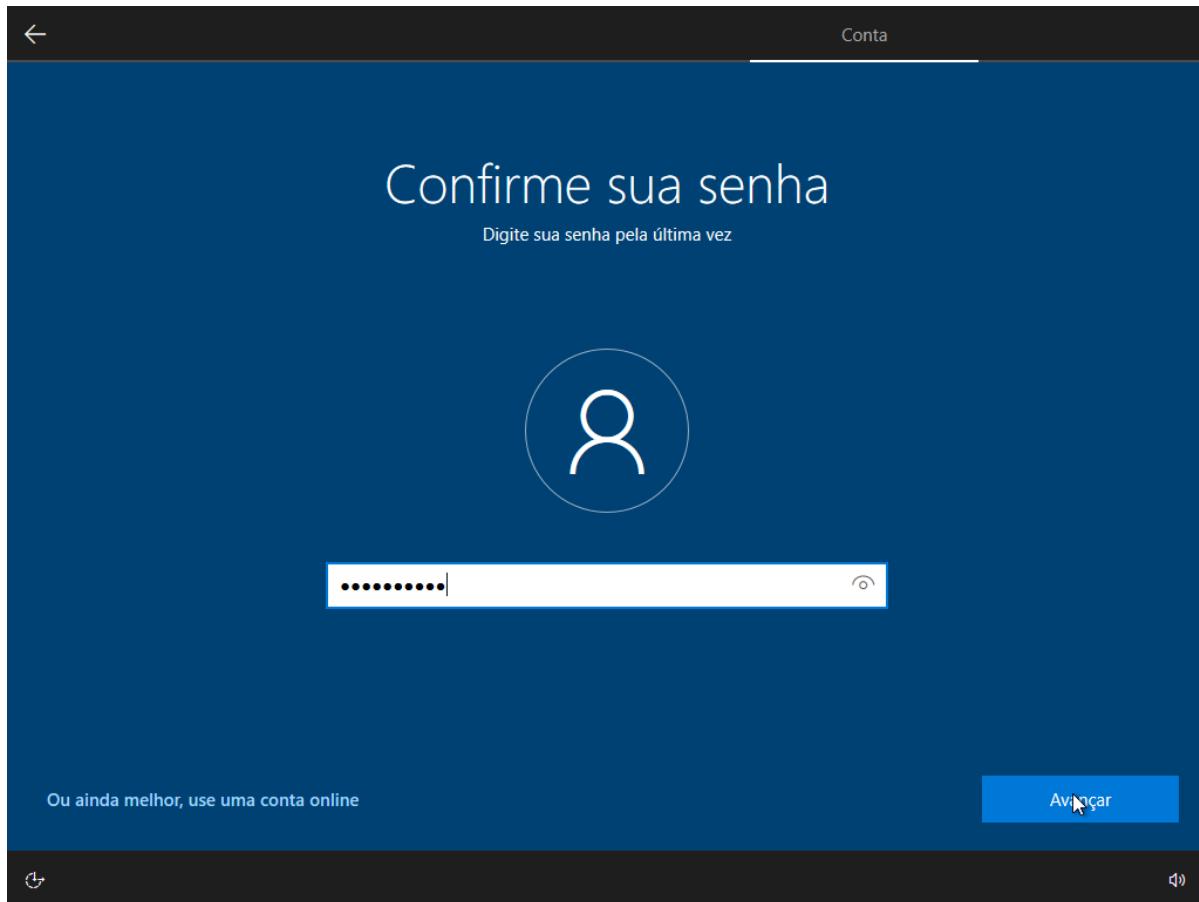
Configuração de conta: Opções de login



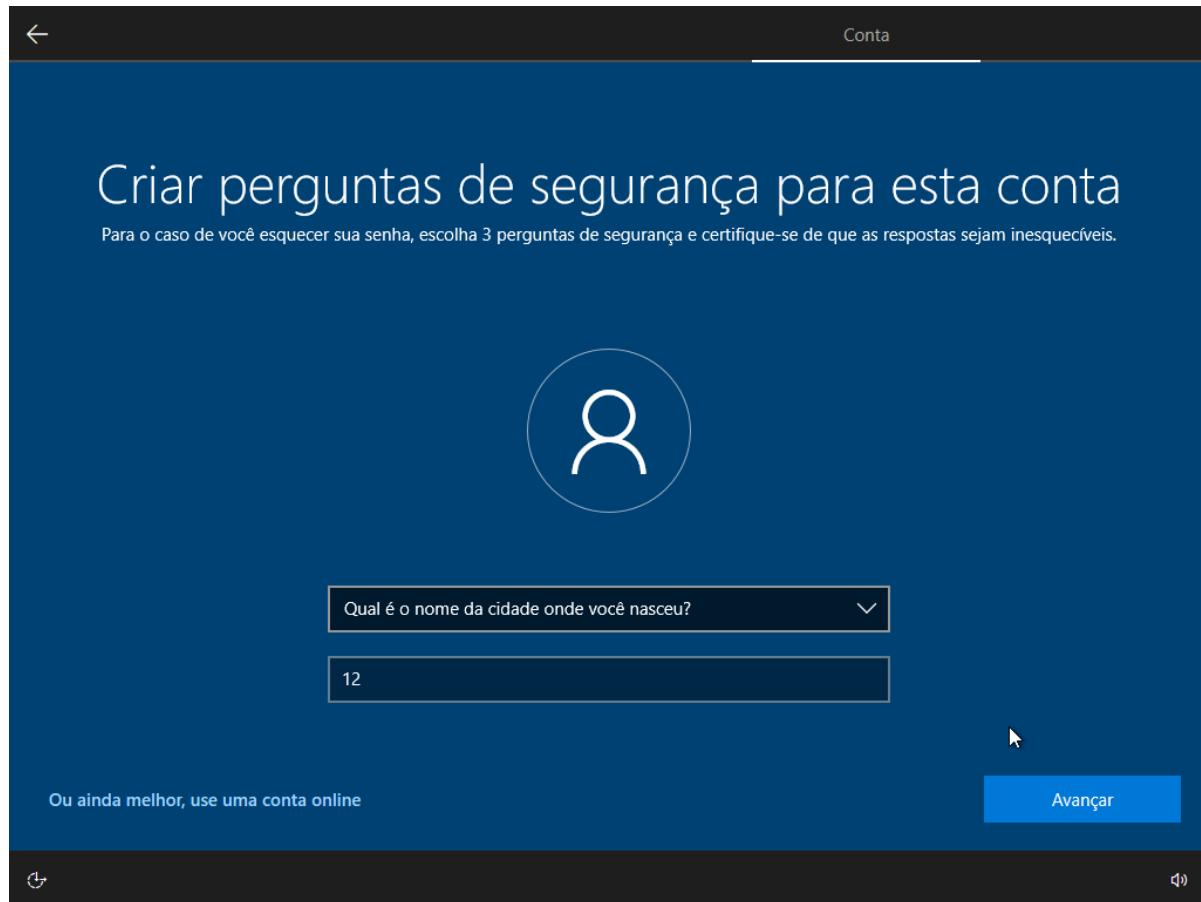
Definição de senha para a conta local



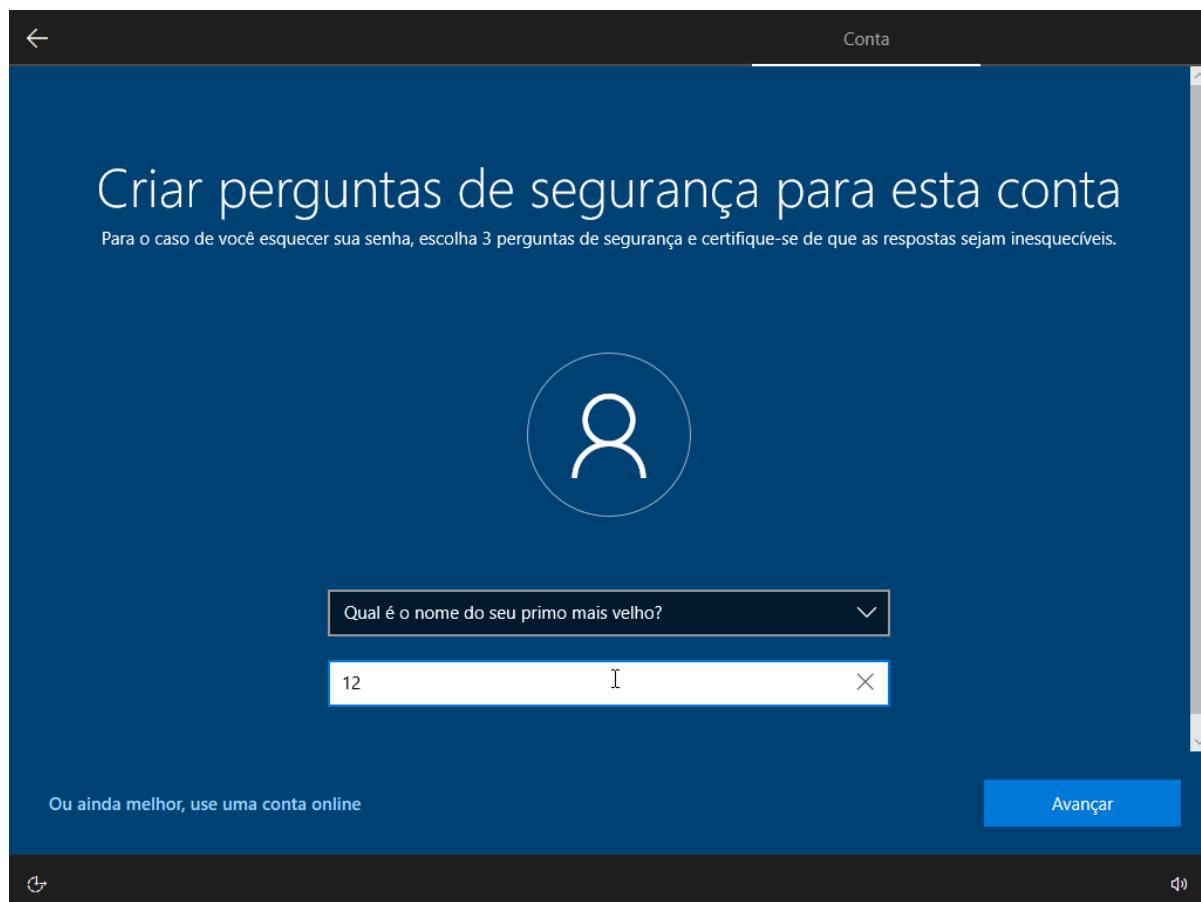
Configuração de perguntas de segurança



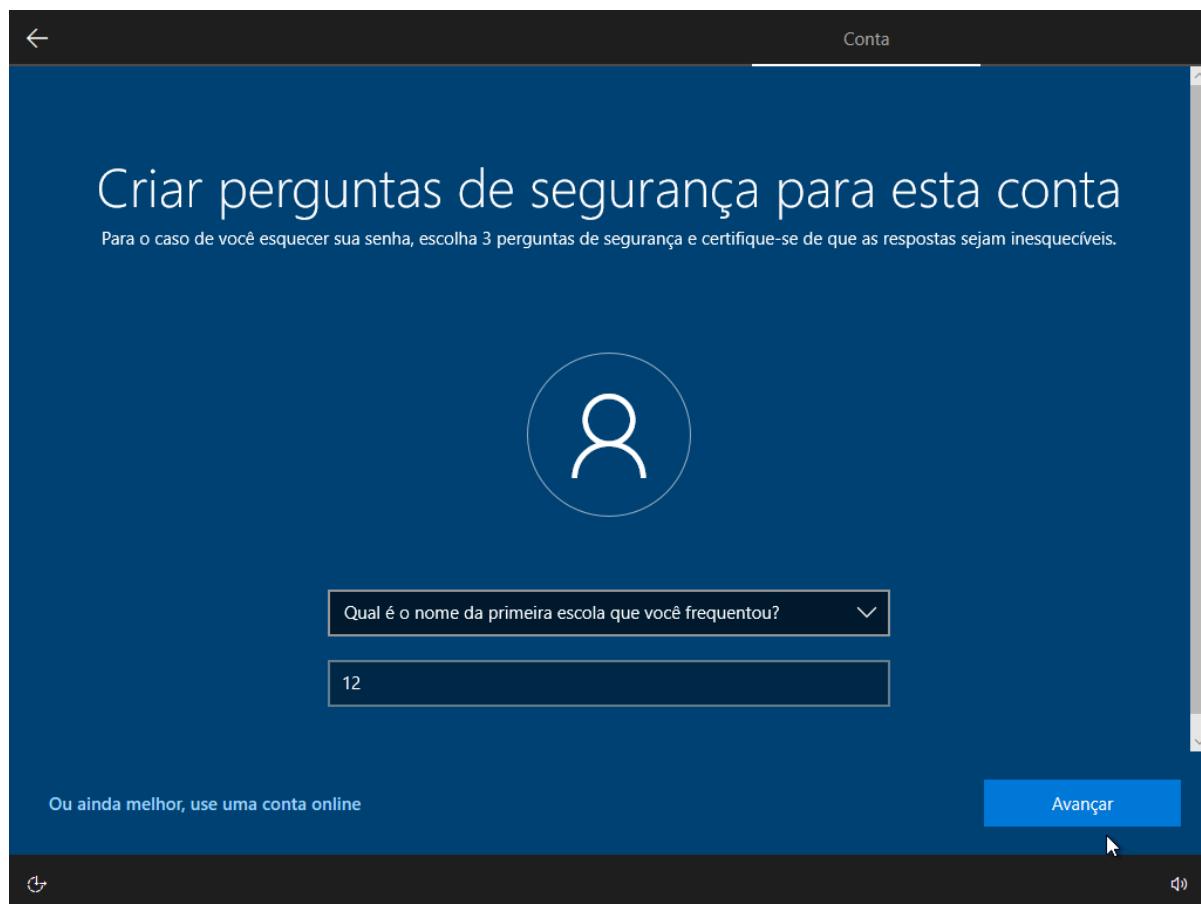
Configurações de privacidade: Escolha de permissões



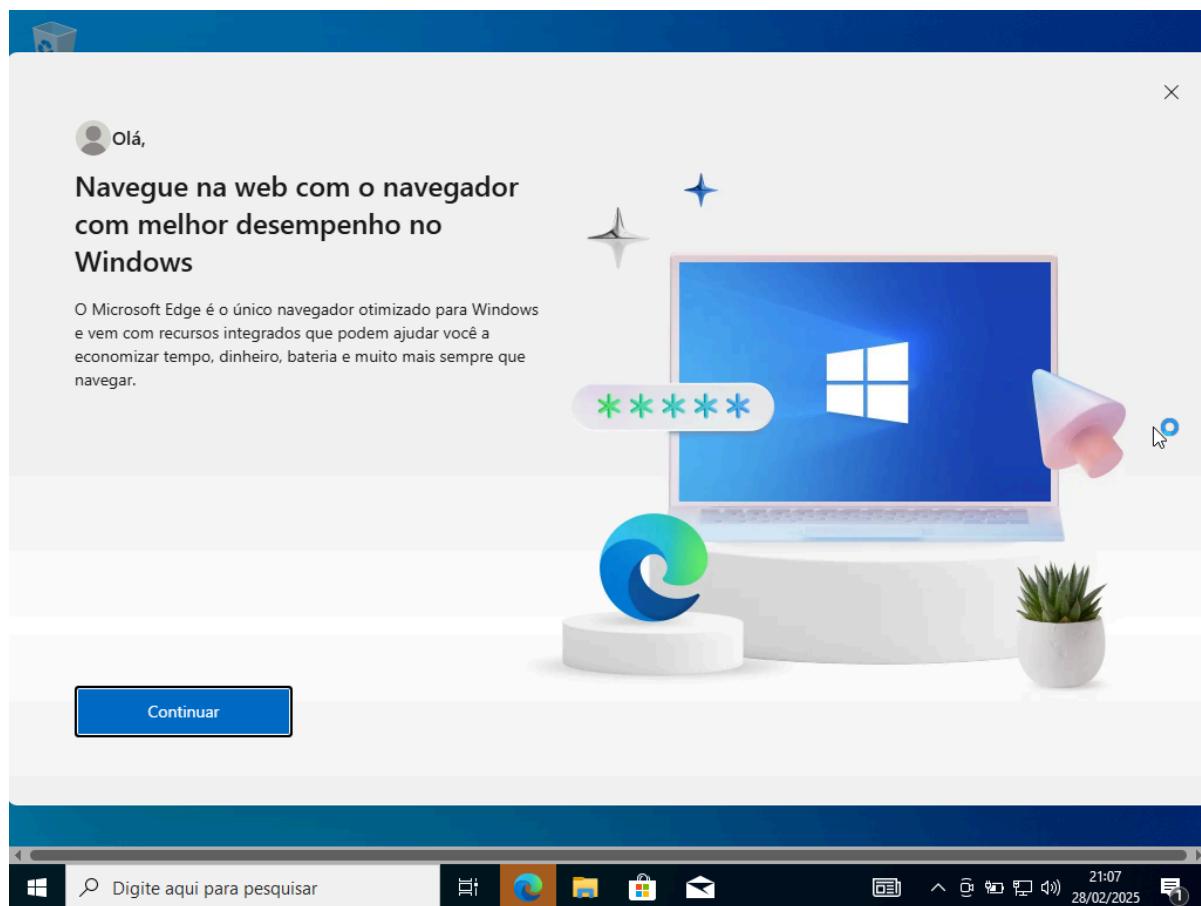
Configuração de experiência personalizada



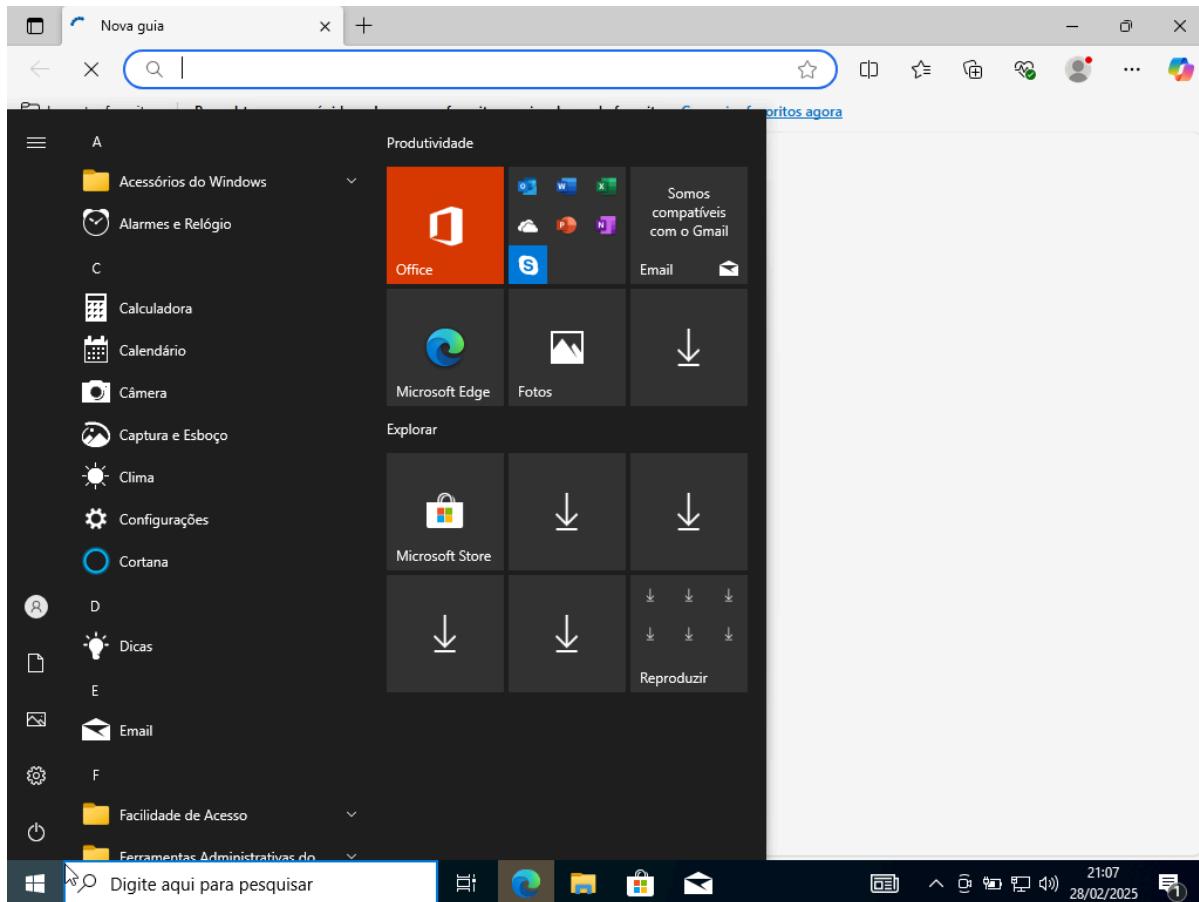
Configuração do assistente digital Cortana



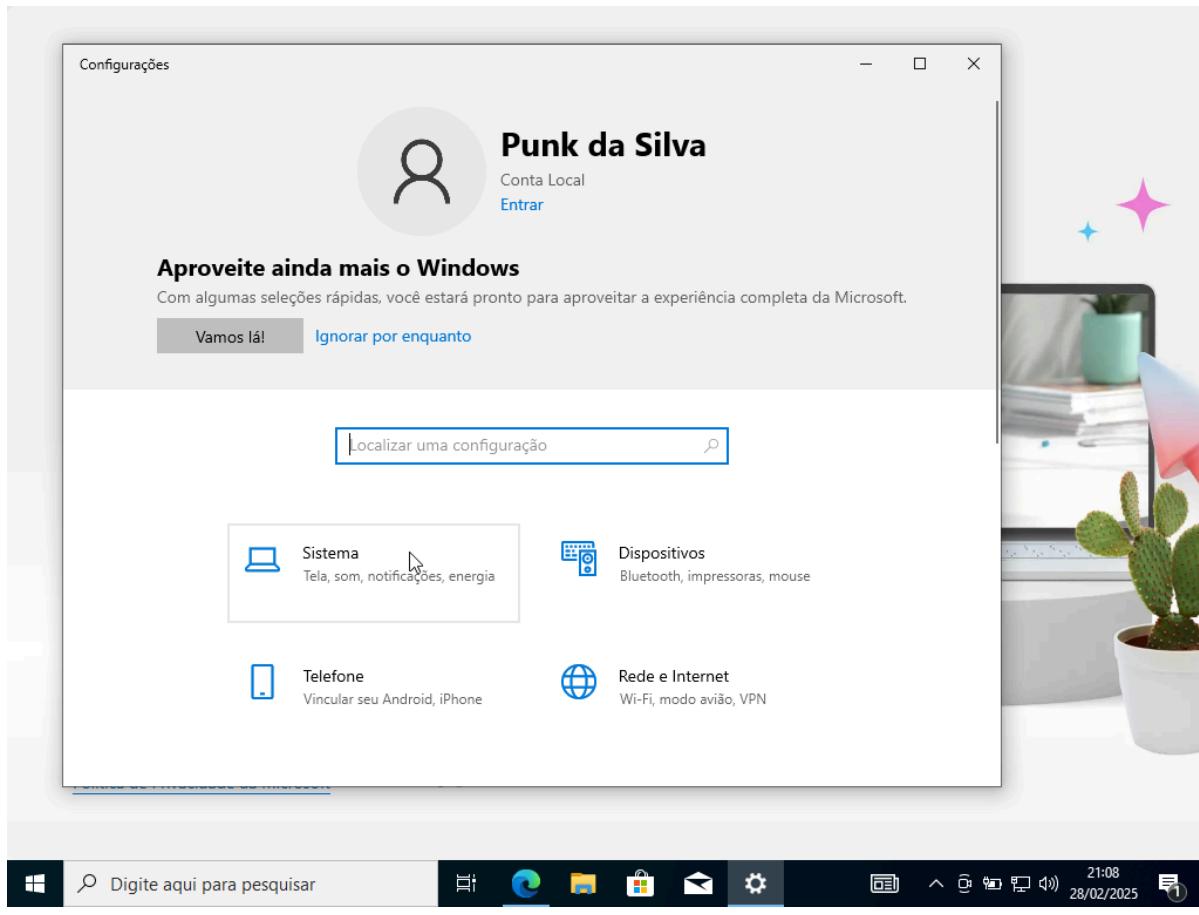
Finalização da configuração do Windows



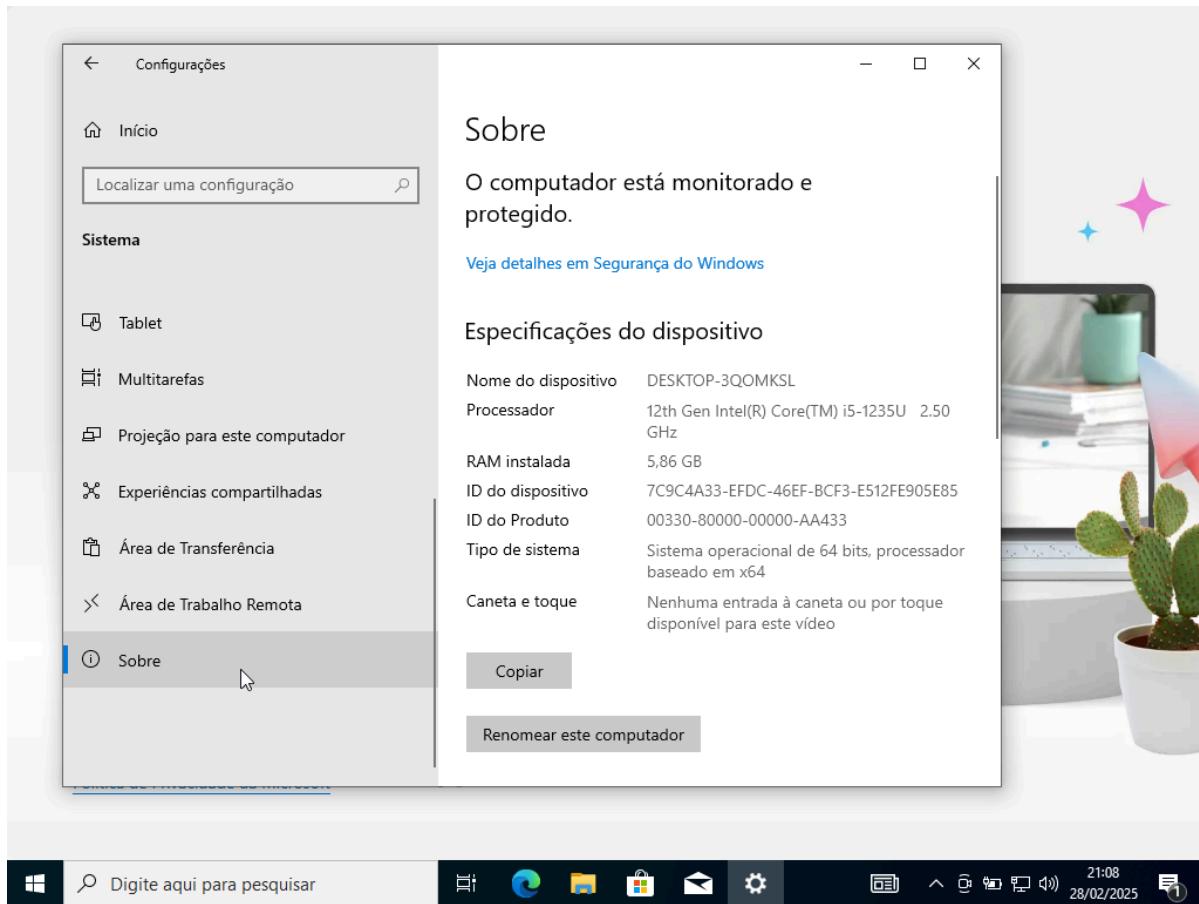
Área de trabalho do Windows após a instalação completa



Menu Iniciar do Windows recém-instalado e digite "Configurações"



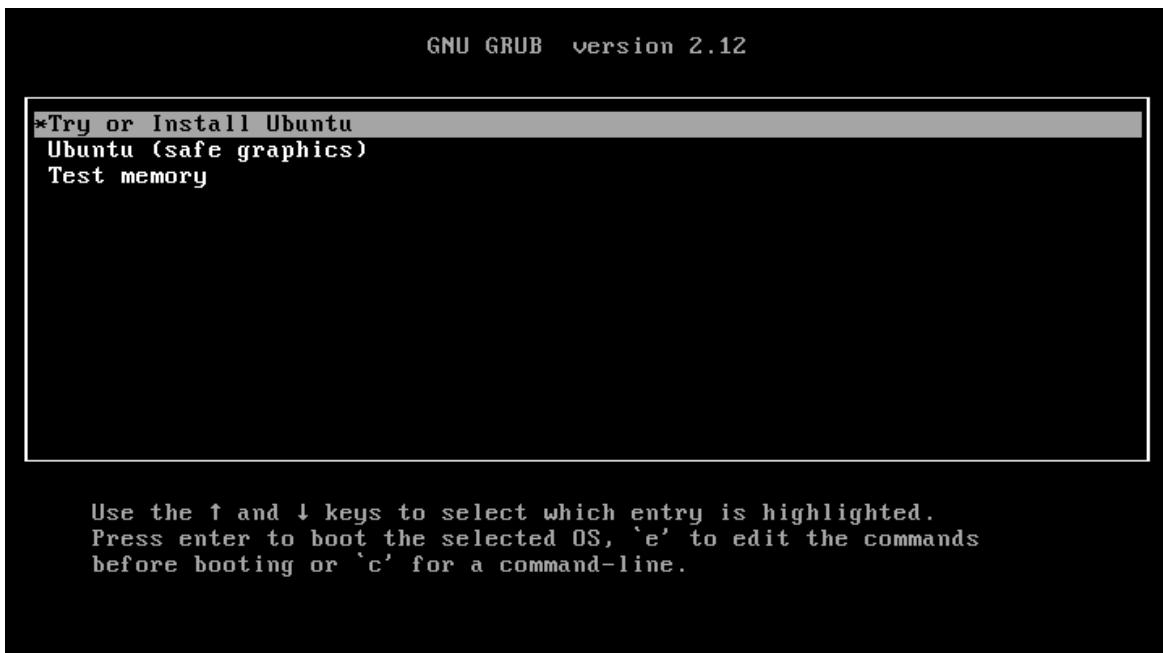
Vá em "Sistema"



Na seção "Sobre" do sistema podemos ver as configurações da máquina que foi instalada

Ubuntu

- Configurações de instalação:

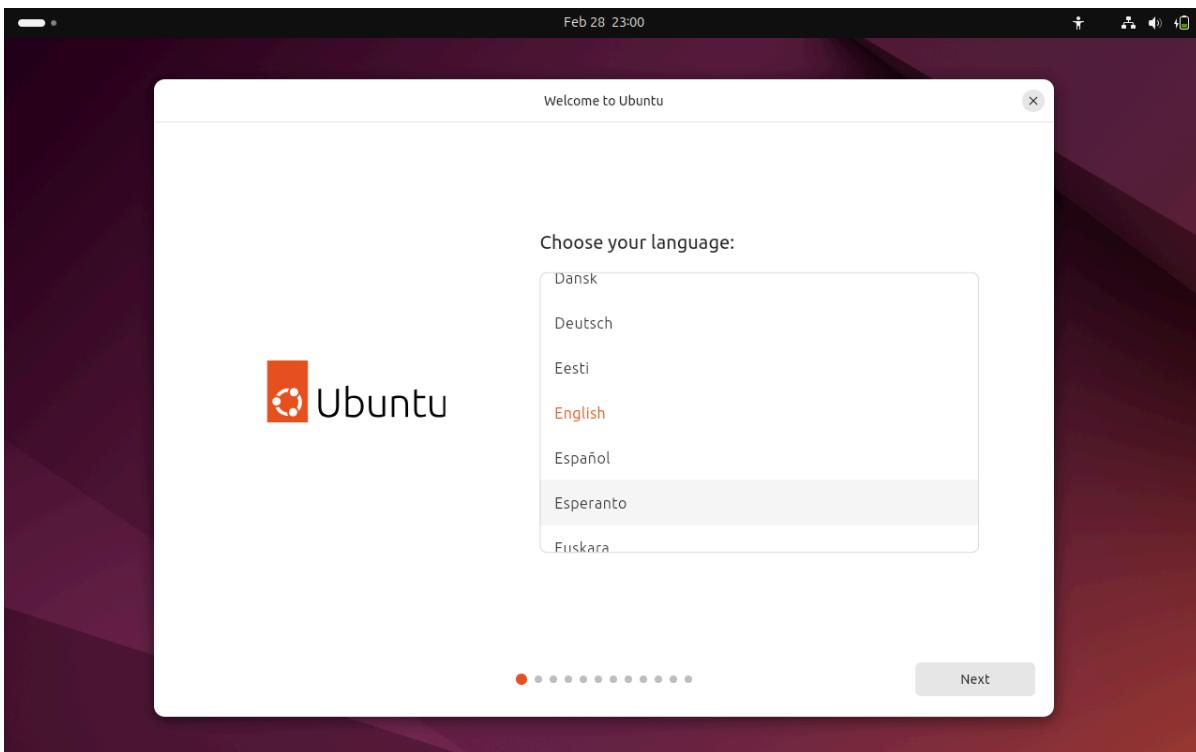


Assim que rodarmos a máquina, vai aparecer a tela do GRUB e então selecionamos a primeira opção

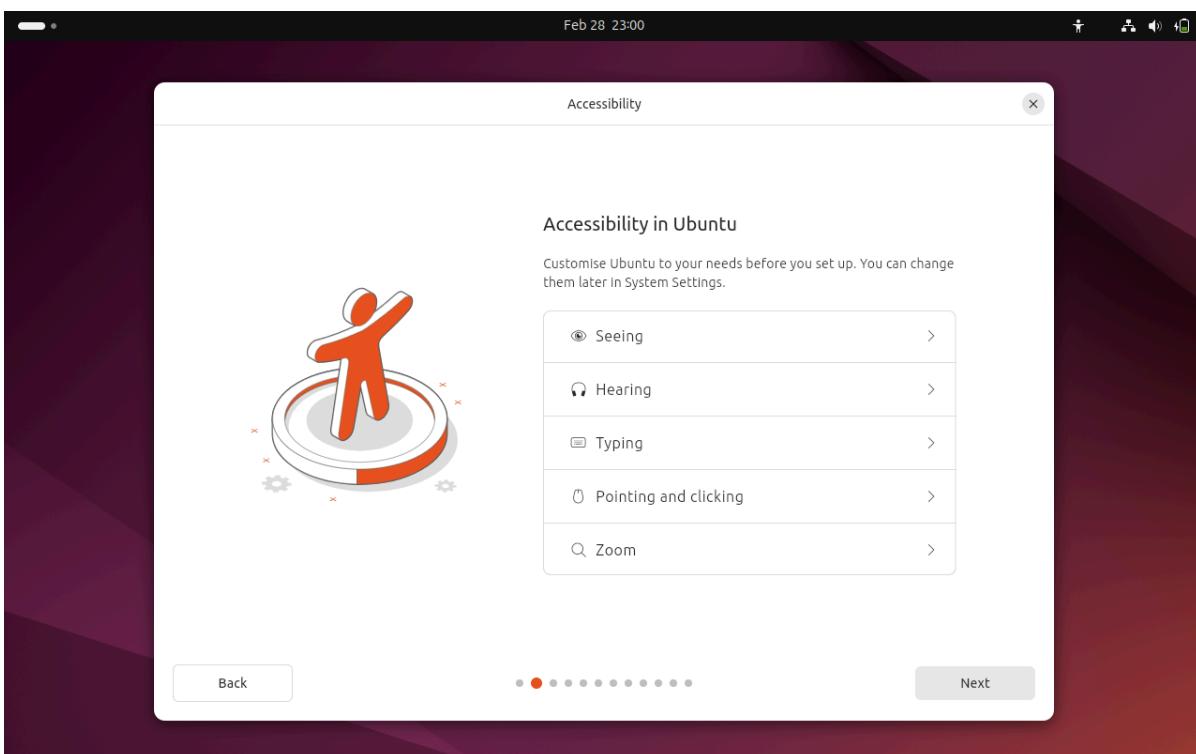
- i** O GRUB (Grand Unified Bootloader) é o menu de inicialização que aparece quando você inicia o sistema Ubuntu. Ele permite que você escolha entre diferentes opções de inicialização.

- A primeira opção geralmente é "Ubuntu", que inicia o sistema normalmente.
- Outras opções podem incluir modos de recuperação ou versões anteriores do kernel.

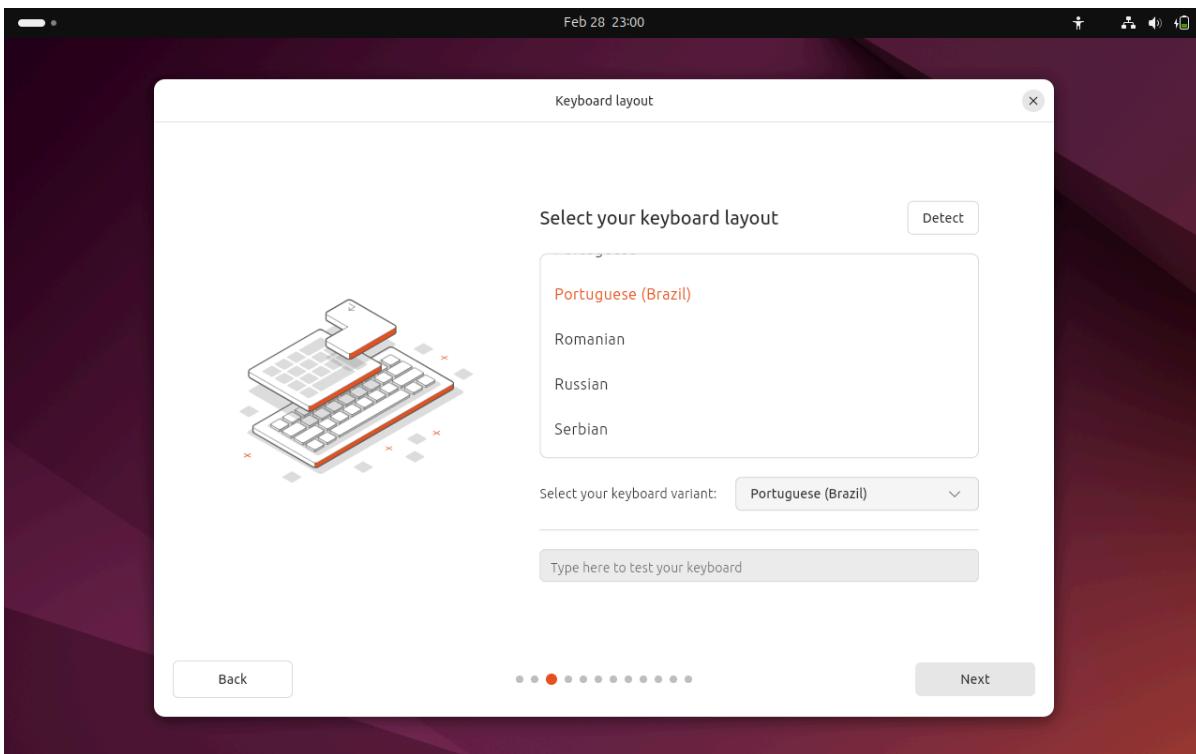
Para a instalação padrão, selecione a primeira opção "Ubuntu" e pressione Enter para continuar.



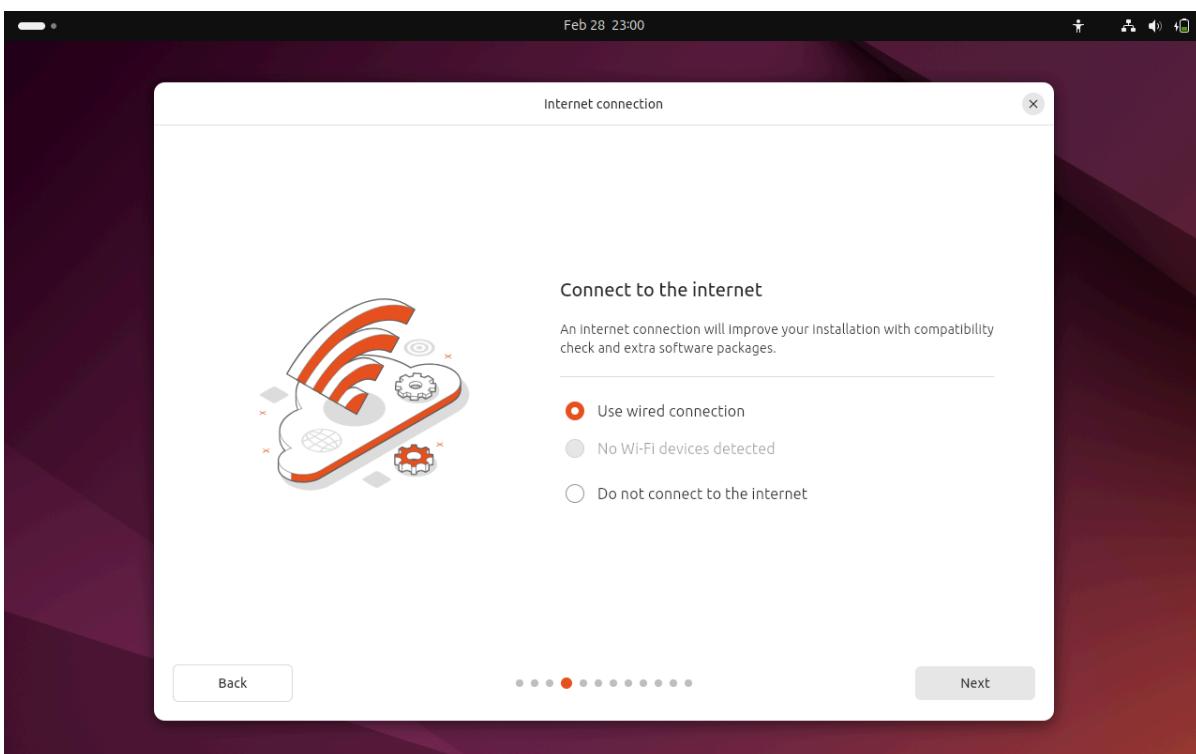
Selecionamos o idioma



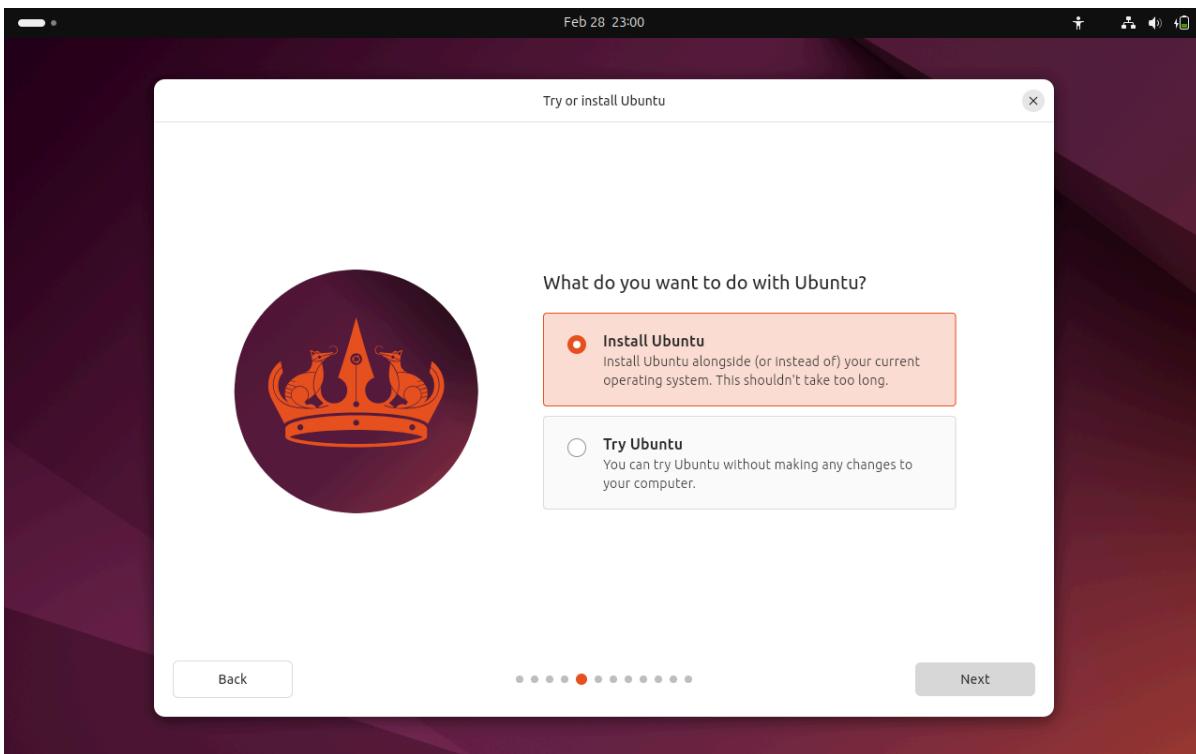
Na parte de Acessibilidade, é só se for necessário



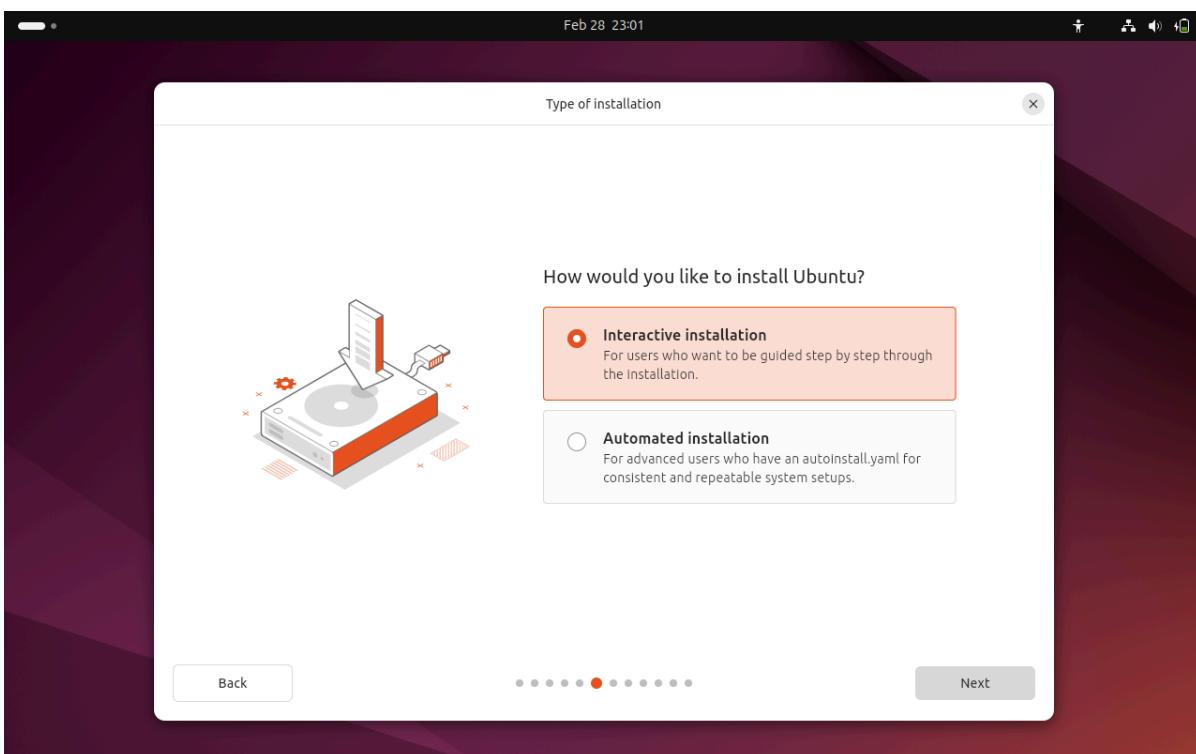
Selecionaremos agora o layout do teclado



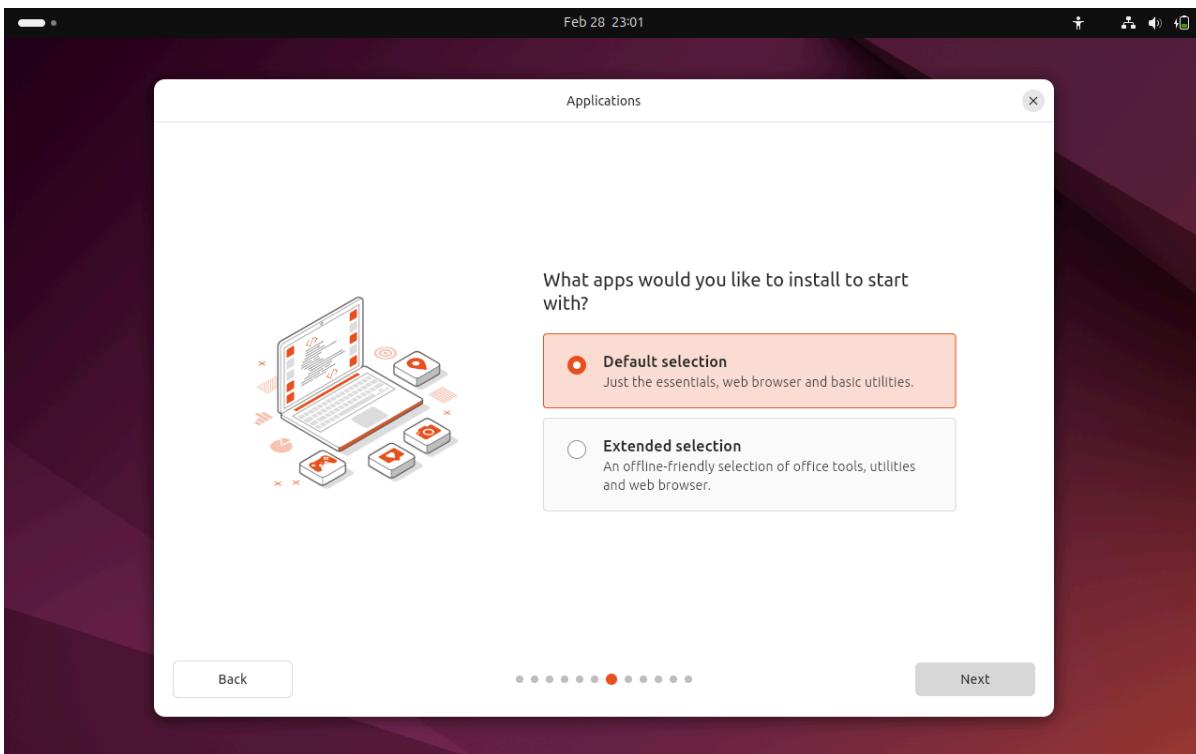
Deixe na forma padrão de conexão



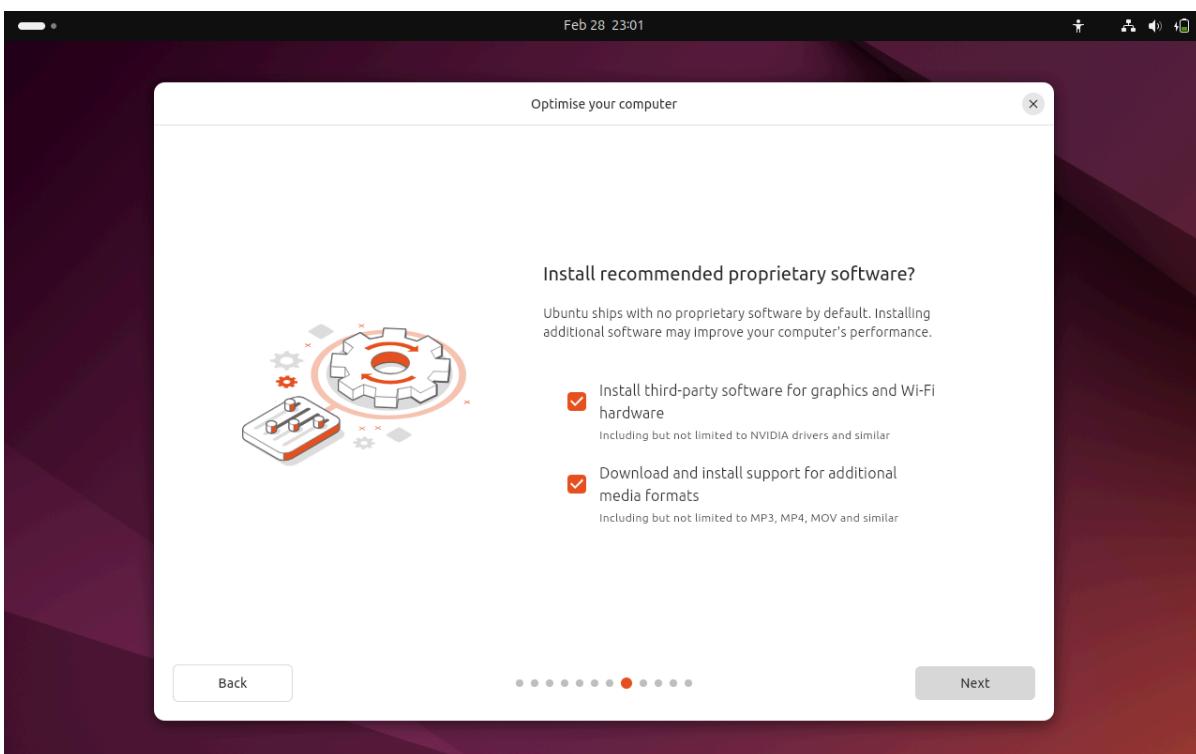
Aperte em "Next" ou "Próximo", deixe selecionada a forma padrão de instalação



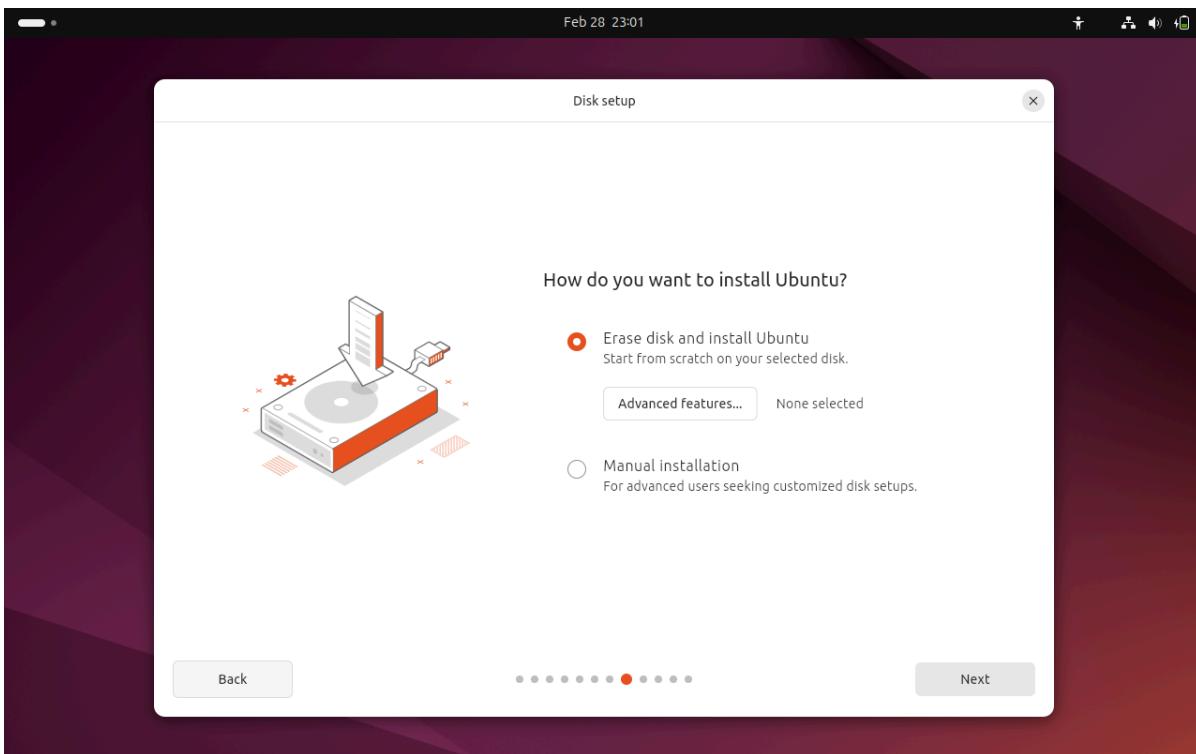
Deixe na forma interativa de instalação, ou seja, primeira opção



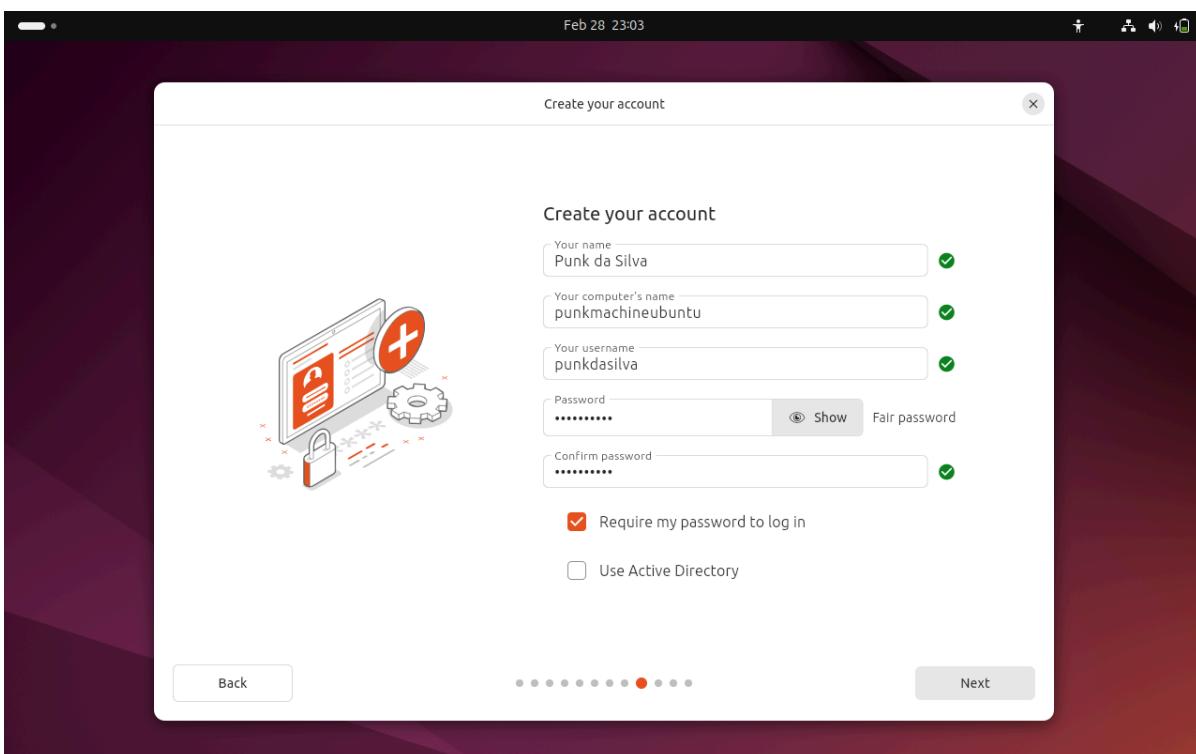
Para a próxima parte, é onde definimos se queremos que ao instalar o Ubuntu sejam instalados aplicativos adicionais, além dos básicos como navegador e outros utilitários. Neste caso, deixe na opção padrão



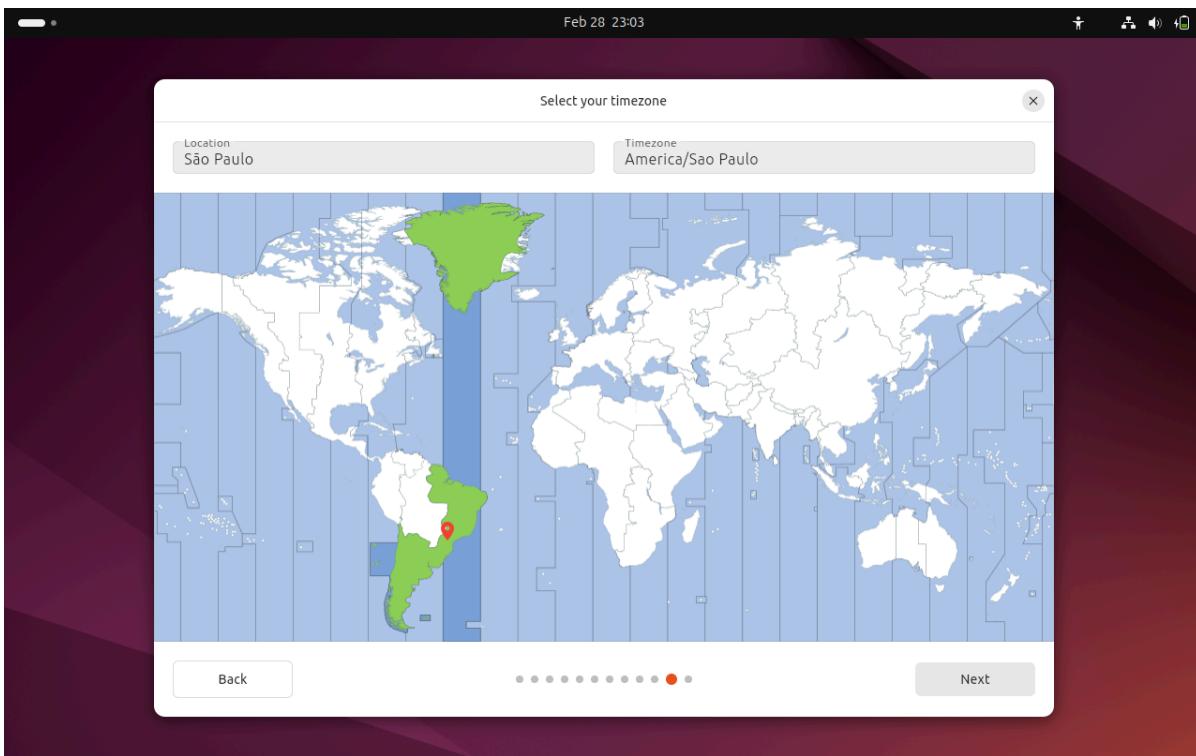
Nesta etapa, selecione todas as opções, que são para instalar softwares de terceiros e download de formatos de mídia adicionais



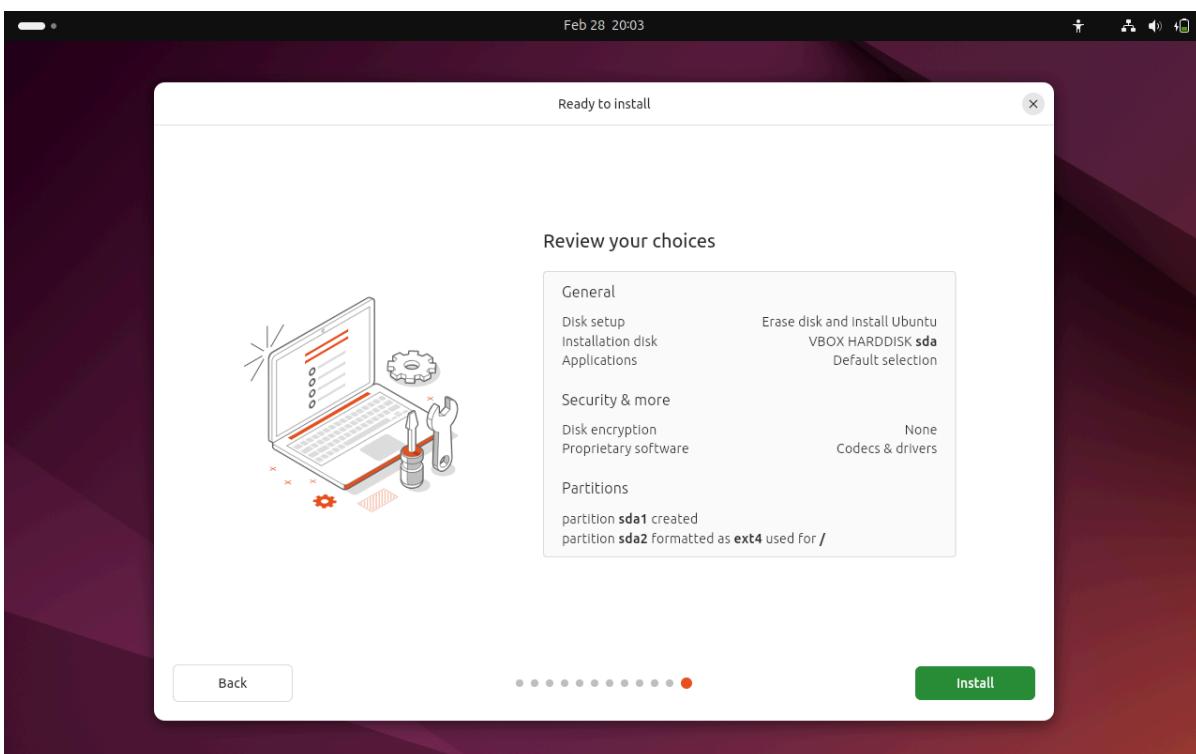
Nesta parte é a definição de se iremos instalar limpando o disco ou se faremos o particionamento do disco. Deixe a opção como padrão e aperte em next



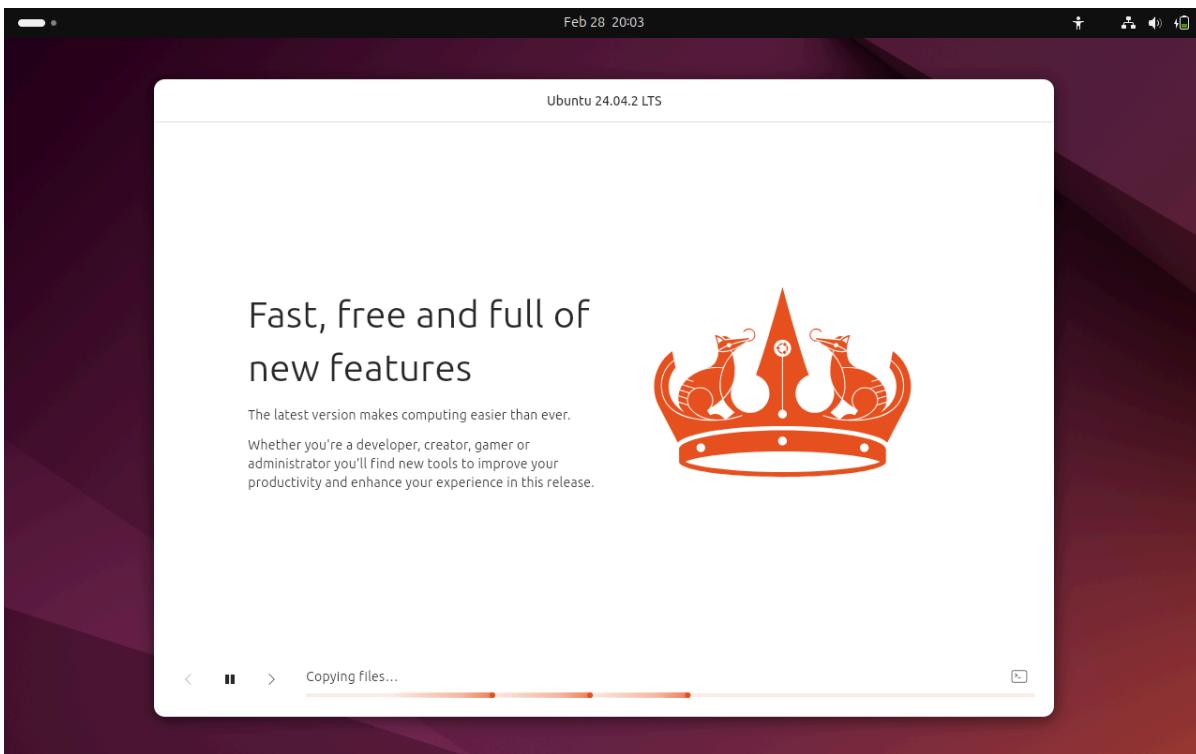
Agora na parte de criação de conta, defina suas credenciais



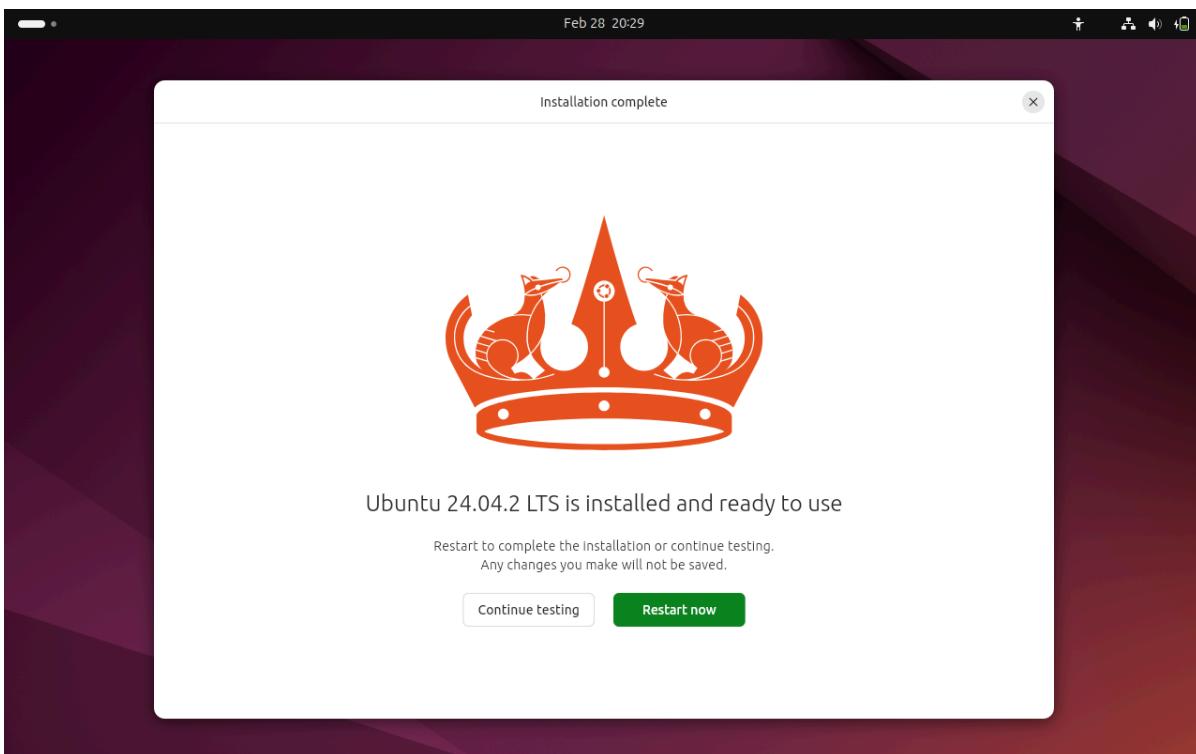
Agora é só fazermos a configuração do fuso horário, ou seja, do tempo que o computador irá usar



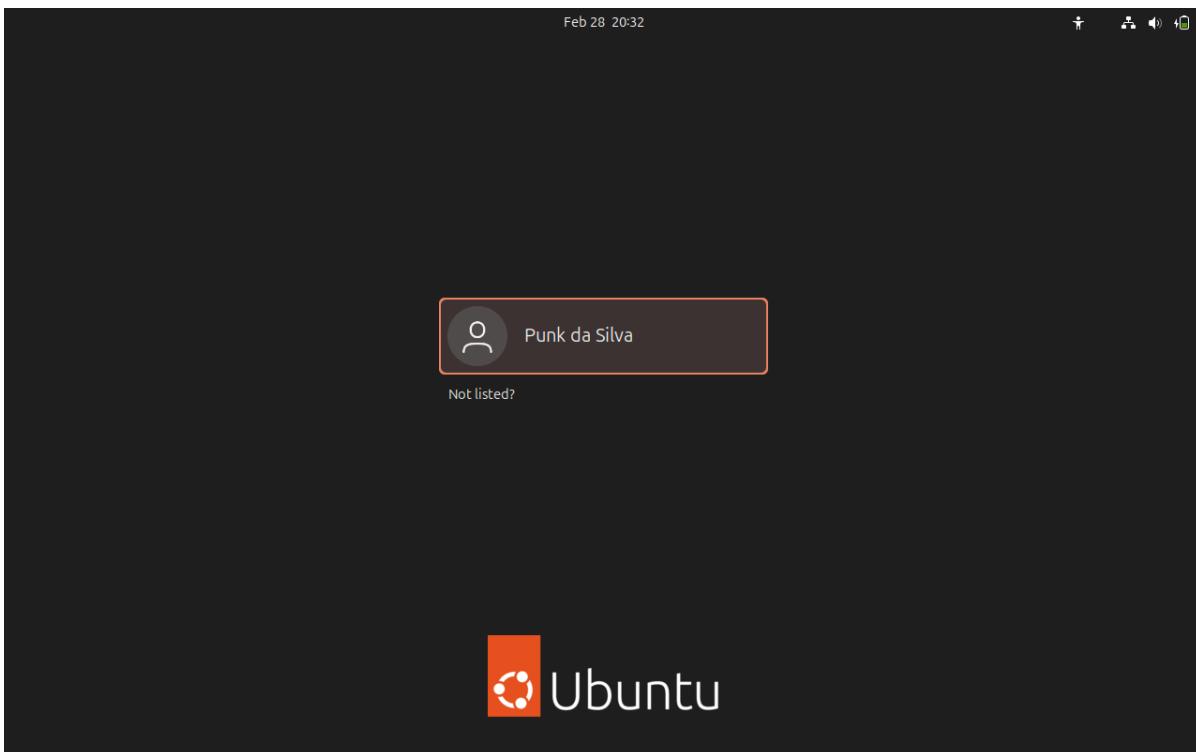
Nesta página será apenas para mostrar o resumo da instalação, uma visão geral das configurações para instalação



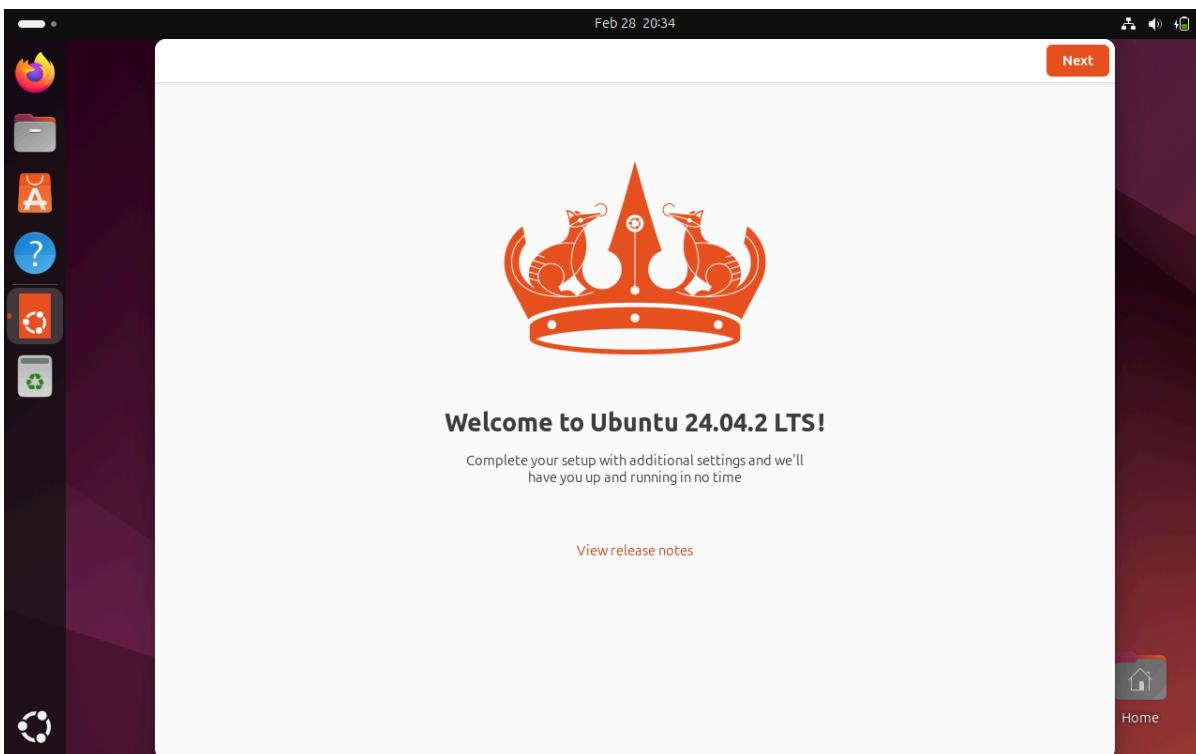
Aqui é a etapa em que o sistema será instalado, pode demorar uns 30 minutos



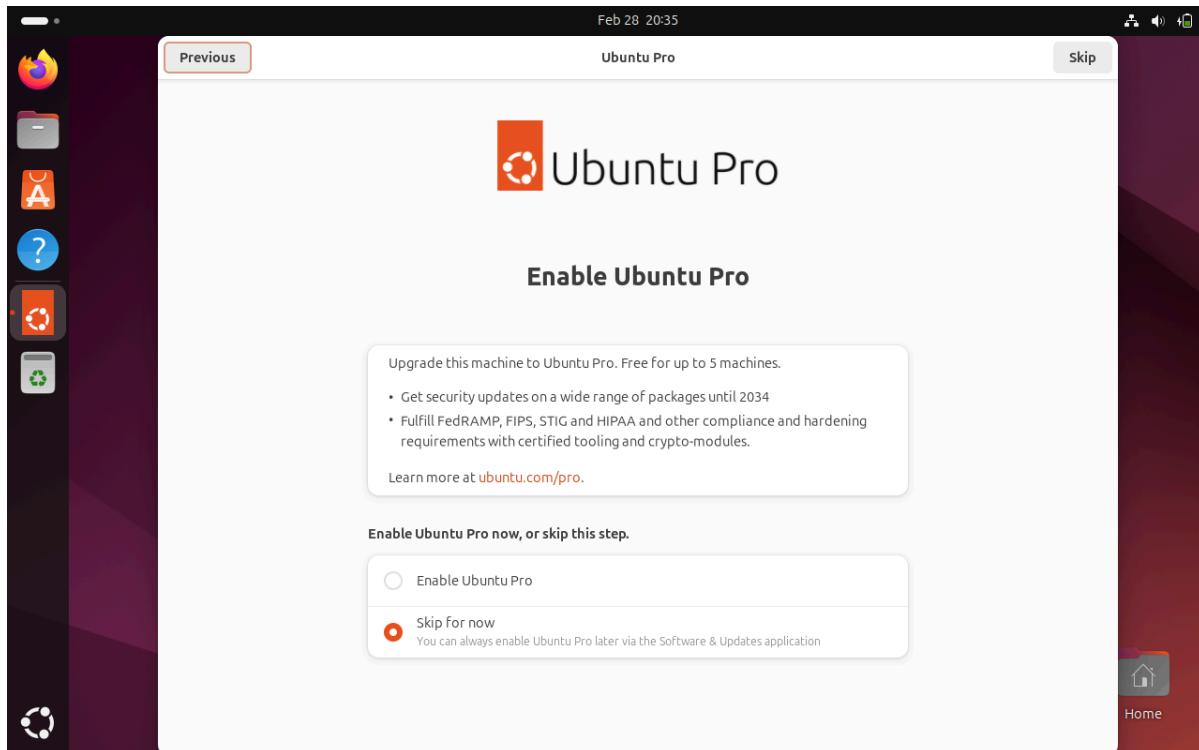
Com a etapa anterior concluída, o sistema já foi instalado e já podemos reiniciar a máquina



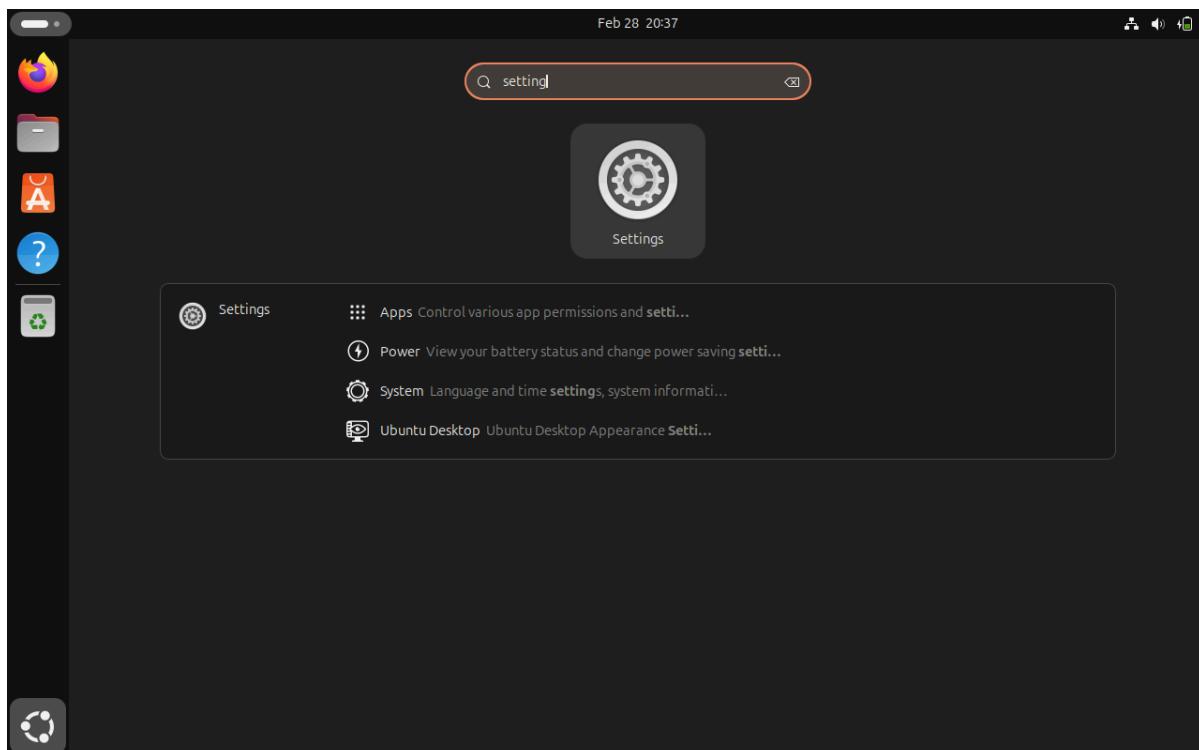
Ao reiniciarmos a máquina, aparece então a tela de login do usuário que foi criado



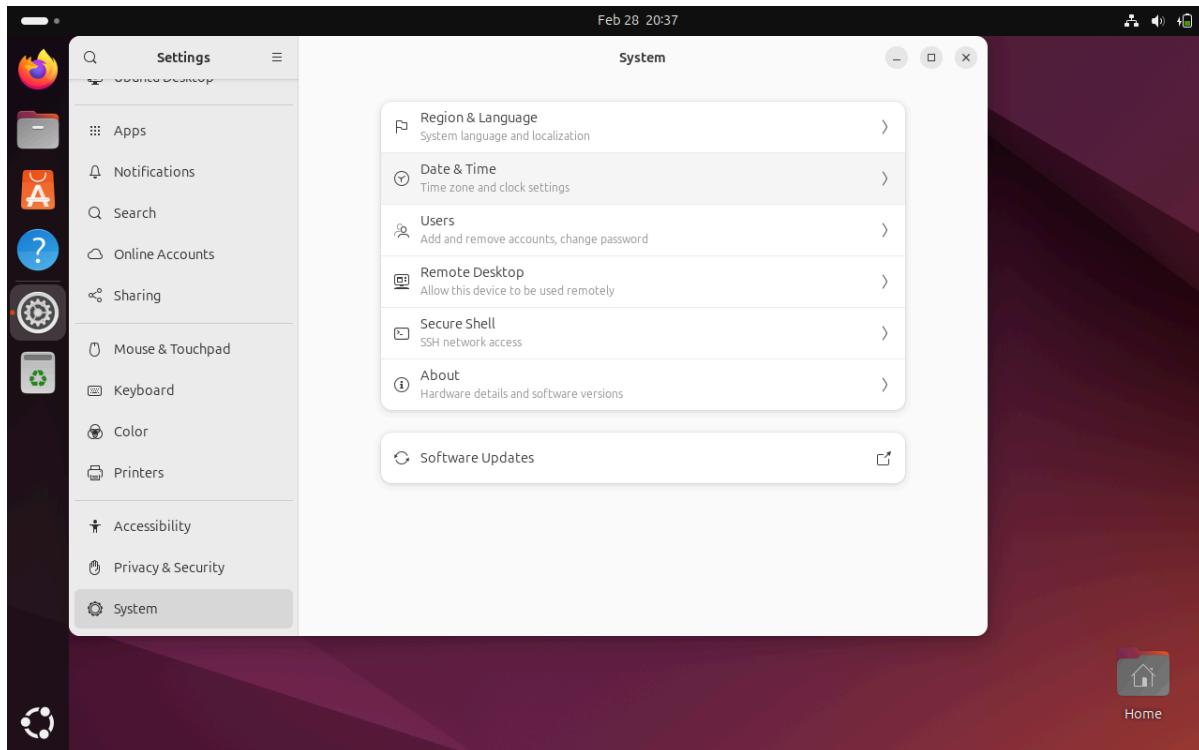
Ao logarmos, temos a visão desta tela



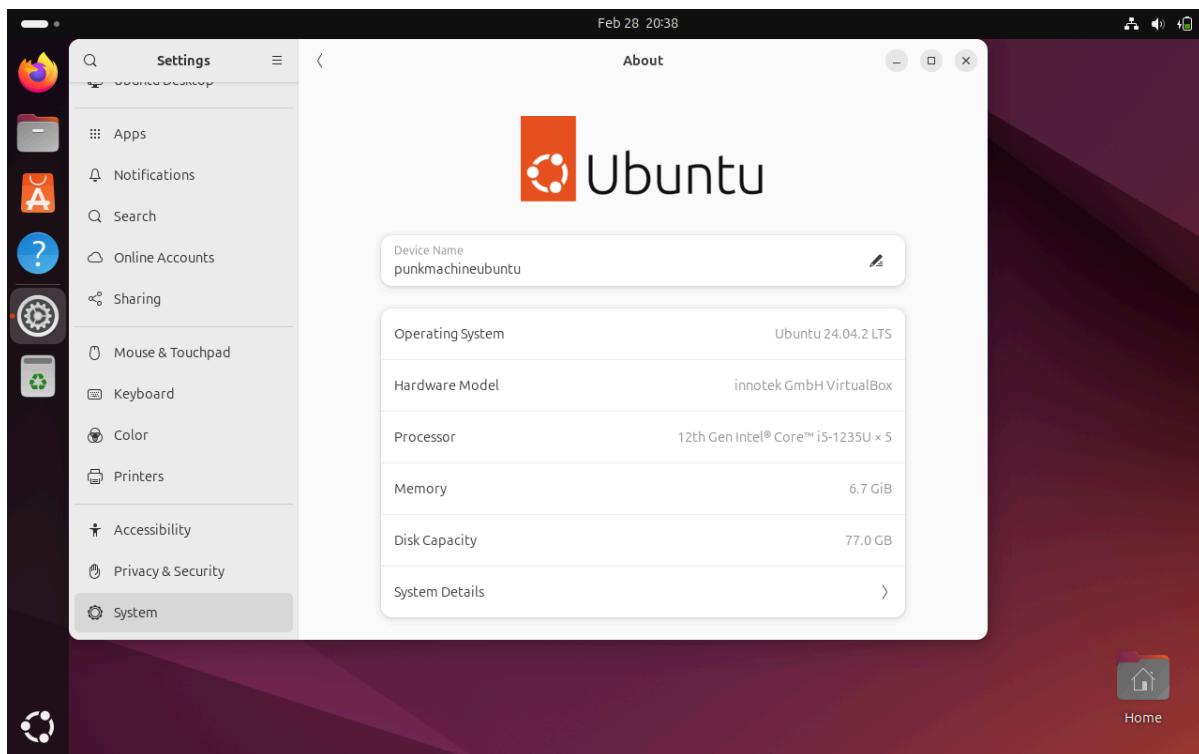
Nessa parte, basta passarmos adiante - clique em Skip ou Prosseguir



Aperte a tecla Super do seu teclado e digite "settings" ou "configurações" e aperte no primeiro item



Vá na parte inferior do menu lateral esquerdo e aperte em "System" ou "Sistema" e em seguida aperte em "About" ou "Sobre"



Assim podemos visualizar as informações do sistema que está instalado

Respostas - Questões 1

#	Resposta Correta
1	b
2	b
3	b
4	b
5	c
6	b
7	d
8	b
9	d
10	d
11	d
12	a
13	d
14	d
15	b
16	a
17	d

18	d
19	a
20	c
21	a
22	b
23	a
24	b
25	b

Bibliografia

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Sistemas Operacionais com Java. 8. ed. Rio de Janeiro: Elsevier, 2010.

TutorialsPoint - Operating System. TUTORIALSPOINT. **Operating System Tutorial. **Disponível em: https://www.tutorialspoint.com/operating_system/index.htm (https://www.tutorialspoint.com/operating_system/index.htm).

TutorialsPoint - OS Overview. TUTORIALSPOINT. Operating System - Overview. Disponível em: https://www.tutorialspoint.com/operating_system/os_overview.htm (https://www.tutorialspoint.com/operating_system/os_overview.htm).

GeeksforGeeks - Operating Systems. GEEKSFORGEEKS. Operating Systems. Disponível em: <https://www.geeksforgeeks.org/operating-systems/> (<https://www.geeksforgeeks.org/operating-systems/>).

GeeksforGeeks - What is an Operating System? GEEKSFORGEEKS. What is an Operating System? Disponível em: <https://www.geeksforgeeks.org/what-is-an-operating-system/> (<https://www.geeksforgeeks.org/what-is-an-operating-system/>).

TechTarget - Operating System (OS) TECHTARGET. What is an Operating System (OS)? Disponível em: [https://www.techtarget.com/whatis/definition/operating-system-OS#:~:text=An%20operating%20system%20\(OS\)%20is,application%20program%20interface%20\(API\)](https://www.techtarget.com/whatis/definition/operating-system-OS#:~:text=An%20operating%20system%20(OS)%20is,application%20program%20interface%20(API)) ([https://www.techtarget.com/whatis/definition/operating-system-OS#:~:text=An%20operating%20system%20\(OS\)%20is,application%20program%20interface%20\(API\)](https://www.techtarget.com/whatis/definition/operating-system-OS#:~:text=An%20operating%20system%20(OS)%20is,application%20program%20interface%20(API))).