A Revolução das Máquinas - A problemática de robôs em contexto de produção de bens sob o paradigma da programação concorrente

Maria Eduarda Carvalho Santos - 190092556

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 0202 - Programação Concorrente

auntduda@gmail.com

1. Introdução

A programação concorrente é um paradigma de programação que lida com a execução simultânea de várias tarefas em um programa. Em vez de executar as tarefas de forma sequencial, a programação concorrente permite que diferentes partes do programa sejam executadas em paralelo, aproveitando o poder de processamento de sistemas com vários núcleos de CPU ou múltiplos dispositivos de processamento. [2]

A aplicação da programação concorrente é ampla e ocorre em vários campos, tais como

- Sistemas operacionais: Os sistemas operacionais modernos utilizam programação concorrente para gerenciar e executar várias tarefas ao mesmo tempo. Por exemplo, um sistema operacional pode permitir que um usuário execute vários aplicativos simultaneamente, como um navegador da web, um editor de texto e um reprodutor de música;
- Servidores web: Servidores web recebem e processam simultaneamente múltiplas solicitações de diferentes clientes. Cada solicitação é tratada em uma *thread* separada, permitindo que o servidor atenda a várias solicitações concorrentemente;
- Aplicações distribuídas: Em sistemas distribuídos, diferentes partes do programa podem ser executadas em diferentes máquinas. A programação concorrente é usada para coordenar a comunicação e a sincronização entre os diferentes componentes distribuídos;
- Jogos: Jogos modernos exigem uma execução concorrente eficiente para processar gráficos em tempo real, lidar com entradas de jogador e simular comportamentos complexos. Várias tarefas, como renderização, física, inteligência artificial e som, podem ser executadas simultaneamente para proporcionar uma experiência de jogo fluida;
- Processamento de dados em tempo real: Em sistemas que lidam com grandes volumes de dados em tempo real, como análise de dados, processamento de *streaming* e sistemas de monitoramento, a programação concorrente é usada para processar e analisar dados em paralelo, garantindo a eficiência e a tomada de decisões em tempo hábil.

Este paradigma é especialmente útil em cenários em que existem tarefas independentes que podem ser executadas simultaneamente para melhorar a eficiência, a capacidade de resposta e o desempenho do sistema como um todo [1].

Na programação concorrente, as principais preocupações incluem a sincronização e a comunicação entre as tarefas concorrentes. Mecanismos como semáforos, *mutexes*, monitores e filas de mensagens são usados para garantir a corretude e a consistência dos dados compartilhados entre as tarefas. Além disso, é importante considerar os possíveis problemas de concorrência, como condições de corrida, *deadlocks* e *starvations*, e aplicar técnicas adequadas para evitá-los ou mitigá-los [3].

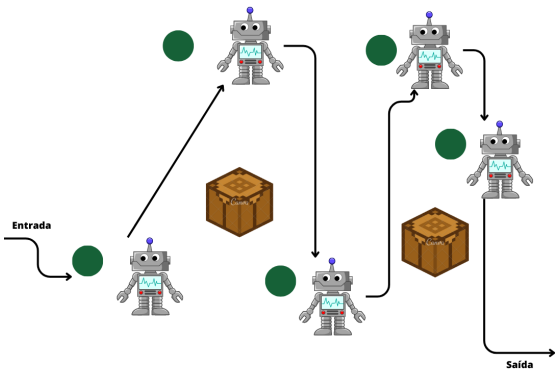
No presente relatório, será apresentado um problema que visa exercitar os conhecimentos adquiridos em aula, a fim de modelar um cenário cuja necessidade do uso de ferramentas que realizam o controle de processos se faz presente. Na seção 2 será apresentado o problema e as características que compõem o cenário de concorrência; na seção 3 será apresentada a solução dada para o problema proposto, de modo que o algoritmo desenvolvido para solucionar este problema estará disponível em outra plataforma de armazenamento, enquanto a elucidação da solução será aqui relatada; e na seção 4 serão apresentadas as problemáticas encontradas no desenvolvimento do problema, os avanços e descobertas feitas ao longo do processo, as considerações finais e a disponibilização de direcionamento ao ambiente de armazenamento de algoritmos (GitHub).

2. Formalização do Problema Proposto

O problema proposto se dá pela seguinte configuração:

Ao longo da década de 1760, a revolução industrial deu início a um novo modo de produzir bens: por meio de robôs. Estes robôs operam exatamente como um ser humano, mas não possuem direitos trabalhistas, nem paradas e muito menos mal funcionamento. O mundo agora opera exclusivamente por meio de máquinas que controlam máquinas, e todos os serem humanos podem se beneficiar da falta de necessidade do trabalho para morar no meio do mato, ou desfrutar de uma vida de luxos às custas da mão de obra automática. A história foca em um robô em especial, de nome Gregório, cuja rotina de receber produtos a serem processados é a mesma há muito tempo.

Este robô trabalha em uma fábrica cujo funcionamento é dado pelo seguinte esquema: os produtos que chegam à fábrica devem ser processados por todos os robôs ali presentes, os quais são dispostos ao longo da fábrica em um formato de zigue-zague [1]; assim, quando um produto chega, ele deve passar pelas mãos metálicas de um robô, o qual terá como procedimento padrão analisar qual é o tipo deste produto, e decidir se este produto deve ser processado por uma, duas ou três ferramentas no máximo, as quais são dadas por machado, martelo ou laço. Assim que um robô identifica o tipo do produto, ele deve se deslocar até a bancada que reside entre 3 robôs consecutivos, e então, selecionar apenas uma ferramenta para processar o produto, levá-lo até seu ambiente de trabalho, usar a ferramenta, e devolvê-la para a bancada. Caso seja necessário utilizar duas ou mais ferramentas, o robô deve pegá-las de maneira singular do ambiente de disposição de materiais, e então, utilizá-las. Cada bancada, compartilhada entre três diferentes robôs, dispõe apenas de uma ferramenta de cada tipo, uma vez que o rápido desenvolvimento industrial e produção de bens culminou na destruição do meio ambiente terrestre, fazendo com que aqueles que optam por viver no campo tenham que se deslocar a outro planeta, e então, a produção de maquinário auxiliar aos robôs tornou-se escasso.



Assim, um produto que chega à fabrica recebe o tratamento devido de todos os robôs, os quais são dispostos em formato linear, ligados por uma esteira de transmissão de produtos, e então, este produto pode ser entregue ao destinatário.

Certo dia, Gregório perde um ciclo de *clock* de seu habitual funcionamento, e acaba parando para refletir sobre toda sua existência e propósito de vida. Ainda que esta problemática de Gregório deva ser explorada, o presente relatório visa modelar o funcionamento desta fábrica por meio de um algoritmo computacional, desenvolvido na linguagem C de programação, fazendo uso de ferramentas descritas na seção 1, tais como *locks* e variáveis de condição, para garantir o controle das atividades desempenhadas na fábrica, e a integridade de processos ali realizados.

3. Descrição do Algoritmo Desenvolvido para a Solução do Problema Proposto

O algoritmo desenvolvido para a solução do problema é dado pelo uso das estruturas de *locks* e variáveis de condição. No paradigma de programação concorrente, *locks* (também conhecidos como bloqueios ou *mutex*) são mecanismos utilizados para controlar o acesso exclusivo a recursos compartilhados entre múltiplas *threads* [3]. Eles garantem que apenas uma *thread* por vez possa acessar um determinado recurso, evitando problemas de concorrência, como condições de corrida e inconsistências nos dados.

Quando múltiplas *threads* estão executando simultaneamente e compartilham um recurso, pode ocorrer uma situação em que duas ou mais *threads* tentam acessar ou modificar o recurso ao mesmo tempo. Isso pode levar a resultados indeterminados ou incorretos, uma vez que as operações podem se sobrepor ou intercalar de maneira imprevisível. Ao usar *locks*, uma *thread* que deseja acessar o recurso primeiro deve adquirir o bloqueio associado a ele. Se o bloqueio estiver disponível, a *thread* o adquire e continua sua execução. Se o bloqueio estiver sendo usado por outra *thread*, a *thread* atual fica bloqueada, esperando até que o bloqueio seja liberado pela *thread* que o está mantendo. Quando uma *thread* termina de usar o recurso, ela libera o bloqueio, permitindo que outras *threads* possam adquiri-lo. Dessa forma, apenas uma *thread* por vez pode ter acesso exclusivo ao recurso compartilhado [1].

Ainda, as variáveis de condição são mecanismos que permitem a comunicação e sincronização entre *threads*, além de possibilitarem a espera por certas condições específicas antes de continuar a execução.

As variáveis de condição são geralmente associadas a um *lock* (bloqueio), como um *mutex*, e são usadas em conjunto com ele para criar seções críticas protegidas [2]. A ideia principal por trás das variáveis de condição é permitir que as *threads* esperem até

que uma determinada condição seja satisfeita antes de prosseguir.

As variáveis de condição admitem dois tipos de operação, as quais são dadas por:

- `pthread_cond_wait`: Essa função é chamada dentro de uma seção crítica protegida por um *lock*. Quando uma *thread* chama `pthread_cond_wait`, ela libera o *lock* e entra em um estado de espera até que outra *thread* sinalize a variável de condição. Quando a variável de condição é sinalizada, a *thread* é acordada e tenta adquirir o *lock* novamente antes de continuar a execução;
- `pthread_cond_signal` ou `pthread_cond_broadcast`: Essas funções são usadas para sinalizar a variável de condição. Quando uma *thread* chama `pthread_cond_signal`, ela sinaliza a variável de condição, acordando uma única *thread* (se houver alguma em espera) que estava esperando nessa variável de condição. Já a função `pthread_cond_broadcast` sinaliza a variável de condição e acorda todas as *threads* que estavam esperando nela.

Para resolver o problema, os robôs foram considerados como *threads* independentes, as quais operam sobre uma lista de produtos, declarada globalmente no escopo do programa. Existe uma quantidade estática de robôs, bancadas e produtos, os quais são declarados por meio das variáveis `BANCADAS`, `ROBOS` e `PRODUTOS` de maneira global. Para auxiliar o processo de correção e programação do problema, as estruturas de dados `bancada` e `produto` forma desenvolvidas, e seus respectivos atributos são incializados na função `main`.

A função `main` é responsável por inicializar os tipos de cada produto, bem como declarar a quantidade de materiais que existe em cada bancada compartilhada por três robôs consecutivos, além de realizar a criação e execução das *threads* desenvolvidas para o problema.

A função *esteira* caracteriza o funcionamento de cada robô, o qual é dado pela coleta de cada produto existente na esteira de produção, o cálculo da bancada/das bancadas que o robô poderá utilizar ao longo do processo, e a identificação de cada produto por meio da estrutura `switch case`.

Visando restringir o acesso não-monitorado das *threads* às variáveis locais de controle e endereçamento, um *lock* foi criado para realizar os cálculos de bancadas disponíveis sem que haja mudança na indexação do robô (`pthread_mutex_t lock_calculo`), e ainda, a variável de condição `cond_ferramenta` foi implementada para garantir que cada robô pegará uma ferramenta da bancada, e o outros robôs terão que aguardar sua vez para coletar a ferramenta. Após identificar o tipo de produto e quais ferramentas serão utilizadas para sua produção, o robô entra em algum caso da estrutura `switch case`, e então, identifica se a ferramenta necessária para o processamento está disponível nas bancadas que tem acesso. Caso não seja possível encontrar as ferramentas necessárias, o robô deverá esperar até que outro robô devolva a mesma para a bancada. Por outro lado, o robô poderá pegar a ferramenta da bancada, processar o produto, devolver a ferramenta à bancada, e processar o próximo produto. Este procedimento se repetirá até que todos os robôs tenham processado todos os produtos.

4. Conclusão

Em conclusão, o projeto empregou *locks* e variáveis de condição da programação concorrente para lidar com a concorrência e sincronização entre *threads*. Durante o de-

senvolvimento, alguns gargalos foram identificados, como a limitação de operação em uma única *thread* e a dependência da biblioteca `"string.h"`, a qual não permite uso quando na linguagem C. Apesar desses desafios, o projeto proporcionou valiosas oportunidades de aprendizado e aplicação dos conceitos básicos da programação concorrente. Os conhecimentos adquiridos em sala de aula foram exercitados e reforçados, permitindo relembrar o uso de estruturas básicas, como listas de caracteres em C.

O projeto também ofereceu a oportunidade de modelar um problema fictício de forma algorítmica, demonstrando a capacidade de pensar de maneira estruturada e desenvolver soluções eficientes para problemas complexos. Ao enfrentar os desafios encontrados, foram adquiridas habilidades valiosas de resolução de problemas e aprofundamento no domínio da programação concorrente, apesar das limitações e restrições.

References

[1] Gregory R Andrews. *Concurrent programming: principles and practice*. Benjamin-Cummings Publishing Co., Inc., 1991.

[2] Mordechai Ben-Ari and Môtî Ben-Arî. *Principles of concurrent and distributed programming*. Pearson Education, 2006.

[3] Clay Breshears. *The art of concurrency: A thread monkey's guide to writing parallel applications*. " O'Reilly Media, Inc.", 2009.