Laboratório 2 - Jantar do Filósofos - Locks

O problema do jantar dos filósofos é uma problema clássico no campo da programação concorrente. Ele serve para comparar vários formalismos e provê programas concorrentes. É um problema suficientemente simples para ser tratado, ainda que bastante desafiante.

O problema é estabelecido em uma comunidade isolada de 5 filósofos (0-4). Os filósofos engajam-se somente em duas atividades: *pensar* e *comer.*

*Task Filosofo is*

*Begin*

*Loop*

*Think;*

*Pre-Protocolo; // tarvamento com locks*

*Eat; // seção crítica do código do filósofo*

*Pos-Protocolo; // destravamentocom locks*

*End loop;*

*End Filosofo;*

As refeições são feitas comunitariamente em uma mesa – uma travessa de *spaghetti* - é colocado no centro de uma mesa com 5 pratos (0-4) e 5 garfos (0-4), que interminavelmente reabastecido. Infelizmente, o *spaghetti* é irremedialvelmente emaranhado e um filósofo necessita 2 garfos para comer. Cada filósofo pode pegar os garfos ao seu lado esquerdo e direito, mas somente 1 garfo em um tempo.

O problema é implementar pré e post protocolos, usando o mecanismo de locks, para garantir que um filósofo somente come se ele tem 2 garfos. A solução deve também satisfazer as propriedades de correção para exclusão mútua (instruções de seções críticas de dois ou mais filósofos não podem ser intercalados) , tais como:

1. *Um filósofo come somente se ele tem 2 garfos.*
2. *Nenhum de dois filpósofos pode reter o mesmo garfo simultaneamente.*
3. *Não pode exitir deadlock.*
4. *Nenhum filósofo entra em starvation.*
5. *Comportamento do programa concorrente é eficiente sob ausência de* ***contenção*** *(dois processos ou duas thread concorrem pelo mesmo recurso).*

Uma primeira tentativa de implementação é mostrada no que segue. Assumimos que cada filósofo é inicializado com seu índice I. Cada garfo é implementado usando o mecanismo de locks. Um filósofo deve completar uma espera (wait) sobre ambos seus garfos, direito e esquerdo, antes de comer.

Fork: array (0..4) of Locks

*Task Filosofo is*

*Begin*

*Loop*

*Think;*

*Wait(Fork(I));  
 Wait(Fork((I+1) mod 5));*

*Eat; // seção crítica do código do filósofo*

*Signal(Fork(I));  
 Signal(Fork((I+1) mod 5));*

*End loop;*

*End Filosofo;*

Infelizmente, esta solução apresenta um problema. Descreva qual problema aparece nesta primeira tentativa de implementação.

Para resolver, imagine agora, uma sala de jantar (Room) e para garantir correção (com a propriedade de vivacidade) para o Problema do Jantar dos Filósofos, limite o número de filósofos entrando na sala (Room) para 4.

Room: que deve ter a entrada controlada por um mecanismo de sincronização para o limite de 4 filósofos, por exemplo usando um lock ou um semáforo.

Fork: array (0..4) of Locks

*Task Filosofo is*

*Begin*

*Loop*

*Think;  
 Wait(Room);*

*Wait(Fork(I));  
 Wait(Fork((I+1) mod 5));*

*Eat; // seção crítica do código do filósofo*

*Signal(Fork(I));  
 Signal(Fork((I+1) mod 5));*

*Signal(Room);  
 End loop;*

*End Filosofo;*

Uma outra solução que é livre de *starvation* (**se um filósofo indica sua intenção de entrar em sua seção crítica, por começar a execução do seu pré-protocolo, eventualmente ele será bem sucedido**) e de *deadlock* (**se nenhum filósofo é bem sucedido para fazer a transição de seu pré-protocolo para a sua seção crítica;** livre de **deadlock é a situação em que algum filósofo está tentando entrar em sua seção crítica e então um deles deve eventualmente ser bem sucedido**) é a solução assimétrica, na qual os primeiros quatro filósofos executam a solução original, mas o quinto, espera primeiro obter o garfo a sua direita, e então, para o garfo a sua esquerda.