# Sistemas Operacionais
# 2023-2 TP3

cfmcc@ufop.edu.br
Prof. Dr. Carlos Frederico MC Cavalcanti,

"Produtor e consumidor v2"

É importante entender a necessidade de sincronização na execução concorrente de código. Threads e processos são duas formas de implementar a concorrência.

Veja  o exemplo clássico do problema do produtor-consumidor que estudamos e está no capitulo 2 do livro  "Sistemas Operacionais Modernos" de Andrew S. Tanenbaum. Neste problema, há um buffer limitado que é compartilhado por um produtor e um consumidor. O produtor coloca itens no buffer e o consumidor retira os itens para processá-los. No entanto, o produtor e o consumidor devem sincronizar suas ações para evitar condições de corrida.

O projeto do “produtor” e do  “consumidor” precisa  evitar problemas como condições de corrida (race condition), buffer cheio ou buffer vazio. Eles também devem ser capazes de implementar a solução utilizando semáforos ou outras estruturas de sincronização adequadas.

Este TP é para implementar “na prática” o produtor e  consumidor esboçados no livro do Tanenbaum. No livro, o produtor gera um inteiro e o buffer é limitado a 100. Antes de iniciar o programa vc indagar, via prompt (linha de comando) quantos inteiros por segundo serão gerados e o consumidor deve “imprimir” quantos inteiros foram consumidor por segundo.

Sugiro a seguinte abordagem (se vc fizer em C++, mas pode fazer em C):

Crie uma classe chamada Buffer com os seguintes métodos:

> produzir(item): adiciona um item ao buffer. Se o buffer estiver cheio, o produtor deve esperar até que haja espaço disponível.
>
> consumir(): retira um item do buffer. Se o buffer estiver vazio, o consumidor deve esperar até que haja um item disponível.

Implemente a classe Buffer de forma que ela utilize semáforos ou outras estruturas de sincronização adequadas para garantir a exclusão mútua e a sincronização entre o produtor e o consumidor.

Crie uma classe chamada Produtor que será responsável por produzir os itens a serem colocados no

buffer. Essa classe deve herdar da classe Thread e implementar um método `run()`.

Crie uma classe chamada Consumidor que será responsável por consumir os itens do buffer. Essa classe também deve herdar da classe Thread e implementar um método `run()`.

Na classe principal (por exemplo, `main`), crie uma instância do buffer, do produtor e do consumidor.

No método `run()` do produtor, produza uma sequência de itens e os coloque no buffer usando o método produzir(item).

No método `run()` do consumidor, consuma os itens do buffer usando o método consumir().

Execute o programa e observe a sincronização entre o produtor e o consumidor.

Teste diferentes cenários, aumentando a "velocidade de produção" e veja se o "consumidor responde", assim como um produtor produzindo mais itens do que o buffer pode armazenar ou um consumidor tentando consumir um item de um buffer vazio.

É importante entender a necessidade de sincronização entre o produtor e o consumidor para evitar problemas como condições de corrida, buffer cheio ou buffer vazio. Eles também devem ser capazes de implementar a solução utilizando semáforos ou outras estruturas de sincronização adequadas.

Entregáveis:

1- Imagem do programa no Docker HUb

2- Vídeo explicando o que vc fez em até 3 minutos.

Alguns links interessantes

**Sistemas Operacionais: Conceitos e Mecanismos © Carlos Maziero, 2019 – CAP 11**

https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-11.pdf

**Sistemas Operacionais: Conceitos e Mecanismos © Carlos Maziero, 2019 -CAP 10**

https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=socm:socm-10.pdf

**Novos recursos de simultaneidade no Visual C++ 11**

https://learn.microsoft.com/pt-br/archive/msdn-magazine/2012/march/c-new-standard-concurr

ency-features-in-visual-c-11

Pattern Singleton em C e Threads

https://medium.com/c-c/pattern-singleton-em-c-e-threads-76bf9298cfb6