UNIVERSIDADE DO VALE DE ITAJAÍ
ESCOLA POLITÉCNICA

ALEXANDRE DEBORTOLI DE SOUZA
BERNAR FREITAS DUARTE

Memória Principal e Memória Virtual

AVALIAÇÃO 03

GRADUAÇÃO EM ENGENHARIA E CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA DE SISTEMAS OPERACIONAIS
PROFESSOR FELIPE VIEL

ITAJAÍ, OUTUBRO DE 2023

REPOSITÓRIO

https://github.com/alexandredebortoli/os-memory-management-project

AV3

Enunciado:

Projeto Suponha que um sistema tenha um endereço virtual de tamanho entre 16 bits à 32 bits com deslocamento na página de 256 b à 4 Kb. Escreva um programa que receba um endereço virtual (em decimal) na linha de comando ou leitura do arquivo addresses.txt faça com que ele produza o número da página e o deslocamento do endereço fornecido, sendo que essa posição indica qual a posição que será lido do arquivo data_memory.txt. Você irá encontrar esses arquivos no github da disciplina, mais especificamente na pasta Memory (link repositório).

Por exemplo, seu programa seria executado da seguinte forma:
./virtual_memory_translate.exe 19986 ou ./virtual_memory_translate.exe addresses.txt

Seu programa produzirá:
• O endereço 19986 contém: o número da página = 4
o deslocamento = 3602
o Valor lido: 50 (exemplo)

No caso, o número em binário é 0100 1110 0001 0010, sendo que 0100 diz respeito à página e 1110 0001 0010 diz respeito ao deslocamento na página. Você consegue conferir isso com a calculadora do Windows/Linux no modo programador. Para manipular os números em nível de bit, é recomendado usar os operadores bitwise (bita-bit) da linguagem escolhida. No caso o exemplo apresentado é para 16 bits. No caso de 32 bits, haveria mais 16 bits a esquerda (mais significativo) referentes ao número de páginas, 0000 0000 0000 0000 0100 1110 0001 0010, porém, ainda será traduzido para página 4 e deslocamento 3602.

Escrever este programa exigirá o uso do tipo de dados apropriado para armazenar 16 à 32 bits (short ou int). É recomendado que você também use tipos de dados sem sinal. Além disso, para endereços de 32 bits deve ser possível usar

paginação hierárquica de 2 níveis mantendo 4 Kb, com cada nível tendo 10 bits de tamanho. Para a implementação do código, você pode fazer um fork do repositório da disciplina no

github e usar o codespaces do github para a implementação, onde o mesmo irá executar o Visual Code em uma distro Linux Ubuntu com 2 núcleos e 8 GB de memória principal. Mas onde será executado seus códigos fica a critério do(s) aluno(s)

Solução Proposta Para Paginação:

Desenvolvemos um programa em C++ que é capaz de receber endereços virtuais tanto por meio de argumentos na linha de comando quanto a partir de um arquivo chamado 'addresses.txt'. O programa oferece três versões da função 'calculate' para lidar com endereços virtuais de 16 e 32 bits, e suporta tanto a paginação normal quanto a de dois níveis no caso de endereços de 32 bits. Essas sobrecargas da função compartilham princípios semelhantes, diferindo apenas no tipo da variável utilizada: 'unsigned short' para endereços de 16 bits e 'unsigned int' para endereços de 32 bits. No caso da paginação de dois níveis, com 32 bits, o parâmetro 'pageOffsetBytes' é fixado em 4096 bytes, conforme especificado no enunciado.

A lógica da função 'calculate' para endereços de 16 bits é a mais detalhada, servindo de base para as outras variações da função. O objetivo da função é calcular três elementos distintos: o número da página, o deslocamento e o valor do dado lido.

Para encontrar o número da página, realizamos um deslocamento de bits à direita no endereço virtual, usando o número de bits do deslocamento (calculado a partir do logaritmo na base 2 da variável 'pageOffsetBytes').

Para o deslocamento, aplicamos uma máscara ao endereço virtual. Essa máscara é obtida subtraindo 1 do valor 'pageOffsetBytes' e, em representação binária, consiste em uma sequência de bits '1' onde o deslocamento se encontra no endereço virtual, e '0' no restante dos bits (que representam o número da página).

Por fim, para obter o valor do dado no endereço, utilizamos a função 'getDataByPagination', a qual recebe o índice desejado para encontrar o dado no arquivo 'data_memory.txt' e percorre linha a linha esse arquivo em procura do dado no índice especificado. O cálculo desse índice é realizado multiplicando o número da página pelo tamanho de deslocamento de cada página e adicionando o valor do deslocamento.

Implementação:

Função Calculate 16 bits

```
void calculate(unsigned short virtualAddressDecimal, unsigned short pageOffsetBytes)
{
    std::cout << "O endereço " << virtualAddressDecimal << " contém:" << std::endl;

    unsigned short pageNumber = (virtualAddressDecimal >> logBase2(pageOffsetBytes)); // Right-shift
(remove the offset, least significative bits).
    std::cout << "\t* número da página = " << pageNumber << std::endl;

    unsigned short offset = (virtualAddressDecimal & (pageOffsetBytes - 1)); // Extract the offset, least
significative bits, by masking.
    std::cout << "\t* deslocamento = " << offset << std::endl;

    const unsigned int targetIndex = ((pageNumber * pageOffsetBytes) + offset);
    unsigned short dataAtAddress = getDataByPagination(targetIndex);
    std::cout << "\t* Valor lido: " << dataAtAddress << std::endl;
}
```

Função Calculate 32 bits

```
void calculate(unsigned int virtualAddressDecimal, unsigned short pageOffsetBytes)
{
    std::cout << "O endereço " << virtualAddressDecimal << " contém:" << std::endl;

    unsigned int pageNumber = (virtualAddressDecimal >> logBase2(pageOffsetBytes)); // Right-shift (remove
the offset).
    std::cout << "\t* número da página = " << pageNumber << std::endl;

    unsigned short offset = (virtualAddressDecimal & (pageOffsetBytes - 1)); // Mask out the page bits.
    std::cout << "\t* deslocamento = " << offset << std::endl;

    const unsigned int targetIndex = ((pageNumber * pageOffsetBytes) + offset);
    unsigned short dataAtAddress = getDataByPagination(targetIndex);
    std::cout << "\t* Valor lido: " << dataAtAddress << std::endl;
}
```

Função Calculate 32 bits - 2 níveis

```
void calculate(unsigned int virtualAddressDecimal)
{
    const unsigned short pageOffsetBytes = 4096;    // 2 ^ 12 = 4096
    const unsigned short pageLevelSizeBytes = 1024; // 2 ^ 10 = 1024

    std::cout << "O endereço " << virtualAddressDecimal << " contém:" << std::endl;

    unsigned short outerPageNumber = (virtualAddressDecimal >> (logBase2(pageOffsetBytes) +
logBase2(pageLevelSizeBytes))); // Right-shift (remove offset + inner page level).
    std::cout << "\t* número da página de fora = " << outerPageNumber << std::endl;

    unsigned short innerPageNumber = ((virtualAddressDecimal >> logBase2(pageOffsetBytes)) &
(pageLevelSizeBytes - 1)); // Right-shift (remove the offset) + Mask out outer page level.
    std::cout << "\t* número da página de dentro = " << innerPageNumber << std::endl;

    unsigned short offset = (virtualAddressDecimal & (pageOffsetBytes - 1)); // Mask out both outer and inner
pages.
    std::cout << "\t* deslocamento = " << offset << std::endl;

    const int targetIndex = ((outerPageNumber * pageLevelSizeBytes) + (innerPageNumber *
pageOffsetBytes) + offset);
    unsigned short dataAtAddress = getDataByPagination(targetIndex);
    std::cout << "\t* Valor lido: " << dataAtAddress << std::endl;
}
```

Solução Proposta Para Paginação hierárquica de 2 níveis:

Para tratar a paginação hierárquica de dois níveis, introduzimos uma sobrecarga da função 'calculate' que recebe apenas o endereço virtual. Nessa abordagem, determinamos o número da página externa realizando um deslocamento à direita no endereço virtual, considerando a soma dos bits de deslocamento de página (12 bits) e dos bits do tamanho da página interna (10 bits). O número da página interna, por sua vez, é calculado com um deslocamento à direita no endereço virtual, utilizando o valor dos 12 bits de deslocamento de página e uma máscara que abrange os bits correspondentes à página externa. O cálculo do deslocamento envolve a aplicação de uma máscara nos bits que correspondem à soma da página externa com a página interna. Quanto ao valor do endereço, ele segue a mesma lógica descrita anteriormente, mas com uma complexidade adicional devido à paginação de dois níveis. O valor é obtido através da função 'getDataByPagination', e o cálculo do índice desejado envolve mais um nível, definido como:

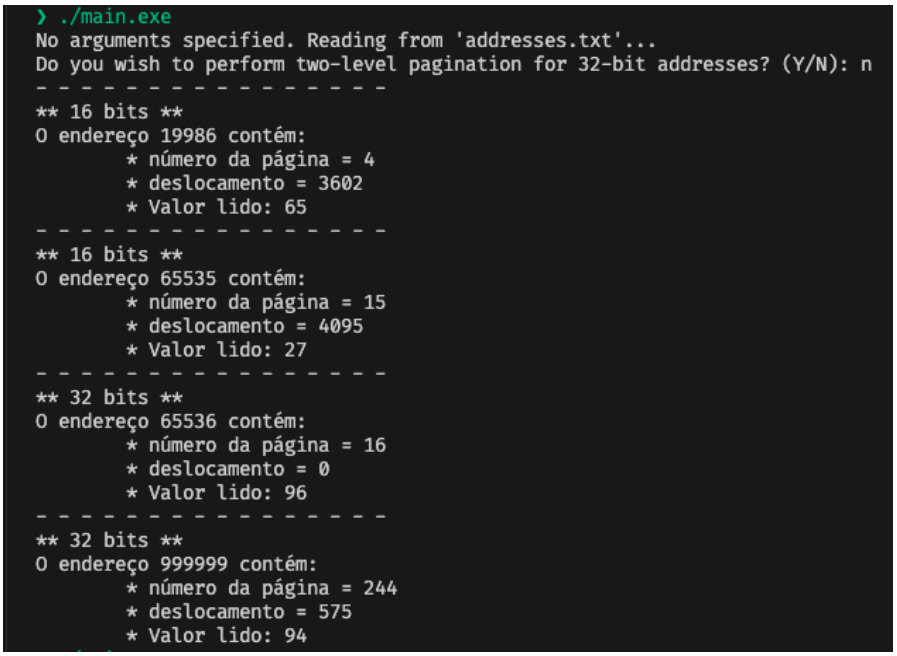
((outerPageNumber * pageLevelSizeBytes) + (innerPageNumber * pageOffsetBytes) + offset)

Essa fórmula considera o número da página externa, o tamanho da página interna e o deslocamento, permitindo a localização eficiente do dado correspondente no arquivo 'data_memory.txt'.

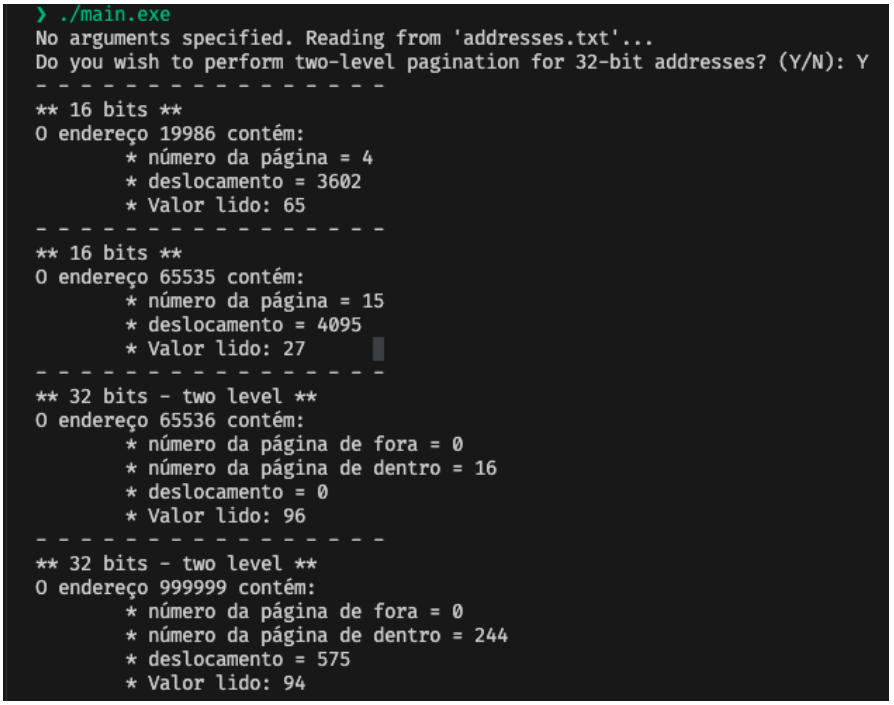
Resultados Obtidos com a Implementação:

A seguir, apresentam-se alguns resultados obtidos com a implementação a partir dos endereços especificados no arquivo 'addresses.txt':

Paginação Normal:



Paginação de Dois Níveis:



Conclusão:

A implementação do projeto de gerenciamento de memória virtual fornece uma visão detalhada de como os endereços virtuais são convertidos em números de página e deslocamentos, além de demonstrar a leitura de dados da memória com base nesses cálculos. O projeto abrange tanto a paginação normal quanto a de dois níveis, permitindo aos usuários entender e comparar esses dois métodos de gerenciamento de memória.

É importante observar que, devido à forma como os dados de memória são lidos a partir do arquivo data_memory.txt, os resultados entre a paginação normal e a de dois níveis não mostraram diferenças significativas. Isso ocorre porque a aplicação lê os dados linha por linha até encontrar o índice requerido. Para uma comparação mais significativa entre esses métodos de paginação, seria necessário manipular o arquivo de dados em estruturas como vetores e matrizes, o que permitiria um acesso mais direto aos dados, refletindo de forma mais precisa as características de cada método de paginação.

Além disso, é importante destacar que o número de páginas externas (outer pages) sempre permanece como 0. Isso ocorre devido à falta de dados suficientes no arquivo data_memory.txt para alcançar um número de páginas externas significativo. Para que fosse necessário o uso de páginas externas, seria preciso um dado no endereço virtual com um valor de 2^(12 bits de offset + 10 bits de página interna), o que não está presente nos dados do arquivo atual.

Em resumo, o projeto oferece uma oportunidade valiosa para explorar e compreender o funcionamento da memória virtual em sistemas operacionais, bem como os desafios associados à implementação eficaz de diferentes métodos de paginação.