SUSTech CS302 OS Lab9 Report

Title: Page-replacement Algorithm

Name: Lu Ning , Student ID: 11610310

Time: 2019 Year 5 Month 6 Day

Experimental Environment: linux

Objective: Master the principle of page replacement and deeply understand the pros and cons of different page-replacement algorithms

Deadline: 11:59 AM, 2019-05-08

Summit by: Blackboard

Task：

Task 1. Understand the details and ideas of different page-replacement algorithms

Task 2. Write a program

Experiments:

1. Algorithms：

- **Briefly describe the FIFO page-replacement algorithm and analyze its algorithm complexity**

  FIFO maintains a queue. When a new page comes in and miss, if the queue is not full, just push it into the queue, otherwise pop the front of the queue and push the new one.

  O(1)

  Analysis: Just maintains a queue for FIFO and map for check hit.

- **Briefly describe the MIN page-replacement algorithm and analyze its algorithm complexity**

  MIN maintains a list. When a new page misses and cache is full, just replace the old page which will be used furthest in the future.

  O(k) k: cache size

  Analysis: Need to record the occur times of each page, so need to scan the page sequence. O(n) before run the algorithm. The average time for each step is O(1)

  And each time it need to scan the cache to get the replace page O(k)

So the time complexity is O(k)

- **Briefly describe the LRU page-replacement algorithm and analyze its algorithm complexity**

  Replace old page that is least recently used (last occurs is the oldest) when the cache is full.

  O(1)

  Analysis: LRU page just use the list end O(1), the map will check hit O(1), the map will store the page to the position in the list, so move the page to the head of list is O(1).

- **Briefly describe the clock algorithm and analyze its algorithm complexity**

  Use a circular list to store the cache. Head always point to the cache after the last hit/replace.

  Hit: set valid bit to 1

  Miss: From head to find the position whose valid bit is 0. If valid bit is 1, set it to 0, find next. If we find a position whose valid bit is 0, replace it and set bit to 1.

  O(k) k: cache size

  The worst case is scan all cache to find the replace position O(k). The best case is O(1). So the average is O(k).

- **Briefly describe the second-chance algorithm and analyze its algorithm complexity**

  To avoid removing some important page from the cache. If hit in FIFO, do nothing. If hit in LRU, move the hit page from LRU to FIFO and pop the front in FIFO to LRU. If miss happens, push new page into FIFO, pop the oldest and put it into LRU.

  O(1)

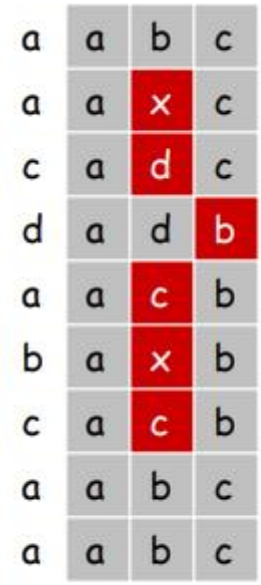
  FIFO and LRU operation are both O(1).

2. Fundamental:

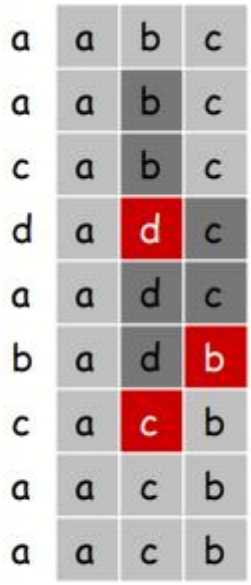
- **In theory, the optimal page-replacement algorithm is ____min_____, and prove it:**

Def: A **reduced** schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

Claim: We can transform an unreduced schedule S into a reduced one S' with no more cache misses.



an unreduced schedule



a reduced schedule

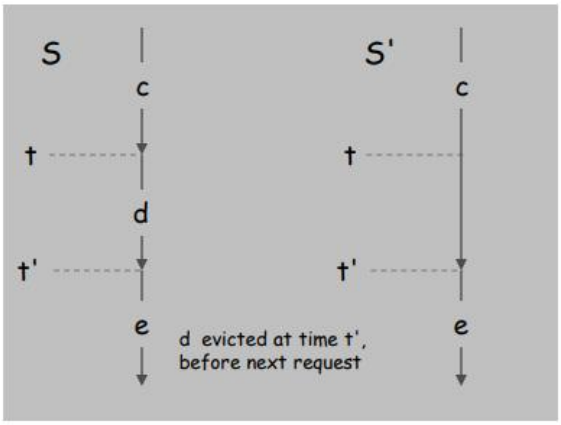
Proof of the claim:

Suppose S brings d into the cache at time t, without a request.
Let c be the item S evicts when it brings d into the cache.

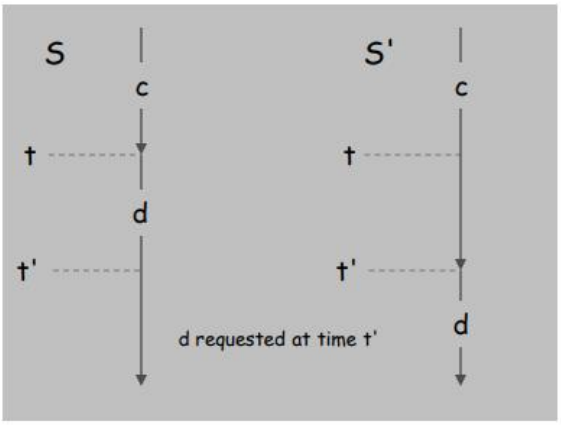
Case 1: d evicted at time t', before next request for d.

Case 2: d requested at time t' before d is evicted.

In both cases, the S' has less or equal replacement time than S.



Case 1



Case 2

Def :

Invariant: There exists an optimal replacement schedule **S** that makes the same replacement as schedule **S_{min}** produced by min algorithm through the first j requests.

Prove the Optimal by induction:

Let S be reduced schedule that satisfies invariant through j requests. We produce S' that satisfies invariant through the first j+1 requests.

Assume the (j+1)th request is **d(j+1)**

From invariant, we know that the optimal **S** and **S_{min}** has the same cache contents before the (j+1)th request.

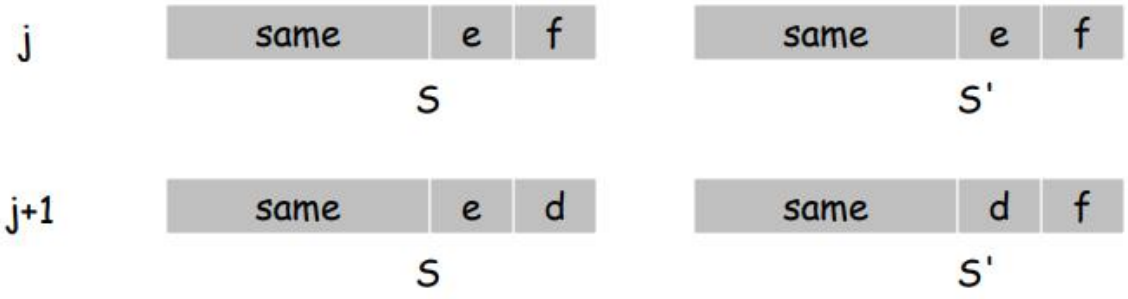
There are 3 cases:

Case 1: (**d(j+1)** is already in the cache). Just hit, S' = S satisfies invariant.

Case 2: (**d(j+1)** is not in the cache but **S** and **S_{min}** choose the same element to be replaced). S' = S satisfies invariant.

Case 3: **d(j+1)** is not in the cache; S_{min} replace e; S replace a different element f.

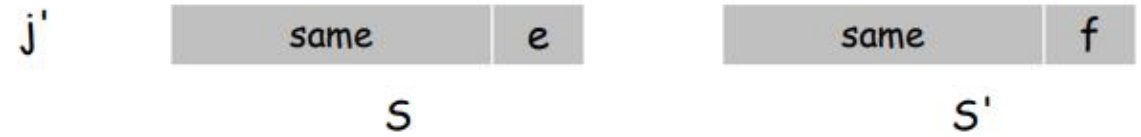
Because of the claim of reduced S, we can reduce the new optimal S to new S_{min}, then the j+1 case is also invariant.



Proof: Having e in cache is not worse than having f in cache.

Let **k** be the first time after j+1 that S and S_{min} take a different action, and let g be item requested at time **k**.

Time K:



Case i: g == e

Cannot happen because it means e occurs before f but min algorithm chooses the the farthest-in-future elements.

Case ii: g == f, S_{min} hit.

Element f can't be in cache of S, so let e' be the element to be replaced in S.

If e' == e, then S_{min} and S have the same cache

If e' != e, then it becomes:

S: [same' f e']

S_{min} : [same' f e]

The difference is e' and e. We can replace e with e' in S_{min}

(S_{min} is no longer reduced, but it can be transformed into a reduced schedule that agrees with S through step j+1)

Case iii: g != e or f

Because at time **k** S and S_{min} must do different action, then S must replace e and

S_{min} must replace f. Then S and S_{min} have the same cache.

Hence, in all these cases, we have a new reduced schedule S that is same as S_{min} through the first j + 1 items and incurs no more misses than S does, which means it is optimal.

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

  No, because the complexity now is O(1)

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

  No, the ,the complexity now is O(1). But the time will be long in practical use because it needs to update the LRU list every memory reference, which may slow the every memory reference.

3. Problems you meet and your solutions

- **Problem1:**

  Some optimization in the algorithm.

- **Solution1:**

  Use unordered_map instead of map.(RB tree to hash, the search time from O(logn) to O(1))

4.Program running result：(hit percentage)

Algorithms/test	1.in	2.in	3.in
FIFO	11.98	11.85	82.36
MIN	42.4	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

Conclusion:

Learn to use the STL lib and learn the page replacement algorithm.