SUSTech CS302 OS Lan9 Report

Title: Page-replacement Algorithm

Name: Shijie Chen, Student ID: 11612028

Experimental Environment: Linux and C++11

Experiments

1. Algorithms
    - FIFO algorithm and its complexity
        - Replace the page that arrives first in the FIFO queue with the missed page.
        - $O(N)$ lookup and $O(1)$ replacement where N is the size of FIFO.
    - MIN algorithm and its complexity
        - Replace the page that won't be used for the longest time with the missed page.
        - $O(N)$ lookup and $O(MN)$ replacement. Where M is the total number of pages and N is the size of MIN.
    - LRU algorithm and its complexity
        - Replace the least recently used page with the missed page.
        - (N) lookup and $O(1)$ replacement.
    - Clock algorithm and its complexity
        - Using circular linked list to simulate a clock. The hand is always point at the next position of last replacement. When we check whether the element in the list, we don’t move the hand. And during the check procedure, the valid bit does not change. But if the element in the list, we finally change its valid bit to 1. When miss occurs, we start from the hand, and replace the first element whose valid bit is 0. During this procedure, remember to set the valid bit to 0. Increase the hand after replacement.
        - $O(N)$ lookup and replacement.
    - Second-Chance algorithm and its complexity
        - Second-Chance Algorithm
            1. We check in both FIFO list and LRU list. (hit)
            2. When the FIFO is not full, we just push in this part.
            3. When the FIFO is full and page miss happens, we move the last element in the FIFO list into the LRU list, and push the new element into the FIFO list.
            4. When the LRU list is full, we remove the least recent used element in the LRU list.
            5. When we find the query page in the LRU list, we move it into the FIFO list. Note that we also need to move the last element in FIFO to LRU list.
        - On average, $O(N)$ lookup and $O(N)$ replacement.
2. Fundamenal

- In theory, the optimal page-replacement algorithm is **MIN**, and prove it:

  In short, Min minimizes miss rate by removing the page that won't be used for the longest time.

  Strict proof of this optimality is hard. There are many papers dedicated to it. e.g. "Cohen, Ariel, and Walter A. Burkhard. "A proof of the optimality of the MIN paging algorithm using linear programming duality." Operations Research Letters 18.1 (1995): 7-13."

- Can the FIFO page-replacement algorithm be improved? If yes, please provide a plan; If no, please give your proof.

  Yes, use a hash set (`unordered_set` in C++11) to cut the lookup time complexity to $O(1)$. Since replacement is in $O(1)$, the overall complexity is $O(1)$.

- Can the LRU page-replacement algorithm be improved? If yes, please provide a plan; If no, please give your proof. Yes, use a hash set (`unordered_set` in C++11) to cut the lookup time to $O(1)$; Then use a heap (`priority_queue` in C++11) to cut the cost of moving the element to the front of LRU to $O(log(N))$ on page hit. Page replacement now is in $O(1)$. The overall complexity is $O(log(N))$.

3. Problems and Solutions With the help of stl, there is no significant problems during my implementation.

   I implemented LRU using `list`(linked list) instead of `priority_queue` (heap) for convenience. The implemented time complexity should be $O(N)$.

4. Program running result: (hit percentage)

Algorithm/ test	1.in	2.in	3.in
FIFO	11.98%	11.85%	82.36%
MIN	42.40%	43.27%	88.58%
LRU	11.76%	11.85%	82.39%
Clock	11.93%	11.83%	82.38%
Second-chance	11.85%	11.85%	82.39%