LAB5: Relatório de implementação

GRUPO

- **DAVID FERREIRA QUARESMA**
- **EMANUEL JOIVO BEZERRA MARTINS**
- **RENATO DANTAS HENRIQUES**

ESTRUTURAS PARA ALGORITMOS DE MAPEAMENTO

- **MMU**
    - Representa a **M**emory **M**anagement **U**nit que implementa os algoritmos de mapeamento de endereço.
    - provê uma interface para mapeamento de memória virtual em memória física. O mapeamento é associado à implementação.
        - Linear: utiliza uma tabela salva em memória para mapear chave em valor, onde a chave é o id da página e o valor o endereço em memória. O 20 bits mais significativos do endereço virtual identificam a página, os 12 bits restantes o deslocamento na página.
        - Hierárquico: utiliza uma tabela (primeiro nível, ou nível mais externo) que aponta para outras tabelas (segundo nível, ou nível mais interno), as quais implementam a lógica do mapeamento linear. Os 10 bits mais significativos do endereço virtual identificam na primeira tabela a segunda tabela de página (de nível mais interno), os 10 bits seguinte (em significância) identificam a página na segunda tabela, os 12 bits restantes são para deslocamento na página.
        - Tabela Invertida: procura na tabela de processos o processo e a página identificada e o deslocamento identifica o endereço da página na memória. os 22 bits mais significativos do endereço virtual identifica o pid do processo, os 30 bits seguinte a página do processo, os 12 bits restantes referem-se ao deslocamento na página.

- **Mapeamento linear (tabela de um nível)**
    - Bytes necessários para armazenam a tabela de páginas em memória: 7340032 bytes (7MB)
        - cálculo:
            - bytes necessários para cada página na tabela de páginas: 7, 20 bits para o id da página na

tabela de páginas e 32 para o endereço da página na memória física.
                - número máximo de endereços possíveis: dois elevado ao número de bits dedicados para o endereço virtual da página -> 2 ** 20 = 1048576
                - Bytes necessários para armazenam a tabela: número de endereços possíveis vezes os bytes necessários -> 1048576 * 5 = 7340032
        - Quais estrutura utilizamos:
            - um dicionário em python para representar a tabela de páginas
            - uma instância PhysicalMemory com capacidade de 2GB e páginas do tamanho de 1k
            - Uma estrutura para gerenciar o swap usando o algoritmo de second chance
        - Formato de dados das PTEs:
            - Nossa PTE consiste apenas de um mapeamento simples, chave-valor, onde a chave é o id com 20 bits da página na tabela de páginas e o valor é o endereço físico onde a página começa, consistindo de 32 bits.

- **Mapeamento hierárquico (tabela de dois níveis)**
    - Bytes necessários para armazenar a tabela de páginas em memória: 6291456 bytes (6MB)
        - cálculo:
            - bytes necessários para cada página na tabela de páginas: 6, 10 bits para o id da página na tabela de páginas e 32 para o endereço da página na memória física.
            - número máximo de endereços possíveis: dois elevado ao número de bits dedicados para o endereço virtual da página no segundo nível -> 2 ** 10 = 1024
            - número de páginas de segundo nível: igual ao número de endereços na tabela de primeiro nível -> 2 ** 10 = 1024
            - Bytes necessários para armazenar a tabela: número de endereços possíveis na tabela de segundo nível vezes os bytes necessários vezes o número de tabelas de segundo nível -> 1024 * 1024 * 6 = 6291456
    - Quais estrutura utilizamos:
        - um dicionário em python para representar a tabela de páginas de nível mais externo (primeiro nível), onde

cada entrada dessa tabela aponta para outras tabelas de segundo nível
        - uma instância PhysicalMemory com capacidade de 2GB e páginas do tamanho de 1k
        - Uma estrutura para gerenciar o swap usando o algoritmo de second chance
    - Formato de dados das PTEs:
        - As PTEs dessa implementação consistem em um mapeamentos simples, chave-valor, onde a chave é o id com 10 bits da página na tabela de páginas e o valor é o endereço físico onde a página ou tabela começa, consistindo de 32 bits.

- **Inverted Mapping**
    - Bytes necessários para armazenar a tabela de páginas em memória: 14680064 bytes (14MB)
        - cálculo:
            - bytes necessários para cada página na tabela de páginas: 7, 22 bits para o PID do processo e 30 bits para o id da página na tabela de páginas.
            - número de páginas em memória: memória dividido pelo tamanho da páginas -> 8589934592 / 4096 = 2097152
            - número de páginas para a tabela de páginas: número de páginas vezes os bytes necessários para cada páginas na tabela, dividido pelo tamanho das páginas -> 2097152 * 7 / 4096 = 3584
            - Bytes necessários para armazenar a tabela: número de páginas para armazenar a tabela vezes o tamanho da página -> 3584 * 4096 = 14680064
    - Quais estrutura utilizamos:
        - um dicionário em python para representar a tabela de páginas de nível mais externo (primeiro nível), onde cada entrada dessa tabela aponta para outras tabelas de segundo nível
        - uma instância PhysicalMemory com capacidade de 8GB e páginas do tamanho de 4k
        - Uma estrutura para gerenciar o swap usando o algoritmo de second chance

TESTES EXECUTADOS

- **SIMPLE PAGE FAULT TEST**
    - Ao acessar uma página nova, espera:
        - 2 page faults para mapeamento hierárquico, 1 para os demais.
    - Ao acessar uma página já acessada com offset diferente, espera:
        - 0 page faults em todas as opções

- **HEAVY PAGE FAULT TEST**
    - Ao acessar um número N de páginas novas, espera:
        - no primeiro acesso:
            - 2 page faults para mapeamento hierárquico, 1 para os demais.
        - no segundo acesso:
            - 0 page faults em todas as opções
    - Ao preencher a memória e não haver mais endereço físico livre, espera:
        - ao acessar N endereços já acessados:
            - 0 page faults em todas as opções
        - ao acessar N endereços novos:
            - 2 page faults para mapeamento hierárquico, 1 para os demais.
        - ao acessar os N primeiro endereços acessados:
            - 2 page faults para mapeamento hierárquico, 1 para os demais.
                - justificativa: foram removidos nos N acessos anteriores devido a falta de memória endereço livre.

ESTRUTURAS AUXILIARES

- **Physical Memory**
    - Representa a memória física;
    - Contém como propriedades
        - Tamanho de memória
        - Tamanho de página
        - Tabela de endereços
    - Utilizamos um dicionário em Python como estrutura que identifica se a memória está em uso ou não.

- **Page**
    - Uma página foi definida contendo as seguintes propriedades
        - Um id de página que o identifica

        - Um id do processo ao qual a página está associada

- **FIFO**
    - Define o mecanismo básico de swap do algoritmo de mapeamento.
    - Guarda uma fila como estrutura básica.
    - Define as operações necessárias para manipulação dessa fila.

- **Frame**
    - Define cada unidade de espaço de endereçamento. Tem como propriedade apenas um ID que referencia onde o frame começa na memória física.

- **Second Chance**
    - Reutiliza o FIFO
    - Aprimora utilizando um bit de controle que sinaliza se as páginas mais antigas foram utilizadas ou não.
    - Define as operações de manipulação da pilha levando em consideração o bit de controle.