Pontifícia Universidade Católica do Rio de Janeiro

Trabalho 2

Alunos:

2220373 Giovana Nogueira

2211667 Livian Essvein

Professor:

Luiz Fernando Seibel

Monitores:

Enzo Soares Abate

Rio de Janeiro,

Dezembro 2025

SUMÁRIO

1. Visão geral do projeto

O simulador desenvolvido tem como objetivo reproduzir o funcionamento básico de um sistema de memória virtual utilizando paginação. Para isso, o programa lê uma sequência real de acessos à memória, registrada em um arquivo .log, e processa cada acesso como se um processo estivesse sendo executado na máquina. A partir desses dados, o simulador realiza o mapeamento entre endereços lógicos e páginas, controla os quadros disponíveis na memória física e aplica diferentes algoritmos de substituição quando ocorre uma falta de página.

A estrutura geral do simulador é organizada em três componentes principais: a leitura e armazenamento dos acessos, a representação da memória física e a implementação dos algoritmos de substituição. O arquivo de entrada é carregado integralmente em um vetor de estruturas que guardam o endereço, o tipo de operação (leitura ou escrita) e o número da página calculado por meio do deslocamento adequado. Esse conjunto de acessos serve como base para toda a simulação, permitindo que cada algoritmo avance passo a passo pelos acessos reais realizados pelo programa original.

A memória física é representada por um vetor de frames, onde cada frame guarda informações essenciais para os algoritmos de substituição: validade do quadro, número da página carregada, bits R e M (indicando referência e modificação) e o instante do último acesso, utilizado para o comportamento do LRU. Em paralelo, o simulador mantém uma tabela de páginas que associa cada número de página ao índice do quadro em que ela se encontra. Essa tabela permite verificar rapidamente se um acesso resulta em um hit ou em um page fault.

Cada algoritmo utiliza esses dados de maneira diferente. O LRU se baseia no tempo do último acesso para escolher a página vítima, simulando a política de remover a página menos recentemente utilizada. O NRU segue a classificação clássica em quatro categorias formadas pelos bits R e M, escolhendo sempre a página com menor classe. Já o algoritmo Ótimo realiza uma análise futura, percorrendo os acessos que virão para descobrir qual página será utilizada mais distante no tempo, reproduzindo a estratégia teórica que minimiza o número de

faltas de página. A cada iteração, o simulador atualiza os bits de controle, registra page faults e contabiliza as escritas de páginas sujas, encerrando com um relatório contendo as estatísticas finais.

2. Análise de desempenho

compilador.log

Algoritmo	Página	Page Faults	Escritas
LRU	8 KB	21.091	3.839
LRU	16 KB	30.686	5.068
LRU	32 KB	38.641	6.186
NRU	8 KB	38.774	3.335
NRU	16 KB	36.684	4.052
NRU	32 KB	46.574	5.801
ÓTIMO	8 KB	10.118	2.190
ÓTIMO	16 KB	17.049	3.233
ÓTIMO	32 KB	23.758	3.750

matriz.log

Algoritmo	Página	Page Faults	Escritas
LRU	8 KB	4.745	1.623
LRU	16 KB	7.876	2.585
LRU	32 KB	13.230	3.877
NRU	8 KB	14.747	1.629
NRU	16 KB	15.071	2.123
NRU	32 KB	16.712	2.895
ÓTIMO	8 KB	2.969	1.109
ÓTIMO	16 KB	4.136	1.482
ÓTIMO	32 KB	8.008	2.618

compressor.log

Algoritmo	Página	Page Faults	Escritas
LRU	8 KB	255	0
LRU	16 KB	366	86
LRU	32 KB	515	180
NRU	8 KB	255	0
NRU	16 KB	709	0
NRU	32 KB	812	146
ÓTIMO	8 KB	255	0
ÓTIMO	16 KB	239	55
ÓTIMO	32 KB	334	109

simulador.log

Algoritmo	Página	Page Faults	Escritas
LRU	8 KB	8.556	3.395
LRU	16 KB	12.822	4.813
LRU	32 KB	22.126	7.003
NRU	8 KB	22.708	4.609
NRU	16 KB	21.794	5.690
NRU	32 KB	26.612	7.069
ÓTIMO	8 KB	4.748	2.160
ÓTIMO	16 KB	7.269	3.017
ÓTIMO	32 KB	11.870	4.421

Os testes de desempenho foram realizados mantendo a memória física fixa em 2 MB e variando o tamanho das páginas em 8 KB, 16 KB e 32 KB, para os programas compilador.log, matrix.log, compressor.log e simulador.log. Em todos os cenários observou-se que o aumento do tamanho das páginas provoca crescimento no número de faltas de página, consequência direta da redução do número de quadros disponíveis na memória, o que intensifica a frequência de substituições.

O algoritmo Ótimo apresentou consistentemente o melhor desempenho em todos os programas e configurações, com os menores valores de page faults. Esse resultado é esperado, uma vez que sua política se baseia no conhecimento antecipado das referências futuras, selecionando sempre como vítima a página cuja próxima utilização ocorrerá mais distante no tempo. O LRU teve desempenho intermediário, aproximando-se do Ótimo para páginas menores, mas sofrendo degradação conforme o número de quadros diminuiu, pois depende apenas do histórico de acessos passados e pode remover páginas que ainda seriam reutilizadas em breve.

O NRU apresentou o pior desempenho geral, com números mais elevados de faltas de página na maioria dos testes. Por utilizar apenas os bits R e M para classificação das páginas, o algoritmo não mantém uma ordem precisa de uso recente, o que limita sua eficácia em padrões de acesso mais intensivos e variados, como os observados nos programas avaliados.

O programa compressor.log foi a exceção entre os testes, apresentando baixos valores de faltas de página para todos os algoritmos. Isso indica um padrão de acesso mais local e previsível, reduzindo a necessidade de substituição de páginas independentemente da política utilizada.

Quanto às escritas de páginas sujas, verificou-se crescimento proporcional tanto ao aumento das faltas quanto ao tamanho das páginas, já que páginas maiores tendem a concentrar mais modificações antes da substituição. O algoritmo Ótimo apresentou novamente os menores valores de escrita, seguido pelo LRU, enquanto o NRU apresentou maior variação devido à sua política menos precisa de escolha de vítimas.

De forma geral, os resultados confirmam a expectativa teórica: o algoritmo Ótimo representa o melhor cenário possível, o LRU oferece uma boa aproximação prática desse ideal, enquanto o NRU apresenta desempenho inferior por sua abordagem mais simples.

3. Algoritmo Ideal

O algoritmo de substituição ótimo (ou ideal) é uma política teórica que supõe que o sistema sabe exatamente quais páginas serão acessadas no futuro. Sempre que ocorre uma falta de página e todos os quadros estão ocupados, ele escolhe como vítima a página que será utilizada mais tarde do que todas as outras, ou que nem chegará a ser utilizada novamente. Dessa forma, o algoritmo minimiza o número total de page faults, servindo como referência para comparar o desempenho de algoritmos reais como LRU e NRU.

A ideia básica é, para cada falta de página, olhar para frente na sequência de acessos e descobrir, para cada quadro, em qual posição futura aquela página será usada novamente. A página que não aparece mais no futuro é imediatamente escolhida como vítima. Se todas aparecerem novamente, é escolhida aquela cuja próxima utilização está mais distante.

Pseudo código para o algorítmo ótimo:

Entrada:

- *sequência de acessos A[0..N-1]*
- *número de quadros F*

Inicializar:

- *frames[0..F-1] vazios*
- *page_faults = 0*

```
Para i de 0 até N-1:
    página_atual = página(A[i])
    se página_atual já está em algum frame:
        // HIT: nenhuma substituição é necessária
        continue
    senão:
        // MISS: ocorre uma falta de página
        page_faults++
        se ainda existe algum frame vazio:
            carregar página_atual em um frame vazio
        senão:
            // todos os quadros estão ocupados, escolher vítima
            para cada frame f em frames:
                página_f = página armazenada em frames[f]
                posição_futura[f] = +infinito

                para j de i+1 até N-1:
                    se página(A[j]) == página_f:
                        posição_futura[f] = j
                        interromper o laço interno

            // escolher a página com maior posição_futura
            vítima = frame f com maior posição_futura[f]
            substituir a página em frames[vítima] por página_atual

Fim Para
Saída:
    - page_faults (número total de faltas de página)
```