

Sistemas Operacionais



Material Teórico



Introdução a Sistemas Operacionais

Responsável pelo Conteúdo:

Prof. Me. Claudney Sanches Junior

Revisão Textual:

Prof.^a Dr.^a Selma Aparecida Cesarin



- **Introdução;**
- **Definição de SO;**
- **História do SO;**
- **Interação com o SO;**
- **Tipos de SO.**



OBJETIVO DE APRENDIZADO

- Estudar os conceitos básicos dos SO. A unidade apresenta a definição de SO e mostra sua evolução histórica; detalha os tipos de interação e apresenta a classificação dos SOs.
- Entender o relacionamento entre hardware e software permitindo que consiga se adaptar aos principais SOs facilmente, bem como utilizá-los.



Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

Contextualização



Veja a charge a seguir: <https://goo.gl/UEiHmP>

É comum entre Empresas de desenvolvimento haver dois ambientes de desenvolvimento, um de teste e outro de produção.

A maneira mais fácil de você criar um ambiente de teste é instalando um Sistema Operacional em uma máquina virtual.

Você deverá escolher entre a *VirtualBox*, a *VMWare Player* e a *Microsoft Virtual PC*, todas gratuitas e que não vão comprometer o SO residente em seu computador.

Após instalar a máquina virtual, instale uma distribuição do SO que preferir. O *Linux Ubuntu*, o *Debian* ou outra distribuição podem ser instalados facilmente com interface gráfica, ou você poderá escolher uma versão do *Windows* que preferir.

Como aluno(a), você pode pegar um *serial* original pelo convênio *MSDNAA*.

Crie um documento *Word* com as telas dos passos que você deu e as telas dos testes.

Introdução

Nesta unidade, o objetivo será estudar os conceitos básicos, teóricos e práticos dos Sistemas Operacionais (SO) e conhecer suas classificações. Você será apresentado(a) à história do funcionamento, à terminologia utilizada e, sempre que possível, ao motivo da evolução dos SOs.

Ao se deparar com a Disciplina, é comum que muitos imaginem que irão aprender determinada distribuição de um SO, como, por exemplo, aprender *Linux*, mas não é esse o objetivo da Disciplina, e sim, propiciar aos alunos uma visão detalhada sobre os recursos contidos, independente da distribuição ou do fabricante em qualquer SO.

Ao final, espera-se que você seja capaz de entender o relacionamento entre *hardware* e *software*, permitindo que consiga se adaptar e utilizar os principais SOs facilmente, pois estará familiarizado(a) com as principais ideias de funcionamento existentes.

Como os SOs são grandes e complexos, seu estudo o(a) ajudará a se tornar um melhor programador por dar ênfase à sua organização interna e a sua estrutura.

Cada seção aqui apresentada tem um grau de dificuldade a ser superado na sua aprendizagem; é como uma escada: você deve subir um degrau de cada vez.

Se você decidir pular, pode sentir dificuldades e cair, sendo necessário voltar e consultar alguma coisa que não viu. Assim, para que possa entender esses conceitos, esta Unidade está organizada da seguinte forma:

- A Seção 2 apresenta a definição de SO;
- A Seção 3 mostra o surgimento e a história;
- A Seção 4 detalha os tipos de interação;
- A Seção 5 apresenta os tipos de SO.

Ao final do estudo e das atividades desta Unidade, você deve ser capaz de:

- Entender e caracterizar os SOs;
- Conhecer o relacionamento entre *hardware* e *software*.

Não deixe de utilizar os fóruns associados à unidade para apresentar e discutir qualquer dificuldade encontrada.

Definição de SO

O surgimento dos SO permitiu à computação uma mudança drástica no cenário de desenvolvimento de *software*. O SO deixou a tarefa do programador mais rápida. Ele passou a gastar menos tempo de desenvolvimento por permitir que se concentrasse apenas na lógica do problema, sem ter a necessidade de conhecer o *hardware* e os comandos de baixo nível dos diferentes dispositivos ligados a um computador.

O SO é uma camada entre *hardware* e aplicação que fornece à aplicação maior racionalidade, portabilidade e dedicação a problemas de alto nível ou abstratos.

A Figura 1 apresenta uma visão de onde o SO se enquadra entre o *hardware* e as aplicações ou processos.



Figura 1 – Apresentação SO em camadas

Alguns autores definem SO como um programa ou conjunto de programas inter-relacionados, cuja finalidade é agir como intermediário entre usuário e o *hardware* ou como um *software* gerente do *hardware*. Entretanto, essa definição é bastante simples, sendo mais bem definido como um gerenciador de recursos em virtude de outras funções de que se encarrega. Os recursos podem ser memória principal e secundária com seus arquivos, periféricos, tais como, unidade de *CD-ROM* e impressoras, entre outras.

Entre as principais funções do SO, pode-se listar: apresentar ao usuário uma máquina mais flexível; permitir o uso eficiente e controlado dos componentes de *hardware*; permitir o uso compartilhado, eficaz, protegido e seguro dos diversos componentes de *hardware* e de *software* por diversos usuários e prover mecanismos de Gerenciamento de Processos, como criação, escalonamento, controle de concorrência, proteção e destruição.

Além disso, cabe ao SO esconder ou tornar transparente aos aplicativos os detalhes do *hardware*, cabendo apenas ao SO conhecer e negociar com ele.

Os serviços que um SO pode oferecer são muitos, mas os principais são:

- Meios para criação de Programas;
- Meios para execução de Programas como mecanismo para carregar na memória, ler e escrever dados (i/o) e inicialização de arquivos;
- Acesso i/o e arquivos negociando as especificidades e formatos dos arquivos;
- Proteção e acesso a recursos e dados;
- Resolver conflitos e contenção de recursos;
- Detectar erros e responder aos erros;
- Fornecer estatísticas dos recursos;
- Monitorar *performance*.

História do SO

Os primeiros computadores (1945-1955) não tinham SO e sua programação e operação eram feitas diretamente em Linguagem de Máquina. Os programadores controlavam o computador por meio de chaves, fios e luzes.

Nos primeiros SO (1955-1965), a interação entre ser humano e computador era feita por meio de periféricos de baixa velocidade, ou seja, o Sistema lia cartões perfurados para a entrada de dados, como ilustrado no link a seguir, e utilizava impressora para a saída de dados.



Leitor de Cartão Perfurado da IBM em 1923: <https://goo.gl/ZCHLnV>

Os programas com as tarefas ou, simplesmente, *job* ou *task*, em inglês, eram agrupados fisicamente e processados sequencialmente, executando uma tarefa após a outra, sem interrupção. Esse tipo de Sistemas é conhecido como processamento *batch* ou em lote, em português, que tem como base um programa monitor, usado para enfileirar as tarefas.

O Programa Monitor estava sempre na memória principal do computador, disponível para execução, mas, após passar o controle para o Programa do Usuário, só executava novamente quando houvesse necessidade, por erro ou fim do Programa do Usuário.

Uma característica que se buscou nestes SOs foi criar uma área de memória protegida a que os programas dos usuários não tivessem acesso, em que estaria residindo o monitor.

O Sistema *batch* apresentou, em sua evolução, um Sistema de interrupção para que um determinado *job* não monopolizasse o Sistema, assim, para cada *job* é atribuída uma fatia de tempo e se este utilizasse toda a fatia de tempo, o Programa Monitor era chamado.

Como mecanismo de segurança, atribui ao Programa Monitor o privilégio de ter algumas chamadas exclusivas; se algum Programa de Usuário tentasse chamá-las, essas chamadas acarretariam uma interrupção.

Num Sistema do tipo *batch*, procura-se maximizar o número de tarefas processadas por unidade de tempo e minimizar o tempo médio de espera para a execução das tarefas, conforme mostra o link a seguir.



Processamento Batch: <https://goo.gl/RwkR8m>

Devido aos tempos significativos de espera de leitura e impressão, o processador acabava ficando a maior parte do tempo ocioso. Lembre-se de que essas máquinas eram muito caras e existiam poucas; assim, era um desperdício mantê-las em espera pela entrada ou saída de dados e programas.

Uma solução para melhor utilizar o processador foi reduzir o tempo de espera de leitura e escrita utilizando uma técnica de *spooling* (1957) (*Simultaneous Peripheral Operation On Line*).

A técnica consiste em ler os dados previamente e gravá-los agrupados em fitas e discos, que são muito mais rápidos, ficando prontos para serem utilizados quando solicitados pela tarefa.

No início, esse processo era manual, mas se notou que poderia ser automatizado (1960). Os primeiros SOs eram únicos, desenvolvidos para cada computador específico, isso se dava sobre encomenda, pois cada máquina tinha sua arquitetura e linguagem proprietária.

Outra solução consistiu na introdução entrada e saída em paralelo com o processamento (1959). Essa técnica permite a existência de mais de um programa na memória principal, aumentando o desempenho do processador; assim, toda vez que um programa encerra a saída de dados, outro, que foi previamente carregado na memória pode ser alocado, evitando o tempo de espera de carga de novos programas.

O surgimento dessa técnica dá inicio ao SO multiprogramado ou *multiprogramming*, pois, se na memória pode existir mais de um programa, a CPU pode chavar (*switch*) de um programa para o outro, sempre que um programa esteja esperando uma operação de entrada e saída, e isso é conhecido como multitarefa ou *multitasking*.

O problema observado com essa solução foi que a interação com usuários foi prejudicada devido à necessidade de espera pelo processamento completo das demais tarefas do programa que estava rodando, reduzindo a produtividade dos usuários.

Com o objetivo de solucionar o problema de espera do usuário, surgiram os sistemas de *time-sharing* (1965), ou tempo repartido, que utilizam multiprogramação, dividindo o tempo de processamento da *CPU* entre os processos ativos e os múltiplos usuários. Cada um recebe uma fatia de tempo ou *time-slice* para executar.

Num sistema de tempo repartido, procura-se dar um tempo de resposta por comando dentro de limites aceitáveis. O atendimento eficiente com pequenas frações de tempo fornece aos usuários a ilusão de que o Sistema está dedicado unicamente à sua tarefa.

Para entender os Sistemas Multiusuários, imagine que, conhecendo o tempo lento da reação humana, um usuário típico necessitava de 2 segundos de um minuto do processador de um computador típico na época, o que permitia que um computador atendesse até 30 usuários sem atrasos. Esse era o cálculo que os fabricantes faziam para saber quantos usuários poderiam acessar o sistema.

Os SOs multiusuários tinham como desafio proteger os arquivos de vários usuários, evitando a sincronização imprópria, ou seja, que um programa esperando I/O fique em espera enquanto outro recebe sua solicitação de dados; que os programas accessem a região de dados um do outro, o que recebe o nome de exclusão mútua falida; e que os programas fiquem em *deadlock*, ou seja, um fique esperando o outro, como se dois caminhões estivessem querendo atravessar uma ponte de uma via, conforme ilustra a Figura 2.

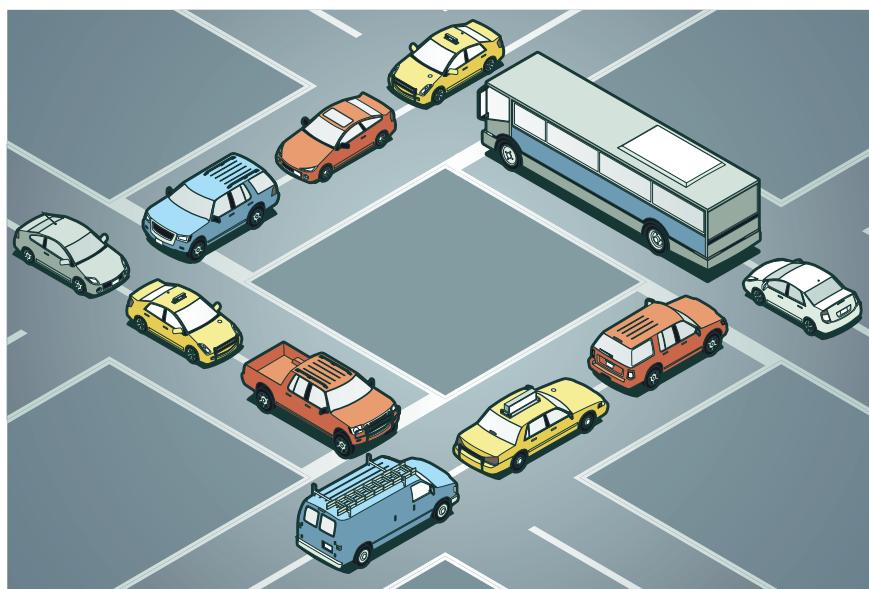


Figura 2 – *Deadlock* – Ninguém pode continuar

Um dos primeiros SOs de propósito geral foi o CTSS (1961), descrito como um Sistema de compartilhamento experimental em Computação interativa.

Após o CTSS, o MIT, os laboratórios Bell da AT&T e a General Electric iniciaram o desenvolvimento do Multics (1964-2000), com o objetivo de suportar centenas de usuários. Multics serviu como base para o estudo e o desenvolvimento de outros SOs.

Um dos desenvolvedores do Multics, Ken Thompson, começou a reescrever um SO num conceito menos ambicioso, criando o Unics (em 1969) que, mais tarde, passou a se chamar Unix (1973). Alguns SOs derivados do *Unix* são: *IRIXG*, *HP-UX*, *AIX*, *Tru64*, *SCO*, família *BSD* (*FreeBSD*, *NetBSD*, *OpenBSD*, *Solaris* etc.), *Linux* e até o *Mac OS X*.

Na década de 1970 começaram a aparecer os computadores pessoais, e em 1980, William (Bill) Gates e seu colega de faculdade, Paul Allen, fundadores da Microsoft, compram o sistema *QDOS* de Tim Paterson, batizando-o de *DOS* (*Disk Operating System*). O *DOS* vendeu muitas cópias, e evoluiu se tornando o SO padrão para os computadores pessoais chamado de Windows.

Durante esse período, surgiu a necessidade de existir outra classe de Sistemas chamados de tempo real ou *real-time*, caracterizados pelo suporte de aplicações críticas, como o controle de tráfego aéreo, usinas nucleares, centrais telefônicas em que o tempo de resposta deve sempre estar entre limites rígidos predefinidos. O tempo de resposta em SO desse tipo é chamado de prazo e a perda de um prazo, ou seja, o não cumprimento de uma tarefa dentro do prazo é caracterizado como falha do Sistema.

Outra característica de SO de tempo real é sua interação com o meio ao redor e a sua previsibilidade. Um Sistema pode ser considerado previsível quando se pode antecipar seu comportamento independente de falhas, sobrecargas e variações de *hardware*.

A
Z

BATCH: processamento em lote, usado para enfileirar tarefas;

SPOOLING: processo de transferência de dados, colocando-os em uma área de trabalho temporária em que outro programa poderá acessá-lo;

TIME-SHARING: tempo repartido que utiliza multiprogramação dividindo o tempo de processamento da CPU entre os processos ativos;

TIME-SLICE: uma fatia de tempo do time-sharing;

REAL-TIME: Sistema com tempo de resposta predefinidos.

Avanços mais recentes foram a estruturação de Tecnologias para redes locais ou *Ethernet* e o uso de Protocolos como *TCP/IP*, o aumento do poder de processamento dos computadores proporcionando, assim, a pesquisa e o desenvolvimento de SO distribuídos e SO de rede.

A ideia é ter um conjunto de computadores independentes que apresente ao usuário como se fosse um único Sistema consistente, permitindo ao usuário o acesso de recursos compartilhados como *hardware*, *software* e dados.

Assim, teríamos o poder de diversos computadores interligados em rede.

Interação com o SO

A maioria dos SOs permite que o usuário interaja de maneira direta, mas é comum encontrar maneiras de interagir por meio de quatro tipos de interface.

A interface de terminal permite que o usuário envie comandos pertencentes à uma Linguagem de Comunicação especial chamada de Linguagem de Comando (*Command Line Interface*) que funciona, exclusivamente, com teclado e mouse. A partir de um *prompt* ou uma tela de acesso simples, os comandos são digitados e interpretados pelo *shell*, um interpretador de comandos.

Costuma ser utilizada por usuários avançados, pois permite a realização de atividades específicas, tais como, o gerenciamento remoto, pois utiliza poucos recursos de *hardware* e requer que o usuário conheça os comandos, os parâmetros e a sintaxe da Linguagem.

A Figura 3 apresenta um console de comandos em *Linux*, que também é chamada simplesmente de *shell* no SO *Unix*. Tem o objetivo de capturar a entrada do usuário e interpretar, enviando os comandos ao núcleo do SO, e imprimir o resultado do processamento na tela.

Quase todos os consoles de comandos dos SOs modernos podem ser usados de forma interativa e no modo *batch* ou em lote. Na forma interativa, o usuário digita um comando após outro, enquanto no modo *batch*, cria-se um arquivo com uma sequência de comandos que podem ser reutilizados quantas vezes for necessário.

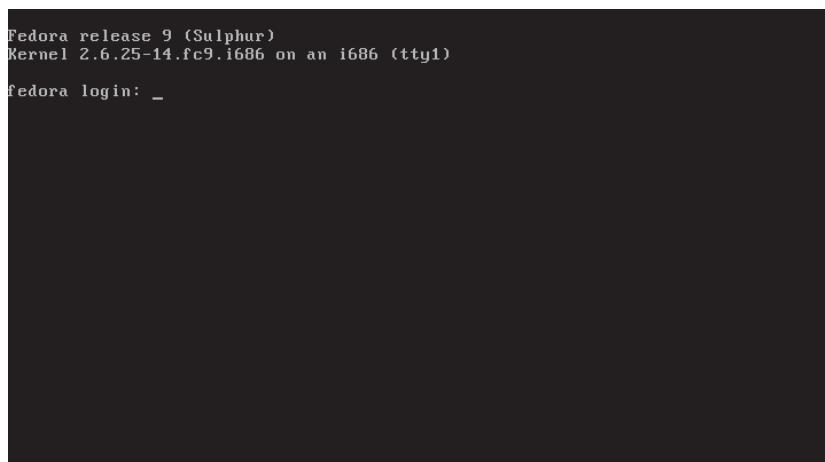


Figura 3 – Console de comandos ou Shell

A interface textual é baseada em texto, menus, janelas e botões. É uma evolução da interface de terminal, que foi difundida em aplicações baseadas no SO *MS-DOS*; porém, seu uso, atualmente, tornou-se raro, sendo comum encontrá-la no processo de instalação do SO, como apresentado na Figura 4.

Também se pode encontrar a interface textual em Sistemas desenvolvidos entre a década de 1980 e o início da década de 1990.

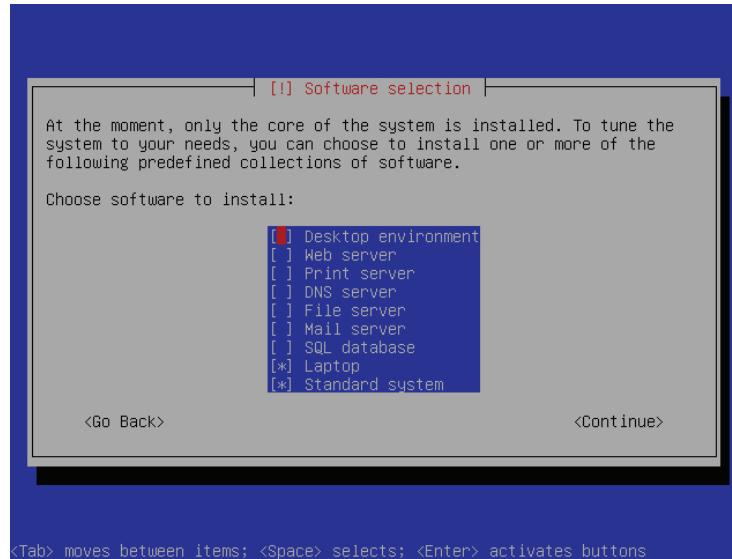


Figura 4 – Console textual apresentado durante a instalação

A interface gráfica, chamada de *GUI* (*Graphic User Interface*), além de apresentar *menus*, janelas e botões, apresenta elementos gráficos como ícones, Figuras e imagens. Em SOs que rodam em computadores pessoais, a mais conhecida é o *WIMP*, que consiste de janelas, ícones, menus e ponteiros.

O usuário interage usando um ponteiro. Movendo o *mouse*, é possível controlar a posição do cursor e apresentar as informações em janelas.

Também é possível interagir com o teclado, teclas de atalhos, toque em dispositivos sensíveis ao toque ou *touchscreen* e, recentemente, com dispositivos de reconhecimento de gestos e expressões faciais. O usuário é capaz de selecionar símbolos e manipulá-los de forma a obter resultados práticos.

A Figura 5 apresenta as *GUI*: *SO Windows Blue*, *OS X Mavericks*, *SO Linux Debian Wheezy* e *SO Linux Ubuntu 13.04*.

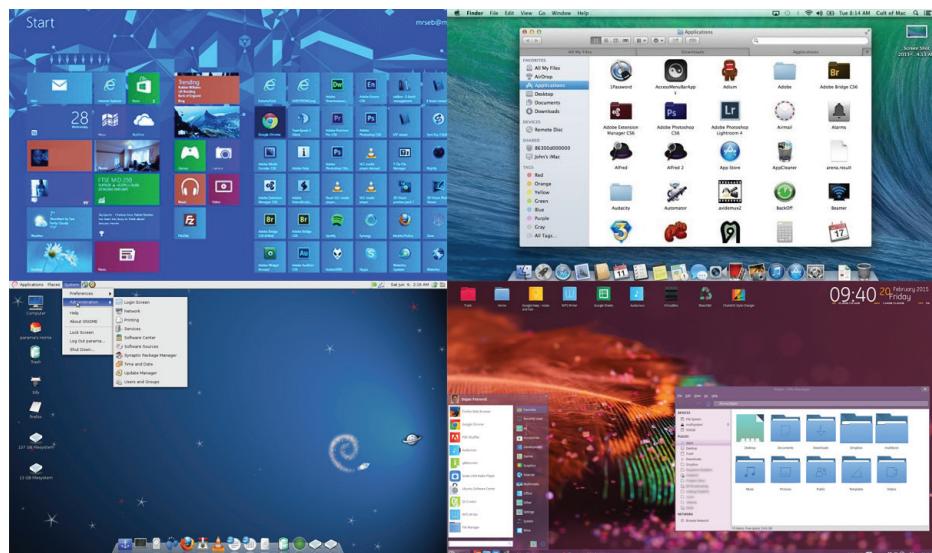


Figura 5 – SO Windows Blue, OS X Mavericks, SO Linux Debian Wheezy e SO Linux Ubuntu 13.04

A mais nova interface é a voz, ou *VUI* (*Voice User Interface*), na qual o usuário interage com o SO por meio de comandos sonoros; tem recentemente encontrado aceitação em SO Mobile como *smartphones* e *tablets*.

Tipos de SO

Existem diversas maneiras de classificar os SOs, você deve aprender a diferenciar a terminologia para poder classificar corretamente um SO.

Quanto à carga de trabalho, ou *workload*, o SO pode ser serial, ou seja, ter todos os recursos dedicados a um único programa até o seu término ou ser concorrente, no qual os recursos são dinamicamente reassociados entre uma coleção de programas ativos e em diferentes estágios de execução.

Outra classificação é referente ao compartilhamento do *hardware*, que podem ser monoprogramados ou multiprogramados. O primeiro permite apenas um programa ativo em um dado período de tempo, o qual permanece na memória até o seu término. O segundo mantém mais de um programa simultaneamente na memória principal para permitir o compartilhamento efetivo do tempo da CPU e dos demais recursos.

SOs podem ser classificados quanto ao tempo de resposta, se será em ***batch*** ou ***interativo***.

Nos **SOs** em ***batch***, os *jobs* são submetidos em ordem sequencial de execução e, enquanto ocorre o processamento, o usuário fica sem interação, até o término da execução dos *jobs*.

Os **SO interativos** permitem o diálogo com o usuário, podendo ser projetado como sistemas monousuários ou multiusuários, usando conceitos de multiprogramação e *time-sharing*.

Os sistemas monousuários só permitem que um usuário possa interagir com o Sistema, como foi o caso do SO *MS-DOS*. O SO é dito multiusuário quando provê atendimento a mais de um usuário, como ocorre com o *Unix* e o *Linux*.

Multiprogramação é a capacidade de o Sistema permitir que mais de um programa esteja presente na memória principal em contraste à monoprogramação em que apenas um programa reside na memória principal.

Quanto à estrutura interna do SO, pode ser monolítica ou de núcleo (*kernel*). Os SOs monolíticos também conhecidos como monobloco são os mais primitivos, consiste de um conjunto de rotinas que executam sobre o *hardware* como se fosse um único programa residindo na memória principal protegida e os Programas dos Usuários são vistos pelo Sistema como sub-rotinas invocadas pelo SO quando ele não está executando nenhuma das funções do Sistema.

Os SOs com núcleo ou *micro-kernel* ou, ainda, cliente-servidor, incorporam somente as funções de baixo nível, ou seja, as mais vitais, dando assim, a base para ser construído o resto do SO. A maioria desses Sistemas é construída como coleções de processos concorrentes.

O *micro-kernel* deve fornecer os serviços de alocação de *CPU* e de comunicação dos processos sendo que outras funções, como sistemas de servidor de diretórios e arquivos e gerenciamento de memória, são executadas no espaço do usuário. Assim como os serviços e as aplicações, os programas dos usuários são os clientes.

Ainda, o SO pode ser em camadas, nas quais as funções do núcleo irão executar em camadas distintas, de acordo com seu nível de privilégio, como ocorreu no SO *Multics*.

Novos elementos estão em pauta como máquinas multiprocessadas, com redes de alta velocidade, processadores rápidos e grandes memórias.

Pode também existir um *software* que pode fornecer uma abstração do *hardware* para vários SOs recebendo o nome de monitor de Máquinas Virtuais (VM), como ocorre no *VMware*.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Leitura

O que são máquinas virtuais?

<https://goo.gl/nMoLUu>

VM Ware, Virtual Box ou Virtual PC: qual é o melhor programa para virtualização?

<https://bit.ly/3oxzhvN>

Referências

DEITEL, H.M.; **Sistemas Operacionais**; 3º Edição; São Paulo; Pearson Prentice Hall, 2005.

TANENBAUM, A.S.; **Sistemas Operacionais Modernos**; 3º Edição; São Paulo; Pearson Prentice Hall, 2009.



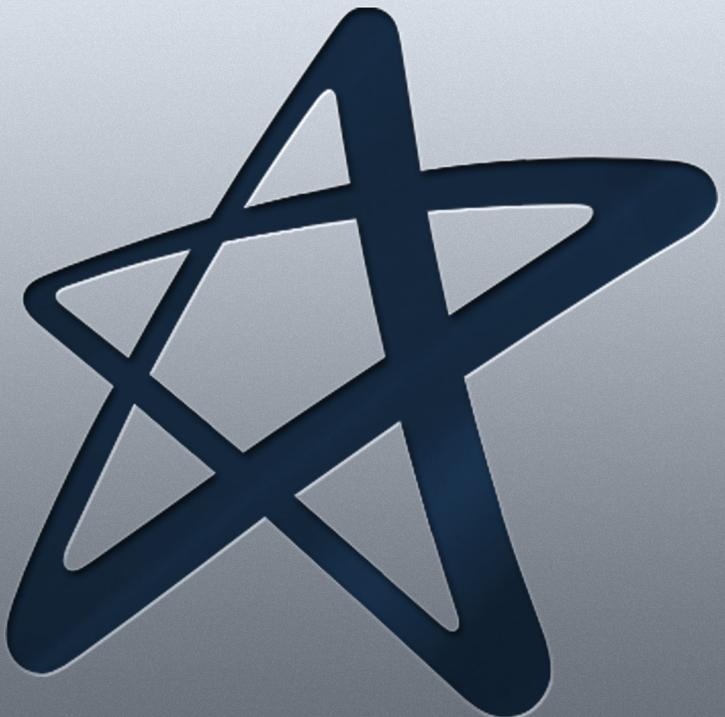
Cruzeiro do Sul Virtual
Educação a Distância

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo - SP - Brasil
Tel: (55 11) 3385-3000



Cruzeiro do Sul
Educacional

Sistemas Operacionais



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Gerenciamento de Processos

Responsável pelo Conteúdo:
Prof. Ms. Claudney Sanches Junior

Revisão Textual:
Profa. Dra. Patrícia Silvestre Leite Di Iório



- Gerenciador de Processos
- Parâmetros de Escalonamento
- Interrupção
- Threads



O objetivo desta unidade será estudar os conceitos de processos dos SO. A unidade apresenta a definição de processos e mostra o gerenciamento dos mesmo.

Detalha os estados do processos e apresenta o conceito de threads. Neste contexto, espera-se que ao final da unidade você seja capaz de entender o gerenciamento de processos e interrupção.

Para que possa entender os conceitos de processos e escalonamento do SO, esta unidade está organizada da seguinte forma:

- a seção 2, apresenta o gerenciador de processos;
- a seção 3, mostra os parâmetros de escalonamento;
- a seção 4, detalha a interrupção;
- a seção 5, apresenta as threads.

Ao final do estudo e das atividades desta unidade, você deve ser capaz de:

- entender e caracterizar processos em SOs;
- conhecer o escalonador e seus principais algoritmos.

Contextualização



Gerenciador de Processos



Para resolver as dificuldades de multiprogramação e tempo compartilhado ou *multiprogramming* e *time-sharing*, há cinco grande assuntos que todo SO moderno deve considerar. Vamos abordar o primeiro que é o gerenciamento de processos. Os SO mutiprogramados devem proporcionar o melhor aproveitamento da CPU quer ela tenha um processador ou vários processadores dividindo o tempo da mesma, para que vários programas rodem em fila, com intervalos de tempo tão pequenos e com trocas tão rápidas que acabam dando a ilusão ao usuário que todos estão sendo executados simultaneamente, independentemente se estão trabalhando fisicamente em paralelo ou lógicamente em paralelo através do SO.

Um programa de computador é o algoritmo escrito em uma linguagem de programação, como a linguagem C ou linguagem Java, com o objetivo de resolver um determinado problema. Quando um programa está em execução em uma CPU receberá o nome de processo. Processo é um termo mais genérico do que *job* ou *task*, introduzido para obter uma maneira sistemática de monitorar e controlar a execução de um programa. O conceito de processo é dinâmico em contraposição ao conceito de programa que é estático. O programa reside em no disco, não faz nenhuma ação sem entrar em execução enquanto que o processo reside na memória principal da máquina e está no processo de execução. O processo consiste em um programa executável associado as seus dados e ao seu contexto de execução. Nem sempre um programa irá equivaler a apenas um processo, pois existe SO que permitem a reentrância, permitindo a um programa gerar diversos processos. Uma característica marcante do processo é que o programa ou código que o gera não pode apresentar nenhuma característica de suposição de temporização. Em geral, o SO determina a fatia de tempo entre os processos e esta fatia de tempo é imprevisível. Um algoritmo de escalonamento determina quando um processo irá parar e outro irá entrar em execução.

O contexto de execução de um processo é o conjunto de dados necessários a sua execução, como por exemplo, a identificação do processo chamado de *pid*, todos os conteúdos dos registradores dos processadores tais como o contador de programa ou simplesmente *PC - Program Counter*, os ponteiros *SP - Stack Pointer*, as variáveis e dados armazenados na memória, a lista de arquivos que estão sendo utilizados, tempo de CPU disponível, prioridade de execução, eventos que o processo pode estar esperando entre outros. Essas informações são fundamentais para que um processo interrompido pelo escalonador possa voltar a executar exatamente a partir do ponto de parada sem perda de dados ou inconsistências. Tais informações são armazenadas em estruturas de dados conhecidas como tabela de processos ou descriptor de processos ou bloco descriptor de programa. O bloco descriptor de programa (BCP) consistem de uma estrutura de dados contendo informações importantes sobre o processo. A figura 1 apresenta o BCP contendo os dados do contexto de execução.

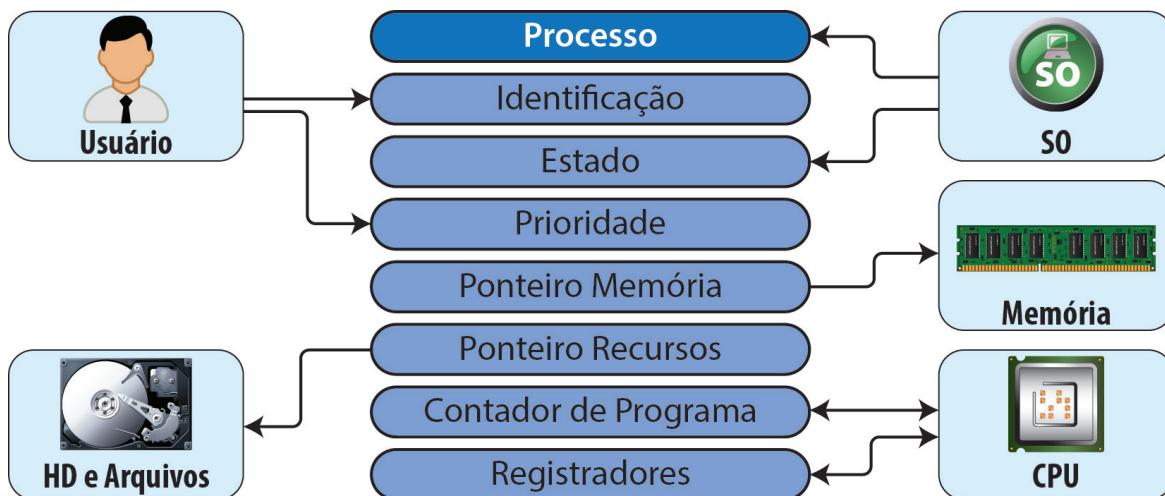


Figura 1. BCP contendo dados do contexto de execução

A troca entre processos concorrentes pela disputa da CPU chama-se *mudança de contexto*. Quando o processo é interrompido pelo sistema operacional, seu contexto é salvo no seu BCP. Ao retornar a execução, o sistema operacional restaura o contexto do processo, o qual continua a executar como se nada tivesse ocorrido. A figura 2 apresenta os sistemas logicamente paralelos concorrentes em uma única CPU.

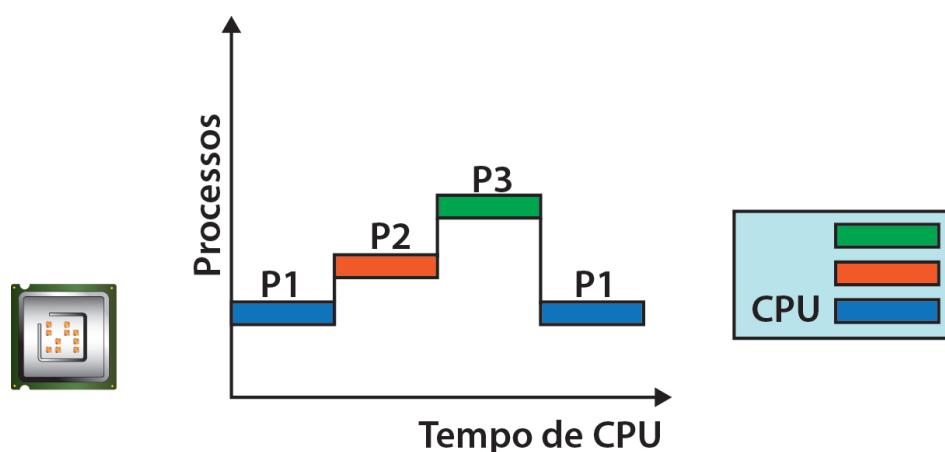


Figura 2. Sistemas Lógicamente Paralelos

Um dos principais requisitos de um SO é intercalar ou fazer a mudança de contexto entre processos visando maximizar a utilização da CPU fornecendo um razoável tempo de resposta. Atualmente, tem se popularizado os computadores com múltiplas CPUs que compartilham os demais recursos da máquina, tais como memória principal e dispositivos de entrada e saída. A figura 3 apresenta os processos em um sistema fisicamente paralelo.

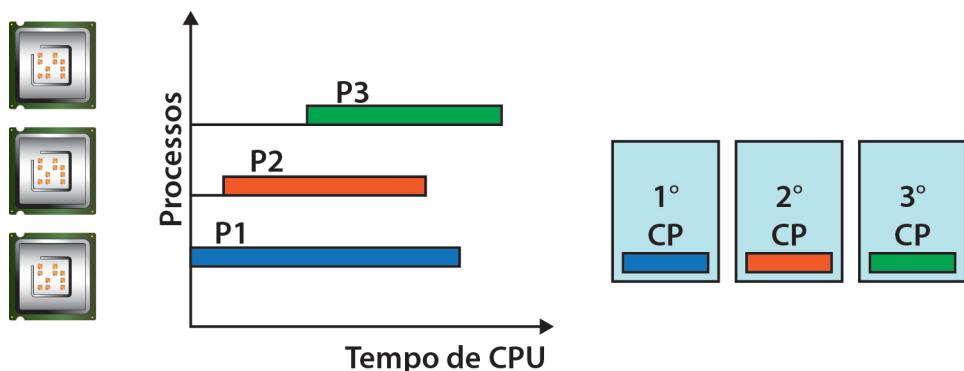


Figura 3. Sistemas Fisicamente Paralelos

O SO deve evitar que os processos entrem em *deadlock* e permitir a criação e comunicação entre processos. Os processos após sua criação podem assumir alguns estados. Os primeiros SO multiprogramados tinham a previsão de poucos estados, como o criação, o em execução, o estado pronto, o estado bloqueado e o estado encerrado, conforme ilustrado na figura 4.

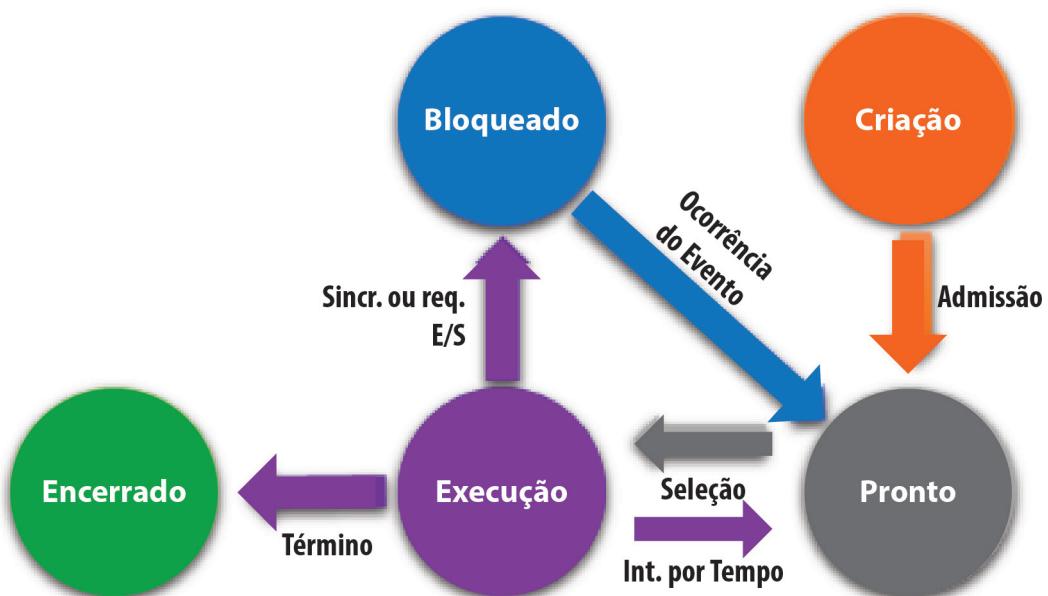


Figura 4. Estados dos processos de um SO

Quando um usuário solicita a execução de um programa o SO cria um processo com a identificação do programa e deverá criar o contexto de execução para poder escalonar ou seja colocar ele em execução. Quando o SO conclui todos os preparativos para rodar o processo ele muda o estado do processo de criação ou indefinido para pronto que este fica a disposição do escalonador do *kernel* do SO.

O software escalonador deverá cuidar dos processos definindo quem deverá permanecer em execução. Uma das políticas para o escalonador é dar fatia de tempo iguais a todos os processos mudando o estado do processo de pronto para em execução. Quando o processo está em execução, ele tem todos os recursos da CPU para utilizar, ou seja, está em seu estado progressivo tendo um andamento normal. Enquanto não ocorrer uma chamada de entrada e saída ou enquanto não findar sua fatia de tempo o processo deverá permanecer neste estado.

Ao terminar sua quota de tempo para execução, o escalonador o interrompe, colocando-o no estado pronto, ou seja, o processo só não está em execução pelo fato da CPU estar sendo utilizada por outro processo. Ao chegar novamente sua vez de executar, o escalonador invoca ou acorda o processo permitindo sua continuidade. Um processo pode ser bloqueado se os dados de entrada ou saída, ainda, não estiverem disponíveis ou ele estiver parado à espera da ocorrência de um evento ou ele não poder continuar sua execução ou andamento progressivo.

O processo pode ir para o estado bloqueado informando ao escalonador que entrou em espera, simplesmente, enviando um sinal solicitando que outro processo entre em execução ou por decisão do escalonador. O processo é desbloqueado por um evento externo como, por exemplo, a chegada dos dados ou sinalização de outro processo e ele então passa para o estado pronto.

Para efetuar o compartilhamento da CPU entre processos, o SO possui duas filas de controle: a de processos prontos ou *ready-list* e a de processos bloqueados ou *blocked-list*. A manipulação dessas filas depende da política de escalonamento adotada pelo sistema. Um escalonador poderia funcionar da seguinte forma: quando a CPU torna-se disponível, o primeiro elemento da fila de processos prontos é retirado e inicia sua execução. Caso seja bloqueado, este irá para o final da fila de processos bloqueados. Se a sua quota de execução se esgotar, este será retirado da CPU e colocado no final da lista de processos prontos.

O escalonador ou *scheduler* do SO é o elemento responsável pela alocação de processo(s) no(s) processador(es), definindo sua ordem de execução. A política de escalonamento ou *scheduling* em inglês é um problema complexo e depende do tipo de sistema suportado e da natureza das aplicações. Em sistemas do tipo *batch*, o escalonamento era feito simplesmente selecionando o próximo processo na fila de espera. Em sistemas multiusuário de tempo repartido, geralmente combinados a sistemas em *batch*, o algoritmo de escalonamento deve ser mais complexo em virtude da existência de diversos usuários solicitando serviços e da execução de tarefas em segundo plano ou *background*.

Parâmetros de Escalonamento



Em um sistema multitarefa, vários programas compartilham a mesma CPU e, portanto, num dado instante de tempo somente um processo estará executando e vários processos podem estar a espera de sua fatia de tempo. Isto introduz filas de processos no estado de pronto. Quando estoura a fatia de tempo do processo em execução, o SO deve decidir qual processo da fila de prontos deverá receber a CPU. A parte do SO responsável por decidir qual processo, dentre os prontos, deve ganhar o direito de uso da CPU é denominada escalonador de processos ou *scheduler*. O escalonador ordena a fila de prontos de acordo

com sua política de escalonamento, a qual pode levar em consideração a prioridade do processo, a sua ordem de chegada no sistema, seu prazo para ser atendido etc.

O escalonamento de um SO pode ser classificado como preemptivo e não-preemptivo. O escalonamento é dito preemptivo se o processo em execução na CPU puder ser interrompido para a execução de outro processo. Preempção é utilizada em sistemas multitarefa para garantir que todos os processos possam progredir e para evitar que um processo monopolize a CPU. E o escalonamento é dito não-preemptivo se durante a execução de um processo a CPU não puder ser liberada para outro processo.

Diversos critérios devem ser considerados para a implementação de um bom algoritmo de escalonamento. Alguns deles são:

- Justiça: garantir que cada processo tenha direito de acesso a CPU;
- Eficiência: procurar maximizar a utilização da CPU;
- Tempo de Resposta: procurar minimizar o tempo de resposta para aplicações interativas. O Tempo de resposta é o tempo decorrido entre o momento em que um usuário submete uma tarefa ao sistema e instante em que ele recebe de volta os resultados;
- *Throughput* ou vazão: maximizar o número de tarefas processadas por unidade de tempo;
- *Turnaround* ou tempo de utilização da CPU: a gestão estratégica do tempo de utilização da CPU por job procura minimizar o tempo de execução das tarefas do tipo lote.

Um dos algoritmos de escalonamento mais conhecidos é o **FIFO** do inglês *First In First Out* ou em português primeiro a entrar na fila é o primeiro a sair. Em um SO do tipo FIFO os processos vão sendo colocados na fila e retirados por ordem de chegada. A ideia fundamental é a de uma fila, em que só se pode inserir um novo elemento no final da fila e só se pode retirar o elemento do início. Neste algoritmo, os processos são selecionados a partir da sua ordem de chegada, ou seja, o primeiro a chegar é o primeiro a ser servido, dai o motivo do mesmo algoritmo receber o nome de *First Come First Served (FCFS)*. O mecanismo adotado pelo escalonador é não-preemptivo, ou seja, as tarefas ao conseguirem a CPU executam até o final. Processos maiores fazem com que processos menores esperem em demasia e devido a igualdade total entre tarefas, processos mais importantes não tem acesso privilegiado a CPU. A falta de garantia quanto ao tempo de resposta torna-o inadequado para sistemas interativos, porém pode ser utilizado em sistemas *Batch*.

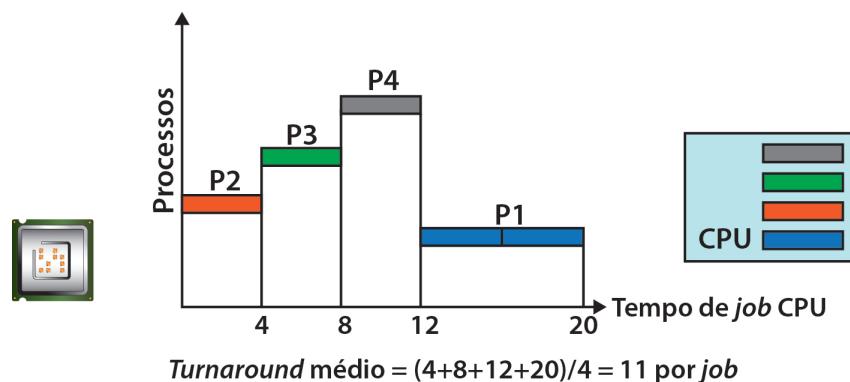
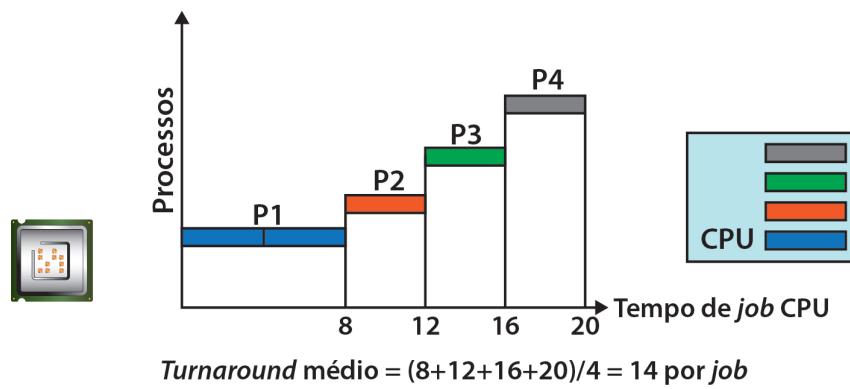
Outro algoritmo muito conhecido para o escalonamento é *Round Robin* ou circular. É um algoritmo antigo, porém justo e simples. O processo é colocado para executar na ordem em que fica pronto para execução, porém só pode executar por uma fatia de tempo, após a qual o processo é interrompido e, caso ainda não tenha terminado sua execução, volta para o final da fila de prontos. Um dos problemas do algoritmo *Round Robin* é determinar a fatia de tempo ou como alguns autores preferem chamar a fatia de tempo de quantum. Um quantum muito pequeno pode levar a sucessivas trocas de contexto reduzindo a eficiência do processador. Um quantum muito grande pode se tornar inviável para sistemas interativos elevando o tempo de resposta.

Como nem sempre todos os processos tem a mesma prioridade o algoritmo de escalonamento com prioridade passa a ser interessante. A ideia do algoritmo é simples, a cada processo é associada uma prioridade e o processo pronto com maior prioridade é executado primeiro. Para evitar o monopólio do processador pelo processo com maior prioridade a cada quantum o sistema deverá decrementar

a prioridade do processo em execução. O escalonamento por prioridade pode ser por prioridade estáticas ou por prioridade dinâmicas. A maioria dos SO de tempo compartilhado implementa um esquema de prioridades. Pode ser conveniente, algumas vezes, agrupar os processos em classes de prioridades e usar o escalonamento entre as classes e o round robin dentro das classes.

Um dos mais antigos escalonadores com prioridade foi projetado para o SO CTSS. O CTSS só mantinha na memória principal um processo pequeno, logo foi necessário atribuir um quantum longo para processos que utilizavam muito a CPU como uma forma de reduzir a *swap*. Para evitar o tempo de resposta ruim o SO dividiu os processos em classes de prioridades. As classes com maior prioridade rodavam 1 quantum, a próxima 2 quantum, a outra 4 quantum e assim sucessivamente. Outro SO que atribuiu classes de prioridade foi o XDS 940, onde as classes terminal e de entrada e saída tinha maior prioridade que as demais.

O algoritmo mais apropriado para sistemas que executam jobs em *batch* é o de menor tarefa primeiro ou *Shortest Job First* (SJF). Neste algoritmo, o processo com menor tempo de execução previsto é o próximo escolhido para executar. O algoritmo é não-preemptivo e visa reduzir o tempo médio de espera das tarefas. O maior problema do algoritmo reside na estimativa prévia do tempo de execução do processo. Alguns processos *em batch* que executam regularmente como folha de pagamento e contabilidade, tem geralmente tempos de execução conhecidos, porém processos interativos normalmente não usufruem dessa propriedade. Considere o caso ilustrado na figura 5 que apresenta 4 jobs com seus tempos de execução e compare com o a figura 6 que apresenta os mesmos jobs mas agora com o algoritmo da menor tarefa primeiro e observe o tempo médio menor de CPU por job.



É interessante notar que o algoritmo do menor *job* só conduz resultados ótimos se todos os *jobs* estão disponíveis ao mesmo tempo. Essa abordagem pode ser utilizada em sistemas interativos, tendo que considerar que cada comando enviado pelo usuário passa a ser um *job* e deverá ter o seu tempo estimado a partir de seu comportamento no passado. Uma maneira de estimar com mais facilidade é utilizar a técnica chamada de *aging*. Suponha que o tempo para um comando de terminal na primeira vez que foi chamado seja T_0 . Agora, na rodada seguinte o tempo medido seja T_1 . Podemos atualizar nossa estimativa considerando $aT_0 + (1-a)T_1$. A escolha fácil de implementar é fazer $a = \frac{1}{2}$. Assim já na quarta estimativa teríamos:

$$1^{\circ} - T_0$$

$$2^{\circ} - T_0/2 + T_1/2$$

$$3^{\circ} - T_0/4 + T_1/4 + T_2/2$$

$$4^{\circ} - T_0/8 + T_1/8 + T_2/4 + T_3/2$$

Note que o peso de T_0 ou o tempo inicial com provavelmente o maior erro de estimativa caiu a $1/8$ e a estimava tende a ficar mais exata a cada interação.

Para sistemas de tempo real é comum encontrar o escalonamento por prazo ou *deadline*. Os processos são escolhidos para execução em função de suas urgências. Esta política é utilizada em sistemas *real-time* onde, a partir do momento em que o processo entra na fila de prontos, inicia seu prazo para receber a CPU. Caso não seja atendido dentro desse prazo pode haver perda de informação. Os processos com escalonamento garantidos são similares aos escalonadores de tempo real, pois é feita uma promessa ao usuário a respeito do desempenho que tem de ser cumprida. Muitos SO prometem que o escalonador irá atribuir a n usuários ativos $1/n$ da capacidade do processador.

Algumas vezes é importante separar o mecanismo de escalonamento da política de escalonamento. Por exemplo, quando um processo-pai possui vários processos-filhos, pode ser importante deixar o processo-pai interferir no escalonamento do processo filho. Para isso, deve-se ter um escalonador parametrizado, cujos parâmetros são passados pelos processos do usuário.

Se não existir memória disponível no sistema alguns processos devem ser mantidos em disco. Esta situação tem grande impacto no escalonamento devido ao atraso provocado pelo *swapping*. Uma forma prática para lidar com o *swapping* é através do escalonamento em dois níveis: um para o subconjunto dos processos mantidos na memória principal e outro para os processos mantidos em disco. O escalonador de mais baixo nível se preocupa em escolher qual processo da memória usará a CPU enquanto que o escalonador de mais alto nível se preocupa em fazer a troca dos processos da memória com os em disco.

Interrupção



Para que se possa realizar o escalonamento de processos, é necessário um mecanismo para interromper a execução do processo atualmente usando a CPU e fazer com que a CPU passe a executar as instruções do SO para que ele possa escalonar um novo processo. Este mecanismo é denominado interrupção.

Portanto, interrupção é um evento gerado pelo *hardware* ou por *software* e que altera a sequência na qual o processador executa as instruções. Por exemplo, toda máquina possui em clock que é um dispositivo de hardware que gera uma interrupção em tempos regulares no PC 18 vezes por segundo, fazendo com que a CPU pare de fazer o que estava fazendo e execute uma rotina que incremente a hora e data do sistema. Outros tipos de interrupção são:

- interrupção de software: quando um processo solicita um serviço do SO;
- interrupção de entrada e saída (E/S): geradas por dispositivos de E/S tais como impressora, teclado, drive de disco, para indicar ao SO alterações no seu estado final de escrita ou leitura de dados, falha no dispositivo, etc;
- traps: interrupções causadas por erros na execução de programas, tais como tentativa de divisão por zero, *overflow*, entre outras.

O tratamento de interrupções é feito pelo SO. Toda vez que uma interrupção acontece, o SO assume o controle, salva o contexto de execução do processo interrompido no seu BCP, analisa a interrupção e passa o controle para a rotina de tratamento apropriada.

Threads



A ideia da *threads* é levar para dentro das aplicações a multitarefa. A ideia é permitir que determinados trechos de código do mesmo programa possam ser executados de forma concorrente ou paralelas. No modelo tradicional, um programa é composto de uma única *thread*, ou seja, existe apenas uma linha de execução e as instruções são executadas sequencialmente, do início ao final, uma após a outra. O conceito de *thread* implica a possibilidade de passar a executar vários pedaços de um processo ao mesmo tempo. Tudo se passa como se estivesse invocando a execução de uma função. No entanto, enquanto no modelo tradicional ao chamar a função se transfere o controle de fluxo de execução para ela, quando se utiliza uma *thread* simplesmente se ativa a execução da função e prossegue a execução da função invocadora. As duas *threads*, a função chamadora e a chamada, vão ser executadas ao mesmo tempo, concorrendo pelo uso dos recursos alocados ao processo.

Material Complementar

Com as Máquinas Virtuais (VM) que você instalou na Unidade I, inicie no SO Linux a gerência de seus processos. O SO Linux apresenta um dos melhores conjuntos de algoritmos para gerenciamento de processos. Pesquise os comandos, teste e experimente gerenciar seus processos no Linux.



Gerenciamento de Processos no Linux

- http://marcelotoledo.com/stuff/artigos/processos_no_linux/gerenciamento_de_processos_no_linux.html

Comandos para criar, gerenciar, monitor e eliminar processos

- <http://linuxelpi.blogspot.com.br/2010/06/gerenciamento-de-processos.html>

Referências

Tanenbaum, A.S. **Sistemas Operacionais Modernos**. 3º Edição. São Paulo. Pearson Prentice Hall, 2009.

Deitel, H.M. **Sistemas Operacionais**. 3º Edição. São Paulo. Pearson Prentice Hall, 2005.

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca

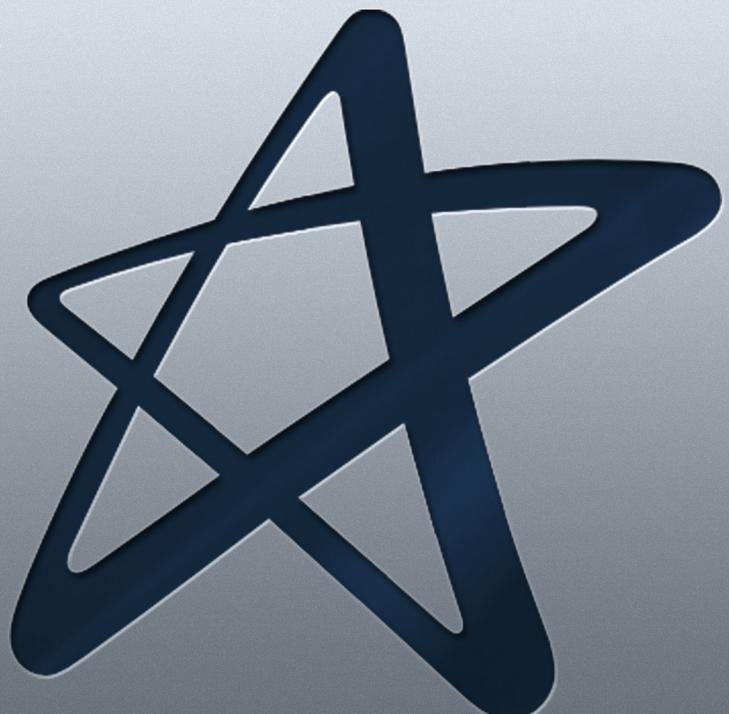


UDF
Centro
Universitário



Módulo
Centro
Universitário

Sistemas Operacionais



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Gerenciamento de Memória

Responsável pelo Conteúdo:
Prof. Ms. Claudney Sanches Junior

Revisão Textual:
Profa. Dra. Patrícia Silvestre Leite Di Iório

UNIDADE

Gerenciamento de Memória



- Memória
- Gerência de Memória
- Memória Virtual
- Segmentação



O objetivo desta unidade será apresentar o gerenciamento da memória em SO e estudar um pouco mais sobre os tipos de memória, sua classificação e como o sistema gerencia a memória primária e secundária. Você será apresentado também à memória virtual, aprenderá sobre paginação e segmentação. Neste contexto, espera-se que ao final da unidade você seja capaz de entender o gerenciamento de memória dos SOs modernos.

Para que possa entender os conceitos de gerenciamento de memória do SO, esta unidade está organizada da seguinte forma:

- seção 2, apresenta o conceito e tipos de memória;
- seção 3, mostra a gerencia de memória;
- seção 4, detalha a memória virtual;
- a seção 5, apresenta as paginação e segmentação.

Ao final do estudo e das atividades desta unidade, você deve ser capaz de:

- entender a gerência da memória em SOs;
- conhecer o gerenciador de memória e seus principais algoritmos.

Contextualização



Uma empresa comprou uma CPU com muita memória física. Faça um documento no word com os passos necessários para desabilitar no Windows a Memória Virtual. Demostre, capturando as telas, que você consegue desabilitar e utilizar o SO. Isto é, ilustre, com imagens da tela, os passos necessários para a desabilitação da Memória Virtual.

Memória



Antes de começar a falar do gerenciamento de memória do SO, precisamos revisar os conceitos de memória, memória principal e memória secundária. O primeiro conceito a se guardar é que ao se referir à memória em computação estamos nos referindo a todos os dispositivos que permitem a um computador guardar dados temporariamente ou permanentemente. O termo memória é genérico servindo tanto para o armazenamento de dados como para o armazenamento de programas.

As memórias de alta velocidade, localizadas no processador que guardam dados para uso imediato, são as mais velozes e caras de um sistema computacional, pois operam na mesma velocidade dos processadores e recebem o nome de registradores.

Como existe uma diferença de velocidade muito grande dos registradores em relação à memória principal, surgiu a necessidade de um tipo de memória interna que intermedia o processador e o dispositivo de armazenamento, normalmente, com um serviço que antecipa a probabilidade de dados serem utilizados novamente. Esta memória mais lenta que os registradores e mais rápida que a memória principal recebeu o nome de Cache.

Com o avanço tecnológico, várias caches foram desenvolvidas, sendo, algumas conhecidas e outras não. As que daremos destaque são as caches L1, L2 e L3. As memórias caches são medidas por sua capacidade de armazenamento e sua latência. Latência é o tempo decorrido entre um ciclo de *clock* da máquina e o tempo de transferência de dados. A latência é mediada em nanosegundos ou em ciclos de processador, ou seja, ciclos de *clock* que o processador tem que rodar sem executar nenhuma operação, pois fica aguardando a memória. Assim, a Cache L1 demora 2 ou 3 ciclos para responder a uma solicitação, enquanto que se um dado for solicitado a cache L2 demorará 10 ciclos. Assim, os algoritmos de previsão de uso são também de extrema importância.

A memória principal consiste em memória volátil de acesso aleatório (*Random Access Memory* – RAM). O termo aleatório vem do sentido de que os processos podem acessar dados em qualquer ordem. O que diferenciava os sistemas de armazenamento em fita é que os dados somente eram lidos em uma determinada sequência, mas foram os meios de armazenamento populares dos primeiros computadores.

Você Sabia ?



Todos usamos *caching*. Guardamos coisas próximas em lugares estratégicos para ter acesso fácil e rápido. Por exemplo, deixar um lápis ou caneta e papel sobre a mesa do escritório é uma forma de *caching*. Contudo, os projetistas de SO tem que tomar muito cuidado ao utilizar *caching*. A cache em computação é uma cópia do dado armazenado sendo que está cópia sofrerá constante alteração e o original não. Portanto, o SO tem que frequentemente copiar os dados da memória cache para o original – esse processo é denominado esvaziamento de cache.

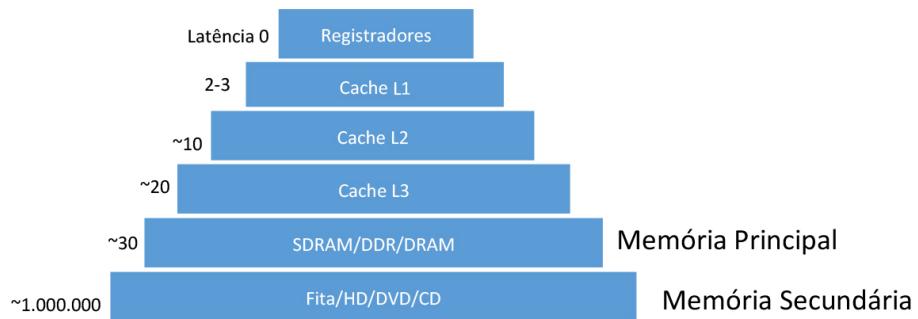
Toda memória principal ou RAM inicia com uma classificação que pode ser SIMM (*Single In-line Memory Module*) ou DIMM (*Double In-line Memory Module*), dependerá do pente de memória, ou seja, a base em que os chips são soldados. Para saber de qual se tratava basta olhar o pente que pode ter chips de memórias RAM soldadas de apenas um lado (*Single*) ou de ambos os lados (*Double*) (DEITEL, 2005).

Ainda, a memória principal pode ser classificada quando a frequência e sincronização com o barramento. Se a memória trabalha na mesma frequência do processador obedecendo ao mesmo ciclo de *clock*, sincronizando a saída de dados com os demais componentes do computador, recebe o nome de SDRAM (*Synchronous Dynamic Random Access Memory*). As memórias assíncronas ou simplesmente DRAM (*Dynamic Random Access Memory*) tem um custo mais baixo e conseguem armazenar mais dados no mesmo espaço, mas devido ao suporte a múltiplos pentes das SDRAM suplantou a DRAM.

O suporte a múltiplos pentes propicia que enquanto um atenda uma solicitação de leitura de um dado outro pode já enviar uma resposta a outra solicitação, de forma que o barramento tem uma alimentação continua. Outra característica que diferencia a DRAM é que a mesma necessita de um circuito de *refresh* ou renovação do sinal algumas vezes em um milissegundo para não perder os dados enquanto que a SDRAM não necessita de tal recurso (DEITEL, 2005).

Para finalizar a classificação de memória principal, devemos pensar na entrega dos pentes de memória. A memória poderia entregar o dobro de dados em uma só transmissão. Assim, surgiu o padrão DDR (*Double Date Rate*) que dobrou a taxa de transferência de dados. Depois surgiram a DDR2 e DDR3 que aumentaram este fator para 4x e 8x respectivamente. E já foi lançada a DDR4 que deverá chegar à produção em massa após 2.014.

A memória secundária é uma expansão da memória principal menos dispendiosa e mais lenta. A latência do armazenamento em disco (HD) é, normalmente, medida em milissegundos, em geral um milhão de vezes mais lento do que as memórias cache que são memórias colocadas próximas aos processadores com latência de 10 a 20 ciclos de *clock*. A figura 1 apresenta a latência das memórias (DEITEL, 2005).



Uma vantagem dos dispositivos de armazenamento secundário é que eles tem grande capacidade de armazenamento e os dados são guardados em meio persistente, portanto preservados quando se retira a fonte de energia do dispositivo (TANENBAUM, 2009).

Gerência de Memória



O que todo programa deseja é dispor de maneira infinita, rápida, não volátil e a um baixo custo de uma memória que pudesse conter todo seu conteúdo. A função do SO é abstrair a hierarquia e latência das memórias existentes em um modelo útil e então gerenciar essa abstração. (TANENBAUM, 2009).

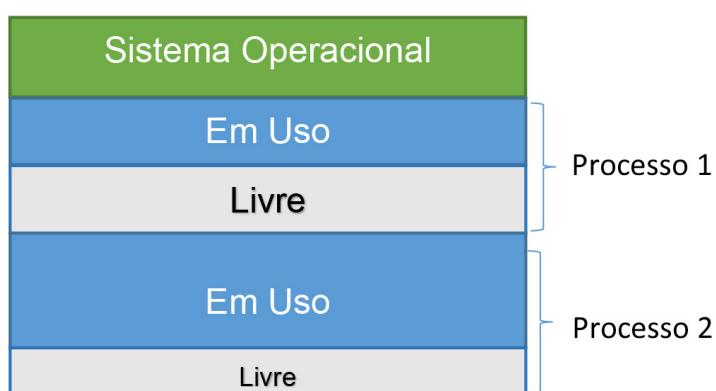
A parte do SO que gerencia parcialmente a memória do computador é denominada Gerenciador de Memória. Entre suas tarefas, podemos citar a tarefa de alocar memória para os processos quando estes precisam, a tarefa de liberar memória quando um processo termina e tratar a tarefa de tratar do problema de swapping. O gerenciador de memória é o componente do SO que se preocupa com o esquema de organização da memória e com a estratégia do gerenciamento.

Programas
tendem a se
expandir e ocupar
toda a memória
disponível. Lei de
Parkinson.

O Gerenciador de Memória tem como primícias otimizar a utilização da memória principal. Isso era fácil nos primeiros SO que simplesmente utilizavam a memória física disponível (1960-1970). Neste período não era possível rodar mais de um programa por vez, pois o SO entregava a gerência dos endereços ao programa que estava em execução e o programa do usuário podia acessar os endereços que estavam alocados para o próprio SO.

Para permitir que mais de um programa execute simultaneamente, o SO deve resolver dois problemas: o de segurança e o de realocação dos endereços atribuídos ao programa. A solução é criar um espaço de endereçamento para cada processo. Assim, o endereço 28 na memória de um processo é diferente do endereço 28 em outro processo. Uma forma simples de criar os espaços é, via hardware, criar dois registradores um registrador-base e outro registrador-límite e cada processo deverá somar o registrador-base em seus endereços e comparar o resultado com o registrador-límite para ver se está dentro do espaço de endereçamento reservado para ele. Apenas o SO poderá atribuir valores para esses registradores.

O primeiro PC 8088 utilizava parte desta solução denominada de realocação dinâmica, pois tinha apenas o registrador-base (TANENBAUM, 2009). O maior problema desta solução é a fragmentação da memória conforme ilustrado na figura 2 que acaba desperdiçando memória, por isso surgiu a necessidade de tentar outra solução.



O grande problema desta solução que adotava limites fixos de memória é a fragmentação e a limitação de processos ativos, pois o SO dividia toda a memória quando era carregado e alocava partes iguais aos processos. Adotando-se um número variável para o espaço da memória um número maior de processos podem ser alocados. Essa solução demanda do Gerenciador de Memória um algoritmo de alocação da memória livres. Assim, os novos processos sempre poderão ser alocados.

Elaboraram-se três métodos de seleção de uma região livre: Melhor Escolha (*best fit*), Pior Escolha (*worst fit*) e Primeira Escolha (*first fit*). O método melhor escolha coloca o processo na menor região livre, o pior escolha coloca o processo com a maior região livre e o primeira escolha aloca a primeira região livre para o processo. Veja a figura 3 que apresenta um processo que demanda 14 kbytes em uma memória com 5 processos em uso.

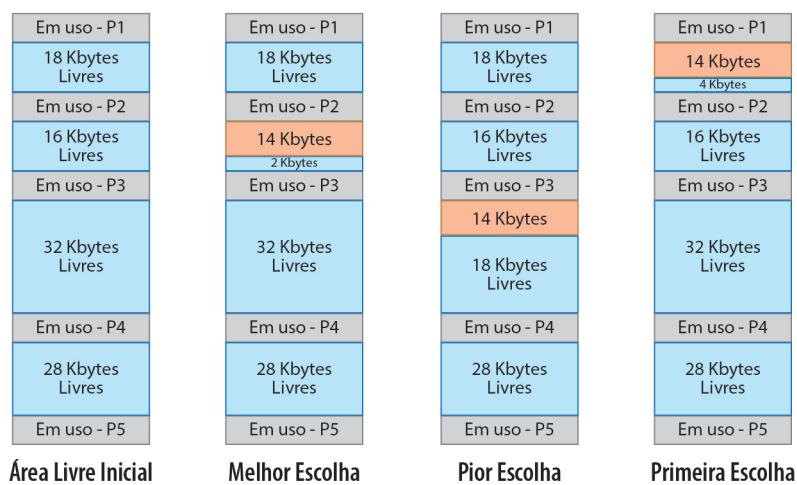


Figura 3. Gerencia de Memória – Alocação de Região Livre

O uso do algoritmo de seleção de memória livre melhor escolha deixa o menor resto, porém após um longo período de processamento deixará espaços muito pequenos na memória para serem úteis a algum processo. O algoritmo pior escolha deixa o maior espaço após cada locação, mas tende a espalhar as porções não utilizadas sobre área não contínua de memória tornando difícil alocar grandes processos. O algoritmo primeira escolha tende a ser um meio termo com característica adicional de fazer com que os espaços vazios migrem para o final da memória. O gerenciador de memória deverá com qualquer uma dessas abordagem manter uma lista de chamada “Lista Livre” dos blocos disponíveis com informações sobre o seu tamanho. Antes de retornar um bloco à lista livre, o gerenciador de memória deve verificar se o bloco liberado está próximo a outros blocos de forma que possam ser combinados, formando um bloco maior.

Memória Virtual



A maioria do SO enfrenta a dificuldade da falta de memória física para a demanda dos processos ativos. Por exemplo, facilmente ao ligar o seu computador com o SO Windows ou SO Linux, uns 50 – 200 MB serão alocados. A estratégia de manter um processo na memória principal até sua finalização recebe o nome de swapping que pode ser traduzida por troca ou permuta. O Gerenciador de Memória deverá trocar os dados que estão na memória principal (RAM) por dados que estão na memória secundária (HD) e utilizar a memória secundária como uma extensão da memória principal.

Com o tempo, o Gerenciador de Memória foi aperfeiçoado para antecipar a necessidade das trocas, permitindo que vários processos permanecessem na memória principal ao mesmo tempo. Os novos algoritmos de swapping trocam um processo somente quando outro precisar daquele espaço de memória, visto que a memória secundária é mais lenta. Com uma quantidade de memória principal suficiente, esses sistemas reduzem muito o tempo gasto nas trocas. Os sistemas de swapping do início da década de 1960 levaram a uma nova estratégia – a memória virtual com paginação.

A estratégia da Memória Virtual é permitir que a memória física e memória secundária sejam combinadas numa nova e única memória. A ideia básica é que cada programa tenha seu espaço de endereçamento dividido em blocos chamados de páginas. Cada página é uma série continua de endereços. Estas páginas têm seus endereços mapeados principalmente na memória física, de forma que ao acessar um endereço que tenha o equivalente em memória física o sistema apenas passa o endereço físico correspondente. Caso o endereço seja da parte gravada na memória secundária, o sistema irá solicitar a troca de forma que os dados que estão na memória física serão gravados na memória secundária e os dados da memória secundária serão carregados para a memória física. Quando o processo finalizar as trocas, o sistema informa o endereço físico correspondente (DEITEL, 2005).

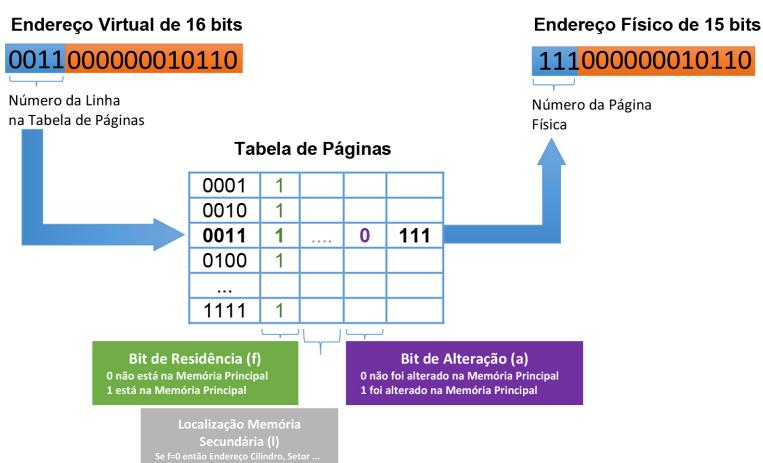
Glossário



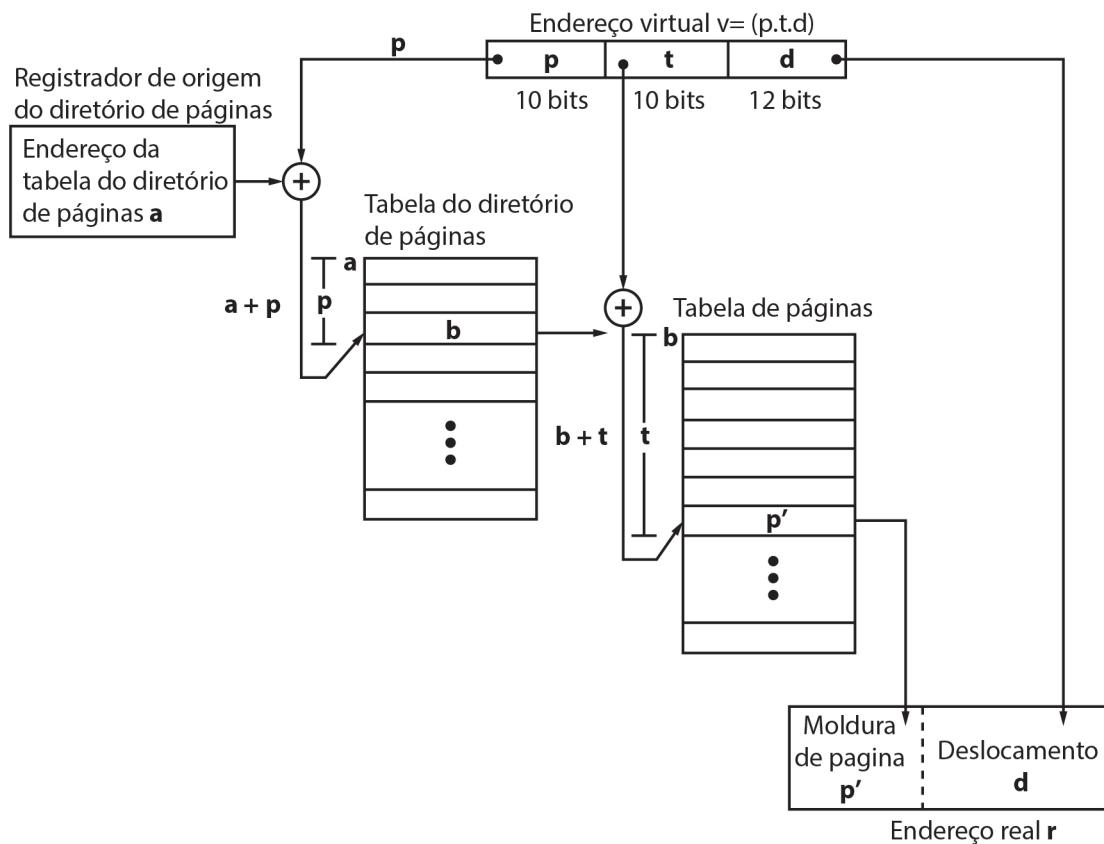
- Espaço de endereçamento Virtual: são os endereços que o programa pode referenciar;
- Espaço de endereçamento Físico: são os endereços reais de memória;
- Tabela de Páginas: relaciona os endereços virtuais com os endereços físicos.

A paginação tem como premissa que existe memória secundária disponível e suficiente para manter o programa completo com seus dados. A cópia do programa na memória secundária pode ser considerada o original enquanto que as partes trazidas da memória principal de vez em quando podem ser consideradas copias. Quando modificações são feitas na memória principal, essas modificações devem ser refletidas na memória secundária. O espaço de endereçamento virtual é dividido em páginas de tamanhos iguais. Sempre que um processo solicitar o acesso a um endereço, ele está solicitando através de um endereço virtual, que deve ser traduzido em um endereço físico.

Isso acontece com tanta frequência que solicitar ao processador fazer a tradução custaria na performance, de modo que o sistema de memória virtual tem que ter um hardware para este propósito especial. Existe a unidade de gerenciamento de memória (Memory Management Unit – MMU) que mapeia rapidamente endereços virtuais para endereços reais e o mecanismo de tradução dinâmica de endereços (Dynamic Address Translation – DAT) que convertem endereços virtuais em físicos durante a execução. A tradução de um endereço de memória virtual para um endereço físico é feita como na Figura 4. O sistema mantém uma tabela de mapas de blocos para cada processo.



O uso dessa solução pode levar a tabelas muito extensas e tornar a MMU muito lenta. Para ilustrar: de 32 bits de endereços e páginas de 4K tem-se 1 milhão de entradas na tabela. Para solucionar este problema, a ideia básica é manter todas as tabelas na memória optando pelo uso de dois apontadores e um deslocamento. Assim, o primeiro apontador é do Diretório (p) e o segundo (t) aponta para os quadros e o terceiro para o deslocamento (d) (DEITEL, 2005).



Segmentação



A segmentação parte do princípio que um programa pode ser dividido em dados e instruções e que estes podem ser armazenados em blocos chamados segmentos. Os segmentos não precisam ser do mesmo tamanho, nem ocupar posições adjacentes na memória principal. Um segmento que corresponda a um array é tão grande quanto o array. O segmento gerado para conter um código é do tamanho do código. O sistema gerenciador de memória segmentado mantém na memória principal apenas os segmentos necessários para a execução em um determinado instante. Um processo pode executar enquanto suas instruções e dados estiverem localizados na memória principal. Se o processo referenciar um segmento que não está na memória principal, o gerenciador deverá recuperar o segmento solicitado. Um segmento que chegar poderá ser alocado para qualquer área disponível na memória principal que for grande suficiente para contê-lo (DEITEL, 2005).

Há muitas estratégias para implementar a tradução de endereços de segmentação, pois um sistema pode empregar mapeamento direto, associativo ou mapeamento combinado/associativo. Algumas vantagens da segmentação são:

- facilidade de compilação e ligações entre os procedimentos separados em segmentos;
- a mudança do tamanho de um segmento não afetará os demais;
- a segmentação facilita partilhar dados entre vários processos;
- o programador tem ciência do conteúdo do segmento e pode protegê-lo.

A desvantagem é que os segmentos dependem dos processos, mas os blocos de memória são recursos de hardware e seu tamanho é dependente do sistema. Uma solução para este problema é a combinação de segmentação com paginação.

Material Complementar

Com a Máquinas Virtuais (VM) que você instalou na Unidade I, inicie no SO Windows a gerência de memória virtual. O SO Windows apresenta uma interface amigável para gerenciar Memória Virtual. Pesquise os comandos, teste e experimente modificar e desabilitar a Memória Virtual.



Memória Virtual

- <http://canaltech.com.br/o-que-e/windows/O-que-e-e-como-gerenciar-a-memoria-virtual-do-Windows/>

Comandos para alterar o tamanho

- <http://windows.microsoft.com/pt-br/windows-vista/change-the-size-of-virtual-memory>

Referências

TANENBAUM, A.S. **Sistemas Operacionais Modernos**. 3 ed. São Paulo; Pearson Prentice Hall, 2009

DEITEL, H.M. **Sistemas Operacionais**. 3 ed. São Paulo; Pearson Prentice Hall, 2005

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca

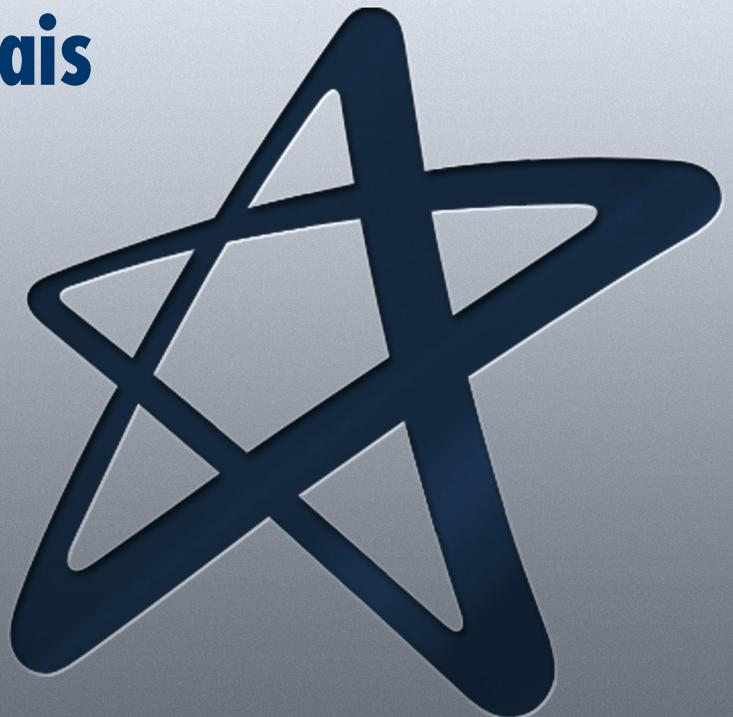


UDF
Centro
Universitário



Módulo
Centro
Universitário

Sistemas Operacionais



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Gerenciador de Arquivos

Responsável pelo Conteúdo:

Prof. Ms. Claudney Sanches Junior

Revisão Textual:

Profa. Ms. Magnólia Gonçalves Mangolini



- Introdução
- Arquivo
- Sistemas de Arquivos
- Diretórios
- Metadados
- Montagem



- Nesta unidade você deverá estudar os conceitos do gerenciador de arquivos dos SO.
- A unidade apresenta a definição de arquivo e mostra o gerenciamento dos diversos dispositivos de armazenamento. Detalha sistema de diretório e apresenta o conceito de metadados e montagem de sistemas de arquivos.
- Neste contexto, espera-se que ao final da unidade você seja capaz de entender o gerenciamento de arquivos dos SOs modernos.

Para que você consiga atingir o objetivo desta unidade, sugiro o seguinte plano de estudos:

1. Leia cuidadosamente a contextualização e o material teórico relativo a esta unidade.
2. Assista a apresentação narrada, onde serão expostos os principais conceitos de Gerenciamento de Arquivos.
3. Realize as atividades de sistematização, que serão compostas de 6 questões de múltipla escolha, sobre os conceitos-chave tratados na unidade. Quando o gabarito das questões for liberado, verifique quais foram as suas respostas incorretas e procure novamente pelo conceito no material teórico.
4. Participe ativamente da atividade de aprofundamento, que será realizada por meio de uma atividade da unidade.
5. Consulte aindicações de referências complementares desta unidade e na Máquina Virtual experimente criar uma partição no Linux.

Contextualização

SUP@RTE_

Baseado em histórias
enviadas por @vitiazze
e @F2lipePedroso

VOU TER QUE LEVAR O
SEU COMPUTADOR PRA
MANUTENÇÃO.



PUTZ! TÁ PESADO PRA
CARAMBA.



SE QUISER PODE DELETAR A
PASTA DE MÚSICAS PRA
FICAR MAIS LEVE.



vidadesuporte.com.br

1. Introdução



Todas as aplicações precisam armazenar e recuperar informações. Enquanto um processo tiver executando, ele poderá armazenar uma quantidade limitada de informação dentro do seu próprio espaço de armazenamento, mas para algumas aplicações este espaço é insuficiente, sendo necessário mais espaço de armazenagem. Outra questão é que algumas informações precisam ser retidas por semanas, meses ou anos. E pode ser necessário que múltiplos processos tenham acesso a mesma informação. Assim, para atender a esses problemas surgiu o sistema de gerenciamento de arquivos do SO. A maioria dos usuários está acostumada com os arquivos que são armazenados, na maioria das vezes, na memória secundária, como sendo uma coleção nomeada de dados.

Nesta unidade você vai estudar como o SO organiza, controla e acessa os dados de arquivos para que possam ser lidos rapidamente dos dispositivos que apresentam tempo de latência alto. Também será apresentada como o SO pode proteger os arquivos com regras de segurança e de danos por falta de energia, perda total ou falha nos discos.

Para que possa entender os conceitos de gerenciamento de arquivo do SO, esta unidade está organizada da seguinte forma:

- a seção 2 apresenta o conceito de arquivo;
- a seção 3 detalha o sistema de diretórios;
- a seção 4 mostra metados;
- a seção 5 apresenta montagem de sistemas de arquivos.

Ao final do estudo e das atividades desta unidade, você deve ser capaz de:

- entender o funcionamento do gerenciador de arquivos de SOs;
- conhecer o gerenciador de arquivos na visão de usuário.

Não deixe de utilizar os fóruns associados à unidade para apresentar e discutir qualquer dificuldade encontrada.

2. Arquivo



Um arquivo é um mecanismo de abstração, uma coleção nomeada de dados que pode consistir de um ou mais registros. Um registro físico é a unidade de informação que realmente é lida e escrita em uma unidade de armazenamento.

Um registro físico ou bloco físico é uma coleção de dados que é tratada como uma única unidade pelo software. Quando um arquivo existe em apenas uma única unidade física recebe o nome de registros não blocados. Em um arquivo com registros de tamanho fixo, todos os registros tem o mesmo tamanho e o tamanho do arquivo será um múltiplo inteiro do tamanho dos registros. Em um arquivo com registros de tamanho variado poderá variar o tamanho do registro até o limite do bloco do disco(DEITEL, 2005).

A característica mais importante em um mecanismo de abstração é o nome, e por isso iniciaremos falando sobre a nomeação de arquivos. Quando um processo cria um arquivo, ele dá um nome que irá continuar a existir após o término do processo. Assim, outros processos poderão ter acesso ao arquivo buscando pelo seu nome(TANENBAUM, 2009).

As regras exatas para se dar um nome a um arquivo variam de acordo com o SO, mas todos os SO atuais permitem cadeias mínimas de até oito caracteres, sendo que muitos atualmente permitem cadeias de nomes de 256 caracteres (normalmente em SO de 32 bits ou 64 bits). Alguns SO distinguem letras maiúsculas de minúsculas, como, por exemplo, o SO Unix que pode ter vários arquivos separados e distintos com o mesmo nome, como maria, Maria e MARIA(TANENBAUM, 2009).

Muitos SO suportam os nomes de arquivos em duas partes separadas por um ponto como arquivo.zip. A parte após o ponto recebe o nome de extensão do arquivo e normalmente indica que tipo de software gerou ou consegue ler o arquivo. No Unix o tamanho da extensão, se houver, fica por conta do usuário e poderá ter mais de uma extensão. No Windows só é permitido três ou quatro caracteres após o ponto e mais de uma extensão não é suportada, sendo apenas a última válida. Em alguns SO a extensão é apenas uma convenção que serve de lembrete ao proprietário do conteúdo, como é o caso do Unix, cujo arquivo file.txt indica que o arquivo deve ter algum texto. Isso difere dos SO que a extensão especifica quais programas possui ou é capaz de abrir tal extensão. Assim, um arquivo.doc, ao ser clicado duas vezes, inicia o Microsoft Word para abrir automaticamente o arquivo. A figura 1 apresenta algumas extensões conhecidas(TANENBAUM, 2009).

arquivo.com	Comando externo do MS-DOS (programas curtos)
arquivo.mp3	Música codificada no formato de áudio MPEG – camada 3
arquivo.pdf	Arquivo no formato portátil de documento
arquivo.exe	Arquivo executável, consistindo no arquivo principal do programa
arquivo.txt	É a extensão de qualquer arquivo de texto que não possui qualquer formatação
arquivo.dat	Arquivo de dados, executável apenas dentro de outro programa
arquivo.ico	Arquivo de ícone
arquivo.bmp	Arquivo de imagem de mapa de bits
arquivo.wav	Arquivo com multimédia de áudio
arquivo.avi	Arquivo com multimédia de vídeo
arquivo.htm	Documento da Internet (ou HTML)
arquivo.url	Atalho para site na Internet
arquivo.inf	Arquivo de informações de hardware
arquivo.jpeg	Arquivos de imagem padrão bitmap (jpeg ou jpg) comprimido
arquivo.wmv	Extensão de vídeos do Windows Media Player

Figura 1. Extensões típicas de arquivos

Segundo Deitel (2005), os arquivos podem ser manipulados por operações, tais como:

- abrir – prepara um arquivo para ser referido
- fechar – impedir mais referências a um arquivo
- criar – criar um arquivo
- destruir – remover o arquivo
- copiar – copiar o conteúdo de um arquivo para outro
- renomear – mudar o nome do arquivo
- listar – imprimir o conteúdo do arquivo

Quanto aos dados internos de um arquivo, podem ser manipulados por operações como:

- ler – copiar conteúdo do arquivo para a memória
- escrever – copiar dados da memória para um arquivo
- atualizar – modificar um item de dados do arquivo
- apagar – apagar um item de dados do arquivo

Os arquivos podem ser caracterizados por atributos como:

- tamanho
- localização
- acessibilidade de acesso (quem pode acessar)
- tipo
- volatilidade (frequência de alterações)
- atividade (frequência de uso)

3. Sistemas de Arquivos



Um sistema de arquivos organiza e gerencia o acesso aos dados. Deve garantir que os arquivos armazenados fiquem disponíveis, compartilhados e em segurança. Deve garantir ainda que as informações armazenadas não sejam corrompidas.

Em um sistema de arquivos a sua implementação lógica não é apresentado aos usuários do SO. Ao usuário é apresentado duas interface:uma do tipo texto e outra do tipo gráfica, para manipulação dos arquivos armazenados e gerenciados pelo SO. A interface texto oferece uma linha de entrada de comandos no console de comandos. Os comandos são simples instruções diretas, que são chamadas para a execução, conforme apresentada na figura 2.

Propósito	Windows	Unix
Copiar arquivo	copy	cp
Renomear arquivo	ren	
Apagar arquivo	del	rm
Mover arquivo	move	mv
Exibir arquivo	type	more
Listar arquivo	dir	ls
Criar diretório	md	mkdir

Figura 2. Alguns comandos diretos do console de comandos

Sistemas de arquivos preocupam-se primordialmente com o gerenciamento do espaço secundário de armazenamento, dando prioridade para o disco, mas eventualmente pode acessar outras áreas, como a memória principal.

Os sistemas de arquivos devem ser capazes de habilitar os usuários a compartilhar seus arquivos de modo seguro e controlado, criando, para isso, mecanismo de acesso compartilhado de leitura, escrita, execução e várias combinações desses.

Devem exibir independência do dispositivo, permitindo aos usuários se referir aos seus arquivos por nomes simbólicos em vez de utilizar os nomes dos dispositivos físicos. Nomes simbólicos são nomes lógicos amigáveis ao usuário, como MeuDiretorio:MeuArquivo.txt. Nomes de dispositivos físicos especificam o lugar em que o arquivo pode ser encontrado, por exemplo, disco 1, blocos 132-251. Nomes simbólicos permitem que os usuários atribuam nomes significativos aos arquivos, enquanto que a visão física irá se preocupar com os arranjos dos dados do arquivo nos dispositivos de armazenamento (DEITEL, 2005).

Projetar um sistema de arquivos exige conhecer o tipo, o número de usuários, as características das aplicações que serão utilizadas, o tamanho e as operações sobre os arquivos.

Para evitar perdas, o sistema de arquivos deve fornecer capacidade de gerar e recuperar cópias de segurança ou backup. Além disso, em sistemas onde a segurança é primordial, o sistema de arquivo deve fornecer mecanismo de criptografia e decriptação de forma que a informação seja útil apenas a quem se destina.

Você Sabia ?



Os atuais SO passam quantidades maciças de informações entre computadores, especialmente quando conectados à internet. Os meios de transmissão são inseguros e vulneráveis. Para proteger suas informações, os sistemas de gerenciamento de arquivos dos SO fornecem a capacidade de criptografia e decriptação. Ambas as operações podem usar o processador tão intensivamente que somente com o aumento da capacidade e velocidade do processamento se tornou viável habilitar tais serviços para arquivos que o usuário deseja proteger.

4. Diretórios



Para controlar, organizar e localizar os arquivos, os sistemas de arquivos têm, em geral, diretórios ou pastas, que em muitos sistemas também são arquivos que contêm o nome e as localizações dos arquivos do sistema de arquivos. Diferente de outro arquivo, um diretório não armazena dados do usuário e sim dados do próprio sistema de arquivos.

O modelo mais simples que se conhece é o sistema de arquivo de nível único ou diretório-raiz, que armazena todos os seus arquivos em um único diretório. Neste sistema dois arquivos não podem ter o mesmo nome. Esse foi o sistema implementado nos primeiros computadores pessoais e no primeiro supercomputador do mundo, o CDC6600. Mas hoje é raramente implementado(TANENBAUM, 2009).

Um sistema de arquivos mais apropriado para a maioria dos ambientes é o sistema de arquivamento estruturado hierarquicamente. Neste modelo, uma raiz indica onde começa o armazenamento ou diretório-raiz. Diretórios são arquivos que podem apontar para os vários diretórios de usuários. Um diretório de usuário contém a entrada para cada um dos arquivos daquele usuário, e cada entrada aponta para a localização do arquivo correspondente no dispositivo de armazenamento. Os nomes dos arquivos passam a ser exclusivos somente dentro de um dado diretório de usuário. A figura 3 apresenta a imagem do sistema de arquivamento estruturado hierarquicamente(TANENBAUM, 2009).

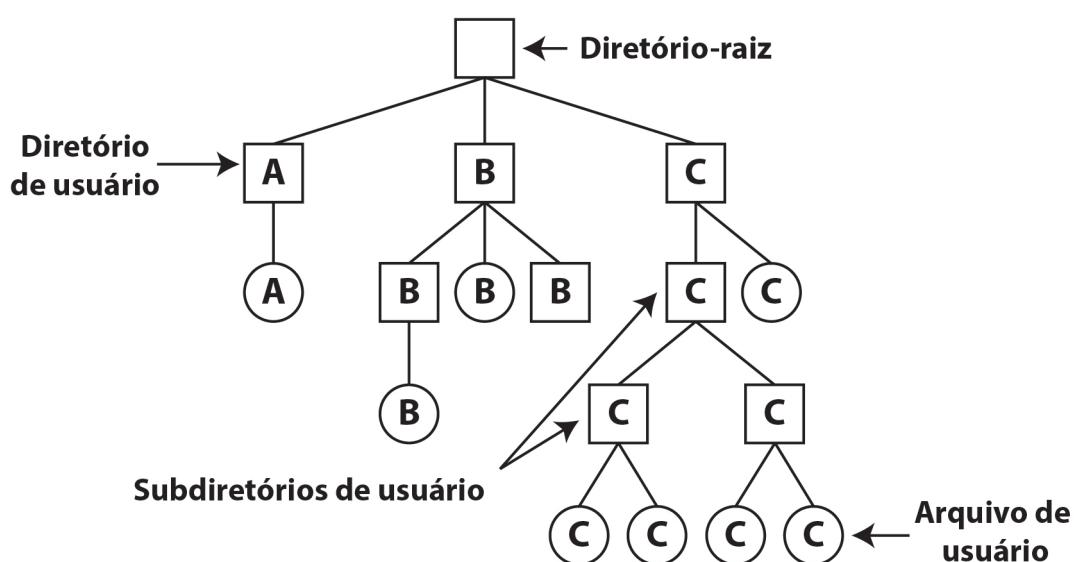


Figura 3. Sistema de Arquivamento Estruturado Hierarquicamente (TANENBAUM, 2009)

Os sistemas hierárquicos de arquivos são implementados na maioria dos SO, mas o nome da raiz e o tipo de delimitador podem variar entre os SO. O sistema Windows utiliza o identificador do diretório-raiz como sendo uma letra seguida de dois pontos (por exemplo, C:), enquanto que o Unix utiliza a barra inclinada (/). Quanto ao delimitador o Windows utiliza a barra inclinada invertida (\) e o Unix utiliza a barra inclinada (/).

Assim, teríamos os nomes de arquivos absolutos:

Windows C:\Users\Claudiney\Documents

Unix /usr/Claudiney/Documents

Os nomes dos caminhos absolutos sempre iniciam nos diretórios-raiz e são únicos. Outro tipo de nome é o de caminho relativo. É usado juntamente com o conceito de diretório de trabalho ou atual. Todos os nomes de caminhos não iniciados no diretório-raiz são assumidos como relativo ao diretório de trabalho. A forma relativa é, muitas vezes, mais conveniente de realizar as mesmas coisas que a forma absoluta(DEITEL, 2005).

A maioria dos SO que suportam o sistema de diretórios hierárquicos tem duas entradas especiais em cada diretório, ‘.’ e ‘..’ sendo que o ponto (.) refere-se ao diretório atual e ponto ponto (..) refere-se a seu pai. Para verificar como pode ser usado, considere que um dado processo tem o diretório de trabalho /usr/ast. O ponto ponto(..) pode ser utilizado para subir um nível na árvore. A figura 4 apresenta a árvore de diretórios do Unix.

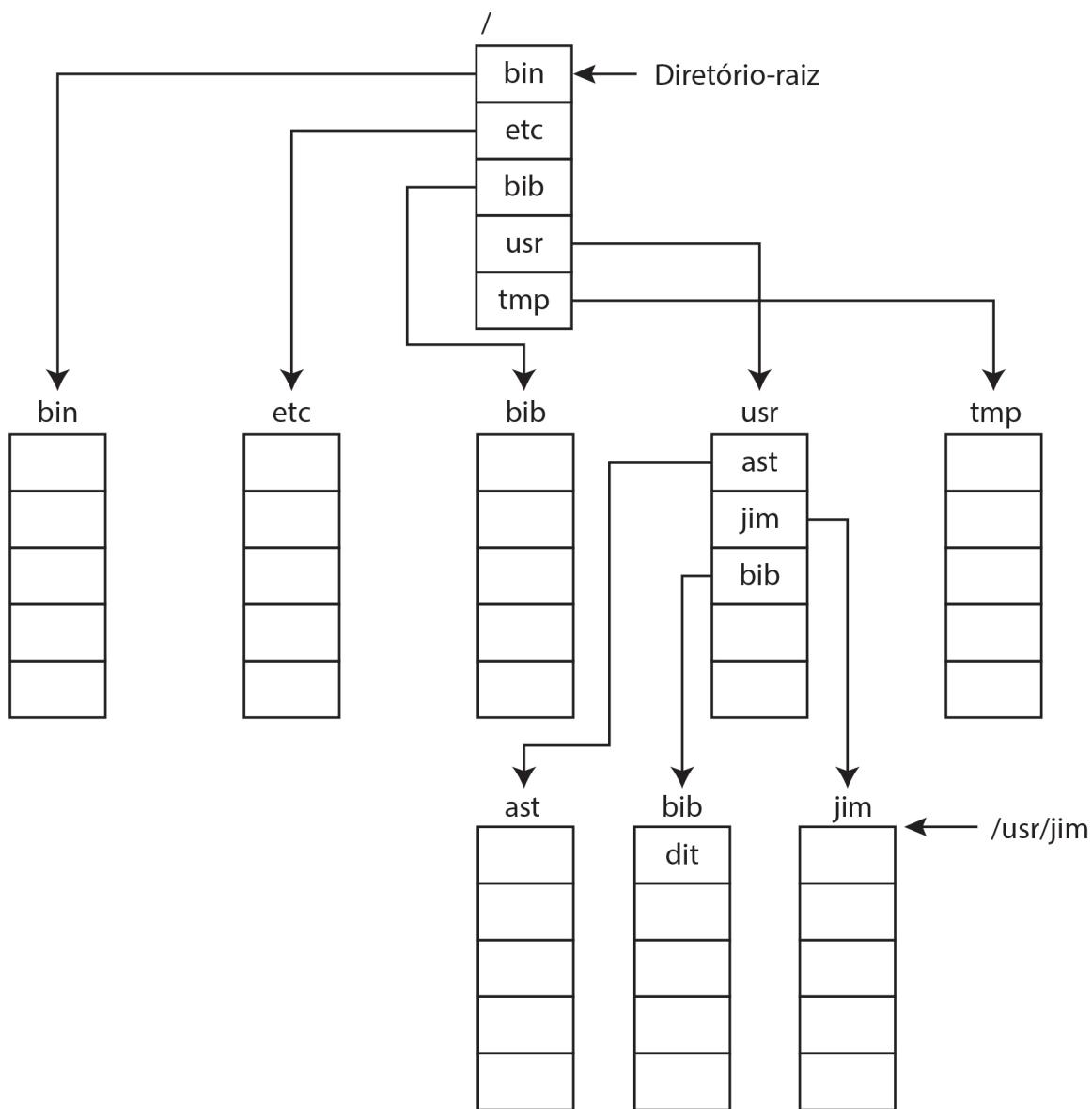


Figura 4. Árvore de diretório do Unix (TANENBAUM, 2009)

Por exemplo, ele pode copiar o arquivo /usr/bib/dicionário para seu próprio diretório usando o comando:

```
cp .../bib/dicionário.
```

O primeiro caminho instrui o sistema a subir e depois descer até o diretório bib, e encontrar o arquivo dicionário. O segundo argumento, o ponto, refere-se ao diretório atual. Quando o comando cp tem em seu último argumento um nome de diretório (o ponto), ele copia os arquivos para lá. Claro que você poderia escrever o comando de forma diferente:

```
cp /usr/bib/dicionário .
```

Neste caso o uso do ponto evita que o usuário digite dicionário duas vezes. De qualquer maneira funcionará.

```
cp /usr/bib/dicionário dicionário
```

ou o comando

```
cp /usr/bib/dicionário /usr/ast/dicionário
```

também funcionará da mesma forma (TANENBAUM, 2009).

Existe o link ou ligação que é um arquivo que aponta para outro arquivo, localizados em diretório diferente. Usuários normalmente empregam ligações para simplificar a navegação do sistema. A ligação flexível é conhecida no Unix como ligação simbólica e no Windows como atalhos (shortcut), e como apelido (alias) no MacOS. É uma entrada de diretório que contém o nome de caminho para outro arquivo. A ligação estrita é uma entrada de diretório que especifica a localização do número do bloco no dispositivo de armazenamento. O sistema de arquivo localiza os dados acessando diretamente o bloco físico(DEITEL, 2005).

Lembre-se que a reorganização e desfragmentação podem ser usadas para melhorar o desempenho do disco. Durante essas operações, a localização física de um arquivo pode mudar, exigindo que o sistema atualize a localização de um arquivo e sua entrada de diretório. Mas ao criar uma localização estrita, ela se referirá a um arquivo inválido que teve sua localização modificada. Uma maneira fácil de resolver é utilizar um ponteiro para indicar o endereço da localização estrita. Assim, quando o endereço ou dados forem movidos, o sistema irá atualizar o ponteiro, que indicará o novo endereço (DEITEL, 2005).

A atualização ou mudança de ligações flexível não é implementada por muitos SO, deixando a atualização ou remoção das ligações flexível por conta dos usuários.

5. Metadados



A maioria dos sistemas de arquivos, além de armazenar os dados de usuários e os diretórios, tem que armazenar a localização dos blocos livres e o horário que um arquivo foi modificado. Essas informações denominadas metadados protegem a integridade dos arquivos e não podem ser modificadas diretamente pelos usuários. Quando um dispositivo é formatado, o sistema de arquivos cria uma lista de blocos livres, localização do diretório-raiz, data e hora que o sistema foi modificado, informações sobre falhas, e cria uma identificação inequívoca deste arquivo chamado de superbloco. Se o superbloco for corrompido ou destruído, o SO poderá tornar-se incapaz de acessar dados de arquivos. Para impedir erros e perdas, o sistema de arquivos distribui cópias redundantes do superbloco para garantir que o mesmo não irá se perder ou ser danificado(DEITEL, 2005).

6. Montagem



Os SOs vêm com o sistema de arquivos nativos montados, mas às vezes os usuários necessitam acrescentar outros sistemas de arquivos, como um segundo HD. Por essa razão, os sistemas de arquivos permitem montar vários sistemas de arquivos combinando o atual em um único espaço de nomes. O espaço unificado permite que os usuários acessem os dados de maneira integrada ao sistema nativo(DEITEL, 2005).

Os primeiros sistemas de montagem do Windows apresentavam uma estrutura achatada, ou seja, sempre iniciavam da raiz, sendo C: o que continha o SO, e o segundo disco de dados D:, e assim por diante.

Os sistemas de arquivos compatíveis com o Unix, e a partir da versão NTFS da Microsoft 5.0, apresentam pontos de montagem que podem ser localizados em qualquer lugar do sistema de arquivos. No Unix alguns sistemas de arquivos são montados no diretório /mnt/.

Os sistemas de arquivos normalmente gerenciam os diretórios montados com tabelas de montagem. As tabelas de montagem contêm informações sobre nomes de caminhos do ponto de montagem e sobre o dispositivo que armazena cada arquivo montado. A maioria dos SO suportam vários sistemas de armazenamento removível, como por exemplo o Universal Disk Format(UDF) para DVD, e o ISSO 9.960 para CD. O comando desmontar (umount) permite desmontar ou desconectar o sistema montado (DEITEL, 2005).

Material Complementar

Com as Máquinas Virtuais (VM) que você instalou na Unidade I, inicie no SO Linux o gerenciador de arquivos. O SO Linux apresenta um dos melhores gerenciadores de arquivos. Pesquise os comandos, teste e experimente gerenciar seus arquivos no Linux. Pesquise como criar uma pasta, apagar e mover conteúdo, copiar um arquivo. Monte uma unidade removível. Tudo deve ser feito na tela de comandos.



Gerenciamento de Arquivos no Linux

Artur de Paula Coutinho, Curso de Linux Básico, [http://www.oocities.org/br/
arturpc/apostila3/curso_linux.html](http://www.oocities.org/br/arturpc/apostila3/curso_linux.html)

Referências

DEITEL, H.M. **Sistemas Operacionais**. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

TANENBAUM, A.S. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009.

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca

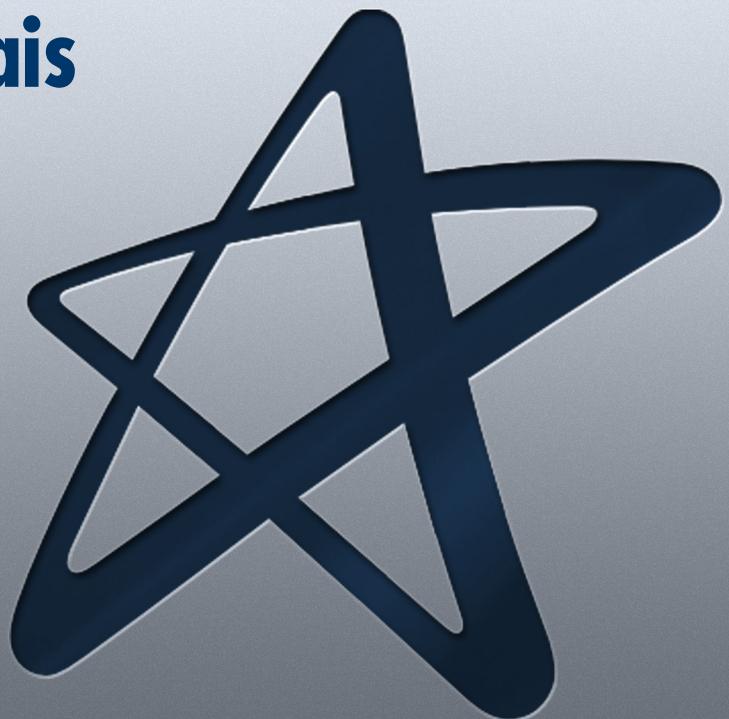


UDF
Centro
Universitário



Módulo
Centro
Universitário

Sistemas operacionais



Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

Material Teórico



Gerenciador de Entrada e Saída

Responsável pelo Conteúdo:

Prof. Ms. Claudney Sanches Junior

Revisão Textual:

Profa. Ms. Magnólia Gonçalves Mangolini

UNIDADE

Gerenciador de Entrada e Saída



- Introdução
- Gerenciamento de Entrada e Saída
- Rotina de E/S
- Controladores
- DMA - Direct Memory Access (Acesso Direto à Memória)
- Driver de Dispositivo ou Device Driver



- Nesta unidade você deverá estudar os conceitos do gerenciador de entrada e saída dos SO.
- A unidade apresenta a definição do gerenciador de entrada e saída e mostra o funcionamento do SO e os controladores. Detalha os device drivers e apresenta o conceito de DMA. Neste contexto, espera-se que ao final da unidade você seja capaz de entender o gerenciamento de E/S dos SOs modernos.

Para que você consiga atingir o objetivo desta unidade, sugiro o seguinte plano de estudos:

1. Leia o material teórico relativo a esta unidade.
2. Assista a apresentação narrada, onde serão expostos os principais conceitos de Gerenciamento de Entrada e Saída.
3. Realize as atividades de sistematização (AS_V), que serão compostas de 6 questões de múltipla-escolha sobre os conceitos-chave tratados na unidade. Quando a solução das questões for liberada, verifique quais foram as suas respostas incorretas e procure novamente pelo conceito nos nossos materiais.
4. Participe ativamente da atividade de aprofundamento (AP_V), que será realizada através da participação no fórum de discussão.
5. Consulte as indicações de referências complementares desta unidade e pesquise exclusão mútua relacionada com E/S.

Contextualização



A empresa comprou uma nova impressora e gostaria de compartilhar com outros usuários. Na máquina virtual com o servidor Linux, monte um servidor de impressão com o Samba para garantir a segurança e autenticação.

1. Introdução



O gerenciamento de dispositivos de Entrada e Saída (E/S) é uma das principais funções de um SO e consiste em controlar o acesso a todos os dispositivos de E/S, tais como: teclado, vídeo, impressoras, disco, fita magnética, etc.

Nesta unidade você vai estudar como o SO organiza, controla e acessa os dispositivos de E/S. Também serão apresentadas as rotinas de E/S e você conhecerá como funciona os controladores e os device drivers presentes em um SO.

Para que possa entender os conceitos de gerenciamento de E/S no SO, esta unidade está organizada da seguinte forma:

- a seção 2 apresenta o conceito de gerenciador de E/S;
- a seção 3 detalha as rotinas de E/S;
- a seção 4 mostra Controladores;
- a seção 5 apresenta DMA - Direct Memory Access;
- a seção 6 mostra o conceito de device drivers.

Ao final do estudo e das atividades desta unidade, você deve ser capaz de:

- entender sobre o relacionamento entre hardware e software;
- conhecer os fundamentos básicos de equipamento (hardware) e sistemas operacionais.

Não deixe de utilizar os fóruns associados à unidade para apresentar e discutir qualquer dificuldade encontrada.

2. Gerenciamento de Entrada e Saída



Uma das principais funções do SO é controlar todos os dispositivos de E/S do computador. Ele deve enviar comandos para os dispositivos, tratar erros, atender as interrupções e fornecer uma interface simples e fácil de usar entre os dispositivos e o resto do sistema. De maneira geral, gerenciar E/S é uma parte significativa do código do SO.

O principal objetivo do Gerenciamento de E/S é facilitar ao usuário o acesso aos dispositivos, sem que ele tenha que se preocupar com detalhes de funcionamento do hardware dos diversos dispositivos. Essa tarefa é bastante complexa, devido ao grande número de dispositivos e as diferenças eletromecânicas entre eles. Além disso, muitos dispositivos como os discos podem ser utilizados por vários usuários simultaneamente e o SO é responsável pela integridade dos dados compartilhados.

O software de E/S pode ser estruturado em uma das camadas do SO, sendo que cada camada apresenta uma tarefa bem definida para executar. Veja, na figura 1, a estrutura de um sistema computacional em camadas.

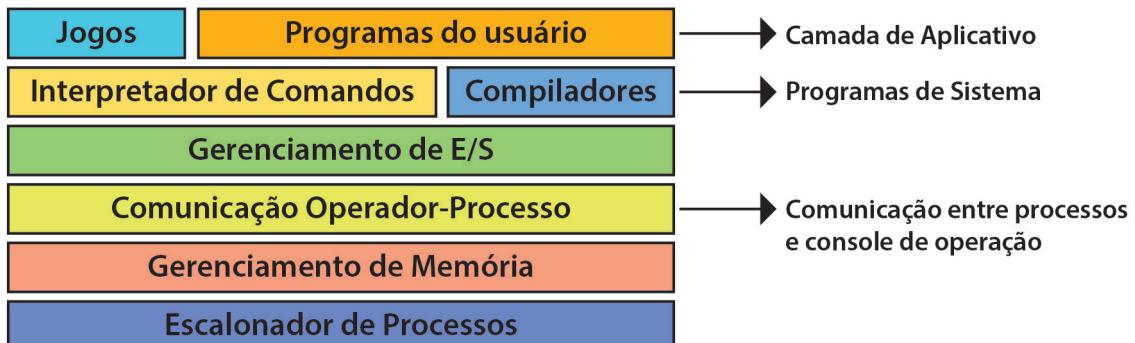


Figura 1. Apresentação das camadas de SO

Cada profissional da área computacional encara os dispositivos de E/S de uma forma diferente. Os engenheiros elétricos veem como chips, ligações elétricas, motores e componentes enquanto que os programadores veem os comandos, as interfaces, as funções e os erros que podem ser repassados ao software. No entanto a programação dos dispositivos de E/S muitas vezes está interligada com o hardware e como ele se relaciona com o programa.

De modo genérico, podem-se classificar os dispositivos de E/S como sendo: dispositivos de bloco e dispositivo de caractere. O dispositivo de bloco é aquele que armazena a informação em blocos de tamanho fixo, normalmente de 512 bytes a 32 768 bytes. A principal característica dos dispositivos de bloco é que cada bloco pode ser lido e escrito independentemente. Os discos são os dispositivos de bloco mais comum. O dispositivo de caractere envia e recebe um fluxo sem considerar quaisquer estruturas de bloco, ele não é endereçável e não dispõe de qualquer operação de posicionamento ou operação de acesso aleatório - seek operation. As impressoras, interfaces de redes, terminais e mouse são dispositivos de caractere.

Contudo este sistema de classificação não é perfeito. Os relógios ou timer que causam as interrupções não são classificados como dispositivos de bloco ou caractere. Os vídeos mapeados na memória também não se enquadram na classificação. Os sistemas de arquivos tratam os blocos como abstratos, mas, ainda assim, o modelo calcado em blocos ou caractere é amplamente utilizado para o desenvolvimento de softwares de E/S.

Os dispositivos de E/S apresentam uma ampla variação de velocidades, o que impõem a necessidade de aceitar diferentes ordens de magnitudes. Veja algumas variações na figura 2.

Dispositivos	Taxa de Dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem	7 Kb/s
Impressora Laser	100 Kb/s
Scanner	400 Kb/s
Ethernet	1,25 Mb/s

USB	1,5 Mb/s
Câmera de Vídeo Digital	4 Mb/s
Disco IDE	5 Mb/s
CD-ROM 40x	6 Mb/s
Disco ATA-2	16,7 Mb/s
FireWire	50 Mb/s
Disco SCSI Ultra-2	80 Mb/s
Ethernet Gigabit	125 Mb/s
Barramento PCI	528 Mb/s

Figura 2. Taxa de dados típicos

O SO fornece ao usuário uma interface de acesso aos dispositivos independente das características do hardware. Por exemplo, quando um usuário, ao desenvolver um programa em C, quer apresentar uma mensagem no monitor ele simplesmente utiliza o seguinte comando:

```
println("Oi");
```

Esse simples comando não é capaz de executar todos os procedimentos necessários para instruir o hardware do monitor a mostrar a mensagem. Na realidade, o compilador C traduz esse comando para uma chamada a uma rotina de E/S apropriada do SO, a qual envia os comandos para o driver do dispositivo que, por sua vez, instrui o controlador do dispositivo ou placa controladora. O controlador é quem envia os sinais ao monitor de vídeo para que este mostre a mensagem. Veja a figura 3 que apresenta o caminho que um comando escrito em um processo percorre até sua execução.

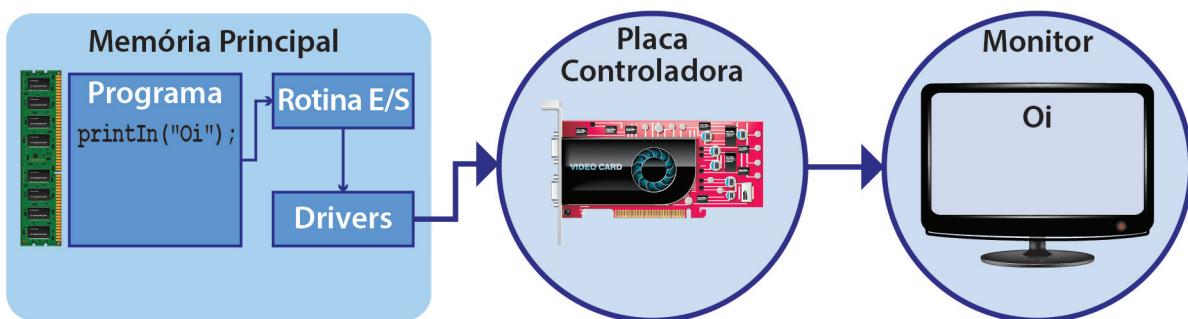


Figura 3. Relacionamento entre os componentes de software e hardware em uma operação de escrita no monitor de vídeo.

Outro exemplo seria ler um registro de um arquivo gravado em um disco. Para isso, o programador somente necessita fornecer o nome do arquivo e o registro que deseja ler. Entretanto, para que os dados sejam lidos do disco, são necessários vários comandos, por exemplo:

- dizer a localização física do registro no disco, ou seja, em qual disco, qual trilha, qual setor;
- enviar o comando para rotacionar o disco e posicionar a cabeça de leitura e gravação sobre a trilha;
- enviar o comando para ler os dados quando o setor passar pela cabeça;
- enviar os dados lidos para a localização de memória especificada;

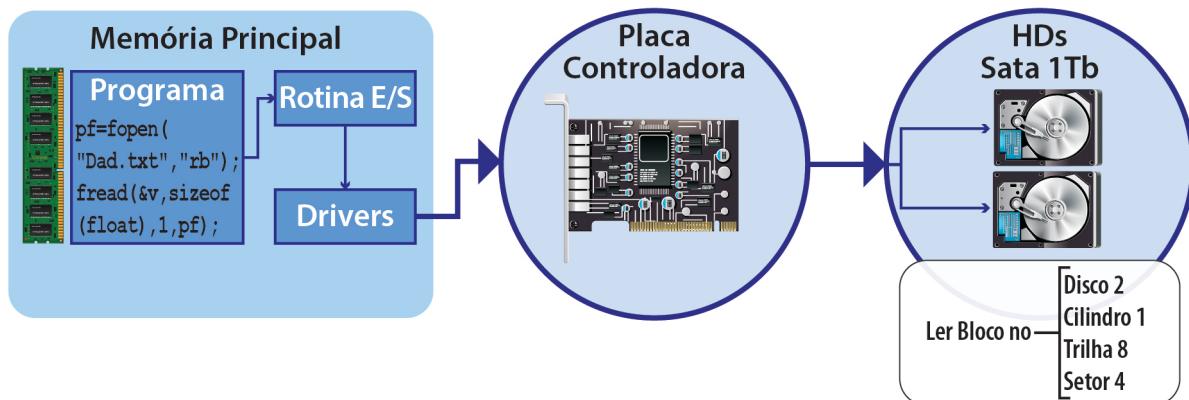


Figura 4. Relacionamento entre os componentes de software e hardware em uma operação de leitura do disco.

O sistema de gerenciamento de dispositivos é feito dividindo-se a tarefa de acesso aos dispositivos em várias camadas, cada qual, desempenhando uma função, conforme ilustrado na figura abaixo:

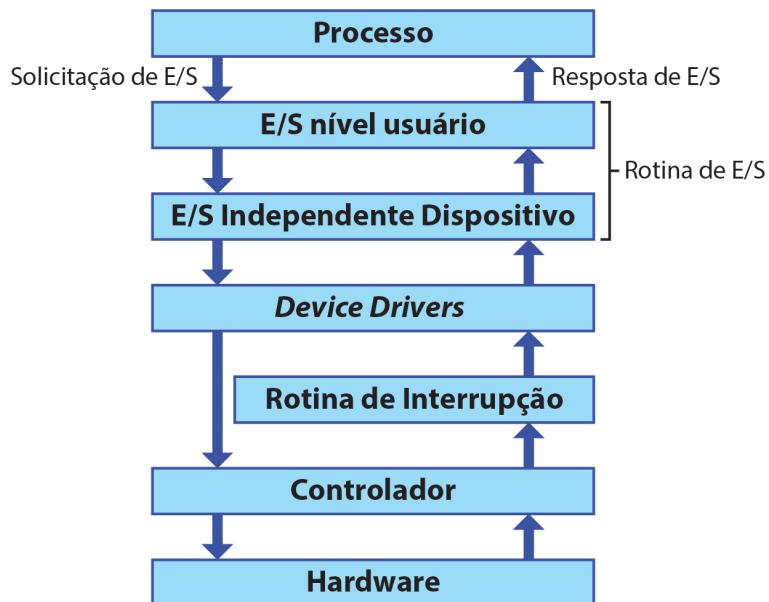


Figura 5. Componentes do sistema de gerenciamento de dispositivos de E/S.

3. Rotina de E/S



Apesar de a maior parte do software de E/S estar dentro do SO, uma pequena parte dele pode ser acessada pelo usuário desenvolvedor. Esta parte está na camada de E/S em nível de usuário, onde se encontram as bibliotecas que podem ser ligadas a programas de usuário. Por exemplo, os comandos printf, read, write, wiretel e o spooling.

Os objetivos da camada E/S independente do dispositivo são abrangentes, e pode ser citado como principais objetivos: criar uma interface uniforme para os device drivers, fornecer um mecanismo de nomeação do dispositivo, criar um tamanho de bloco independente do dispositivo, criar um espaço de bufferização para dispositivos de bloco e de caractere, cuidar da alocação de blocos livres em dispositivos de bloco, alocar e liberar dispositivos e manipular erros.

A rotina de interrupção ou manipuladores de interrupção devem ser escondidas do usuário ou transparente ao usuário. A forma de implementação mais comum dos SOs é a de quea rotina de solicitação de um processo, quando solicita uma E/S, o coloca no estado bloqueado e bloqueia o devicedirvers. Quando a interrupção de E/S ocorrer, a rotina de interrupção deve desbloquear o devicedrivers que, por sua vez, irá desbloquear o processo colocando-o na fila dos prontos.

4. Controladores



O dispositivo de E/S se conecta ao sistema através de um controlador, o qual corresponde à parte eletrônica do dispositivo responsável por enviar os comandos para o dispositivo externo (a parte eletromecânica).

As unidades de E/S são geralmente compostas de dois componentes principais: componentes eletrônicos e os componentes mecânicos. Os componentes eletrônicos são conhecidos como o controlador de dispositivos ou adaptador. Nos computadores pessoais é comum encontrá-los na forma de uma placa controladora ou placa de circuito impressa que pode ser inserida em algum conector de expansão ou slots do barramento. Por exemplo, existe a placa controladora de vídeo, a placa controladora de disco, a placa de rede, etc.

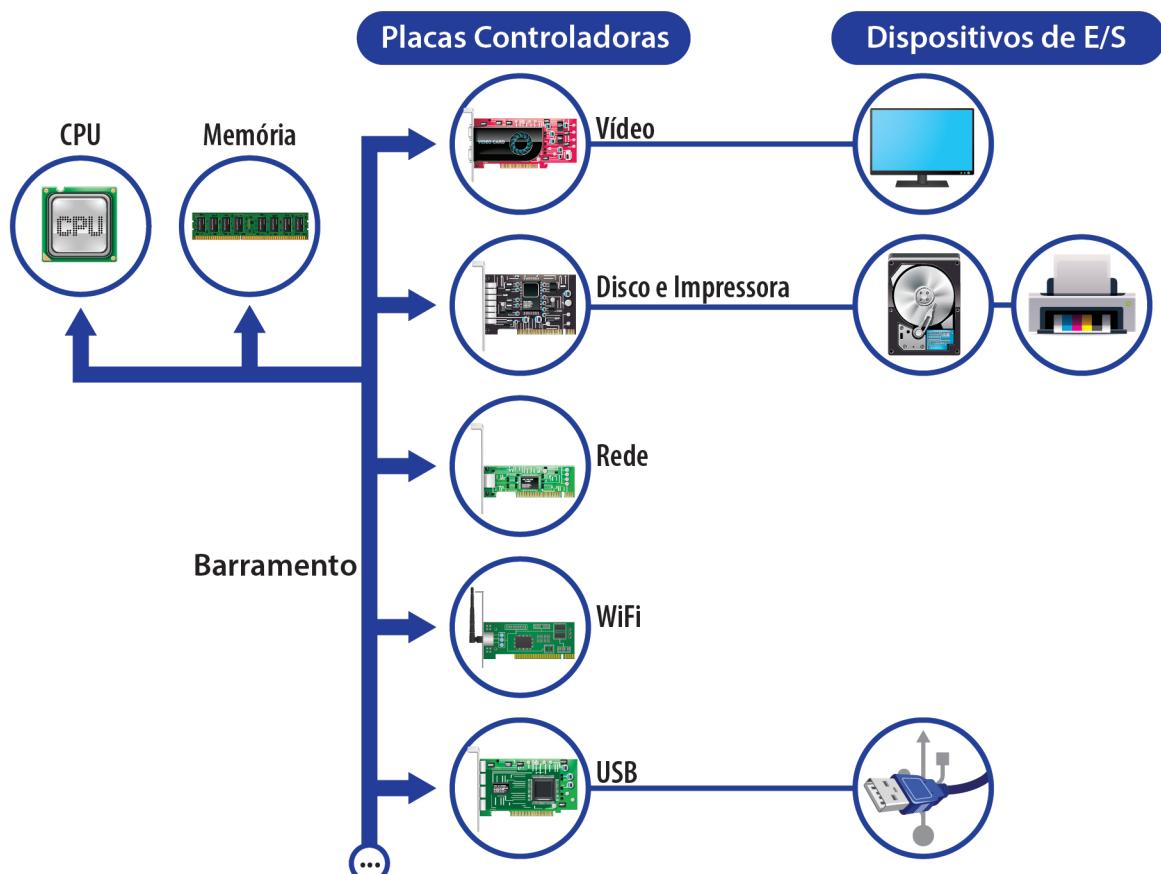


Figura 6. CPU, Memória, Barramento e Placas Controladoras

Muitos controladores foram criados para controlar mais de um dispositivo mecânico, e normalmente são padronizados (IEEE, ISO, ANSI etc.). Com o controlador, o SO não necessita baixar o nível ou enviar instruções ao dispositivo explicitando cada ação. Por exemplo, o controlador de vídeo controla o disposto monitor explicitando como o sinal será gerado, a intensidade, o feixe e o rendering. Enquanto o SO pode enviar ao controlador instruções como: desenhe algo, escreva, utilize a resolução x etc. Se não existisse o controlador, o SO deveria conhecer o dispositivo e enviar instruções detalhadas para a execução dos mesmos comandos.

Cada controlador apresenta registradores para a comunicação com a CPU. Por meio da escrita nestes registradores, o SO comanda o controlador e com a leitura de alguns registradores ele sabe o estado do dispositivo. Além disso, alguns controladores apresentam um buffer de dados, no qual o SO pode ler e escrever.

A CPU acessa os registradores e buffers do controlador, basicamente de duas maneiras. A primeira é associar o registrador de controle ao número da porta de E/S. Assim, a instrução em Assembly ficaria IN REG, PORT, ou seja, a CPU deve ler o registrador do controlador PORT e armazenar o resultado no registrador REG. Os primeiros computadores utilizavam esta solução. A segunda maneira é mapear todos os registradores no espaço de endereçamento da memória. Assim, cada registrador tem associado um endereço de memória único. Este sistema é chamado de E/S mapeada na memória. Em geral, os endereços associados estão no topo da memória principal. Um esquema híbrido, onde você encontrará as duas soluções juntas, foi elaborado, sendo o buffer associado a E/S mapeado na memória e portas de E/S associada aos registradores.

O SO não envia os comandos diretamente para o dispositivo de E/S e sim para o controlador do dispositivo. Este é quem comanda diretamente o dispositivo. Por exemplo, quando o SO identifica um comando para mostrar alguma mensagem no monitor de vídeo (writeln ou println), ele envia a mensagem para a placa controladora de vídeo, a qual enviará os sinais para comandar o monitor de vídeo para formar a mensagem.

Os controladores, em geral, possuem registradores internos que são utilizados para armazenar dados e comandos. O SO envia comandos para os controladores carregando-os nos registradores do controlador. Os parâmetros do comando também são armazenados em outros registradores.

Após o comando ter sido aceito pelo controlador, a CPU fica livre para realizar outra tarefa, enquanto o controlador executa o comando de E/S.

Por exemplo, quando um programa quer ler um bloco de dados do disco:

- o SO, através da CPU, envia os comandos de quais dados ler, onde colocar para o controlador, cabendo ao controlador comandar a leitura dos dados;
- enquanto a leitura está sendo feita, a CPU pode ser direcionada para executar outra tarefa;
- completada a leitura, o controlador gera uma interrupção para permitir que o SO ganhe o controle da CPU, verifique os resultados da operação e passe o controle para o programa que solicitou a leitura dos dados.

5. DMA - Direct Memory Access (Acesso Direto à Memória)



Os dados que entram no sistema através de um dispositivo de entrada devem ser armazenados em uma área de memória, quer seja uma variável, ou um buffer de dados, para que possam ser utilizados por um programa.

Nos primeiros sistemas de computadores, essa entrada de dados era feita em três etapas:

1. o controlador lê os dados do dispositivo e armazena-os em um buffer dentro do próprio controlador;
2. o controlador gera uma interrupção para avisar o SO de que os dados já estão disponíveis;
3. o SO é acionado, lê os dados do buffer do controlador e coloca-os em um buffer na memória principal.

Desse modo, na etapa 3 o SO utiliza a CPU para fazer a transferência dos dados do controlador para a memória, sendo que a CPU poderia estar executando outro processo se o próprio controlador se encarregasse de colocar os dados na memória.

A maioria dos controladores de hoje já conseguem fazer DMA. Nesta técnica, o próprio controlador já transfere os dados para a memória principal, liberando o SO desse trabalho e, consequentemente, liberando a CPU para realizar outro trabalho enquanto a transferência é efetuada. Por exemplo, imagine que um programa quer ler dados do disco:

- o SO envia um comando para o controlador de disco informando-o quais dados devem ser lidos e onde devem ser armazenados na memória principal;
- o controlador lê os dados do disco, colocando-os em seu buffer interno;
- após ter lido os dados do disco, o controlador transfere os dados para a localização da memória principal indicada pelo SO;
- após a transferência, o controlador gera uma interrupção para avisar ao SO que os dados já se encontram na memória.

6. Driver de Dispositivo ou Device Driver



Um devicedriver é a parte do SO que é dependente do hardware. Ou seja, seu código é específico para manipular um dispositivo de E/S. A função de um devicedriver é receber os comandos da Rotina de E/S, reconhecê-los e enviar os comandos correspondentes para o controlador para que este possa comandar o dispositivo de E/S. O devicedrivers, que também podem ser descritos como direcionadores de dispositivos ou ser conhecido popularmente apenas por driver, tem como objetivo geral aceitar os pedidos abstratos de serviços independentemente do dispositivo e ver como este pode ser executado. Normalmente, um sistema possui diversos drivers, tais como drivers para discos - disk drivers, fitas magnéticas - tape drivers, terminais - terminal drivers, impressoras, etc.

Para entender melhor o funcionamento de um driver, considere o que acontece quando um processador de textos, por exemplo o Word, manda um documento ser impresso em negrito:

- cada impressora tem seu conjunto de comandos específicos, que são diferentes entre impressoras. Um desses comandos será o de impressão em negrito;
- mesmo que o processador de texto conheça o conjunto de comandos de várias impressoras, quando uma nova impressora é lançada, o processador não saberá manipulá-la;
- para resolver esse problema, o processador de texto não envia os comandos diretamente para a impressora. Ele envia um comando abstrato (por exemplo, o comando negrito) para o driver da impressora;
- o driver da impressora converte este comando abstrato para o comando correspondente da impressora. Esse comando é passado para o controlador da impressora e posteriormente para a impressora.

Como visto, existe um conjunto de comandos abstratos que o driver utiliza, independentes de dispositivo que o programa utiliza e um conjunto de comandos correspondentes, comandos dependentes de dispositivo.

No caso de driver de impressoras, quando um fabricante lança uma nova impressora no mercado, ele também disponibiliza o driver para utilização da mesma. Portanto, a interface entre os comandos abstratos e o driver deve ser bem definida. Ou seja, o driver deve conhecer todos os comandos abstratos para saber identificá-los e enviar os comandos correspondentes para o controlador do dispositivo.

O drivers atende as requisições da seguinte forma: se estiver ocioso no momento de uma requisição ou pedido, ele o atende de imediato, mas se estiver ocupado, ele cria uma fila de pedidos pendentes. Para atender ao pedido de E/S, o drivers deve traduzir os termos abstratos para uma forma mais concreta, ou seja, deve definir quais operações devem ser programadas no controlador. Deve ainda escrever os comandos dos registradores do controlador, se bloqueando até ser acordado por uma interrupção. Após a operação ser concluída, o device drivers deve verificar a presença de erros e despertar o processo solicitador passando os dados solicitados bem como o código do status da operação. Por fim, irá verificar na fila de pedidos se existe algum pendente, e caso não encontre nenhum irá se bloquear e ficar assim até um novo pedido.

Material Complementar

Com as Máquinas Virtuais (VM) que você instalou na Unidade I, inicie no SO Linux. O SO Linux apresenta um dos melhores gerenciador de E/S, assim, monte um gerenciador de impressão e de arquivos. Instale e configure o Samba para autenticar máquinas Linux e Windows, mantendo o Linux como servidor central. Monte outras máquinas virtuais - a clientes que deverão autenticar no servidor e, dependendo do privilégio do usuário, libere acesso à Impressão e a algumas pastas.

- **Gerenciamento de Arquivos no Linux**

Artur de Paula Coutinho, Curso de Linux Básico, http://www.oocities.org/br/arturpc/curso_linux.pdf

- **Como instalar um servidor de impressão**

<http://www.tp-link.com.br/article/?id=352>

- **Como instalar o Samba**

http://www.oficinadanet.com.br/artigo/450/configurando_o_samba_no_ubuntu

<http://www.hardware.com.br/artigos/servidor-rede-local/>

Referências

DEITEL, H.M. **Sistemas Operacionais**. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

TANENBAUM, A.S. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson Prentice Hall, 2009.

Anotações





Educação a Distância
Cruzeiro do Sul Educacional
Campus Virtual

www.cruzeirodosulvirtual.com.br
Campus Liberdade
Rua Galvão Bueno, 868
CEP 01506-000
São Paulo SP Brasil
Tel: (55 11) 3385-3000



Universidade
Cruzeiro do Sul



UNICID
Universidade
Cidade de S. Paulo



UNIFRAN
Universidade
de Franca



UDF
Centro
Universitário



Módulo
Centro
Universitário