Cruzamento inteligente

Luís Felipe Braga Gebrim Silva, 16/0071569

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 117935 - Programação Concorrente - Turma A

arueluis@gmail.com

1. Introdução

A programação concorrente é um paradigma de programação que permite otimizar o uso dos recursos do computador e simular processos de natureza concorrente, onde vários agentes disputam por um mesmo recurso.

Neste relatório será apresentado um algoritmo que utiliza da programação concorrente para simular um problema real e apresentar um método de controlar a concorrência para solucionar este problema.

2. Formalização do Problema

O problema proposto é a implementação de um cruzamento controlado por semáforos inteligentes.

Diversos carros se movimentando em diferentes sentidos tentam atravessar um mesmo cruzamento, todos este carros concorrem pelo espaço da pista, e somente um carro pode ocupar cada espaço desta pista.

Existem regiões em que os carros concorrem pelo espaço somente com carros que se movimentam na mesma direção, sendo que alguns carros tem a velocidade limitada pelos carros que estão em sua frente.

Existe uma região do cruzamento, a região central, onde os carros que se movem em diferentes direções podem acessar, fazendo com que diferentes carros concorram pelo mesmo espaço e tenham seus movimentos impedidos.

O tráfego de carros na região do cruzamento que pode ser ocupada por carros que se movem em diferentes direções pode ficar caótico, fazendo com que alguns carros tenham que esperar indefinidamente tentando atravessar essa região.

Um método para manter o tráfego organizado na região central do cruzamento é a utilização de semáforos de trânsito, que decidem quais carros podem entrar na região central a cada momento. Será apresentado um algoritmo que simula um semáforo que visa otimizar o tráfego, evitando bloquear o tráfego quando não é necessário e garantindo que todos os carros atravessem a região central do cruzamento.

3. Descrição do Algoritmo

O cruzamento é representado por uma matriz que contém uma cruz dividida em quatro faixas, cada uma para um tipo de carro, os que vão de cima para baixo, de baixo para cima, da esquerda para a direita e da direita para a esquerda. No centro da cruz existe uma área que é atravessada por diferentes tipos de carros. Cada carro é representado por um caractere que aponta para a direção para qual ele se dirige.

Cada carro executa uma thread e se movem em diferentes velocidades definidas aleatoriamente, não é permitido que dois ou mais carros ocupem a mesma posição na matriz, portanto os carros concorrem pelas posições da matriz.

Os carros são gerados aleatoriamente a cada 180 milissegundos, cada tipo de carro tem uma probabilidade diferente de aparecer a cada 180 milissegundos, e no máximo um tipo de cada carro é gerado nesse período de tempo.

A velocidade de cada carro é definida por uma chamda da função usleep, que a cada vez que um carro se movimenta faz com o que este carro espere por um período de tempo aleatório entre 35 e 335 milissegundos antes de tentar realizar o próximo movimento.

Para impedir que dois carros ocupem uma mesma posição foi utilizada uma matriz de locks, antes de um carro se mover para uma determinada posição da matriz ele deve possuir o lock correspondente àquela posição e após sair de uma posição ele deve liberar o lock correspondente à posição deixada.

Com está implementação todos os espaços são ocupados sempre por no máximo um carro, no entanto dois carros que se movem em direções diferentes, um verticalmente e o outro horizontalmente, podem colidir e um dos carros ficará travado até que o outro se mova, com carros se movendo em quatro direções diferentes é possível que estas colisões causem um deadlock, como na situação apresentada na figura 1

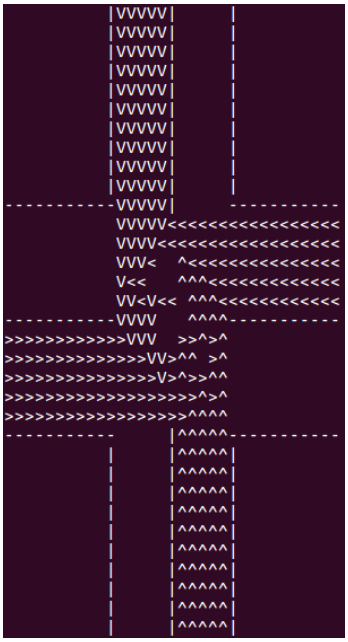

Figura 1. Representação do cruzamento onde nenhum carro consegue se mover.

Para resolver o problema da colisão entre carros que se movem em diferentes direções foi implementado um semáforo de trânsito, que bloqueia todos os carros que se movem horizontalmente ou verticalmente e permite que apenas carros que se movimentam paralelamente acessem a região central do cruzamento simultaneamente.

Existem quatro semáforos, cada um para controlar uma faixa do cruzamento, os

semáforos das duas faixas verticais sempre são abertos e fechados simultaneamente, assim como os dois semáforos das faixas horizontais.

Duas variáveis de condição são utilizadas para parar os carros que se movimentam nas faixas controladas pelos semáforos, uma variável para as faixas verticais e outra para as faixas horizontais.

Quando um semáforo é fechado uma variável do tipo inteiro tem o valor alterado para '1' para indicar que todas as threads dos carros das faixas correspondentes ao semáforo devem dormir ao tentar adentar a área central do cruzamento. Quando o semáforo é aberto a variável tem o valor alterado para '0' para indicar que as threads dos carros das faixas correspondentes não precisam mais parar, e é realizado um broadcast para acordar as threads que estavam dormindo.

Os semáforos são controlados por uma única thread, que decide quais faixas serão bloqueadas de acordo com três fatores:

1. A quantidade de carros se movimentando nas faixas horizontais e verticais.
2. A quantidade de carros se movimentando no centro do cruzamento.
3. A quantidade de tempo entre o último bloqueio de um dos semáforos e o momento atual.

Inicialmente o controlador do semáforo mantém as faixas horizontais liberadas e as faixas verticais fechadas.

O controlador do semáforo fica em execução constantemente verificando o estado do cruzamento, e realiza as seguintes verificações e ações:

1. Caso não exista nenhum carro no cruzamento os semáforos são mantidos no mesmo estado.
2. Caso só existam carros em um sentido, vertical ou horizontal, o semáforo deste sentido é liberado e outro sentido é bloqueado.
3. Caso existam carros atravessando no sentido horizontal e vertical é verificado qual dos sentidos está com semáforo aberto e a quanto tempo esse semáforo está aberto, caso este tempo seja maior que 6 segundos o semáforo aberto é fechado e o semáforo do outro sentido é aberto.

É importante destacar que, antes de liberar o fluxo para um dos sentidos do trânsito, o controlador do semáforo fecha o sentido contrário e espera o centro do cruzamento ficar vazio, só então um dos sentidos tem o fluxo liberado.

Na implementação dos semáforos foi preciso garantir que quando o semáforo decidisse parar o fluxo de alguma faixa de trânsito ele conseguisse realizar esse procedimento sem concorrer com os carros que tentavam atravessar tal faixa, a utilização de variáveis de condição se mostrou uma solução simples e eficiente para tal fim, enquanto a utilização de locks se mostrou problemática, já que o semáforo poderia se tornar ineficiente, precisando esperar muitos carros atravessarem a área que ele pretende bloquear enquanto concorre pelo lock.

4. Conclusão

Poder verificar os diferentes problemas gerados pelo livre fluxo de carros por um cruzamento, e os desafios de se controlar este fluxo de forma eficiente foi uma boa experiência

e um bom exercício para verificar onde é mais eficiente o uso de locks e variáveis de condição.

O fato do problema de trânsito ser algo comum e constantemente observado nas cidades fez com que fosse possível verificar o poder que o paradigma de programação concorrente possui para simular situações reais com fidelidade.

5. Referências

http://www.cic.unb.br/~alchieri/disciplinas/graduacao/pc/pc.html