**Assignment**

Couse: Operating Systems Design(146043-31001), 2021

Name: Chang wan Woo

Programming Assignment NUMBER: Implementation assignment #4

Due DATE: 2021/06/01

**Program environment**

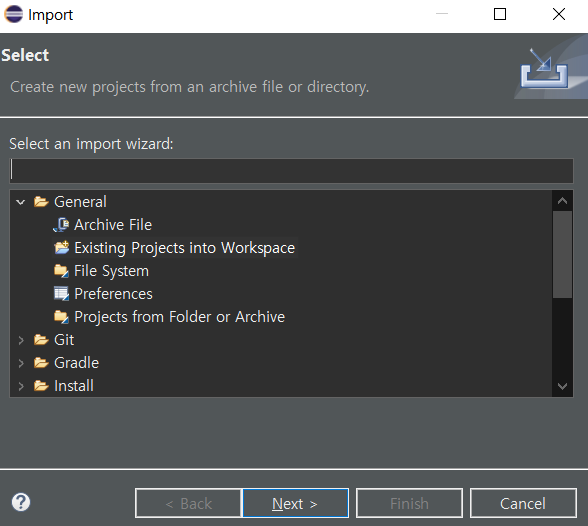
Window 10, Interllij, java version "1.8.0\_291",

Java(TM) SE Runtime Environment (build 1.8.0\_291-b10)

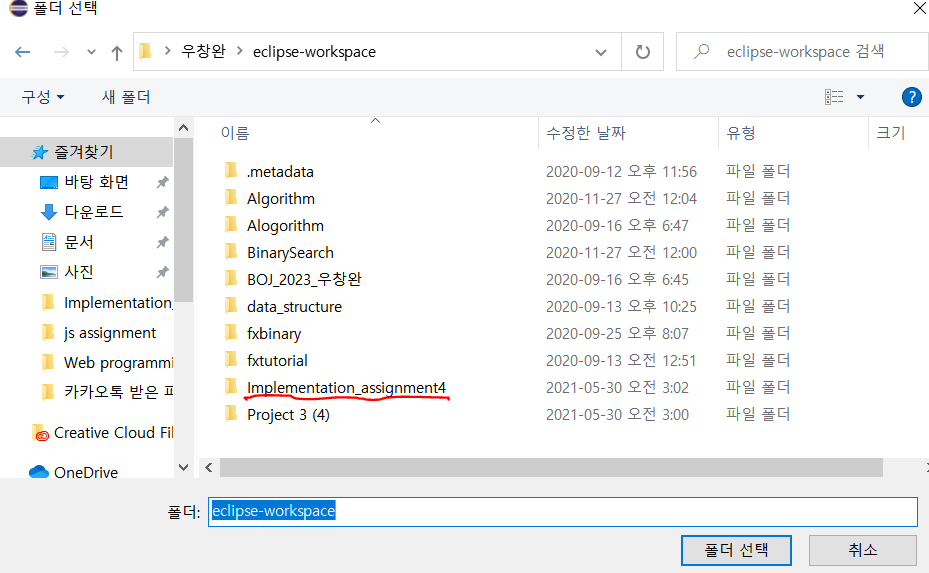
**How to start?**

**1.Unzip file into eclipse workspace.**

**2. Import ( File > import > existing Projects into workspace**



**3. Browse > implementation\_assignment4(it is located in your workspace)**



By these steps you can show my program an run my program as well. There are three algorithms to start. If you enter 0, algorithm without page replacement is proceeded. If you enter 1, virtual memory with FIFO algorithm is proceeded and lastly if you enter 2, virtual memory with LRU algorithm is proceeded.

Finally, you can see the result for TLB hit rate and number of page fault on the command line.

**Problem statement**:

In memory management, we use logical memory to separate physical memory to enhance performance. So, in this project I mapped logical memory and physical memory with page replace algorithms: FIFO and LRU. Also, we can see number of page fault and TLB hit rate for each algorithm.

**Software design:**

To make LRU and FIFO algorithms, I used linked list and queue in this project. When program need to replace page, each algorithm selects victims and replace its page and allocate frame. In LRU algorithm, linked list operated stack. If page reference is requested, program first check TLB, and if page is in TLB, it refers TLB, otherwise it looks page table. If there is page, it refers corresponding physical memory and otherwise it allocates free frame for request.

I implemented LRU and FIFO algorithms when size is larger than 128 which is frame size for two algorithms. In this process each algorithm selects victim for replacement. When **replacement** is occurred, I print out which frame is reallocated to page.

Analyzation

We need to know that main difference between original implementation and modification version is size of physical memory. The former does not happen page replacement because page size is larger than number of page faults. However, in FIFO and LRU algorithms page replacement is happened.

Origin:



In original problem, there are 256 frames. If there is no values which is not in page table page fault is occurred and put this page number and frame number to tlb and page table.

FIFO:

텍스트이(가) 표시된 사진

자동 생성된 설명

FIFO uses queue to manage page. If the page is full, page replacement is implemented so that victim for FIFO is selected. In FIFO, 538 pages replacement is happened. The reason why page fault is doubled is frame number decrease by 128 so that page replacement is often occurred by selecting victim. In FIFO algorithm, it just selects victim which is placed in header of queue.

LRU:

The LRU algorithm uses stack as data structure to store page table. Similarly, FIFO the LRU puts page number and frame number to page table and TLB until list is full. After the queue is full, LRU algorithm selects victim. The victim is selected in header in the list. The difference is that in this algorithm if page number is already in page remove this page and puts stack again.

텍스트이(가) 표시된 사진

자동 생성된 설명

**Result Analyzation**

As we can see above results, page fault happens 538, and 539 respectively. In general, LRU happens less page fault because of **consideration of locality**. However, in this projects number is random which does not consider locality.

**Implementation**

**Code explanation**

**1. If user takes input 0**

텍스트이(가) 표시된 사진

자동 생성된 설명

Frame size is 256, so page replacement does not occur, so it assigns new frame for corresponding page\_number and add frame to physical memory.

2.If user takes input 1

텍스트이(가) 표시된 사진

자동 생성된 설명

If user takes input 1, it executes page replacement with FIFO algorithm. FIFO algorithm selects victim if there are more than 128 inputs like as below.

텍스트이(가) 표시된 사진

자동 생성된 설명

Victim is selected in front of queue and rest table[victim].frameNumber=-1 and table[victim].valid=false. After reset, it updates page table, TLB and replacedPage list.

If size is less than 128, it allocate in physical memory and call FIFO\_add to add in page table

텍스트이(가) 표시된 사진

자동 생성된 설명

**3.If user takes input 2,**

텍스트이(가) 표시된 사진

자동 생성된 설명

Unlikely FIFO algorithm, LRU algorithm has to update list. If page is already in list, remove corresponding value and add to the top.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

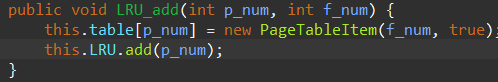
자동 생성된 설명

The LRU selects in front of value as a victim and likely FIFO, reset its frame number and validation.

텍스트이(가) 표시된 사진

자동 생성된 설명

If frame is less than 128, it adds page\_number without page replacement.



**Error handling:**

**In this program, it takes input from user. Input should be range from 0 to 2 otherwise program consider input invalid and print error message and take input again.**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Test Description and Results**

**Test1: Input:0**

**Input:**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Output:**

**TLB hit rate= 5.5% (55 happens)**

**Page fault rate 24.4% (244 happens)**



**We can see there is no page replacement,**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Test 2**

**Input: 1 (page replacement with FIFO)**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Output:**

**TLB hit rate 5.3%( 53 happen), page fault rate: 53.8% (538 happen)**

**The below image shows which page is replaced during executions.**



텍스트이(가) 표시된 사진

자동 생성된 설명

**We can see replacement and reference on the command line.**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Test 3**

**Input: 3 (Page replacement with LRU algorithm)**

텍스트이(가) 표시된 사진

자동 생성된 설명

**Output**

**TLG hit rate: 5.7%(57 times happen) Page fault rate 53.9%(539 times happen)**

**The below image shows which page is replaced during executions.**



텍스트이(가) 표시된 사진

자동 생성된 설명

**Also, as same as FIFO algorithm, if frame is needed to be replacement, program prints out which frame is reallocated to page.**

텍스트이(가) 표시된 사진

자동 생성된 설명

[Introduction]: The program being executed must be in main memory. However, in many cases, the entire program in not necessary, so by introducing virtual memory, it has benefits from virtual memory is that programs can be larger than physical memory and abstracts main memory into large, uniform array of storage so that it can implement multi programming and increase CPU utilization. Further it requires less I/O or swapping programs.

[Process]: If there is reference page before, it refers page table and mapped to physical memory using page number and offset. Otherwise, if the bit is set to invalid, it causes trap to operating system and store its current state. Operating system check internal table to determine the reference was a valid or an invalid memory access. If it is invalid, terminate the process. If it is valid operating system schedule secondary storage operation to bring desired page into newly allocated frame. After it is completed, operating system modifies page table to indicate the page is now in memory and restore instructed instruction.

[Analyzing code]

[AddressTranslator]

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

**[PageTable]**

텍스트이(가) 표시된 사진

자동 생성된 설명

The page table constructor allocate page table array and initiates it value by -1 and false.

텍스트이(가) 표시된 사진

자동 생성된 설명

The get(int p\_num) method is accessed by other class by sending parameter p\_num(parameter number). If there is corresponding frame number, return frame number otherwise returns -1.

텍스트이(가) 표시된 사진

자동 생성된 설명

The add method adds page number and frame number in table

**[PageTableItem]**

텍스트이(가) 표시된 사진

자동 생성된 설명

The page table item stores its frame number and valid information.

**[TLB]**

텍스트이(가) 표시된 사진

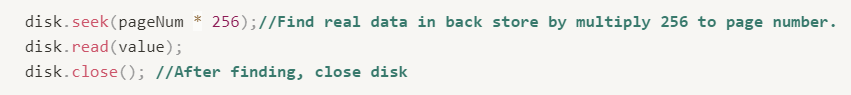
자동 생성된 설명

[BackStore]

In the BackStore, declare its value and result type and find corresponding real data in backstore.

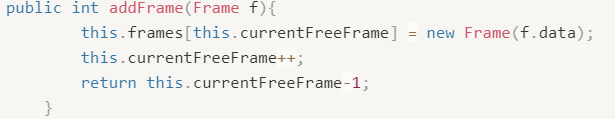
텍스트이(가) 표시된 사진

자동 생성된 설명



[Frame]

The addFrame method returns frame number.



The int getValue returns corresponding value in frame.

텍스트이(가) 표시된 사진

자동 생성된 설명