Logotipo, nome da empresa

Descrição gerada automaticamente

**ANHANGUERA EDUCACIONAL**

**UNIDADE SANTANA**

**Curso de Tecnologia em Análise e Desenvolvimento de Sistemas**

**Thiago Melo da Silva – RA: 401966816563**

**João Victor Martins da Silva – RA: 402000516563**

**João Victor Souza de Lucena – RA: 402278816563**

**Elementos de Software para Sistemas Operacionais**

**São Paulo**

**2024**

**Thiago Melo da Silva – RA: 401966816563**

**João Victor Martins da Silva – RA: 402000516563**

**João Victor Souza de Lucena – RA: 402278816563**

**Elementos de Software para Sistemas Operacionais**

Pesquisa apresentada ao curso de Análise e desenvolvimento de sistemas da Universidade Anhanguera como requisito parcial à obtenção de nota para aprovação da disciplina de Sistemas Operacionais aplicado aos cursos de Tecnologia Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Nicolas

**São Paulo**

**2024**

Sumário

[**Elementos de Software para Sistemas Operacionais** 8](#_Toc167130238)

[**PROCESSOS E THREADS** 8](#_Toc167130239)

[**PROCESSOS** 8](#_Toc167130240)

[**THREADS** 9](#_Toc167130241)

[**REGIÃO CRÍTICA** 9](#_Toc167130242)

[**COMUNICAÇÃO ENTRE PROCESSOS** 10](#_Toc167130243)

[**O QUE É IPC?** 11](#_Toc167130244)

# **Elementos de Software para Sistemas Operacionais**

## **PROCESSOS E THREADS**

Em um sistema operacional, processos e threads são entidades fundamentais para a execução de programas e a gestão dos recursos do sistema.

### **PROCESSOS**

Um processo é uma instância em execução de um programa. Ele inclui o código executável do programa, seus dados e recursos (como arquivos abertos, sockets de rede, etc.). Cada processo tem sua própria área de memória separada, o que significa que eles não podem acessar diretamente a memória uns dos outros. O sistema operacional gerencia os processos, atribuindo recursos a eles, escalonando sua execução na CPU e fornecendo mecanismos para comunicação e sincronização entre eles.

Os recursos de um processo são os elementos necessários para sua execução e funcionamento adequado. Aqui estão alguns dos principais recursos associados a um processo em um sistema operacional:

**Espaço de endereçamento:** Cada processo possui sua própria área de memória virtual, que inclui o código executável, os dados do programa, a pilha de execução e outras áreas necessárias para a execução do programa. O espaço de endereçamento é isolado entre processos, garantindo que um processo não possa acessar diretamente a memória de outro processo.

**Identificadores de processo (PID):** Cada processo é identificado por um número único chamado PID. O PID é usado pelo sistema operacional para identificar e gerenciar processos. Isso é importante para operações como escalonamento de CPU, gerenciamento de memória e comunicação entre processos.

**Recursos do sistema:** Um processo pode ter vários recursos do sistema associados a ele, como arquivos abertos, dispositivos de entrada/saída, conexões de rede, pipes, semáforos, entre outros. Esses recursos são compartilhados entre as threads do processo e são gerenciados pelo sistema operacional.

**Informações do estado do processo:** O sistema operacional mantém informações sobre o estado atual de um processo, como se está em execução, suspenso, pronto para execução ou terminado. Isso é crucial para o escalonamento da CPU e o gerenciamento de recursos.

**Contexto de execução:** Cada processo possui um contexto de execução, que inclui o estado da CPU (registradores, contador de programa, etc.) e outras informações necessárias para retomar a execução do processo de onde parou. O contexto de execução é salvo pelo sistema operacional quando um processo é interrompido e restaurado quando o processo é retomado.

**Permissões e privilégios:** Os processos têm diferentes níveis de permissões e privilégios, dependendo do usuário ou conta sob a qual estão sendo executados. Isso ajuda a garantir a segurança do sistema, controlando o acesso aos recursos do sistema por parte dos processos.

Esses são apenas alguns dos recursos associados a um processo em um sistema operacional. Eles são gerenciados pelo sistema operacional para garantir que os programas possam ser executados de maneira ordenada, eficiente e segura.

### **THREADS**

Uma thread é uma unidade de execução dentro de um processo. Um processo pode ter uma ou várias threads. Cada thread compartilha o mesmo espaço de endereçamento e recursos do processo pai, o que significa que elas podem acessar e modificar os mesmos dados. As threads são úteis para realizar tarefas simultâneas dentro de um processo, como processamento paralelo ou manipulação de eventos de interface do usuário.

Em resumo, os processos representam programas em execução, enquanto as threads representam as unidades de execução dentro desses programas. Ambos são gerenciados pelo sistema operacional para garantir que os recursos do sistema sejam utilizados eficientemente e que as operações do programa ocorram de maneira ordenada e segura.

## **REGIÃO CRÍTICA**

Uma região crítica em um sistema operacional ou software é uma seção de código onde dados compartilhados estão sendo acessados e manipulados por múltiplos processos ou threads concorrentes. O conceito de região crítica é fundamental para garantir a consistência e a integridade dos dados compartilhados em um ambiente multi-processo ou multi-thread.

A ideia por trás da região crítica é evitar condições de corrida, que ocorrem quando dois ou mais processos ou threads tentam acessar e modificar os mesmos dados simultaneamente, levando a resultados não determinísticos e possivelmente incorretos. Para prevenir condições de corrida, é necessário garantir que apenas um processo ou thread por vez tenha acesso à região crítica.

Para gerenciar o acesso concorrente à região crítica, geralmente são utilizados mecanismos de sincronização, como semáforos, mutexes (exclusão mútua), locks, entre outros. Esses mecanismos garantem que apenas um processo ou thread por vez possa entrar na região crítica, enquanto os outros aguardam em um estado de espera até que a região crítica esteja disponível.

Além disso, é importante minimizar o tempo gasto dentro de uma região crítica, já que isso pode afetar o desempenho do sistema. Portanto, o código dentro de uma região crítica deve ser o mais eficiente possível.

Em resumo, uma região crítica é uma seção de código onde dados compartilhados são acessados e manipulados, e o acesso a essa região precisa ser controlado para evitar condições de corrida e garantir a integridade dos dados.

## **COMUNICAÇÃO ENTRE PROCESSOS**

Nos sistemas operacionais tais como Windows e Linux muitos dos seus recursos operam sobre a forma de processos, e processos são instâncias de programas em execução, ou seja, um programa gera um processo, e também existem processos que rodam em segundo plano disponibilizando algum tipo de serviço, a seguir alguns exemplos:

* alsa – usado para inicializar o Alsa (Advanced Linux Sound Architecture).
* crond – usado para executar programas de usuários em tempo periódicos pré-determinados.
* dhcpd – usado para inicializar o servidor de Host Dinâmico dchp.
* httpd – usado para inicializar o servidor Web Apache.
* mysql – usado para inicializar o servidor de Banco de Dados MySQL.
* sshd – é um daemon que serve conexões ssh.
* syslog – usado para fornecer um arquivo unificado de log do sistema.

Quando o sistema operacional lança um processo, esse processo tem uma memória reservada para poder realizar a sua execução, ou seja, sua memória fica protegida somente para o seu uso, e para se comunicar com outros processos é necessário algum mecanismo que permita essa comunicação ou alguma outra forma que os dados sejam compartilhados entre os processos, é nesse momento que entra em cena o IPC.

### **O QUE É IPC?**

O IPC é um mecanismo que permite que dois ou mais processos realizem a troca de dados entre si. Isso é extremamente útil devido ao fato que um processo possui sua própria região de memória onde outros processos não tem a permissão de acessar aquele espaço de memória.

Existem dois tipos de IPC o pull-based e o push-based:

* **PULL-BASED**

Requer um meio, como um armazenamento para compartilhar os dados, isso porque os processos que querem ler esses dados precisam dar um pull desses dados. Neste caso os dados serão lidos através de um elemento intermediário, como um repositório de dados, onde um processo pode escrever dados nesse repositório enquanto outro processo realiza a leitura desses dados. Para esse grupo de IPC é fortemente recomendado realizar a sincronia para que não haja concorrência, os IPC referente a esse grupo são:

1. FIFO
2. Shared File
3. Shared Memory
4. Message Queue
5. MMAP
6. Semaphore

* **PUSH-BASED**

Neste caso os dados serão entregues diretamente ao processo que está realizando a leitura, os IPC referente a esse grupo são:

1. Signal
2. Socket
3. PIPE
4. DBUS

### **SEMÁFOROS**

Semáforos em sistemas operacionais são estruturas de dados usadas para controlar o acesso a recursos compartilhados entre processos ou threads, evitando condições de corrida e sincronizando sua execução. Existem dois tipos principais de semáforos:

* Semáforo Binário (Mutex):

Funciona como um cadeado que pode estar aberto (1) ou fechado (0).

Suas operações principais são:

1. ‘wait’ (ou ‘P’): Decrementa o semáforo. Se estiver fechado (0), bloqueia o processo ou thread até que seja aberto.
2. ‘signal’ (ou ‘V’): Incrementa o semáforo, liberando o recurso e permitindo que outros processos ou threads possam acessá-lo.

* Semáforo de Contagem:

Pode ter qualquer valor inteiro não negativo, suas operações principais são:

1. ‘wait’ (ou ‘P’): Decrementa o semáforo. Se o valor for zero, bloqueia o processo ou thread até que seja incrementado.
2. ‘signal’ (ou ‘V’): Incrementa o semáforo.

* Uso de Semáforos:

1. Exclusão Mútua: Proteção de seções críticas do código para garantir que apenas um processo ou thread acesse um recurso por vez.
2. Sincronização: Coordenação da ordem de execução entre processos ou threads.
3. Prevenção de Deadlock: Evitam a ocorrência de situações onde processos ou threads ficam bloqueados indefinidamente aguardando uns aos outros.

Suas vantagens são a flexibilidade e eficiência, já suas desvantagens são a complexidade e a espera ocupada.

### **TROCA DE MENSAGENS**

A troca de mensagens é uma técnica de IPC em que processos se comunicam enviando e recebendo mensagens entre si. Esse método é frequentemente usado em sistemas distribuídos e pode ser implementado de diferentes formas:

• Síncronas: O processo de envio fica bloqueado até que o destinatário receba a mensagem, o processo de recepção fica bloqueado até que uma mensagem esteja disponível. Sua vantagem é que ele garante a entrega e a sincronização, já sua desvantagem é que pode levar à ineficiência se os processos frequentemente bloquearem.

• Assíncronas: O processo remetente envia a mensagem e continua a executar sem esperar que a mensagem seja recebida. O processo destinatário pode receber a mensagem quando estiver pronto. Sua vantagem é o maior paralelismo e eficiência, sua desvantagem é a complexidade de lidar com buffers de mensagens e garantir a ordem correta das mensagens.

• Filas de Mensagens: As mensagens são armazenadas em filas até serem processadas pelo destinatário. Pode haver gerenciamento de prioridades nas filas para garantir que mensagens mais importantes sejam processadas primeiro. Utiliza mecanismos de enfileiramento.

• Sockets: Usados principalmente em comunicação de rede. Proporcionam um canal bidirecional para troca de mensagens entre processos em diferentes máquinas.

### Monitores

Os monitores são uma estrutura de sincronização de alto nível que facilita a comunicação entre processos, especialmente em ambientes de programação concorrente. Um monitor é um módulo ou objeto que encapsula variáveis compartilhadas, procedimentos que operam sobre essas variáveis e o próprio mecanismo de sincronização. Suas principais características são:

1. Encapsulamento:

Variáveis compartilhadas dentro do monitor só podem ser acessadas por procedimentos definidos dentro do próprio monitor. Isso previne acessos concorrentes não controlados e condições de corrida (race conditions).

2. Controle de Acesso:

Monitores usam primitivas de sincronização, como semáforos ou mutexes, para garantir que apenas um processo por vez possa executar um procedimento dentro do monitor. Implementam mecanismos de espera e sinalização, como wait e signal, para gerenciar o acesso e a sincronização entre processos.

3. Espera Condicional:

Um processo que não pode continuar a execução (por exemplo, porque uma condição necessária não está satisfeita) pode liberar o monitor e entrar em estado de espera. Outro processo pode então sinalizar (signal) quando a condição se tornar verdadeira, despertando o processo que estava esperando.

### **Troca de mensagens vs Monitores**

1. Troca de Mensagens:

* Mais adequada para sistemas distribuídos ou onde os processos não compartilham memória.
* Facilita a escalabilidade e a distribuição de processos em várias máquinas.
* Pode introduzir latências devido à necessidade de envio e recebimento de mensagens.

1. Monitores:

* Melhor aplicados em ambientes onde processos compartilham a mesma memória, como em sistemas multiprocessados ou multicore.
* Simplificam a sincronização e evitam condições de corrida através do encapsulamento e controle de acesso.
* Podem ser menos eficientes em sistemas distribuídos devido à dificuldade de gerenciar memória compartilhada entre máquinas.

Resumo:

Ambas as técnicas têm seus méritos e são usadas de acordo com os requisitos específicos do sistema ou aplicação em questão. A escolha entre troca de mensagens e monitores depende de fatores como a arquitetura do sistema, a necessidade de sincronização, a complexidade da implementação e os requisitos de desempenho.

# **Referências**

<https://embarcados.com.br/comunicacao-entre-processos/>

<https://profandreluisbelini.wordpress.com/wp-content/uploads/2012/03/aula-08-cap-07-sincronizac3a7c3a3o-e-comunicac3a7c3a3o-entre-processos.pdf>

https://www.ibm.com/docs/pt-br/engineering-lifecycle-management-suite/design-rhapsody/9.0.1?topic=frameworks-message-queues

https://www.zendesk.com.br/blog/comunicacao-sincrona-assincrona/

https://www.oitchau.com.br/blog/comunicacao-assincrona-e-sincrona/

Semáforos são uma ferramenta poderosa para a programação concorrente em sistemas operacionais, permitindo o controle seguro e eficiente de recursos compartilhados entre processos ou threads.Parte superior do formulário

Parte superior do formulário