UniFil
EDUCAÇÃO A DISTÂNCIA

Unidade 1

Introdução aos Sistemas Operacionais

Introdução

Atualmente temos uma enorme quantidade de equipamentos que utilizam tecnologia digital, desde equipamentos de um avião até a um pequeno celular. Todos esses dispositivos precisam ter seus recursos gerenciados, essa é a função do que chamamos de sistema operacional.

De uma maneira simples, podemos dizer que um sistema operacional é um *software* especial que controla vários dispositivos digitais. Com certeza, mesmo que de maneira superficial, você já ouviu ou leu sobre esse sistema. Os mais conhecidos são o *Linux* e *Windows*.

Nessa unidade, iremos aprender alguns conceitos primordiais de sistemas operacionais e apresentar algumas definições de processos e as técnicas para se garantir confiabilidade da informação.

Para quem está começando a estudar a área tecnológica, esse assunto é de essencial importância, pois quem trabalha com tecnologia precisará aprender alguma plataforma tecnológica que irá conter um sistema operacional.

Bons Estudos!

Objetivos

- Aprender o que é um Sistema Operacional;
- Entender o que é Processos e *Threads*.

Conteúdo programático

Aula 01 – O que é um Sistema Operacional.
Aula 02 – Processos e *Threads*.

Você poderá também assistir às videoaulas em seu celular! Basta apontar a câmera para os **QRCodes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular

Aula 01 – O que é um Sistema Operacional

Videoaula Apresentação

Utilize o QRcode para assistir!

Videoaula Apresentação

Videoaula Mini Currículo

Utilize o QRcode para assistir!

Descrição do Professor...

Olá, querido aluno! Como vai? Vamos começar agora a mais uma jornada nesse mundo da computação. Vamos estudar o mundo dos sistemas operacionais. É tão comum usar um sistema operacional atualmente que às vezes nos passa despercebido a sua presença. Os sistemas operacionais estão presentes nas Tv's, celulares, computadores pessoais, servidores da internet, sistemas de aviões e dezenas de outros dispositivos existentes.

Qualquer **sistema computacional** atual é muito complexo, pois consiste em processadores, memórias, monitores, interfaces de rede, discos e outros dispositivos de entrada e saída. Se cada desenvolvedor de aplicação fosse se preocupar com o funcionamento de cada dispositivo pertencente a um computador por exemplo, certamente seria algo extremamente difícil e caro.

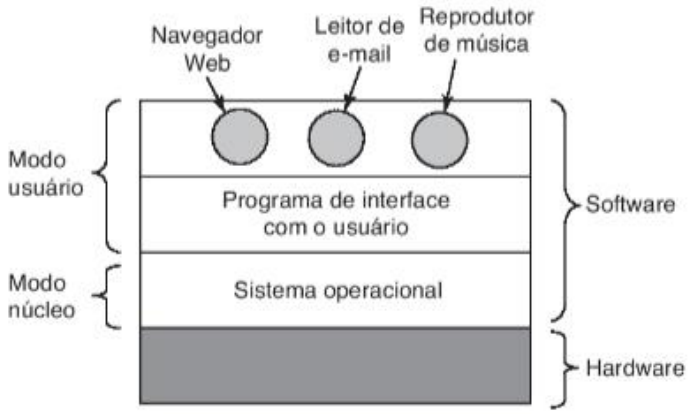
O *hardware* nos dias atuais executa dezenas de aplicações ao mesmo tempo, se essas aplicações não forem bem controladas problemas graves acontecem. Visando controlar toda essa complexidade de *hardware* e aplicações, foi criado um dispositivo de *software* denominado de **sistema operacional.**

Para Tanenbaum (2015, p. 1), um sistema operacional é um *software* que gerencia todos os recursos de *hardware* de um computador e fornece base para todos outros *softwares*, tornando assim, o uso da máquina mais fácil para os usuários e desenvolvedores de aplicações. Podemos destacar então as principais funções de um sistema operacional:

- Gerenciar o uso do *hardware* de um sistema computacional, garantindo o seu emprego específico e o armazenamento correto dos dados;
- Fornecer uma camada de abstração para a utilização e desenvolvimento de outros *softwares*;
- Prover uma interface de acesso a dispositivos com as mais distintas tecnologias tais como: USB, IDE.

Um *hardware* resume-se em alguns componentes que podem ser vistos na parte inferior da figura 1, abaixo. Consiste em *chips*, placas, teclados, *Hard Disc,* etc. Na parte superior notamos o *software*, que na maioria dos computadores trabalha em dois níveis de operação: **modo núcleo e modo usuário**. No caso, o sistema operacional é a peça mais básica de *software* e opera em modo núcleo.

Figura 1 - Posição do sistema operacional

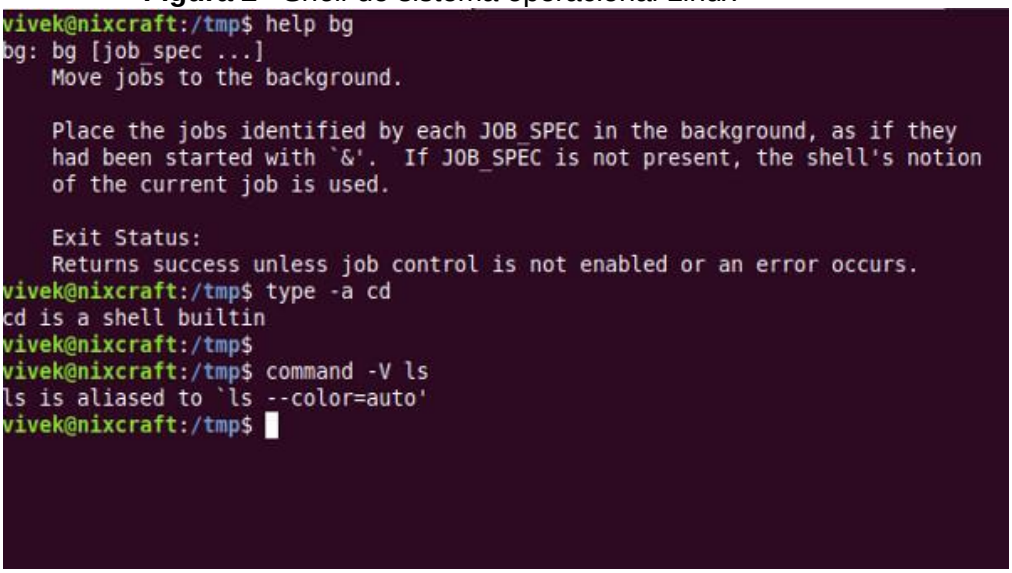


Fonte: Tanenbaum (2015, p. 1)

No modo núcleo, o sistema operacional tem acesso completo ao *hardware* e pode executar qualquer instrução que a máquina seja capaz de executar (TANENBAUM, 2015). No modo usuário apenas um subconjunto de instruções da máquina está disponível. Nesse modo, as instruções que mexem com o controle de máquina ou trabalham com entradas e saídas são proibidas no modo usuário.

A interface mais comum entre um usuário e o sistema operacional é o *shell* (interpretador de comandos) e a GUI (*Graphical user Interface* – Interface gráfica com o usuário), essas duas maneiras é o nível mais inferior de *software* de modo usuário e permite que sejam inicializados outros programas, como Navegadores *Web*, reprodutores de filmes ou música. Na figura 2 abaixo, é apresentado um exemplo de *shell* do sistema operacional *Linux*.

Figura 2 - *Shell* do sistema operacional Linux



Fonte: disponível em: https://imgs.developpaper.com/imgs/2018225161714117.jpg. Acesso em: 8 maio 2021.

Uma observação importante que devemos fazer é diferenciar um sistema operacional (SO) de um programa de usuário. Os sistemas operacionais são *softwares* que tem uma vida muita longa, são grandes e complexos. A título de exemplo, o código fonte do sistema operacional *Linux* ou do *Windows* tem aproximadamente mais de cinco milhões de linhas de código, imagina escrever tudo isso novamente? Em vez disso, os SO's são basicamente atualizados.

Indicação de Vídeo

Agora assista o vídeo da Bóson treinamentos em que é explicado alguns conceitos sobre os sistemas operacionais. Vídeo bem objetivo e interessante.

Disponível em: https://www.youtube.com/watch?v=T7lCM3l7vAQ. Acesso em: 8 maio 2021.

Até agora, estudamos o conceito mais formal sobre o que é um sistema operacional. Nesse momento, é importante sabermos que existem diversas categorias de sistemas operacionais e que são empregados em diferentes contextos.

Categorias de Sistemas Operacionais

Vale ressaltar que existem várias categorias de sistemas operacionais. Segundo Tanenbaum (2015), os sistemas operacionais podem ser classificados em nove categorias, vamos então verificar quais são elas:

Sistemas operacionais de grande porte: estes computadores distinguem dos computadores pessoais em termos de capacidade de entrada e saídas (E/S). Possui milhares de discos com milhares de *terabytes* de dados. Podem ser encontrados atualmente em servidores da *Web* e servidores para transações de negócios. Geralmente são orientados a trabalhar com um fluxo de informações muito grande. Esses sistemas operacionais trabalham com três tipos de serviços: em lote (*batch*), processamento de transações e tempo compartilhado.

Um sistema em lote processa rotinas sem a presença interativa dos usuários. Sistemas de processamento de transações gerenciam grandes quantidades de requisições, por exemplo, processamento de verificações de um banco. Já os sistemas de tempo compartilhado permitem que milhares de usuários remotos executem suas tarefas simultaneamente no computador, como na realização de consultas a um grande banco de dados.

Sistemas operacionais de servidores: servidores são computadores pessoais maiores e mais potentes. Eles atendem a múltiplos usuários de uma vez em uma rede permitindo-lhes recursos computacionais (*hardware* e *software*). Podemos ter servidores de impressão, servidores de arquivos ou de *Web*. Os principais sistemas operacionais que trabalham nesse tipo de computador são o *Linux* e o *Windows Server*. Alguns operam em modo *shell* e outros interface gráfica, que é o caso do *Windows Server*.

Sistemas operacionais de multiprocessadores: atualmente para obter uma potência melhor dos computadores utiliza-se conectar múltiplas CPU´s em um único sistema. Tais sistemas precisam de sistemas operacionais especiais, porém muitos deles são variações do sistema operacional de servidores, com alguns aspectos especiais de comunicação, conectividade e compatibilidade.

Sistemas operacionais de computadores pessoais: essa categoria de sistemas operacionais é amplamente conhecida. Faz uso intenso de multiprogramação para não permitir a ociosidade do processador e faz uso de *GUI (*Interface Gráfica). Ficaram populares a partir do IBM-PC (utilizando *Microsoft Windows*) e do *Apple Macintosh*

(utilizando *Apple OS*) e também com o *Linux* nas suas diversas distribuições. São utilizados nos *notebooks* e *desktop*.

Sistemas operacionais de computadores portáteis: atualmente o uso de sistemas portáteis se tornou muito comum em nossa sociedade. A nanotecnologia se desenvolveu muito rapidamente e com isso os sistemas computacionais foram ficando cada vez menores.

O precursor de um sistema portátil foi o PDA (*Personal Digital assistant)* que era um pequeno computador com número limitado de funções. Hoje, esse sistema foi substituído pelos telefones celulares atuais.

Os celulares são sistemas avançados que contêm dezenas de funções em um pequeno aparelho. Esses aparelhos contêm sistemas operacionais sofisticados com a capacidade de lidar com a telefonia, som, fotografia e outras funções. Os mais utilizados são *IO´s* e *Android*.

Sistemas operacionais embarcados: Se você já notou, a sua *smart* TV contém alguns aplicativos para serem executados ou também já reparou que os carros atuais vêm controlados por alguns *softwares*. Boa parte dessas aplicações são controladas por um sistema operacional.

Os sistemas operacionais do tipo embarcado são utilizados geralmente em sistemas que não são computadores e tem um uso muito específico, tais como: *Tv´s*, carros, relógios, robôs, geladeiras, forno micro-ondas. Tanto o sistema operacional e os *softwares,* que são executados nesses sistemas, ficam armazenados em uma memória ROM.

Curiosidades

Conheça um sistema operacional voltado para atuar em sistemas que contenham processos altamente críticos. Siga o *link*, e conheça o sistema operacional *QNX*.

Disponível em: https://blackberry.qnx.com/en/software-solutions/embedded-software/qnx-neutrino-rtos. Acesso em: 8 maio 2021.

Na web:

Se você deseja manusear o sistema operacional QNX, assista o vídeo no qual mostra a instalação utilizando-se de uma máquina virtual.

Disponível em: https://www.youtube.com/watch?v=3yjVaIwzPsc. Acesso em: 8 maio 2021.

Sistemas operacionais de nós sensores: com certeza você já ouviu falar em internet das coisas (IoT). Isso é uma maneira em que os objetos físicos estão conectados e se comunicando entre si e com o usuário, através de sensores inteligentes e *softwares* que transmitem dados para uma rede.

Esses sensores podem ser considerados nós que podem ser um minúsculo computador, pois são capazes de conter uma CPU (Unidade Central de processamento), RAM (memória volátil), ROM (Memória apenas de leitura) mesmo que pequenas. Tudo isso, deve ser gerenciado por um sistema operacional pequeno, pois os nós têm uma memória RAM pequena e a duração da bateria é uma questão importante. Um sistema operacional conhecido nesse ramo é o ***TinyOS*.**

Sistemas operacionais de tempo real: sistemas de tempo real tem como parâmetro principal o tempo. Os processos aqui têm que ser cumpridos em determinados intervalos de tempo, tem-se então, nesse caso, um sistema de tempo real crítico. Muitos deles encontrados em aviônica, sistemas industriais, sistemas médicos etc.

Por outro lado, temos os sistemas operacionais não críticos, no qual o descumprimento ocasional de algum prazo, embora não seja desejável, é aceitável e não causa nenhum dano permanente. Sistemas de telefones digitais são sistemas operacionais não críticos.

Lendo o texto, você pode ter notado que sistemas portáteis, embarcados e de tempo real têm coisas comuns. Quase todos eles têm pelo menos alguns aspectos de tempo real. Uma diferenciação importante que pode ser feita é que sistemas portáteis e sistemas embarcados são planejados para consumidores, enquanto que sistemas de tempo real são projetados ao uso industrial.

Sistemas operacionais de cartões inteligentes (*smart cards)*: quando você vai ao mercado e passa seu cartão de crédito na máquina, como você acha, querido aluno, que acontece a transação bancária? Os menores sistemas operacionais são executados em nesses cartões inteligentes – esses cartões contêm um chip de CPU. Possuem grande restrição de consumo de energia. Alguns obtêm a energia pelo contato com o leitor em que estão inseridos.

Videoaula 1

Agora assista ao vídeo no qual abordaremos os primeiros conceitos importantes da aula.

Videoaula 1

Utilize o QRcode para assistir!

Agora assista ao vídeo no qual abordaremos os primeiros conceitos importantes da aula.

Arquiteturas de sistemas operacionais

Como comentamos anteriormente, os sistemas operacionais tendem a ser complexos, uma vez que gerenciam muitos serviços e suportam uma variedade de recursos de *hardware* e *software*. Uma arquitetura determinada pode ajudar os projetistas a gerenciar essa complexidade, organizando os componentes dos sistemas e especificando o privilégio com que cada componente é executado (DEITEL *et al.*, 2005).

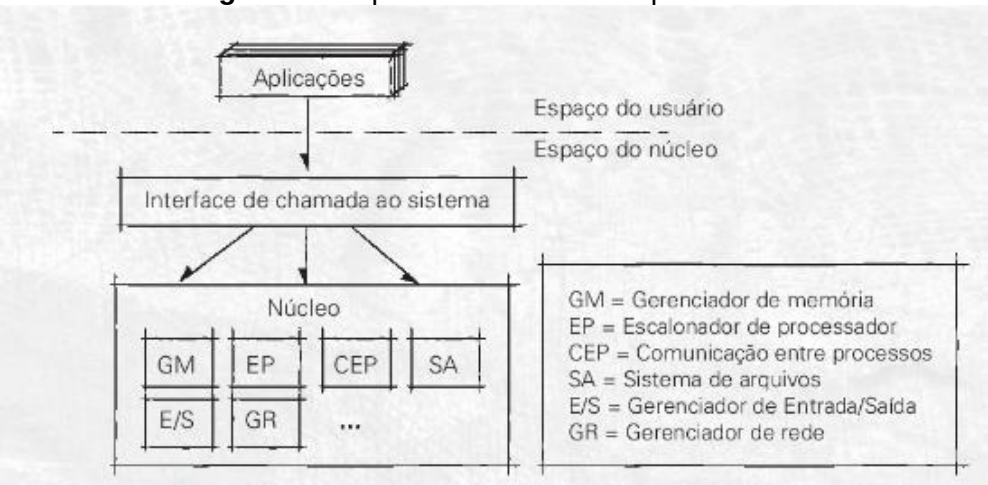
Da mesma forma que os arquitetos usam abordagens diferentes para projetar edifícios, projetistas de sistema operacionais também empregam diferentes abordagens arquitetônicas para o projeto de sistemas operacionais. Podemos dividir essas abordagens da seguinte forma segundo (DEITEL *et al.*, 2005):

- Arquitetura monolítica;
- Arquitetura em camadas;
- Arquitetura de micronúcleo;
- Sistemas operacionais de rede e distribuídos.

Um sistema operacional monolítico é a arquitetura mais antiga e mais comum. Nesta abordagem, o sistema por completo é executado como um único programa no modo núcleo. Cada componente do sistema está contido no núcleo e pode comunicar-se diretamente com qualquer outro, utilizando-se de chamadas à função. O núcleo também é executado com acesso irrestrito ao sistema de computador.

O fato de a intercomunicação entre os componentes serem diretas o torna um sistema altamente eficiente. Porém, o seu acesso irrestrito ao sistema do computador pode lhe deixar suscetível a danos provocados por códigos sujeitos a erros ou mal-intencionados. A figura 3 abaixo, mostra a arquitetura de um sistema monolítico.

Figura 3 - Arquitetura de sistema operacional monolítico



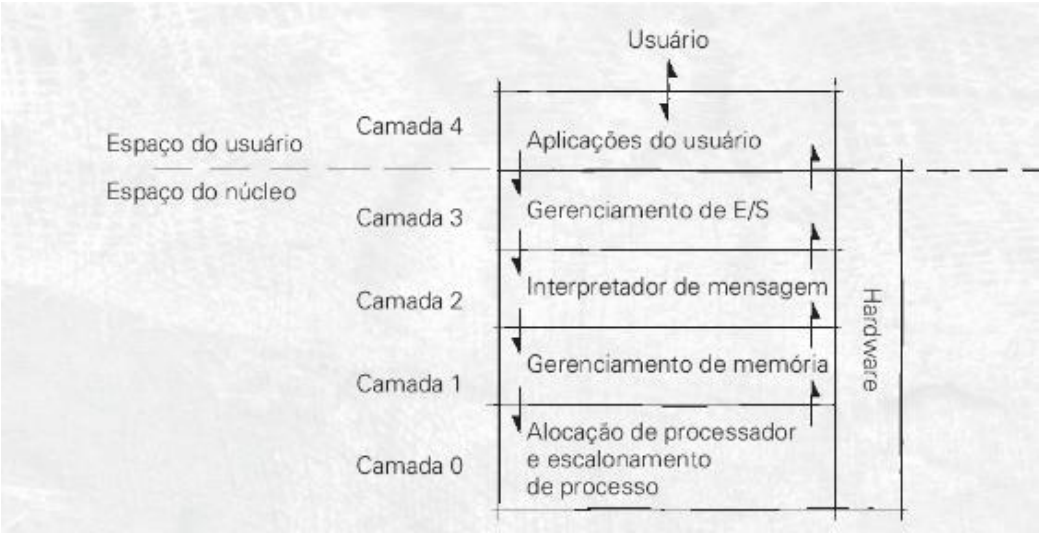
Fonte: Deitel *et al.* (2005, p. 20)

O sistema operacional **de arquitetura em camadas** divide o sistema operacional em sistemas sobrepostos, ou seja, uma camada acima da outra. Cada camada comunica-se exclusivamente com as camadas imediatamente acima ou abaixo dela.

Esse tipo de sistema torna-se mais modular do que os monolíticos, dado que a implementação de cada camada pode ser alterada sem exigir nenhuma modificação na outra. Também em um sistema modular, existem componentes que podem ser reutilizados por todo o sistema.

O uso em camadas faz com que um pedido de uma aplicação demore mais tempo para chegar até o dispositivo periférico ou ao recurso a ser acessado, prejudicando dessa maneira o desempenho do sistema. Um exemplo de sistema operacional de arquiteturas em camadas é o sistema operacional *THE*, cuja sigla deriva do *Technische Hogeschool Eindhoven*, na Holanda, onde foi implementado. Esse sistema contém seis camadas, conforme mostra a figura 4 abaixo.

Figura 4 - Camada do sistema operacional THE



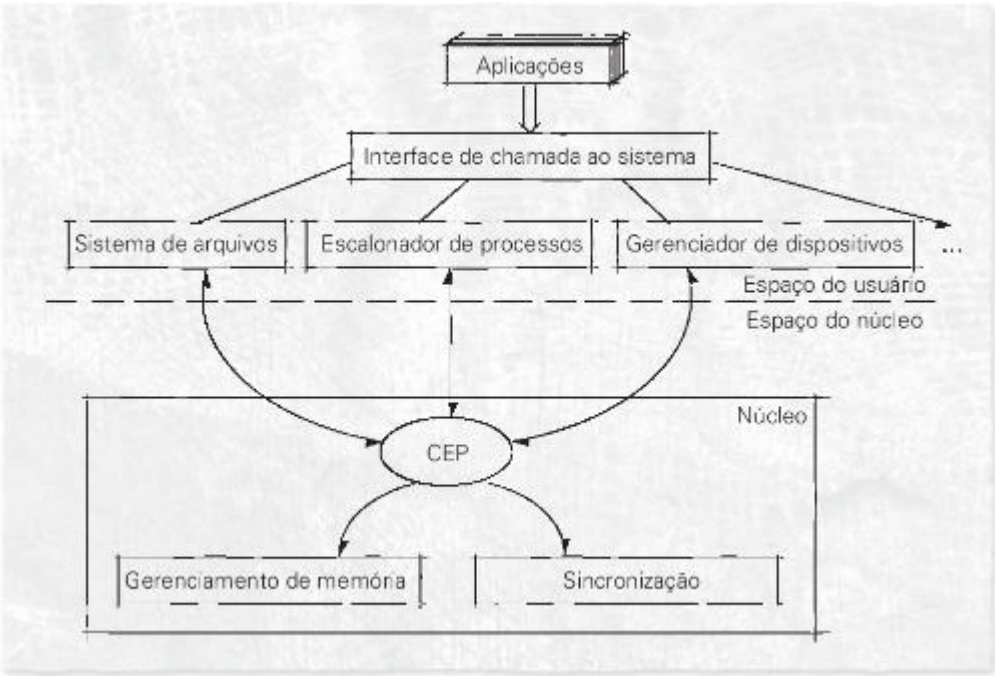
Fonte: Deitel *et al.* (2005, p. 20)

Uma tendência dos sistemas operacionais é tornar o núcleo menor e mais simples. Uma arquitetura de **sistema operacional de micronúcleo** provê um número pequeno de serviços com o objetivo de manter o núcleo pequeno e escalável. Entre esses serviços estão: gerenciamento de memória de baixo nível, comunicação entre processos e sincronização básica de processos para habilitar a cooperação entre eles.

Os gerenciamentos de processos, rede, sistemas de arquivo e gerenciamento de dispositivos são executados fora do núcleo, com um nível de privilégio mais baixo, observe a estrutura do micronúcleo na figura 5, abaixo. Micronúcleos exibem alto grau de modularidade, isso é importante, visto que se algum módulo falhar não causará problema ao sistema operacional.

Por outro lado, a modularidade necessita de maior comunicação entre os módulos afetando assim o desempenho do sistema. Poucos sistemas populares atuais adotam o projeto de micronúcleo (DEITEL *et al.*, 2005).

Figura 5 - Arquitetura de sistemas operacionais de micronúcleos

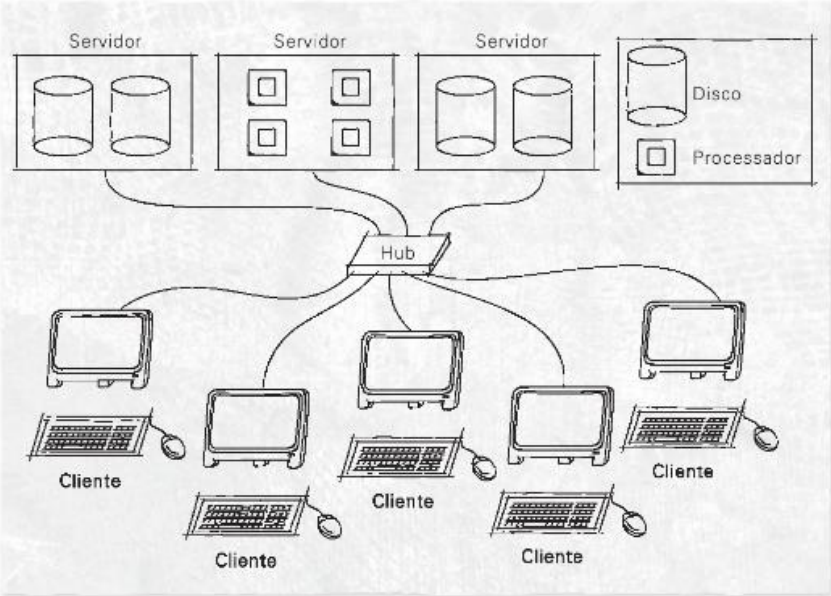


Fonte: Deitel *et al.* (2005, p. 22)

Vamos agora estudar **o sistema operacional de rede e distribuídos.** Um **sistema operacional de rede** é executado em um computador e habilita seus processos a acessar recursos como arquivos e processadores em um computador remoto. Conhecidos também como modelo cliente-servidor, o cliente requisita recursos - como arquivos e tempo de processador a um computador remoto chamado de servidor – via protocolo de rede. Os servidores então respondem seus clientes fornecendo os recursos adequados. A figura 6 abaixo, exemplifica um sistema operacional de rede.

Os **sistemas operacionais distribuídos** são sistemas que gerenciam recursos em mais de um sistema de computador. Sistemas distribuídos dão a impressão de que vários computadores compõem apenas um computador de grande capacidade. Esses sistemas são difíceis de implementar, porque necessitam de algoritmos complexos para habilitar os processos a se comunicarem e a compartilhar dados.

Figura 6 - Modelo de sistema operacional de rede cliente/servidor



Fonte: Deitel *et al.* (2005, p.23)

Vamos agora falar um pouco de algumas características dos sistemas operacionais. Apesar de existirem sistemas operacionais diferentes é interessante saber que existem recursos comuns entre eles. Preste atenção nos conceitos abaixo:

Multiprocessamento – Os SO´s vêm dotados com a capacidade de realizar dois ou mais processos simultaneamente. Para isso é necessário ter mais de um processador no computador. Não deve confundir com a multitarefa, pois ela apenas simula a execução simultânea de mais um processo, basicamente compartilhando o tempo do processador com os processos em execução.

Multiprogramáveis - É a capacidade de um sistema operacional executar vários programas na memória simultaneamente.

Time-sharing – A expressão inglesa ***time-sharing*** significa “tempo compartilhado”. É a capacidade de um sistema operacional de compartilhar o uso do processador ao longo do tempo entre os vários processos em execução. Os processos são executados, um a um,

de maneira sequencial, mas como o pedaço de tempo é muito pequeno para cada um, tem-se a impressão de que os processos estão sendo executados de maneira simultânea.

Memória virtual - É uma técnica do sistema operacional de usar a memória secundária como um *cache* para armazenamento temporário, permitindo o compartilhamento seguro e eficiente da memória principal entre vários processos e também para remover as limitações de memória. Essa técnica fornece ao usuário a ilusão de existir uma memória muito maior do que a principal.

Videoaula 2

Agora, assista o vídeo no qual trataremos sobre as arquiteturas de sistemas operacionais.

Videoaula 2

Utilize o QRcode para assistir!

Agora, assista o vídeo no qual trataremos sobre as arquiteturas de sistemas operacionais.

Leitura Obrigatória

Com certeza, existe um conteúdo muito grande e interessante sobre esse tema. Por isso, para conhecer mais leia o capítulo 1 do livro "**Sistemas Operacionais Modernos**" de Andrew Tanenbaum. Essa leitura com certeza vai complementar seus estudos.

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/40?code=MFISpf9I7aYlvFeiMFtsnq2CSvvd3BIHKG7wn4jVO98qM5hgfoykrQY0YQaRRkmKEYRSvvz9+twoAJyu8gBxxQ==. Acesso em: 8 maio 2021.

Tipos de Sistema Operacionais

Existem diferentes tipos de sistemas operacionais, para explicá-los uso aqui as ideias descritas em (SIQUEIRA, 2021). Os tipos de sistemas operacionais existentes são os seguintes:

- Sistemas Monoprogramáveis/Monotarefa;
- Sistemas multiprogramados/multitarefa;
- Sistemas com múltiplos processadores.

Sistemas monoprogramáveis

Este tipo de sistema operacional permite que apenas um programa seja armazenado na memória para execução. Os demais programas devem aguardar o término da execução do programa que está em memória.

Todos os recursos do computador permanecem exclusivamente dedicados ao único programa que está em memória. Este sistema operacional permite, portanto, a execução de uma única tarefa de cada vez, alocando de forma exclusiva todos os recursos do computador, sendo chamado por isso de sistema monotarefa.

Sistemas multiprogramados/multitarefa

Este tipo de sistema operacional é uma evolução dos sistemas operacionais monoprogramados. Enquanto os sistemas operacionais monoprogramados permitiam apenas a execução de um único programa e uma única tarefa, os sistemas multiprogramados/multitarefas permitem que vários programas sejam executados compartilhando os recursos do computador, tais como: discos, impressora, memória e processador.

Os sistemas multiprogramados podem ser classificados de duas formas:

- Pelo número de usuários que interage com o sistema operacional;
- Pela forma como as tarefas são gerenciadas.

Com relação ao número de usuários, os sistemas multiprogramados podem ser: **monousuário ou multiusuário**. Os sistemas multiprogramados monousuário são usados em computadores pessoais e estações de trabalho em que um usuário pode realizar várias tarefas como acessar a internet, enviar um e-mail e editar um documento. Os sistemas multiprogramados multiusuários são sistemas acessados por vários usuários que interagem com o sistema operacional realizando várias tarefas.

Com relação a forma como as tarefas são gerenciadas, os sistemas multiprogramados podem ser:

1. **Sistemas *Bach*:** neste tipo de sistema, os programas chamados de *jobs*, aguardavam disponibilidade de memória principal para serem executados;
2. **Sistemas de Tempo compartilhado (*Time-Sharing*)**: permitem a execução de várias tarefas pela divisão de tempo do uso do processador em pequenos intervalos de tempo, denominados fatia de tempo (*time slice*). Os programas em execução se alternam no uso do processador, cada programa em execução fazendo uso de sua fatia de tempo. Desta forma, o tempo de uso do processador é compartilhado pelas várias tarefas em execução pelo sistema operacional. Por este motivo, estes sistemas também são chamados de sistemas *time-sharing*;
3. **Sistemas de Tempo real (*real time*):** permitem a execução de várias tarefas de acordo com a prioridade de execução de cada tarefa. A diferença entre os sistemas de tempo real e os sistemas de tempo compartilhado é que nos sistemas de tempo compartilhado o tempo de resposta pode variar um pouco sem, no entanto, comprometer a execução das tarefas, enquanto que nos sistemas de tempo real, o tempo de resposta é rigidamente controlado e devem estar dentro de limites de tempo definidos que devem ser obedecidos, caso contrário podem ocorrer problemas irreparáveis com as aplicações. Costuma-se dizer que estes sistemas têm tempo de resposta quase instantâneo, por isso, são conhecidos por sistemas de tempo real.

Sistemas com múltiplos processadores

Os sistemas com múltiplos processadores caracterizam-se por ter mais uma CPU interligadas e trabalhando com conjunto. Desta forma, é possível a execução simultânea

de mais de uma tarefa ou a divisão de uma tarefa em partes que são executadas simultaneamente em mais de um processador.

Os sistemas com múltiplos processadores incorporam todos os recursos dos sistemas multiprogramados e acrescentam novas características e vantagens, tais como: escalabilidade, maior disponibilidade e balanceamento de carga.

Videoaula 3

Agora, assista o vídeo no qual trataremos sobre os tipos de sistemas operacionais.

Videoaula 3

Utilize o QRcode para assistir!

Agora, assista o vídeo no qual trataremos sobre os tipos de sistemas operacionais.

Aula 02 – Processos e Threads

Olá alunos! Na aula anterior você estudou sobre o que é um sistema operacional (SO), seus tipos e arquitetura do SO. Isso é importante para que você comece a entender o funcionamento desse componente tão importante da computação. Iremos começar agora com mais algumas definições existentes nos sistemas operacionais que são: Processos e *Threads*.

Quando utilizamos o computador, geralmente gostamos de realizar várias tarefas ao mesmo tempo: navegar na internet, escutar música e jogar. Cada um desses itens é considerado um **processo** dentro do contexto de sistema operacional e precisam ser gerenciados. Então vamos discutir o que são processos.

Para Deitel *et al.* (2005, p. 66), processo pode ser considerado um **programa em execução** e consiste em três grandes partes:

- **Região de texto**: armazena o código que o processador executa;
- **Região de dados**: é uma região que armazena variáveis e memória alocada dinamicamente que o processo utiliza durante a execução;
- **Região de pilha**: armazena instruções e variáveis locais para chamadas ativas ao procedimento. O conteúdo da pilha aumenta à medida que um processo emite chamadas aninhadas ao procedimento, e diminui quando o procedimento chamado retorna.

Os sistemas operacionais precisam de métodos para criar e finalizar processos durante uma operação quando necessitar. Para Tanenbaum (2015, p. 52) um processo pode ser criado mediante quatro situações:

- **Início do sistema**: quando o sistema operacional é inicializado, em geral criam-se vários processos. Alguns, são processos que vão trabalhar em segundo plano, tal como o antivírus;
- **Requisição de um usuário para criar um novo processo**: quando você deseja executar um determinado programa, geralmente clicamos ou digitamos seu nome no console para que o sistema operacional o execute;
- **Criação de processo por um processo em execução**: é quando você está utilizando um programa e dentro desse programa você necessita

chamar outro programa. Por exemplo, utilizando um editor de texto e desejamos imprimir o que está escrito no editor. Nesse caso, é um processo chamando outro processo;

- **Tarefas em lote (*batch job*):** utilizado em sistemas de grande porte. Quando o sistema julgar que existe recursos para executar outras tarefas, o sistema operacional criará um novo processo e executará nele a próxima tarefa da fila de entrada.

Quando um processo é inicializado, obviamente o tempo de execução deve-se chegar ao fim. A ação de término pode ocorrer mediante quatro situações, segundo (TANENBAUM, 2015).

Término normal voluntário: acontece quando o processo termina seu objetivo com êxito, ou quando o próprio usuário solicita voluntariamente o encerramento do processo. Isso acontece quando você usa um programa na área de trabalho do seu computador e clica no **X** no canto superior direito da janela em uso. Esse tipo de sistema é usado tanto em *Linux* como em *Windows*.

Término por erro voluntário: ocorre quando o processo descobre um erro fatal, acionando assim o estado de encerramento. Por exemplo, quando o usuário tenta abrir um arquivo que não existe o sistema simplesmente termina a execução.

Término por erro fatal involuntário: Geralmente é um erro causado pelo processo, em alguns casos causados por um erro de programa. Como exemplo, podemos citar a execução de uma instrução ilegal, na qual faz referência a memória inexistente ou uma divisão por zero.

Cancelamento por outro processo involuntário: acontece quando um processo executa uma chamada de sistema dizendo ao sistema operacional para cancelar algum outro processo. Em alguns sistemas quando um processo termina de maneira voluntária ou não, todos os processos criados por ele também são imediatamente cancelados.

Pois bem, então notamos acima as formas que um processo se inicia e a forma como ele pode terminar. Vamos verificar agora que durante o tempo de vida de um processo ele passa por uma série de estados distintos. Vários eventos podem fazer que os processos mudem de estado.

Videoaula 1

Agora assista o vídeo no qual trataremos sobre os processos.

Videoaula 1

Utilize o QRcode para assistir!

Agora assista o vídeo no qual trataremos sobre os tipos de sistemas operacionais.

Basicamente um processo ativo pode encontrar-se em três estados diferentes a conhecer:

- **Execução (*Running)*** – É quando está sendo executado pela UCP (Unidade Central de Processamento). Em determinadas situações os processos monopolizam o processador de maneira maliciosa ou acidental. Para resolver isso, o SO estabelece um tempo para o processo fazer sua tarefa. Se o processo não devolver o processador de maneira voluntária antes que o tempo expire, é gerada uma interrupção, fazendo que o SO obtenha o controle do processador. Nesse caso, o processo é mudado de execução para pronto;
- **Pronto (*Ready)*** – É quando o processo está aguardando para ser executado. Nesse caso, o sistema operacional é que determina a ordem e o critério para que um processo em estado de pronto possa acessar o processador. Esse mecanismo é chamado de escalonamento de processos;
- **Espera (*wait)*** – Diz-se que um processo está em espera, se estiver esperando por algum recurso para prosseguir seu processamento. Por exemplo, a conclusão de um evento de entrada e saída.

Durante o processamento, um processo pode mudar de estado por causa de eventos gerados pelo sistema operacional ou pelo próprio processo. Assim, quatro transições são possíveis entre esses três estados. Observe abaixo:

Pronto -> Execução – Quando o processo é criado, ele é colocado em uma lista de execução em estado de pronto, que por sua vez fica aguardando até seu momento de ser executado.

Execução -> Espera – O estado de execução passa para o estado de espera por eventos externos ou por eventos gerados pelo próprio processo.

Espera -> Pronto – Um processo em que esteja em espera, passa para o estado pronto quando o recurso solicitado é concedido ou quando a operação solicitada é concluída. Um detalhe importante é que um estado em espera terá que voltar ao estado de pronto antes de prosseguir para sua execução. Ou seja, nenhum processo em espera pode passar diretamente para execução.

Execução -> Pronto – Um processo em execução pode passar para o estado de pronto por eventos gerados pelo próprio sistema, como exemplo, o final do tempo que o processo possui para execução.

Videoaula 2

Agora, assista o vídeo no qual trataremos sobre os conceitos abordados até o momento.

Videoaula 2

Utilize o QRcode para assistir!

Agora, assista o vídeo no qual trataremos sobre os conceitos abordados até o momento.

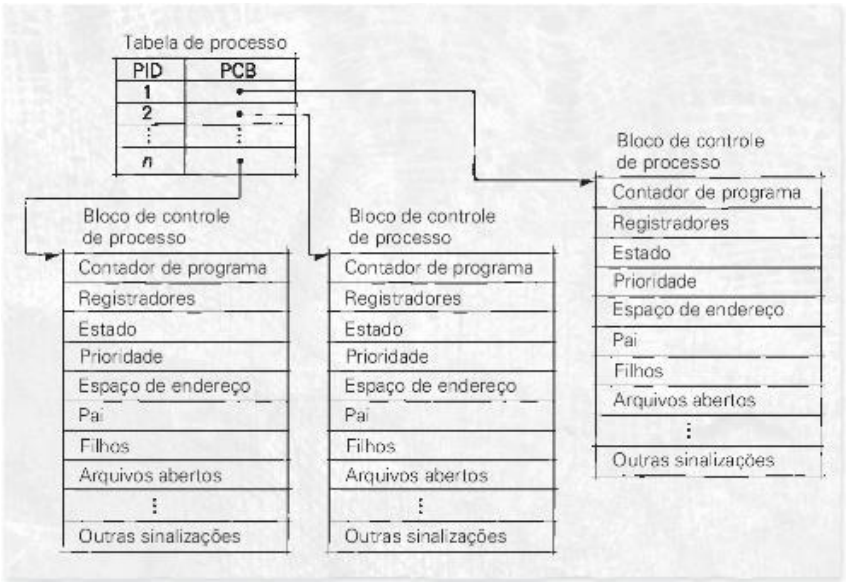
Então, caro aluno, você notou que o assunto processos tem bastante regras para serem seguidas. Imagine na prática, como deve ser controlar todos os processos que acontecem em um computador. Bem complexo, não é mesmo?

Por isso, para controlar a complexidade dos processos é usado o bloco de controle de processos (*PCB's*). Primeiramente o sistema operacional deve ser capaz de identificar cada processo existente, para isso designa ao processo um número de identificação (*Process Identification Number - PID).* Após isso, o sistema operacional cria o bloco de controle de processo (*Process Control Block - PCB),* cuja finalidade é manter as informações que o sistema operacional necessita para gerenciar o processo (DEITEL *et al.*, 2005). Os PCBs de modo geral podem incluir as seguintes informações:

- PID - *Process Identification Number;*
- Estado do processo (por exemplo, em execução, pronto ou espera);
- Contador do programa (um valor que determina qual instrução o processo deve executar em seguida;
- Prioridade de escalonamento;
- Ponteiros para localizar os dados e instruções do processo na memória;
- Ponteiros para recursos alocados (tais como arquivos);
- Conteúdo dos registradores.

Nas transições dos estados nos processos, o sistema operacional tem que atualizar as informações no PCB do processo. Normalmente o SO mantêm ponteiros para cada PCB do processo em uma tabela de processos. Dessa forma, o SO acessa o PCB mais rapidamente. Ou seja, temos uma tabela de processo que apontam para os blocos de controle de processo, observe essa ideia na figura 1 abaixo, para ter mais clareza nesse conceito.

Figura 1 - Tabela de processo e blocos de controle de processo



Fonte: Deitel *et al.* (2005, p.70)

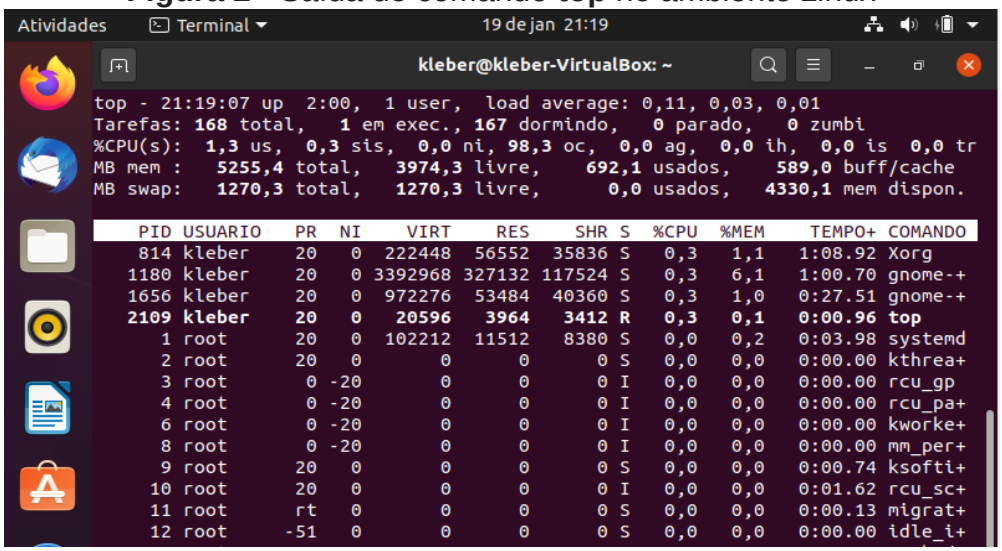
Para finalizar a ideia de processos que tal algo prático agora? Podemos trabalhar com processos nos sistemas operacionais usados nos computadores pessoais. Vamos aqui fornecer um exemplo utilizando o sistema *Linux*. Nesse sistema, existem determinados comandos que auxiliam no gerenciamento dos processos. No *Windows* é utilizado o Gerenciador de tarefas.

Utilizando-se o ***shell*** do *Linux*, pode-se utilizar o comando **top.** O comando top é a maneira mais comum de verificar o uso de processos do sistema e constatar quais deles estão consumindo mais memória ou processamento por exemplo. Observando a figura 2 abaixo notamos que os processos são apresentados em colunas que mostram sua propriedade, apresentaremos apenas 4 importantes colunas mostradas na saída do comando:

- **PID** (Identificador do processo): a identificação do processo (identificador único);
- **Usuário**: usuário proprietário do processo;
- **PR** (Prioridade): a prioridade de agendamento do processo. Alguns valores neste campo são RT. Isso significa que o processo está sendo executado em tempo real (*Real Time*);

- **S** (*State* – estado): este é o estado do processo. No *Linux* ele pode ter um dos seguintes valores: D – ininterrupto, R – executando, S – dormindo, T – rastreado ou parado, Z – zumbi.

Figura 2 - Saída do comando **top** no ambiente *Linux*



Fonte: o autor

Conceito de *Threads*

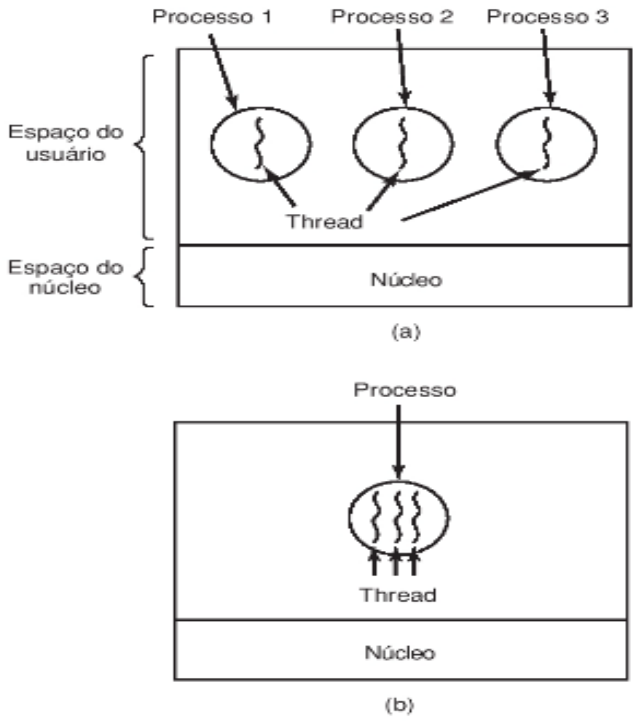
Os sistemas operacionais nos finais dos anos 70, suportavam apenas um programa associado a um processo, este era o conceito monothread. Ou seja, não permitiam mais de um programa associado a um processo do sistema operacional.

Quando um processo suporta mais do que duas tarefas dentro dele mesmo, podemos afirmar que o processo trabalha com multithreads. Os *threads* então permitem que múltiplas execuções ocorram no mesmo ambiente do processo, com um grande grau de independência uma da outra (TANENBAUM, 2015).

Com esse conceito de multithread é possível implementar programas do usuário e serviços do sistema operacional que são executados de forma concorrente. Como os *threads* de um processo compartilham o mesmo espaço de endereçamento, a comunicação entre os *threads* é feita de forma muito eficiente porque não exige os mecanismos de comunicação entre processos.

Vamos utilizar-se de uma ilustração para alicerçar ainda mais esse conceito tão importante. Observe a figura 3 abaixo.

Figura 3 - a) Três processos, cada um com um *thread*. b) Um processo com três *threads*



Fonte: Tanenbaum (2015, p. 61)

Observando a figura 3(a) notamos que cada processo possui seu próprio espaço de endereçamento e um *thread* de controle. Em contrapartida, na figura 3(b) observamos um único processo com três de controle. Contudo em ambos casos existe três *threads*. Ainda na figura 3(a), cada *thread* trabalha em um espaço de endereçamento diferente, enquanto que na figura 3(b) todos os três compartilham o mesmo espaço de endereçamento.

Continuando ainda o estudo sobre os *threads*, podemos citar mais algumas características desse assunto tratado até aqui.

- *Threads* tem exatamente o mesmo espaço de endereçamento, o que implica que eles também compartilham as mesmas variáveis globais;
- Como nos processos tradicionais, um *thread* pode estar em um dos vários estados: em execução, espera, pronto;
- Um *thread* pode auto responder-se sem que seja preciso duplicar um processo inteiro, economizando recursos como memória, processamento e aproveitando dispositivos de I/O, variáveis e outros meios;
- Além do espaço de endereçamento, todos os *threads* compartilham o mesmo conjunto de arquivos abertos, processos filhos, sinais etc.

Comunicação entre processos

É normal processos que trabalham juntos (concorrentes) compartilhem recursos do sistema, como arquivos, registros, dispositivos e áreas de memória. Para que o compartilhamento funcione entre os processos, o sistema operacional deve garantir a comunicação e a sincronização bem estruturada entre eles.

Quando dois processos acessam o mesmo recurso em um mesmo instante, cria-se uma condição no contexto de sistema operacionais chamada de **condição de corrida**. É uma condição que deve ser evitada para evitar problemas de corrupção dos dados do arquivo.

Então qual o mecanismo implementado para evitar a condição de corrida em processos e *threads*? A técnica utilizada para evitar a condição de corrida é a **exclusão mútua**, na qual os processos são impedidos de acessar algum recurso compartilhado e que já esteja em uso por outro processo.

A exclusão mútua deve afetar somente os processos concorrentes quando um deles estiver fazendo acesso ao recurso compartilhado. A parte do código do programa onde é feito o acesso ao recurso compartilhado é denominado **região crítica**. O ideal é tentar evitar que dois processos atinjam as suas regiões críticas ao mesmo tempo (MACHADO; MAIA, 2000).

Então, para tratar o problema de exclusão mútua e sincronização é utilizado técnicas no âmbito do *hardware* e do *software* (MACHADO; MAIA, 2000). Vamos verificar primeiramente a técnica por *hardware*.

Para soluções via *hardware* temos:

- Desabilitação de interrupções: uma solução simples para a exclusão mútua é fazer com que o processo, antes de entrar em sua região crítica, desabilite todas as interrupções externas e as reabilite após deixar a região crítica;
- Instrução *Test-And-Set*: os processadores possuem uma instrução especial (*test – and - set)*, que permite ler uma variável e armazenar seu conteúdo em outra área e atribuir um novo valor a essa variável. Tem como característica, ser executada sem interrupção, sendo assim uma instrução indivisível. Dessa forma, não existe a possibilidade de dois

processos estarem manipulando uma variável compartilhada ao mesmo tempo.

Além da exclusão mútua que soluciona o problema de compartilhamento de recurso, três fatores que são fundamentais para a solução dos problemas de sincronização deverão ser atendidos (MACHADO; MAIA, 2000):

- O número de processadores e o tempo de execução dos processos concorrentes devem ser irrelevantes;
- Um processo, fora de sua região crítica, não pode impedir que outros processos entrem em suas próprias regiões críticas;
- Um processo não pode permanecer indefinidamente esperando para entrar em sua região crítica.

Videoaula 3

Agora, assista o vídeo no qual abordaremos sobre os conceitos citados.

Videoaula 3

Utilize o QRcode para assistir!

Agora, assista o vídeo no qual abordaremos sobre os conceitos citados.

Vamos agora estudar alguns recursos em *software* que procuram solucionar problemas para a sincronização e exclusão mútua dos processos (MACHADO; MAIA, 2000).

Semáforos: no contexto de sistemas operacionais, semáforo é uma variável especial protegida que tem como função controlar o acesso a recursos compartilhados em um ambiente multitarefa.

O valor que um semáforo contém indica quantos processos ou *threads* podem acessar a um recurso compartilhado. As operações sobre os semáforos são:

- **Inicialização**: recebe um valor inteiro indicando a quantidade de processos que podem acessar um determinado recurso;
- **Operação *wait*** ou P: decrementa o valor do semáforo. Se o semáforo está com valor zero, o processo é posto para dormir;
- **Operação *signal*** ou V: se o semáforo estiver com o valor zero e existir algum processo adormecido, um processo será acordado. Caso contrário, o valor do semáforo é incrementado.

Semáforos são utilizados em situações na qual necessita-se exclusão mútua, ou seja, apenas que um processo execute por vez.

Monitores: a utilização de semáforos ajuda bastante a função dos programadores e o seu uso é bastante simplificado. Porém, exige do programador muito cuidado, pois qualquer engano pode levar a problemas de sincronização imprevisíveis e difíceis de reproduzir. Para isso, foi criada uma unidade básica de sincronização de alto nível que são os monitores.

Para Machado e Maia (2000), o monitor é um conjunto de procedimentos, variáveis e estruturas de dados definido dentro de um módulo. Sua característica mais importante é a implementação automática de exclusão mútua entre seus procedimentos. Toda vez que um processo chama um desses procedimentos, o monitor verifica se já existe outro processo executando algum procedimento do monitor. Caso exista, o processo ficará aguardando a sua vez até que tenha permissão para executar.

Troca de mensagens: é uma técnica de comunicação e sincronização entre processos, implementado pelo sistema operacional através de duas rotinas do sistema: *SEND e RECEIVE.* A rotina *SEND* é responsável por enviar uma mensagem para um processo receptor, e a rotina *RECEIVE* por receber uma mensagem de um processo transmissor. Essas rotinas permitem a comunicação e a sincronização entre processos.

A comunicação entre processos pode ser feita diretamente ou indiretamente. Da forma direta, simplesmente o processo que deseja enviar ou receber uma mensagem tem que endereçar explicitamente o nome do processo receptor ou transmissor. Uma característica do endereçamento direto é que ele só permite a comunicação entre dois

processos e que o seu maior problema é justamente a necessidade da especificação do nome dos processos envolvidos na troca de mensagens.

Já a comunicação indireta, utiliza uma área compartilhada, onde as mensagens podem ser colocadas pelo processo transmissor e retiradas pelo receptor. Esse tipo de buffer é conhecido como *mailbox.* No endereçamento indireto, vários processos podem estar associados a *mailbox*.

Por fim, existem duas formas de comunicação entre processos através da troca de mensagens: comunicação síncrona, onde processos enviam as mensagens e esperam a resposta em determinado tempo e não precisam de *buffer* e a comunicação assíncrona, nem o receptor permanece aguardando o envio de uma mensagem, nem o transmissor o seu recebimento. Para esses casos há a necessidade de *buffer* para armazenar as mensagens.

Leitura obrigatória

Para complementar o assunto estudado nessa aula, indico a leitura do capítulo 2 os itens 2.1 a 2.4 do livro do TANENBAUM, **Sistemas Operacionais Modernos.**

Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/1233/pdf/72. Acesso em: 8 maio 2021.

Encerramento da Unidade

Que pena! Chegamos ao final da nossa unidade. Começamos estudando sobre o que é um sistema operacional. Vimos que um SO é um conjunto de rotinas e serviços executado pelo computador que tem como objetivo facilitar o uso do computador pelo usuário e gerenciar e controlar o uso dos recursos do computador.

Na aula 2, aprendemos que um processo pode ser um programa em execução. Com isso, podemos entender como os sistemas operacionais de hoje realizam e controlam muitas atividades simultâneas. Um processo pode ter alguns estados: execução, pronto, espera. Como são muitos processos compartilhando recursos, precisam existir métodos

para gerenciar o compartilhamento e sincronizar os processos. Dentre esses métodos podemos citar: semáforo, monitores e troca de mensagens.

Por fim, o assunto abordado nessa unidade é de fundamental importância para que você entenda como é complexo o trabalho do sistema operacional. Com esses conceitos estudados em mente, meu caro aluno, você poderá entender o funcionamento interno de um SO que poderá ajudá-lo a trabalhar com esses sistemas.

Referências

DEITEL, Harvey M. *et al*. **Sistemas Operacionais**. 3. ed. São Paulo: Pearson Education Brasil, 2005. 784 p.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais**. 2. ed. Rio de Janeiro: Ltc, 2000. 232 p.

SIQUEIRA, Fernando de. **Gerência de Memória**. Disponível em: https://sites.google.com/site/proffernandosiqueiraso/aulas/2-a-evolucao-dos-sistemas-operacionais Acesso em: 27 fev. 2021.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2015. 864 p.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR