Trabalho Prático 2

Braian Melo, Igor Cunha, Leonardo Ribeiro

2 de setembro de 2024

1 Introdução

O objetivo do trabalho prático é desenvolver um simulador de memória virtual usando os algoritmos de substituição de páginas discutidos em sala de aula, como o LRU (Least Recently Used), NRU (Not Recently Used) e Segunda Chance. Ao longo da documentação, iremos apresentar como cada algoritmo foi implementado e discorreremos sobre os pontos negativos e positivos de cada um.

2 Fluxo de funcionamento

2.1 Receber dados da linha de comando

```
./tp2 <algoritmo> <arquivo_de_entrada>.log <tamanho da pagina> <total de memoria disponivel>
```

Assim, o programa usa os argumentos `argc` e `argv` recebidos na função `main()` para verificar a validade das informações recebidas da linha de comando e continuar a execução. Esses argumentos representam, por ordem:

- O algoritmo de substituição a ser usado:
  - LRU.
  - NRU.
  - Segunda Chance.
- O arquivo de entrado que contém a sequência de endereços de memória acessados
- O tamanho de cada página/quadro de memória em KB.
- O tamanho total de memória física disponível para o processo em KB.

Assim que o programa inicia, ele checa se os argumentos foram passados corretamente. Caso positivo, ele cria um tabela de páginas de tamanho $n = \frac{tamanho\_pagina}{total\_memoria}$.

2.2 Ler e armazenar os dados do arquivo de entrada

O arquivo de entrada é formatado da seguinte forma: cada linha contém dois elementos. O primeiro é um número hexadecimal, n, que representa um endereço de memória acessado. O segundo é um caractere, *tipo*, que representa se o acesso é de leitura(R) ou escrita(W).

2.3 Processamento

Enquanto o arquivo de entrada é lido linha por linha, o programa faz a inserção das páginas na memória. Nessa parte, o código pode rodar de três maneiras diferentes, dependendo do algoritmo de substituição informado pelo usuário como argumento.

2.4 Informando os resultados obtidos na saída

Após o termino da leitura do arquivo de entrada e da execução do algoritmo de substituição de páginas, o programa irá imprimir algumas informações no terminal, entre elas:

- Nome do arquivo de entrada.
- Tamanho total da memória.
- Tamanho das páginas.
- Algoritmo de substituição.
- Quantidade de páginas lidas.
- Quantidade de páginas escritas.
- Page misses.
- Page hits.

Esses dados serão importantes para a comparação entre os diversos algoritmos implementados em diversos casos de teste.

3 TADs

3.1 Implementação das páginas

A nossa pagina foi implementada com os seguintes atributos:

- R: bit para indicar se a página foi referenciada. Esse bit é fundamental para o funcionamento dos três algoritmos.
- M: bit para indicar se a página foi modificada Usada bastante no NRU.
- $Valid$: bit para indicar se a página é válida memória.
- $Virtual\_page\_number$: número da página virtual.
- $count$: contador que registra o tempo do último referenciamento.

3.2 Hash

Para evitar a necessidade de percorrer constantemente a tabela de páginas durante a execução dos algoritmos de substituição, foi implementada uma tabela hash. A tabela facilita o processo de busca e gerenciamento das páginas de memória. Nea, cada entrada aponta para uma lista encadeada de páginas, onde cada nó da lista é uma estrutura chamada $Node$. Essa abordagem reduz o tempo de busca e melhora a eficiência na implementação dos algoritmos de substituição.

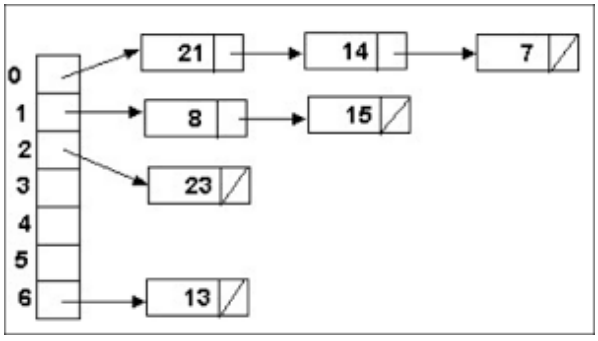

Figura 1: Exemplo de uma Hash implementada com lista encadeada

Abaixo está as funções da nossa hash:

```
#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c

// Pagina
typedef struct page {
    int r;
    int m;
    int valid;
    unsigned virtual_page_number;
    long long count;
}Page;

// Estrutura para armazenar uma pagina
typedef struct Node {
    Page value;
    struct Node* next;
} Node;


// Funcao hash simples para calcular o indice da chave
unsigned hash(unsigned virtual_page_number, unsigned num_pages);

// Funcao para criar um novo no
Node* create_node(unsigned virtual_page_number, long long count);

// Funcao para inserir um valor na tabela hash
void insert(Node* hash_table[], unsigned virtual_page_number, unsigned TABLE_SIZE,
    long long count);

// Funcao para buscar um valor na tabela hash
Node* search(Node* hash_table[], unsigned virtual_page_number, unsigned TABLE_SIZE);

// Funcao para remover uma pagina pelo numero de pagina virtual
void delete(Node* hash_table[], unsigned virtual_page_number, unsigned TABLE_SIZE);

// Funcao para liberar a memoria da tabela hash
void free_table(Node* hash_table[], unsigned TABLE_SIZE);

#endif
```

4 Algoritmos de substituição de páginas

Como visto anteriormente, três algoritmos de substituição de páginas foram implementadas: o LRU (Least Recently Used), NRU (Not Recently Used) e Segunda Chance.

4.1 LRU (Least Recently Used)

Baseando-se no princípio de que as páginas referenciadas recentemente têm mais chances de serem usadas novamente, o LRU (Least Recently Used) percorre toda a tabela de páginas e procura a página que tem o maior tempo de referenciamento.

```c
#include "lru.h"

unsigned lru(Node* page_table[], unsigned TABLE_SIZE){
    long long last_moment;
    unsigned last_page_num;
    for(int i = 0; i < TABLE_SIZE; i++){
        if(page_table[i] != NULL){
            last_moment = page_table[i]->value.count;
            last_page_num = page_table[i]->value.virtual_page_number;
            break;
        }
    }
    for(int i = 0; i < TABLE_SIZE; i++){
        Node* atual = page_table[i];
        while(atual != NULL){
            if(atual->value.count < last_moment){
                last_moment = atual->value.count;
                last_page_num = atual->value.virtual_page_number;

```

```
            }
            atual = atual->next;
        }
    }
    return last_page_num;
}
```

4.2 NRU (Not Recently Used)

O NRU percorre toda a tabela de páginas e divide as páginas em quatro classes diferentes. Em ordem de prioridade, elas são as seguintes:

1. Classe 0: páginas não referenciadas e não modificadas ($R = 0$ e $M = 0$)
2. Classe 1: páginas não referenciadas, mas modificadas ($R = 0$ e $M = 1$)
3. Classe 2: páginas referenciadas, mas não modificadas ($R = 1$ e $M = 0$)
4. Classe 3: páginas referenciadas e modificadas ($R = 1$ e $M = 1$)

Após dividi-las em três subgrupos, o NRU tenta substituir a primeira página da menor classe possível, começando pela classe 0. Caso não houver nenhuma, então ele tenta da classe 1 e assim por diante.

```
#include "nru.h"

unsigned nru(Node *page_table[], int num_frames){
    unsigned class0[num_frames];
    unsigned class1[num_frames];
    unsigned class2[num_frames];
    unsigned class3[num_frames];
    int id0 = 0, id1 = 0, id2 = 0, id3 = 0;
    for(int i = 0; i < num_frames; i++){
        Node* atual = page_table[i];
        while(atual != NULL){
            if(atual->value.m == 0 && atual->value.r == 0){
                class0[id0] = atual->value.virtual_page_number;
                id0++;
            }else if(atual->value.m == 1 && atual->value.r == 0){
                class1[id1] = atual->value.virtual_page_number;
                id1++;
            }else if(atual->value.m == 0 && atual->value.r == 1){
                class2[id2] = atual->value.virtual_page_number;
                id2++;
            }else if(atual->value.m == 1 && atual->value.r == 1){
                class3[id3] = atual->value.virtual_page_number;
                id3++;
            }
            atual = atual->next;
        }
    }
    if(id0) return class0[0];
    if(id1) return class1[0];
    if(id2) return class2[0];
    if(id3) return class3[0];
    return 0;
}
```

Entretanto, para o algoritmo, após certo tempo de uso, não ficar lotado de páginas de classe 3, ele percorre periodicamente a lista de páginas e muda o R para 0. Isso faz com que classes sejam classificadas com base no uso recente.

```
// main.c
if(count>1 && count % 1000 == 0){
    routine(page_table, num_pages);
}

// hash.c
void routine(Node* page_table[], unsigned TABLE_SIZE){
    for(int i = 0; i < TABLE_SIZE; i++){
        Node* atual = page_table[i];
```

```
        while(atual != NULL){
            atual->value.r = 0;
            atual = atual->next;
        }
    }
}
```

4.3 Segunda Chance

Esse algoritmo é uma variação do algoritmo FIFO (First In, First Out). Nele é implementado uma lista circular onde a página mais antiga, ou seja, a primeira que entrou na lista, é a primeira a ser retirada.

A diferença do Segunda Chance é que ele leva em consideração o bit R, ou seja, ele percorre a lista circular em busca da primeira página com $R = 0$, enquanto isso, ele vai alterando para $R = 1$ as páginas que ele já consultou. Sendo assim, se ele chegar a encontrar a página com $R = 0$, ele a substitui, senão, ele percorre toda a lista até ele retornar à primeira página e substituí-la.

```
#include "segunda_chance.h"

unsigned second_chance(Node* page_list, unsigned new_page,  unsigned num_pages){
    Node* atual = page_list;
    while(atual != NULL && atual->value.r == 1){
        atual->value.r = 0;
        atual = atual->next;
    }
    unsigned ans = atual->value.virtual_page_number;
    atual->value.virtual_page_number = new_page;
    atual->value.r = 1;
    return ans;
}
```

5 Comparação de resultados

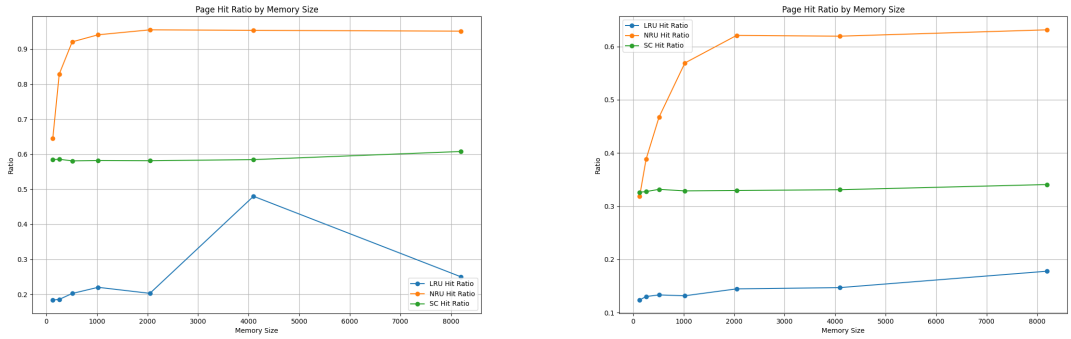

Figura 2: Taxa de page hit por quantidade de memória dos 3 algoritmos para o compressor e o compilador

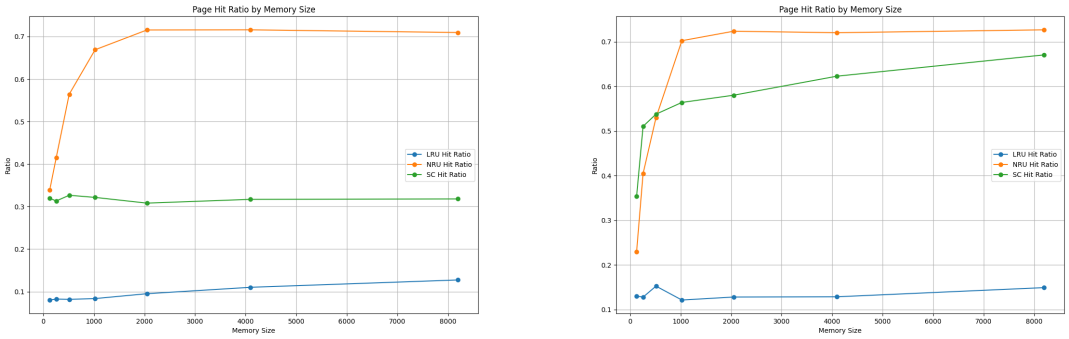

Figura 3: Taxa de page hit por quantidade de memória dos 3 algoritmos para o simulador e a matriz

Nos quatro gráficos apresentados nas figuras 2 e 3 é possível observar que no geral, com o aumento de memória disponível, a tava de page hit tende a aumentar, isso se dá ao fato de que com mais memória, é possível armazenar mais páginas na memória principal, e, com isso, a chance que uma página procurada já esteja lá aumente, porém é possível também ver que, em algumas simulações, alguns algoritmos mantiveram a taxa de page hit mesmo com o aumento de memória.

Nas análises realizadas, o algoritmo NRU se comportou melhor em todos os cenários.

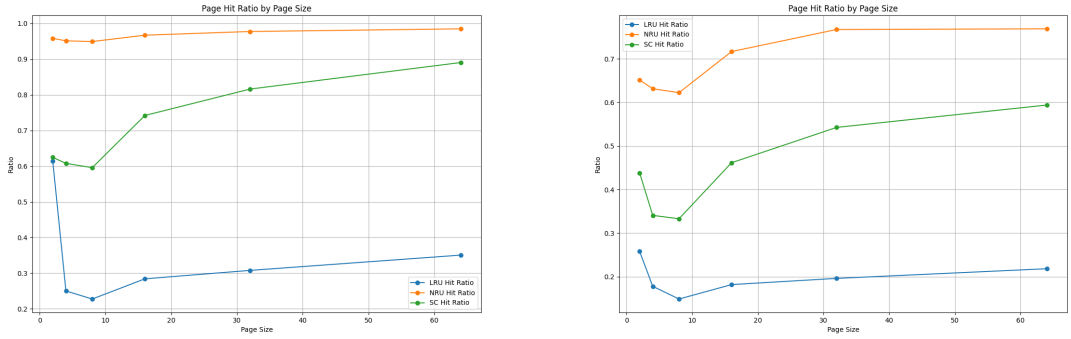

Figura 4: Taxa de page hit por tamanho de página dos 3 algoritmos para o compressor e o compilador

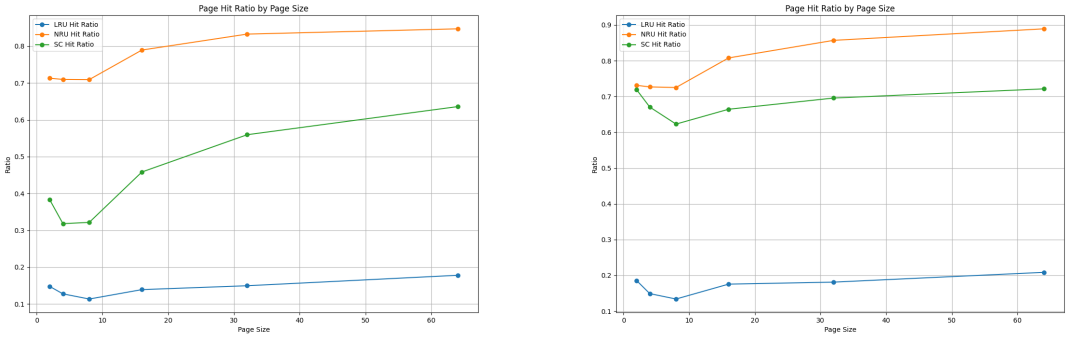

Figura 5: Taxa de page hit por tamanho de página dos 3 algoritmos para o simulador e a matriz

Nas figuras 4 e 5 estão representadas as taxas de page hit por tamanho de página, e assim como

na análise por quantidade de memória, o algoritmo NRU se comportou melhor em todos os cenários, porém, diferente da memória, aumentar o tamanho da página não necessariamente aumenta a taxa de page hit, inclusive, é possível observar que simulando o compressor, o algoritmo LRU teve um desempenho muito melhor com páginas de 2 kiloBytes do que com qualquer outro tamanho de página.

6 Conclusão

Neste trabalho, foram implementados e analisados os algoritmos de substituição de páginas LRU (Least Recently Used), NRU (Not Recently Used) e Segunda Chance, com o objetivo de avaliar o desempenho de cada um em diferentes cenários de gerenciamento de memória. Para isso, desenvolvemos um simulador de memória virtual em C, que inclui uma tabela de páginas invertida armazenada em uma estrutura de tabela hash, otimizando a eficiência das operações de busca e inserção de páginas.

Os testes realizados focaram na análise das taxas de acertos (page hits) e falhas (page misses) para cada algoritmo, variando a quantidade da memória e o tamanho da página. Os resultados mostraram que o algoritmo NRU consistentemente apresentou a melhor taxa de acertos em todos os cenários testados. O desempenho superior do NRU pode ser atribuído à sua abordagem de classificação das páginas em diferentes categorias, o que permite uma escolha mais criteriosa na hora de substituir uma página.

Por outro lado, os algoritmos LRU e Segunda Chance, embora apresentem boas taxas de acerto em determinados casos, não conseguiram superar o NRU em termos de eficiência geral. O LRU, conhecido por sua eficácia em ambientes onde o padrão de acesso às páginas é estável, foi prejudicado em cenários de acesso aleatório. O algoritmo Segunda Chance, apesar de ser uma variação simplificada do FIFO, demonstrou desempenho intermediário, mas ainda inferior ao NRU.

Em resumo, a implementação do NRU se mostrou a escolha mais robusta para os cenários analisados, destacando-se como a melhor estratégia de substituição de páginas entre as opções testadas. A utilização da tabela hash para a implementação da tabela de páginas invertida contribuiu significativamente para a otimização do desempenho dos algoritmos, permitindo uma gestão de memória mais eficiente. Este trabalho reforça a importância de se escolher o algoritmo adequado para cada contexto de uso, evidenciando como pequenas mudanças na estratégia de substituição de páginas podem impactar de forma significativa a performance do sistema.