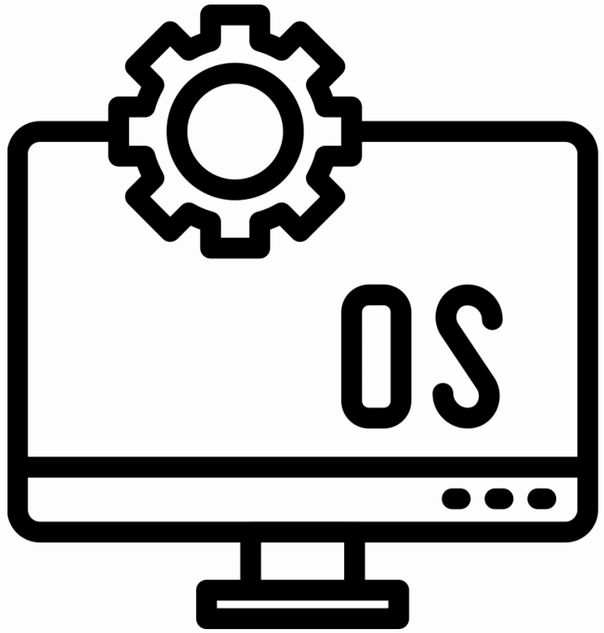
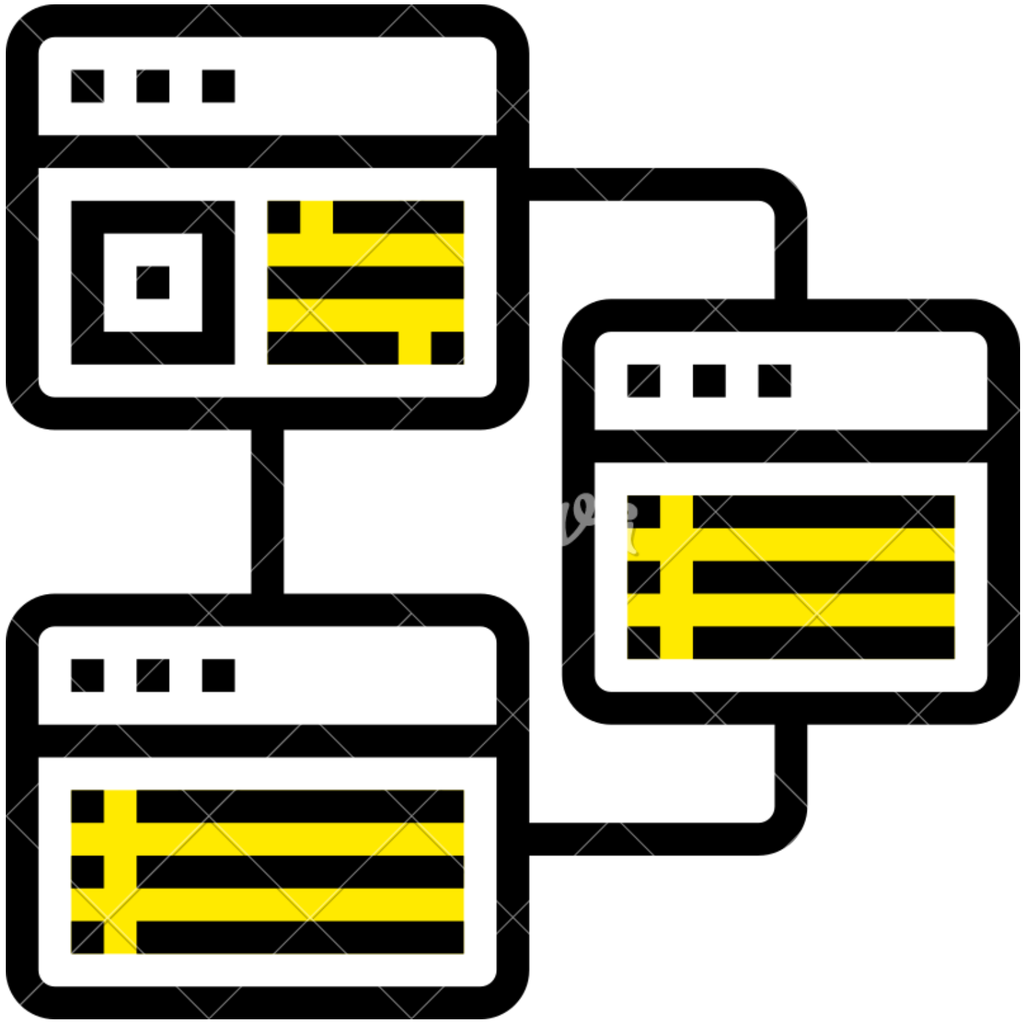


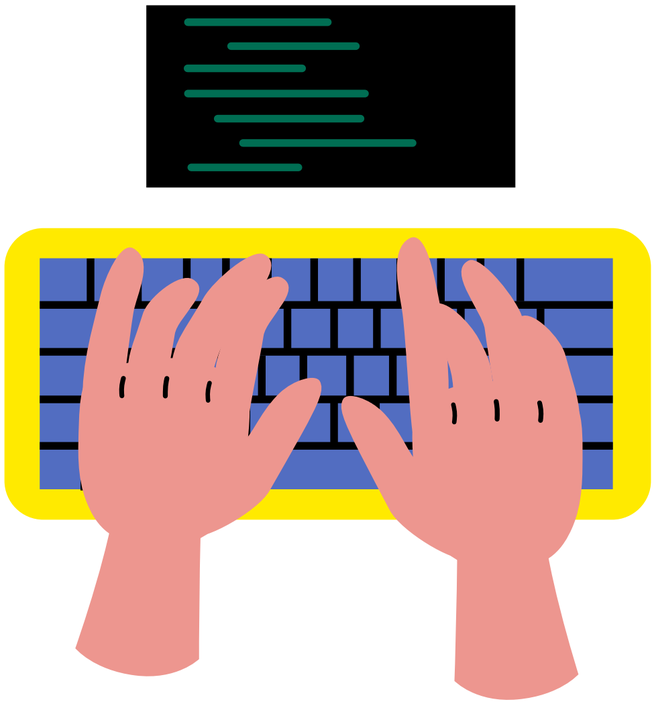
SINCRONIZAÇÃO CONDICIONAL

UTILIZANDO MONITORES

O que são Monitores?

Programas ou componentes que monitoram e controlam atividades do sistema, como monitores de desempenho que acompanham o uso da CPU, memória e outros recursos.



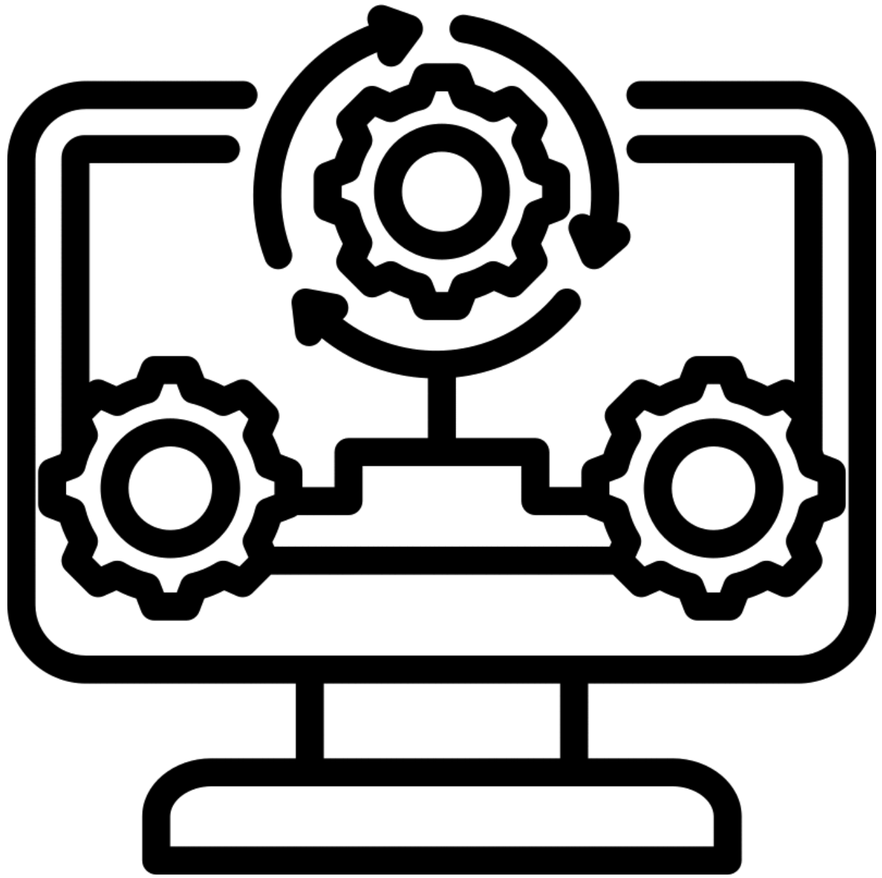


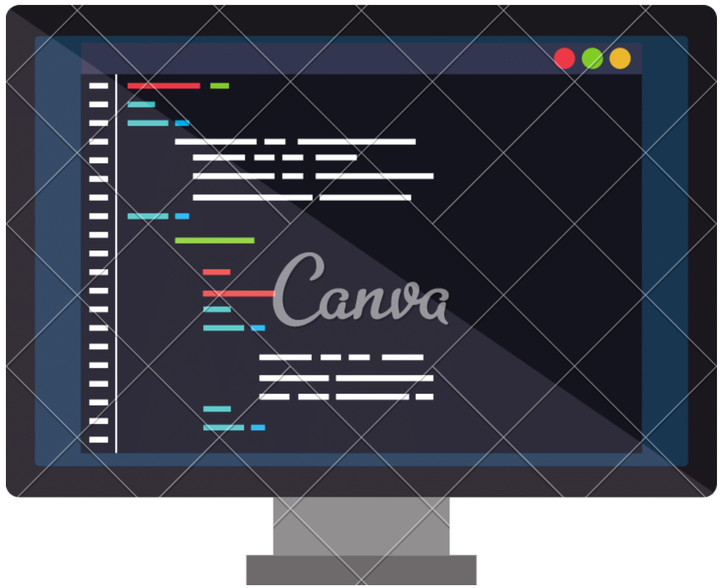
O que é Sincronização Condicional?

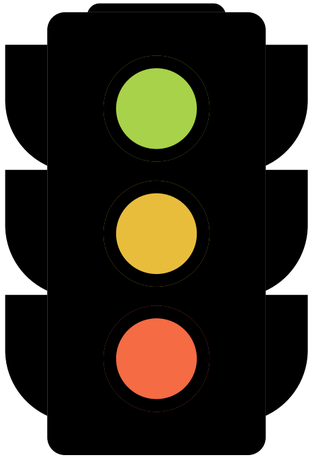
Organizar tarefas com base em eventos específicos, garantindo que algumas ações aconteçam apenas quando as condições certas acontecerem. Isso é usado para manter a ordem e a consistência em situações complicadas.

O que é Sincronização Condicional utilizando Monitores?

Usando "monitores" para garantir que apenas uma parte do código (ou "thread") mexa em uma coisa compartilhada por vez. Os monitores ajudam a evitar confusão e problemas quando várias partes do programa estão tentando usar algo ao mesmo tempo. Além disso, eles fornecem um jeito de dizer a outras partes do programa quando algo mudou, para que possam agir de acordo.





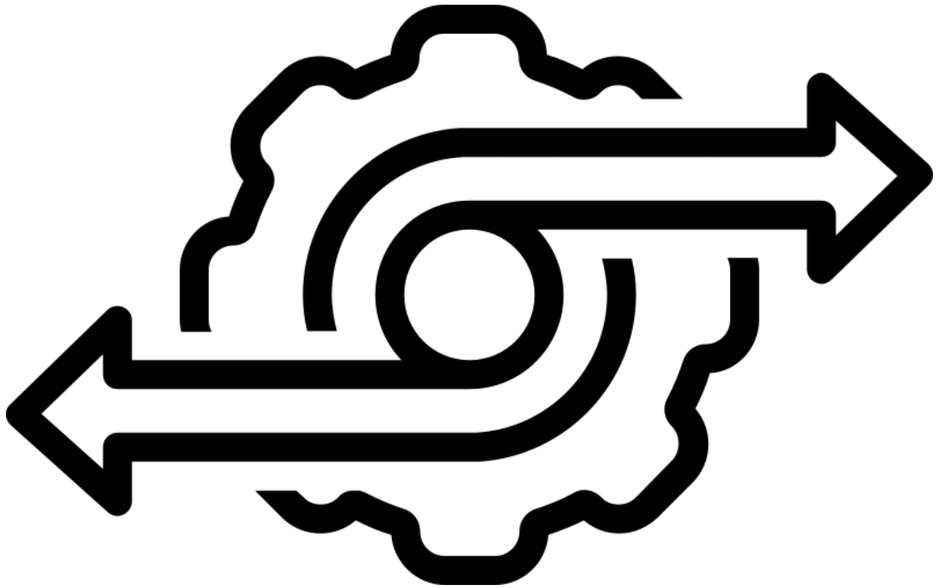


Diferença entre Monitores e Semáforos

- Nível de Abstração
- Escopo de Controle
- Comunicação entre Threads
- Uso de Recursos
- Facilidade de Uso

Qual escolher???

A escolha entre monitores e semáforos dependerá das necessidades específicas do programa e da complexidade desejada na implementação da sincronização concorrente. Em muitos casos, os monitores são preferidos por sua abstração mais elevada e pela facilidade de uso.



Exemplo de situações que necessitam usar sincronização condicional utilizando monitores

A sincronização condicional utilizando monitores é necessária em situações em que as threads precisam aguardar uma condição específica antes de prosseguir. Um exemplo clássico é o problema produtor-consumidor, onde threads produtores colocam dados em um buffer compartilhado e as threads consumidores retiram esses dados. O uso de monitores com sincronização condicional ajuda a evitar condições de corrida e garante que as threads aguardem com eficiência economizando recursos.

Possíveis problemas ao lidar com sincronização condicional utilizando monitores

- Deadlocks
- Overhead de Sincronização
- Desempenho Ineficiente
- Complexidade de Código

Possível resolução

1. Wait:
- **Função:** Utilizado para fazer uma thread aguardar até que uma condição específica seja atendida.
- **Cenário de Uso:** Quando uma thread precisa esperar por uma condição antes de prosseguir, `Wait` é empregado para suspender temporariamente a execução da thread até que a condição seja satisfeita por outra thread.

2. Signal:
- **Função:** Usado para notificar uma ou mais threads que estão esperando que uma condição seja atendida.
- **Cenário de Uso:** Quando uma thread atende a condição desejada, `Signal` é acionado para notificar outras threads que estão em estado de espera (`Wait`). Isso permite que elas continuem sua execução.

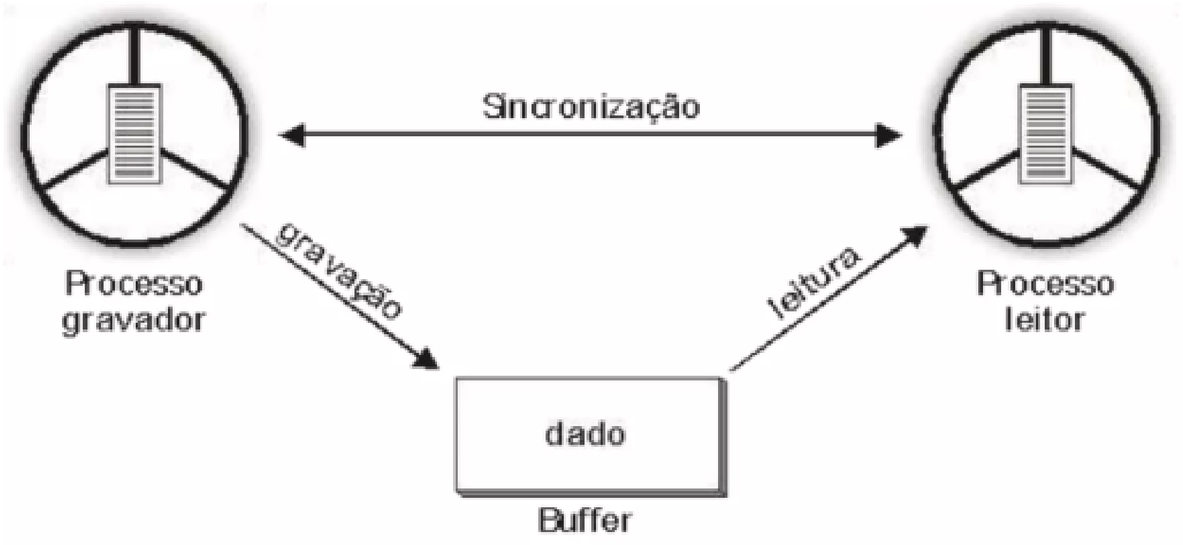
Importância na Evitação de Problemas

Deadlocks: O uso adequado de `Signal` e `Wait` pode ajudar a evitar situações de deadlock, onde as threads ficam permanentemente bloqueadas umas esperando pelas outras.

Race Conditions: Essas estratégias contribuem para uma sincronização mais precisa, reduzindo o risco de race conditions, onde múltiplas threads competem por recursos simultaneamente.

Como funciona a sincronização condicional utilizando monitores

- Monitorização de Recursos Compartilhados
- Condições de Entrada
- Notificações



Exemplo

Produz - Quando o buffer estiver cheio o procedimento executa um wait sobre a variável condicional Cheio, bloqueando o processo.
O processo bloqueado poderá continuar após o Consumidor executar um Signal na variável Cheio, liberando espaço no buffer.

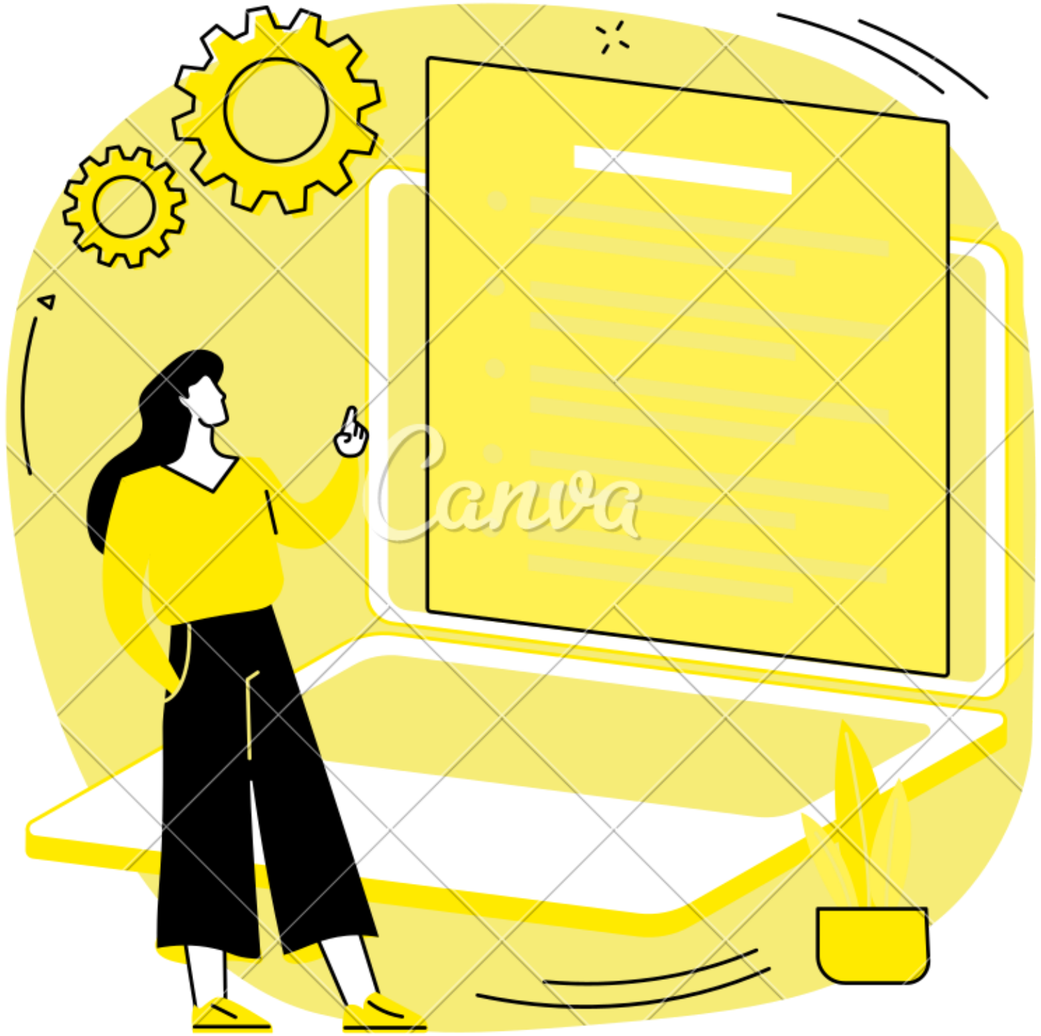
Consome - Acorda os processos adormecidos a partir de um Signal na váriável condicional Cheio, liberando apenas um processo.
Caso o buffer esteja vazio, o procedimento deve executar um Wait em Vazio indicando que não há mais dados a serem consumidos.

Exemplos de projetos ou sistemas que se beneficiam de sincronização condicional utilizando monitores

- Sistemas de Controle de Estoque
- Simulações em Ambientes Multithread
- Bancos de Dados Multithread

Orientações sobre otimizações e boas práticas

- Minimizar Regiões Críticas
- Evitar Aninhamento de Monitores
- Usar Estruturas de Dados Atômicas
- Priorizar Semáforos e Mutexes Leves
- Testar e Perfilar



Conclusão

Nesta aula, exploramos os fundamentos da sincronização condicional e sua implementação utilizando monitores. Compreendemos que a programação concorrente apresenta desafios únicos, e a sincronização condicional desempenha um papel crucial na criação de sistemas seguros e eficientes.

Os monitores emergem como uma ferramenta valiosa nesse contexto, proporcionando uma abstração poderosa ao encapsular variáveis compartilhadas e os procedimentos que operam sobre elas. Através do uso de mecanismos como Signal e Wait, conseguimos coordenar a execução de threads, evitando problemas comuns como deadlocks e race conditions.

Agradeço por sua atenção nesta aula. Não hesite em explorar mais, questionar e aplicar esses conhecimentos em seus próprios projetos. Boa sorte em suas futuras explorações na vasta paisagem da programação concorrente!

Alunos

Emelly Yasmin Andrade Formigário
Gabriélly Custódio Ferreira
Marco Antonio Alcindo Gitti

Referências

Sistemas Operacionais Modernos - TANENBAUM
Apresentação Semáforos, monitores, troca de mensagens, Deadlock