Resumo sobre Comunicação entre processos

Nome: Daniel Sant' Anna Andrade
Matrícula: 20200036904
Nome: Luan Gadioli Mendonça Berto
Matrícula: 20200034042

1 - Condições de corrida

São quando dois ou mais processos estão lendo ou escrevendo dados compartilhados e o resultado depende de quem executa sua parte precisamente, isso pode levar a um processo ficar esperando uma entrada ou saída que nunca irá acontecer. Por quando do aumento na quantidade de núcleos em um processador, está se tornado mais comum as condições de corrida.

2 - Regiões críticas

São partes de um programa onde um processo pode acessar uma memória ou arquivo compartilhado ou então realizar tarefas críticas, que podem levar a condição de corrida anteriormente explicado. Para isso as regiões críticas determinam condições para que se realize a tarefa sem que ocorra a condição de corrida, sendo elas:

1. Dois processos jamais podem entrar em regiões críticas.
2. Nenhuma suposição pode ser feita a respeito de velocidades ou número de CPUs.
3. Nenhum processo executando fora de sua região crítica pode bloquear qualquer processo.
4. Nenhum processo deve ser obrigado a esperar eternamente para entrar em sua região crítica.

3 - Exclusão mútua com espera ocupadas

3.1 - Desabilitando interrupções

Em sistemas que utilizem um único processador, é melhor que cada processo desabilite as interrupções até que o mesmo saia de sua região crítica, só as ligando novamente, ao sair da região.

O problema dessa abordagem é que caso um processo desligue uma interrupção, pode acontecer dele nunca mais ligar novamente. No geral, desabilitar as interrupções é uma boa abordagem dentro do SO, mas não como um mecanismo de exclusão mútua para processos de usuário. Mesmo desligando as interrupções, ainda é possível acontecer exclusão mútua com processadores multinúcleos. Nesses processadores, desligar o sistema de interrupções não evita que os outros núcleos interfiram com as operações que a primeira está realizando.

3.2 - Variáveis do tipo trava

É definido uma variável para travar a entrada na região crítica, que inicialmente será 0 para não travado e 1 para travado.

Quando um processo entra na região crítica, ele verifica a variável de trava, caso ela seja 0, o processo a define para 1 e entra na região crítica. Ao sair da região crítica, ele a

configura para 0 e sai. Caso um processo tente entrar na região e veja que a trava está como 1, ele aguarda até que ela fique 0, para poder fazer o seu processo.

O problema dessa técnica é caso dois processos verifiquem simultaneamente que a trava está como 0 e os dois entrem na região crítica ao mesmo tempo.

3.3 - Alternância explícita

Essa solução cria uma variável do tipo inteiro turn que controla quem tem a vez para se entrar em na região crítica.

Para um processo é definido como 0 seu momento para entrar na zona crítica e para o outro é definido 1, e os dois se alternam trocando o valor de turn quando saem da zona crítica.

O problema dessa solução é que quando um processo é muito lento e o outro é muito rápido, terá casos onde os dois processos estarão fora de suas zonas críticas. Terá vezes na qual o processo mais rápido tenha que esperar o processo lento terminar sua execução fora da zona crítica para então entrar na execução da zona crítica e só depois que terminar sua execução, alterar o turn para que o processo mais rápido faça a sua execução de zona crítica.

3.3 - Solução de Peterson

É um algoritmo que consiste em duas rotinas escritas em ANSI C. Antes de usar variáveis compartilhadas, cada processo chama enter_region com o seu número de processo como parâmetro. Essa chamada fará que ele espere até que seja seguro entrar. Após terminar com as variáveis compartilhadas, o processo chama leave_region para indicar que terminou.

Essa solução funciona caso dois processos chamem enter_region simultâneamente, apenas o último a armazenar na variável que importa, pois o primeiro será sobrescrito e perdido. Assim não ocorrerá de dois processos acessem a região crítica.

3.4 - A instrução TSL

Nesse método, se utiliza a instrução TSL (Test and Set Lock) e uma variável compartilhada, lock para dizer quem pode acessar a memória compartilhada. Quando lock é 0, qualquer processo que precisar utilizar a região crítica pode configurá-lo para 1 usando a instrução TSL. Ao terminar ele configura lock novamente para 0 utilizando uma instrução move.

Essa instrução TSL tem uma sub-rotina de quatro instruções. A primeira copia o valor antigo de lock para o registrador e configura lock para 1. A segunda compara o valor antigo a 0. Caso não seja 0, o programa volta para o início e testa novamente. Quando ela virar 0, ele continua com a sua execução normal.

4 - Dormir e acordar

É muito custoso manter um processo testando por um longo tempo se é permitido acessar uma região crítica, logo, bloquear um processo durante esse período, auxilia a não desperdiçar tempo da CPU. Um modo simples é utilizar o par sleep e wakeup que são chamadas de sistema que suspendem um processo e que desperte um processo, respectivamente.

4.1 - o problema do produtor-consumidor

Esse problema acontece quando dois processos compartilham de um buffer de tamanho fixo com uma variável count para controlar o máximo de itens do buffer. Um dos processos que compartilham o buffer é chamado de produtor, ele insere informações no buffer. O outro processo é chamado de consumidor, ele retira informações do buffer.

Quando o produtor quer colocar um novo item no buffer e ele está lotado, a solução é colocar o produtor em estado de sleep até que o consumidor retire alguns itens do buffer. Ou o inverso, caso o buffer esteja vazio e o consumidor queira retirar um item dele, ele é posto para dormir até que o produtor coloque um ou mais itens nele.

Essa abordagem pode levar a ocorrer um problema semelhante ao de variáveis do tipo trava, mas com algumas particularidades. Esse erro irá ocorrer por que o acesso a counter não é restrito, ou seja, mais de um processo pode utilizá-lo ao mesmo tempo, ocorrendo a seguinte situação. O buffer está vazio e o consumidor lê o valor de count para ver se é 0, nesse momento o escalonador para de executar o consumidor temporariamente e executa o produtor, incrementando o valor de count, que agora é 1, e chama wake up para o consumidor que não estava dormindo. Ao voltar a executar o consumidor, a leitura recebida anteriormente para count é 0 e ele executa a chamada sleep até que o produtor o acorde. Com o passar do tempo o produtor entrará em sleep também pois o buffer estará cheio e os dois processos estarão bloqueados. O inverso pode ocorrer também.

5 - Semáforos

Para evitar o problema anterior de se perder o sinal para acordar o processo que deveria estar dormindo, foi proposta a utilização de semáforo. Um semáforo tem duas operações: down e up.

A operação down confere se o valor é maior do que 0, se ela for, ele decrementa, gastando um sinal de acordar armazenado, e continua o processo, se o valor for 0, o processo é colocado para dormir. Conferir o valor, modificá-lo e dormir são feitos como uma única ação atômica, garantindo que uma vez que a operação de semáforo tenha iniciado, nenhum outro processo pode acessar o semáforo até que a operação tenha sido concluída ou bloqueada.

A operação up incrementa o valor de um processo caso ou um mais processos estiverem dormindo naquele semáforo.

6 - mutex

Um mutex é uma variável compartilhada que pode estar em um de dois estados: destravado ou travado. É uma técnica de sincronização que possibilita assegurar o acesso exclusivo (leitura e escrita) a um recurso compartilhado por duas ou mais entidades. Utilizada na prevenção de problema de condição de disputa em regiões críticas.

Quando um thread (ou processo) precisa de acesso a uma região crítica, ele chama mutex_lock. Se o mutex estiver destravado naquele momento (significando que a região crítica está disponível), a chamada seguirá e o thread que chamou estará livre para entrar na região crítica. Por outro lado, se o mutex já estiver travado, o thread que chamou será bloqueado até que o thread na região crítica tenha concluído e chame mutex_unlock.

6.1 - Requisitos:

1. Nunca duas entidades podem estar simultaneamente em suas regiões críticas.

2. Deve ser independente da quantidade e desempenho dos processadores
3. Nenhuma entidade fora da região crítica pode ter a exclusividade desta
4. Nenhuma entidade deve esperar eternamente para entrar em sua região crítica

7 - Monitores

Um monitor é um conjunto de procedimentos, variáveis e estruturas de dados, todas agrupadas em um módulo especial. Somente um processo pode estar ativo dentro do monitor em um instante. O monitor organiza quais processos devem acessar os recursos, apenas um por vez. A característica mais importante do monitor é a exclusão mútua automática entre os seus procedimentos. Basta codificar as regiões críticas como procedimentos do monitor e o compilador irá garantir a exclusão mútua.

Variáveis de condição " são tipos de dados especiais dos monitores " são operadas por duas instruções Wait e Signal !

Wait(C): suspende a execução do processo, colocando-o em estado de espera associado a condição C

Signal(C): permite que um processo bloqueado por wait(C) continue a sua execução. Se existir mais de um processo bloqueado, apenas um é liberado.

Se não existir nenhum processo bloqueado, não faz nada.

8 - Troca de mensagens

Quando é necessário trocar informações entre processos que não compartilham memória.

Usados para comunicação e sincronização. Os processos cooperativos podem fazer uso de um buffer para trocar mensagens através de duas rotinas: send(destino, mensagem) e receive(origem, mensagem).

Quando um send é executado existe a possibilidade de bloquear ou não o processo até que a mensagem seja recebida no destino..

Quando o processo executa um receive existem duas possibilidades: se a mensagem já foi enviada o processo a recebe e continua a sua execução. Se a mensagem ainda não foi enviada (não foi executado um send() por alguém) o processo é bloqueado até que a mensagem chegue ou o processo continua a executar e abandona a tentativa de recebimento .

8.1 - Comunicação direta

Entre dois processos exige que, ao enviar ou receber uma mensagem, o processo enderece explicitamente o nome do processo receptor ou transmissor. Uma característica deste tipo de comunicação é só permitir a troca de mensagem entre dois processos.

8.2 - A comunicação indireta

Entre processos utiliza uma área compartilhada, onde a mensagens podem ser colocadas pelo processo transmissor e retiradas pelo receptor. Esse tipo de buffer é conhecido como mailbox ou port, e suas características, como identificação e capacidade de armazenamento de mensagens, são definidas no momento de criação.

9 - Barreiras

Quando todos os processos precisam alcançar um mesmo estado, antes de prosseguir eles impõem uma barreira no fim de cada fase. Processos que alcançam a barreira são bloqueados e os processos são liberados quando todos alcançam a barreira.

10 - Leitura-cópia-atualização

Não permitir a leitura de dados durante a atualização. Leitor acessa dados anteriores até alteração completa.