UC Sistemas Operacionais (ADE)
ICT/UNIFESP

Prof. Bruno Kimura
bruno.kimura@unifesp.br
12/10/2020

TRAB_3: Gerenciamento de Memória

Objetivo:	Implementar a abordagem de MMU baseada em tabela de páginas invertidas.
Metodologia:	Trabalho individual ou em grupo de no máximo 3 (três) alunos.
Entregáveis:	• Protótipo em arquivos .c • Relatório em documento PDF ou na forma de video-apresentação.
Data de entrega:	**19/10/2020**
Observação:	Somente serão aceitos trabalhos **autênticos**. Cópias (entre grupos e/ou de fontes da Internet) serão anuladas.

Requisitos do relatório:

O relatório, oral (video-apresentação) ou escrito (na forma de relatório científico), deverá:

1. Apresentar explicação do trabalho desenvolvido: o que, como e porque foi feito.
2. Descrever as contribuições de cada integrante do grupo no desenvolvimento do trabalho.
3. Apresentar uma auto-avaliação do grupo sobre o trabalho realizado.

Descrição:

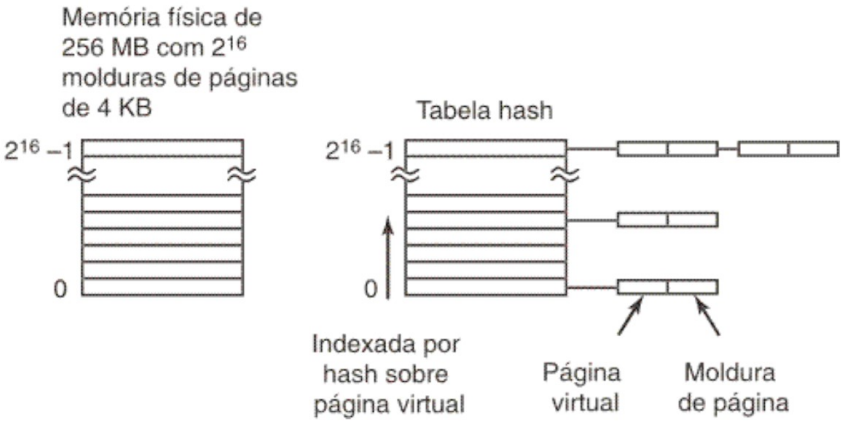
No gerenciamento de memória, o endereço virtual em espaço de página virtual deve ser mapeado para endereço físico conforme uma moldura de página (*page frame*) na memória principal. Quando um endereço virtual é referenciado pela CPU, tal endereço deve ser tratado pela MMU. Se o endereço virtual estiver indicado na tabela de páginas virtuais como ausente em memória, ocorre ausência de página (*page miss*), gerando uma interrupção (*trap*), para que o SO carregue a página requerida em memória e a mapeie na entrada na tabela de página virtual. Se não houver espaço para um novo mapeamento dentro do espaço de molduras de páginas físicas em memória, o SO (gerenciador de memória) tem de substituir uma moldura de página em memória, movendo-a temporariamente para o disco, de modo a liberar o espaço necessário para carregar o quadro de página o qual gerou o evento de *page miss*.

Em sistemas de 64 bits é impraticável implementar o mapeamento de 1:1 (um-para-um), ou seja, uma entrada na tabela virtual para mapear uma moldura de página real em memória. Por exemplo, assumindo molduras de páginas de 4KB em memória principal, um sistema de 64 bits demandaria um total de 2^52 entradas em uma tabela virtual de mapeamento 1:1. Se cada entrada na tabela virtual, por exemplo, contiver 8 bytes (i.e., o tamanho de uma palavra de 64 bits) para acomodar os campos de controle para o mapeamento (e.g., *flags* - cache, modificada, referenciada, proteção, presente/ausente, juntamente com o *offset* de deslocamento dentro da página de 4KB), a tabela virtual toda teria um tamanho de 32 de Peta Bytes.

Para contornar este problema, a abordagem de tabelas de páginas invertidas prevê manter somente os mapeamentos existentes em uma tabela virtual de comprimento proporcional ao total de molduras de páginas da memória principal. Por exemplo, uma memória física de 4GB demandaria uma tabela de 1 milhão (mega) de entradas para molduras páginas de 4KB. Nessa caso, essa tabela de 1M entradas seria uma tabela *hash*, onde cada entrada seria um ponteiro para uma lista, de modo que cada elemento da lista então corresponderia a um mapeamento da tabela virtual a qual estaria indexada sobre um mesmo *hash*.

Neste trabalho, implemente a abordagem de MMU baseada em tabela de páginas invertidas. Para fins de verificação e validação, considere um sistema de menor capacidade, como o ilustrado na figura abaixo, com

uma memória física de 256MB e uma tabela *hash* de 64K entradas. Assuma que a memória virtual consiga duplicar o tamanho da memória física existente. Preencha algumas posições da tabela *hash* de forma aleatória (ou seja, posições as respectivas listas contendo mapeamentos hipotéticos) para testar/validar sua implementação. Como *inputs* para testar sua implementação, gere endereços virtuais aleatórios para simular os endereços requisitados pela CPU. Caso o endereço um virtual esteja mapeado em um moldura de página presente em memória, imprima o conteúdo dos campos de controle do mapeamento. Caso contrário, imprima apenas uma mensagem indicando a ausência do mapeamento (i.e., “page miss”).



Como exemplo (apenas) para iniciar a sua implementação, considere o código de cabeçalho abaixo. Um registro `map_virtual_t` descreve os campos de controle em uma entrada na tabela virtual invertida (ou seja, um elemento de uma lista) contendo o respectivo mapeamento para a respectiva moldura de página real. A função `busca_map_virtual()` recebe como parâmetro um endereço virtual a ser buscado e retorna um ponteiro do tipo `map_virtual_t`. Caso o endereço `virtual_addr` esteja mapeado em uma moldura de página presente na memória, a função retorna um ponteiro para a respectiva entrada `map_virtual_t` na tabela de páginas invertidas. Caso contrário, a função retorna `NULL`.

```
#include <stdint.h>

struct map_virtual_t{
       uint8_t flag_cache:1;
       uint8_t flag_referenciada:1;
       uint8_t flag_modificada:1;
       uint8_t flag_protecao:3;
       uint8_t flag_padding:2; // inutilizado,apenas para preenchimento
       uint16_t offeset_moldura:12;
       uint16_t offeset_padding:4; // inutilizado, apenas para preenchimento
};

typedef struct map_virtual_t map_virtual_t;

map_virtual_t *busca_map_virtual(uint64_t virtual_addr);
```