UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO

SISTEMAS OPERACIONAIS
PROF. JOÃO MARCOS BASTOS CAVALCANTI

Trabalho Prático 1

*** Data de entrega: até 10/10/2025 ***

Este trabalho prático envolve programação concorrente. O objetivo do trabalho é familiarizar os estudantes com os conceitos de programação usando threads e o controle de concorrência usando semáforos. Considere no texto abaixo os termos processo e *thread* como equivalentes.

1. Processos Leitores × 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 sem "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 (naturalmente o problema ocorre somente com *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.

2. Processos Produtores × 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*).

OBSERVAÇÕES SOBRE AS IMPLEMENTAÇÕES:

- A implementação deve obrigatoriamente utilizar semáforos em sua solução ou mecanismos equivalentes como *mutex, lock/unlock*. A escolha do mecanismo feita pela equipe faz parte da avaliação.
- Sugerimos a utilização das linguagens C ou C++, por meio das bibliotecas "pthread" e "semaphore".
- Deve-se utilizar atrasos, por exemplo usando a função *sleep*, para deixar as *threads* com diferentes velocidades. Isso torna os resultados bastante variados e mais interessantes. Em geral as *threads* escritores são mais lentas (maior tempo de *sleep*) que os leitores. Produtores e consumidores podem ter velocidades bastante variadas. Deve ser possível testar qualquer configuração de atraso para qualquer tipo de *thread*.

OBSERVAÇÕES SOBRE A ENTREGA DO TRABALHO:

- O trabalho pode ser feito em equipe de no máximo 3 (três) estudantes.
- O trabalho deverá ser relatado em um documento com comentários gerais sobre as escolhas da equipe para a implementação e exemplos de execução para demonstrar o funcionamento do programa para cada questão. É importante detalhar qualquer ponto que a equipe julgar necessário para contribuir com a avaliação do trabalho.
- Os arquivos com código fonte dos programas e o relatório deverão ser entregues pelo ColabWeb com a identificação da equipe (basta um membro da equipe submeter o trabalho no ColabWeb).
- Deve-se registrar a(s) fonte(s), seja IA assistente, websites, livros, slides, etc., de onde foram obtidas descrições das soluções e/ou códigos fonte para o desenvolvimento do trabalho.