Resumo Prova 2 SO

Guilherme Christopher Michaelsen Cardoso *

15 de maio de 2017

*baseado no livrão do Tanenbaum

1 Resumão de SO (Prova 2)

1.1 Espaços de endereçamento

Expor memória física para processos tem diveras desvantagens:

- Permitir que programas de usuário enderecem qualquer byte de memória torna possível quebrar o sistema operacional.
- Com esse modelo, é dificil ter vários programas rodando ao mesmo tempo.

1.1.1 Noção de espaço de endereçamento

É necessário resolver dois problemas antes de permitir múltiplas aplicações em memória ao mesmo tempo: proteção e relocação.

- Espaço de Endereçamento:
  - Conjunto de endereços que um processo pode usar para endereçar memória.
  - Cada processo tem seu próprio espaço de endereçamento, independente dos outros processos (exceto em circunstâncias especiais onde processos querem compartilhar seus espaços de endereçamento).

1.2 Memória Virtual

- Enquanto a capacidade das memórias cresce rapidamente, o tamanho dos programas cresce ainda mais rápido.
- Necessidade de rodar programas que são grandes demais para caber em memória
- Swapping não é uma alternativa desejável, devido à lentidão dos discos rígidos.
- **Memória virtual** foi a solução encontrada para esse problema.
  - Cada programa tem seu próprio espaço de endereçamento, que é dividido em partes chamadas **páginas**.
  - Cada página é um intervalo contíguo de endereços, que são mapeados em memória física.
  - Nem todas as páginas precisam estar em memória física ao mesmo tempo para rodar o programa.
  - Quando o programa referencia uma parte do endereço que já está em memória física, o hardware realiza o mapeamento necessário na hora.
  - Quando o programa referencia uma parte do espaço de endereçamento que não está em memória, o SO recebe um aviso para buscar a página faltante e reexecutar a instruçao que falhou.

- Funciona muito bem em sistemas multiprogramados com pedaços de vários programas na memória ao mesmo tempo. Quando um programa está esperando uma página ser buscada do disco, a CPU pode ser concedida a outro processo.

1.2.1 Paginação

- A maioria dos sistemas de memória virtual usam **paginação**
- Programas geram **endereços virtuais** de memória, formando o **espaço de endereçamento virtual**.
- Em sistemas com memória virtual, esses endereços não são enviados diretamente ao barramento da memória. Ao invés disso, vão para uma **MMU (Memory Management Unit)** que mapeia o endereço virtual em endereços de memória reais.
- O espaço de endereçamento virtual consiste de unidades de tamanho fixo chamadas **páginas**. AS unidades correspondentes em memória física são chamadas de **molduras de página**. As páginas e molduras de página geralmente tem o mesmo tamanho.
- Supondo um sistema de memória com páginas de 4KB, 64KB de espaço de endereçamento virtual e 32KB de memória física, temos 16 páginas virtuais e 8 molduras de páginas.
- Sempre que se necessita buscar um item no disco, é necessário que a página inteira seja buscada.
- Vários processadores suportam múltiplos tamanhos de página que podem ser misturados pelo SO de acordo com a necessidade.
  - Ex.: A arquitetura x86_64 suporta páginas de 4KB, 2MB e 1GB, sendo possível, por exemplo, utilizar páginas de 4KB para aplicações de usuário e uma única página de 1GB para o kernel do SO.
- Exemplo: Supondo páginas de 4KB, 64KB de espaço de endereçamento virtual e 32KB de memória física. Um programa solicita um dado que está no endereço virtual 0. Supondo que a página que contém os endereços 0 a 4095 está mapeada na moldura de página numero 2. Qual o endereço de memória real que será emitido pela MMU?
  - A moldura 0 contém os endereços 0 a 4095. A moldura 1 contém os endereços 4096 à 8191. Logo, a moldura 2 começa no endereço 8192 e vai até 12287. Portanto, o endereço físico que será emitido será 8192.
- Exemplo 2: O endereço virtual 20500 é 20 bytes à partir do começo da página virtual 5 (endereços virtuais 20480 a 24575). Supondo que essa página seja mapeada para a moldura de página número 3 (12k-16k), o endereço físico emitido pela MMU será 12288 + 20 = 12308.
- Como existem mais páginas virtuais do que moldura física, é necessário controlar quais páginas virtuais estão presentes em memória física. Para isso, é usado um bit de controle (**bit presente/ausente**).

- Caso o programa referencie um endereço não mapeado, a MMU detecta isso e faz com que a CPU interrompa o sistema operacional. Essa interrupção é chamada de **page fault**. O SO então escolhe uma moldura de memória pouco usada, salva seu conteúdo no disco (caso já não esteja lá), e então busca (também do disco) a página que acabou de ser referenciada, colocando-a na moldura de página que foi liberada, refaz o mapeamento, e reinicia a instrução interrompida.
- Exemplo: supondo um endereço virtual 8196 (0010000000000100 em binário), sendo mapeado pela MMU. Esse endereço virtual de 16 bits é dividido em duas partes: um número de página (4 bits) e um offset (12 bits). Com 4 bits de offset, podemos endereçar $2^4 = 16$ páginas, e com um offset de 12 bits, podemos endereçar todos os $2^{12} = 4096$bytes em uma página.
- O número da página é usado como índice na **tabela de páginas**, representando o número da moldura de página correspondente àquela página virtual. Se o bit *presente/ausente* é 0, uma interrupção ao SO é causada. Se é 1, o número da moldura de página encontrada na tabela de páginas é copiado para os 3 bits mais significativos do registrador destino, junto com o offset de 12 bits, que é copiado do endereço virtual. Juntos, eles formam um endereço físico de 15 bits. Esse endereço é então enviado ao barramento da memória como o endereço físico de memória.

1.2.2 Tabelas de página

Em uma implementação simples, o mapeamento de endereços virtuais em endereços físicos pode ser feito da seguinte forma:

- O endereço virtual é dividido em um número de página virtual (bits mais significativos) e um offset (bits menos significativos)
  - Por exemplo, em um endereço de 16 bits com páginas de 4KB, os 4 bits mais significativos especificam uma das 16 páginas virtuais e os 12 bits menos significativos representam o byte offset dentro da página selecionada.
  - É possível usar endereços menores ou maiores para indexar a página, definindo tamanhos diferentes de páginas.
- O número de página virtual é usado como índice na tabela de páginas para encontrar a entrada para essa página virtual. A partir da entrada na tabela de páginas, o número da moldura de página (se existir), é encontrado.
- O número da moldura de página é concatenado aos bits mais significativos do offset, substituindo o número da página virtual, para formar um endereço físico que pode ser enviado para a memória.

Portanto, o propósito da tabela de páginas é mapear páginas virutais em molduras de página.

1.2.3 Estrutura de uma entrada na tabela de páginas

O layout exato de uma entrada na tabela de páginas é altamente dependente da máquina, mas a informaçao presente normalmente é a mesma.

- O tamanho varia de computador a computador, mas 32 bits é um tamanho comum.
- O campo mais importante é o o *número da moldura de página*.
- Em seguida, existe o bit *presente/ausente*. Se esse bit for 1, a entrada é válida e pode ser usada, caso contrário, a página virtual não está atualmente em memória. Acessar uma entrada na tabela de páginas que tenha esse bit em 0 causa um page fault.
- Os bits de proteção dizem quais tipos de acesso são permitidos. Na forma mais simples, se tem apenas um bit, que permite apenas leitura se estiver em 0, e leitura ou escrita se estiver em 1. Uma alternativa é usar 3 bits, um para leitura, um para escrita e um para execução.
- Os bits *modificado* e *referenciado* controlam a utilização da página. Quando ocorre uma escrita em uma página, o hardware seta o bit *modificado*. Esse bit é importante quando o SO decide usar aquela moldura de página. Se a página tiver sido modificada (suja), ela deve ser escrita novamente no disco. Se não, ela pode ser abandonada, já que a cópia em disco ainda é válida. Esse bit normalmente é chamado de **dirty bit**.
- O bit *referenciado* é setado toda vez que uma página for referenciada, tanto para leitura quanto para escrita. Seu valor é usado para ajudar o SO a escolher uma página quando ocorre um page fault. Páginas que não estão sendo usadas são melhores candidatas a serem substituídas.
- O último bit permite desabilitar o caching na página. Isso é importante para páginas que mapeiam direto em registradores de dispositivos ao invés de memória. Se o sistema operacional estiver em loop esperando um dispositivo de I/O responder um comando, ele precisa buscar constantemente a word do dispositivo, e não usar uma cópia em cache desatualizada. Máquinas que não utilizam I/O mapeado em memória não precisam deste bit.

1.3 Acelerando a Paginação

Em qualquer sistema de paginação, dois problemas devem ser enfrentados:

1. O mapeamento do endereço virtual para o endereço físico deve ser rápido.
2. Se o endereço virtual for grande, a tabela de páginas também será.

O primeiro problema ocorre devido à natureza dos programas. A cada instrução executada, é necessário fazer mapeamento de endereços virtuais para físicos, já que todas as instruções são buscadas em memória e muitas delas também referenciam operandos em memória.

O segundo problema ocorre porque todos so computadores modernos utilizam endereços virtuais de 32 ou 64 bits. Usando páginas de 4KB, um endereço de 32 bits é capaz de referenciar 1 milhão de páginas, e um endereço de 64 bits referencia um número absurdamente grande de páginas. Com 1 milhão de páginas no espaço de endereçamento virtual, a tabela de páginas deve ter 1 milhão de entradas, e cada processo precisa da sua própria tabela de páginas (por possuir seu próprio espaço de endereçamento).

1.3.1 Translation Lookaside Buffers (TLB)

A maioria das técnicas de otimização de paginamento parte do princípio que a tabela de páginas está em memória. Se uma instrução copia um registrador para outro, por exemplo, em um sistema sem paginação, ela faz apenas uma referência à memória (para buscar a instrução em si). Com paginamento, pelo menos uma referência a mais é necessária, para acessar a tabela de páginas. Devido ao impacto dos acessos de memória na performance da CPU, dobrar o número de acessos à memória diminui potencialmente pela metade o desempenho.

Baseado na observação de que a maioria dos programas tende a fazer um grande número de referências a um número pequeno de páginas, foi-se possível desenvolver uma solução para esse problema.

- Essa solução consiste em equipar computadores com um componente em hardware que é capaz de mapear endereços virtuais em endereços físicos sem precisar acessar a tabela de páginas. Esse dispositivo é chamado de **TLB** (Translation Lookaside Buffer).
- Esse dispositivo normalmente faz parte da MMU e consiste de uma tabela com um pequeno número de entradas, por exemplo, 8 (raramente mais do que 256). Cada entrada contém informação sobre uma página, incluindo o número da página virtual, um bit que indica se a página foi modificada, o código de proteção (permissões read/write/execute), e o número da moldura de página física na qual a página está localizada. Esses campos possuem correspondência 1 para 1 com os campos na tabela de páginas, exceto pelo número da página virtual, que não é usado pela tabela de páginas. Outro bit indica se a entrada é válida ou não.
- Quando um endereço virtual é enviado à MMU para tradução, o hardware primeiro checa para ver se é um número de página virtual presente na TLB comparando esse endereço paralelamente com todas as entradas na TLB. Se um cnadidato válido é encontrado e o acesso não viola os bits de proteção, a moldura de página é buscada diretamente do TLB, sem necessitar acessar a tabela de páginas.
- Se o número de página virtual está presente na TLB, mas a instrução está tentando escrever em uma página protegida contra escrita, uma protection fault é gerada.
- Quando um número de página virtual não está no TLB, a MMU detecta o miss e faz uma busca na tabela de páginas. Ela então remove uma das entradas do TLB e substitui com a entrada na tabela de páginas que acabou de encontrar. Se essa página for usada novamente em breve, ela será encontrada no TLB.

- Quando uma entrada é removida do TLB, o bit de modificado é copiado de volta para a entrada da tabela de páginas em memória. Os outros valores já estão lá, exceto o bit de referência. Quando o TLB é carregado da tabela de páginas, todos os campos são buscados em memória.

1.3.2 Gerenciamento de TLB por software

Até agora, assumiu-se que todas as máquinas com memória virtual paginada possuem tabelas de páginareconhecidas pelo hardware, mais um TLB. Nesse tipo de projeto, o gerenciamento do TLB e das faltas de TLB são feitos inteiramente pela MMU. Traps para o SO ocorrem apenas quando uma página não está presente em memória.

Porém, muitas máquinas RISC fazem todo o gerenciamento de páginas em software. Nessas máquinas, entradas na TLB são carregadas explicitamente pelo SO.

- Quando um miss no TLB ocorre, ao invés da MMU buscar a referência necessária na tabela de páginas, ele apenas gera uma faulta de TLB e lança o problema para o SO. O sistema então deve encontrar a página, remover uma entrada do TLB, inserir a nova entrada, e reiniciar a instrução faltante. Como misses de TLB são mais comuns do que page faults, esse processo deve ser rápido.
- Se o TLB for razoavelmente grande para reduzir a miss rate, o gerenciamento por software de TLB se torna aceitavelmente eficiente.
- Vantagem: MMU muito mais simples, liberando espaço no chip da CPU para caches e outros componentes que possam melhorar a performance.

Várias estratégias foram desenvolvidas para melhorar o desempenho em máquinas que gerenciam TLB por software.

- Uma forma é buscar reduzir tanto os misses de TLB quanto o custo de um miss na TLB quando eles ocorrem.
- Para reduzir misses no TLB, o sistema operacional pode usar sua intuição para descobrir quais páginas tendem mais a serem usadas logo e carregar suas entradas na TLB.
- A forma normal de processar um miss de TLB, tanto em hardware quanto em software, é buscar na tabela de páginas a página referenciada. O problema de se fazer essa busca em software é que as páginas que contém a tabela de páginas podem não estar no TLB, o que causa faltas adicionais no TLB durante o processamento. Essas faltas podem ser reduzidas mantendo um grande cache em software de entradas na TLB em uma localização fixa cuja página sempre seja mantida na TLB. Ao checar primeiro o cache em sofware, o SO pode reduzir substancialmente as faltas na TLB.

Ao usar gerenciamento de TLB por software, é essencial entender a diferença entre diferentes tipos de misses. Um **soft miss** ocorre quando a página referenciada não está na TLB,

mas está em memória. Messe caso, tudo que é necessário é atualizar o TLB, sem necessidade de I/O no disco. Tipicamente, um soft miss leva de 10 a 20 instruções para ser tratado e pode ser completado em alguns nanosegundos. Em contraste, um **hard miss** ocorre quando a página em si não está em memória e nem na TLB. Um acesso a disco é necessário para buscar a página, que pode levar vários milisegundos, dependendo do disco sendo usado. Um hard miss é mais de 1 milhão de vezes mais lento do que um soft miss. O processo de busca na hierarquia da tabela de páginas é chamado de **page table walk**.

Os misses não são apenas soft ou hard. Por exemplo, se o page walk não encontrar o a página na tabela de páginas de um processo, resultando em uma page fault. Existem 3 possiblidades.

1. A página pode estar em memória, mas não na tabela de páginas do processo. Por exemplo, pode ter sido trazida do disco por outro processo. Nesse caso, não é necessário acessar o disco novamente, bastando mapear a página apropriadamente nas tabelas de página. Esse tipo de soft miss'e chamado de **minor page fault.**
2. A página precisa ser trazida do disco. Este é o **major page fault**.
3. O programa simplesmente acessou um endereço inválido e não é necessário realizar nenhum tipo de mapeamento no TLB. Nesse caso, o SO normalmente mata o programa com um **segmentation fault**. Este é o único caso em que o programa fez algo de errado. Os outros casos são automaticamente tratados pelo hardare e/ou pelo SO.

1.4 Tabelas de página para memórias grandes

1.4.1 Tabelas de página multinível

- Solução para lidar com espaços de endereçamento virtuais muito grandes.
- Assumindo um endereço virtual de 32 bits que é particionado em um campo PT1 de 10 bits, um campo PT2 de 10 bits e um campo Offset de 12 bits. Como offsets são de 12 bits, cada página tem 4KB, e existe um total de 2^{20} delas.
- O segredo para o método de tabela de páginas multinível está em evitar manter todas as tabelas de páginas em memória o tempo todo. As que não são necessárias, não devem ser mantidas.
- O campo PT1 de 10 bits endereça a tabela de página top-level, com 1024 entradas. Quando um endereço virtual é enviado para a MMU, ele extrai o campo PT1 e usa ese valor como um índice na tabela de páginas top-level. Cada uma dessas 1024 entradas na tabela de página representa 4M porque o espaço de endereçamento vitual de 4GB (32bits) foi dividido em fatias de 4096 bytes.
- A entrada encontrada indexando a tabela de páginas top-level contém o endereço ou o número da moldura de página de uma tabela de páginas second-level. A entrada 0 da tabela top-level aponta para a tabela de páginas da área de texto do programa, a entrada

1 para a tabela de páginas da área de dados, e a entrada 1023 para a tabela de páginas da pilha. As outras entradas não são usadas.

- O campo PT2 agora é usado como um índice na tabela de páginas de segundo nível selecionada para encontrar o número da moldura de página da página em si.
- Exemplo: Considere o endereço virtual de 32 bits 0x00403004, que se refere a 12,292 bytes dentro do segmento de dados. Esse endereço virtual corresponde a PT1 = 1, PT2 = 3 e Offset = 4. A MMU usa PT1 para indexar a tabela top-level, e obter a entrada 1, que corresponde aos endereços 4M a 8M -1. Ele então usa a PT2 para indexar a tabela second-level que acabou de encontrar e extrair a entrada 3, que corresponde aos endereços 12288 a 16383 dentro dessa fatia de 4M. Essa entrada contém o número da moldura de página contendo o endereço virtual 0x00403004. Se essa página não está presente em memória, o bit *presente/ausente* na tabela de páginas terá o valor zero, causando um page fault. Se a página estiver presente em memória, o número da moldura de página tirada da tabela de páginas second-level é combinado com o offset(4) para construir o endereço físico. Esse endereço é enviado ao barramento da memória.
- É interessante notar que apesar de o espaço de endereçamento conter mais de um milhão de páginas, apenas quatro tabelas de páginas são necessárias: a tabela top level, e as tabelas second-level de 0 a 4M (para o segmento de texto), 4M a 8M (para o segmento de dados) e os ultimos 4M (para a pilha). Os bits *presente/ausente* do resto das 1021 entradas da tabela de página top-leval são setados para 0, forçando um page fault se eles forem acessados. Se isso ocorrer, o sistema operacional vai notificar que o processo está tentando referenciar memória que não deveria referenciar, e tomar uma ação apropriada, como enviar um sinal para matar o processo.
- Esse sistema de dois níveis de tabelas de páginas pode ser expandido para um sistema de três, quatro ou mais níveis. Mais níveis provêm mais flexibilidade. O processador Intel 80386, por exemplo, era capaz de endereçar até 4GB de memória usando uma tabela de dois níveis que consistia de um **diretório de páginas** cujas entradas apontavam para a tabela de páginas, que, por sua vez apontava para as molduras de página de 4KB. Tanto o diretório de páginas quanto as tabelas de página continham 1024 entradas, dando um total de $2^{10} * 2^{12} = 2^{32}$ bytes endereçáveis, como desejado.
- 10 anos depois, o Pentium Pro introduziu outro nível: **tabela de ponteiros do diretório de páginas**. Além disso, extendeu cada entrada em cada nível da hierarquia da tabela de páginas de 32 para 64 bits, de forma a tornar possível acessar memória acima do limite de 4GB. Como só possuía 4 entradas na tabela de ponteiros do diretório de páginas, 512 em cada diretório de páginas e 512 em cada tabela de páginas, o total de memória que conseguia acessar ainda estava limitado a um máximo de 4GB.
- Quando um suporte apropriado de 64 bits foi adicionado à familia x86 (originalmente pela AMD), foi adicionado mais um nível, o **page map level 4**. Todas as tabelas agora possuem 512 entradas, tornando possível endereçar $2^9 * 2^9 * 2^9 * 2^9 * 2^{12} = 2^{48}$ bytes. Ainda era possível adicionar mais um nível, mas provavelmente se atingiu a conclusão de que 256TB são suficientes por um bom tempo.

1.5 Algoritmos de substituição de páginas

Quando um page fault ocorre, o sistema operacional precisa escolher uma página para remover da memória, liberando espaço para outras páginas. Se a página que será removida foi modificada enquanto estava em memória, ela deve ser reescrita no disco, para atualizar a cópia em disco. Se a página não foi alterada, não é necessário reescrever nada. A página a ser lida apenas substitui a página removida.

1.5.1 Algoritmo NRU: Not Recently Used

- A maioria dos computadores com memória virtual possuem dois bits de status: R e M, associados a cada página. R é setado toda vez que a página é referenciada (lida, ou escrita). M é setado toda vez que ocorre uma escrita na página (isto é, ela é modificada). Os bits são contidos em cada entrada na tabela de páginas. É importante que esses bits sejam atualizados em todas as referências à memória, para que eles sejam setados pelo hardware. Quando um bit é setado para 1, ele permanece em 1 até o SO resetá-lo.
- Se o hardware não tiver esses bits, eles podem ser simulados usando os mecanismos de page fault e de clock interrupt do sistema operacional. Quando um processo é iniciado, todas as suas tabelas de página são marcadas como não estando em memória. Assim que qualquer página for referencia, um page fault ocorrerá. O sistema operacional então seta o bit R (nas suas tabelas internas), muda a entrada da tabela de páginas para apontar para a página correta em modo READ ONLY, e reinicia a instrução. Se a página for subsequentemente modificada, outro page fault vai ocorrer, permitindo que o sistema operacional sete o M bit e mude o modo da página para READ/WRITE.
- Os bits R e M podem ser usados para implementar um algoritmo de paginação simples.
  - Quando um processo é iniciado, ambos os bits para todas as páginas são setados para 0 pelo SO. Periódicamente (e.g., em cada interrupção de clock), o bit R é zerado, para distinguir páginas que não foram referenciadas recentemente das que foram referenciadas.
  - Quando um page fault ocorre, o SO inspeciona todas as páginas e as divide em quatro categorias, baseadas nos valores atuais de seus bits R e M:
    * Classe 0: não referenciado, não modificado.
    * Classe 1: não referenciado, modificado.
    * Classe 2: referenciado, não modificado.
    * Classe 3: referenciado, modificado.
  - O algoritmo **NRU (Not Recently Used)** remove uma página aleatória da classe não-vazia de menor número. Implícita neste algoritmo está a idéia de que é melhor remover uma página modificada que não tenha sido referenciada em pelo menos um ciclo de clock, do que uma página limpa que está sendo muito usada. A principal vantagem do NRU é que ele é simples de entender, moderadamente eficiente de implementar e provê uma performance que, apesar de não ser ótima, pode ser adequada.