Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Sistemas Operacionais (DCC605) - 2014/2
Professor: Ítalo Cunha.

Trabalho Prático 2 - Memória Virtual.

Alunos: Vitor Guilherme Lopes, Elias Soares, Nivaldo Teixeira e Jean-Luc Coelho.

Introdução:

Nesse trabalho iremos implementar o funcionamento da memória virtual em um sistema operacional. Temos um programa em C que simula um controlador de memória de uma determinada arquitetura. A arquitetura tem endereços virtuais têm 24 bits e esses endereços virtuais enumeram palavras de inteiros de 32 bits. São endereçados palavras e não bytes.

Perguntas a serem respondidas:

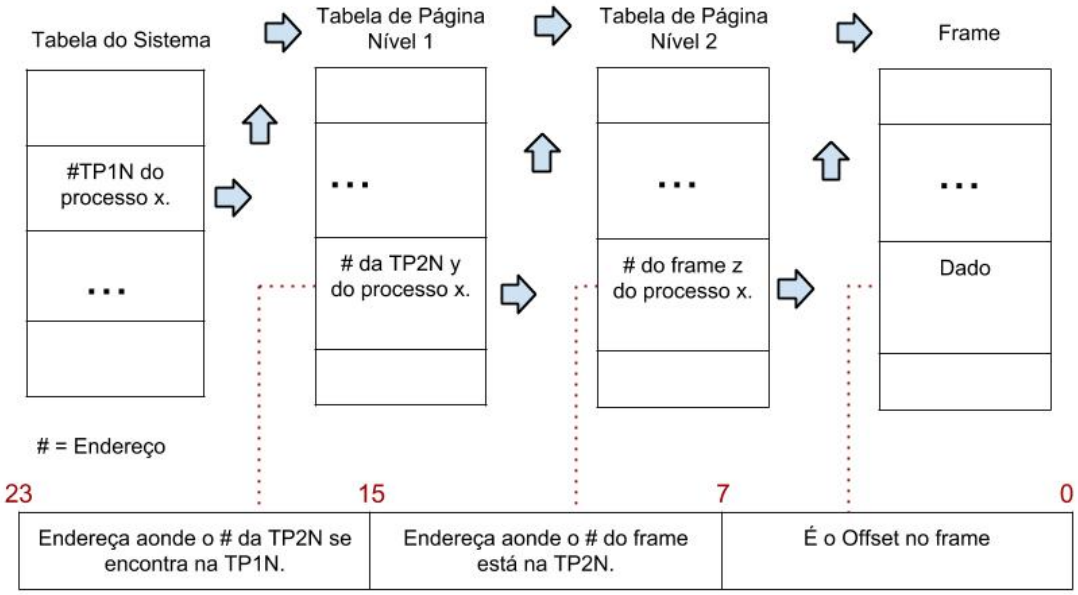
Questão: Qual a memoria física disponível no computador? Qual a quantidade de bits necessária nos endereços físicos?
Resposta: Cada frame armazena 256 palavras, a memória tem 4096 frames, desses 4096, o SO usa 16 frames para guardar dados importantes, sobrando 4080 frames, temos então capacidade de armazenar 1044480 palavras.

Questão: O que acontece quando os_pagefault retorna o valor VM_ABORT? Cite um exemplo onde os_pagefault retorna VM_ABORT.
Resposta: O programa para de funcionar. Quando, por esxemplo, passa uma entrada para a tabela de nível 1 inválida.

Questão: Por que o valor pte (page table entry) nunca sera confundido com o valor de VM_ABORT que pode ser retornado no if acima?
Resposta: Por causa de como os bits estão organizados. Em hexadecimal o VM_ABORT é 0x80000000, valor do qual o pte nunca será igual.

Questão: Descreva o funcionamento do controlador de memoria analisando o código das funções dccvmm_read, dccvmm_write, e dccvmm_get_pte. Explique como um endereço virtual é convertido num endereço físico. Faca um diagrama da tabela de paginas.
Resposta: A resposta será bem completa, explicando todas as tabelas, incluindo a do nosso sistema. No controlador de memória tem-se uma paginação multi-nível, no nosso sistema, temos uma tabela chamada Tabela do Sistema, que guarda o endereço da

tabela de página de primeiro nível (TP1N) do processo, cada processo pode ter apenas uma TP1N, cada TP1N armazena 256 endereços de tabelas de segundo nível (TP2N), com isso um processo pode ter até 256 TP2N, já as TP2N armazenam os endereços dos frames onde o dado se encontra. Abaixo segue uma visualização do que acontece:



Quando o endereço virtual chega no write ou read, eles chamam o get_pte para que esse “quebre” o endereço virtual para que cada parte enderece uma tabela, no diagrama acima mostra quais bits do endereço virtual endereça cada tabela.

Questão: Qual o tamanho do disco? Qual o minimo de processos que o sistema de memoria virtual suporta?
Resposta: O disco possui 2^17 = 131.072 = 128K setores, onde cada setor possui o mesmo tamanho de um frame = 2KB. Portanto, temos um disco de tamanho 256MB.

Questão: Qual o máximo de memoria que um processo pode endereçar?
Resposta: 4.294.967.296 (4 gigas) posições de memória de 32 bits (4 bytes).

Questão: As macros abaixo extraem informações de um endereço virtual. Explique qual informação é extraída por cada macro.
Resposta: O nosso esquema oferece uma sistema de paginação multinível onde há uma paginação na memória e outra paginação entre as páginas que estarão presentes na memória. A macro PTE1OFF(addr) identifica se o endereço passado corresponde à mesma seção em que a página desejada se encontra. A macro PTE2OFF(addr)

identifica qual página daquela seção é desejada. E a macro PAGEOFFSET(addr) identifica o offset dentro da página desejada.

Questão: Qual o tamanho em bytes de cada quadro de memoria física?
Resposta: Um quadro é formado por um array de 256 elementos do tipo uint32_t que possui 32 bits (4 bytes). Portanto, cada frame possui 256 elementos de 4 bytes = 1024 bytes ou 1KB.

Questão: Quantas palavras existem em cada quadro de memoria fisica?
Resposta: Cada quadro possui 256 elementos de 4 bytes. Portanto, existem 256 palavras dentro de cada quadro de memória física.

Sobre o SO:

Gerenciamento de quadros livres: A memória possui 4096 frames. Pegaremos 4096 primeiros bits da memória de sistema para mapearmos cada bit a um frame da memória. Se o bit é 0, o frame está livre e se for 1 o frame correspondente está sendo usado. Usando assim, meio frames, 128 palavras, para mapear os frames livres.
Inicialmente toda a memória é livre, quando o sistema se inicia, a memória por ele utilizada é atualizada para usada. Ao alocar a memória, procuramos frames livres, com a função procurar_frame_livre_dados( ), ela retorna o número do frame livre, depois de encontrado atualizamos como ocupado. Para liberar a memória, assumimos que o os_free( ) será usado para liberar apenas memória do processo corrente. Nessa função, nós olhamos se o endereço é múltiplo do tamanho do frame, se não for abortamos, conferimos se as tabelas de páginas são válidas, ou seja, se o frame realmente pertence ao processo corrente, se não pertencer abortamos, se tudo ocorreu corretamente, liberamos o dado e atualizamos o bit como livre.

Tratamento de falhas de páginas: Nosso tratamento de falha de páginas verifica se a entrada na tabela de páginas é válida e verifica se as permissões estão compatíveis.

Gerenciamento de processos ativos e suas tabelas de páginas: Quando um processo é criado, por alloc ou por swap, é atribuida uma entrada na tabela de sistema (conforme explicado no diagrama de tabelas lá em cima) para abrigar a tabela de nível 1 deste processo. Se for por swap, é apenas isso que é feito, se for por alloc, criamos a tabela de nível 1, tabela de nível 2 e pegamos um frame para alocar o dado.

Armazenamento de quadros no disco: A parte 5 do TP foi implementada pela metade por falta de tempo.
Parte implementada:
Reserva 128 setores do disco para metadados de utilização do disco. A função dump_setor_livre procura um setor livre, armazena o quadro nele e retorna o endereço. A função restaurar_setor resgata um setor do disco e armazena ele no quadro especificado,

como o dccvmm_load_frame. A diferença é que o restaurar_setor seta os metadados de utilização.

Parte não implementada:
Definir um bit no endereço virtual que diria se o endereço é de disco ou de RAM. A parte difícil seria modificar o acesso ao membro caso ele estivesse no disco.
Como uma política de troca de páginas ainda não estava implementada, não foi possível implementar essa parte sem perder uma quantidade de pontos considerável no trabalho.

Outras observações:
Não pode ter processo com id 0, porque quando a memória é limpa, coloca-se zero em tudo e quando há buscas de processo na memória, procura-se o id, se tiver um processo com id zero, não será possível saber se é id de memória limpa ou id do processo mesmo.
Percebemos um erro em um comentário a respeito do endereçamento da memória virtual, onde o correto seria de 0 a 23 estava de 0 a 24. Foi um erro que nos atrasou até termos um bom entendimento a respeito do programa. O erro foi postado no Moodle pelo aluno Vitor.
Por problemas de declaração de variáveis, tivemos que colocar algumas variáveis que estavam na vmm.c na vmm.h, modificando ligeiramente o código original.