Trabalho de Threads

Problema do Bar dos Filósofos

Gabriel de Souza Ferreira – 17250447
Jean Marcelo Mira Junior – 16102369

Resumo. *O presente trabalho da disciplina de sistemas operacionais tem por finalidade explorar o uso de threads e seus métodos de sincronização e controle de acesso a regiões críticas.*

Palavras chave: threads, simulação, cpp.

Introdução

Esse trabalho tem como objetivo a resolução do problema do bar dos filósofos que é uma generalização do problema jantar dos filósofos, onde consiste no gerenciamento dos recursos compartilhados através da utilização de threads. Sendo assim, foram desenvolvidas três classes diferentes: a classe Filosofo, Mesa e Entrada. A classe Filosofo tem as principais características do filósofo, a classe Mesa que é instanciada para cada filósofo tem como objetivo fazer a integração entre os filósofos no sentido de ter variáveis que possam ser visualizadas por todos os filósofos e pôr fim a classe entrada que tem função principal ler e identificar características dos arquivos .txt onde estão os grafos para análise.

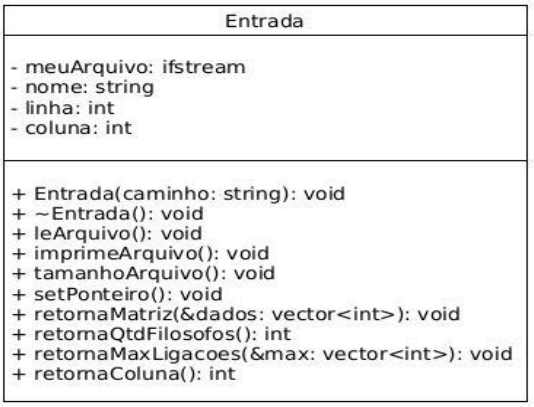

Figura 1: Diagrama UML classe Entrada.

Discussão

Classe Entrada - explorando entrada.h e entrada.cpp:

Esta classe é responsável por fazer toda análise de entrada de dados através de arquivos .txt que contém os grafos. Aprofundando na funcionalidade desta classe simples podemos verificar que ela tem como função ler, identificar e retornar para o usuário a quantidade de garrafas, filósofos e os dados contidos nos arquivos de texto.

Classe Mesa - explorando mesa.h e mesa.cpp:

Esta classe tem papel fundamental na comunicação entre os filósofos, ela é inicializada na main.cpp e depois passada como referência para cada filósofo assim fazendo com que cada filósofo tenha noção no sentido de acompanhar ao mesmo tempo a quantidade de garrafas e a disponibilidade de cada garrafa.

Além disso, tem a principal função de conter e executar os conceitos de semáforos juntamente com as funções de mutex do c++ fazendo assim o controle de acesso à memória compartilhada entre as threads filósofos e controle sobre o fluxo de impressão dos dados no terminal do usuário.

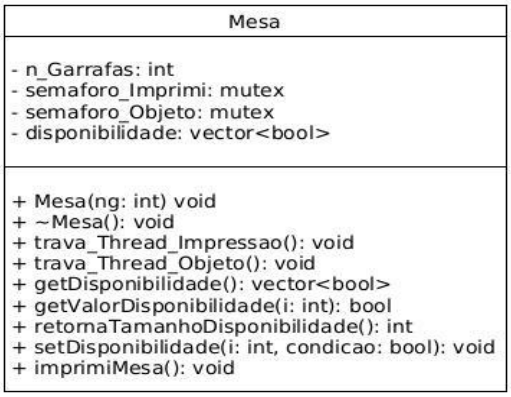

Figura 2: Diagrama UML classe Mesa.

Classe Filosofo - explorando filosofo.h e filosofo.cpp:

Esta classe faz o gerenciamento da rotina do filósofo, desde quantidade de garrafas, quem e quais os vizinhos o filósofo compartilha garrafa e as rotinas tranquilo, com sede e bebendo. Todo filósofo tem um objeto do tipo mesa que recebe por referência, assim tendo comunicação efetiva da movimentação das garrafas e a partir disso tendo capacidade lógica para tomar decisões sobre o que fazer com o ambiente que lhe foi dado. Além disso, tem funções e variáveis que permitem o controle do tempo de execução de cada estado.

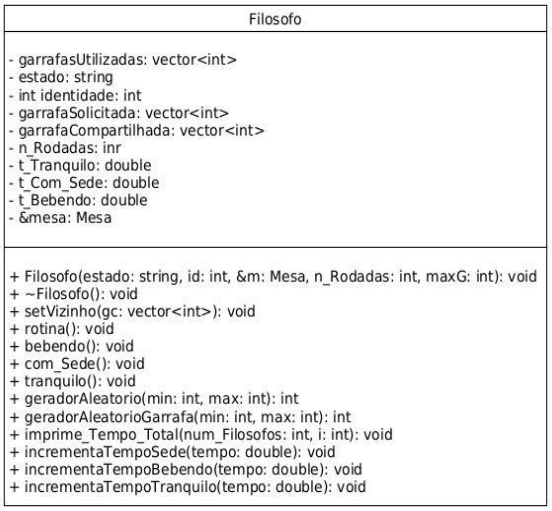

Figura 3: Diagrama UML classe Filosofo.

Funcionamento do Programa - main.cpp

Iniciamos o funcionamento com o usuário especificando o nome do grafo, por exemplo grafo5.txt, depois o usuário deve especificar a quantidade de rodadas que deseja executar a rotina. Após o usuário determinar as entradas do sistema, o programa identifica os valores do grafo bem como a quantidade de filósofos e de garrafas.

Assim criando o objeto mesa e os filósofos, cada um com suas características individuais como identidade além de apontar quais vizinhos pode fazer comunicação, e a quantidade de ligações que pode fazer. Posteriormente com todo sistema criado podemos então aplicar os conceitos de thread e inicializar o programa fazendo com que cada filósofo inicialize sua rotina.

Funcionamento da Rotina - Threads

Após criar as threads na main.cpp e inicializá-las a função rotina da classe Filosofo e executada, rotina executa os três principais estados do filósofo além de controlar o tempo de execução das funções tranquilo, com sede e bebendo. Todo processo de impressão e ou onde se compartilha memória são utilizadas funções que trabalham como conceitos de semáforos através da biblioteca de mutex empregada na linguagem c++ para fazer o controle da impressão bem como o controle de cada filósofo para alterar e ver na mesa quais vizinhos pegaram ou não as garrafas que lhe são permitidas.

Tranquilo que é o primeiro estado imprime na tela a identidade do filósofo e o seu estado atual, depois chama uma função que gere um número aleatório de 0 até 2 e multiplica por um segundo fazendo assim o tempo de espera (delay) variar aleatoriamente para que o filósofo saia deste estado.

Subsequentemente temos a função com sede que tem protagonismo em nosso código, a função imprime a identidade do filósofo e o estado em que ele se encontra depois inicializa um laço quase que infinito de espera pelas condições adequadas para que o filósofo possa beber. Primeiro geramos um número aleatório de 2 até n onde n é a quantidade máxima de garrafas que o filósofo tem direito a pegar.

Agora entrando no laço de repetição verificamos a quantidade de garrafas disponíveis para aquele filósofo, posteriormente verificamos o máximo de garrafas que o mesmo pode consumir, se a garrafa disponível for a mesma que o filósofo tem conexão e portanto tem direito de pegar ele adiciona essa garrafa a um auxiliar que não irá pegar para si a garrafa mas apenas indicar que o filósofo tem condições de obter uma garrafa, assim adicionando até chegar ao número de garrafas que foi gerado aleatoriamente para o filósofo naquela rodada. Aqui vale a pena indagar um ponto importante da lógica do programa, o vetor auxiliar que salva todas as garrafas que o filósofo pode vir a pegar serve para que o filósofo não adicione as garrafas para si uma de cada vez pois se

adicionar uma de cada vez pode vir a atrapalhar os seus vizinhos filósofos.

Assim que se constata que o filósofo consegue pegar a quantidade que lhe foi ordenada naquela rodada ele adiciona a garrafa para si fazendo com que assim a mesma fique indisponível perante seus vizinhos. Caso o filósofo não consiga pegar a garrafa por indisponibilidade ele apaga o vetor auxiliar e fica no loop da função com sede até que consiga as garrafas.

Por fim a função bebendo imprimir a identidade do filósofo e a condição atual que ele se encontra, depois espera por 1 segundo delay preestabelecido pelo problema e libera as garrafas para que seus vizinhos ou até mesmo ele depois de passar pelo estrado de tranquilo possa beber.

Resumindo nossa ideia base é fazer com que a mesa seja uma um objeto do filósofo passado por referência e assim podendo fazer o controle em tempo real das garrafas que estão disponíveis, o filósofo só pega as garrafas quando tem certeza que pode saciar sua sede com a quantidade que lhe foi estabelecida aleatoriamente e as devolve 1 segundo depois de beber. Os semáforos são utilizados com auxílio de mutex, no programa se localizam acima dos trechos do código que compartilham tanto memória como acesso a área de impressão do terminal, assim evitando possíveis problemas relacionados à utilização das threads e tendo total eficácia na implementação.

Conclusão

Após tentar implementar as threads nós deparamos com alguns problemas nas questões de acesso de memória e na parte de imprimir no terminal as saídas. Ambos os problemas são gerados pela falta de sincronização, que é o principal motivo pelo qual se utiliza thread.

Através da lógica e empregando uso dos conceitos de semáforos juntamente com as bibliotecas de mutex podemos dar uma solução eficaz ao problema do bar dos filósofos, evitando deadlocks e starvation. Constatando que a thread está funcionando de forma assincronamente através do tempo total de execução das threads em relação ao tempo de execução de cada thread individualmente, pois o tempo total de execução equivale ao tempo de execução de cada thread individualmente assim comprovando que as mesmas executam em paralelo.