오퍼레이팅 시스템

-과제2-

-LRU, LRU Approximation 구현-

이름: 오윤석

학번: 12171651

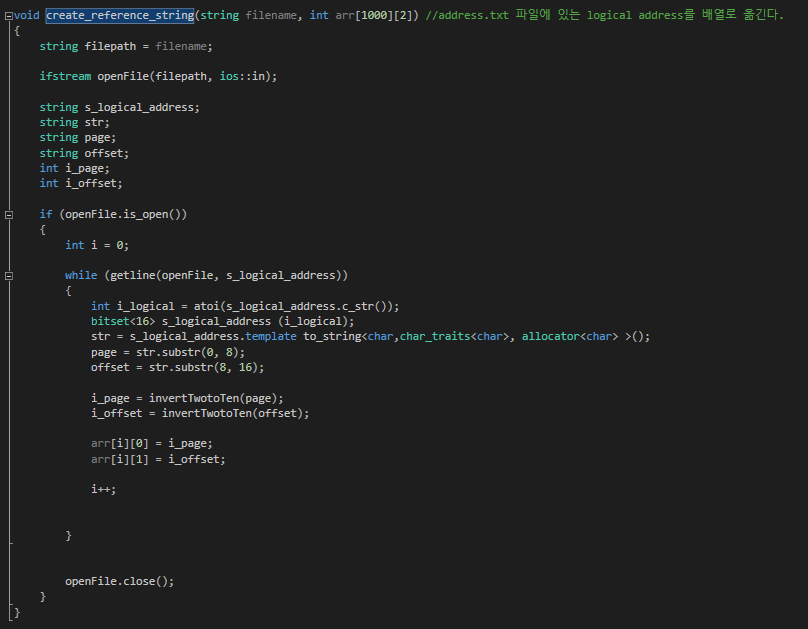
학과: 컴퓨터 공학

분반: 004

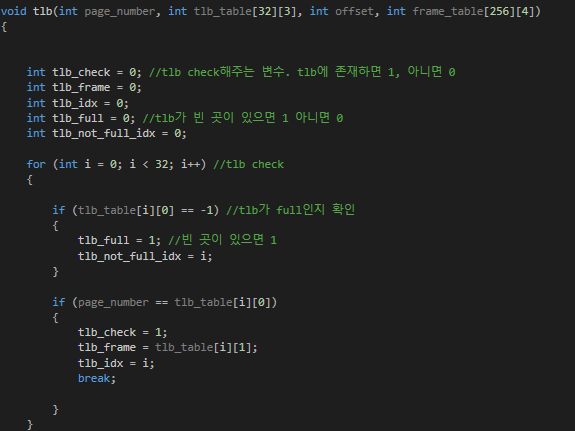
**목적**

과제 2의 목적은 세부적으로 보면 LRU와 LRU Approximation의 구현이다. 그리고 크게 보면 logical address가 frame에 할당되었을 때 진행되는 과정을 구현하는 것이다. 구현을 하기 전에알아야 되는 개념이 있다면 page replacement와 address translation이다. 이 과제의 목적은 tlb table을 사용하여 lru, lru approximation 알고리즘으로 page fault를 줄이면서 frame에 page를 할당하는 것이다.

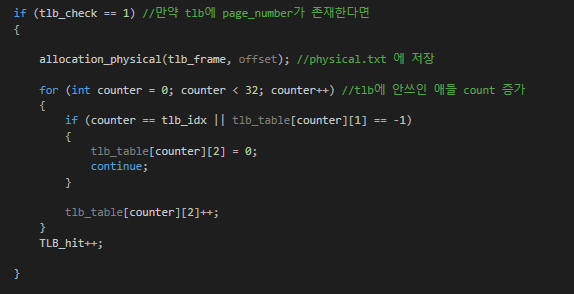
**설계방식 및 구현 상세**

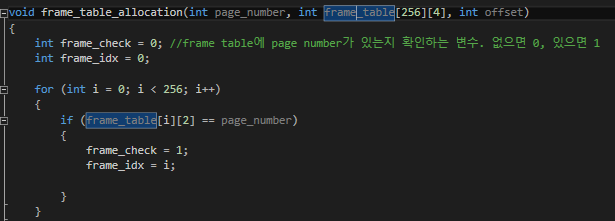
우선 설계 방식은 원래는 page table이 있어야 되지만 우리 과제에서는 frame table을 사용하였다. 그리고 과제 pdf에 지정되어 있는 크기를 사용해야 한다.  
우선 address.txt에 있는 1000개의 주소를 input으로 받는다는 가정하에 코드를 설계하였다. Logical address 1000개가 들어오면 이를 create\_reference\_string 함수를 가지고 arr[i][[0]에는 page number를 할당, arr[i][1]에는 offset을 할당하였다.

그러고 나서 본격적으로 logical address가 이동을 한다. 여기서 offset은 frame에 할당할 때만 필요해서 나중에만 다루었다.   
우선 LRU에 대해서 설명을 하겠다. Arr에 할당된 page\_number들은 tlb 함수를 통해 모든 과정이 이루어진다. Tlb 함수에 대해 설명을 하자면 우선 page\_number가 들어오면 tlb에 존재하는지 확인을 한다. 또한 tlb가 꽉차있는지 확인한다.

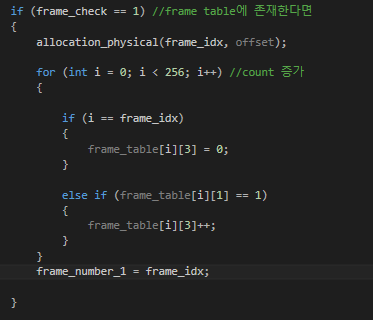


그리고 나서 만약에 tlb에 존재하게 된다면 이를 allocation\_physical 함수를 가지고 physical에 address translation후 physical.txt에 physical address로 전환을 해서 저장을 한다. 그리고 나서 쓰이지 않은 TLB\_table에 할당되어 있는 page들만 count를 1개씩 증가시킨다. 여기서 tlb\_table array에 대해 설명을 하자면 tlb\_table[32][3]으로 되어있고, 순서는 page#, frame#, lru counter# 로 구성되어 있다.

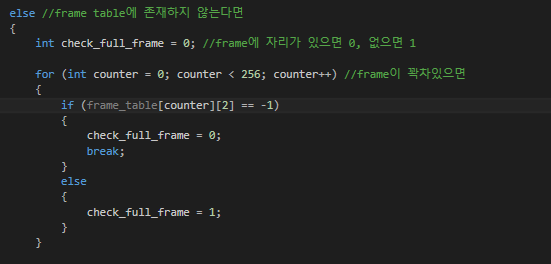


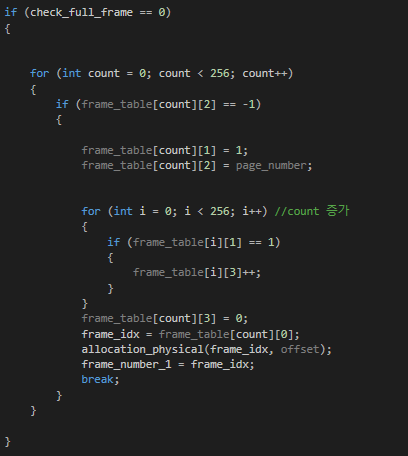
다시 본론으로 돌아와서 만약 tlb에 존재하지 않는다면 frame table을 확인하러 가기 위해 frame\_table\_allocation 함수로 이동하게 된다. 이 함수인에서는 우선 들어가면 frame tabe에 해당 page number가 존재하는지 확인을 하게 된다. 여기서 frame table은 frame\_talbe[256][4]로 구성되어 있고, 순서는 #frame, #flag, #page, #counter로 구성되어있다.

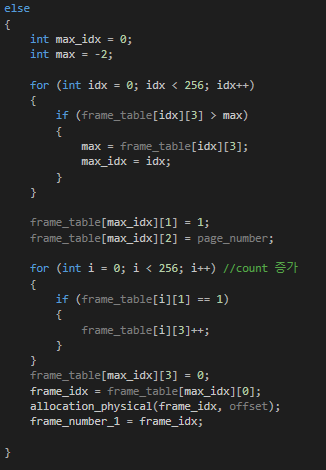
다시 frame\_table\_allocation 함수로 돌아와서, 만약 frame table에 해당 page number가 존재하게 된다면 physical에 address translation 후physical.txt에 저장하게 되고, 해당 frame을 제외한 나머지 frame에 대해서 count를 증가하게 된다.



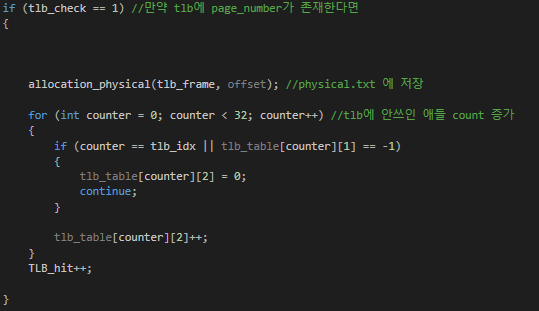
그러고 나서 다시 tlb 함수로 돌아가게 되는데 frame table에 존재하지 않았을 때 설명을 하고 tlb에 돌아가는 것을 설명하겠다.   
만약 frame table에 page number가 존재하지 않는다면, 우선 frame table이 꽉 차있는지 확인을 한다. 만약 꽉 차 있으면 여기서도 replacement가 발생하게 되는데 우리 과제에서는 page number가 0~255이고 frame table의 size도 256이어서 replacement는 절대 발생하지 않는다.  
그러고 나서 frame table에 page number와 frame number를 할당하게 된다.

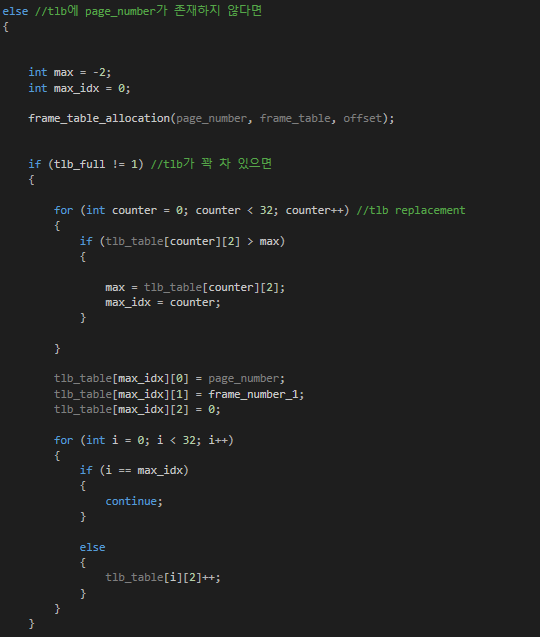


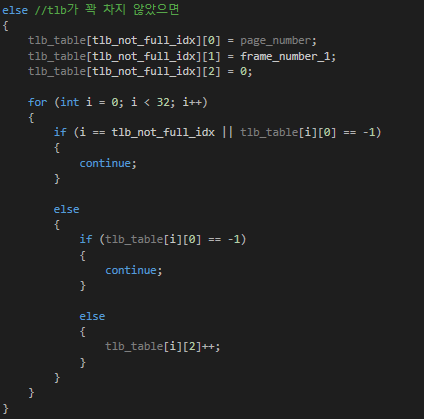


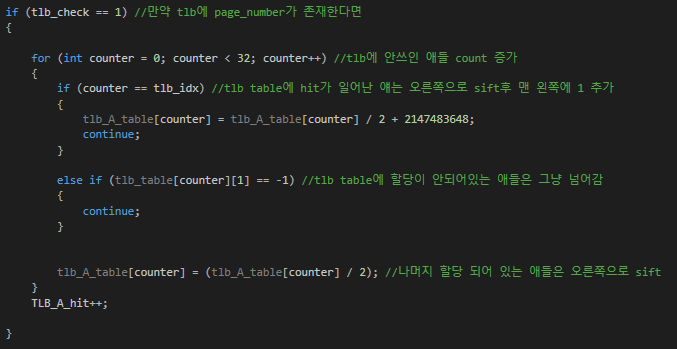


그러고 나서 frame\_table에 갔다 온 애들을 tlb table에 할당을 해준다. 그리고 얘는 0으로 초기화 해주고, 나머지는 다 count를 증가시켜준다.

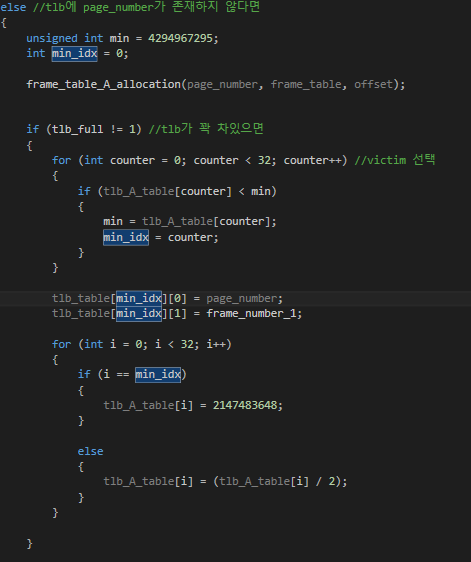


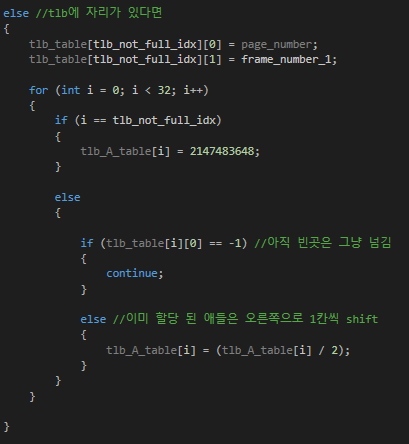




그리고 LRU Approximation이 진행된다. 대체적으로 진행 과정은 LRU와 같다. 차이점은 LRU에서는 LRU count를 사용하여 잘 사용이 안 된 얘가 victim으로 설정되지만 LRU Approximation에서는 additional reference bits algorithm을 사용하여 LRU보다는 더욱 효율적으로 victim을 사용한다. LRU Approximation 구현은 LRU와 구현 방식이 비슷해 달라진 부분만 설명하겠다. LRU Approximation에서는 unsigned int 자료형으로 되어 있는 tlb\_A\_table[32]가 쓰이게 된다. 이 변수는 tlb\_table과 같은 idx로 mapping 되어 있다. 역할은 총 32비트의 count를 저장하는 것이다. 만약 해당 tlb가 쓰이게 되면 tlb\_A\_table에서는 쓰인 얘만 오른쪽으로 shift 한 후 1을 할당하고, 나머지 애들은 오른쪽으로 sift만 하게 된다. Victim은 tlb\_A\_table에서 가장 작은 값을 가지고 있는 얘가 victim으로 설정하게 된다. 원래는 bitset 라이브러리를 사용하여 직접 2진수로 할당을 하려고 했지만 그 방법보다는 오른쪽으로 shift 할 때는 2로 나누어 주고, 맨 왼쪽에 1을 할당해줄 때는 2147483648(10000000000000000000000000000000)를 더해주는 방식으로 진행하였다.

위 사진은 tlb에 page number가 존재할 때 진행되는 알고리즘이다.



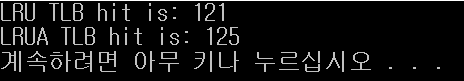


이것은 존재하지 않을 때 가장 작은 값을 victim으로 설정하고 page replacement가 진행된 후 쓰인 애만0으로 초기화, 안쓰인 애들은 shift만 한 알고리즘이다.

LRU Approximation의 frame 할당 과정은 LRU와 같아서 설명은 생략하겠다.

**결론**

결론은 LRU와 LRU Approximation의 tlb hit count를 보고 설명하겠다.



그냥 LRU는 TLB hit가 121번 일어났고, LRU approximation은 총 125번의 TLB hit가 일어났다. 물론 둘의 작동방식은 비슷하지만 LRU approximation이 더 효율적인 방식으로 page replacement가 덜 방생하는 것을 알게 되었다.

참고 사항은 다른 logical으로 테스트 할 때는 main에 filepath 변경과 for문 1000번 count를 logical 개수로 바꾸시면 됩니다. frame\_table2.txt은 lru a가 일어났을 때 frame 결과입니다.