Atividade 5 - Org. Arg. 1

Profa. Dra. Cíntia Borges Margi
(cintia@usp.br)

Guilherme S. Salustiano
(salustiano@usp.br)

Contexto

Cache é um conceito universal da computação usado em diversas contextos para melhorar o desempenho de programas. Nessa atividade iremos implementar e analisar o cache para um banco de dados chave valor inspirado no cache estudado do RISC-V.

O banco de dados é o mais simples possível com apenas duas funções:

```
db_data_t get(db_key_t key);
void put(db_key_t key, db_data_t data);
```

O `put` deve salvar o valor `data` em um arquivo com o nome `key`, além de colocar esse valor na cache. O `get` procura o dado na cache, a partir de sua `key`, caso não encontre, lê o dado do arquivo com o nome `key` e armazena ele na cache, além de retornar o dado.

A chave é um número inteiro sem sinal de 64 bits, enquanto o dado é um struct de 32 bytes definidos em `database_t.h` - composto por 4 interios sem sinal de 64 bits.

Tarefa

Você deve implementar um cache associativo com o número de conjuntos configurável pela variável `ASSOCIATIVITY` no arquivo `database.h`. Você deve usar a variável `static cache_entry_t cache[CACHE_SIZE];` para armazenar o cache, e as variáveis estáticas `cache_hits` e `cache_misses` para contar quantos acertos/falhas houveram na cache (ambos são inteiros de 32 bits sem sinal). Além dessas variáveis, não é permitido usar outras variáveis globais.

Cada linha da cache é um struct `cache_entry_t`, definido em `database.h`, composto pelos seguintes campos: valid, last_access, tag e data. O campo valid indica se o dado daquela posição é válido. O campo last_access representa o instante de tempo em que houve o último acesso a essa posição da cache, ele será utilizado para a lógica substituição dos blocos (LRU). Recomenda-se atualizar o valor dessa variável com a função clock(). O campo tag contêm a tag da determinada linha da cache, por simplicidade serão utilizados apenas os 32 bits menos significativos da key. O campo data tem o dado que é armazenado nessa linha da cache.

Além disso, na biblioteca `database_file.h` são definidos dois métodos `file_get` e `file_put`. O `file_get` obtêm o valor data de um arquivo com o nome key. O `file_put` salva o valor data em um arquivo com o nome key.

Resumindo, temos o seguinte:

- A função `put` deve analisar os campos valid, last_access e tag para escrever um dado no banco,

respeitando os conjuntos definidos pela variável ASSOCIATIVITY. Além disso, deve-se escrever no arquivo utilizando a função `file_put` (write-through).

- A função `get` deve analisar os campos valid, last_access e tag para ler um dado no banco,

respeitando os conjuntos definidos pela variável ASSOCIATIVITY. Caso o dado não esteja no banco, deve-se ler esse dado do arquivo utilizando a função `file_get` e colocá-lo no banco. Sendo que, nesse exercício pressupõe-se que o dado sempre existe no respectivo arquivo. Em caso de acerto/falha, deve-se incrementar as variáveis `cache_hits` e `cache_misses`.

Em caso de substituição de um bloco da cache (em ambas as funções), use a estratégia LRU com base no campo last_access.

Você pode baixar o código base no repositório do experimento, disponivel da pasta `src`. Um makefile também foi fornecido para facilitar a compilação e execução do programa, para executar basta rodar `make run`.

Para rodar os testes com diferentes configurações você pode executar seguinte comando:

```
make -s run_all
```

Atenção: isso altera o valor de `ASSOCIATIVITY` no arquivo `database.h`! Para voltar ao valor padrão você pode rodar `make reset`.

Você pode encontrar testes que mostram o comportamento esperado do cache no arquivo `main.c`.

Entrega final

Ao final, gere um zip `atv5.zip` com os arquivos.

```
atvr5.zip
├── database.c
```