# PROJETO

Este trabalho envolve programação concorrente. O objetivo do trabalho é familiarizar os membros com os conceitos de programação usando threads e controle de concorrência usando semáforos.

Beatriz Emily Silva Aguiar - 22154303
Matheus Souza de Oliveira - 22153438
Natanael Bezerra de Oliveira - 22152258

# Leitores x Escritores

Crie uma variável (ex: um registro com dados, uma variável com um valor numérico, etc), que será compartilhada por várias threads, algumas do tipo leitor e outras do tipo escritor. As threads leitoras devem exibir os dados lidos e as threads escritoras devem atualizar e exibir os dados após a atualização. O trabalho pode ter um “tema” como o problema do banco (operações de consulta ao saldo, saque, depósito), portal do aluno (consulta às notas, lançamento de notas), banco de dados de clientes de uma loja (inserção, remoção, consulta de dados de compras), etc.

Implemente três versões:

1. Leitores e escritores com mesma “prioridade”. Pode ocorrer “leitura suja”.

2. Escritores com prioridade sobre leitores. Nesse caso, leitores somente têm acesso aos dados quando não há nenhum escritor ativo. Não ocorre “leitura suja”.

3. Versão sem controle de concorrência para mostrar o problema do acesso simultâneo aos dados compartilhados (somente threads do tipo escritoras).

O programa deve receber como entrada a definição dos processos (ex: via teclado, arquivo de entrada ou gerados aleatoriamente) com a quantidade de threads leitoras e escritoras e os valores que serão atualizados pelos escritores.

A saída do programa deve ilustrar o funcionamento do sistema, mostrando quando cada processo é criado, quando eles entram e saem da região crítica, quando eles são bloqueados, seu tipo (leitor ou escritor), o que fizeram (mostrar os valores dos dados compartilhados) e o momento em que ele é finalizado.

**Project Name:** Leitores e escritores com mesma “prioridade”.

**Date:** Jun 19, 2024

| Objective | Metrics |
| --- | --- |
| Simular a leitura e escrita de um registro compartilhado entre múltiplas threads utilizando mutexes e variáveis de condição para sincronização. | Leitor: lê a nota do registro, se há escritores ativos o leitor espera. Se há escritores ativos, espera até que eles terminem. |
| | Escritor: lê a nota, atualiza a nota e escreve de volta no registro. Se há leitores ativos, espera até que eles terminem. |
| | Main: Cria threads leitoras e escritoras de forma alternada ou conforme as condições, para que metade sejam leitoras e a outra metade escritoras. |

## Análise dos Resultados

- **Leitores e Escritores com Mesma Prioridade:** Leitores e escritores têm a mesma prioridade e esperam uns pelos outros conforme necessário.
- **Leituras Sujas:** Como não há prioridade clara, podem ocorrer leituras enquanto escritores estão escrevendo, resultando em leituras inconsistentes. O uso de ***usleep*** para introduzir atrasos aleatórios, é possível observar leitura suja, onde leitores podem ler valores intermediários não finalizados por escritores.
- **Semáforo para Escritores:** Garante que apenas um escritor pode modificar a nota de cada vez, prevenindo escrita simultânea.

**Project Name:** Escritores com prioridade sobre leitores.

**Date:** Jun 20, 2024

| Objective | Metrics |
| --- | --- |
| Simular a leitura e escrita de um registro compartilhado entre múltiplas threads utilizando mutexes e variáveis de condição para sincronização. | **Leitor**: A thread leitora adquire o mutex e verifica se há escritores ativos. Se houver, ela espera na variável condicional withoutWriter. Somente leem após os escritores terminarem. A thread libera o mutex e termina. |
| | Escritor: A thread escritora adquire o mutex, decrementa o contador de escritores e libera o mutex. A thread escritora espera pelo semáforo, lê, atualiza e escreve a nota no registro, e libera o semáforo. A thread escritora adquire o mutex novamente, incrementa o contador de escritores e sinaliza a condição 'withoutWriter', notificando os leitores que não há mais escritores ativos. |
| | Main: Cria threads leitoras e escritoras de forma alternada ou conforme as condições, para que metade sejam leitoras e a outra metade escritoras. |

## Análise dos Resultados

- **Semáforo e Mutex:** O semáforo writer garante exclusividade na escrita, impedindo que múltiplos escritores modifiquem a nota ao mesmo tempo. O mutex mutex protege o acesso ao contador de escritores e sincroniza a espera dos leitores.
- **Leitores e Escritores:** O uso de um contador de escritores (writers) e uma variável condicional (withoutWriter) dá prioridade aos escritores. Os leitores esperam até que não haja escritores ativos.
- **Prioridade dos Escritores:** Escritores têm prioridade sobre leitores. Leitores só podem proceder quando não há escritores ativos, o que é garantido pelo controle do contador writers e a variável condicional withoutWriter.

**Project Name:** Sem controle de Concorrência.

**Date:** Jun 20, 2024

| Objective | Metrics |
| --- | --- |
| Removeu-se o controle de threads, o que gerou situação de concorrência. | Escritor: lê a nota, atualiza a nota e escreve de volta no registro. |
| | Main: Cria as threads alternando entre funções de leitura e escrita. |

### Análise dos Resultados

- **Condições de Corrida:** Sem mecanismos de sincronização, múltiplas threads podem acessar e modificar ***registro.nota*** simultaneamente, levando a condições de corrida.
- **Inconsistência de Dados :** A nota final pode não ser a esperada devido à sobrescrita das atualizações pelas threads.

# Produtores x Consumidores

Considere um buffer compartilhado, onde processos do tipo produtor enviam dados para o buffer colocando os dados em ordem e processos do tipo consumidor retiram dados deste mesmo buffer também em ordem.

Há duas situações básicas de sincronização a considerar:

- Se o buffer estiver vazio, os processos consumidores vão “dormir”, ou seja, ficam bloqueados aguardando um sinal para avisar que um dado foi colocado no buffer.
- Caso o buffer fique cheio, os processos produtores devem ser bloqueados, aguardando um sinal para avisar que um dado foi retirado do buffer.

Desenvolva um programa que faça a coordenação dos processos produtores e consumidores. O programa deverá mostrar o “status” de cada processo, por exemplo, “processo 1 produzindo”, “processo 2 dormindo”, “processo 3 consumindo”, etc. Implemente os processos como threads. O buffer pode ser uma estrutura do tipo vetor, que será compartilhada por todas as threads (produtores e consumidores). Deve-se exibir o buffer com os valores de momento

Implemente três versões:

1. Uma versão com vários processos produtores e 1 consumidor;
2. Versão com vários processos produtores e vários consumidores;
3. Versão sem controle de concorrência (mostrar onde e quando ocorre o problema de atualização do buffer).

**Project Name:** Vários processos produtores e 1 consumidor.

**Date:** Jun 15, 2024

| Objective | Metrics |
| --- | --- |
| Simular a leitura e escrita de um registro compartilhado entre múltiplas threads utilizando mutexes e variáveis de condição para sincronização. | Producer: espera até que haja espaço no buffer, acessa a região crítica e espera que esteja disponível produz um item, libera a região crítica e sinaliza que o buffer está cheio. |
| | Consumer: Espera até que haja um item no buffer, aguarda que a região crítica esteja disponível, consome um item, libera a região crítica e sinaliza que há espaço disponível no buffer. |
| | Main: Cria threads para os produtores e consumidores. |

## Análise dos Resultados

- **Produtores e Consumidores:** O código garante que produtores esperem por espaço no buffer antes de produzir e os consumidores, por sua vez, esperem por itens no buffer antes de consumir.

**Project Name:** Vários processos produtores e vários consumidores.

**Date:** Jun 15, 2024

| Objective | Metrics |
| --- | --- |
| Simular produtores e consumidores utilizando semáforos para sincronização. | Producer: O produtor deve produzir itens até que o buffer esteja cheio, então para isso, espera até que haja espaço disponível no buffer, espera para entrar na região crítica, produz um item e insere no buffer, atualiza o tamanho da fila, libera a região crítica, incrementa o semáforo e sinaliza que um novo item está disponível para consumir. |
| | Consumer: Espera até que haja um item disponível para consumo, aguarda para entrar na região crítica, consume um item do buffer, atualiza o tamanho da fila, libera a região crítica, incrementa o semáforo e sinaliza que há um novo espaço disponível no buffer. |
| | Main: Cria threads para os produtores e consumidores. |

## Análise dos Resultados

- **Produção e Consumo Balanceados:** O número de itens produzidos deve ser igual ao número de itens consumidos, resultando em uma fila vazia no final da execução.
- **Execução Concorrente:** As threads produtoras e consumidoras executam concorrentemente, simulando um ambiente real de produção e consumo.

**Project Name:** Sem controle de concorrência. **Date:** Jun 15, 2024

| Objective | Metrics |
| --- | --- |
| implementação do problema dos produtores e consumidores utilizando threads, mas **sem sincronização** | Producer: O produtor deve produzir itens até que o buffer esteja cheio, então para isso, espera até que haja espaço disponível no buffer, espera para entrar na região crítica, produz um item e insere no buffer, atualiza o tamanho da fila, libera a região crítica, incrementa o semáforo e sinaliza que um novo item está disponível para consumir. |
| | Consumer: Espera até que haja um item disponível para consumo, aguarda para entrar na região crítica, consume um item do buffer, atualiza o tamanho da fila, libera a região crítica, incrementa o semáforo e sinaliza que há um novo espaço disponível no buffer. |
| | Main: Cria threads para os produtores e consumidores. |

## Análise dos Resultados

- **Concorrência:** Sem mecanismos de sincronização, múltiplas threads podem acessar e modificar a fila simultaneamente, levando a condições de corrida.

- **Acesso Simultâneo:** threads produtoras e consumidoras podem acessar simultaneamente a mesma posição no buffer.

# Referências


- Aula 15 - Locks
- Aula 16 - Variáveis de Condição -
- Aula 18 - Semáforos e problemas clássicos de controle de concorrência
- Aula 8 - Controle de Concorrência entre Processos
- Usando Mutexes
- Códigos Auxiliares