Projeto 1 de Organização de Computadores

Eduardo de O. Andrade[1], **Guilherme H. Apostolo**[1]

[1]Instituto de Computação (IC) – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, s/n – Boa Viagem – Niterói, RJ – Brasil

eandrade@ic.uff.br,guilherme_apostolo@id.uff.br

Descrição. *Construa um simulador de algoritmos de substituição de página de memória em cache. O simulador deve receber como entrada a sequência de referências às páginas de memória principal (endereços), e simular as substituições realizadas em cache após a ocorrência de um miss, para os algoritmos FIFO, LRU, LFU e RANDOM. O programa deve receber como parâmetro a capacidade total da memória cache (ou seja, número total de páginas), o esquema de mapeamento (direto, associativo e associativo por conjunto) que a cache vai operar, e o nome do arquivo de entrada a ser lido pelo programa, contendo as sequências de referências aos acessos de páginas da memória. O formato do arquivo de entrada consiste em um valor de endereço de memória (um número inteiro por linha) a ser carregado no programa. A saída do simulador deve consistir de, para cada política de substituição:*
a) *A cada nova referência de memória do arquivo de entrada, imprimir a lista de todas as páginas armazenadas na memória cache;*
b) *Ao final da execução, a fração de acertos às referências de memória para cada política.*

1. Introdução

As memórias cache tornaram-se dispositivos cada vez mais importantes nos sistemas computacionais. O princípio da localidade destas memórias tem como objetivo manter uma cópia dos dados e instruções utilizadas mais recentemente. Desta maneira, não é necessário buscar estes mesmos dados e instruções na memória principal. Entretanto, o tamanho da memória cache é inferior ao tamanho da memória principal, pois a cache ocupa parte da área do chip do processador, permitindo uma troca de informações mais rápida. Logo, diferentes esquemas de mapeamento e algoritmos de substituição de páginas de memória em cache buscam otimizar o tempo de espera da requisição de dados e instruções por memória.

Este relatório apresenta as principais características dos esquemas de mapeamento, assim como os principais algoritmos de substituição, desenvolvimento e funcionamento de um simulador de memória cache capaz de empregar essas diferentes políticas de substituição e esquemas de mapeamento. O código do simulador foi escrito na linguagem Python. O restante deste documento ficou organizado da seguinte maneira:
2) Mapeamento Direto
3) Mapeamento Associativo
4) Mapeamento Associativo por Conjunto
5) FIFO
6) LRU
7) LFU
8) RANDOM
9) Execução do Programa

2. Mapeamento Direto

O tipo de mapeamento direto da memória cache associa cada posição da memória principal, a qual possui um espaçamento maior, a cada outra posição da memória cache, a qual possui um espaçamento menor. Esta é a forma mais simples de mapeamento. Como há uma grande diferença no espaçamento entre as diferentes memórias, o mapeamento é necessário para que um grupo de palavras da memória principal seja mapeado na mesma posição da memória cache. Para que isto ocorra, são necessários alguns bits que possibilitam a referência aos blocos de endereçamento destas posições da memória. Neste trabalho utilizou-se a função de *mod* para a separação dos blocos, como ensinado em sala de aula.

3. Mapeamento Associativo

No caso do mapeamento associativo, não há uma relação entre a posição da memória principal e da memória cache, como no caso do mapeamento direto. É necessário vasculhar toda a memória cache para verificar se a posição da memória principal encontra-se na memória cache. Este tipo de mapeamento é melhor que o direto no sentido de não ter o risco de uma cache ociosa. Porém, a questão de ter que averiguar todas as posições da memória cache pode fazer com que o processo seja muito mais demorado.

4. Mapeamento Associativo por Conjunto

Há a ideia de conjuntos para mapear a memória principal neste tipo de mapeamento. Esta divisão da memória cache em conjuntos permite uma consulta mais rápida mas é diferente

do mapeamento direto. Neste caso, existe uma flexibilidade em relação a substituição de blocos na memória cache e o emprego dos algoritmos de substituição descritos, assim como no mapeamento associativo. Logo, o mapeamento associativo por conjunto acaba sendo um meio termo entre os outros dois mapeamentos. A sua desvantagem é o aumento de complexidade do circuito necessário para comparar as *tags*.

5. FIFO

No algoritmo *first-in-first-out* (FIFO), como o nome já diz, os dados e instruções que "entram" primeiro na memória, são os primeiros a "sair". Através de uma lista com todas as páginas atualmente na memória, a página mais antiga até a página mais recententemente referenciada, podem ser consultadas. Caso ocorra uma falta de página, a página mais antiga presente nesta lista é removida e uma nova página então, passa a ser a mais recente da lista.

6. LRU

No algoritmo *least recently used* (LRU), verifica-se qual das páginas em uma lista é a que foi referenciada por último. O LRU é um algoritmo bastante utilizado e que tem como ideia substituir esta página com referência mais antiga, pois provavelmente ela não está sendo referenciada mais frequentemente como outras páginas por um "bom tempo".

7. LFU

No algoritmo *least frequently used* (LFU), escolhe-se a página que foi menos acessada entre todas que estão carregadas na memória. Isto ocorre através de um contador de acessos para cada página. Toda vez que uma página deixa a lista de páginas carregadas na memória, o contador de acessos para aquela página é zerado. Uma solução para as páginas mais recentes (contador de acessos igual ou próximo de zero) que podem sofrer uma quantidade maior de substituições, consiste na criação de um tempo de carência que impossibilite estas permutações.

8. RANDOM

No algoritmo aleatório, não há critério de seleção para as páginas alocadas na memória cache, ou seja, todas possuem a mesma chance de serem acessadas. É um algoritmo de baixa eficiência e pouco utilizado no geral.

9. Execução do Programa

O link com todos os arquivos do projeto está disponível em:

`https://github.com/eduardoandrade/projeto_1_oc.git`

Para a execução do programa basta ter o Python 3.6 ou superior instalado em sua máquina, seja qual for o sistema operacional (SO) (Windows, macOS ou Linux). Após a instalação, para verificar se a versão instalada do Python está correta, basta abrir o terminal de comando do seu SO e digitar:

```
$ python --version
```

Caso a versão apresentada seja *python 2.X.X*, você pode precisar utilizar o seguinte comando para executar o programa (no mesmo diretório dos arquivos do projeto):

```
$ python3 main.py
```

Caso a versão apresentada seja *python 3.6.X* ou superior, basta o comando abaixo (no mesmo diretório dos arquivos do projeto):

```
$ python main.py
```

Para exibir o seu diretório atual no Windows:

```
$ cd
```

No macOS ou Linux:

```
$ pwd
```

Após a execução do comando *python 3 main.py* ou *python main.py*, o programa iniciará. Uma linha perguntando qual o tamanho da cache que será utilizado aparecerá no seu terminal. Para passar o tamanho da cache que deseja, basta apenas digitar um número inteiro e apertar a tecla *Enter*.

O próximo argumento a ser pedido pelo programa será o tipo de mapeamento da memória cache que será utilizado. Se deseja o mapeamento direto digite *DI*, se deseja o mapeamento associativo digite *AS* e se quiser o associativo por conjunto, digite *AC*. Após digitar as letras, aperte a tecla *Enter* mais uma vez.

O próximo parâmetro pedido pelo código será então, o arquivo de leitura contendo as entradas da memória a serem lidas. Toda entrada deve seguir o formato que está presente no arquivo de exemplo, com apenas números e um número por linha. Ao dizer o arquivo a ser lido, também é necessário informar o caminho completo onde o arquivo se encontra e também a extensão ao fim do nome (*.txt*, no exemplo). No Windows seria algo como:

```
C:/Users/usuario/Desktop/Siscomp/exemplo.txt
```

No macOS ou Linux não é muito diferente:

```
/Users/usuario/Desktop/Siscomp/exemplo.txt
```

Caso a política escolhida seja a associativo por conjunto, será necessário escolher o número de conjuntos a ser utilizado. Para isto, basta digitar um número inteiro maior que 2 e menor que o tamanho da cache, apertando a tecla *Enter* logo em seguida. Por fim, o programa exibirá uma linha pedindo para que seja informado qual a política de substituição deverá ser utilizada. Caso o usuário queira a politica FIFO, basta digitar *FIFO*, para a política RANDOM, digitar *RANDOM* e assim por diante. Para visualizar todas as políticas possíveis, digite *ALL*. Novamente aperte a tecla *Enter* para finalizar.

Todos os arquivos no repositório do GitHub deste projeto estão comentados e divididos de acordo com as políticas de substituição e mapeamentos para fácil compreensão dos códigos. Também há um arquivo *exemplo.txt* já disponível para a entrada. O programa executa de forma interativa com o usuário, esperando os parâmetros desejados e retornando as saídas de acordo com a descrição do projeto.