****

컴퓨터 정보 공학과 3학년

12131579 이진아

**오퍼레이팅 시스템**

**3차 과제**

담당 교수: 송민석 교수님

**1. 개요**

구현 목표 : Virtual Memory의 주소를 Physical Memory의 주소로 Paging 기법을 통해 Address translation을 구현하여 Virtual Memory을 이해한다. 또한 TLB를 이용하여 MMU의 사용을 이해하고 TLB와 Page Table을 LRU방식을 사용하여 구현하여 Memory가 찼을 시 어떻게 해결하여야 하는지 이해한다.

**2. 개념 설명**

**1)Virtual Memory**

RAM을 관리하는 방법의 하나로, 각 프로그램에 실제 메모리 주소가 아닌 가상의 메모리 주소를 주는 방식을 말한다. 이러한 방식은 멀티태스킹 운영 체제에서 흔히 사용되며, 실제 주기억장치보다 큰 메모리 영역을 제공하는 방법으로도 사용된다.

**2)MMU(Memory Management Unit)**

CPU가 메모리에 접근하는 것을 관리하는 컴퓨터 하드웨어 부품이다. 가상 메모리 주소를 실제 메모리 주소로 변환하며, 메모리 보호, 캐시 관리, 버스 중재 등의 역할을 담당하며 간단한 8비트 아키텍처에서는 뱅크 스위칭을 담당하기도 한다.

**-TLB(Translation Lookaside Buffer)**

가상 메모리 주소를 물리적인 주소로 변환하는 속도를 높이기 위해 사용되는 캐시로, 약칭은 **TLB**이다.현재 모든 데스크탑 및 서버용 프로세서는 하나 또는 그 이상의 TLB를 메모리 관리 하드웨어에 가지고 있다. 페이지 단위나 세그먼트 단위로 사용하는 가상 메모리를 사용하는 모든 하드웨어는 TLB를 사용한다. CPU는 1차적으로 TLB에 접근하여 원하는 페이지가 존재하는지 탐색하고, TLB에 존재하지 않을 경우 MMU의 페이지 테이블을 참조한다.

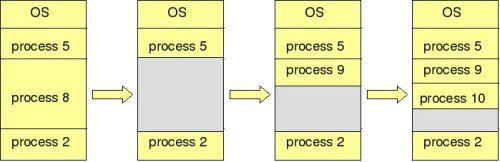
**3)Address Translation**

가상적으로 주어진 주소를 가상 주소(virtual address) 또는 논리 주소(logical address) 라고 하며, 실제 메모리 상에서 유효한 주소를 물리 주소(physical address) 또는 실주소(real address)라고 한다. 가상 주소의 범위를 가상 주소 공간, 물리 주소의 범위를 물리 주소 공간이라고 한다.

**<주소변환 방법>**

**-Continuous Allocation**

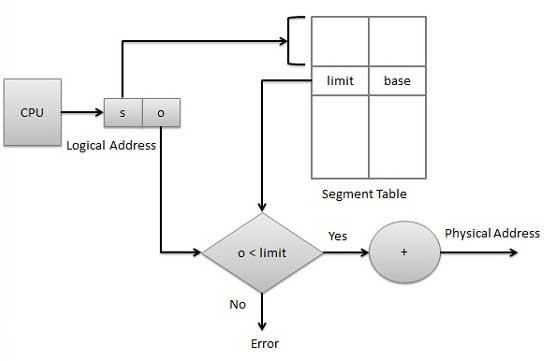
**Base and Limit**방식이다. 메모리에 프로세스를 연속적으로 할당하는 방식으로, 특정 프로세스가 메모리의 어떤 위치에 올라가 있다면 그 시작 위치가 base이고, 그 끝 위치가 limit이다.

[](http://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjOjp2og6DNAhUHW6YKHY8sCisQjRwIBw&url=http://www.google.co.kr/url?sa%3Di%26rct%3Dj%26q%3D%26esrc%3Ds%26frm%3D1%26source%3Dimages%26cd%3D%26ved%3D0ahUKEwjOjp2og6DNAhUHW6YKHY8sCisQjRwIBw%26url%3Dhttp%3A%2F%2Fwww.it.uu.se%2Fedu%2Fcourse%2Fhomepage%2Fos%2Fvt06%2Foverview%2Fmemory%26psig%3DAFQjCNGo7enztgHI-xybbeQCi7991xN9KA%26ust%3D1465736158517029&psig=AFQjCNGo7enztgHI-xybbeQCi7991xN9KA&ust=1465736158517029)

* First Fit : 그냥 들어갈 수 있는 가장 앞에 있는 공간에 넣는다.
* Best Fit : 들어갈 수 있는 공간 중 가장 작은 공간에 넣는다.
* Worst Fit : 들어갈 수 있는 공간 중 가장 큰 공간에 넣는다.

**-Segmentation**

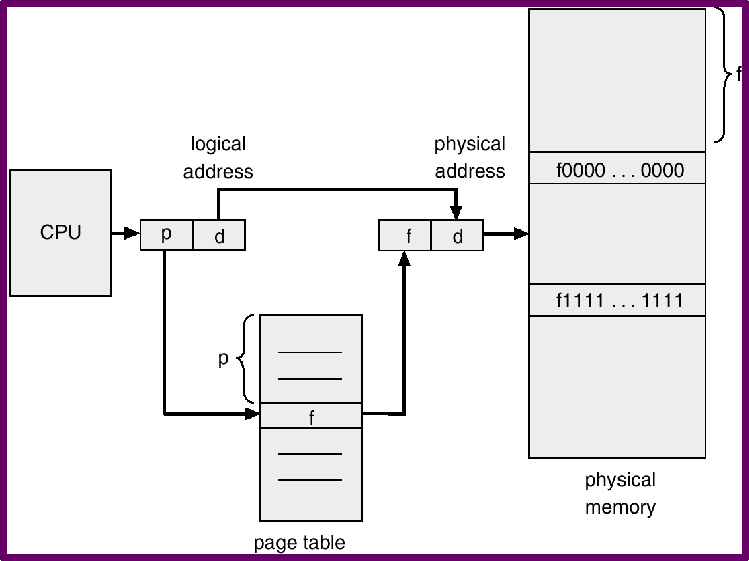
일정한 frame의 크기대로 page를 나누는 것이 아니라, 프로그램 내부적으로 용도별 segment로 세분화 한다.

[](http://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwj5gIfBhqDNAhUnW6YKHXWcBCsQjRwIBw&url=http://www.tutorialspoint.com/operating_system/os_memory_management.htm&bvm=bv.124272578,d.dGY&psig=AFQjCNHVR78tvxBi67jlLI8zvXiAyjooOQ&ust=1465737139181863)

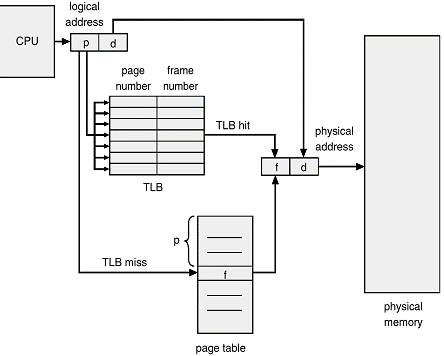
**-paging**

가상기억장치를 모두 같은 크기의 블록으로 편성하여 운용하는 기법이다. 이때의 일정한 크기를 가진 블록을 **페이지**(page)라고 한다. 주소공간을 페이지 단위로 나누고 실제기억공간은 페이지 크기와 같은 프레임으로 나누어 사용한다.

페이징 기법이 적용된 시스템에서 가상주소는 순서쌍(p,d) ( p , d ) {\displaystyle (p,d)} 로 나타낼 수 있다. pp {\displaystyle p} 는 가상기억장치 내에서 참조될 항목이 속해 있는 페이지 번호이고, dd {\displaystyle d} 는 페이지 p {\displaystyle p} 내에서 참조될 항목이 위치하고 있는 곳의 변위이다.

[](http://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjc3q3bhqDNAhUFLKYKHeZMCrsQjRwIBw&url=http://www.ustudy.in/node/2693&bvm=bv.124272578,d.dGY&psig=AFQjCNF3u53wgCjlwGgiQSCJQJpEbKNexA&ust=1465737190468223)

**-TLB사용**

[](http://www.google.co.kr/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwjn8r65iKDNAhXjG6YKHZ3FA6sQjRwIBw&url=http://www.gitam.edu/eresource/comp/gvr(os)/8.4.htm&psig=AFQjCNGd5pyyW2Q9-iEgQ7KH7jzb3dEKjA&ust=1465737651542839)

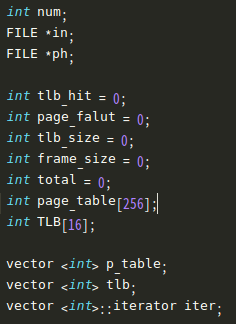
출처 : 위키백과, <http://iamflower.tistory.com/9>, <http://www.gitam.edu/eresource/comp/gvr(os)/8.4.htm>

**3.프로그램 설계 원칙 및 아이디어**

Paging을 통한 address translation을 해야 하므로 page table과 TLB를 구현하여야 했다. Page table을 index의 값이 page number, 배열 안에 들어 있는 값이 frame number인 일 차원 배열로 구현하였다. stack형식의 LRU를 사용하기 위해 처음에 배열을 사용하려 했지만 배열을 사용 하면 해당 page number를 찾은 뒤 맨 앞으로 옮겨 주고 맨 앞부분부터 page number의 전까지를 뒤로 하나 씩 다 미뤄주는 작업을 해야 하므로 시간을 좀 더 줄이기 위해 vector를 사용하였다. Vector를 사용 하면 중간에 값을 빼주면 알아서 차례대로 연결을 해주기 때문이다. 따라서 page table과 TLB 둘 다 LRU를 사용 하기 위한 vector를 따로 둬 구현하였다. Vector의 앞 부분에는 가장 최근에 접근 하지 않은 page number를 저장하고 vector의 뒷부분에는 가장 최근에 접근 한 page number를 두어 구현하였다. 처음에 page number를 입력 받고 해당 page가 TLB에 있는지 여부를 가장 먼저 확인 한 후 TLB를 먼저 update 해주고 그 다음에 page table을 update해주는 방식을 생각하여 구현하였다. 또한 TLB와 page table둘다 다 찼을 경우와 다 차지 않았을 경우를 따로 생각해 줘야 하기 때문에 각각 table의 크기를 저장하는 변수를 global 변수로 두었다. 또한 page\_fault와 TLB hit수를 저장 하는 변수 도 global변수로 두어 사용하였다.

**4.코드 설명**

**1)구조체 및 변수 생성**

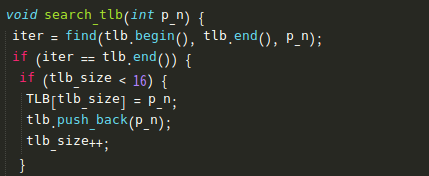


num은 입력이 들어오는 virtual address의 값을 저장하는 변수이다. FILE \*in은 읽을 파일의 이름이고 ph는 결과 값을 저장할 파일의 이름을 담는 변수이다. tlb\_hit는 tlb에 hit하는 수를 저장하고 page\_falut는 falut의 수를 젖아한다. tlb\_size와 frame\_size는 각각 tlb의 크기과 frame의 크기를 저장하며 total은 총 들어오는 virtual address의 수를 저장한다. Page\_table배열은 page에 대한 frame의 값을 저장하고 있고 tlb는 tlb에 들어있는 page의 number를 저장하고 있다 벡터로 구현한 p\_table과 tlb는 각각 page\_table과 TLB에 저장되어 있는 page number를 LRU에 의해 순서대로 저장된다. 앞부분이 가장 늦게 접근 한 page이고 뒷 부분이 가장 최근에 접근한 page가 저장된다. Iter는 LRU할 시 벡터 주소를 받아 사용하기 위해 선언하였다.

**2)search\_tlb 함수**

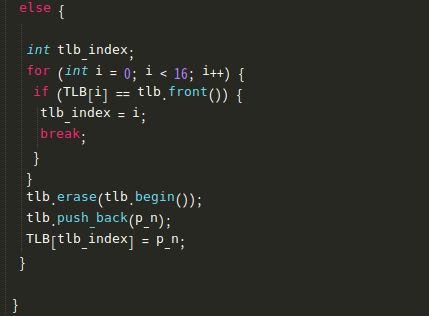
**2.1)TLB안에 없는 경우**

**2.1.1)TLB table이 다 차지 않은 경우**



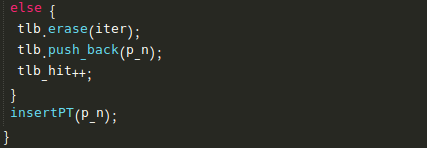
Virtual address의 값이 들어오면 TLB에 있는지 여부를 먼저 확인을 해주는 함수이다. Page의 number를 파라미터 값으로 넘겨 받아 find함수를 통하여 TLB안에 해당 page가 들어 있는지 여부를 판단한다. Iter의 값이 tlb.end()와 같다는 의미는 해당 page가 TLB안에 들어 있지 않다는 뜻이다. 이 때 아직 tlb의 크기가 16보다 작은 즉 다 차지 않은 상태라면 TLB에 넣어주고 TLB의 순서를 가지고 있는 벡터의 마지막 부분에 해당 page의 number를 넣어준다.

**2.1.2)TLB table이 다 찬 경우**



16보다 크거가 같으면 TLB table이 다 차있다는 뜻이다. TLB table이 다 찼지만 TLB안에 없기 때문에 가장 늦게 접근한 page를 지우고 그 위치에 입력 받은 page를 넣어주어야 한다. 따라서 0부터 16까지 TLB의 배열을 search하여 제일 마지막에 접근한 page number의 TLB에서의 위치를 찾아낸다. tlb벡터의 가장 앞 부분이 마지막에 접근한 page이므로 앞부분을 지워주고 뒷 부분에 입력받은 page의 number를 넣어준다. 그 후 TLB의 찾은 위치에 page number를 넣어준다.

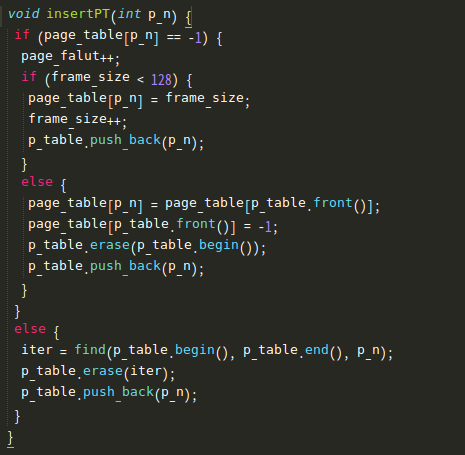
**2,2)TLB안에 있는 경우**

****

이미 TLB table안에 있으므로 tlb벡터의 순위만 바꿔주면 된다. 따라서 입력받은 page number가 있던 위치를 지워주고 가장 뒷 부분에 page number를 넣어 준다. 이 경우 TLB hit를 했으므로 tlb\_hit를 하나 증가시켜 준다.

TLB를 update시켜주었으면 Page table을 update해주기 위해 insertPT를 실행하여 준다.

**3)insertPT 함수**

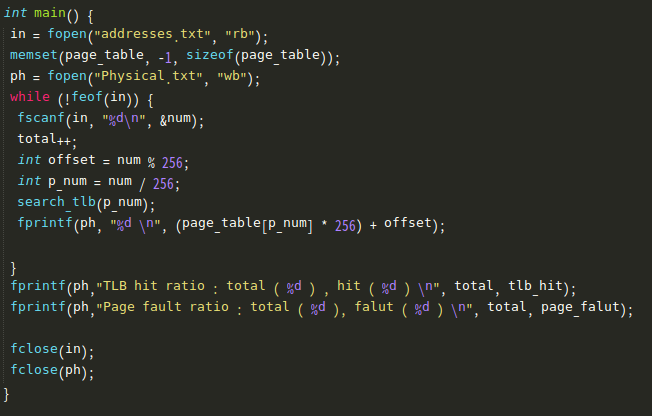


TLB table을 update한 후 Page table을 update해주기 위한 함수이다. 이 함수도 파라미터로 page의 number를 넘겨 받는다. page\_table에 연결 된 frame이 없는 경우 -1로 표시를 해주었다.

따라서 입력 받은 page number에 저장 된 page\_table의 값이 -1인 경우 연결 된 frame이 없는 경우 이므로 page\_falut의 수를 하나 증가 시켜 준다. Falut가 되었으나 frame\_size가 128보다 작은 경우 아직 모든 frame을 할당하지 않은 경우 이므로 page\_table에 입력 받은 page number에 frame을 하나 할당 해 주고 frame\_size를 하나 증가 시켜 준다. 그리고 page\_table의 순위를 저장하는 p\_table벡터에 입력받은 page number를 넣어준다. Frame\_size가 128보다 크거나 같은 경우 즉 모든 frame을 다 할당 해준 경우이므로 p\_table벡터의 가장 앞부분에 있는 page number에 할당 해주었던 frame number를 입력 받은 page number에 할당해준다. 그리고 가장 앞부분에 있던 page의 page\_table에는 할당 된 frame이 없다는 의미로 -1을 넣어준다. 그 후 p\_table의 맨 앞부분을 없애주고 가장 뒷 부분에 page number를 넣어준다.

이미 할당 된 frame이 있는 경우 find함수를 통해 p\_table에서 page number가 저장 된 위치를 찾고 그 위치를 없애 준 후 가장 뒷 부분에 page number를 넣어준다.

**4)main 함수**

****

fopen함수를 통해 파일을 읽어 드린다. 읽은 것은 in에 저장하고 page\_table을 -1로 초기화 해준다. Ph는 저장하는 file이다. File을 읽으면서 num을 읽는다. 읽을 때 마다 total값을 증가 시켜주고 밑에 8bit를 offset 상위 8bit를 page number로 쓰기 때문에 256을 나눈 나머지를 offset에 몫을 p\_num에 저장시켜준다. 그 후 p\_num을 넘겨 search\_tbl를 실행시키고 physical address를 출력한다. while문을 다 수행 한 후 전체 입력 받은 수 중 TLB hit ratio, Page fault ratio를 출력하고 fclose로 file을 닫은 후 종료한다.