**Lab assignment (Memory)**

15146317 Seungjun Lee

**Variable and method analysis (Original file)**

**Address Translator**

String **inputFile** put context of context of InputFile in inputFile as string type

**addr** is logical address which is smaller than 65536. Because logical address is 65536byte

**p\_num** is page number which means that divide logical memory into block of 256 size

**f\_num** is frame number which means that divide physical memory into block of 256 size

**offset** is page offset which indicate specific unit in the frame that page number indicate frame

**value** is some value which is stored in address. Offset is smaller than 256byte because frame size is 256 byte.

**phy\_addr** is physical address which is calculated by translator

**tlb\_miss** is count that number of failed match TLB and page number

**page\_fault** is count that number of failed match page table and page number simply, how many times page table didn’t have page number when this page number check page table.

**TLB (tlb)**

**List** is linked list that each component represent sequence of input table initial value is all minus value total size is 16

**Table** is hashtable that each row represent page number and frame number. Page and frame number Initial value is minus total size is 16

**tlb.get** if table has key of page number, return frame number which is connected with page number. Or not return -1.

**Tlb.put** it is constructed FIFO algorithm. Integer I is first value of list which is some page number. Remove this value in the list and same integer value of i remove in the table with connected frame number. And input page number add in list and table with frame number

**Page Table (pt)**

**Table** is array which size is 256. Initial value is all minus value and false

**PageTableItem** frame number is int type of frame number, valid is check whether this frame is used. True is using, false is not.

**GetFrameNumber** return frame number of table.

**Pt.get** get frame number which is connected with page number. If frame number is -1 which means that page table didn’t contain proper frame numbe,. Simply, page fault, return -1. Or not return frame number in the page table.

**Pt.add** logical address size and physical address size are same. So didn’t consider page replacement in the original code. Just input frame number in row which is decided by page number. And when it is entered in page table, valid will be true.

**Physical Memory (pm), frame size is 256byte, number of frame is 256**

**Frames** array which size is 256.

**CurrenctFreeFrame** integer value which is represent frame which is not using.

**Class Frame** each frame size is 256byte. Each frame has number from 0 to 255, when It is indicated by code, represent this data which is in physical memory by using arraycopy. Arraycopy(a,b,c,d,e) means that array a copy from a[b] to a[b+e] and paste to c array from c[d].

**Pm.addFrame** use free frame through sequentially free frame number. Return frame number which added to memory

**Pm.getValue** return data from specific frame number and offset in physical memory.

**Back Store (bs)**

**Backing store file** already save some value in the file

**Bs.getdata** using randomAccessFile.

🡪contain function in randomaccessfile: disk.seek define point position page number\*256. In this case, indicate from page number data in backing store

🡪contain function in randomaccessfile: disk.read Read from current pointer position till byte array length in this case value is 256 so read 256 byte length from backing store

Return result is data which is located from page number\*256 to page number\*256+255 in backing store

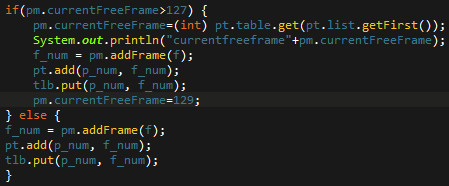
**Sequence of address translator**

Till logical address data will be finished, get frame number from tlb using **tlb.get**. if frame number is -1 in other words, no page number in tlb, tlb miss is increased and check page table, also no frame number is -1, page fault is occurred, read data from backing store using get data using **bs.getData**. And this data will be added in frame by using **pm.addframe** also set frame number. Than update page table and tlb using **pt.add, tlb.put.** physical memory is frame number \* 256 + offset, value get data by using **pm.getValue.**

**Compare original code and FIFO, LRU algorithm**

 **🡪 original code result**

 **🡪 FIFO code result**



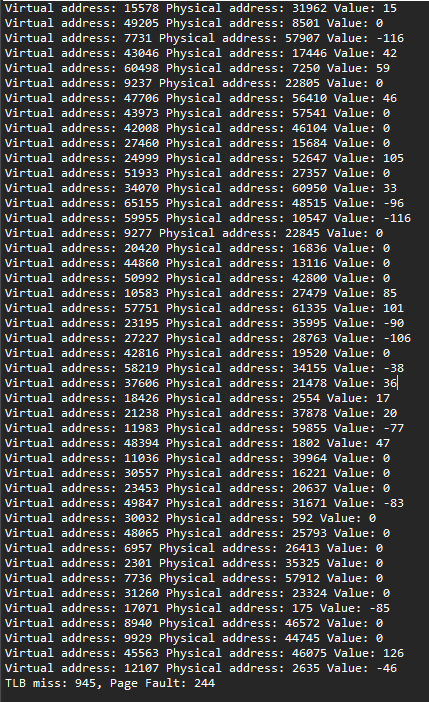
Original code contains valid and frame number in the page table. I modify the page table to 128 size hash table and make linked list to save page number which is arrange sequentially like TLB. And physical memory change 128 frame \* 256byte. First fill the physical memory sequentially and after fill all of the memory, remove first data which is input first from physical memory by sequence of list. Number of Page fault is increased because Physical memory size is smaller than original.

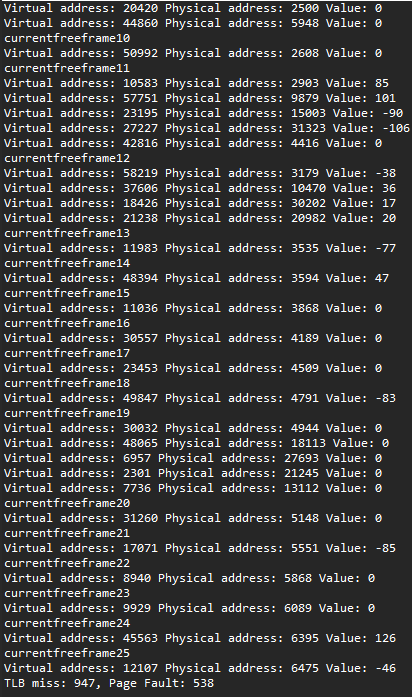
 **🡪 LRU code result**



**+add FIFO algorithm code**

Main difference between FIFO and LRU is occurred when page number which is already in page table enter page table. Although that page number input first than others, this page number is recently enter page table. So priority of removing is decreased, other first input page number will be deleted. In this LRU algorithm code, this list will be update this situation. If enter these page number, then delete that page number in the list, new add page number in the list. Also I consider that current free frame after 128 times to fill number of physical memory frame, extract page number of list. And check page table to find frame number which is matched with page number. And allocate this frame to current free frame. Of course, number of page fault is more than original.

 🡨 original code result

 🡨FIFO LRU🡪