# Trabalho Prático 2, Sistemas Operacionais

Vítor Rezende Silva

Maio 2023

## 1 Introdução

O objetivo desse trabalho é implementar um simulador de memória virtual na linguagem C.

## 2 Memória Virtual

A memória virtual surge com a necessidade de executar programas que são grandes demais para encaixar na memória principal do computador.

A ideia básica da memória virtual é que cada programa tem seu próprio espaço de endereçamento, o qual é dividido em blocos chamados páginas. Cada página é uma série de endereços, elas são mapeadas na memória física, mas nem todas precisam estar na memória física.

## 3 Paginação

Paginação é uma técnica empregada por sistemas que usam memória virtual. Essa técnica consistem em gerar endereços usando indexação, esses endereços gerados pelo computador são chamados de endereços virtuais, e formam o espaço de endereçamento virtual. O endereço virtual é depois mapeado para a memória física do computador. O espaço de endereçamento virtual consiste em unidades de tamanho fixo chamados páginas, e as unidades correspondentes na memória fisica são chamadas de quadro de página.

## 4 Tabela de páginas

Para poder mapear corretamente o endereço virtual para o endereço físico, o sistema operacional usa uma estrutura chamada tabela de páginas. Isso é feito da seguinte forma: o endereço virtual é dividido em um número de página virtual e um deslocamento. O número de página virtual é usado como um índice dentro da tabela de páginas para encontrar a entrada para essa página virutal. A partir da entrada, chega-se ao número do quadro. O número do quadro de página é

colocado com os bits mais significativos do deslocamento, substituindo o número de página virtual, a fim de formar o endereço físico.

# 5 Tabela de páginas invertidas

O sistema operacional pode precisar tratar de endereços virtuais muito grandes. Para isso é preciso repensar a estrutura da tabela de páginas para que não existam problemas. Uma estrutura usada para essas situações é a tabela de página invertida. Nessa estrutura, existe apenas uma entrada por quadro de página na memória física, no lugar de uma entrada por página de espaço. A entrada controla qual página virutal está localizado na moldura da página.

# 6 Substituição de páginas

Quando ocorre uma falta de página, o SO tem que escolher uma página para remover da memória a fim de abrir espaço para a que está chegando. Se a página a ser removida foi modificada enquanto estava na memória, ela precida ser reescrita para o disco a fim de atualizar a cópia no disco. Substituir uma página não é algo trivial quando deseja-se evitar futuras faltas de páginas, para isso algumas estratégias devem ser pensadas. Nesse trabalho serão implementados três algoritmos de substituição: o Least Recently Used (LRU), Not Recently Used (NRU), e o Algoritmo de Substituição de Páginas Segunda Chance.

## 6.1 Least Recently Used(LRU)

O LRU substitui aquela página que foi menos usada a mais tempo. Para sua implementação é preciso manter as páginas numa lista encadeada. Estruturando as páginas dessa forma, sempre que uma página for referenciada ela vai para a primeira posição da lista. Assim, o algoritmo deve substituir aquela página que ocupa a última posição da lista.

## 6.2 Not Recently Used(NRU)

O NRU substitui aquela página que não foi usada recentemente. Ele funciona da seguinte forma: quando ocorre uma falta de página, o sistema operacional inspeciona as páginas e as divide em quatro categorias

1. Não referenciada, não modificada
2. Não referenciada, modificada
3. Referenciada, não modificada
4. Referenciada, modificada

Separando as páginas nessas categorias, o algoritmo seleciona uma página ao acaso de sua categoria mais baixa que não esteja vazia.

### 6.3 Algoritmo de Substituição de Páginas Segunda Chance

Algoritmo baseado em algoritmos FIFO (First In, First Out), onde a primeira página a ser mapeada para a memória física deve ser a primeira a ser removida. O diferencial do algoritmo de segunda chance, é que além de verificar qual foi o primeiro endereço mapeado na memória fisica, ele também vai verificar se aquele endereço foi referenciado recentemente, se sim, ele será reposicionado no fim da lista, e seu tempo de carregamento será zerado.

## 7 Implementação

### 7.1 Parâmetros

O programa deve receber quatro parâmetros da linha de comando, a estratégia de substituição de página empregada, o nome do arquivo com os endereços que devem ser lidos, o tamanho de cada página(2 a 64KB) e o tamanho total da memória fisica(128 a 16384KB).

### 7.2 Estruturas

Para implementar o simulador da memória virtual, será necessário o uso de três TADs implementadas, a page_table, memory e page.

Memory é uma lista encadeada do tipo page, e page_table é uma tabela hash que será usada para simular uma tabela de páginas.

O primeiro passo para usar essas TADs é determinar o tamanho da page_table, isso é o número de entradas da tabela, para isso o algoritmo divide o número de páginas pelo tamanho das páginas, esse valor também será usado para determinar o tamanho máximo da memória e para realizar a função hash.

```
struct page{
    int addr;
    int values[3];
    struct page* next;
};

struct memory{
    struct page* first;
    struct page* last;
    int size;
    int max_size;
};

Memory** create_page_table(unsigned long int num_addr,
                           unsigned long int num_pages){
    Memory** m = (Memory**)malloc(num_addr*sizeof(Memory*));
    for(int i = 0; i <= num_addr; i++){
```

```
        m[i] = create_memory(num_pages);
    }
    return m;
}
```

A TAD page é composta por 2 valores, o endereço addr e um vetor que armazena 3 valores, o primeiro é o bit que indica se a página foi referenciada, o segundo indica se foi modificada e o terceiro é o instante de tempo em que a página foi mapeada, isso é feito com a biblioteca time.h.

### 7.2.1 Funcionamento

O algorimo vai ler as entradas do arquivo especificado na linha de comando, e encerra com o fim do arquivo.

A cada iteração ele recebe um endereço e a forma como ele vai acessar esse endereço, escrita ou leitura. Em seguida ele converte o endereço para uma página da tabela de páginas usando a seguinte função.

```
long int get_page(unsigned long int page_size,
                  unsigned long int addr){
    unsigned long int tmp = page_size;
    unsigned long int s = 0;
    unsigned long int page;
    while(tmp>1){
        tmp = tmp >> 1;
        s++;
    }
    page = addr >> s;
    return page;
}
```

Com a página calculada, ele tenta acessar a tabela de páginas usando o número da página como entrada, nisso pode ocorrer 3 situações:

- Página ja mapeada: Nesse caso, a página buscada ja foi mapeada na tabela de páginas.
- Página não mapeada, mas com espaços livres na memória: Nesse caso a página ainda não foi mapeada para a memória física, mas ainda existe espaço na linha de memória na tabela de páginas, assim a página é inserida na tabela.
- Página não mapeada e memória cheia: Nesse caso ocorre um page fault, a página não foi mapeada e um algoritmo de substituição deve ser empregado.

A cada operação de escrita, se a página já for mapeada, o valor que representa o bit modificado dela é alterado para 1, mas se for uma operação de leitura ele é mantido.

### 7.3 Substituição de Páginas

Os algoritmo de substituição de página foram implementados seguindo as descrições fornecidas nas seção 6.

## 8 Analise Experimental

Na analise experimental do código, será analisado o tempo gasto por cada algoritmo para processar as entradas de cada arquivo, o valor do tamanho de página e da memória fisica disponível será 4KB e 128KB em todos os testes

### 8.1 Least Recently Used

Tempo gasto para cada arquivo:

compilador.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual lru compilador.log 4 128
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: lru
Paginas lidas: 838622
Paginas escritas: 102672
Numero de acesso a memoria: 35190573
Numero de page faults: 1022
Numero de paginas sujas: 34248257
Tempo gasto: 0.326297
```

compressor.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual lru compressor.log 4 128
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: lru
Paginas lidas: 877581
Paginas escritas: 122419
Numero de acesso a memoria: 1000317
Numero de page faults: 317
Numero de paginas sujas: 0
Tempo gasto: 0.244485
```

matriz.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual lru matriz.log 4 128
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: lru
Paginas lidas: 901712
Paginas escritas: 57984
Numero de acesso a memoria: -4226893201
Numero de page faults: 975
Numero de paginas sujas: -4227853872
Tempo gasto: 0.275263
```

simulador.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual lru simulador.log 4 128
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: lru
Paginas lidas: 769020
Paginas escritas: 142645
Numero de acesso a memoria: 294540980
Numero de page faults: 1024
Numero de paginas sujas: 293628291
Tempo gasto: 0.314944
```

## 8.2 Not Recently Used

Tempo gasto para cada arquivo:

compilador.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual nru compilador.log 4 128
Executando o simulador...
Arquivo de entrada: compilador.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: nru
Paginas lidas: 823931
Paginas escritas: 99583
Numero de acesso a memoria: 2417829112
Numero de page faults: 1022
Numero de paginas sujas: 2416904576
Tempo gasto: 0.345035
```

compressor.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual nru compressor.log 4 128
Executando o simulador...
Arquivo de entrada: compressor.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: nru
Paginas lidas: 877581
Paginas escritas: 122419
Numero de acesso a memoria: 1000317
Numero de page faults: 317
Numero de paginas sujas: 0
Tempo gasto: 0.266624
```

matriz.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual nru matriz.log 4 128
Executando o simulador...
Arquivo de entrada: matriz.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: nru
Paginas lidas: 850290
Paginas escritas: 60636
Numero de acesso a memoria: 2929071510
Numero de page faults: 975
Numero de paginas sujas: 2928159609
Tempo gasto: 0.314475
```

simulador.log

```
vitor@vitor-VivoBook-ASUSLaptop-X570DD-X570DD:~/Documentos/SO/TP2$
prompt> ./tp2virtual nru simulador.log 4 128
Executando o simulador...
Arquivo de entrada: simulador.log
Tamanho da memoria: 128
Tamanho das paginas: 4
Tecnica de reposicao: nru
Paginas lidas: 790638
Paginas escritas: 153061
Numero de acesso a memoria: 2551113032
Numero de page faults: 1024
Numero de paginas sujas: 2550168309
Tempo gasto: 0.343055
```

## 8.3 Algoritmo de segunda chance

A implementação do algoritmo de segunda chance obteve mals resultados, embora ele conseguisse terminar sua execução, o tempo era longo demais. O algoritmo usa a função time() da biblioteca time.h, o que não foi uma boa decisão de implementação, uma vez que a diferença de tempo de uma página para outro é muito pequena, o que contribuiu para a ineficiência da implementação.

## 8.4 Analise dos resultados obtidos

O resultado obtido pelos algoritmos NRU e LRU foram muito semelhantes entre si, a diferença de um algoritmo para outro é muito pequena, claro que esse é um

resultado obtido para um mesmo tamanho de página e de memória, mas mesmo aumentando a diferença não é tão expressiva. Existem alguns pontos que chamar atenção nos resultados, o primeiro é na execução do arquivo compressor.log, que em ambos os casos não houve a substituição de páginas com bits sujos. Um outro resultado interessante é no número de acesso a memoria no arquivo matriz.log com o algoritmo LRU, onde o valor ficou negativo, mesmo que isso não faça sentido. A razão disso ocorrer é devido ao tamanho das variáveis. Embora o resultado não tenha sido apresentado, devido ao tempo excessivo de execução o algoritmo de segunda chance não se mostrou eficiente.

# 9 Desafios de implementação

A etapa mais desafiadora da implementação do trabalho foi compreender o funcionamento da memória virtual e como implementar um simulador, porém, a implementação em si do algoritmo não é tão complicada.

# 10 Conclusões

A implementação do trabalho é uma ótima forma de compreender com mais detalhes o funcionamento do sistema de memória virtual, desde a estrutura até o funcionamento dos algoritmos de substituição de página.