Sistemas para Multiprocessadores

Introdução

A computação com multiprocessadores, onde um único sistema computacional possui duas ou mais Unidades Centrais de Processamento (CPUs) que compartilham acesso à memória principal e periféricos, tornou-se o padrão na maioria dos domínios da computação, desde servidores e desktops até dispositivos móveis. Os sistemas operacionais para multiprocessadores enfrentam desafios únicos e oferecem oportunidades significativas para aumentar o desempenho, a vazão e a confiabilidade em comparação com sistemas monoprocessadores. Este capítulo explora o hardware de multiprocessadores, os diferentes tipos de sistemas operacionais projetados para eles, e as questões críticas de sincronização e escalonamento de processos/threads em ambientes multiprocessados. Compreender esses aspectos é fundamental para projetar e utilizar eficientemente o poder do hardware paralelo moderno.

Hardware de Multiprocessadores

Os sistemas multiprocessadores podem ser classificados com base em como a memória é compartilhada e como os processadores são interconectados.

- **Multiprocessamento Simétrico (Symmetric Multiprocessing - SMP):**
  - É a arquitetura mais comum para multiprocessadores modernos. Todas as CPUs são tratadas como iguais, e qualquer CPU pode executar qualquer tarefa, incluindo código do sistema operacional e processos de usuário.
  - Todas as CPUs compartilham a mesma memória principal e dispositivos de E/S, geralmente através de um barramento comum ou uma interconexão mais complexa (e.g., crossbar switch, NUMA interconnect).
  - Cada CPU pode ter seu próprio cache de alta velocidade (L1, L2), e pode haver um cache compartilhado (L3) entre algumas ou todas as CPUs. Manter a coerência entre esses caches (cache coherence) é um desafio de hardware significativo.

- **Multiprocessamento Assimétrico (Asymmetric Multiprocessing - AMP):**
  - Uma CPU (mestre) controla o sistema, enquanto as outras CPUs (escravas) executam tarefas atribuídas pelo mestre ou código de usuário predefinido. Menos comum em sistemas de propósito geral hoje em dia.
- **Arquiteturas Multicore:** Uma forma de SMP onde múltiplos núcleos de processamento (cores) são integrados em um único chip de CPU. Cada núcleo aparece para o sistema operacional como uma CPU lógica separada.
- **Arquiteturas NUMA (Non-Uniform Memory Access):**
  - Em sistemas SMP maiores, o acesso à memória pode se tornar um gargalo. Em arquiteturas NUMA, a memória é fisicamente distribuída entre os processadores (ou grupos de processadores). Cada processador pode acessar sua própria memória local (nó de memória) mais rapidamente do que a memória local de outros processadores (memória remota).
  - O SO precisa estar ciente da topologia NUMA para otimizar a alocação de memória e o escalonamento de threads (e.g., tentar alocar memória para uma thread no mesmo nó NUMA onde ela está executando).
- **Coerência de Cache (Cache Coherence):** Em sistemas com caches privados por CPU, se múltiplas CPUs armazenam em cache a mesma localização de memória, uma modificação feita por uma CPU em seu cache deve ser propagada para os outros caches (e para a memória principal) para garantir que todas as CPUs vejam uma visão consistente da memória. Protocolos de coerência de cache (e.g., MESI) são implementados em hardware para gerenciar isso.

Tipos de Sistemas Operacionais para Multiprocessadores

O design do sistema operacional deve ser adaptado para explorar eficientemente o hardware multiprocessador.

1. **Cada CPU com seu Próprio SO (Master-Slave ou Assimétrico):**
   - Uma CPU atua como mestre, executando o SO e gerenciando as outras CPUs (escravas) que executam apenas código de usuário. Simples de implementar a partir de um SO monoprocessador, mas o mestre pode se tornar um gargalo e o sistema não é verdadeiramente simétrico.
2. **SO Monolítico Simétrico (Symmetric Operating System):**
   - O kernel do SO pode ser executado em qualquer CPU. Todas as CPUs são tratadas de forma igual. Esta é a abordagem mais comum e flexível.
   - **Desafios:** Requer mecanismos de sincronização cuidadosos dentro do kernel para proteger estruturas de dados compartilhadas do acesso concorrente por múltiplas CPUs (e.g., spinlocks, mutexes do kernel).
   - **Granularidade do Locking:**
     - **Big Kernel Lock (BKL):** Uma abordagem mais antiga onde um único lock global protege todo o kernel. Apenas uma CPU pode estar executando código do kernel por vez, limitando o paralelismo do kernel.

- **Fine-Grained Locking:** Múltiplos locks protegem diferentes partes do kernel ou diferentes estruturas de dados. Permite maior concorrência dentro do kernel, mas é mais complexo de implementar e depurar.

Sincronização em Multiprocessadores

A sincronização é ainda mais crítica em sistemas multiprocessadores do que em monoprocessadores, pois múltiplas CPUs podem tentar acessar e modificar dados compartilhados simultaneamente, levando a condições de corrida no kernel ou em aplicações multithreaded.

- **Spinlocks:** Um tipo de lock onde uma thread que tenta adquirir um lock já ocupado simplesmente "gira" em um loop ocupado (busy-waiting), testando repetidamente o lock até que ele se torne disponível. Eficientes para seções críticas muito curtas, onde o tempo de espera esperado é menor que o custo de bloquear e reescalonar a thread. Em sistemas monoprocessadores, spinlocks não fazem sentido (a thread girando impediria a liberação do lock). Em multiprocessadores, a thread que detém o lock pode estar executando em outra CPU e liberá-lo em breve.
- **Mutexes e Semáforos:** Como em sistemas monoprocessadores, mas sua implementação deve ser segura para multiprocessadores (e.g., usando instruções atômicas de hardware para manipulação do estado do lock).
- **Barreiras:** Usadas para sincronizar múltiplas threads em um ponto específico, garantindo que todas as threads cheguem à barreira antes que qualquer uma possa prosseguir. Comum em aplicações paralelas.
- **Read-Write Locks:** Permitem acesso concorrente para múltiplos leitores, mas acesso exclusivo para escritores. Útil para estruturas de dados que são lidas com frequência, mas modificadas raramente.
- **Problemas de Sincronização Específicos de Multiprocessadores:**
  - **Falsa Compartilhamento (False Sharing):** Ocorre quando duas ou mais threads em CPUs diferentes acessam variáveis diferentes que, por acaso, residem na mesma linha de cache. Mesmo que as threads não estejam compartilhando dados logicamente, as operações de coerência de cache podem invalidar a linha de cache para outras CPUs, degradando o desempenho.
  - **Escalabilidade dos Locks:** À medida que o número de CPUs aumenta, a contenção por locks pode se tornar um gargalo significativo. Técnicas como locks por CPU, algoritmos de locking sem espera (lock-free) ou read-copy-update (RCU) são usadas para melhorar a escalabilidade.
- **Read-Copy-Update (RCU):** Um mecanismo de sincronização avançado usado no kernel do Linux. Permite que leitores acessem estruturas de dados compartilhadas sem adquirir locks, enquanto atualizadores criam uma cópia da estrutura, modificam a cópia e depois, atomicamente, publicam a nova versão. Os leitores que estavam usando a versão antiga continuam a usá-la até terminarem, e a

memória da versão antiga só é liberada após todos os leitores existentes terem concluído.

Escalonamento em Multiprocessadores

O escalonamento de threads em um sistema multiprocessador envolve decidir não apenas qual thread executar, mas também em qual CPU executá-la.

- **Afinidade de Processador (Processor Affinity):**
  - Tentar manter uma thread executando na mesma CPU o máximo possível. Isso é benéfico porque a thread pode ter dados em cache naquela CPU (cache quente), e movê-la para outra CPU exigiria que o cache da nova CPU fosse preenchido (cache frio), causando degradação de desempenho.
  - **Afinidade Suave (Soft Affinity):** O SO tenta manter a thread na mesma CPU, mas não garante.
  - **Afinidade Rígida (Hard Affinity):** Permite que um processo especifique em qual conjunto de CPUs suas threads podem executar.
- **Balanceamento de Carga (Load Balancing):**
  - Distribuir a carga de trabalho (threads prontas) de forma equilibrada entre todas as CPUs para maximizar a utilização e o paralelismo.
  - Pode entrar em conflito com a afinidade de processador (mover uma thread para outra CPU para balancear a carga pode invalidar seu cache).
  - **Push Migration:** Um processo periódico verifica a carga em cada CPU e, se encontrar um desequilíbrio, move (empurra) threads de CPUs sobrecarregadas para CPUs ociosas ou menos carregadas.
  - **Pull Migration:** Uma CPU ociosa puxa uma tarefa de uma CPU ocupada.
- **Escalonamento Consciente de NUMA (NUMA-Aware Scheduling):** Em sistemas NUMA, o escalonador tenta colocar uma thread na CPU que está no mesmo nó de memória onde os dados da thread estão alocados, para minimizar a latência de acesso à memória remota.
- **Escalonamento Gang (Gang Scheduling):** Usado para aplicações paralelas onde as threads cooperam intensamente. Tenta agendar todas as threads de um "gang" (grupo de threads cooperantes) para executar simultaneamente em diferentes CPUs. Se algumas threads do gang não puderem ser agendadas, as outras também podem ser bloqueadas para evitar que girem esperando pelas threads não agendadas.
- **Simultaneous Multithreading (SMT) / Hyper-Threading:** Uma técnica de hardware onde uma única CPU física pode manter o estado de múltiplas threads lógicas e alternar rapidamente entre elas (ou executar partes delas em paralelo se houver unidades de execução duplicadas). Do ponto de vista do SO, cada thread lógica aparece como uma CPU separada. O escalonador deve estar ciente do SMT para tomar decisões eficientes (e.g., preferir escalonar threads em CPUs físicas diferentes antes de usar múltiplas threads lógicas na mesma CPU física, para evitar competição por recursos dentro do mesmo núcleo).

Conclusão

Os sistemas operacionais para multiprocessadores são essenciais para explorar o poder do hardware paralelo moderno. Eles devem gerenciar eficientemente o acesso concorrente a recursos compartilhados através de mecanismos de sincronização robustos e escaláveis, como spinlocks e RCU, e devem implementar políticas de escalonamento inteligentes que considerem a afinidade de processador, o balanceamento de carga e as características da arquitetura subjacente (como NUMA e SMT). O design de um SO multiprocessador envolve um trade-off constante entre complexidade, desempenho e escalabilidade. À medida que o número de núcleos por chip continua a aumentar, os desafios e a importância do design eficiente de SOs para multiprocessadores só tendem a crescer.

Referências

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th ed.). Wiley.
- Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson Education.
- Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.
- McKenney, P. E. (2013). *Is Parallel Programming Hard, And, If So, What Can You Do About It?*. (Disponível online, foca em RCU e outros tópicos de concorrência)
- Love, R. (2010). *Linux Kernel Development* (3rd ed.). Addison-Wesley Professional. (Capítulos sobre sincronização e escalonamento no Linux)
- Russinovich, M. E., Solomon, D. A., & Ionescu, A. (2012). *Windows Internals, Part 1 & 2*. Microsoft Press. (Detalhes sobre multiprocessamento e escalonamento no Windows)