Aula 8 - Exclusão Mútua

Existem diversos recursos em um sistema que não podem ser utilizados simultaneamente por diversos processo se desejamos o correto funcionamento de um programa. Desta forma acesso exclusivo a recursos deve ser fornecido pelo sistema. Esta exclusividade de acesso é chamada *exclusão mútua*. Um algoritmo que implementa exclusão mútua deve satisfazer os seguintes critérios:

- *Exclusão mútua.* Dado um recurso compartilhado que pode ser acessado por diversos processos ao mesmo tempo, somente um processo pode acessar aquele recurso a qualquer momento. Isto é, um processo que ganhou acesso ao recurso deve liberá-lo antes que outro processo ganhe acesso ao mesmo.
- *Starvation.* Cada processo que requesita o recurso deve recebe-lo em algum momento.

Algoritmo Centralizado

Neste método, um processo do sistema é eleito como o coordenador e coordena as entradas nas seções críticas (SC). Cada processo que deseja entrar em uma SC deve antes pedir autorização para o coordenador. Se não existe processo acessando a SC então o coordenador pode imediatamente garantir o acesso ao processo que fez a requisição. Entretanto, se dois ou mais processos pedem acesso à SC então o coordenador garante acesso a somente um processo de acordo com algum algoritmo de *scheduling*. Após o término da execução na SC, o processo informa o coordenador de forma que o coordenador possa garantir acesso à SC para outro processo (se existir outro).

Algoritmo Distribuído

Quando um processo deseja entrar em uma SC ele envia uma mensagem para todos os outros processos. A mensagem contem as seguintes informações:

- o identificador do processo
- o nome da seção crítica que ele deseja acessar
- um *timestamp* único gerado pelo processo que enviou a mensagem

Ao receber uma mensagem, o processo envia uma resposta imediata ao processo que enviou a mensagem garantindo o acesso ou não envia uma resposta baseado nas seguintes regras:

- Se o processo que recebeu a mensagem está executando na SC, ele enfilera a mensagem e não envia resposta

- Se o processo que recebeu a mensagem não está executando na SC, mas está esperando sua vez para acessar a SC, ele compara o *timestamp* que está na mensagem, com seu *timestamp*. Se o *timestamp* que veio com a mensagem é menor que o seu *timestamp*, então ele envia uma resposta, caso contrário ele enfilera a mensagem e não envia resposta.

Algoritmo baseado na passagem de um *token*

Neste método , exclusão mútua é conseguida pelo uso de um *token* único que circula entre os processos do sistema. Um *token* é uma mensagem especial que dá ao detentor da mensagem direito de acesso à SC. Para se ter um algoritmo justo, os processos são organizados logicamente em um anel, o *token* circula entre os processos baseado no anel e sempre na mesma direção.

Deadlock

Um *deadlock* é causado pela situação onde um conjunto de processos está bloqueado permanentemente, i.e. não conseguem prosseguir a execução, esperando um evento que somente outro processo do conjunto pode causar.

Condições necessárias para *deadlock*

As seguintes condições são necessárias para que uma situação de *deadlock* aconteça no sistema:

- exclusão mútua
- segura e espera
- não preempção
- espera circular

Estratégias

Várias estratégias são utilizadas para trabalhar com *deadlock*. As quatro mais conhecidas são listadas abaixo:

- algoritmo do avestruz (ignora o problema)
- detecção (permite que *deadlock* ocurra, detecta, e tenta recuperar)
- prevenção (estaticamente faz com que *deadlocks* não ocorram)
- evitar (evita *deadlock* alocando recursos cuidadosamente)

Algoritmo de detecção centralizado

Em cada máquina se mantem um grafo de alocação de recursos pelos processos, um coordenador centralizado mantem um grafo completo do sistema (a união dos grafos locais). Quando o coordenador detecta um ciclo, ele mata um dos processos e acaba com o *deadlock*.

Diferente de sistemas centralizados, onde as informações estão disponíveis automaticamente no local correto, em um sistema distribuído estas informações devem ser enviadas explicitamente. Como cada máquina tem um grafo local, sempre que um arco é incluído ou excluído do grafo local, uma mensagem é enviada para o coordenador para que o mesmo atualize seu grafo. O problema acontece quando uma mensagem demora para chegar, o que pode causar um **falso *deadlock***.

Algoritmo de detecção distribuído

O algoritmo para detecção de *deadlock* proposto por Chandy-Misra-Haas funciona da seguinte forma. O algoritmo é executado quando um processo tem que esperar por algum recurso devido a outro processo estar utilizando o mesmo. Quando esta situação ocorre, uma mensagem especial é enviada (*probe message*) e enviada para o processo que esta utilizando o recurso. A mensagem consistem de 3 informações: o número do processo que está bloqueado, o número do processo que está enviando a mensagem, e o número do processo para quem a mensagem está sendo enviada. Quando a mensagem chega em um processo, o processo verifica se ele está esperando por algum recurso que esta em uso por outro processo. Se sim, então a mensagem é atualizada, mantendo-se o primeiro campo, mas trocando o segundo campo por seu número de processo e o terceiro pelo número do processo que ele está esperando desbloquear o recurso. Se ele está bloqueado devido a diversos processos, então a mensagem é enviada para todos os processos que detem um recurso que o processo necessita. Se a mensagem dá toda a volta e chega no processo que iniciou a mensagem, isto é o processo cujo identificador está no primeiro campo da mensagem, um ciclo existe e o sistema está em *deadlock*.

Aula 9 - Algoritmos de Eleição

Diversos algoritmos em sistemas distribuídos necessitam um processo para agir como coordenador, inicializador, sequenciador, ou ter algum papel especial. Nesta seção veremos dois algoritmos para definir qual processo assume o papel de coordenador. Estes algoritmos são conhecidos como *algoritmos de eleição*.

Bully Algorithm

Este algoritmo foi desenvolvido por Garcia-Molina em 1982. Quando um processo detecta que o coordenador não existe ou não está mais respondendo as requisições, ele inicia um processo de eleição. Um processo P inicia uma eleição da seguinte forma:

- P envia um mensagem de ELEICAO para todos os processos que possuam identificador maior que o seu.
- Se nenhum processo responde, P vence a eleição e torna-se o coordenador.
- Se um processo com identificador maior que o do processo responde, aquele processo assume o controle. O trabalho de P está terminado.

A qualquer momento, um processo pode receber uma mensagem ELEICAO de um dos processos com identificador menor que o seu. Quando a mensagem chega, o processo manda uma mensagem OK de volta para indicar que ele está vivo e irá assumir o controle. O processo que recebeu a mensagem inicia então o processo de eleição, a menos que ele já tenha iniciado um. No final, todos os processos menos um desistirá, e aquele que não desistiu será o novo coordenador. Ele anuncia sua vitória com uma mensagem para todos os processos indicando que ele está agindo como coordenador a partir daquele momento.

Ring Algorithm

Um outro algoritmo de eleição é baseado no uso de um anel, mas sem *token*. Assume-se que os processos estão física ou logicamente em ordem, de forma que cada processo conheça seu sucessor. Quando qualquer processo detecta que o coordenador não está funcionando, ele constrói uma mensagem ELEICAO contendo seu identificador e o encaminha para o seu sucessor. Se o sucessor não está funcionando e processo envia para o seguinte no anel. A cada passo o processo que recebe a mensagem inclui seu identificador na mensagem e envia esta mensagem para o seu sucessor. No final a mensagem volta ao processo que iniciou a eleição. Este processo reconhece este evento analisando o conteúdo da mensagem e encontrando lá o seu identificador. Neste ponto, o tipo de mensagem é modificada para COORDENADOR e volta a circular entre os processos, indicando quem é o novo coordenador (e.g. o com o número de processo maior). Uma vez que a mensagem circulou entre todos os processos ela é retirada do sistema.