UNIVERSIDADE FEDERAL DE VIÇOSA
Campus de Rio Paranaíba
Sistemas de Informação

Trabalho Prático II: Implementando um Gerenciador de Memória

Nálbert Wattam Silva Mariano - 4856
Marlon Carvalho Silva - 5210
Arthur Henrique Resende Carvalho - 5166

Dezembro
2020

UNIVERSIDADE FEDERAL DE VIÇOSA
Campus de Rio Paranaíba
Sistemas de Informação

Trabalho Prático II: Implementando um Gerenciador de Memória

Relatório apresentado à Universidade Federal de Viçosa como parte das exigências para a aprovação na disciplina de Sistemas Operacionais.

Alunos: Nálbert Wattam, Marlon Carvalho e Arthur Henrique

Dezembro
2020

Conteúdo

1 Algoritmo de substituição de páginas FIFO

Quando todas as páginas físicas da memória já estão ocupadas, e uma nova página é requisitada, alguma página, já presente na memória, deve sair para ceder espaço para a nova que está chegando. Para decidir qual página será removida da memória física é necessário algum algoritmo que realize essa tarefa.

FIFO (*First in, first out*) ou **primeiro a entrar, primeiro a sair** é um algoritmo de substituição de página que decide qual página será removida em caso de uma falta de página. Nele todas as páginas inseridas na memória são colocadas no final de uma fila (ou lista) enquanto as mais antigas ocupam as posições mais a frente. Caso seja necessária uma troca de página a que estiver mais na frente será removida, cedendo assim espaço para a nova página.

Embora o algoritmo possua um baixo custo não é comumente usado, pois a página mais antiga não é necessariamente a menos utilizada, podendo ocasionar a troca de páginas frequentemente acessadas.

2 Abordagem utilizada no projeto

Embora seja comumente utilizada uma lista encadeada no algoritmo FIFO, no trabalho prático houve a criação de um vetor, visando simular a lista, já que sua implementação é muito mais simples. A Figura 1 mostra o trecho do código responsável pela criação do vetor.

```
int *fifoVar;
fifoVar = (int*) malloc(num_frames * sizeof(int));
for (int i = 0; i < num_frames; i++) {
    fifoVar[i] = -1; // É dado -1 a todas as posições do vetor, indicando que estão vazias
}
```

Figura 1: Criação do vetor fifoVar

É criado um ponteiro do tipo *int* que, em seguida, é dinamicamente alocado de acordo com o tamanho de molduras de página. Depois são atribuídos o valor -1 a todas as posições do vetor, para indicar que estão vazias.

Dentro da função *simulate()*, caso ainda exista memória física livre, é verificado se a última posição do vetor (a que indica a página a ser removida) possui o valor -1 (ou seja, se está vazia) e caso esteja é atribuído o índice da página em questão a posição do vetor. Caso a última posição não esteja vazia a próxima posição vazia do vetor recebe o valor da página atual. Isso

acontece até que todas as posições do vetor sejam preenchidas. O trecho de código responsável por isso pode ser observado na Figura 2.

```
// Caso a fila FIFO esteja vazia a primeira página trazida para a memória é inserida na fila
if (fifoVar[num_frames - 1] == -1) {
    fifoVar[num_frames - 1] = virt_addr;
}
// Caso a fila FIFO não esteja vazia a página é inserida na próxima posição vaga
else{
    fifoVar[*num_free_frames - 1] = virt_addr;
}
```

Figura 2: Preenchimento do vetor fifoVar

Na Figura 3 é apresentada a função *fifo()* que é responsável por retornar a página a ser trocada para a função *simulate()*. Ela recebe como parâmetro a última posição do vetor FIFO e retorna o valor para a função que a chamou.

```
int fifo(int8_t** page_table, int num_pages, int prev_page, int fifoVar, int num_frames, int clock) {
    int page;
    // Pega a última posição do vetor (primeira da fila) que indica a página a ser trocada
    page = fifoVar;

    return page;
}
```

Figura 3: Função *fifo()*

Após remover uma página é necessário reajustar o vetor, fazendo com que todas os valores avancem uma posição a frente, e colocando a página inserida no começo do vetor. Essa operação pode ser vista na Figura 4.

```
// Reajustando as posições do vetor vivoVar
int i = num_pages-1;
for(i = num_pages-1; i >= 0; i--){
    if(i>0){
        fifoVar[i] = fifoVar[i - 1];
    }
    else{
        fifoVar[i] = virt_addr;
    }
}
```

Figura 4: Reajustando as posições do vetor e inserindo nova página

3 Comparação entre os algoritmos Random e FIFO

A Tabela 1 apresenta o número de falta de páginas para cada algoritmo durante 10 execuções. Embora consiga trabalhar de maneira mais constante o algoritmo FIFO não apresentou maior desempenho em relação ao Random, ambos obtiveram uma média de 9 falhas de página por execução.

	Random	**FIFO**
Teste 1	10	9
Teste 2	11	9
Teste 3	9	9
Teste 4	7	9
Teste 5	9	9
Teste 6	8	9
Teste 7	9	9
Teste 8	8	9
Teste 9	10	9
Teste 10	9	9
Média	9	9

Tabela 1: Número de faltas de página para os algoritmos Random e Fifo ao longo de 10 testes