Sistemas Operacionais I

Transactional Memories

Prof. Leandro Marzulo

Definição

- Uma nova arquitetura para multiprocessadores que objetiva tornar a sincronização livre de bloqueios tão eficiente (e fácil de usar) quanto técnicas convencionais baseadas em exclusão mútua. [Herlihy e Moss, 1993]

Conceito

- O programador define regiões atômicas no código – Transações
- Transações são executadas em paralelo, de forma especulativa
- Cada transação possui um contexto local ou um histórico de modificações.
- Ao final da execução da transação, se não forem encontrados conflitos, as informações são persistidas – *Commit*
- Caso contrário, é feito o *Rollback*

Exemplo de Conflito

Thread 1	Thread 2	Conflito
Load(a)		Thread 2 leu um valor errado
a++		
	Load(a)	
Store(a)		
	a--	
	Store(a)	
Load(a)		Thread 1 leu um valor errado
	Load(a)	
	a--	
	Store(a)	
a++		
Store(a)		

Contexto e Conflitos

- Contexto de Entrada – Todas as posições de memória lidas por uma transação
- Contexto de Saída – Todas as posições de memória escritas por uma transação
- Pode haver conflito entre duas threads quando:
  - A interseção entre os seu conjuntos de escrita não é vazia
  - A interseção entre o conjunto de leitura de uma e o conjunto de escrita da outra não é vazia
- A interseção entre os conjuntos de leitura não precisa ser vazia – Todos podem ler as mesmas variáveis sem problemas
- Semelhante ao problema dos leitores escritores – com múltiplas variáveis (posições de memória)

Preocupações com a implementação

- Versionamento de Dados
- Detecção de Conflitos
- Aninhamento
- Virtualização
  - Tempo
  - Espaço
  - Aninhamento

Versionamento de Dados

- Versionamento adiantado (*eager versioning)*
  - Histórico - *Undo Log*
  - A memória global é modificada diretamente
  - Descartado no *Commit* (*commit* rápido)
  - Utilizado no *Rollback* para restaurar a memória global (*rollback* lento)
- Versionamento tardio (*lazy versioning)*
  - *Buffer* de escrita (memória local)
  - Utilizado no *Commit* (*commit* lento) *para persistir os dados*
  - Descartado no *Rollback* (*rollback* rápido)

Detecção de Conflitos

- Detecção de conflitos adiantada (*eager conflict detection*) *ou* pessimista
  - Detectado no instante em que uma posição de memória é acessada.
  - Evita que a transação continue executando desnecessariamente
  - Mas e se a outra transação abortar depois por conflitos com uma terceira?
- Detecção de conflitos tardia (*lazy conflict detection*) ou otimista
  - Detecção no *commit*

Aninhamento

- Transações poder ser usadas dentro de funções e até mesmo de bibliotecas.
- Programas que usam estas bibliotecas podem ter transações e chamarem funções que já possuem transações

Soluções para o aninhamento

- *Flatenning*
  - Transações filhas se tornam parte da pai
  - Se a filha aborta, o pai também aborta
  - Maior probabilidade de ocorrerem conflitos
- Aninhamento Fechado
  - Transações filhas abortam sem afetar a transação pai.
  - Na ocorrencia de um *commit, o contexto da transação são* mesclados ao da transação pai

Soluções para o aninhamento

- Aninhamento Aberto
  - Efetivação e cancelamento independentes de transações aninhadas
  - Se uma transação é abortada e alguma de suas filhas sofreu o *commit, medidas compensatórias devem ser tomadas*.

Virtualização

- Tempo
  - Transações podem ser tão grandes que sempre irão exceder o quantum ou sempre serão interrompidas.
  - Estado das transações deve fazer parte do contexto das threads e processos.
- Espaço
  - Estrutruturas que armazenam o estado das transações em hardware podem não ser grandes o suficiente para certas transações.
  - Mecanismos que salvam parte do contexto na memória

Virtualização

- Aninhamento
  - Estrutruturas que armazenam o estado das transações em hardware possuem um limite no número de níveis de aninhamento.
  - Uso da memória

Implementação

- Hardware
  - Alto Desempenho
  - Limitações (Virtualização) - Complexidade para implementar
- Software
  - Flexibilidade para implementar mecanismos mais complexos
  - Desempenho Menor
- Híbridas
  - O Melhor dos dois mundos !?

Trabalho 2

- Fazer um daemon que monitore um diretório (passado como argumento de linha de comando) e leia novos arquivos com a extensão .in.ready
- O usuário copiará arquivos para dentro deste diretório usando um script (shell) que copia o aquivo com a extensão .in e depois altera a extensão para .in.ready. Isto garante que arquivos não sejam acessados pelo deamon antes da cópia terminar.
- O daemon irá manter uma lista de arquivos a serem processados. Cada arquivo admitido deverá ser renomeado para a extensão .in.admited.

Trabalho 2

- Cada arquivo contém 2 matrizes quadradas de ordem 10. As colunas em uma linha são separadas por vírgulas e quebras de linha iniciam novas linhas.
- O deamon terá as seguintes classes de threads:
  - Monitoramento de diretório – para fazer a leitura das matrizes.
  - Multiplicador – Para multiplicar as matrizes
  - Determinante – Para calcular o determinante da matriz
  - Escritor de Resultado – Para gerar arquivo de saída contendo as matrizes originais, o resultado da multiplicação e do determinante (tudo em um só arquivo, com o mesmo nome do arquivo de entrada, com a extensão .out)
- As threads terão, nesta ordem, uma relação de pipelining, com buffer de passagem de tarefas ente cada par de estágios do pipe. Podemos der múltiplas threads de cada tipo, número inicial definido por constantes.

Trabalho 2

- Fazer syslog de todo o sistema, inclusive no inicio e fim de cada estágio, informando sempre o nome do arquivo de origem que está sendo trabalhado.
- Fazer a parte de monitoramento de sinais. Em caso de recebimento de um sinal SIGTERM, o programa deve parar de monitorar novos arquivos e só terminar depois de finalizar todas as tarefas existentes. Em caso de SIGKILL, logar o evento e finalizar a aplicação fechando os arquivos abertos para que não sejam corrompidos. Você deve fazer isso através de variáveis compartilhadas com todas as threads dos estágios.

Trabalho 2

- Criar um programa de controle do deamon onde um menu é exibido ao usuário para que o mesmo possa alterar a quantidade de threads de cada tipo. O deamon recebe a mensagem, cria ou destrói threads e responde com o status da solicitação (OK, NOK, MAX)
  - OK – sucesso
  - NOK – Erro
  - MAX – numero de threads solicitado maior que o máximo definido – então serão criadas até MAX threads
- Se o número de threads for menor que o atual, threads serão avisadas para finalizar o trabalho corrente, não buscar mais nada no buffer e finalizarem
- O programa de controle também permite desligar o deamon, enviando SIGTERM para o mesmo
- Alterar o diretório de entrada e saída
- Alterar o mapa de afinidade para cada classe de threads