

Resumo Completo

Conceitos e Definições

Seções críticas: Trechos de código que acessam objetos compartilhados por mais de um fluxo de execução.

Como tratar seções críticas?

R: Devem ser transformadas em ações atômicas de modo que sua execução não possa ocorrer concorrentemente com outra seção crítica que refereça a mesma variável.

Corrida de dados: Precisa haver 2 ações conflitantes em threads distintas com as seguintes cond.:

- Mais de um acesso simultâneo a mesma localização de memória.
- Pelo menos um desses acessos é de escrita. Acessos de leitura não interferem entre si, mas os de escrita são exclusivos.
- Pode ocorrer concorrentemente / simultaneamente, nenhuma ordem é garantida.
- Pelo menos uma das ações não é atômica ou não está protegida por sincronização.

Condição de corrida: Ocorre quando o resultado de uma operação depende da ordem em que duas ou mais threads acedem e manipulam um recurso compartilhado.

Violações

Violação de atomicidade: Decorrente da violação da sequência desejada entre múltiplas operações de acesso à memória. Ou seja, quando uma região de código deve ser atômica, mas a atomicidade não é garantida em tempo de execução. Portanto, as operações que esperavam ser atômicas, quando decompostas em instruções de máquinas, acabam se entrelaçando produzindo um comportamento inusitado.

Atomicidade: Propriedade que garante que quando um fluxo modifica o estado da aplicação, um outro fluxo só consegue observar o estado anterior à operação, ou o estado resultante da operação.

Violação de ordem: Ocorre quando a ordem de execução desejada entre duas operações de acesso à memória é invertida. A ordem correta das operações não é garantida em tempo de execução por mais que exista uma precedência clara entre elas.

Tipos de Sincronização

Sincronização por exclusão nítua: Tipo de sincronização que visa garantir que os trechos de código das threads os quais acessam variáveis compartilhadas não sejam executados simultaneamente.

Solução genérica da exclusão nítua: Agrupar sequências contínuas de ações atômicas de hard-

ware em negócios críticos de software.

Abordagens p/ implementar sincronização por exclusão mútua:

1. Sincronização por espera ocupada: Thread continuamente testa o valor de uma variável até que esse valor lhe permita executar a seção crítica com exclusividade.

Ex. Questão 11 do Estudo dirigido - Abordagem de Peterson.

Em quais casos isso é válido? Quando ..

.. Não há nada melhor p/ CPU fazer enquanto aguarda.

.. Tempo de espera é menor que o tempo exigido p/ troca de contexto.

2. Sincronização por escalonamento (visual): A thread é bloqueada temporariamente enquanto a CPU é cedida a outra thread, evitando desperdício de recursos.

Sincronização por condições: Busca garantir que uma thread seja retardada / bloqueada até que uma condição lógica da aplicação seja satisfeita. A solução é implementada bloqueando a execução de uma thread até que o estado da aplicação seja correto p/ sua execução.

↳ Fluxos em execução trocam informações → Suspensão / Retomada da execução em função do estado da aplicação.

Implementação da Sincronização por condições com variáveis de condições: Utilização de variáveis especiais conhecidas como variáveis de condições.

↳ Permite com que threads aguardem (bloqueando-se) até que sejam sinalizadas por outras threads que a condição lógica foi atendida.

↳ Três operações principais:

Wait (condvar, mutex): Bloqueia a thread na fila da variável de condição.

Signal (condvar): Desbloqueia a thread na fila da variável de condição.

Broadcast (condvar): Desbloqueia todas as threads na fila da variável de condição.

Perguntas:

1. O que é sincronização por condição? Já foi respondido.

2. Como funciona o mecanismo de sincronização por condição com vars condicionais na biblioteca Pthread?

O mecanismo de sincronização por cond. com variáveis cond. funciona por meio da troca de mensagens entre as threads ou quais, dependendo do estado da aplicação, bloqueiam-se ou sinalizam outras threads p/ retomarem a execução.

3. Por que o código da função barreira usou uma variável de lock? O código da função barreira acessa uma região crítica que, por definição, referencia uma localização na memória compartilhada por mais de um fluxo de execução para checar o estado da aplicação.

Tipos de Distribuição de Carga entre Threads

Distribuição calculada nas threads.

1. Pularindo de nthreadz em nthreadz:

```
for (int i = t->id; i < t->dime; i += t->nthreads);
```

mais ruim p/ mais threads

Não é apropriado p/ vetores de tamanho médio → grande.

2. Calculando o chunk:

```
int base = t->dime / t->nthreads;
```

```
int reme = t->dime % t->nthreads;
```

```
int chunk = base + ((t->id % reme) > 0);
```

```
int start = t->id * base + (id < reme) ? id : reme;
```

mais custoso, porém mais eficiente
p/ vetores de tam. médio-grande.

id == reme	id < reme	id > reme
reme	id	reme