Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 07 - Escalonamento multi CPU

Aula de Hoje

Aula de Hoje

Na aula de hoje veremos alguns conceitos sobre escalonamento de processos em ambientes com mais de um processador (multiprocessados).

Nos dias de hoje, praticamente qualquer PC (e boa parte dos celulares!) tem processadores com múltiplos cores (varios processadores no mesmo chip)

Esses cores novos não são triviais de utilizar: Seu programa em C tradicional só usa uma CPU; adicionar novas não vai afetá-lo diretamente. Para resolver isto será necessário rodar esta aplicação em **paralelo**, o que estudaremos no terceiro módulo deste curso.

Contudo, antes disso precisamos entender como o SO faz para alocar os processos em várias CPUs ao mesmo tempo. Todo escalonamento que vimos até agora era para uma CPU. Como extender estas idéias para este novo cenário?

Aula de Hoje

Arquitetura Multiprocessada

Antes de irmos ao escalonamento, é importante entender as diferenças no hardware entre os sitemas mono e multiprocessados.

A principal diferença reside nas caches, e em como os dados são compartilhados entre os diferentes processadores.

Vamos ver isto em alto nível, já que esta não é uma aula de arquitetura ;-)

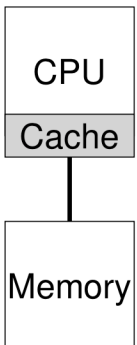
Monoprocessados

Em sistemas monoprocessados, existe uma hierarquia de caches de memória para ajudar os programas a serem mais rápidos.

Caches são memórias mais rápidas e menores, que servem para armazenar cópias de dados *populares*.

A primeira vez que uma informação é acessada, ela é inserida na cache, fazendo acessos subsequentes a este dado muito mais rápidos.

Localidade **Espacial** e **Temporal**.



Relembrando Localidade

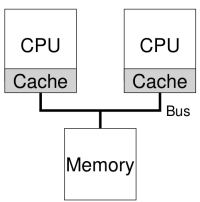
Localidade temporal: Se um dado foi acessado, existe uma boa chance dele ser referenciado novamente em um tempo próximo: Variáveis em um bloco de código, loops.

Localidade espacial: Variáveis próximas (em termos de endereços de memória) tem mais chances de serem acessadas juntas: Em vetores e estruturas, geralmente quando se acessa um dado, outros dados próximos podem ser acessados na sequência.

Todo o subsistema de cache é feito pensando em se aproveitar da localidade para acelerar o processamento.

Mas como fica isso quando existe mais de uma CPU?

Múltiplas CPUs

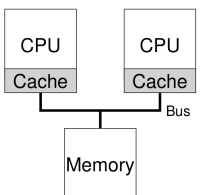


Aí fica Muito mais complicado .

Cada CPU tem uma cache própria (isso pode ser ainda mais complicado dependendo da arquitetura).

Imaginem o seguinte: Um programa quer alterar o valor de uma variável saldo.

Múltiplas CPUs

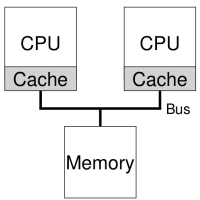


Aí fica Muito mais complicado .

Cada CPU tem uma cache própria (isso pode ser ainda mais complicado dependendo da arquitetura).

Imaginem o seguinte: Um programa quer alterar o valor de uma variável saldo.
Imagine agora que o programa termina o quantum e, no próximo escalonamento ele é colocado na CPU2

Múltiplas CPUs

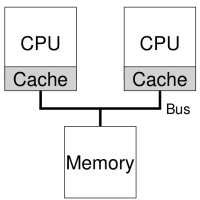


Aí fica Muito mais complicado .

Cada CPU tem uma cache própria (isso pode ser ainda mais complicado dependendo da arquitetura).

Imaginem o seguinte: Um programa quer alterar o valor de uma variável saldo.
Imagine agora que o programa termina o quantum e, no próximo escalonamento ele é colocado na CPU2
E que ele altere o valor da variável saldo **que está no seu saldo**.

Múltiplas CPUs



Aí fica Muito mais complicado .

Cada CPU tem uma cache própria (isso pode ser ainda mais complicado dependendo da arquitetura).

Imaginem o seguinte: Um programa quer alterar o valor de uma variável saldo.
Imagine agora que o programa termina o quantum e, no próximo escalonamento ele é colocado na CPU2
E que ele altere o valor da variável saldo **que está no seu saldo**.
Agora imagine isso com **write-back**.

Coerência de Cache

Este problema é conhecido como coerência de cache, e existe vasta literatura descrevendo os pequenso detalhes e tretas aí.

As soluções mais básicas são feitas via hardware:

- Bus Snooping: A cache fisicamente monitora o dado que ela tem para achar atualizações no dado na memória principal.

Não funciona tão simples com write-back, mas vocês conseguem imaginar como podem haver hardwares para ajudar com isso.

Afinidade de Cache

Um outro problema que surge em escalonadores multiprocessados é a afinidade de cache.

Quanto mais um processo em uma CPU, mais o estado da cache/TLB vai estar favorável para ele.

Na próxima vez que ele for rodar, é mais proveitoso que ele rode na mesma CPU. Se rodar em uma CPU diferente, toda a vantagem das caches será desperdiçada.

Aula de Hoje

Fila Única

Abordagem mais direta: usa uma única fila de escalonamento para todas as CPUs.

- O algoritmo se mantém igual, selecionando para as N CPUs usando o mesmo critério.
- SQMS: Single Queue Multiprocessor Scheduling.

Vantagem: Simples de implementar. Basta usar o mesmo algoritmo de escolher o próximo toda vez que uma CPU estiver livre.

Fila única

Problemas:

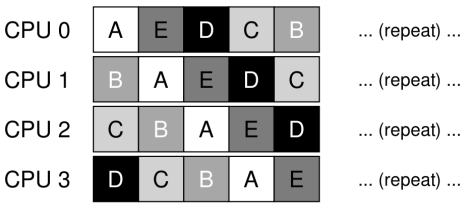
- **Escalabilidade**: Como veremos mais adiante (em concorrência), o código que escolhe o próximo processo só pode atender uma CPU por vez (através do uso de travas). Se duas ou mais CPUs ficam prontas, elas devem esperar pelo escalonador servir uma por uma.

- **Afinidade de cache**.

Afinidade de Cache no SQMS

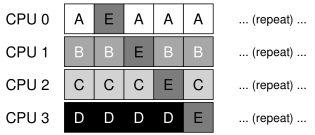
Imagine um caso com 5 processos em 4 processadores. A fila poderia ser algo assim:

Assumindo um caso simples, em que cada processo roda pelo mesmo tempo e rola um round robin, pode acontecer o cenário ao lado:



Solução

A maioria dos escalonadores SQMS tenta adicionar algum mecanismo de afinidade. Por exemplo, alguns processos podem ter preferência por algumas CPUs enquanto outros não:



SQMS tem vantagens e desvantagens: É simples de implementar, mas tem problemas de escala devido a sincronização da fila e não ser simples de presenrvar a afinidade.

Aula de Hoje

Escalonamento Multi-Filas

Uma solução mais complexa é usar várias filas (p.ex. uma por CPU). MQMS - Multi-queue multiprocessor Squeduling.

Nele, temos múltiplas filas de escalonamento, cada uma com seu algoritmo (igual ou não).

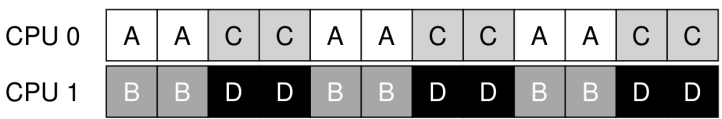
Quando um processo novo surge, ele é adicionado a alguma fila de acordo com uma heurística qualquer.

A partir daí, o escalonamento em cada fila é feito individualmente, evitando os problemas de sincronização dos SQMS.

Exemplo

Um sistema com duas CPUs e quatro processos:

Cada CPU agora pode ter seu próprio escalonamento. Imagine um Round Robin:



Que problemas podem surgir desse escalonamento?

Exemplo

Desbalanceamento de carga. Imagine o mesmo cenário, mas o processo C termina. As filas ficam:

Vamos ver como fica um round robin nesse cenário:

Exemplo

Pior ainda, imagine que logo após isso, A acabe também:

Se isso ocorrer, a CPU0 ficará inutilizada!

Migração

Como resolver isso?

Migração

Como resolver isso?

Migrando processos de uma fila para a outra.
Vamos voltar ao exemplo em que a CPU0 está com a fila vazia:

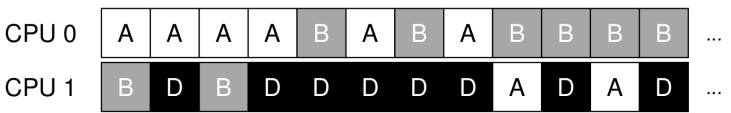
O que desejamos aqui é simples, que um processo de Q1 vá para Q0.

Migração

E o caso anterior, onde uma fila tem 1 e a outra 2?

Neste caso, a migração não resolve. O que fazer?

Uma saída é a migração contínua. Ou seja, os processos de tempos em tempos trocam de CPU para manter o balanceamento:



Migração

Obviamente, existem vários padrões para a migração de processos.

Uma abordagem básica é o roubo de trabalho. Uma fila mais vazia pode observar as outras filas, procurando alguma com muito mais processos e roubar algum pra ela.

Obviamente, esta abordagem tem prós e contras: Quanto mais uma fila checar as outras, mais overhead é gerado no sistema. Em contrapartida, se uma fila demorar demais para olhar nas outras, ela pode ficar ociosa de forma desnecessária.

Resumindo: Vocês tem sorte de que nesta disciplina não entramos a fundo nesses escalonamentos.

Finalizando

Vimos duas abordagens para escalonamento multi processador:

- SQMS, que é simples de implementar e faz um bom balanceamento de carga, mas gera atrasos de sincronização e dificuldades na afinidade de cache.
- MQMS, que escala melhor e é naturalmente melhor na afinidade de cache, mas que pode gerar desbalanceamento de carga na CPU.

"Ah professor pra que estudar isso?"Em sistemas Web grandes, é super normal fazer o balanceamento de requisições entre diversos servidores (eu mesmo já tive que implementar isto)