Pontifícia Universidade Católica do Paraná
Escola Politécnica
Bacharelado em Engenharia de Software

Disciplina: Performance em Sistemas Ciberfisicos
Professor: Guilherme Schnirmann

Nome Estudantes: Ary Farah, Ícaro Kuchanovicz, Vinicius Dorneles

Trabalho RA2

Representação didática do mapeamento direto e associativo em python

Entrega e equipe

Deverá ser entregue no AVA:

- Relatório com as evidências de funcionamento (edição deste arquivo) seguindo os exemplos do relatório
- Código do projeto
- Vídeo do grupo explicando o que cada um fez (divisão das tarefas) e mostrando o funcionamento com alguns exemplos (3 minutos)
- O trabalho deverá ser feito em equipes de 2 ou 3 integrantes.

Descrição da Atividade:

Estudamos que a memória cache é muito menor do que a memória principal, em contrapartida ela é muito mais rápida e próxima ao processador. Dessa forma, com interesse em performance com acessos de forma otimizada, temos políticas de mapeamento e substituição de quais endereços ficam na cache, visto que sempre há disputa por endereços nessa memória. Nessa prática, não vamos pensar em implementação da memória principal (apenas usá-la), ou seja, nos preocuparemos com o endereçamento na cache – testando endereços na cache a princípio vazia e somente fazendo as comparações ditadas pelo método estudado. Ainda, não nos preocuparemos com os endereços em bits ou bytes, mas sim, com fins didáticos, utilizaremos referências de endereços inteiros.

Importante: vamos trabalhar com linhas da cache, não importando o tamanho do bloco. Só, queremos entender de forma didática como os mapeamento direto e associativo funcionam.

Mapeamento direto

Nesse mapeamento, temos a associação de cada endereço da memória principal em um correspondente específico na cache. Para fazermos essa associação vamos

utilizar a operação **"mod"** ou **%** que retorna o resto de uma divisão – entre a posição desejada e o número de linhas. Por exemplo:

O processador pede a posição 23 e o tamanho da cache são 10 linhas:

23 *mod* 10 = 3 (O endereço 23 da memória principal está mapeado no 3 da cache).

Prova: 2*10 + 3 , ou seja, cabem 2 inteiros de 10 no número 23 e sobram 3

Note que se o primeiro operando for menor que o segundo, o *mod* retorna o próprio operando:

7 mod 10 = 7 (O endereço 7 da memória principal está mapeado no 7 da cache).

Prova: 7 = 7*0 + 7 (na divisão não temos nenhum inteiro, mas só temos resto, ou seja, o resultado é 7).

Objetivo:

Queremos analisar o passo-a-passo de acesso no mapeamento direto (assim como foi visto em aula feito a mão). Ou seja, enxergar qual posição está sendo disputada nesse mapeamento e como é realizada a troca. Ainda, queremos analisar a quantidade de hits (acertos) e misses (falhas) de acordo com as posições solicitadas pelo processador.

Entrega:

O estudante deverá entregar um arquivo ".pdf" contendo as respostas da atividade proposta no roteiro.

Roteiro da Atividade:

A implementação sugerida aqui é um roteiro/ajuda, caso prefira, pode-se implementar de outras formas ou até mesmo linguagens, desde que o resultado esperado seja mostrado.

1. Crie um arquivo .py
2. Inicialmente vamos criar uma função para inicialização da nossa cache:

```
def inicializar_cache(tamanho_cache):
```

   a. Nessa função inicialize um dicionário (chave, valor) vazio
   b. O dicionário deve ser populado com os índices dos endereços inteiros (0 até tamanho da cache) e com os valores iniciais de -1 (para representar que o endereços ainda não foi utilizado):
   c. retorne a memória cache inicial
   d. A cache inicial deverá ser assim (exemplo com tamanho 10):

Posição cache	Posição Memória
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	-1

3. Agora crie uma função que imprima o estado atual da cache, utilize como base a cache inicial (exemplo acima). Deve-se criar uma estrutura simples para impressão de todas as linhas da cache

```
def imprimir_cache(cache):
```

   a. Imprima o tamanho da cache

   b. Imprima a tabela. Dica, para impressão em dicionário você deve pegar a chave e valor. Nessa ordem são retornados chamando: *nome_dicionario.items()*

4. Crie a função que faz o mapeamento direto.

```
def mapeamento_direto(tamanho_cache, pos_memoria):
```

   a. O parâmetro pos_memoria é uma lista com as posições de memória que pretendemos acessar.

   b. Inicialize a cache – utilize o método criado no item 2.

   c. Imprima a situação inicial da memória cache

   d. Agora precisamos iterar em cada posição de memória que queremos acessar e verificar na cache (no mapeamento associado) se encontramos ou não. Agora, atualizamos a cache com o endereço em questão.

Lembre-se, para calcular a posição do mapeamento da cache deve-se fazer:

posicao_cache = posicao_memoria % tamanho_cache

Agora que temos nossa memória cache inicializada e a posição na cache que queremos checar, devemos comparar a nossa memória cache na posição calculada com a posição de memória que estamos buscando. Se for igual, temos um hit (incremente uma variável de número de hits), se não, temos um miss (incremente uma variável com o número de misses).

Atualize a memória cache na respectiva posição com o valor da memória.

   e. Em cada iteração imprima se tivemos um hit ou um miss e imprima a memória cache atualizada.

   f. Por fim, imprima um resumo com:

- Total de posições de memórias acessadas

- Total de hits:

- Total de misses:

- Taxa de cache hit:

Resultado esperado (exemplo):

```
posicoes_memoria_acessar = [33,3,11,5]
tamanho_cache = 5
mapeamento_direto(tamanho_cache,posicoes_memoria_acessar)
```

```
Cache inicial
Tamanho da Cache:           5
Pos Cache |Posição Memória
        0|             -1
        1|             -1
        2|             -1
        3|             -1
        4|             -1
```

```
 Linha 0 | posição de memória desejada 33
 Status: Miss
 Tamanho da Cache:          5
 Pos Cache |Posição Memória
         0|             -1
         1|             -1
         2|             -1
         3|             33
         4|             -1

 Linha 1 | posição de memória desejada 3
 Status: Miss
 Tamanho da Cache:          5
 Pos Cache |Posição Memória
         0|             -1
         1|             -1
         2|             -1
         3|              3
         4|             -1

 Linha 2 | posição de memória desejada 11
 Status: Miss
 Tamanho da Cache:          5
 Pos Cache |Posição Memória
         0|             -1
         1|             11
         2|             -1
         3|              3
         4|             -1

 Linha 3 | posição de memória desejada 5
 Status: Miss
 Tamanho da Cache:          5
 Pos Cache |Posição Memória
         0|              5
         1|             11
         2|             -1
         3|              3
         4|             -1
```

```
Conectividade em Sistemas Ciberfisicos - Mapeamento Direto - Guilherme
*---------------------------------------------------------------------*
Memórias acessadas: 4
Número de hits: 0
Número de misses: 4
Taxa de acertos (hits): 0.00%
```

5. A seguir represente esses exemplos para testar o seu programa, no seu vídeo você deverá apresentar esses exemplos e outros para mostrar o funcionamento do programa:

   a. **Execute com os seguintes parâmetros:**

```
posicoes_memoria_acessar = [0,1,2,3,1,4,5,6]
tamanho_cache = 5
```

Qual a taxa de acerto?

R: 12.50%

Quantas e quais trocas na cache ocorreram? Explique.

R:

```
----------------------------
Resumo Mapeamento Direto - Ary | Vini | Ícaro
----------------------------
Total de acessos: 8
Misses: 7
Hits: 1
Taxa de acertos (hits): 12.50%
----------------------------
```

```
posicoes_memoria_acessar =
[0,1,2,2,22,32,42,20,1,10,11,12,13]
tamanho_cache = 5
```

Qual a taxa de acerto?

R: 15.38%

Quantas e quais trocas na cache ocorreram? Explique.

R:

```
----------------------------
Resumo Mapeamento Direto - Ary | Vini | Ícaro
----------------------------
Total de acessos: 13
Misses: 11
Hits: 2
Taxa de acertos (hits): 15.38%
----------------------------
```

b. Execute com os seguintes parâmetros:

```
posicoes_memoria_acessar = [1,6,1,11,1,16,1,21,1,26]
tamanho_cache = 5
```

Qual a taxa de acerto?

R: 0%

Quantos endereços de memória da cache foram utilizados? Explique.

R: Apenas 1 endereço, pois o resto de todas as divisões da posição da memória pelo resto da cache dão resto 1

```
----------------------------
Resumo Mapeamento Direto - Ary | Vini | Ícaro
----------------------------
Total de acessos: 10
Misses: 10
Hits: 0
Taxa de acertos (hits): 0.00%
----------------------------
```

Explique a taxa de acerto encontrada. O endereço 1 sendo solicitado diversas vezes não deveria trazer uma taxa de acerto maior? Explique.

R: Não, pois todas as vezes que o endereço 1 foi chamado ele já tinha sido ocupado por outra posição anteriormente, já que todas as posições brigam pela mesma linha da memória cache (posição_memória % tamanho_cache)

Faça uma configuração de acesso em memória, com endereços diferentes, que pelo mapeamento direto sempre a mesma posição na cache seja utilizada. Há alguma vantagem nisso? E desvantagem? Imprima a sua configuração e explique.

R: Desvantagem -> enquanto uma linha da cache está sendo muito utilizada, muitas outras não estão, resultando em mais acessos à memória principal (misses)

```
mapeamento_direto(cache, 4, [17, 13, 17, 25, 33, 9])
```

```
----------------------------
Resumo Mapeamento Direto - Ary | Vini | Ícaro
----------------------------
Total de acessos: 6
Misses: 6
Hits: 0
Taxa de acertos (hits): 0.00%
----------------------------
```

6. Agora faça o mesmo para o mapeamento associativo por conjunto. O usuário poderá escolher o tamanho do conjunto (1 bloco – totalmente associativo ; 2 blocos ; 4 blocos ; 8 blocos ; 16 blocos). A técnica de substituição será a LRU.
    a. Crie exemplos para mostrar o funcionamento do seu programa (faça pelo menos com 4 e 8 blocos)

R: 4 Blocos

```
tam_cache = 16
num_blocos = 4
pos_memoria = [0, 1, 2, 2, 5, 6]
----------------------------------------
Resumo Mapeamento Associativo Por Conjunto - Ary | Vini | Ícaro
----------------------------------------
Total de acessos: 7
Misses: 6
Hits: 1
Taxa de acertos (hits): 14.29%
----------------------------------------
```

R: 8 Blocos

```
tam_cache = 16
num_blocos = 8
pos_memoria = [1, 0, 5, 4, 7, 2, 3, 3]
----------------------------------------
Resumo Mapeamento Associativo Por Conjunto - Ary | Vini | Ícaro
----------------------------------------
Total de acessos: 8
Misses: 7
Hits: 1
Taxa de acertos (hits): 12.50%
----------------------------------------
```

R: 2 Blocos

```
tam_cache = 16
num_blocos = 2
pos_memoria = [4, 2, 5, 6, 7, 2, 1, 1, 2]
----------------------------------------
Resumo Mapeamento Associativo Por Conjunto - Ary | Vini | Ícaro
----------------------------------------
Total de acessos: 9
Misses: 6
Hits: 3
Taxa de acertos (hits): 33.33%
----------------------------------------
```

b. Crie exemplos para mostrar a diferença entre mapeamento direto e mapeamento associativo por conjunto.

Enquanto no mapeamento direto as posições brigam pela mesma linha na cache, causando mais misses, no associativo por conjunto elas sao armazenadas em linhas diferentes, resultando em hits ao reacessar alguma posição (como acontece no exemplo abaixo).

MAPEAMENTO DIRETO | **ASSCOCIATIVO POR CONJUNTO**

```
tamanho_cache = 16
pos_memoria = [0, 4, 8, 12, 16, 20, 24, 28, 4, 8, 12]

cache = inicializar_cache(tamanho_cache)
title("Cache Inicial")
imprimir_cache(cache)
mapeamento_direto(cache, tamanho_cache, pos_memoria)
```

```
tam_cache = 16
num_blocos = 4
pos_memoria = [0, 4, 8, 12, 16, 20, 24, 28, 4, 8, 12]

cache = inicializar_cache(tam_cache, num_blocos)
title("Cache Inicial")
imprimir_cache(cache)
mapeamento_associativo_conjunto(cache, tam_cache, num_blocos, pos_memoria)
```

Pos Cache	Pos Memória
0	16
1	-1
2	-1
3	-1
4	4
5	-1
6	-1
7	-1
8	8
9	-1
10	-1
11	-1
12	12
13	-1
14	-1
15	-1

Conjunto	Bloco
0	[0, 16, -1, -1]
1	[4, 20, -1, -1]
2	[8, 24, -1, -1]
3	[12, 28, -1, -1]

```
Resumo Mapeamento Direto - Ary | Vini | Ícaro
Total de acessos: 11
Misses: 11
Hits: 0
Taxa de acertos (hits): 0.00%
```

```
Resumo Mapeamento Associativo Por Conjunto - Ary | Vini | Ícaro
Total de acessos: 11
Misses: 8
Hits: 3
Taxa de acertos (hits): 27.27%
```