Project 3 Report

15146301 Kang shin hyeok

**AddressTranslator.java**

First, read inputfile through scanner from addresstranslator.java.

Save the remainder of the sequence within the Inputfile divided by 65536 back into addr and cut to numbers up to 65536. Then, put the addr value divided by 256 (addr address of page number divided by 256) in p\_num and store the remaining value when divided by 256 to offset(indicating the position within the divided p\_num).

Then, reset the f\_num value where frame number is stored to -1. Then, check f\_num with tlb.get(p\_num) to see if the frame number corresponding to the corresponding page number is present in TLB first. If the value does not change from f\_num=-1, then increase tlb\_Miss by 1 and check whether the page table has frame number present through f\_num=pt.get(p\_num) because the value does not exist in TLB. If the page table does not have a value too, increase the page\_fault value by 1 and read from the backup store. Store the data obtained through bs.getData(p\_num) in Frame f. Store f in the location of f\_num,with use the pm.addFrame(f), which is the physical address. Then use pt.add(p\_num, f\_num); and tlb.Put(p\_num, f\_num); to save it in the page table and TLB respectively.

When all storage is completed, the actual physical memory address is saved to phy\_addr through f\_num\*256 + offset. In addition, corresponding value is printed through pm.getValue(f\_num, offset).

**TLB.java**

This class was created to reproduce the Translation Look-aside Buffer on a computer. This class stores only 16 pairs of page numbers and frame numbers, just like the actual TLB. In addition, print frame number matching page number according to external request or vice versa, delete the first inserted data out of the original 16 pairs of data and enter a new pair of data in place. Let's look at the methods one by one.

TLB()

First, create a hashtable and linkedlist, and then add value -1 to the table in all keys from 0 to 15. Insert 0 to -15 in the list. The reason why initial values are not positive is that there may be values from 0 to 15 in the actual page number.

get(int p\_num){

This method is used when you want to print the value of the corresponding p\_num in the TLB. Use 'If statement' to check if p\_num is present in the table. If present, use this.table.get(p\_num) to return the f\_num value. If it does not exist, return -1 to find the value in the page table.

put(int p\_num, int f\_num){

This method is used when you want to add the desired value to the TLB. Here we have to do two things. 1. Delete oldest data 2. Insert new data into place. The oldest data is first deleted and inserted there, with the oldest selected using a queue. Use this.list.poll() to select the oldest data and store it in Integrator I. Then, if this data is not a null value, delete it from the table. And put new data into the list and table respectively.

Main method

In the main method, we can see that TLB maintains only 16 datasets strictly. A total of 17 datasets were inserted in tlb from 0 to 16 through the For statement. When the last "16" is inserted here, because there are already 16 data in tlb, '0' is deleted via 'poll' from the list and new data '16' is inserted. Therefore, if you attempt to output a value of key ‘0’ with System.out.println(tlb.get(0)), -1 is returned because it does not exist.

**PageTable.java**

This class is designed to reproduce PageTable on a computer. Print frame number matching the page number according to external request or insert the page number and frame number pair. Let's look at the methods one by one.

PageTable(){

As the constructor, first make an array with 256 spaces in the PageTableItem format in table and enter a value (-1,false) in each.

get(int p\_num){

This method is used when you want to find out if a value with corresponding p\_num exists in a PageTable. Use getFrameNumber() method to return frame number corresponding to p\_num. If frameNumber =-1, the frame number does not exist in the page table, so return -1 to allow the AddressTranslator to access the BackStore.

add(int p\_num, int f\_num){

This method is used when you want to insert the desired value into the PageTable. Use this.table[p\_num] to specify the location of the p\_num. Then, use PageTableItem(f\_num, true) to enter f\_num and boolean values.

PageTableItem (int i, boolean b){ }

A method created to store the value of PageTable.

getFrameNumber(){

The method that returns frameNumber in the table when called.

**PhysicalMemory.java**

This class was created to recreate the computer's PhysicalMemory.

PhysicalMemory()

Constructor. The frames of the datasets in the PhysicalMemory are arrayed with 256 spaces, and the currentFreeFrame is reset to zero.

addFrame (Frame f)

This method is used when you want to insert a new frame into the memory. In the default state, frames[] should be inserted from position 0. Then, use this.currentFreeFrame++ to increase the insertion position one by one.

getValue(int f\_num, int offset)

The method that outputs the value that exists in the frame and offset entered. Declares frame corresponding to f\_num entered first. Then, the data in the location corresponding to the offset of the frame is returned.

**BackStore.java**

This class was created to act as a storage device, such as a hard disk, on a computer. Therefore, getData() method for reading data from disk is implemented.

getData (int pageNum)

This method is used to read and output data from disk. First, create the RandomAccessFile "BACKING\_STORE" to refer to as disk. Then place the pointer in the frame corresponding to the pageNum entered using disk.seek (pageNum\*256). Then, extract 256 data from the pointer through disk.read(value) and store it in value[]. Then, copy this value to result[] and return the result.

Main method

The main method shows what data is in the backstore. Using System.arraycopy() copy and paste the data in getData(100) from 0 to 32 locations in the i array. From this, the value in frame with page number 100 is printed from 0 to 32nd. Repeat the same process when the page number is 255.

With original implementation, TLB miss: 945, Page Fault: 244

**-Modifications**

1. FIFO algorithm (Lab8)

First of all, change the rules that determine the physical memory frame. Set to Physical memory size = 128. If all 128 spaces are filled, repeat the currentFreeFrame=0 and rotate it again.

Next is the modifying pagetable.java. First, declare linkedlist and save p\_num value whenever you use add method. The first p\_num inserted into the list is extracted using 'poll()' from the time the count is over 128. Then replace the 'valid' with 'false' and save the new p\_num.

With this algorithm, TLB miss: 947, Page Fault: 538

2. LRU algorithm (Lab9)

In case of LRU algorithm, the p\_num referenced through the get method was sent to the back of the list to list in the most unreferenced order, and among them, the old order was deleted. Therefore, the list node corresponding to p\_num referenced in the get method was deleted and 'list.add(p\_num)' was used again. PhysicalMemory also generates a list and operates it in the same way because it has to select the frame to be replaced in accordance with LRU algorithm. However, because the frame must be sent to the rear of the list only when a reference occurs in pagetable, the code was placed in the changeOrder() method and executed only if f\_num=pt.get(p\_num) in the addresstranslator was successful.

With this algorithm, TLB miss: 941, Page Fault: 540