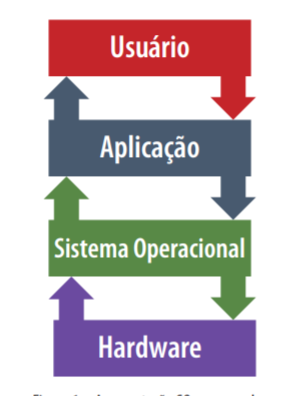
# Sistemas operacionais

Um sistema operacional éum **conjunto de rotinas e serviços** queocomputadorexecutapara **gerenciar recursos** e **facilitar a interação do usuário.**

O SO funciona como uma **interface** **entre** **usuário/aplicação** **e** **hardware**, tornando o seu **uso mais simples**, **eficiente** e **seguro e facilitando o desenvolvimento de aplicações através da abstração de hardware/comandos de baixo nível**.



Enquanto os programas do usuário são executados de forma linear (iniciam, executam e terminam) as **rotinas** **e** **os** **serviços** do SO são **executados de maneira assíncrona**, **de acordo com os eventos que ocorrem.**

## Conceitos sobre recursos do SO

* **Multiprocessamento**: É a capacidade do SO **executar mais de um processamento ao mesmo tempo**, o que **requer mais de um processador.**
* **Multitarefa**: **Simula** o **multiprocessamento** **compartilhando** **tempo** **entre os processos**
* **Multiprogramação**: É a capacidade do SO de **executar vários programas na memória ao mesmo tempo.**
* **Time**-**sharing**: É a capacidade do SO de **compartilhar o tempo de uso do processador entre os processos em execução** q**ue** são **executados sequencialmente,** a fatia de tempo em que eles executam é pequena.
* **Memória virtual: É a capacidade do SO de usar a memória secundária como um cache para armazenamento temporário, permitindo o compartilhamento da memória principal entre os processos de maneira segura e removendo limitações de memória.**

## Funções do SO

As **duas** **principais** funções são:

### Facilitar o acesso aos recursos

Quando o usuário precisa usar um recurso como a memória por exemplo, ele envia uma solicitação para o SO que deve disponibilizar os recursos encapsulando detalhes de implementação ou como os recursos são usados.

### Compartilha**r** recursos de forma organizada e protegida

Em um sistema onde usuários acessam os mesmos recursos é função do SO controlar e gerenciar o acesso concorrente a esses recursos, evitando que um usuário acesse recursos privados de outro e etc.

Mesmo em **sistemas** **monousuários** o SO deve gerenciar o acesso aos recursos por programas.

#### Reentrância

Permite que um programa em execução seja compartilhado por vários usuários simultaneamente, evitando que várias cópias do programa sejam carregadas na memória para evitar desperdício de recursos.

Na prática existe apenas um programa em memória e o código executável é compartilhado por vários usuários de maneira isolada.

## **Histórico do SO**

### **Sistemas em lotes**

Osprimeiros computadores não tinham um SO, os programas eram criados em linguagem de máquina para um hardware específico, a interação com os computadores era feita com cartões perfurados para a entrada de dados e uma impressora para saída.

Os **programas eram divididos tarefas, também chamadas de job ou task que eram agrupadas fisicamente (cartões) e processadas em sequência**, sem interrupção, esse é o chamado sistema em batch ou sistema em lotes.

Um **programa monitor** que fica numa área restrita da memória é responsável por enfileirar as tarefas, e esse programa **executava sempre que um programa do usuário não estava executando.**

Com o tempo foi necessário criar um sistema de interrupção para que um job não monopolizasse o sistema. Esse sistema atribuía uma fatia de tempo para cada job e quando o tempo acabava o monitor chamava interrompia a job atual com uma syscall e chamava a próxima job.

Uma característica desses SO era que **o processador ficava a maior parte do tempo ocioso já que os tempos de espera para I/O eram altos.** Como essas máquinas eram caras havia um grande desperdício em mantê-las nesse estado de espera e então surgiu o spooling para reduzir o tempo de I/O.

#### Spooling (Simultaneous Peripheral Operation On Line)

Consiste em usar uma área do disco como um buffer que guarda os dados que devem ser lidos/escritos ou que já foram lidos/escritos enquanto um dispositivo de E/S/programa está ocupado.

É o que ocorre quando um programa solicita a impressora enquanto ela está ocupada, o SO cria uma fila de solicitações e guarda o arquivo a ser impresso num arquivo de spool até a liberação da impressora.

Com as evoluções citadas **surgiram os primeiros SO na década de 60, eles eram únicos, criados para um hardware específico** já que cada máquina tinha sua arquitetura linguagem proprietária.

### **SO multiprogramados**

Em 1959 foi criada uma técnica chamada **multiprogramação** que permite fazer I/O enquanto um programa é processado, possibilitando a existência de mais de um programa na memória principal e consequentemente aumentando o desempenho do processado.

O segredo da multiprogramação é bloquear o dispositivo de I/O e mudar o processo sendo executado (switch) enquanto a operação (que era demorada) ocorria.

### SO’s Time-sharing

Em 1965 surgiram os sistemas de time-sharing (tempo repartido), esses programas utilizavam multiprogramação, mas dividiam o tempo de processamento da CPU entre os processos/usuários ativos de modo que cada um recebe um time-slice(fatia de tempo) para executar.

O objetivo desse sistema é dar uma resposta em tempos aceitáveis e o atendimento fracionado dá a impressão de que o SO está se dedicando unicamente a tarefa do usuário.

### **SO’s multiusuários**

São sistemas que permitem a mais de um usuário usar um computador por vez, isso se dá através de time-sharing, cada usuário recebe um tempo de processador estipulado pelo fabricante.

O maior desafio desse sistema é isolar os dados dos usuários, evitando a **exclusão mútua falida** (que um programa acesse os dados de outro) e o **deadlock** (que um programa fique esperando a execução do outro).

### **Unix e DOS**

Um dos **primeiros SO’s criados e liberados para uso geral foi o CTSS** em 1961, depois dele surgiu o **Multics, que tinha como propósito suportar vários usuários e que foi usado como base para outros sistemas**.

Um dos desenvolvedores do Multics, Ken Thompson, criou um SO parecido com o Multics, mas menos ambicioso, chamado Unics, que depois de um tempo foi renomeado para Unix, o Unix foi um sucesso e diversos SO’s se baseiam nele, até o Linux e o Mac OS X. Os sistemas baseados em Unix são chamados de sistema Unix ou Unix like.

Em 1970 surgiram os computadores pessoais e **Bill Gates** e seu colega de faculdade **Paul Allen** **compraram o sistema QDOS** e o **renomearam** **para** **DOS** (Disk Operating System), o DOS foi um sucesso e evoluiu, **se tornando o (r) Windows.**

### **Sistemas em tempo real (real-time)**

Durante o período do DOS surgiu a necessidade de sistemas que suportavam aplicações críticas como controle de trafego aéreo, usinas nucleares, centrais de telefone, etc.

Esses sistemas precisavam ter um tempo de resposta dentro de um limite rígido, chamado de **prazo.** A **perda de um prazo** (não cumprir a tarefa no tempo limite) é caracterizada como uma falha do sistema.

Outra **característica** **importante** desses sistemas é sua **interação com o meio ao redor e a previsibilidade** (um sistema é previsível quando se pode antecipar seu comportamento, independente de falhas, variações de hardware e sobrecargas).

### SO’s distribuídos e de rede

Os avanços na web e o aumento do poder de processamento possibilitaram a criação de computadores independeres que se apresentam ao usuário como um único sistema que permite acesso aos recursos, fornecendo o poder vários computadores interligados em rede.

## **Interação com o SO**

A **maioria dos sistemas permite a interação direta** com o usuário, mas **a maneira mais comum é por meio de interfaces**, existem 4 tipos de interfaces:

### Interface terminal

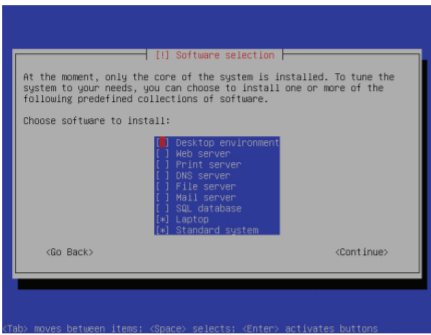
Permite que o usuário envie comandos de uma linguagem de comunicação especial chamada de linguagem de comando ou CLI (Command Line Interface) que funciona com mouse e teclado e partir uma tela simples(chamada de shell ou console de comandos) onde os comandos são digitados.

Normalmente é **usada por usuários avançados** e **permite fazer atividades específicas** como gerenciamento remoto, t**em um baixo consumo de hardware** e **requer conhecimento dos comandos, parâmetros e sintaxe do CLI.**

Os comandos podem executados no console de duas formas, a primeira é a forma interativa, onde o usuário digita um comando por vez no console e a outra é a forma de batch ou em lote onde o usuário cria um arquivo (.bat por exemplo) com comandos que serão executados em sequência toda vez que o arquivo for executado.

### Interface textual

É uma interface baseada em texto, menus, janelas e botões, é uma evolução da interface terminal e foi popularizada pelo MS-DOS. Atualmente o uso de interfaces textuais é raro.



### GUI (Graphic User Interface)

É uma interface com os itens de uma interface textual, mas que também apresenta ícones, figuras e imagens. A mais conhecida é a **WIMP**, que consiste de janelas, ícones, menus e ponteiros.

**O usuário interage com o sistema utilizando o mouse que move o ponteiro**, também existem teclas de atalho, toque em dispositivos ou touchscreen. É basicamente a interface do Windows, Linux, MacOS, etc.

### VUI (Voice User Interface)

É uma interface recente, nela o usuário interage com o SO por meio de **comandos sonoros**, ela é muito comum em SO mobile.

## Classificações de SO

Existem várias formas de se classificar um SO

### Quanto ao modo que ele dedica os recursos (workload)

* **Serial**: Dedica os recursos para um programa até ele terminar de executar
* **Concorrente**: Aloca os recursos de maneira dinâmica entre vários programas ativos em estágios de execução diferentes

### Quanto ao modo que ele executa jobs

* **Batch**: Jobs são submetidos em lote e o usuário só pode interagir com o SO quando ele terminar de processar os jobs
* **Interativos**: Permitem o dialogo com o usuário enquanto o sistema processa os jobs, podem ser sistemas monousuários ou multiusuários e usam multiprogramação e time-sharing.

### Quanto ao número de programas na memória

* **Monoprogramado**: Apenas um programa na memória
* **Multiprogramado**: Mais de um programa na memória

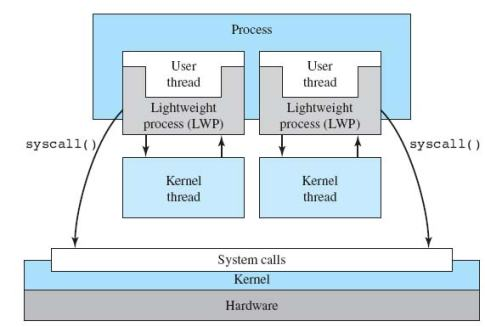
### Quanto a estrutura interna do SO

* **Monolítica ou monobloco**: São os mais primitivos, consistem em um conjunto de rotinas que executam sobre o hardware na memória protegida, os **programas são vistos como sub-rotinas invocadas pelo SO**
* **Núcleo, micro-kernel(kernel) ou cliente-servidor:** Uma parte do sistema chamada de núcleo é composta pelas funções de baixo nível e o resto do sistema é construído em volta dela. O micro kernel fornece serviços de alocação CPU, comunicação de processos, diretórios, gerenciamento de memória, etc agindo como um servidor para programas, usuários e serviços.

## Kernel

Os SO atuais utilizam a estrutura de Kernel. Um Kernel é o núcleo do sistema que possui todas as rotinas do SO executadas de maneira assíncrona para responder a eventos, ele é basicamente todo o SO, excluindo a interface do usuário e aplicações. As principais funções do kernel são:

* Tratar interrupções e exceções
* Criar processos e threads
* Sincronização e comunicação entre processos e threads
* Gerência de recursos e I/O
* Segurança e auditoria
* Garantir que nenhum programa monopolize a CPU



### Modo de acesso

O kernel possui rotinas perigosas e que podem comprometer o sistema, para evitar que qualquer um acesse executa rotinas o SO utiliza um mecanismo de segurança chamado de modo de acesso que não permite o processador executar certas instruções sem permissão.

**O Kernel não é ser acessível diretamente** (compromete segurança e integridade), para resolver esse problema muitos SO usam um mecanismo de segurança chamado modo de acesso.

Existem **dois modos de acesso ao processador,** o **modo usuário** que só permite executar algumas instruções no processador e o **modo kernel** que permite executar todas instruções.

Quando o processador está em modo Kernel uma aplicação tem acesso indireto, pelo SO, as instruções privilegiadas, o modo de acesso é controlado por alguns bits no registrador de status.

#### **System Calls ou System services**

As rotinas do SO que precisam de modo kernel para executar são chamadas (por aplicações) por meio de system calls. System call são interfaces entre aplicações e o kernel que tem como função chamar verificar o privilégio do programa que fez a syscall, entrar em modo kernel (se possui privilégio), chamar a rotina solicitada e voltar para modo usuário.

Um programa não invoca uma rotina do kernel e sim uma system call que verifica se o programa pode invocar rotinas do sistema e que faz o processo necessário para isso

## Sistemas com múltiplos processadores

São sistemas com processadores de mais de uma CPU interligada, possibilitando execução simultânea ou divisão de uma tarefa em mais de um processador. A memória principal e os outros recursos são compartilhados entre os processadores.

Esse tipo de sistema traz as características de sistemas multitarefas e mais outras vantagens:

* **Escalabilidade**: Permite aumentar o poder de processamento com a adição de novos processadores a execução
* **Maior** **disponibilidade**: Se um processador falha outro assume o lugar sem causar perdas
* **Balanceamento** **de** **carga**: Distribuição de tarefas entre os processadores.

Dependendo de como os processadores trocam mensagens e como a memória é compartilhada é possível **classificar esse tipo de sistema em:**

* **Fortemente** **acoplados**/**Multiprocessadores**: Os processadores compartilham a memória física e os recursos que são gerenciados por um único SO.
* **Fracamente** **acoplados**/**Multicomputadores**: Vários SOs são conectados via linha de comunicação e cada SO possui processadores, memória principal e recursos. Cada SO é independente, mas **eles podem se comunicar e se coordenar para executar a mesma tarefa.**