**Introdução da Unidade**



**Objetivos da Unidade**

Ao longo desta Unidade, você irá:

* definir o conceito, as características, a hierarquia e os estados dos processos e *threads*;
* analisar as características sobre a comunicação, como condições de disputa, regiões críticas e exclusão mútua com espera ociosa;
* explicar como é realizado o escalonamento entre processos e *threads,*distinguir os tipos de escalonamento e seus principais algoritmos.

**Introdução da Unidade**

Olá estudante, boas-vindas!

Nesta unidade veremos: o conceito, as características e o funcionamento de processo e *threads* e como eles são implementados, como os processos são criados e finalizados pelo sistema operacional e a hierarquia e os estados de processos. Um processo pode ser definido como um programa em execução. Porém, o sistema operacional não executa apenas os programas que solicitamos que sejam rodados.

Por exemplo, quando abrimos uma página em um navegador, ela pode acionar outros processos para que seja devidamente carregada, sem impactar as demais abas abertas. Além disso, existem outros processos executados no computador que não são propriedade do usuário, mas que são acionados para garantir a performance do sistema operacional, como o antivírus e o gerenciamento da rede.

Após o término desta unidade, você terá condições de entender o funcionamento e a implementação dos processos e *threads*. Aprendendo sobre os processos, você terá um conhecimento sobre o Gerenciador de Tarefas de seu sistema operacional, o que o ajudará a manter o bom funcionamento do seu computador.

Neste contexto, Lucas acabou de ser contratado como gerente da área de Tecnologia da Informação de uma empresa prestadora de serviços hospitalares. Além de fornecer suporte aos usuários dos sistemas hospitalares, outra função realizada na empresa é fazer pesquisas na internet sobre a qualidade de serviços na área da saúde, baixar livros e arquivos da área que auxiliam o desenvolvimento de aplicativos em TI e editar textos sobre o processo de avaliação e certificação da qualidade dos serviços hospitalares.

Sempre que uma dessas atividades realizadas por Lucas é executada, um processo é criado e ele pode chamar outros processos para auxiliá-lo por meio da transferência de informações entre eles. Para escolher qual processo será executado, o escalonador é acionado pelo sistema operacional e define os critérios de acesso à CPU que cada processo terá para que não afete a execução das aplicações. Ao receber um chamado de atendimento de suporte aos usuários dos sistemas hospitalares, um estagiário da equipe liderada por Lucas o aciona para ir até a sala do usuário que realizou a abertura do chamado.

Durante o percurso, Lucas é questionado sobre como os processos são criados e finalizados, sobre como o sistema operacional trata a hierarquia de processos e seus estados, sobre o funcionamento da comunicação entre processos e seus problemas clássicos da comunicação e sobre como trabalham o escalonador de processos e os algoritmos de escalonamento.

Para entender melhor o funcionamento dos processos e os algoritmos de escalonamento, Lucas sugere ao estagiário que seja implementado um algoritmo de escalonamento de processos por meio de semáforos.

Vamos juntos conhecer mais sobre essas questões através de algoritmos!

**Introdução da aula**



**Qual é o foco da aula?**

Nessa aula, vamos apresentar o conceito, as características, a hierarquia e os estados dos processos e *threads* e como se dá a criação e o término de processos.

**Objetivos gerais de aprendizagem**

Ao longo desta aula, você irá:

* descrever o modelo responsável por organizar os programas executáveis em processos sequenciais;
* explicar os eventos que fazem um processo: início do sistema, execução de uma chamada ao sistema de criação por um processo em execução, requisição do usuário para criar um novo processo e início de um *job* em lote;
* explicar o conceito, as características, a implementação e os gerenciamentosde*thread*.

**Situação-problema**

Os processos são programas ou tarefas em execução e o sistema operacional é o responsável por administrá-los, por meio do gerenciador de processos. Existem os processos iniciados pelo usuário, como executar um editor de textos, abrir uma página na internet, abrir o aplicativo de músicas, entre outros.

Há também os processos iniciados por outros processos, por exemplo, uma página da internet solicitando a ajuda de outro processo para fazer o carregamento dos seus elementos.

Relembrando nosso contexto, Lucas acabou de ser contratado como gerente da área de Tecnologia da Informação de uma empresa prestadora de serviços hospitalares. Durante o atendimento do chamado (supervisionado por Lucas), o usuário questiona sobre o *software* de edição de arquivos usado na empresa, por travar muito ao formatar e salvar o arquivo.

Para ambientar e auxiliar o estagiário, Lucas assumiu o atendimento, fechou o programa e o executou novamente, porém o *software* continuou travando. Ele então acionou o Gerenciador de Tarefas e identificou que o *software* de edição de arquivo ainda estava em execução.

O estagiário fez os seguintes questionamentos:

* o que aconteceu para que o programa não tenha sido finalizado corretamente?
* como fazer para conseguir executar o *software* corretamente?
* no caso de o *software* travar durante o salvamento e formatação, a implementação de *threads* auxiliaria o processo?

Não se esqueça de compilar todas as informações obtidas para lhe auxiliar no desenvolvimento de um algoritmo de escalonamento de processos por meio de semáforos.

Para que você consiga responder esse e outros questionamentos, nesta aula vamos conhecer mais sobre os processos e os conteúdos pertinentes a este tema.

Bons estudos!

**Processos: conceito e criação**



Um dos conceitos principais em sistemas operacionais gira em torno de processos.

Um processo pode ser definido como um programa em execução, porém o seu conceito vai além desta definição (MACHADO; MAIA, 2007).

Nos computadores atuais, o processador funciona como uma linha de produção executando vários programas ao mesmo tempo de forma sequencial, como ler um livro on-line, baixar um arquivo e navegar na internet. A CPU é responsável por alternar os programas, executando-os por dezenas ou centenas de milissegundos, para que cada um tenha acesso ao processamento, dando a ilusão ao usuário de paralelismo ou pseudoparalelismo (TANENBAUM, 2003).

O pseudoparalelismo é a falsa impressão de que todos os programas estão executando ao mesmo tempo, mas na verdade o que acontece é que um processo em execução é suspenso temporariamente para dar lugar ao processamento de outro processo e assim sucessivamente.

Segundo Tanenbaum (2003), para tratar o paralelismo de forma mais fácil, foi desenvolvido um modelo responsável por organizar os programas executáveis em processos sequenciais. Um processo pode ser definido como um programa em execução incluindo os valores do contador de programa atual, registradores e variáveis. A CPU alterna de um processo para outro a cada momento, essa alternância é conhecida como multiprogramação.

A diferença entre processos e programa é importante para que o modelo seja entendido.

A seguinte analogia é utilizada para exemplificar a diferença entre processo e programa: fazer um bolo. Para fazer um bolo, é necessário todos os ingredientes e a receita. A receita pode ser considerada como o programa, os ingredientes são os dados de entrada e a pessoa que prepara o bolo é o processador.

Os processos são as atividades que a pessoa faz durante a preparação do bolo: ler a receita, buscar os ingredientes, misturar a massa e colocar o bolo para assar, que é o processo final desse programa “receita de bolo”.

Ainda neste exemplo, imagine que o filho da pessoa que está fazendo o bolo tenha se machucado. A pessoa guarda em que parte do processamento parou e vai socorrer o filho. Ela pega o kit de primeiros socorros e lê o procedimento para tratar do machucado do filho. Neste momento, vemos o processador (a pessoa) alternando de um processo (fazer o bolo) para outro processo com prioridade maior (socorrer o filho), cada um com seu programa – receita versus procedimento de tratamento do machucado. Assim que o filho estiver medicado, então a pessoa retornará a fazer o bolo do ponto em que parou.

Podemos considerar então que um processo é uma atividade que contém um programa, uma entrada, uma saída e um estado. Veremos a seguir a criação de processo e os estados dos processos.

\_\_\_\_\_\_

**📝 Exemplificando**

Os serviços que os sistemas operacionais podem implementar através de processos são (MACHADO; MAIA, 2007):

* auditoria e segurança do sistema.
* a contabilização do uso de recursos.
* a contabilização de erros.
* gerência de impressão.
* comunicação de eventos.
* serviços de redes.
* interface de comandos (*Shell*), entre outros.

\_\_\_\_\_\_

**Criação de processos**

Os sistemas operacionais devem oferecer formas para que processos sejam criados. Segundo Tanenbaum (2003), existem quatro eventos que fazem com que um processo seja criado:

**1) início do sistema**: quando o sistema operacional é inicializado, são criados vários processos. Existem os de primeiro plano, que interagem com os usuários e suas aplicações, e os de segundo plano, que possuem uma função específica, como um processo para atualizar e-mails quando alguma mensagem é recebida na caixa de entrada.

Para visualizar os processos em execução no Windows, pressione as teclas CTRL+ALT+DEL e no Linux utilize o comando ps.

**2) execução de uma chamada ao sistema de criação por um processo em execução**: por exemplo, quando um processo está fazendo *download*, ele aciona um outro processo para ajudá-lo. Enquanto um faz o *download*, o outro está armazenando os dados em disco.

**3) uma requisição do usuário para criar um novo processo**: quando o usuário digita um comando ou solicita a abertura de um ícone para a abertura de um aplicativo.

**4) início de um *job* em lote**: esses processos são criados em computadores de grande porte, os *mainframes*.

\_\_\_\_\_\_

**🔁 Assimile**

Vimos nesta aula que existem os processos de primeiro plano, que são processos dos usuários e interagem com suas aplicações, e os de segundo plano, que possuem função específica. Os processos de segundo plano, ao executarem uma função específica, são chamados de *daemons*.

\_\_\_\_\_\_

**Término de processos**

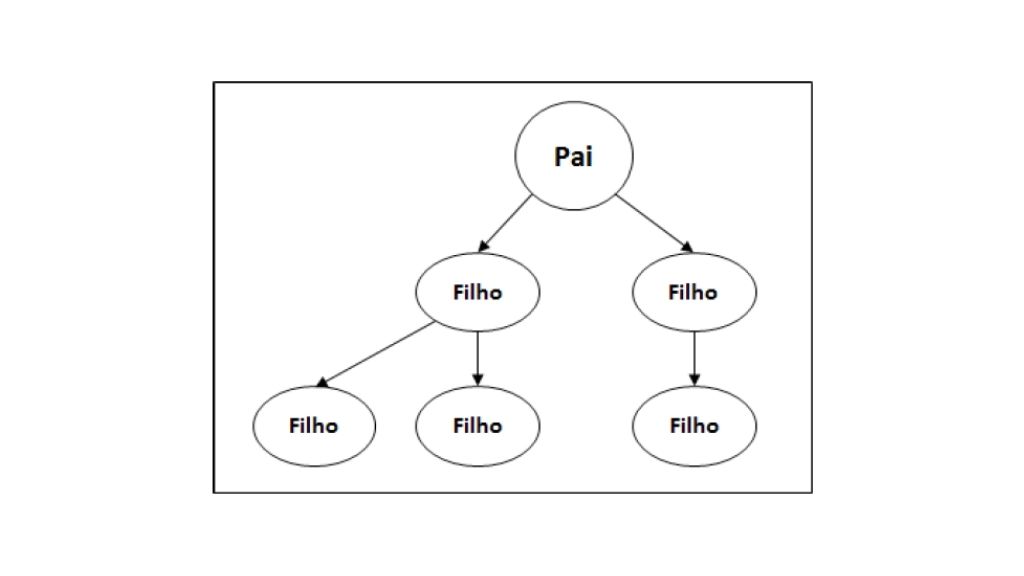
Após a criação, os processos podem ser finalizados nas seguintes condições, (TANENBAUM, 2003):

* **saída normal (voluntária):** acontece quando o processo acaba de executar por ter concluído seu trabalho.
* **saída por erro (voluntária):** acontece quando o processo tenta acessar um arquivo que não existe e é emitida uma chamada de saída do sistema. Em alguns casos, uma caixa de diálogo é aberta perguntando ao usuário se ele quer tentar novamente.
* **erro fatal (involuntário):** acontece quando ocorre um erro de programa, por exemplo, a execução ilegal de uma instrução ou a divisão de um número por zero. Neste caso, existe um processo com prioridade máxima que supervisiona os demais processos e impede a continuação do processo em situação ilegal.
* **cancelamento por um outro processo:** acontece quando um processo que possui permissão emite uma chamada ao sistema para cancelar outro processo.

**Hierarquia de Processos**



Segundo Tanenbaum (2003), em alguns sistemas, quando um processo cria outro, o processo-pai e o processo-filho ficam associados. O filho pode gerar outros processos, criando, assim, uma hierarquia de processos, conforme apresentado na figura.

Hierarquia de processos. Fonte: elaborada pelo autor.

No Unix, um processo-pai, seus filhos e descendentes formam um grupo de processos. Por exemplo, quando um usuário envia um sinal do teclado (como *CTRL + ALT + DEL*), este sinal é entregue para todos os processos que compõem o grupo de processos do teclado. Quando um processo-pai é “morto”, todos os filhos vinculados a ele são “mortos” também.

O Windows não possui uma hierarquia de processos. Cada um possui um identificador próprio e quando um processo cria outro, existe uma ligação entre eles, mas ela é quebrada quando o processo- pai passa seu identificador para outro processo. Quando um processo-pai é “morto”, os processos vinculados a ele não são mortos.

**Estados do Processo**

Os processos podem passar por diferentes estados ao longo do processamento. Um processo ativo pode estar em três estados (MACHADO; MAIA, 2007):

* **em execução**: um processo está em execução quando está sendo processado pela CPU. Os processos são alternados para a utilização do processador.
* **pronto**: um processo está no estado de pronto quando possui todas as condições necessárias para executar e está aguardando. O sistema operacional é quem define a ordem e os critérios para execução dos processos.
* **espera ou Bloqueado**: um processo está no estado de espera quando aguarda por um evento externo (um comando do usuário, por exemplo) ou por um recurso (uma informação de um dispositivo de entrada/saída, por exemplo) para executar.

Quatro mudanças podem acontecer entre os estados, representadas na figura a seguir.

Transições dos processos. Fonte: Tanenbaum (2003, p. 58).

Conforme apresentado na figura “Transições dos processos”, a mudança 1 (“Em execução” para “Bloqueado”) acontece quando um processo aguarda um evento externo ou uma operação de entrada/saída e não consegue continuar o processamento.

As mudanças 2 (“Em execução” para “Pronto”) e 3 (“Pronto” para “Em Execução”) são realizadas pelo escalonador sem que o processo saiba. O escalonador de processos é o responsável por decidir em qual momento cada processo será executado. A mudança 2 acontece quando o escalonador decide que o processo já teve tempo suficiente em execução e escolhe outro processo para executar.

A mudança 3 ocorre quando os demais processos já utilizaram o seu tempo de CPU e o escalonador permite a execução do processo que estava aguardando.

A mudança 4 (“Bloqueado” para “Pronto”) ocorre quando a operação de entrada/saída ou o evento externo que o processo estava esperando ocorre. Assim, o processo retorna para a fila de processamento e aguarda novamente a sua vez de executar.

**Threads**



Segundo Machado e Maia (2007), o conceito de *thread* foi introduzido para reduzir o tempo gasto na criação, eliminação e troca de contexto de processos nas aplicações concorrentes, assim economizando recursos do sistema como um todo.

*Thread* é um fluxo de controle (execução) dentro do processo, chamado também de processos leves. Um processo pode conter um ou vários *threads* que compartilham os recursos do processo.

A principal razão para o uso de *thread*é que as aplicações da atualidade rodam muitas atividades ao mesmo tempo e quando são compostas por *threads*, podem ser executadas em paralelo. Outro motivo para a criação de *thread* é que são mais fáceis de criar e destruir, por não terem recursos vinculados a eles.

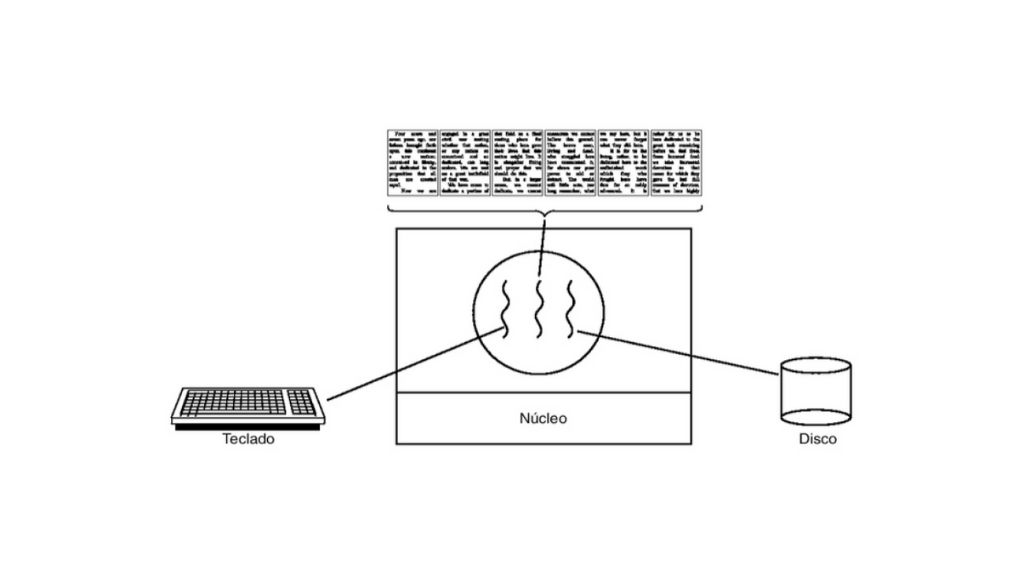
Em relação ao desempenho, quando uma aplicação processa muitas informações de entrada/saída, o uso de *threads* acelera a execução da aplicação.

Para entender melhor o conceito de *thread*, vamos apresentá-lo utilizando como exemplo um processador de texto (TANENBAUM, 2003).

Imagine que você esteja escrevendo um livro e que seja necessário trocar um termo errado escrito em diversos capítulos. Se o livro estiver em um único arquivo, fica fácil fazer a alteração, porém se estiver em arquivos separados, será necessário fazer a correção em cada um.

Outra questão é: se uma ou duas páginas forem removidas de um arquivo de 500 páginas, o processador de textos terá que formatar todo o documento em todas as páginas até chegar à página retirada, o que causará uma grande demora.

Neste caso, se forem utilizados *threads*, um *thread* auxiliaria a formatação do texto, outro atenderia aos requisitos do usuário via teclado e um terceiro *thread* faria o *backup* dos dados em disco sem interferir nas ações dos demais *threads*, conforme apresentado na figura.

Processador de texto com três threads. Fonte: Tanenbaum (2003, p. 64).

Outro exemplo do uso de *threads* é em um navegador web, enquanto um *thread* carrega imagens ou textos de uma página e outro *thread* recupera dados de uma rede.

\_\_\_\_\_\_

**➕ Pesquise mais**

Algumas linguagens de programação da atualidade, como Java, possuem recursos para a implantação de *threads*, chamados programação concorrente.

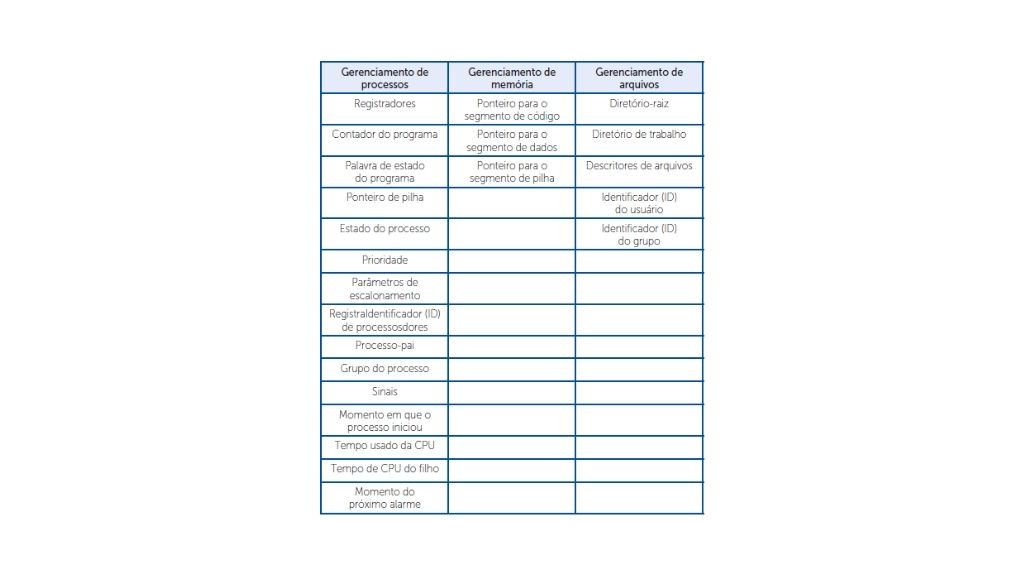
Para saber mais sobre esse assunto, leia o artigo [**Trabalhando com *Threads* em Java**](https://www.devmedia.com.br/trabalhando-com-threads-em-java/28780).

\_\_\_\_\_\_

**Implementação de Processos**

Para implementar o modelo de processos, o sistema operacional mantém um quadro de processos contendo informações sobre o estado do processo, seu contador de programa, o ponteiro da pilha, a alocação de memória, o *status* dos arquivos abertos, entre outros, que permitem que o processo reinicie do ponto em que parou (TANENBAUM, 2003).

O quadro apresenta os campos mais importantes. No primeiro campo estão os dados necessários para o armazenamento, referentes ao gerenciamento de processos. O segundo e o terceiro campos referem-se aos dados do gerenciamento de memória e ao gerenciamento de arquivos.

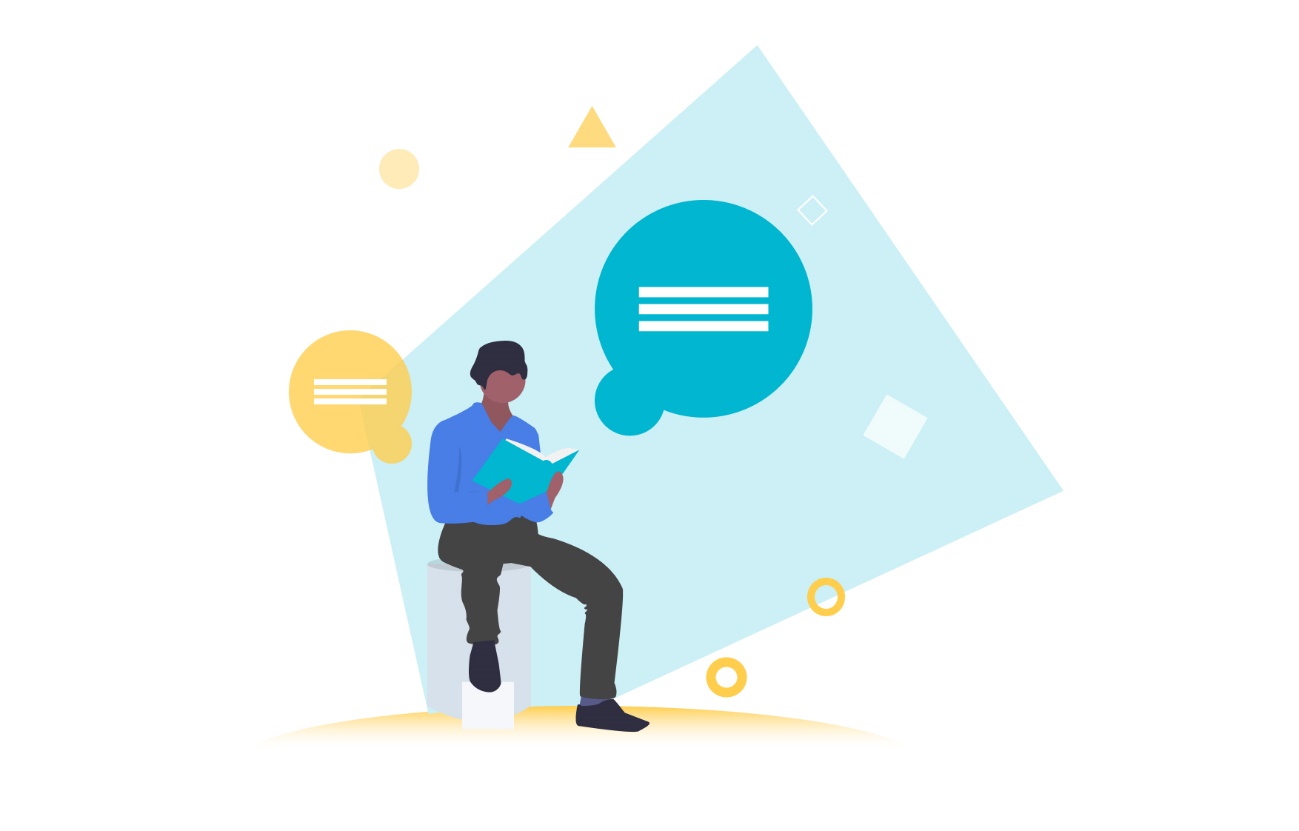
Processos. Fonte: Tanenbaum (2003, p. 59).

**Implementação de *Threads***

A implementação de *threads* pode ocorrer no espaço do usuário, no núcleo do sistema operacional e em uma implementação híbrida (no espaço do usuário e do núcleo).

* ***Thread* de usuário**: são implementados pela aplicação do usuário e o sistema operacional não sabe de sua existência. A vantagem é que não é necessária nenhuma mudança entre os modos de usuário e núcleo, tornando-se rápido e eficiente.
* ***Thread* do núcleo**: são implementados e gerenciados pelo núcleo do sistema operacional. A desvantagem desta implementação é que todo o gerenciamento dos *threads* é feito por chamadas ao sistema, o que compromete a performance do sistema.
* ***Threads*híbridos**: são implementados tanto no espaço do usuário, quanto no núcleo do sistema operacional. O sistema operacional sabe dos *threads* do usuário e faz o seu gerenciamento. A vantagem desta implementação é a flexibilidade em função das duas implementações.

**Conclusão**



Agora que você já aprendeu sobre os conceitos, as características e o funcionamento de processo e *threads*, sobre como os processos são criados e finalizados pelo sistema operacional, sobre a hierarquia e os estados de processos e sobre como os processos e *threads* são implementados, vamos voltar ao nosso contexto?

Lucas estava auxiliando o atendimento realizado por um estagiário de sua equipe e surgiram os seguintes questionamentos:

* o que aconteceu para que o programa não tenha finalizado corretamente?
* como fazer para conseguir executar o *software* corretamente?
* no caso de o *software* travar durante o salvamento e a formatação, a implementação de *threads* auxiliaria o processo?

O problema de travamento de um *software* pode acontecer tanto em função do próprio *software*, quanto por causa de um *hardware*.

Como Lucas identificou que o processo estava em execução ao acionar o Gerenciador de Tarefas, porém estava travado, constatou-se o problema no processo que estava rodando. Neste caso, o processo travou por algum erro encontrado, por exemplo, na memória durante a gravação do arquivo feita pelo editor de texto.

O correto é “matar” o processo e reiniciar o *software* de edição. Ao fazer isso, verifique se outro processo em execução impactará no salvamento do arquivo e, em caso positivo, feche-o antes de reiniciar o *software* de edição. A implementação de *threads* durante a execução do processo agiliza o processamento e melhora o desempenho das aplicações.

O próprio conceito de *threads* diz “são as entidades programadas para a execução sobre a CPU” (TANENBAUM, WOODHULL, 2008, p. 78).

Logo, ao adicionar *threads* em uma aplicação, esta conseguirá processar duas ou mais coisas ao mesmo tempo. Por exemplo, no caso do *software* de edição de arquivos, um *thread* seria responsável por receber os comandos do usuário via teclado, outro *thread* seria responsável por formatar o arquivo e outro seria responsável por salvar o arquivo, todos os *threads* trabalhando em sincronia e executando as atividades em paralelo.