**REPORT**

**| Operating Systems**

**| Virtual Memory Manager**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

# Objective

Operating system 수업 시간에 배운 내용인 Demand paging, Address translation, Page fault, Page replacement를 구현해 Virtual Memory Manager를 design하는 것이 이번 과제의 목표였습니다..

지난 과제들과 마찬가지로 구름IDE를 통해 Linux 환경에서 컴파일, program 실행, 파일 입출력을 했습니다.

# Program

원래는 여러 개의 process가 각자의 page table을 가지면서 frame에 할당되는 방식이어야 하지만 과제에서는 한 개 process의 virtual address만을 고려했습니다. Virtual address space의 크기는 bytes이고 각 page의 크기는 bytes 이므로 page table에는 개의 entry를 받을 수 있습니다. Frame의 크기는 page와 동일하고, 256개의 frame으로 구성되어 있으니 physical address space의 크기는 virtual address space와 같습니다.

**Demand paging**

수업 시간에 배운 내용에서의 demand paging은 비어 있는 frame에 현재 실행될 부분의 page를 할당하고, 실행되지 않는 부분은 할당하지 않는 방식이었습니다. 입력으로 받는 logical address를 이용해 page number와 offset을 구할 수 있습니다.

Page number = logical address / page의 크기

Offset = logical address % page의 크기

위 수식으로 구한 page number가 이미 frame에 할당되어 있는지를 page table을 lookup해서 확인합니다. 할당되어 있지 않을 때 page fault가 발생 했으므로 frame table을 lookup해서 가장 앞쪽에 비어 있는 frame에 해당 page를 할당해 줍니다.

**Frame table**

Demand paging을 통해 page table을 갱신하면서 frame table 또한 갱신했습니다. Frame number를 인덱스로 갖는 구조체 배열로 frame 할당 여부, 할당 되어 있는 page number를 저장해 Frame\_table.txt에 출력했습니다.

**Address translation**

위 수식으로 logical address에 해당하는 page number와 offset을 계산할 수 있습니다. Page number에 해당하는 frame number 또한 demand paging을 사용해 할당이 되어 있다면 그 값을 사용하고, 할당 되어 있지 않다면 비어 있는 frame에 할당 후 그 값을 사용합니다. Page table을 lookup하면 physical address 또한 계산할 수 있게 됩니다.

Physical address = frame number \* frame의 크기 + offset

위 수식으로 구한 physical address를 Physical.txt에 차례대로 출력했습니다.

**TLB / Page replacement**

과제에서 TLB는 32개의 entry를 갖습니다. TLB table에는 page number, frame number가 저장되는데 가장 최근에 사용된, page fault가 일어난 page number를 저장하면서 갱신했습니다. 실제 TLB처럼 빠른 메모리 접근은 구현하지 못했지만, 32개 entry를 갖는 TLB 내에 page number가 존재하는지, 즉 TLB hit가 발생했는지를 확인하는 부분은 구현했습니다. TLB의 entry가 32개이다보니, 갱신을 할 때 page replacement algorithm (FIFO)를 사용해 가장 처음에 들어온 page number의 위치 인덱스를 유지하면서 갱신했습니다. TLB hit ratio와 최종 TLB는 TLB.txt에 출력했습니다.

**변수**

* Struct frame | frame의 할당 정보가 저장되는 구조체
  + int used | 할당 여부 ( free = 0, used = 1 )
  + int pageN | 할당된 page number
* struct pf | TLB에 저장된 page/frame number가 저장되는 구조체
  + int pageN | page number
  + int frameN | frame number
* int p\_table[256] | 256개 entry를 갖는 page table
* pf TLB[32] | 32개 entry를 갖는 TLB
* frame f\_table[256] | 256개 entry를 갖는 frame table
* int l\_Add | 입력 받는 logical address
* double inputs | 입력의 개수
* int frameUses | 할당 중인 frame의 개수, frame table의 인덱스
* double TLBhits | TLB hit 횟수
* int TLBidx | TLB에 접근할 인덱스, FIFO를 위함
* FILE \*in\_lAdd | logical address file을 입력
* FILE \*out\_pAdd | physical address file을 출력
* FILE \*out\_Frame | frame table file을 출력
* FILE \*out\_TLB | TLB hit ratio, 최종 TLB 출력

**코드 실행**

1. Page table의 값을 모두 -1로 초기화합니다. 아무 것도 할당 되어 있지 않은 상태를 -1로 표시했습니다.
2. TLB도 마찬가지로 모두 -1로 초기화합니다.
3. Addresses.txt 파일로부터 logical address를 입력 받습니다.
4. 입력 개수인 inputs를 증가시키고, page table을 lookup해서 frame에 할당되어 있는지 확인합니다.
5. 할당되어 있지 않다면 가장 처음 비어 있는 frame에 해당 page를 할당하고, page fault가 일어났으므로 page table, frame table, TLB를 모두 갱신해줍니다. 이 때, TLB는 FIFO algorithm을 통해 갱신됩니다.
6. 이미 할당되어 있다면 TLB를 lookup해서 TLB 내에 해당 page number가 저장되어 있는지 확인합니다.
7. 저장되어 있다면, TLB hit가 발생한 것이므로 TLB hit 횟수를 증가시켜줍니다.
8. 저장되어 있지 않다면, 해당 page number가 마지막으로 사용되었으므로 TLB에 추가해줍니다. 이때 TLB는 FIFO algorithm을 통해 갱신됩니다.
9. 입력 받은 logical address를 통해 physical address를 계산해 Physical.txt에 출력합니다. 이때 page table을 lookup해 할당되어 있는 frame number를 찾아서 계산합니다.
10. 파일의 끝에 도달할 때까지 3. ~ 9.의 과정을 반복합니다.
11. Frame\_table.txt에 현재 frame table의 상태를 출력합니다.
12. TLB hit ratio를 계산해 TLB.txt에 최종 TLB와 함께 출력합니다.

**결과 및 분석**

출력 이미지를 보면, Linux 환경에서 이 프로그램을 정상적으로 컴파일 및 실행했습니다. Logical address들이 저장되어 있는 input 텍스트 파일에 맞는 출력을 가진 output 텍스트 파일 3개도 알맞게 생성되었습니다.

입력 받는 값은 1, 256, 32768, 32769입니다. 1을 입력 받았을 때, 0번 page를 0번 frame에 할당하고, page table, frame table, TLB를 모두 갱신해 줍니다. 256을 입력 받았을 때, 1번 page를 1번 frame에 할당하고 마찬가지로 갱신해 줍니다. 32768, 32769를 입력 받았을 때, 두 logical address들의 page number는 128로 같습니다. 따라서 32768을 입력 받았을 때는 page fault가 발생하지만, 32769를 입력 받았을 때에는 page fault가 발생하지 않습니다. 그리고 TLB hit도 발생합니다.

출력 이미지를 보면, frame table에 0, 1, 2번 frame에 각각 0, 1, 128번 page가 할당되어 있다는 것을 알 수 있습니다. 따라서 physical address는 각각의 frame number \* frame 크기에 offset을 더한 값인 1, 256, 512, 513입니다. TLB도 page fault가 일어났을 때마다 갱신했으므로 이미지와 같은 결과를 나타냅니다. TLB hit ratio는 128번 page를 두 번 입력 받았는데 두 번째 입력 때에는 TLB에 128번 page가 저장되어 있었으므로 TLB hit는 총 4번 입력 중에 1번 발생합니다. 계산하자면 TLB hit 횟수 / 전체 입력 횟수이므로 이 계산 값을 출력했습니다.