운영체제 실습 과제3 보고서

컴퓨터학과

2017320208

이연준

3-1. Stack\_LRU

먼저 reference string에서 각 요소의 차례가 올 때 마다 frame이 꽉 차있는지 여부를 검사한다. 먼저 꽉 차있지 않은 경우, 현재 reference page가 frame 내에 존재하는지 검사한다. 그 후, frame내에 존재한다면 frame에서 해당 reference page를 frame의 맨 앞으로 즉, top으로 올리고 나머지 원래 0번 index ~ reference page가 있던 직전 index에 있던 것들은 모두 한 칸씩 뒤로 옮겨주는 작업을 한다. 만약 frame 내에 존재하지 않는 다면 frame 내에 존재하는 모든 것들을 한 칸씩 뒤로 옮겨주고 frame의 맨 앞(top)에 해당 reference page를 위치시키고 page fault 횟수를 1만큼 증가시켜준다.

꽉 차있는 경우, 마찬가지로 reference page가 frame 내에 존재하는지 검사하고 만약 존재한다면 위와 같이 해당 reference page를 맨 앞으로, 0번 index로 옮기고 원래 0번 index ~ reference page가 있던 직전 index에 있던 것들을 한 칸씩 뒤로 옮겨준다. 그리고 없는 경우에도 마찬가지로 frame들에 있던 모든 것들을 뒤로 한 칸씩 옮겨주는데 이 과정은 맨 마지막에 있는 reference page, 즉 가장 옛날에 reference된 page(LRU page)를 swap out 시켜주는 것과 같다. 그리고 없는 경우에는 마찬가지로 page fault 횟수를 1만큼 증가시켜준다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ref string  frame | 5 | 1 | 2 | 3 | 5 | 0 | 4 | 9 | 3 | 0 |
| 0 | 5 | 1 | 2 | 3 | 5 | 0 | 4 | 9 | 3 | 0 |
| 1 | . | 5 | 1 | 2 | 3 | 5 | 0 | 4 | 9 | 3 |
| 2 | . | . | 5 | 1 | 2 | 3 | 5 | 0 | 4 | 9 |
| 3 | . | . | . | 5 | 1 | 2 | 3 | 5 | 0 | 4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 5 | 6 | 1 | 1 | 2 | 0 | 4 | 3 | 2 |
| 7 | 5 | 6 | 1 | 1 | 2 | 0 | 4 | 3 | 2 |
| 0 | 7 | 5 | 6 | 6 | 1 | 2 | 0 | 4 | 3 |
| 3 | 0 | 7 | 5 | 5 | 6 | 1 | 2 | 0 | 4 |
| 9 | 3 | 0 | 7 | 7 | 5 | 6 | 1 | 2 | 0 |

3-2. Clock Algorithm

Clock algorithm은 frame을 circular하게 관리하는 방식으로 만든다.

먼저, 포인터가 가리키는 index를 clockpin이라는 변수로 다룬다. 또, 각 frame 별로 reference bit을 ref\_bit 배열을 통해 다룬다.

먼저, 알고리즘도 stack과 다르게 현재 들어온 reference page가 frame 내에 존재하는지 확인하고 있다면 해당 reference page가 저장되어있는 frame의 reference bit를 1로 바꿔준다.

만약 존재하지 않는다면, 현재 clockpin이 가리키고 있는 index로부터 한 칸씩 앞으로 가면서reference bit가 0인 frame을 찾는다. 이 때, reference bit가 1인 index의 경우 reference bit를 0으로 바꾸고 지나간다. 그리고 mod 연산(%) 을 통해 clockpin이 frame을 circular하게 돌아가면서 확인할 수 있도록 한다. 그리고 frame에 존재하지 않는 reference page가 올 때마다 page fault 횟수를 1씩 증가시켜준다.

3-3. Additional Reference Bits Algorithm

Additional Reference Bits Algorithm의 경우 위의 clock 알고리즘과 같이 reference bit를 사용하는데 다른 점은 frame별로 두는 것이 아닌 전체 page별로 reference bit를 8bit만큼 할당한다. 따라서 ref\_bit 배열의 크기가 page\_max와 같도록 선언하였다.

Reference string 순서대로 작업을 진행하는데 먼저 pos 변수를 통해 frame이 꽉 차있는 상태인지 확인을 하고 모든 reference bit에 bit연산인 >> 1을 수행시켜준다.(reference page가 올 때 마다)

꽉 차있지 않은 경우, frame 내에 현재의 reference page가 있는지 확인한다. 만약 존재할 경우 해당 reference page의 reference bit에 2진수로 8번째 bit에 해당하는 값인 128을 더해준다. 그리고 만약 존재하지 않는 경우, 해당 page의 reference bit에 128을 더해주는 것은 같지만 이 경우에는 frame에 존재하지 않는 page를 가져와야 하므로 page fault 횟수를 1만큼 증가시켜준다.

꽉 차있는 경우에는 위와 같이 frame 내에 reference page가 존재하는지 확인한다. 존재하는 경우는 위와 과정과 같이 reference bit에 128을 더해준다. 존재하지 않는 경우에는 위와 다르게 현재 frame에 존재하는 page들의 reference bit값을 확인하고 값이 가장 작은 page를 victim frame으로 지정해 frame에서 swap out 시켜준 뒤, 현재 reference page를 frame으로 swap in 시켜준 뒤, 해당 page의 reference bit에 128을 더해준다. 그 뒤, swap out된 reference page의 reference bit를 0으로 바꿔준다. 마지막으로, page fault 횟수를 1만큼 증가시켜준다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ref string  Page num | 5 | 1 | 2 | 3 | 5 | 0 |
| 0 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 10000000 |
| 1 | 00000000 | 10000000 | 01000000 | 00100000 | 00010000 | 00000000 |
| 2 | 00000000 | 00000000 | 10000000 | 01000000 | 00100000 | 00010000 |
| 3 | 00000000 | 00000000 | 00000000 | 10000000 | 01000000 | 00100000 |
| 4 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 5 | 10000000 | 01000000 | 00100000 | 00010000 | 10001000 | 01000100 |
| 6 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 7 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 8 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 9 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ref string  Page num | 4 | 9 | 3 | 0 | 7 | 5 |
| 0 | 01000000 | 00100000 | 00010000 | 10001000 | 01000100 | 00100010 |
| 1 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 2 | 00001000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 3 | 00000000 | 00000000 | 10000000 | 01000000 | 00100000 | 00010000 |
| 4 | 10000000 | 01000000 | 00100000 | 00010000 | 00000000 | 00000000 |
| 5 | 00100010 | 00010001 | 00000000 | 00000000 | 00000000 | 10000000 |
| 6 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 7 | 00000000 | 00000000 | 00000000 | 00000000 | 10000000 | 01000000 |
| 8 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 | 00000000 |
| 9 | 00000000 | 10000000 | 01000000 | 00100000 | 00010000 | 00000000 |