Assignment

Bang Ngoc Bao Tam

August 2022

1 Student Outcomes

After completing this assignment, students will be able to

- explain how a cache works.
- apply a BST tree to search in a simple cache.

2 Instructions

In a real computer machine, the speed of the main memory is very low in comparison with the speed of its processor. To improve the performance, a fast cache memory is used to reduce the processor's time waiting when accessing instructions and data in main memory. The cache stores copies of the data from the main memory so that future requests for that data can be served faster.

3 How it works

When the processor first references a memory slot in the main memory, the content of the memory slot together with its address are put into the cache. Subsequently, when the processor references the memory slot again, the desired contents are read directly from the fast cache instead of the slow main memory.

4 Cache replacement Policies

As the size of a cache is limited, when the cache is full and a new data is inserted into the cache for future requests, an existing data in the cache must be selected by the cache replacement policy and it is removed to have place for the new data. There are many cache replacement policies, but in this assignment, you will implement one of them. Especially:

1. FIFO (First In First Out): the oldest data is selected
2. LIFO (Last In First Out): the newest data is selected

5 Assignment Requirements

5.1 Descriptions

This assignment requires to implement a cache memory for data accessing (read/write).

5.1.1 Cache interface

The assignment must be implemented as a class Cache with the following methods:

1. Data* read(int addr): returns the data stored at the address addr if the addr is kept in the cache, otherwise, return NULL. The type Data is described later.

2. Elem* put(int addr, Data* data): puts addr and data into the cache and returns the element which is removed out of the cache if any, or NULL otherwise.

3. Elem* write(int addr, Data* cont): searches if the addr is in the cache. if it is, this method replaces the existing value associated with the addr with the cont. If it is not, this method puts the addr and cont into the cache. This method returns the element removed out of the cache if any, otherwise, returns NULL. Note that the cont together with its address addr are put into the cache, not into the main memory, so the data in the cache now is not the copy of the data in the memory anymore. Therefore, the field sync of the element in the cache must be set to false.

4. void print(): prints the value of the elements in the cache in the descending order of the element's living time. Each element must be printed by the method print of class Elem.

5. void preOrder(): prints the value of the elements in the cache in the preorder of the BST tree which must be used to search an address in the cache.

6. void inOrder(): prints the value of the elements in the cache in the inorder of the BST tree

When a new element is put into the full cache (in method put and write), if the address is at the even position, the FIFO replacement policy is applied to remove an existing data in the cache. On the opposite, the LIFO replacement policy is applied. Note that the size of the cache is given in the definition of MAXSIZE in file "main.h". To perform a search in the cache, an BST tree with the address as a key must be implemented.

5.1.2 Explanation of methods

When the processor would like to read data at address addr, it will call method read(addr) of class Cache. If the address addr together with its content are in the cache, the content is returned. Otherwise, NULL is returned and the processor will read data cont from main memory and call put(addr,cont) to put item (addr,cont) into the cache. When the processor would like to write data cont into address addr, it will call method write(addr,cont). This method will modify the content of the address addr if any or put a new item into the cache whose sync is false. When the method put or write returns an item as the cache is full, the processor will check the sync of the item. If it is false, the processor will write the data of the item into the main memory.

The other methods of class Cache are just used to check your implementation.

5.1.3 Data Input and Output

The following classes, which are defined in file "main.h", are used as data input and output:

1. class Float: used for a float data
2. class Int: used for an integer data
3. class Bool: used for a boolean data
4. class Address: used for an address content
5. class Data: used to represent any data
6. class Elem: used to represent an element in the cache which includes 3 main fields:
   (a) addr: keeps the address of the memory slot
   (b) data: keeps the content of the memory slot
   (c) sync: shows whether the content in the cache is the same as the content of the corresponding memory slot in the main memory.

5.2 Memory leak

When a programmer forgets to clear a memory allocated in heap memory, the memory leak occurs. It's a type of resource leak or wastage. In this assignment, you have to handle this issue so that your program will not cause memory leak.

5.3 Instructions

To complete this assignment, students must

- Read the specification of the attached virtual machine

- Download the initial code, unzip it
- There are 4 files: main.cpp, main.h, Cache.cpp and Cache.h. You MUST NOT modify files main.cpp and main.h.
- Modify files Cache.h and Cahe.cpp to implement the cache memory.
- Make sure that there is only one include directive in file Cache.h that is ♯include "main.h" and also one include in file Cache.cpp that is ♯include "Cache.h". No more include directive is allowed in these files.

6 Submission

Files cache.h and cache.cpp are required to submit as attachments before the deadline is given in the link "Assignment Submission".