Design of Paging Simulation

Main method:

- Create an inverted page table which has following information: physical frame number, virtual page number, referenced(R) timestamp, and modified(M) timestamp according to input file read.
- Invoke different page replacement algorithm corresponding to <pr_algo>, showing the output results according to the sequence of page accessed read. It will show number of page hit, number of page fault, and number of page evict.
- There is a global timer that increments on every page access, and a last clock tick time that's updated one every clock tick, both shared by the entire page table.

FIFO method:

- Using FIFO page replacement. It creates a linked list to implement it. When page fault happens, it will remove the page at the head of page table, which is the first one(oldest page) coming in, and load the newest one from disk to the tail of page table, which is the most recent one coming in. Plus, when modified(M) timestamp is not equal to 0, it needs to be evicted.

LRU method:

- Using LRU page replacement. When page fault happens, by invoking method lra_index, it will return which entry is the least recently used. And the page just requested can directly overwrite into that place. Plus, when modified(M) timestamp is not equal to 0, it needs to be evicted.

CLOCK method:

- Using CLOCK page replacement. When page fault happens, the page being pointed to by the clock_index(hand) is inspected. If its R timestamp is less than clock_tick, the page is removed, the new page is loaded into the clock in its place, and the clock_index increments by one, meaning the hand is advanced one position. If R timestamp is greater than clock_tick, R timestamp will be reduced to less than clock_tick, which means cleared and the clock_index increments by one, meaning the hand is advanced to the next page. This process is repeated until a page is found with R timestamp less than clock_tick.Plus, when modified(M) timestamp is not equal to 0, it needs to be evicted.

Show output result:

- Every algorithm invoked will show the output of sequence of messages for page hit, page fault, page eviction or page load when page accessing with its operation.
- Every algorithm invoked will also show how many the total number of PAGE_HIT, PAGE_FAULT, and PAGE_EVICT are.

Comparing outputs from the 3 page replacement algorithms.

Test environment:

Frame size: 16

Virtual page number accessed range: 1~100(randomly produced)

	Page_Hit	Page_fault	Page evict	Hit Rate
FIFO	79	421	285	15.8%
LRU	75	425	285	15.0%
CLOCK	76	424	286	15.2%

Frame size: 256

Virtual page number accessed range: 1~500(randomly produced)

	Page_Hit	Page_fault	Page evict	Hit Rate
FIFO	188	312	45	37.6%
LRU	189	311	37	37.8%
CLOCK	189	311	42	37.8%

Performance analysis:

Based on the test environment and data given above, the final result of the three page replacement algorithms shows that these three perform almost the same. However, this result is not as the same as we expect in real word. Some reasons could explain why this result works in this way.

First of all, according to the experiment conducted in the real world, LRU and CLOCK is almost as efficient as each other. So let us take FIFO and LRU for example, FIFO keeps the things that are most recently added. LRU keeps the things that are most recently used in memory. Basically speaking, LRU is more efficient because there are generally memory items that are added once and never used again, and there are items that are added and used frequently. But what we used for the data is randomly produced, it does not match with what exactly the real situation we have been facing. For instance, people usually use data set which cover data in a certain range like array. That is why FIFO could perform as efficiently as LRU or CLOCK, even better than LRU or CLOCK.

Therefore, it might lead to more reasonable result when the three algorithms are run with real-world data set.

- ✓ *P.S: This project is simulated by using JAVA, use compliance from execution environment 'JavaSE-1.7'*