Mangabits

Algoritmo Produtor-Consumidor

Paticipantes: Eduardo Sousa Passos, Iago Suzart Silva, Iuri Santana Góes da Silva, Maria Eduarda Kassianney da Silva, Otávio dos Santos Souza e Raian Oliveira da Cruz.

Indice

- O Problema Produtor-Consumidor
- Biblioteca Concurrent.Semaphore
  - Mutex
  - Empty
  - Full
  - Acquire e Release (Mutex, Empty, Full)
- O Console
  - Item
  - In/Out
  - VarBuffer
  - SizeBuffer
  - ItemCounter
- Apresentação prática do algoritmo em funcionamento
- Conclusão

Mangabits

Mangabits

Introdução

O algoritmo produtor-consumidor é uma problemática clássica de sincronização de processos. Ele exemplifica como a comunicação e a cooperação entre threads ou processos trabalham juntas em um ambiente compartilhado.

Neste, o "produtor" é responsável por gerar dados ou itens, enquanto o "consumidor" é encarregado de consumí-los.

Os problemas:

- O produtor não pode adicionar dados quando o buffer está cheio.
- O consumidor não pode remover dados quando o buffer está vazio.
- Os dois processos não podem se interromper.

Objetivo

Construir um algoritmo, no qual, dois processos/threads - produtor e consumidor - tenham acesso a um Buffer de forma sincronizada, sem interromperem-se e sem excedência dos limites dele.

Biblioteca

Biblioteca utilizada para controlar o acesso de processos que compartilham do mesmo recurso, o Buffer, no caso do algoritmo Produtor/Consumidor. Limitando o número de processos que pode acessar aquela região crítica

Semaphores são geralmente usados para limitar o número de threads que podem acessar um recurso físico ou lógico.

Biblioteca

Um semáforo com contador. Conceitualmente, um semáforo tem set de "permits". O Semaphore apenas mantém uma contagem do número disponível e age de acordo a ele.
explicações:

mutex

- **Definição:** Implementa a exclusão mútua garantindo que apenas uma única thread ou processo possa acessar um recurso compartilhado por vez.

- **Inicialização (construtor):** Inicializando com 1 informando que um processo pode acessar a região criticar

Biblioteca

empty

- **Definição:** Quando chamado, controla o acesso da thread produtora quando o buffer estiver cheio. Quando uma thread produtora deseja produzir um item no buffer, ela adquire um permit usando empty.acquire(). Se o buffer estiver cheio, a thread produtora será bloqueada até que uma thread consumidora libere espaço no buffer e aumente a cota do semáforo empty.

- **Inicialização (Construtor):** Inicializando com o tamanho do buffer, informando que o produtor pode acessar a região crítica, quando o empty chega em zero bloqueia o produtor de acessar a região crítica.

Biblioteca

full

- **Definição:** O full é a variável do tipo Semaphore responsável por controlar a thread Consumidora, não a deixando consumir o Buffer vazio.

- **Inicialização (Construtor):** Ele inicializa com 0 pois o full representa a contagem de espaços cheios. Se, durante o processo, o contador do full estiver maior que zero, significa que há mais de zero processos disponíveis para consumo, e o full.acquire() receberá um permit, liberando o Thread consumidor.

Biblioteca

acquire();

- **DEFINIÇÃO:** Cada acquire(), se necessário, bloqueia-se até que o "permit" esteja disponível, e então, o recebe.

release();

- **DEFINIÇÃO:** Cada release(), incrementa um permit, potencialmente liberando um acquire() bloqueado. Contanto, nenhum objeto de tipo permit são utilizados.

BIBLIOTECA

Console

item

- **Função no código:** Armazena o valor gerado entre 1-9 a ser inserido pelo Produtor ou removido pelo Consumidor no Buffer.

- **Função no console:** Exibir o valor do item que está sendo inserido ou removido do Buffer

varBuffer

- **Função no código:** o buffer em si, o array que armazena os dados gerados e removidos.

- **Função no console:** Na sessão status do Buffer, serão impressas, ordinalmente, os dados dentro do buffer no loop atual do código;

CONSOLE

Console

in/out

- **Função no código:**
  - **in -** No método produce(), ele irá alocar o dado no Buffer na posição correspondente ao número inteiro, gerado de forma aleatória, armazenado na variável IN.
  - **out -** No método consume(), irá retirar o dado do Buffer na posição correspondente ao número inteiro, gerado de forma aleatória, armazenado na variável OUT.

- **Função no console:** Exibe a posição no qual o dado foi inserido (in - método “produce()”) ou removido (out - método “consume()”);

CONSOLE

Console

sizeBuffer

- **Função no código:** Armazena, utilizando o construtor, o tamanho do buffer no momento em que o objeto é instanciado;

- **Função no console:** Exibir o tamanho máximo do buffer;

itemCounter

- **Função no código:** Conta e armazena a quantidade de itens no buffer, através de uma estrutura de condição no for que percorre o Buffer.

- **Função no console:** Exibir a quantidade de espaços ocupados do buffer;

CONSOLE

Console

```
Produziu: 2 - Posição 4  |  Buffer status: [ 0 2 5 2 0 ]  |  Ocupação: 3/5
  Consumiu: 2 - Posição 4  |  Buffer status: [ 0 2 5 0 0 ]  |  Ocupação: 2/5
```

CONSOLE


Dúvidas?