Programação Concorrente - Problema do Boliche Manual

Eduardo Marciano de Melo Meneses - 211055227

30 de Agosto de 2024

1 Introdução

Este relatório apresenta a formalização, descrição e resolução de um problema de programação conconrrente, problema do boliche Manual, bem como a sua devida resolução utilizando técnicas de sincronização entre várias threads, as quais possuem mémoria compartilhada.

O algoritmo foi desenvolvido em C++ e utiliza diversas ferramentas de sincronização, como semáforos, variáveis de condição e mutexes, para simular o funcionamento de um boliche com pistas quebradas. Um funcionário, representado por uma thread separada, é responsável por levantar os pinos nas pistas quando necessário, respondendo às solicitações dos clientes.

2 Formalização do Problema Proposto

O problema do Boliche Manual é uma simulação do funcionamento de um boliche, onde as máquinas de reposição de pinos estão inoperantes em N pistas, e um funcionário é responsável por levantar os pinos sempre que forem derrubados.

Adicionalmente, o problema foi formulado para maximizar as condições de concorrência. Para isso, o problema foi expandido com os seguintes elementos de concorrência:

Inicialmente, N clientes tentam entrar no estabelecimento, porém há apenas K cadeiras de espera, sendo $K < N$. Portanto, antes de poderem entrar no estabelecimento, os clientes competem por um lugar na fila de espera. Caso não consigam uma cadeira, os clientes desistem de utilizar o estabelecimento.

Após conseguirem uma cadeira de espera, os clientes tentarão acessar uma das N pistas de boliche, onde cada pista pode ser utilizada simultaneamente por até dois clientes. Se um cliente não conseguir uma pista, ele deverá aguardar em sua cadeira de espera até que uma pista fique disponível, sem utilizar espera ocupada.

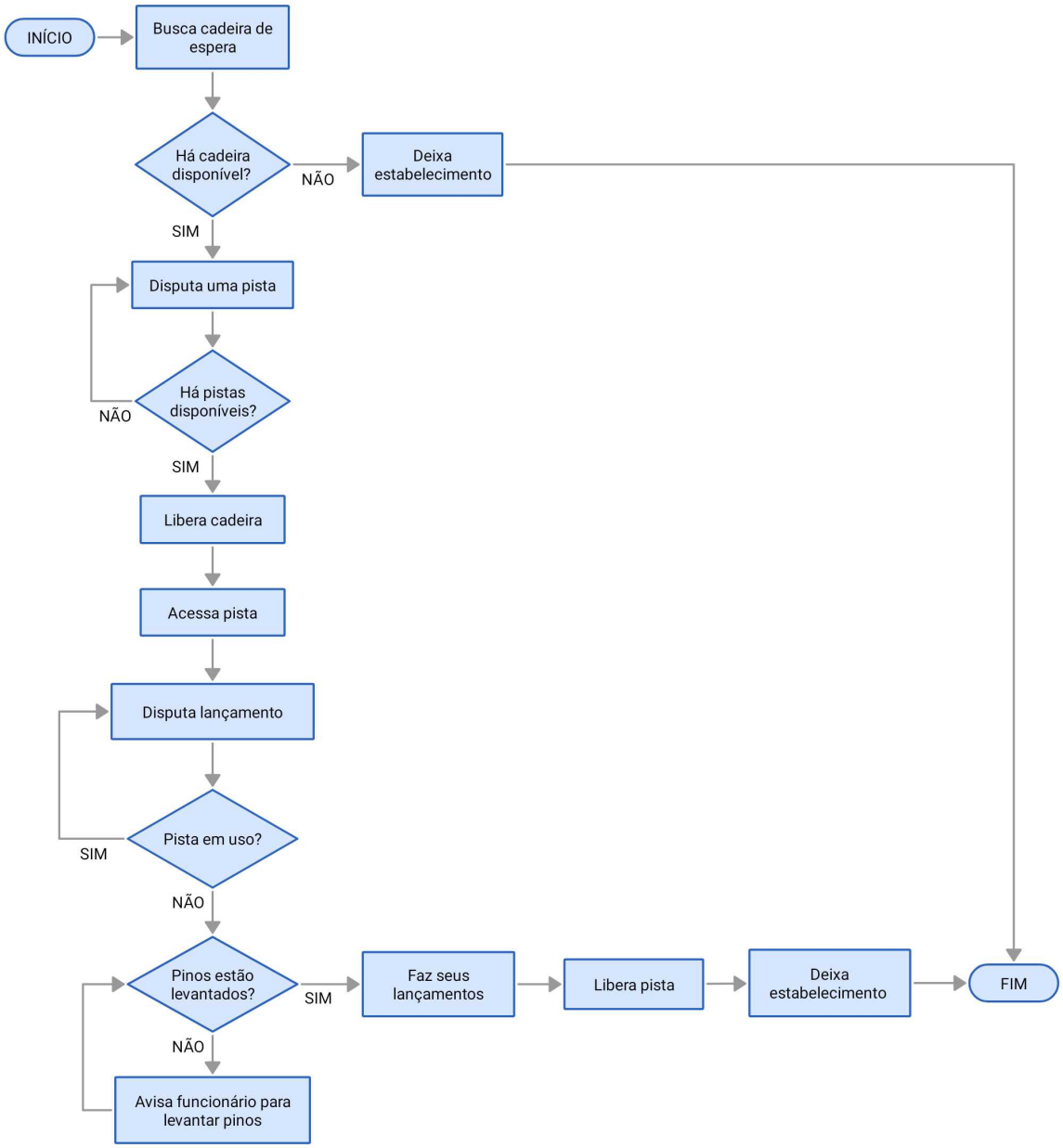
Ao obter acesso a uma pista de boliche, o cliente deve verificar se o seu parceiro está realizando algum lançamento. Se este for o caso, o cliente deve aguardar até que o parceiro complete seus dois lançamentos antes de realizar os seus. Um lançamento só pode ser efetuado se todos os pinos da pista estiverem levantados; assim, após cada jogada, o cliente deve informar ao funcionário que a pista precisa ser preparada novamente.

Por fim, haverá apenas um funcionário encarregado de levantar os pinos quando solicitado por algum cliente. O funcionário percorrerá todas as pistas para verificar quais delas têm pinos derrubados. Ademais, o fluxo de clientes é sempre garantido, e o algoritmo apenas será interrompido manualmente via linha de comando.

Além disso, é fundamental garantir que as ações realizadas em uma pista não interfiram no funcionamento de outra pista. Por exemplo, há duas pistas e uma delas tem os pinos derrubados enquanto a outra tem os pinos levantados, a pista com os pinos levantados não deve ser afetada pela situação da pista com os pinos derrubados. Essa separação é crucial para maximizar a performance da resolução do problema proposto.

2.1 Fluxo do Cliente

A seguir é apresentado um fluxograma que representa o fluxo de operações de um cliente no sistema de boliche. O fluxograma descreve, de forma visual e simplificada, as etapas que o cliente percorre e as suas interações com o funcionário.



3 Descrição do Algoritmo

3.1 Descrição das Funções Utilizadas

Para a implementação do problema proposto, foi-se Implementado as Seguintes Funções:

funcionario

Função que simula o comportamento do funcionário do boliche. O funcionário fica em um loop infinito, esperando até que algum cliente sinalize que há pinos a serem levantados em alguma pista. Quando o sinal é recebido, o funcionário percorre todas as pistas, verifica quais precisam ter seus pinos levantados, reseta o estado da pista e libera o semáforo correspondente para indicar que os pinos estão disponíveis para o próximo cliente. O funcionário repete esse ciclo continuamente.

utilizaPista

Função que simula a utilização de uma pista de boliche por um cliente. Um cliente faz dois lançamentos consecutivos e termina de usar a pista. Antes de cada lançamento, o cliente espera que os pinos estejam levantados (indicado pelo semáforo da pista). Entre os lançamentos, o cliente sinaliza ao funcionário para levantar os pinos novamente. Após o segundo lançamento, o cliente sinaliza uma última vez para o funcionário levantar os pinos para o próximo usuário.

cliente

Função que representa o comportamento de um cliente no boliche. O cliente tenta pegar uma cadeira de espera ao chegar no boliche. Se consegue, ele então tenta encontrar uma pista disponível. Se uma pista está disponível, o cliente a utiliza fazendo dois lançamentos (simulado pela função *utilizaPista*). Caso todas as pistas estejam ocupadas, o cliente entra em espera até que uma pista seja liberada. Se o cliente não consegue uma cadeira de espera, ele deixa o boliche.

main

Função principal que inicializa o sistema. Ela cria as threads para os clientes e o funcionário, inicializa os semáforos para as pistas e cadeiras de espera, e gerencia o ciclo de vida das threads. Cada cliente é executado em sua própria thread para simular o comportamento concorrente de múltiplos clientes utilizando o boliche ao mesmo tempo.

3.2 Descrição das Ferramentas de Sincronização

Em cada função descrita acima, foi-se utilizado diversas variáveis de controle e sincronização. Abaixo, há a descrição para cada uma destas variáveis.

pistasParaLevantarPinos

Essa variável é um vetor que armazena o estado de cada pista, indicando se os pinos precisam ser levantados (1) ou não (0). Cada elemento do vetor corresponde a uma pista específica no boliche.

cadeiras espera

Um semáforo utilizado para gerenciar o número de cadeiras disponíveis na área de espera. Ele controla o acesso dos clientes ao boliche, limitando o número máximo de clientes que podem aguardar simultaneamente.

funcionario levanta pinos

Este semáforo é utilizado para sinalizar ao funcionário que há pinos a serem levantados em uma ou mais pistas. O funcionário aguarda por um sinal deste semáforo para iniciar o processo de levantamento dos pinos.

semafaros pistas

Um vetor de semáforos, onde cada elemento controla o acesso a uma pista específica. Cada semáforo garante que no máximo dois clientes utilizem a mesma pista simultaneamente.

semafaros pistas pinos

Este vetor de semáforos controla o estado dos pinos em cada pista. Um semáforo correspondente a cada pista indica se os pinos estão disponíveis para um novo lançamento.

espera pista

Uma variável de condição usada para gerenciar a fila de espera dos clientes quando todas as pistas estão ocupadas. Ela permite que um cliente espere até que uma pista se torne disponível, sem ocupar CPU desnecessariamente.

lock print

Um mutex utilizado para sincronizar o acesso às saídas de impressão (`stdout`). Ele garante que as mensagens de saída sejam exibidas de forma ordenada e sem interferências entre threads.

lock espera pista

Este mutex protege o acesso à variável de condição *espera pista*, garantindo que a espera por uma pista ocorra de forma segura e ordenada.

lock acesso dupla

Um vetor de mutexes, onde cada elemento corresponde a uma pista específica. Ele coordena o acesso de uma dupla de clientes à pista, garantindo que um cliente só inicie seus lançamentos quando o outro tiver terminado.

lock avisa funcionario

Este mutex é utilizado para proteger o acesso à variável *pistasParaLevantarPinos*, assegurando que o funcionário seja corretamente notificado sobre quais pistas precisam ter os pinos levantados.

4 Conclusão

A implementação do problema do Boliche Manual mostrou-se uma eficaz simulação de um sistema complexo de gerenciamento de recursos compartilhados em um ambiente concorrente. Utilizando-se de semáforos, mutexes e variáveis de condição, foi-se possível coordenar o acesso de múltiplos clientes às pistas de boliche e garantir que o funcionário levantasse os pinos de forma adequada e sincronizada.

O modelo desenvolvido conseguiu evitar diversos problemas clássicos da programação concorrente, como espera ocupada, deadlocks e condições de corrida. Assim, em suma. este exercício proporcionou um entendimento aprofundado sobre a importância das técnicas de sincronização em ambientes de programação concorrente. Ele também destacou a complexidade envolvida na coordenação de threads para alcançar um comportamento consistente e eficiente em um sistema de recursos compartilhados.

5 Referências

- Repositório: https://github.com/EduardoMarciano/PC-Trabalho
- Ben-Ari, M. (2006). *Principles of Concurrent and Distributed Programming* (2ª ed.). Prentice Hall.
- Andrews, G. (1991). *Concurrent Programming: Principles and Practice*. Addison-Wesley. ISBN: 0805300864.
- Breshears, C. (2009). *The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications*. O'Reilly.