Segundo TP - Simulador de Memória Virtual

Sistemas Operacionais

Thiago Adriano, Tiago Trotta, Yan Victor

Maio 2019

1 Introdução

Para este trabalho, foi pedida a implementação, em linguagem C, de um simulador de memória virtual replicando um mecanismo real de gerência e suas particularidades.

2 Desenvolvimento

2.1 Funcionamento

Primeiramente, o programa lê o arquivo passado por parâmetro, que contêm os endereços de memória acessados. A cada endereço lido, o programa calcula a página virtual para a qual ele está mapeado.

Em seguida, é preciso verificar se a página virtual acessada está na tabela de página (e, por consequência, na memória), ou seja, se houve um *Page Hit*, caso no qual o simulador incrementa o contador de *Page Hits* e atualiza os bits referenciado e modificado da página virtual, necessários para os algoritmos de substituição. Para os casos em que a página virtual não está na tabela de página, ou seja, há um *Page Fault*, incrementa-se o contador de *Page Faults* e, em seguida, simula-se o acesso ao disco, para que a página seja adicionada.

Quando ocorre a simulação do acesso ao disco, para colocar uma página na tabela de página e na memória, o contador de leituras em disco realizadas é incrementado e os bits referenciado e modificado da página lida são atualizados. Por fim, a página pode ser adicionada. Caso a memória esteja cheia no momento de armazenar uma nova página, o programa precisará retirar uma página que já esteja na memória para liberar uma moldura. Para tal algum dos algoritmos de substituição de página será aplicado para escolher a moldura a ser liberada. Caso a página retirada esteja suja (i.e.: o bit modificado esteja ativo) é simulada a sua gravação no disco e o contador de páginas escritas no disco é incrementado.

2.2 Decisões de Projeto

Para lidar com as limitações da estrutura da tabela e com as particularidades dos algoritmo de substituição de página, algumas decisões precisaram ser tomadas.

2.2.1 Tabela de Páginas e Memória

Para contornar o problema do tamanho inviável de uma tabela de páginas com muias entradas, utilizamos uma tabela de página invertida, usando uma *hash*. Para tratar colisões, utilizamos uma lista encadeada para cada índice da tabela de páginas. Cada elemento da tabela é um TAD Página, que armazena os dados necessários para a simulação.

Não foi necessário implementar a memória de fato, pois não simulamos a alteração dos dados, sendo necessária, somente, a utilização de um vetor que representa o status das molduras da memória: livre ou ocupada.

2.2.2 Algoritmos de Substituição de Páginas

Para a implementação do algoritmo LRU, usamos um vetor cujo tamanho é o número de molduras na memória. Cada posição do vetor representa a moldura de mesmo índice na memória (i.e. índice 3 representa moldura 3) e armazena o tempo de relógio em que sua respectiva moldura foi referenciada. Desta forma, quando é necessário retirar uma página da memória, basta percorrer o vetor para encontrar a moldura que foi referenciada a mais tempo, tarefa bem mais simples do que percorrer as listas encadeadas da tabela de página. Assim, a página que está na moldura encontrada é a escolhida para ser retirada da memória.

De maneira similar, o algoritmo NRU utiliza o mesmo vetor utilizado pelo LRU, porém, desta vez, armazenando em cada posição do vetor a classe à qual pertence a página que está na respectiva moldura. As classes do algoritmo NRU podem ser: Classe 0: Não referenciada, não modificada; Classe 1: Não referenciada, modificada; Classe 2: Referenciada, não modificada; Classe 3: Referenciada, modificada. Assim, o vetor é percorrido para encontrar a moldura que está na classe mais baixa. A primeira moldura encontrada nessa classe é selecionada para que sua página correspondente seja retirada da memória.

O algoritmo de *SegundaChance* utiliza uma fila auxiliar de páginas, que representa todas as páginas presentes na memória. Dessa maneira, sempre que uma página é inserida na memória deverá, também, ser inserida nessa fila. Além disso, sempre que uma página é acessada na memória, seu bit referenciado é setado é ela é inserida no final da fila. O algoritmo percorre a fila em busca de uma página não referenciada. Caso a página atual tenha o bit referenciado ativo, ela recebe uma segunda chance, sendo inserida no final da fila e tendo o bit referenciado resetado. Assim, a página que será retirada da memória é a primeira página não referenciada encontrada na fila.

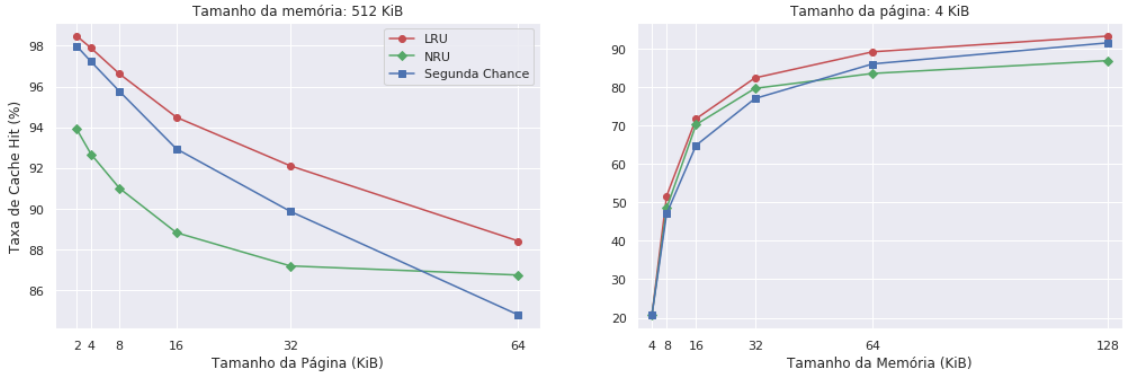
3 Análise de Resultados

Ao analisar os resultados, é possível observar que, com o aumento da capacidade da memória, há também o aumento da taxa de *Page Hits* para todas os algoritmos de substituição. De fato, quanto maior a capacidade da memória, maior o número de páginas que podem ser armazenadas e, consequentemente, maior a chance de um endereço acessado estar numa página virtual que já consta na tabela de página. Em contrapartida, nota-se que, com o aumento do tamanho da página, a taxa de $PageHits$ cai. Isso ocorre pois, com o aumento do tamanho da página, menos páginas poderão ser armazenadas na memória.

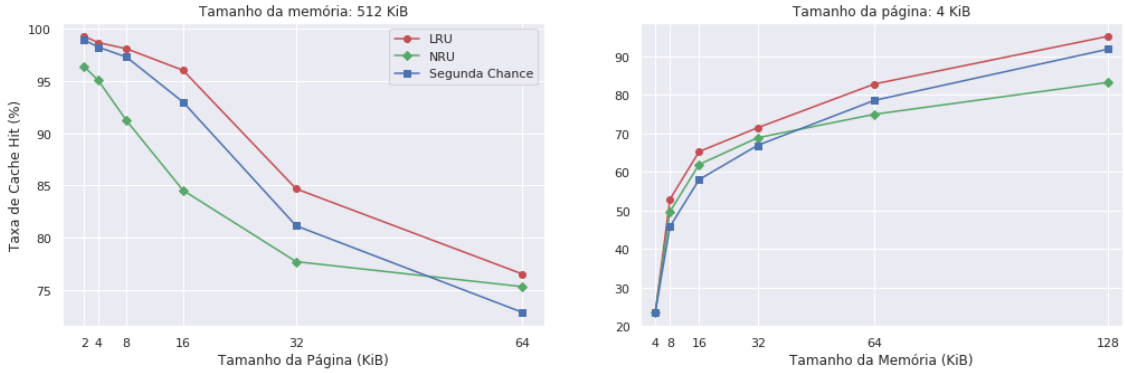
Dentre os algoritmos analisados, o que obteve a melhor taxa de $PageHits$, para todos os arquivos de entrada, foi o LRU. Observamos que conforme o tamanho da página aumenta, a política NRU tende a ter uma maior taxa de $PageHits$ em relação às demais políticas, com exceção, somente, à entrada "*compressor.log*", para a qual obteve o pior resultado para todos os tamanhos de páginas analisados.

A seguir, são apresentados os gráficos com os resultados obtidos:

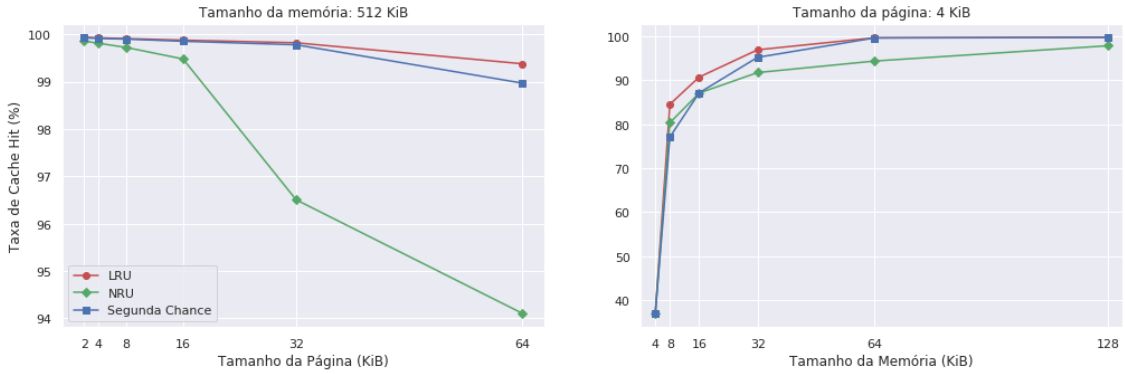
Simulação com a entrada: simulador.log



Simulação com a entrada: matriz.log



Simulação com a entrada: compressor.log



Simulação com a entrada: compilador.log