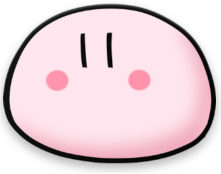


{ refactor by amtg }

resumo encontrado no PETLib + slides do prof Adriano

COMPUTERS ARE LIKE OLD TESTAMENT GODS;

lots of rules and no mercy

Listas e Resoluções

Lista Hierarquia Aqui
Lista E/S Multicore Aqui
Resolução da Lista Hierarquia Aqui
Resolução da Lista E/S Multicore Aqui

Black Market

EE 3 2017.1 [Resolvida]
EE2 20XX.2 [Resolvida]
"Resumo de Alguns Períodos Atrás"

Material Complementar

Todos os Mapeamentos de Cache, na Prática[Playlist]
O que é Memória Virtual? [Vídeo]
Aula de Memória Virtual da Carnegie Mellon University[Vídeo]

Hierarquia de Memória

A memória dos computadores atualmente é dividida de modo a formar uma hierarquia: Memórias mais rápidas e menores são as mais "importantes" e acessadas diretamente pelo processador. Conforme a distância do processador aumenta, a memória aumenta em tamanho, porém fica mais lenta.

A motivação para isso se deve ao **princípio da localidade** : dados que se encontram próximos temporalmente ou espacialmente tendem a serem usados novamente. Por isso, os **dados recorrentes são armazenados a curto prazo nas memórias menores e rápidas**, que obtém tais informações das memórias a longo prazo de menor hierarquia.

Vale lembrar: uma hierarquia de memória apenas "conversa" com outras de hierarquia imediatamente acima ou abaixo. Não é possível que a memória com menor hierarquia converse diretamente com o processador, por exemplo.

Outro motivo pelo qual se vale a pena hierarquizar é a relação custo/desempenho dos diferentes tipos de memórias que temos disponíveis:

SRAM (static random access memory) - Pequeno tempo de acesso (0,5 - 2,5 ms), sem necessidade de *refreshing*. Extremamente cara, mas absurdamente rápida. Com ela fazemos as caches.

DRAM (dynamic random access memory) - Maior tempo de acesso (50 - 70 ms). Preço médio, e velocidade razoável. Grande capacidade de integração (baixo custo por bit). Porém os dados não permanecem por muito tempo, sendo necessária a atualização constante (por isso dynamic). Com ela são feitas as memórias RAM do computador.

DISCO MAGNÉTICO - Extremamente barato, mas terrivelmente lento. Os dados permanecem mesmo quando o computador é desligado, mas se o disco fosse usado diretamente, o processador seria "subestimado": ele levaria mais tempo esperando o dado chegar do disco do que efetivamente processando.

Por mais rápido que o processador seja, a memória precisa tentar acompanhar essa performance.

Na prática, é o que vemos nos computadores do dia a dia: Uma cache geralmente de 4MB, uma memória RAM de 4GB a 16GB, e os discos magnéticos com 1TB para mais.

Memória Ideal

Tempo de acesso de uma SRAM, capacidade e custo de um disco

Definições de termos usados na Hierarquia de Memória

Bloco:

Menor unidade que é copiada da memória principal (RAM) para a cache: Pode ser uma ou várias palavras.

aka: unidade mínima de informação que pode estar presente ou não em uma cache.

Hit e Hit Ratio:

Se o dado já estiver presente na cache quando for necessário acessá-lo, chamamos de **hit**. Podemos medir o **hit ratio** dividindo o número de hits pela quantidade de acessos que realizamos; já **hit time** é o tempo para acessar dado no nível desejado.

Miss e Miss Ratio:

Se o dado não estiver presente na cache quando for necessário acessá-lo, chamamos de **miss**. Podemos medir o **miss ratio** dividindo o número de misses pela quantidade de acessos que realizamos ou (1 - hit rate).

Miss Penalty:

Porém, se o dado está ausente, precisamos *buscá-lo*, e isso *leva tempo*. A isso damos o nome de **miss penalty**, ou seja, é a penalidade que você sofre quando não encontra o dado: Agora terá que buscá-lo da memória e esperar que ele chegue na cache.

tempo de acesso do bloco no nível mais abaixo
+
tempo de transmissão do bloco para nível mais acima
+
tempo de escrita do bloco no nível mais acima
+
tempo de transmissão para a CPU

Princípio da Localidade Espacial:

O programa irá usar em breve instruções e dados que estão próximos (na memória) aos que acabou de usar. Ex: dados de um array.
↬ Mova blocos de dados de palavras contíguas para junto do processador.

Princípio da Localidade Temporal:

O programa irá usar/referenciar em breve instruções e dados que usou recentemente. Ex: instruções de um loop.
↬ Mantenha itens mais recentemente referenciados junto ao processador.

Cache:

É uma memória de **acesso muito rápido**, mas de tamanho reduzido : uma SRAM menor que a memória principal DRAM. Localizada perto da CPU, não utiliza barramento comum aos outros dispositivos.
Utiliza-se do princípio da localidade, armazena dados acessados mais recentemente e de endereços mais próximos - serve para manter os dados mais recentes sendo usados pelo processador, e possui um controlador próprio que determina o que será transferido da memória principal para a cache.

Hierarquia da Memória (cont)

Vantagens

Tira proveito do princípio da localidade espacial e temporal: dados usados recentemente e dados próximos aos usados recentemente provavelmente serão usados de novo.

Otimiza as diferentes tecnologias que relacionam custo e tempo de acesso: memórias mais rápidas e menores são mais caras, e memórias grandes porém lentas são baratas. O que se deseja atingir com a hierarquia é o custo e tamanho de uma memória grande como o HD, mas ter o tempo de acesso de uma SRAM (memória do tipo mais caro).

Dado a essa otimização, o desempenho da CPU aumenta: as memórias não se desenvolveram no mesmo passo dos processadores, o que faz com que, sem um planejamento do sistema de memória, a capacidade do processador tenha um gargalo, sendo impedido de operar com máxima eficiência.

Desvantagens

Maior complexidade na implementação e custo elevado.

{ V ou F }

- ★ Caches se aproveitam da localidade temporal :
- ★ Em uma instrução de leitura, o valor retornado depende dos blocos que estão na cache :
- ★ Maior parte do custo de uma hierarquia de memória é da memória de nível mais alto :
- ★ Maior parte da capacidade de armazenamento em uma hierarquia de memória vem do nível mais baixo :

Layouts de Cache

Ao elaborar as memórias cache, era necessário estabelecer um modo de identificar se uma informação estava na cache e, caso ela estivesse, como encontrá-la. Para isso, alguns métodos foram desenvolvidos:

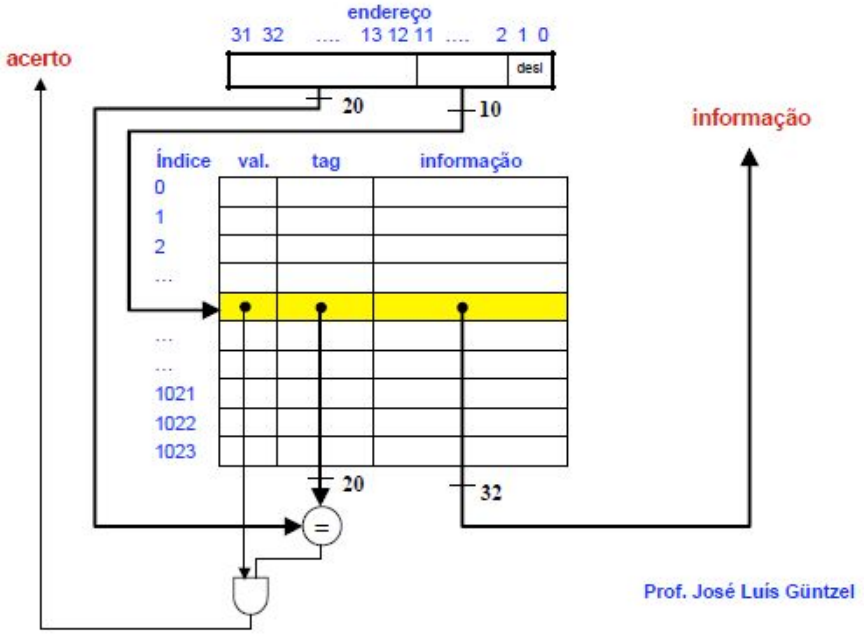
Mapeamento Direto

Para cada palavra na cache, atribuir um endereço com base no endereço da palavra na memória principal. A maioria das caches que usa mapeamento direto o faz usando o seguinte processo:

(Endereço ao byte/bytes por bloco) módulo (Número de blocos da cache)

Como mais de uma entrada da cache pode armazenar o conteúdo de mais de um endereço da memória principal, é utilizado uma *tag* em conjunto com o endereço do mapeamento, de modo a compor o endereço completo, com relação à memória principal. É utilizado também um *bit de validade* para identificar se a informação contida naquele slot da cache é válido.

Com isso, um exemplo de cache de mapeamento direto é dado a seguir:



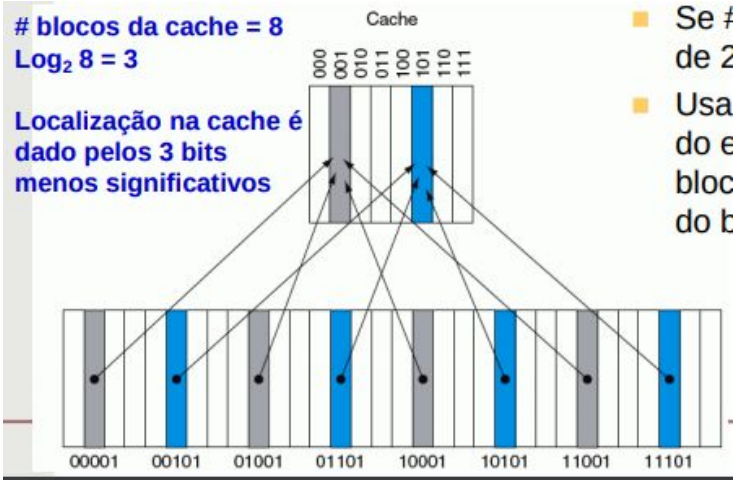
Para calcular o **tamanho de uma cache de mapeamento direto** é necessário conhecer o tamanho do bloco, tamanho da tag e tamanho do campo de validade:

2^n x (tamanho do bloco + tamanho do tag + tamanho do campo de validade)

Já o **layout do endereço de uma cache com mapeamento direto** contém a *tag*, o *índice (mostrado logo acima como calcular)*, o *block offset (indica a word referenciada dentro do bloco)* e o *byte offset (indica o byte dentro da word)*.

Ao calcular a quantidade de bits necessária para endereçar todas as words dentro de um bloco, calcula-se a quantidade de bits de um block offset. Ao calcular a quantidade de bits necessárias para endereçar todos os bytes dentro de uma word, calcula-se a quantidade de bits de um byte offset.

do slide:



No caso acima, os três bits menos significativos vão indicar a localização de bloco na cache; a **tag** é justamente os outros bits, os mais significativos que não indicam tal localização.
E existe também um **bit de validade** - *valid bit* - para saber se o conteúdo do bloco ainda é válido (como nos casos em que o computador é iniciado e e a cache ainda possui algum lixo, ou em fim de execução de programa).

Composição do endereço :
tag | índice do bloco | posição de byte dentro do bloco

Mas, caso cada bloco possa conter mais de uma palavra :
tag | índice do bloco | offset do bloco (qual palavra) | offset de byte

Associativa por Conjunto (Bloco Associativo)

Existe um número fixo de posições na cache (no mínimo duas) nas quais cada bloco pode ser colocado aka aloca "blocos de blocos", fazendo mapeamento direto para estes blocos maiores, e dentro deles catando o primeiro espaço livre. Uma cache n-associative é uma cache associativa por conjunto com n posições possíveis para guardar um bloco. Tal cache é composta por um certo número de conjuntos, cada conjunto com n blocos. Cada bloco da memória principal é mapeado para um dos conjuntos da cache, determinado pelo campo de índice do endereço (sendo que o bloco pode ser colocado em qualquer dos elementos desse conjunto). O aumento do grau de associatividade resulta, em geral, na redução da taxa de faltas. A desvantagem é o aumento do tempo de tratamento do acerto.

O conjunto que contém o bloco de memória é dado por

(Número do bloco) módulo (Número de conjuntos da cache)

Totalmente Associativa

Nesse mapeamento, um bloco da memória principal pode ser colocado em qualquer posição da cache aka vai catando um espaço livre, e para isto precisa percorrer toda a parte ocupada da cache, gasta muito tempo. Um bloco de memória pode ser associado a qualquer "entrada" da cache. É, então, necessário pesquisar todas as entradas da cache, a fim de localizar um determinado bloco (pois ele pode estar em qualquer uma das entradas). Para reduzir o tempo de pesquisa, ela é feita em paralelo, usando um comparador para cada entrada da cache.

O custo do hardware é alto: tal esquema só é atrativo para caches pequenas.

A **escolha** entre um **mapeamento direto**, **associativo por conjunto** ou **totalmente associativo** irá depender do custo de uma falta versus o custo de implementação da associatividade, tanto em termos de tempo quanto em termos de hardware.

Misses

Tipos de Misses

Compulsórios:

Impossíveis de tratar. São misses que obrigatoriamente acontecerão, pois são os que ocorrem quando a cache está inicialmente vazio. São também chamado de misses de “partida a frio”.

Conflito:

Misses que acontecem devido à disputa de dados para ficar no mesmo slot de cache. Não acontecem numa cache completamente associativa.

Capacidade:

Misses que acontecem porque, mesmo que não houvesse nenhum miss de conflito, seria fisicamente impossível colocar o dado sem retirar outro, pois a cache encontra-se cheia.

Como Lidar com Misses

Se a cache informar uma falta o processador deve ser parado (congelando o conteúdo de todos os registradores), um controlador separado ajuda no tratamento das faltas geradas no acesso à cache, comandando a busca da informação (bloco) na memória principal ou na cache de próximo nível. Uma vez que o dado tenha sido obtido, a execução é reiniciada no ciclo que gerou a falta no acesso a cache.

Misses de conflito podem ser tratados aumentando a associatividade da cache, porém isso faz com que ela fique significativamente mais cara: são necessários mais comparadores, e o hardware necessário torna-se mais complexo.

Já **misses de capacidade** podem ser tratados simplesmente aumentando o tamanho da cache.

Ocorrendo uma falta na cache:

(1) Congela o pipeline
(2) Busca bloco na memória com ajuda de controlador de HW que acessa memória e preenche cache.
Se o bloco da cache estiver preenchido com outro bloco, a informação é sobre-escrita

Caso a falta seja **referente a uma instrução**, reinicia a busca da instrução
Caso a falta seja **referente a um dado**, completa o acesso ao dado
Desempenho é penalizado

Penalidade por Falta

À medida que o tamanho do bloco aumenta, aumenta o custo de uma falta. A **penalidade por falta** é determinada pelo tempo necessário à busca de um bloco no nível imediatamente inferior na hierarquia, carregando-a na cache.

O tempo de busca é dividido em duas partes: a **latência** para a busca da primeira palavra e o **tempo de transferência** do resto do bloco.

Medidas do desempenho da cache

Tempo de processador

(ciclos de clock gastos na execução normal do programa + ciclos de clock gastos à espera do acesso ao sistema de memória)

x

tempo de clock

Em que :

Ciclos de clock gastos à espera do acesso ao sistema de memória =

ciclos de clock para tratar faltas de leitura da cache

+

ciclos de relógio para tratar faltas de escrita na cache

Ciclos de clock para tratar faltas na cache =

acessos à memória por programa

x

faltas por instrução

x

penalidade por faltas

Melhorias da Cache

Existem dois tipos de acesso :

Leitura

Simples de implementar

Escrita

Lento e complicado

Dado e tag são atualizados na cache, caracteriza-se por inconsistência entre memória principal e cache : se um bloco da cache foi alterado pela CPU, não pode ser descartado da cache sem garantir que foi copiado para a memória principal.

Políticas de Escrita

São duas: **write-through** e **write-back**.

Ambas as políticas tratam de um problema existente ao se usar memórias cache: Devido à hierarquia de memória, o processador irá modificar dados na cache, por ser mais rápida e de fácil acesso. No entanto, os valores na memória principal agora estão desatualizados! Podemos resolver isso mantendo ambas sempre atualizadas: ou seja, sempre que a cache for modificada, a memória principal também será atualizada, em sincronia. A essa técnica damos o nome de **write through**. Esse sistema consegue, de forma simples de se implementar, manter os dados sempre consistentes. O problema disso é que é muito lento fazer isso toda vez que uma mudança for feita na memória. Isso arruinaria todo o propósito de manter uma cache para se aproveitar de sua velocidade. Uma solução paliativa para essa diminuição do desempenho ao utilizar o **write through** é utilizar um **buffer de escrita**, no qual os dados serão armazenados enquanto aguardam para serem escritos na memória. Após escrever o dado na cache e no buffer de escrita, o processador pode continuar a execução das instruções.

Se o buffer de escrita estiver cheio quando o processador tiver que executar uma instrução de escrita, o processador precisa parar, até que haja posição disponível no buffer.

A outra abordagem, chamada **write back**, mantém um bit extra para cada slot de cache, chamado "**dirty bit**". Esse bit é responsável por indicar se o conteúdo está "sujo", ou seja, se ele sofreu modificações. Dessa forma, a memória principal será atualizada, mas somente quando o slot sofrer uma substituição e o dirty bit estiver em 1. Dessa forma, todas as mudanças serão feitas de uma só vez, e somente quando há a troca de slots. A desvantagem de um sistema desses é o aumento da complexidade para sua implementação, por exemplo, num computador multiprocessado, manter a consistência de cache necessita de cuidados a mais.

Para melhorar a política de **write back**, podemos usar um **write-buffer**. As mudanças vão sendo salvas neste buffer, que monitora o barramento. Quando o barramento estiver livre, as mudanças são escritas na memória. Isso diminui a competição no uso do barramento e possíveis gargalos na performance.

Coerência

Informalmente, digamos que é quando a leitura retorna o valor escrito mais recentemente. Se existe mais de um core compartilhando um mesmo espaço de memória, no final todos terão os mesmo valores, independente de quem fez escrita.

Como ocorre o suporte a compartilhamento com coerência?

Migração de dados para caches locais

Valor mais atualizado da variável é movido entre a cache que tem o valor mais atual e a cache que precisa do dado. Assim, reduz tempo de acesso e largura de banda da memória compartilhada.

Replicação dos dados compartilhados para leitura

Cada cache possui cópias de dados compartilhados que estão usando no momento. Reduz tempo de acesso e contenção para acesso.

Protocolos de Coerência de Cache

Para garantir coerência são necessários mecanismos implementados em hardware.

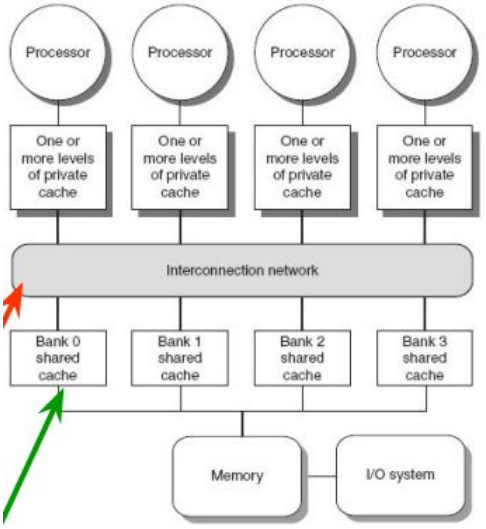
Protocolos Snooping

Cada cache monitora leituras e escritas no barramento, e cada bloco da cache mantém também o estado de compartilhamento do dado.

Existe o **Write Invalidate**[+], o mais utilizado atualmente por multiprocessadores, em que a escrita em uma cache de um dado compartilhado invalida todas as cópias das outras caches. E o **Write Update**, em que tal escrita deve vir acompanhada de atualização de todas as cópias das outras caches, e assim, requer maior utilização de barramento.

★ **Quais as limitações dos Snooping?** O aumento no número de processadores provoca crescimento na demanda por memória, e se a memória é única, se cria um gargalo. Para multiprocessadores bus-based, a quantidade de acessos devido a transações de coerência aumenta, limitando a largura de banda e, uou, se torna gargalo também.

★ **E quais seriam as soluções para isto?** Então, poderíamos ter múltiplos barramentos e redes de interconexão, uma memória configurada em bancos de memória.



ou: diretórios, que é o outro tipo de protocolo.

Protocolos baseados em Diretórios[+][+]

Caches e memória mantém status de compartilhamento de blocos em um diretório;

O diretório mantém informações como: quais caches têm quais blocos, estados dos blocos; e em caso de miss, encontra entrada no diretório, o analisa e comunica somente com os nós que possuem cópia; este que pode ser centralizado (implementado por exemplo numa cache compartilhada L3) ou distribuído (em que a comunicação se dá por rede de interconexões).

[+] Mais do Write Invalidate

Existem **controladores de cache "snoop"** que monitoram todas as transações no barramento. Assim, realizam ações para garantir coerência (invalidam, atualizam ou fornecem valores) e atualizam estados de compartilhamento dos blocos das caches.
Quando vai escrever, cache tem acesso exclusivo ao bloco em que a escrita será realizada e envia uma mensagem de invalidação no barramento por *broadcast*. Assim, a escrita só é completada quando cache obtém acesso ao barramento; dado é atualizado primeiro na memória!
Leituras em outras caches que acontecerão depois disso causam miss, de modo que a que tem a cópia atualizada *fornece* o resultado.

★ **Mas onde está a cópia mais atualizada?**
Depende da política de escrita, se for write-through, sempre se busca na memória, o que resulta numa maior quantidade de acessos. Agora se for write-back, a cache com cópia mais atualizada atende às requisições, reduzindo acessos. Esta segunda política é a mais usada por multiprocessadores.

★ **Como ocorre a localização da cópia mais atualizada em caches write-back?**

Cache com cópia atualizada responde requisições do bloco de memória

Controladores das caches monitoram o barramento para saber se possuem a cópia mais atualizada, assim se cancela requisição à memória principal.

Blocos da cache precisam armazenar também estado do bloco

shared : bloco pode ser lido, pois a cópia está atualizada. Somente escritas neste tipo de bloco precisam mandar requisição de invalidação.
modified/exclusive : {dirty} cache possui a única cópia válida que pode ser escrita
invalid : bloco inválido

Leitura em caches write-back com write invalidate

(1) CPU faz requisição de leitura
(2) Se o cache local não possui o bloco, um ***read miss*** é colocado no barramento
(3) Controladores das outras caches dão um "snoop" no barramento
(4) Cache que possui o bloco marcado como **exclusive** responde à requisição cancelando acesso à memória

Escrita em caches write-back com write invalidate

(1) CPU faz requisição de escrita
(2) Se o cache local não possui o bloco, um ***write miss*** é colocado no barramento

(3) Controladores das outras caches dão um “snoop” no barramento
(4) Cache que possui o bloco marcado como **exclusive** dá um “write-back” ~atualização~ na memória e invalida sua cópia
(5) Cache que possui bloco marcado como **shared** invalida sua cópia

[+][+] Mais do Protocolo por Diretórios

Para cada bloco, mantém estados (abaixo) e envia mensagens de invalidação.

shared : uma ou mais caches possuem o bloco e o valor atualizado está na memória
uncached : está apenas na memória
exclusive/modified : apenas uma cache possui o bloco e o valor na memória está desatualizado; ID de tal cache

Para um bloco que não está na cache ~ uncached ~

read miss
(1) Nó que fez a requisição recebe o bloco
(2) Nó marcado no diretório como o único que possui o bloco compartilhado (shared)

write miss
(1) Nó que fez a requisição recebe o bloco
(2) Nó marcado no diretório como o único e este é exclusivo

Para bloco compartilhado

read miss
(1) Nó que fez a requisição recebe o bloco
(2) Nó entra na lista dos que possuem o bloco compartilhado (shared)

write miss
(1) Nó que fez a requisição recebe o bloco
(2) Todos os demais recebem uma mensagem de invalidação; a lista de nós que possuem o bloco agora vai ser composta somente por este nó, o bloco é marcado como exclusive

Para bloco exclusivo

read miss
(1) Nó dono do bloco recebe uma requisição
(2) Bloco se torna compartilhado
(3) Bloco é enviado para a memória aka diretório
(4) Lista de nós que compartilham o bloco possui agora o antigo dono e o que acabou de fazer a requisição

write back no caso de uma substituição
(1) Bloco é enviado para a memória
(2) Bloco se torna uncached
(3) Lista de nós que compartilha o bloco fica vazia

write miss
(1) Requisição é enviada para o antigo dono
(2) Antigo dono manda bloco para a memória aka diretório

(3) Bloco no antigo dono é invalidado
(4) Nó que fez requisição se torna o novo dono e bloco continua sendo exclusivo

~ quando chegar aqui e tiver revisando, olhar no slide 13.1 o exemplo ~

Alguns conceitos de melhoria da cache

Reinício Precoce (Early Restart):

quando ocorre uma falta, o processamento é congelado e o bloco é substituído, com essa técnica, a execução é retomada quando a word solicitada é retornada, ao invés de esperar todo o bloco ser escrito.

Word Crítica Primeiro:

a word requisitada é transferida primeiro para a memória cache.

Cache multinível:

nessa abordagem, são utilizados vários níveis de cache, onde quanto mais afastada do processador, maior o armazenamento e o tempo de acesso, porém menor o custo é menor. Assim, cada nível pode focar em melhorar um aspecto da performance da cache. Uma estrutura de dois níveis de cache, por exemplo, permite que a primária se concentre em minimizar o tempo de acerto, porém, como a quantidade de misses seria a mesma, ela pode fazer uso da cache secundária, que estará focada em maximizar sua capacidade e reduzir a penalidade de uma falha direta na memória principal.

Aumento no Tamanho do Bloco:

É utilizado para tirar proveito da localidade espacial, visto que, ao ocorrer uma falta, é provável que palavras adjacentes serão necessárias em breve. Pode resultar em um miss rate menor, devido à localidade espacial, porém ao custo de aumentar a penalidade (blocos maiores levam mais tempo para serem transferidos). No entanto, após um certo limite, blocos muito grandes se tornam um empecilho: devido ao tamanho, ocupam espaço demais e as trocas constantes deles na cache não valem a pena.tirar proveito da localidade espacial

Aumento de banda passante da memória principal para a cache:

esta redução no custo da penalidade permite o uso de blocos maiores, a um custo próximo daquele obtido com blocos pequenos.

Desempenho com a Hierarquia de Memória

https://drive.google.com/file/d/1C0ek899U08NmINmRSwPALyBOxL_U40cH/view
{ é o slide 14 KKK. ngm quer fazer essa parte não? D: }

Memória Virtual

A técnica de memória virtual realiza a tradução do espaço de endereçamento de um programa para seus endereços reais. Permite que o tamanho de um único programa exceda a quantidade total de memória real disponível para sua execução. Essa técnica gerencia automaticamente os dois níveis de hierarquia: memória principal (física) e memória secundária. Se cria a ilusão de que a memória é maior do que ela realmente é, funciona como uma "cache para o disco".

É gerenciada pela CPU e pelo SO, que realizam tradução de endereços virtuais em reais, realiza busca de dados do disco para a memória e **proteção** - perceba que programas compartilham memória, mas cada um tem seu espaço de endereçamento, e existe proteção de acesso por parte de outros programas.

Proteção com registradores de Base e Limite, que só podem ser carregados pelo SO :

MMU compara endereço gerado por processo com registradores
Se há violação, avisa ao sistema operacional

Definições

Swap: é a troca de processos, em que processos residentes na memória podem ser movidos para disco e vice-versa. O tempo de mudança de contexto é alto, sendo a maior parte gasto na transferência do disco.

Página: bloco de tamanho fixo. (tam típico : 4KB - 64KB)

Falta de página: é uma falta no acesso à memória virtual.

O processador sempre gera um **endereço virtual**, que é traduzido para um endereço real por meio da MMU[+] (memory management unit), que é um sistema HW+SW. O **endereço real** (também chamado de físico) é, então, usado para acessar a memória.

[+] MMU :

É um dispositivo de HW que mapeia endereços virtuais em endereços físicos. A requisição de CPU passa pela MMU, que decodifica endereço e utiliza tabela de páginas, é quem avisa a falta de páginas à CPU.

Além disso, auxilia na proteção de processos, detecção de falhas, realocação de endereços, etc.

Memória Virtual e Realocação de código

A técnica da memória virtual simplifica a carga dos programas para execução, a partir da realocação: ela mapeia os endereços virtuais usados por um determinado programa em endereços físicos (antes de tais endereços serem usados para acessar a memória). Essa técnica de realocação permite que um programa seja carregado em qualquer posição da memória principal. Todos os sistemas de memória virtual realocam os programas por meio de blocos de tamanho fixo: as páginas.

A ocorrência de uma falta de página consome **milhões de ciclos de relógio**, este tempo é dominado pelo tempo para obter o primeiro endereço de uma página de tamanho típico. Para amortizar o tempo de acesso muito grande, as páginas devem ser grandes o suficiente. Uma técnica de **colocação totalmente associativa** tende a proporcionar uma baixa taxa de falta de página.

Os sistemas de memória virtual usam **write-back**.

◊◊◊

mail.google.com/mail/u/0/#inbox

O conceito de **multiprogramação** vem da ideia de vários processos na memória. A redução das faltas de página é o principal objetivo no projeto de um sistema de memória virtual, esta redução pode ser obtida através do uso de técnicas otimizadas para a colocação das páginas na memória principal.
Nesse sentido, a memória pode ser dividida em partições - que são blocos contíguos - em que o SO aloca para cada processo um destes espaços. Mas, nem sempre tal espaço existe de forma contígua, é onde entra a ...

Paginação

localização da página: associativa | política de escrita: write back | page fault tratada por SW

Esse modo de tradução consiste em dividir a memória em blocos que são chamados de páginas. A alocação de memória é requisitada por essas páginas, sendo a unidade de transferência entre a memória e o disco. É necessário uma tabela de páginas para saber onde qual endereço virtual corresponde na memória física.
Uma vantagem dessa técnica é a **impossibilidade de acontecer fragmentação externa**, já que as páginas de um processo não precisam ficar contínuas na memória; inclusive, algumas partes não utilizadas podem estar em disco. A desvantagem é a **existência da fragmentação interna**, ou seja, desperdício de espaço por consequência do tamanho de páginas ser fixo.

★ **E onde paginação se encaixa com memória virtual?**
O espaço de endereçamento virtual é maior que o oferecido pela memória física, assim, algumas páginas podem estar no disco.

Podemos falar também de :

Segmentação

Já nesse modo, os processos são divididos em segmentos de tamanhos variados.
A alocação é feita por uma **tabela de segmentos**, em que cada entrada guarda o

início do segmento e o tamanho, além de ter **bits de controle** (leitura/escrita, válido, modificado -dirty-); é indexada pelo número do segmento.

Endereços virtuais consistem de:

um número de segmento (bits mais significativos) e deslocamento dentro do segmento (bits menos significativos).

A ideia é dividir o programa em segmentos lógicos: programa principal, funções (métodos), variáveis globais, variáveis locais, classes, etc; é algo baseado em como o programador enxerga um programa.
É o contrário da paginação: **pode existir fragmentação externa** mas **não pode existir fragmentação interna**. Outra vantagem é a facilidade de proteção, já que cada processo tem seu segmento e tamanho determinados.

Segmentação Paginada

Para contornar as desvantagens anteriores, é possível dividir os segmentos em páginas, também permite que segmentos maiores que a memória possam ser carregados.
Uma **desvantagem é quantidade de tabelas**, aumentando o acesso à memória.

Modos de tradução de endereço

Tabela de páginas

Faz a correspondência entre as páginas da memória virtual e os endereços das páginas da memória física.
Permite a localização das páginas indexando totalmente a memória principal e sendo lá armazenada. É indexada com o número da página extraído do endereço virtual e contém o número da página física correspondente;

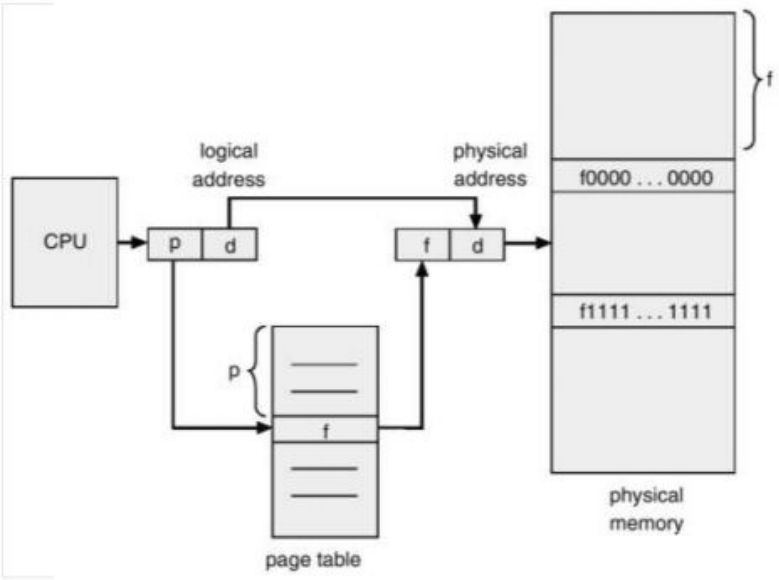
Implementação de tabela de páginas ~para facilitar a visualização~

(1) Tabela de páginas é armazenada na memória
(2) Indexada pelo número de página virtual
(3) Geralmente para cada processo existe uma tabela de páginas associada
(4) Comumente a CPU possui um **registrador da tabela de páginas:** em HW, aponta para a posição inicial da tabela de páginas na memória principal. (Registrador *Context* no MIPS)

Método de tradução de endereços virtuais

Se espaço de endereçamento virtual é 2^m e tamanho de página é 2^n, os (m-n) bits de mais alta ordem de um endereço virtual dão o número da página e os n bits restantes dão o deslocamento dentro da página.

Seja p = número da página virtual; d = deslocamento dentro da página; f = página física :



O que é guardado na tabela:

Endereço da página física	**Dirty Bit** Bit informando se foi modifica na memória	Bit informando se é somente **leitura** ou **leitura/escrita**	Bit informando se a página é **válida** ou não. on : está na memória off : está no disco

Tabelas de páginas multi-níveis

Se o espaço de endereçamento virtual for grande, a tabela também será, e alocar tabela de páginas contiguamente na memória não é muito legal. Assim, a solução são **tabelas de páginas multiníveis**, que consistem em paginar a tabela de páginas. Assim, o espaço de endereçamento virtual é quebrado em múltiplas tabelas de páginas e existe uma **tabela de tabelas de páginas** - a de maior nível contém os índices para as demais.

Desta forma, tabelas de páginas são carregadas na memória de acordo com a necessidade.

Page fault aka Falta de página

Execução de um programa não implica necessariamente no carregamento dele todo do disco para a memória. No geral, algumas páginas são carregadas e quando necessário, outras. Ocorre **page fault** justamente quando a execução requisita uma página que não está na memória ainda. Daí, o sistema operacional assume o controle, por meio do mecanismo de exceção.

A tabela de páginas possui um **bit de residência** (o válido/inválido) que quando igual a 0 indica falta de página. O endereço virtual, por si só, não informa em que posição do HD está a página que gerou a falta de página.

Lembremos que se tiver que substituir uma página e ela foi modificada, primeiro deve escrever ela no disco.

Substituição de Página

Quando ocorre falta de página e não há espaço disponível para carregar página, uma página na memória é removida e substituída. A removida, se alterada, é antes salva em disco.

A escolha para substituição deve ser feira com cuidado pelo SO para nao provocar repetidas faltas. Existem diferentes algoritmos: FIFO, Least Recently Used (muito usado) e o Second Chance são exemplos.

tlb - translation lookaside buffer

Cada acesso à memória exige, na verdade, dois acessos à memória principal: um para obter o endereço físico (consulta à tabela de páginas) e outro para buscar a informação.
Os processadores atuais possuem uma cache especial (muitas vezes, on-chip) que armazena as traduções de endereço mais recentes: **translation-lookaside buffer - TLB**, aproveita assim localidade temporal.
É uma memória cache associativa pequena e rápida, geralmente localizada na MMU que tem uma randômica política de substituição de entrada. Naturalmente, a TLB possui menos entradas do que o número de páginas da memória principal.

Dada uma entrada da TLB, o **rótulo** guarda uma parte do número da página virtual e o **campo de informação** guarda o número do endereço físico correspondente. Em uma referência, a tabela de páginas pode nem ser acessada. Logo, cada entrada da TLB precisa ter também: o **bit de residência** e um **bit de modificação**, para o caso de escrita.

Caso ocorra uma falta no acesso à TLB, é preciso determinar se realmente foi falta no acesso à TLB ou se foi falta devida à falta de página. Se a página estiver na memória principal (**falta no acesso à TLB**), o processador trata a falta colocando na TLB as informações necessárias para realizar a tradução (as informações são pegas na memória principal) e, assim, o processador tenta novamente acessar a página a partir da TLB. Isto pode ser feito por HW, com a MMU, ou por SW, levantando exceção a ser tratada pelo SO. Se a página não estiver na memória principal (**falta na TLB indica falta de página**) o SO trata, buscando a página e atualizando a tabela de páginas. Daí, reinicia a instrução que causou a falta.

É importante salientar que o nº de entradas na TLB < nº de páginas na memória principal. Faltas na TLB são muito mais frequentes que faltas de página.

Interação TLB e Cache

Se tag da cache utiliza endereço físico, existe a necessidade de traduzir o endereço antes de procurar na cache

Sistemas Operacionais e Operação em Dual-Mode

Compartilhamento de recursos requer um SO para garantir que um programa defeituoso não cause erros em outros programas. Computadores atuais dão suporte em hardware para executar instruções em dois modos de operação :
User mode, execução de instrução relativa a um aplicativo do usuário
Kernel mode (ou system mode), execução de uma instrução vinda de um SO

- ★ Instruções de aplicativos não podem ser executados quando o HW está em kernel mode
- ★ Instruções que só podem ser executadas em **kernel mode** são chamadas de **instruções privilegiadas** (privileged instructions). Ex: instruções de E/S, acesso a tabela de páginas. SO usa instruções privilegiadas
- ★ Quando um erro ou uma interrupção ocorre,hardware muda para kernel mode

Entrada e Saída

Armazenamento

Sistema de Entrada/Saída é composto de **dispositivos de naturezas diferentes**, em aspectos como **Comportamento**: entrada, saída, armazenamento | **Interação**: humano ou máquina | **Taxa de transferência**: dados/seg, operações/seg.
Geram gargalos pois se reduz a fração do tempo na CPU, e reduz o valor de CPUs mais rápidas.
Lei de Amdahl : speed-up é limitado pelo sub-sistema mais lento.

Projeto de Sistema de E/S

Deve ter as seguintes considerações: possibilidade de expandir o sistema, ter diversidade de dispositivos, ter determinados comportamento no caso de falhas, analisar custo e desempenho, sendo o maior ênfase em ser tolerante a falhas e o custo.
Desktops e sistemas embarcados são importantes para a diversidade de dispositivos, e **servidores** são importantes para a expansibilidade de dispositivos e tolerância a falhas.

Desempenho de sistema de e/s

Existem diferentes métricas, a depender da aplicação :

latência : tempo de resposta
Ex. Aplicações: desktops e sistemas embarcados

throughput : taxa de transferência
{quantidade de dados transferidos ; quantidade de operações E/S}
Ex. Aplicações: servidores

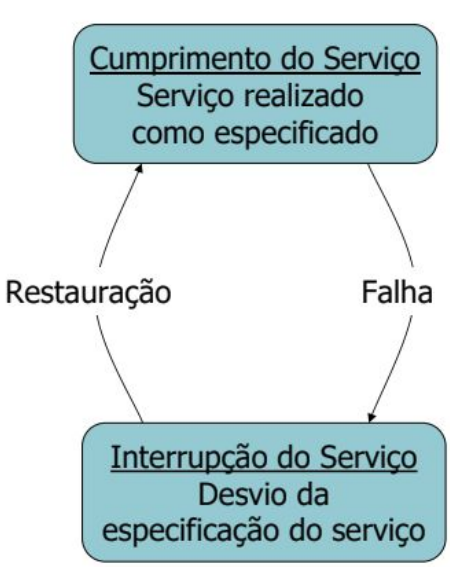
Difícil de medir pois deve se levar em conta as características do dispositivo, conexão com o sistema, hierarquia da memória e sistema operacional (software de I/O)

Dependabilidade (dependability)

é a propriedade que define a capacidade dos sistemas computacionais de prestar um serviço que se pode justificadamente confiar. É particularmente importante para dispositivos de armazenamento.
São atributos importantes: a confiabilidade (reliability); segurança; disponibilidade (availability); mantenabilidade.

fault

pode ser permanente ou transiente, podendo também provocar ou não uma falha no sistema como um todo. É basicamente quando um componente não executa como especificado.



★ **E como medir a dependabilidade?**

Primeiramente, alguns conceitos:

Confiabilidade: Mean Time To Failure (MTTF).
Interrupção de serviço : Mean Time To Repair (MTTR)
Disponibilidade: MTTF / (MTTF + MTTR)

Melhorando a disponibilidade:

Aumentando MTTF -> Fault avoidance | Fault tolerance | Fault forecasting
Diminuindo MTTR -> Melhorando ferramentas e processos de diagnóstico e reparação

Armazenamento Secundário

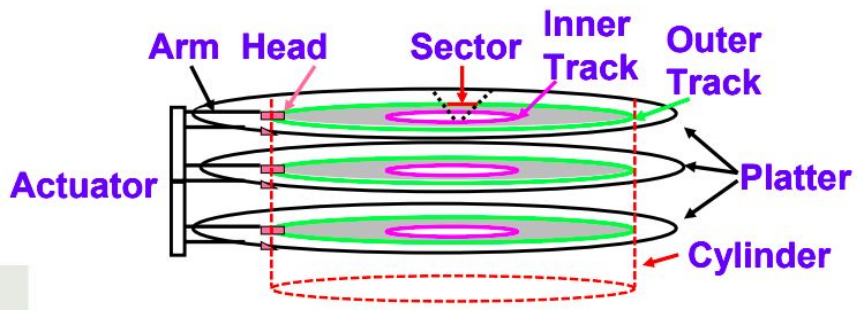
Importante para garantir dependabilidade de um sistema, afinal, memória e cache são voláteis. Com armazenamento secundário os dados são persistidos.
Além disso, tem impacto no desempenho do sistema, pois no caso de faltas na cache e memória é necessário acesso a disco, o que causa gargalo no tempo de execução.
Desafios: Como aumentar dependabilidade? e desempenho?

Tipos de armazenamento secundário

Disco :

Dispositivo magnético em que as partes gravadas são magnetizadas. Possui componentes mecânicos

Terminologias



Tem-se:

- Vários **pratos**, com a informação armazenada magneticamente em ambas superfícies (usual)
- Bits armazenados em **trilhas**, que por sua vez são divididas em **setores**[+] (e.g., 512 Bytes)
- **Atuador** move a **cabeça** (fim do braço, 1/superfície) sobre a trilha ("seek"), seleciona a superfície, espera pelo setor passar sob a cabeça, então lê ou escreve
  - "Cilindro": todas as trilhas sob as cabeças

[+] Setores do Disco e Acesso

Cada setor armazena:

ID do setor | Dados (512 bytes, 4096 bytes proposto) | Error correcting code (ECC), usado para esconder defeitos e erros de leitura | Campos de sincronização e espaços.

Acesso a setores envolve:

Delays devidos a espera se outros acessos foram feitos antes | Procura (Seek): mover as cabeças | Latência rotacional | Transferência de dados | Overhead do controlador.

Calculando o Desempenho de um Disco

Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead

em que:

Seek Time :

Depende do número de trilhas e velocidade de seek do disco.

Fabricantes anunciam seek time "pessimistas" (Seek time real entre 25% - 33% do anunciado)

Rotation Time(Latency), é o tempo para o setor girar embaixo da cabeça :

Depende da velocidade de rotação do disco

Tempo médio = 0,5 rotação/velocidade de rotação

Transfer Time :

Depende do tamanho do setor, densidade dos bits por trilha, tamanho da requisição, taxa de transferência

Flash :

Possui memória semicondutora (EEPROM)

Wearable

Após 100000-1000000 de escritas pode perder capacidade de armazenamento; **Wear leveling** : redistribui dados em áreas menos usadas.

Disco vs Flash

Velocidade de acesso : Flash 100-1000 mais rápida do que disco

Preço : Disco até 40x mais barato (2008), sendo mais popular em desktops e servidores

Wearable : Disco sem limite para escrita

Energia e Robustez : Flash mais eficiente e robusto, sendo mais popular em sistemas embarcados

Melhorando Desempenho

Localidade

Fabricantes anunciam tempo médio de seek, este baseado em todos os seek possíveis. Localidade e escalonamento de acessos pelo SO podem diminuir tempo de seek, depende de aplicação e de algoritmo de escalonamento

Controladores Inteligentes

Controladores com microprocessadores

Podem otimizar desempenho, apresentam interface alto nível permitindo que SO enxergue blocos lógicos e escalone acessos, têm interfaces padrões (comandos, protocolos, interface elétrica) que permitem conexão e transferências de dados entre computadores e periféricos SCSI (Small Computer Systems Interface) SATA (Serial Advanced Technology Attachment).

Controladores incluem cache

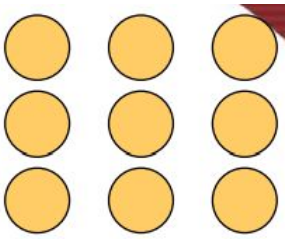
Mantém dados acessados recentemente, o controlador implementa algoritmos que fazem a busca antecipada de setores que têm probabilidade maior de serem buscados, evita seek e latência rotacional.

RAIDs (Redundant Array of Inexpensive Disks)

São arrays de discos pequenos e baratos. O desempenho é maior por causa do paralelismo. O dado fica espalhado em múltiplos discos. Acessos múltiplos são feitos a vários discos simultaneamente. É muito usada em servidores, dependabilidade é essencial.

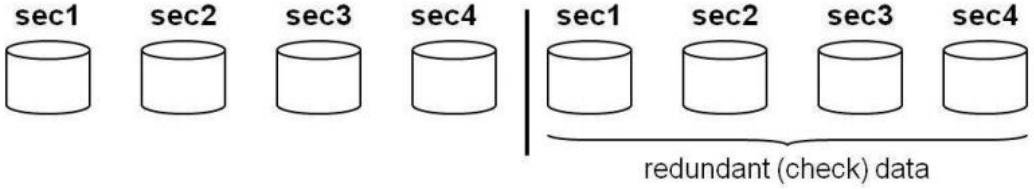
A confiabilidade é menor do que em disco único, mas a disponibilidade pode ser melhorada pela adição de tais discos redundantes, pois a informação perdida

pode ser recuperada através da informação redundante, que tem tolerância a falhas;



RAID 1 (Mirroring ~espelhamento~)

N + N discos, com dados replicados. Assim, a escrita deve ser feita nos discos de dados e nos redundantes. Em caso de falha, leitura do espelho



RAID 10 (1+0)

(RAID 1) +

Distribuição de blocos sobre discos múltiplos – *striping*.

Múltiplos blocos podem ser acessados em paralelo aumentando o desempenho.

RAID Nível 3 (Bit-Interleaved Parity)

Dados espalhados em N discos em nível de byte.

O disco redundante armazena paridade (soma dos discos por stripe mod 2), não é necessário armazenar todo o dado, a leitura é em todos os discos e a cada escrita gera nova paridade e atualiza todos os discos.

Em uma falha, usa da paridade para reconstruir dado: basta subtrair p da soma dos outros discos para recuperar informação.

vs

RAID Nível 4 (Block-Interleaved Parity)

Dados espalhados em N discos em nível de bloco.

O disco redundante armazena paridade para um grupo de blocos. A leitura é apenas no disco contendo o bloco, e a escrita (small writes) é no disco contendo bloco modificado e disco de paridade; cada escrita gera nova paridade e atualiza disco de dado e de paridade.

Em uma falha usa paridade para reconstruir dado.

RAID Nível 5 (Paridade Distribuída) : XOR

N + 1 discos.

Parecido com o 4, mas os blocos de paridade são distribuídos pelos discos. Assim, evita que o disco de paridade seja um gargalo e permite escritas simultâneas a diferentes discos.

Comunicação processador, memória e e/s

https://drive.google.com/file/d/1HLqTmpWMzaDRCZuYqHLvthlvrVmuXtsT/view
slide 17

~ tenho fé que termino essa parte até terça de madrugada ~
^ claramente deu ruim

~ daq pra baixo tinha no resumo original mas ainda nao vi o q tem nos slides ou não ~

RAID
Barramento e Comunicação
Barramento Síncrono e Assíncrono

Síncrono:

Barramentos síncronos necessitam pouca lógica, podendo ser facilmente implementados com FSM, mas é preciso que todos dispositivos operem na mesma frequência de clock, além disso, barramentos não podem ser tão longo se a taxa de transferência for alta.

Assíncrono:

Um barramento assíncrono não é guiado pelo clock. Dessa forma, ele pode acomodar uma diferente gama de dispositivos com diferentes velocidades, e o barramento pode ser ampliado sem o risco de ficar “fora de sincronia”. Para coordenar a transferência de dados, um barramento assíncrono utiliza um **handshaking protocol**, ou seja, são usados sinais de “requerimento” e de “pronto”. A transmissão do dado ocorre somente quando todas as partes envolvidas estiverem prontas para enviar/receber.

Controlando Acesso ao Barramento

O CPU não pode ser usado diretamente para controlar o acesso ao barramento. Muitas interrupções diminuem a eficiência do processador. Podemos ter um **árbitro** centralizado, que decidirá que dispositivo usará o bus, inclusive um **árbitro híbrido**,

centralizado e distribuído, onde existe prioridades. Nessa última abordagem, os dispositivos são ligados em série, os mais próximos do árbitro possuem maior prioridade.

O árbitro é o **bus master**, que decide agora quem será o próximo a usar o barramento, e pode iniciar e controlar requisições de uso ao barramento.

Tipos de Comunicação entre I/O e o Processador:

Polling: É o jeito mais simples de um dispositivo de E/S se comunicar com o processador. O dispositivo coloca a informação num registrador de status, que o processador está periodicamente checando para ver se há alguma alteração. A desvantagem disso é que o processador é muito mais rápido: ele pode checar o registrador várias vezes, sendo que o dispositivo pode não ter feito alteração alguma (como por exemplo, o mouse não ter se mexido nas últimas 1000 checagens).

O uso ideal para polling é quando as taxas de alteração dos dispositivos de E/S são **predeterminadas.** Assim, o overhead é mais previsível. É geralmente usado em sistemas de tempo real.

Interrupt Driven: É um esquema que faz uso de interrupções para dizer quando o dispositivo de E/S precisa da atenção do processador. Mas isso não quer dizer que ele seja síncrono, dependente de instrução, ou que pode evitar que uma instrução seja executada. A checagem se existe algum dispositivo de E/S pendente de ser tratado é feito antes de iniciar uma nova instrução. Feita a comunicação, geralmente o restante do processamento é gerenciado pelo sistema operacional. (Como por exemplo, uma tela pressionada do teclado)

Interrupt Driven Communication é usada por quase todos os sistemas em pelo menos alguns de seus dispositivos de E/S.

DMA: Significa Direct Memory Access, e é implementado com um controlador especial. Esse controlador assume controle do barramento (bus master), o que significa que ele pode iniciar requisições. O DMA é usado porque o overhead usando um sistema Interrupt Driven para transferir dados do disco rígido é absurdo. Por isso, o DMA é usado como um assistente - ele livra o processador de ficar constantemente checando se o dado foi transferido, realiza a transferência de forma autônoma, sem “incomodar” o processador para que ele possa continuar realizando outras tarefas, e só realiza uma interrupção novamente quando a transferência for concluída, para que o processador ou ‘pergunte’ ao DMA ou cheque na memória se a transferência foi bem sucedida.

Processadores DMA geralmente são processadores de propósito único, e são implementados para realizar essa interface entre o disco rígido e o processador, quando uma transferência se faz necessária.

Descrevendo o que acontece:

O DMA é implementado com um controlador que assume o controle do barramento. O processador fornece a este controlador o endereço do dispositivo, a operação que vai ser realizada, o endereço de memória e o número de bytes. Com isto, o controlador vai gerenciar a transferência e quando termina, interrompe o processador.

Proteção de Acesso pelo S.O.

O SO precisa gerir as permissões de cada programa-usuário, um programa qualquer não pode escrever no disco sem permissão concedida. Técnicas como "Read Write Execute" podem ser utilizadas para garantir que programas só leiam, escrevam e no que lhe for permitido.

O SO faz isso impedindo que o programa do usuário se comunique com os dispositivos de E/S diretamente. Existem formas de realizar esse intermédio:

- ➔ Dar comandos aos dispositivos de E/S (read, write, seek, etc)
- ➔ O dispositivo de E/S notificar o SO quando completar uma operação ou encontrar um erro.
- ➔ Os dados só poderão ser transferidos da memória para o dispositivo de E/S. Nunca se pode ler ou escrever diretamente do disco, por exemplo.

Multi-Core

Modelos de Memória de Multi-Cores

Num multi core, é utilizado um mesmo espaço de endereçamento de memória para todos os processadores. Eles se comunicam através de variáveis compartilhadas, e todos os processadores podem acessar qualquer lugar na memória por meio de loads e stores.

Os processadores multi-cores podem ser classificados em dois tipos, de acordo com os modelos de memória: **Memória Centralizada** (SMP ou Symmetric Multiprocessors, também chamado de UMA - Uniform Memory Access) e **Memória Distribuída** (NUMA - Non Uniform Memory Access).

Numa abordagem **UMA**, os acessos à memória principal levam o mesmo período, independente do processador que requisita o acesso e de qual word é requisitada.

Já na abordagem de **NUMA**, alguns acessos à memória são mais rápidos que outros, dependendo de que processador requisita que word.

Também existem os processadores que não fazem uso de memória compartilhada: eles possuem apenas seções de memória **privada**, e se comunicam através de troca de mensagens.

Modelos de Comunicação em Processos Paralelos

Temos o **message-passing,** onde os processadores trocam mensagens por meios de uma rede local. Para isso, é necessário que o sistema tenha rotinas para comunicação. O modelo de memória mais adequado nesse caso, é o de memória privada, muito utilizado em clusters.

Também existem os processadores baseados em **memória compartilhada**, que se comunicam por meio de variáveis compartilhadas usando loads e stores. Nesse caso, a abordagem mais utilizada seria a NUMA, que distribui fisicamente a memória entre os processadores. A abordagem UMA também é viável, mas pouco utilizada.

Resolvendo Coerência de Cache

A coerência de cache diz respeito a **consistência no valor** dos dados entre as versões nas caches de vários processadores. Como cache é atualizada constantemente, é importante que a seus dados estejam coerentes com a memória principal, além disso, em sistemas multiprocessados, cada CPU pode ter uma cache própria, mas com dados compartilhados, é importantes que os dados sejam lidos e escritos corretamente. Ao atualizar seus dados, o processador não deveria se importar se o dado é compartilhado ou não causando várias cópias discrepantes de um mesmo dado.

O problema de coerência de cache surge quando a memória é compartilhada por vários processadores. Uma cache privada não sofre deste problema pois somente um processador pode acessá-la.

Esse problema pode ser resolvido nas caches compartilhadas através da implementação de **protocolos** pelo controlador da cache ou pelo controlador de memória que irão cuidar de **rastrear o estado de compartilhamento dos dados.**

Dois dos protocolos mais comuns são: **snooping** ou **baseada em diretório**.

No protocolo de **snooping**, cada cache que tem uma cópia do dado também tem uma cópia do estado de compartilhamento. Todas as caches são acessíveis através de um mesmo barramento, e os controladores de cache ficam monitorando este barramento (snooping) para saber se alguma cópia que eles detém está sendo requisitada ou não por outras caches.

O protocolo de snooping usa de duas técnicas para manter a consistência dos dados: **write update** e **write invalidate.** No primeiro, após acontecer uma escrita em uma das caches, todas as caches que possuírem uma cópia desse dado atualizam seus valores. No caso do invalidate, sempre que ocorre uma escrita numa variável compartilhada, as caches que possuem uma cópia tratam de invalidá-las. Como o protocolo write update causa maior uso do barramento para manter as variáveis atualizadas, o mais usado é o write invalidate.

No protocolo **baseado em diretório**, o status de compartilhamento de um bloco da memória principal é armazenado em um local centralizado (diretório). Toda mudança feita deve ser comunicada ao diretório, que cuida de comunicar a quem for necessário (quem tiver cópias dos dados) as alterações feitas. Um único diretório mantém o estado de cada bloco na memória principal. As informações no diretório podem incluir quais caches possuem cópias do bloco, se ele é sujo etc.