AULA 4 – SINCRONIZAÇÃO ENTRE PROCESSOS

OBJETIVO DA AULA

Abordar questões de sincronização, *deadlocks*, *starvation* e condições de corrida.

APRESENTAÇÃO

Os sistemas operacionais atualmente lidam com diversos processos na memória (multitarefa) e esses processos disputam os recursos de *hardware* e *software* do computador, o que muitas vezes pode ocasionar alguns conflitos que precisam ser detectados e solucionados.

Além disso, os processos podem trocar dados e informações e isso precisa ser feito sincronizadamente para evitar erros e inconsistências.

Nesta aula veremos como o sistema operacional atua no sentido de evitar e resolver conflitos e promover a sincronização entre processos, garantindo o correto funcionamento do computador.

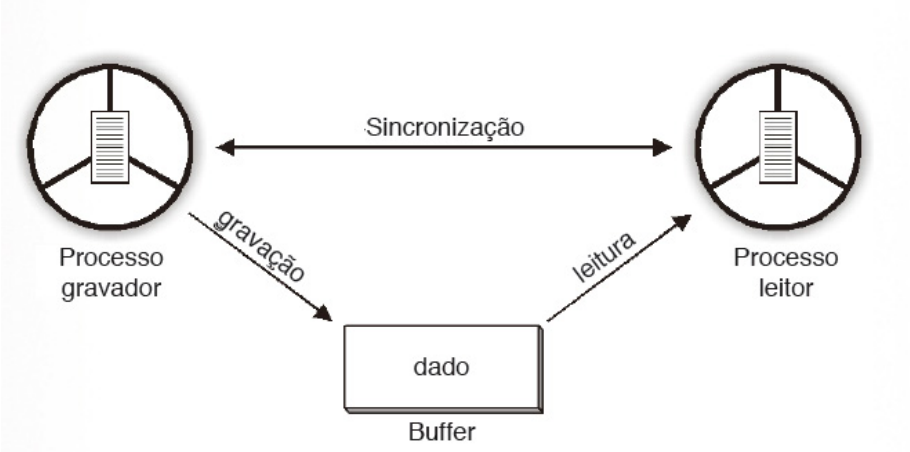
Mãos à obra!

1. INTRODUÇÃO

Com a chegada dos sistemas operacionais multiprogramáveis (multitarefa), surgiu a possibilidade de diferentes programas poderem compartilhar recursos e se comunicar, muitas vezes um processo produzindo informações que funcionam como dados para outro processo.

A comunicação entre processos pode ser realizada através de variáveis compartilhadas na memória principal ou diretamente através do envio de mensagens.

FIGURA 1 | **Sincronização/comunicação entre processos**



Fonte: https://sites.google.com/site/profferreirabrito/aulas/2-sincronizacao-e-comunicacao-de-processos.

Na Figura 1, dois processos utilizam um *buffer* para se comunicar. O processo gravador produz uma informação lida pelo processo leitor. As operações de gravação e leitura devem ocorrer rigorosamente nessa ordem, para evitar a passagem de informações erradas. Esse processo é chamado *sincronização* de processos.

Vamos examinar uma situação que mostra os problemas que podem ser causados por falha na sincronização de processos.

Suponha que dois processos, P1 e P2, manipulam uma variável compartilhada, A. P1 soma um ao valor de A e P2 soma três ao valor de A. Se o valor inicial de A é zero, espera se que seu resultado final, após as execuções de P1 e P2, seja quatro.

A execução de uma instrução de soma é desmembrada em instruções de máquina, que podem ser representadas de forma simples abaixo.

P1	P2
Carregue A, R1	*Carregue A, R2*
Some R1, 1	*Some R2, 3*
Armazene R1, A	*Armazene R2, A*

Se P1 for interrompido após a primeira instrução, teremos o registrador R1 com o valor inicial de A (zero). Se P2 executar suas três instruções antes que P1 retome o processador, ele pegará também o valor inicial de A e somará três a esse valor, e A terminará com valor igual a três.

Em seguida, quando P1 retomar sua execução, ele trabalhará com o valor guardado em R1 (zero), somará um a esse valor e A terminará com valor igual a um e como ele deveria valer quatro, teremos a ocorrência de um erro.

Esse problema é conhecido como *condição de corrida*, e deve ser evitado pelo sistema operacional.

2. EXCLUSÃO MÚTUA

Como vimos, as condições de corrida são nocivas à integridade do sistema, uma vez que podem levar a resultados inconsistentes.

A solução mais simples para evitar que dois ou mais processos acessem um recurso compartilhado e provoquem o erro que vimos é garantir a *exclusão mútua* na utilização desses recursos. De forma simples, a exclusão mútua significa que processos que compartilham recursos só podem usar tais recursos, um de cada vez.

O trecho de código que manipula recursos compartilhado é chamado de *região crítica* e o sistema operacional precisa fornecer protocolos para que as regiões críticas sejam executadas. Esses protocolos são ações que devem ser executadas antes e depois de a execução

de uma região crítica para garantir que os recursos sejam protegidos de erros e não fiquem indefinidamente presos a um processo.

Uma das soluções de *hardware* para isso é a desabilitação das interrupções. Sempre que um processo começar a usar um recurso compartilhado, o sistema operacional impede que ocorram interrupções até que todas as instruções da região crítica tenham sido executadas. Assim, no exemplo que vimos, quando P1 começou a usar a variável A, as interrupções seriam desabilitadas e todas as instruções teriam sido executadas normalmente até o fim, sem o risco de outro processo manipular a mesma variável.

A desabilitação de interrupções é conhecida como um tipo de solução de *hardware*.

Mas a garantia da exclusão mútua também tem soluções de *software* e a mais conhecida delas é a utilização de *semáforos*. Essa solução foi proposta por E. W. Dijkstra em 1965.

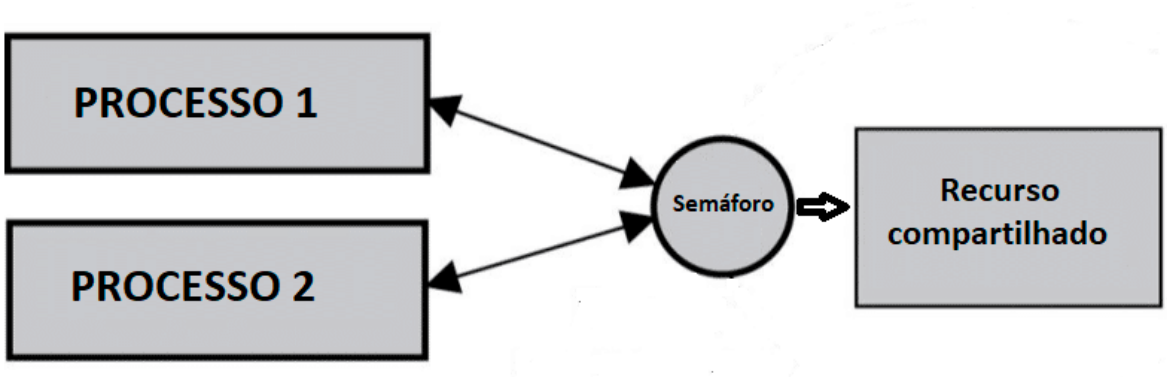
Um semáforo é uma variável inteira e não negativa, sobre a qual só podem ser executadas duas instruções: UP e DOWN. A primeira delas incrementa em uma unidade o semáforo e a segunda o decrementa em uma unidade.

O semáforo é associado a um recurso compartilhado de forma que, quando seu valor for igual a 1, o recurso não está sendo utilizado por nenhum processo e quando for igual a 0, o recurso está em uso.

Seu funcionamento é simples. Quando um processo precisa usar um recurso compartilhado, ele executa a instrução DOWN (protocolo de entrada) no semáforo. Se este estiver valendo 1, ele é decrementado para 0 e o processo pode entrar em sua região crítica. Se ele estiver valendo 0, o processo que executou o DOWN fica impedido de utilizar o recurso e entre em estado de espera, aguardando sua liberação.

Ao terminar de utilizar um recurso, o processo executa a instrução UP (protocolo de saída) e o sistema operacional seleciona um dos processos que está com um DOWN pendente na fila e este pode utilizar o recurso.

FIGURA 2 | **Semáforo para controle de recursos compartilhados**



Elaborado pelo autor.

A importância dos protocolos de entrada e saída das regiões críticas é que, se eles não forem executados corretamente, um ou mais processos poderão ficar aguardando por tempo indefinido a liberação de um recurso e entrar num estado chamado *starvation*.

Por exemplo, suponha que um processo executou a instrução DOWN, utilizou o recurso e não executou a instrução UP ao terminar ou que o sistema operacional desabilitou as interrupções para um processo acessar sua região crítica e não as habilitou novamente ao final. Todos os processos que estiverem aguardando em estado de espera para utilizar tais recursos não conseguirão entrar em execução até que o problema seja resolvido.

3. TROCA DE MENSAGENS

Outra forma de comunicação e sincronização entre processos é a troca de mensagens. Este é um mecanismo que dispensa a existência de variáveis compartilhadas e seu funcionamento se baseia na existência de um canal de comunicação entre os processos que se comunicam e na execução dos comandos SEND e RECEIVE.

É importante que os processos envolvidos na comunicação tenham suas ações sincronizadas, já que não é possível receber uma mensagem que ainda não foi enviada, por exemplo.

A troca de mensagens pode se realizar por *comunicação direta* ou *comunicação indireta*.

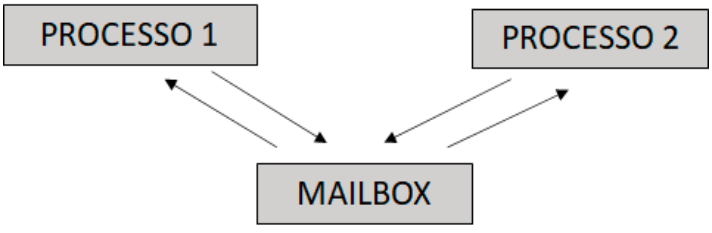
No primeiro caso, os processos envolvidos enderecem de forma explícita o emissor e o receptor e este tipo de comunicação só é possível entre dois processos conforme mostra a Figura 3.

FIGURA 3 | **Comunicação direta**

Elaborado pelo autor.

No caso da comunicação indireta, os processos utilizam uma área compartilhada, chamada *mailbox* ou *port*, cujas características, como, por exemplo, o tamanho, são definidas no momento de sua criação. Neste caso, os parâmetros de endereçamento não se referem mais a processos e, sim, a *mailboxes*. A Figura 4 ilustra a comunicação indireta entre processos.

FIGURA 4 | **Comunicação indireta**



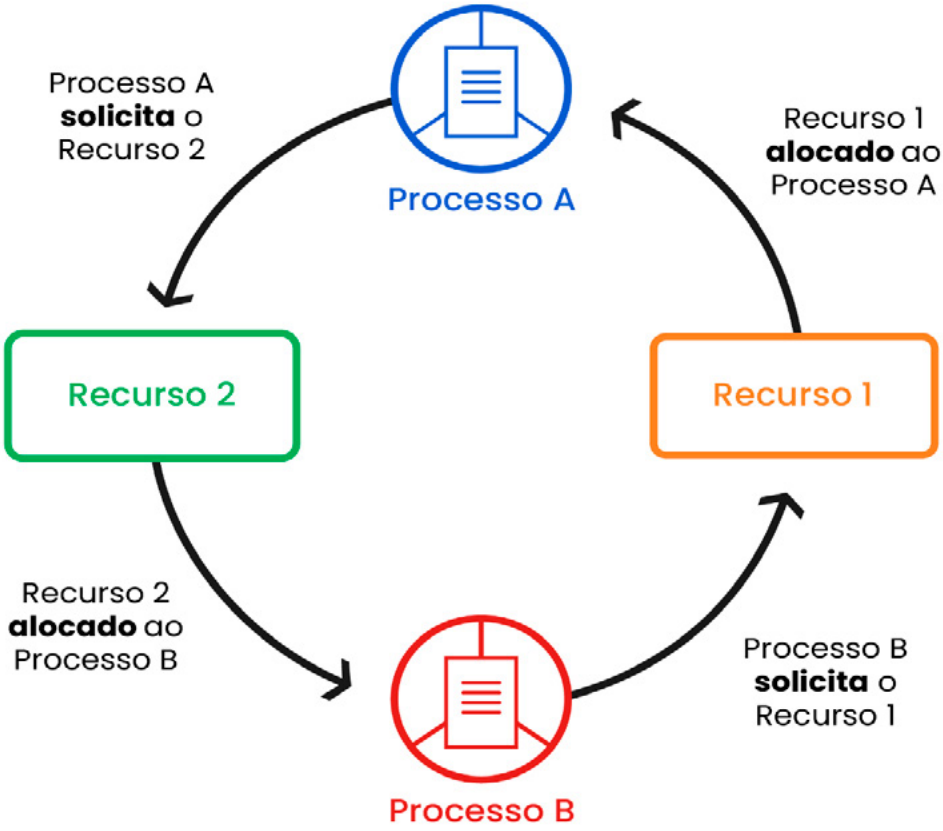
Elaborado pelo autor.

4. DEADLOCK

Deadlock é uma situação em que os processos aguardam por um recurso que não está disponível ou por um evento que nunca ocorrerá.

A tradução da palavra deadlock é *impasse* e isso nos ajuda a começar a entender o que é isso e suas consequências.

FIGURA 5 | **Deadlock**



Fonte: https://saulo.arisa.com.br/wiki/index.php/Comunica%C3%A7%C3%A3º_entre_Processos)

A Figura 5 mostra um deadlock, numa situação conhecida como *espera circular*. Como podemos ver, o processo A está com o recurso 1, mas precisa também do recurso 2 para entrar em execução. Da mesma forma, o processo B está com o recurso 2 e também precisa do recurso 1 para entrar em execução. Dessa forma, nenhum dos dois consegue prosseguir, o que caracteriza um deadlock que, se persistir, poderá envolver outros processos e, num caso mais drástico, causar o travamento do computador.

Há quatro condições que podem levar aos deadlocks:

1) Exclusão mútua: um recurso só pode estar alocado a um processo em um dado momento;

2) Espera por um recurso: um processo pode estar com a posse de alguns recursos e precisar de outros para poder entrar em execução;

3) Não preempção: não liberar um dos recursos de um processo porque outros processos precisam dele;

4) Espera circular: como mostrado na Figura 5, um processo pode estar esperando um recurso alocado a outro processo e vice-versa.

Como gerente de recursos, o sistema operacional precisa atuar da seguinte forma quanto aos deadlocks:

- Prevenção de deadlocks: o sistema operacional deverá garantir que as condições para a ocorrência de deadlocks não sejam satisfeitas;
- Detecção de deadlocks: o sistema operacional deverá ser capaz de perceber a ocorrência de um deadlock o mais rápido possível;
- Diagnóstico de deadlocks: o sistema operacional deverá descobrir a razão da ocorrência do deadlock;
- Solução do deadlock: o sistema operacional deverá tomar alguma providência para desfazer o deadlock. Isso implica em executar algoritmos de *escolha de vítima*, que consistem em sacrificar um dos processos envolvidos no deadlock para que os outros possam prosseguir.

É importante observar que a eficiência do sistema operacional na atuação quanto aos deadlocks podem tornar esse problema, na maioria das vezes, imperceptível ao usuário.

CONSIDERAÇÕES FINAIS

Chegamos ao final de mais uma aula em que vimos a importância das funções de sincronização e comunicação entre processos executada pelo sistema operacional.

Muitos processos compartilham recursos e trocam informações e isso precisa ser rigorosamente controlado para evitar inconsistências na execução dos programas.

Eventos como condições de corrida e deadlocks devem ser evitados pelo sistema operacional porque afetam resultados e eventualmente travam o computador e, portanto, é necessário que o sistema operacional aja preventivamente.

MATERIAIS COMPLEMENTARES

Neste vídeo você poderá aprender um pouco mais sobre deadlocks e seu tratamento. Disponível: https://www.youtube.com/watch?v=SuqbMSnNSEk.

Assista a esse vídeo bastante ilustrativo quanto às condições de corrida. Disponível em: https://www.youtube.com/watch?v=D1S6MlH1l0U.

REFERÊNCIAS

STALLINGS, William. *Arquitetura e Organização de Computadores: projeto para o desempenho*. 8ª edição. Editora Pearson. Livro. (642 p.). ISBN 9788576055648. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576055648>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª edição. Editora Pearson. Livro. (674 p.). ISBN 9788576052371. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788576052371>. Acesso em: 16 out. 2022.

TANENBAUM, Andrew S. *Organização estruturada de computadores*. 6ª edição. Editora Pearson. Livro. (628 p.). ISBN 9788581435398. Disponível em: <https://middleware-bv.am4.com.br/SSO/iesb/9788581435398>. Acesso em: 16 out. 2022.