Universidade Federal de Lavras
PPGCC
PCC508 – Sistemas Operacionais

Tópico 5 Lista Avaliativa

Douglas Aquino T. Mendes
19 de novembro de 2024

Sumário

1 Introdução

Este documento tem como objetivo apresentar o desenvolvimento das atividades avaliativas para o tópico 5 da disciplina de Sistemas Operacionais, focando na implementação de códigos em linguagem C. Serão apresentadas as questões, a resolução, os códigos desenvolvidos, seguidos de uma explicação sobre sua lógica de funcionamento.

2 Questões

2.1 Algoritmo de substituição de páginas ótimo

Pergunta: 1) Poderíamos criar uma ferramenta para analisar cada um dos programas para prever quando cada página seria utilizada, de modo a implementarmos o algoritmo de substituição de páginas ótimo? Explique.

Resposta: Tanenbaum e Bos [1, p. 144] dizem que a criação de uma ferramenta para analisar programas e prever o uso futuro de páginas, com o objetivo de implementar o algoritmo ótimo de substituição de páginas, é teoricamente possível, porém impraticável em sistemas reais. O algoritmo ótimo de substituição de página postula que, em uma falta de página, a página que levará mais tempo para ser referenciada novamente deve ser removida da memória. Este algoritmo levaria ao menor número possível de faltas de páginas, mas seu problema reside na impossibilidade de prever com precisão quando cada página será utilizada no futuro.Embora seja impossível prever o futuro uso das páginas em um sistema em tempo real, o autor sugere que é possível simular o algoritmo ótimo em um ambiente controlado.

2.2 Not Recently Used

Pergunta: 2) Explique o algoritmo de substituição de páginas Not Recently Used (não usada recentemente), apresentando um exemplo.

Resposta: O algoritmo de substituição de páginas Not Recently Used (NRU), é um algoritmo que visa remover páginas da memória com base em seu uso recente e estado de modificação. O NRU utiliza dois bits principais associados a cada página: o bit R (Referenciada) e o bit M (Modificada) [1, p. 145].

Bit R (Referenciada): Este bit é configurado pelo hardware sempre que uma página é acessada, seja para leitura ou escrita. O sistema operacional limpa periodicamente o bit R para identificar páginas que não foram usadas recentemente.

Bit M (Modificada): Este bit é configurado pelo hardware quando ocorre uma escrita em uma página. Ele indica se a página foi modificada desde que foi carregada do disco. Com base nos valores dos bits R e M, o algoritmo NRU classifica as páginas em quatro classes:

- Classe 0: Páginas não referenciadas e não modificadas.
- Classe 1: Páginas não referenciadas, mas modificadas.
- Classe 2: Páginas referenciadas, mas não modificadas.
- Classe 3: Páginas referenciadas e modificadas.

Quando ocorre uma falta de página, o algoritmo NRU seleciona aleatoriamente uma página para remoção da classe de menor ordem não vazia. A lógica por trás dessa escolha é que é preferível remover uma página modificada, mas não referenciada recentemente (Classe 1), do que uma página não modificada que está sendo utilizada ativamente (Classe 2).

Exemplo do Algoritmo NRU

Imagine um sistema com 4 páginas na memória e a seguinte configuração de bits R e M, como ilustrado na tabela 1:

Página	R	M	Classe
0	1	0	2
1	0	1	1
2	1	1	3
3	0	0	0

Tabela 1: Valores de exemplo para os bits R e M

Se ocorrer uma falta de página neste momento, o algoritmo NRU escolheria aleatoriamente uma página da Classe 0. No caso, a única opção seria a página 3. Após a remoção da página 3, a nova página seria carregada em seu lugar, e seus bits R e M seriam inicializados de acordo com o estado da página. O algoritmo NRU é fácil de entender e implementar, e oferece um desempenho razoável, embora não seja ótimo.

2.3 Least Recently Used

Pergunta: 3) Considere o algoritmo de substituição de página "Least Recently Used" (LRU) implementado com um hardware que mantém uma matriz de n x n bits. No exemplo, n será de 4 bits. Agora imagine a seguinte ordem de referência: 3 3 2 1 2 3 0. Mostre a sequência de matrizes de acesso do LRU para a ordem dada. Qual página que seria escolhida para substituição, se fosse necessário retirar uma?

Resposta: O algoritmo de substituição de página Least Recently Used (LRU) pode ser implementado com uma matriz de bits n×nn×n, onde nn é o número de páginas na memória. Essa matriz é atualizada a cada acesso, de forma que a linha correspondente à página acessada tenha seus bits ajustados.

- Cada elemento da matriz $M[i][j]$ representa se a página i foi acessada mais recentemente que a página j. Se $M[i][j] = 1$, significa que i foi acessada mais recentemente que j.
- Quando uma página é acessada, a linha correspondente a essa página é definida com 1 em todas as suas posições, e a coluna correspondente a essa página é definida com 0.

Considerando $n = 4$ (páginas 0, 1, 2 e 3), a matriz de acesso M é uma matriz 44 inicialmente zerada. Dada a ordem de referência $3, 3, 2, 1, 2, 3, 0$, as matrizes são atualizadas conforme descrito a seguir:

$$M_{inicial} = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 0_{10} & 0_{11} & 0_{12} & 0_{13} \\ 0_{20} & 0_{21} & 0_{22} & 0_{23} \\ 0_{30} & 0_{31} & 0_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada : 3 = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 0_{10} & 0_{11} & 0_{12} & 0_{13} \\ 0_{20} & 0_{21} & 0_{22} & 0_{23} \\ 1_{30} & 1_{31} & 1_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada : 2 = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 0_{10} & 0_{11} & 0_{12} & 0_{13} \\ 1_{20} & 1_{21} & 0_{22} & 1_{23} \\ 1_{30} & 1_{31} & 0_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada : 1 = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 1_{10} & 0_{11} & 1_{12} & 1_{13} \\ 1_{20} & 0_{21} & 0_{22} & 1_{23} \\ 1_{30} & 0_{31} & 0_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada:2 = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 1_{10} & 0_{11} & 0_{12} & 1_{13} \\ 1_{20} & 1_{21} & 0_{22} & 1_{23} \\ 1_{30} & 0_{31} & 0_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada:3 = \begin{bmatrix} 0_{00} & 0_{01} & 0_{02} & 0_{03} \\ 1_{10} & 0_{11} & 0_{12} & 0_{13} \\ 1_{20} & 1_{21} & 0_{22} & 0_{23} \\ 1_{30} & 1_{31} & 1_{32} & 0_{33} \end{bmatrix}$$

$$PaginaReferenciada:0 = \begin{bmatrix} 0_{00} & 1_{01} & 1_{02} & 1_{03} \\ 0_{10} & 0_{11} & 0_{12} & 0_{13} \\ 0_{20} & 1_{21} & 0_{22} & 0_{23} \\ 0_{30} & 1_{31} & 1_{32} & 0_{33} \end{bmatrix}$$

Para identificar a página a ser substituída, verificamos a linha com o menor valor binário, pois ela não foi acessada a mais tempo em relação a todas as outras. Nesse caso, observando a matriz final, a página 1 seria escolhida para substituição, pois é a página com menor valor.

2.4 Page Fault Frequency

Pergunta: 4) O algoritmo PFF (Page Fault Frequency – Frequência de Falta de Página) é utilizado para controlar o tamanho do conjunto de páginas alocadas na memória RAM de um determinado processo, quando um algoritmo de alocação global é utilizado. Explique, com exemplos, como funciona este algoritmo.

Resposta:

Pergunta: 5) Explique como funciona a tradução de um endereço MULTICS para o endereço físico da máquina. O MULTICS trabalha com segmentação com paginação. Explique como funciona o descritor de segmento juntamente com as tabelas de páginas.

Resposta:

3 Desenvolver um programa

3.1

Enunciado: Nos dias atuais, sistemas com múltiplos cores dentro do processador estão em toda a parte. No Linux, podemos determinar a afinidade de uma determinada thread a um dado core com a função: sched-setaffinity. Faça um programa que gere 4 vertores de tamanho n com números aleatórios. Cada vetor deve ser ordenado por uma thread diferente. As threads devem ser divididas entre os cores existentes e cada thread deve sempre ser executada no mesmo core. Você deve verificar se a afinidade da thread foi escolhida corretamente com a função sched-getaffinity. Além disso, o comando top deve ser utilizado para a verificação do core que executa cada thread.

3.2 Código

3.3 Testes e Resultados

Como resultado da execução do código exibido na subceção 3.2, obtivemos a saída ilustrada na figura 1. É importante ressaltar que o programa pode apresentar um problema, pois não há verificação de produzir apenas quando há espaço, ou seja, o buffer pode acabar sendo subscrito pela função de produzir, antes da função de consumir.

```
douglas@Virtuoso:~/Documentos/Projetos/Disciplinas/Mestrado_SO/Topic4/Avaliativa/Codes$ ./atvi
Produtor: letra A adicionada na entrada 0
Consumidor: letra A consumida da entrada 0
Produtor: letra D adicionada na entrada 1
Consumidor: letra D consumida da entrada 1
Produtor: letra X adicionada na entrada 2
Produtor: letra N adicionada na entrada 3
Consumidor: letra X consumida da entrada 2
Produtor: letra K adicionada na entrada 4
Consumidor: letra N consumida da entrada 3
Produtor: letra O adicionada na entrada 5
Produtor: letra W adicionada na entrada 6
Consumidor: letra K consumida da entrada 4
Produtor: letra A adicionada na entrada 7
Consumidor: letra O consumida da entrada 5
Produtor: letra E adicionada na entrada 8
Produtor: letra H adicionada na entrada 9
Consumidor: letra W consumida da entrada 6
Produtor: letra L adicionada na entrada 0
Consumidor: letra A consumida da entrada 7
Produtor: letra T adicionada na entrada 1
Produtor: letra Q adicionada na entrada 2
Consumidor: letra E consumida da entrada 8
Produtor: letra U adicionada na entrada 3
Consumidor: letra H consumida da entrada 9
Produtor: letra J adicionada na entrada 4
```

Figura 1: Resultado da execução do programa

Referências

[1] A. S. Tanenbaum e H. Bos, *SISTEMAS OPERACIONAIS MODERNOS*, 4ª ed. Boston: Pearson, 2021.