

Programming Assignment#4

< Designing a Virtual Memory Manager >

AddressTranslator.java file

In this file's main method, initially declared some variables. inputFile String is declared for calling under the file object's parameter. And addr, p_num, offset, f_num, value, phy_addr, tlb_miss, page_fault integer type variables are declared for expressing page replacement implementation. For the setting, this program create 4 objects such as tlb, pt, pm and bs to use implementation.

Because in this program, virtual memory space is represented as 16bit, in the while, addr is modulated as 2^{16} . Page schema's pre-8bit represent page num, so address is divided as 2^8 . And post-8bit represent page offset, so address is modulated ad 2^8 . Initially, frame num is initiated as -1.(Not allocated) program tried to get frame number that is matched to the parameter(in here this parameter is page number.)in TLB object. If fail to get appropriate frame number in tlb, increase tlb_miss 1. Next, we try to get framenum associated to the pagenum in the pageTable object. If fail to get again, increase page_fault 1. Finally program try to get frame from backing store. And add frame to the physical memory object and get frame number associated to the page number. And add the page number and frame number into page table and tlb.

For end this filtering job, physical address is determined as $\text{framenum} * 2^8 + \text{offset}$.

End of the while, we can check how many tlb miss and page fault are occurred.

BackStore.java file

There is a getData method who require the pageNum parameter. With this parameter, program open the backingstore file as filename readonly condition and try to seek(in here, moving pointer in the disk) $\text{page number} * 2^8$ (because this is address when we set in the addressTranslator.) `Disk.read(value)` means read disk data as maximum size of value.length and store date in the value array. Under the for, store the value array's value to the result array. This array is a data

what we want.

PageTable.java file

In this file, there are inner class as PageTableItem(I will call this as pti.). pti has framenumbers and valid bit(as Boolean here.) Pti constructor requires framenumbers and valid bit state. And pti has getFrameNumber method. Go back to the PageTable, pti type array is declared as global variable. In the PageTable(I will call this as pt) constructor, pt is initialized 256 ptis. And allocate to table's each items as frame number is -1, and valid bit is false(Because, constructor is just set initial mode). And there are get and add methods. Get method requires the wanted page number as parameter. If matched frame number is existed, return it. Else, return -1. Add method requires the page number and frame number as parameter. Table's page number's item is set as frame number and valid bit as true.

PhysicalMemory.java file

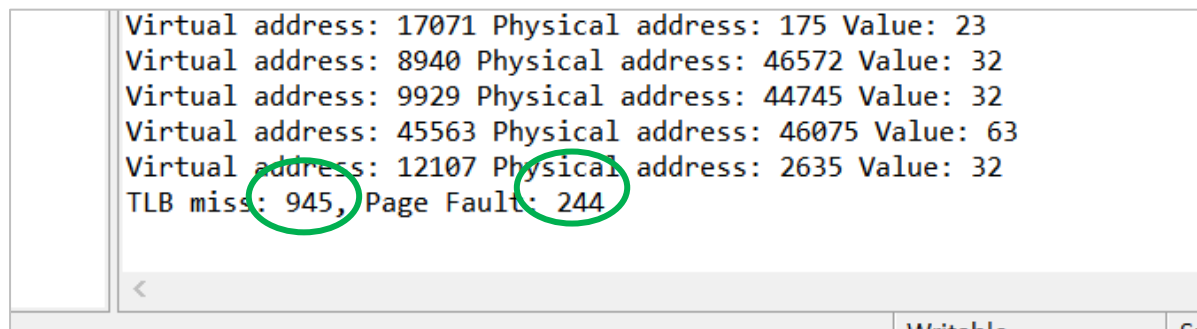
In this file, there are inner class as Frame. This inner class has int type array as data naming. There are two constructors exist. First one is initialization data array's size as 256 and set data as d array that is passed by parameter. Second one is copy the frame if frame is passed by parameter. Go back to the physicalmemory(I will call this as pm) file, there are frame type array and int type current free frame. In pm's constructor, frame size is set as 256 and current free frame is set to 0. There are two methods in pm. addFrame sets this frame's current free frame order's value as new passed frame as parameter. And increase current free frame 1 then return original one. getValue gets frame as frame number that is passed as parameter. And return frame's data as offset that is also passed as parameter.

TLB.java file

In this file, there are two global members Hashtable and LinkedList type. Tlb's constructor initiate Hashtable and put key and values in the table and add key in the list. Someone call get method with page number, TLB check this table has key as pagenum. If exist, return this table's values that is associated to the key. Else, return -1. In the put method, someone pass the pagenum and framenum as parameter. For flash, poll data in the list, if it is not null, remove it from table. And then add page number to the list and put page num and frame num into the table.

-----Step 1~8-----

Original result:



```
Virtual address: 17071 Physical address: 175 Value: 23
Virtual address: 8940 Physical address: 46572 Value: 32
Virtual address: 9929 Physical address: 44745 Value: 32
Virtual address: 45563 Physical address: 46075 Value: 63
Virtual address: 12107 Physical address: 2635 Value: 32
TLB miss: 945, Page Fault: 244
```

Because the frame size is 256, 244 times page fault is occurred.

Continuously update the frame.

FIFO Case:

```
Virtual address: 31260 Physical address: 32540 Value: 0
Virtual address: 17071 Physical address: 32687 Value: -85
Virtual address: 8940 Physical address: 32748 Value: 0
Virtual address: 9929 Physical address: 32713 Value: 0
Virtual address: 45563 Physical address: 32763 Value: -66
Virtual address: 12107 Physical address: 32587 Value: -110
TLB miss: 950, Page Fault: 363, Page Replacement: 235
```

For check the result, page fault is 363 and page replacement is 235. Let's see my algorithm.

```
1 import java.util.ArrayList;
2
3
4 public class StateController {
5
6     public static boolean isfull = false;
7     public static LinkedList<Integer> targetlist = new LinkedList<>();
8
9     public StateController() {
10
11     }
12 }
13
```

Before using static approach to the list, I make StateController class.

```

*/
public int addFrame(Frame f){
    if(this.currentFreeFrame > 127) {
        StateController.isfull =true;

        for(int i = 0 ; i < 127; i++) {
            this.frames[i] = this.frames[i+1];
        }
        this.frames[this.currentFreeFrame-1]= new Frame(f.data);
        StateController.targetlist.add(this.currentFreeFrame);
        replacement++;
        return this.currentFreeFrame-1;
    }
    StateController.isfull = false;
    this.frames[this.currentFreeFrame] = new Frame(f.data);
    StateController.targetlist.add(this.currentFreeFrame);
    this.currentFreeFrame++;
    return this.currentFreeFrame-1;
}
}

```

In physical memory's addFrame method, if this frame is full, set state(isfull) true. And remove frames[0](first in one). Then input the frame the last part (frames[127]). For check victim, add the frame number to the target list in StateController. In this case, page replacement is increased and return its index.

If not full, just add frame in the current free frame index and in the target list.

So, we can control the victim as stack.

```

*/
public void add(int p_num, int f_num){
    if(StateController.isfull) {
        int targetNumber= StateController.targetlist.poll();
        for(int i = 0; i < this.table.length; i++) {
            if(this.table[i].frameNumber == targetNumber) {
                this.table[i].valid = false;
                this.table[i].frameNumber = -1;
                break;
            }
        }
        this.table[p_num] = new PageTableItem(f_num, true);
    }else {
        this.table[p_num] = new PageTableItem(f_num, true);
    }
}
}
}

```

And we have to update page table also. Poll the victim frame number in the target list and find it in the table. If it is in the table, set valid false and frame number as -1. And then break.

And finally add the new page table item at page_num location.

LRU Case:

```

Virtual address: 31260 Physical address: 32540 Value: 0
Virtual address: 17071 Physical address: 32687 Value: -85
Virtual address: 8940 Physical address: 32748 Value: 0
Virtual address: 9929 Physical address: 32713 Value: 0
Virtual address: 45563 Physical address: 32763 Value: -66
Virtual address: 12107 Physical address: 32587 Value: -46
TLB miss: 946, Page Fault: 349, Page Replacement: 221

```

For check the result, page fault is 349 and page replacement is 221. Let's see my algorithm.

Basic logic is same. Select victim is same because we find stack's bottom(list's

head). So add algorithm is same. But we have to focus on get method. If we reference the page, this page is floating to the top of stack(tail of list). As below picture.

```
*/
public int get(int p_num){
    int frameNumber = this.table[p_num].getFrameNumber();

    if(frameNumber == -1){
        // frame number not in page table
        return -1;
    }else {

        int targetInd = StateController.targetlist.indexOf(frameNumber);
        StateController.targetlist.remove(targetInd);
        StateController.targetlist.add(frameNumber);
    }
    return frameNumber;
}
```

First, we find the index of frame number and remove it. And add again to the list. As follow these steps, we can update the stack(list).

Compare with FIFO and LRU:

LRU is more efficient than FIFO because LRU's page fault number is lower than FIFO.

Compare with original one:

Original's frame number is 256, and page fault is occurred 244 times.

For ideally, we down size to 128, page fault will be increased to the 488 times.

LRU case's page fault = 349, FIFO case's page fault = 363. So my modify version is more efficiency.