INHA UNIVERSITY

Operating System

homework#3

과 목 명 : operating system
담당교수 : 송민석 교수님
제 출 일 : 2017년 06월 06일

인하대학교 공과대학
12131597 정민교

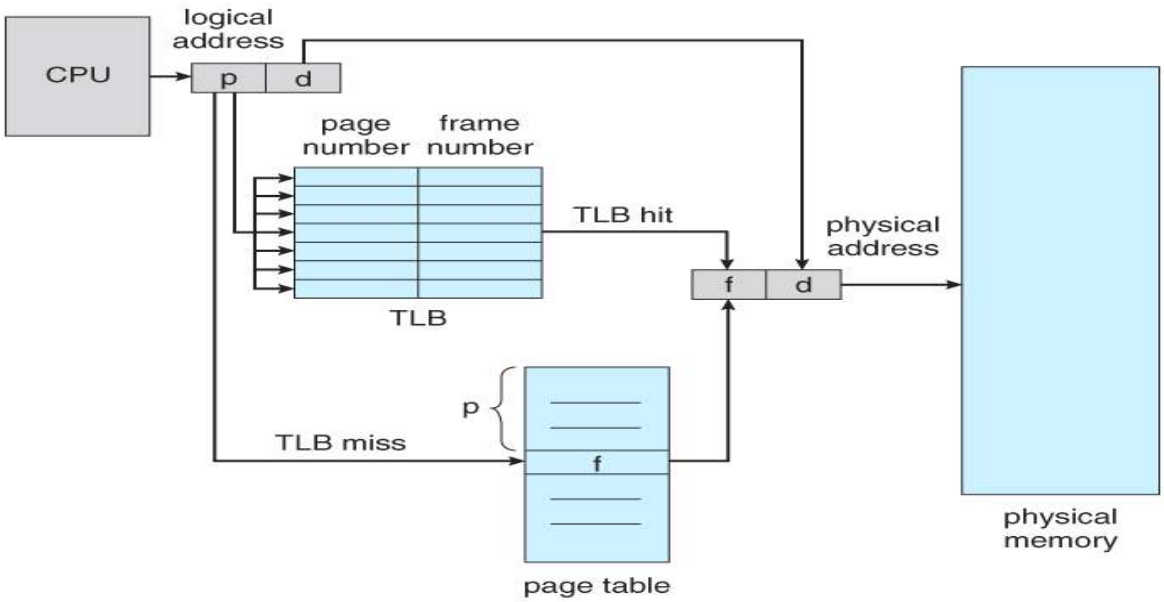
목 차

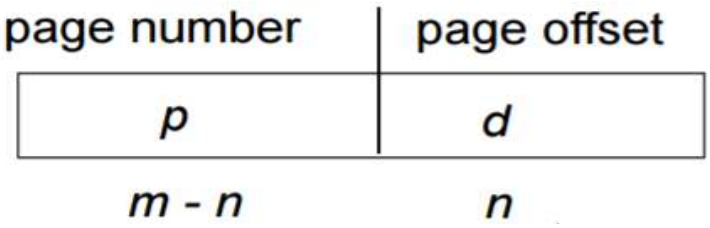
1. paging 프로그램 설명

· 목적

- OS하는 메모리 관리 기법을 직접 구현 해보면서, 어떻게 메모리를 변환하는지, 어떤 알고리즘으로 교체를 하였는지 직접 시뮬레이션을 하기 위한 목적.

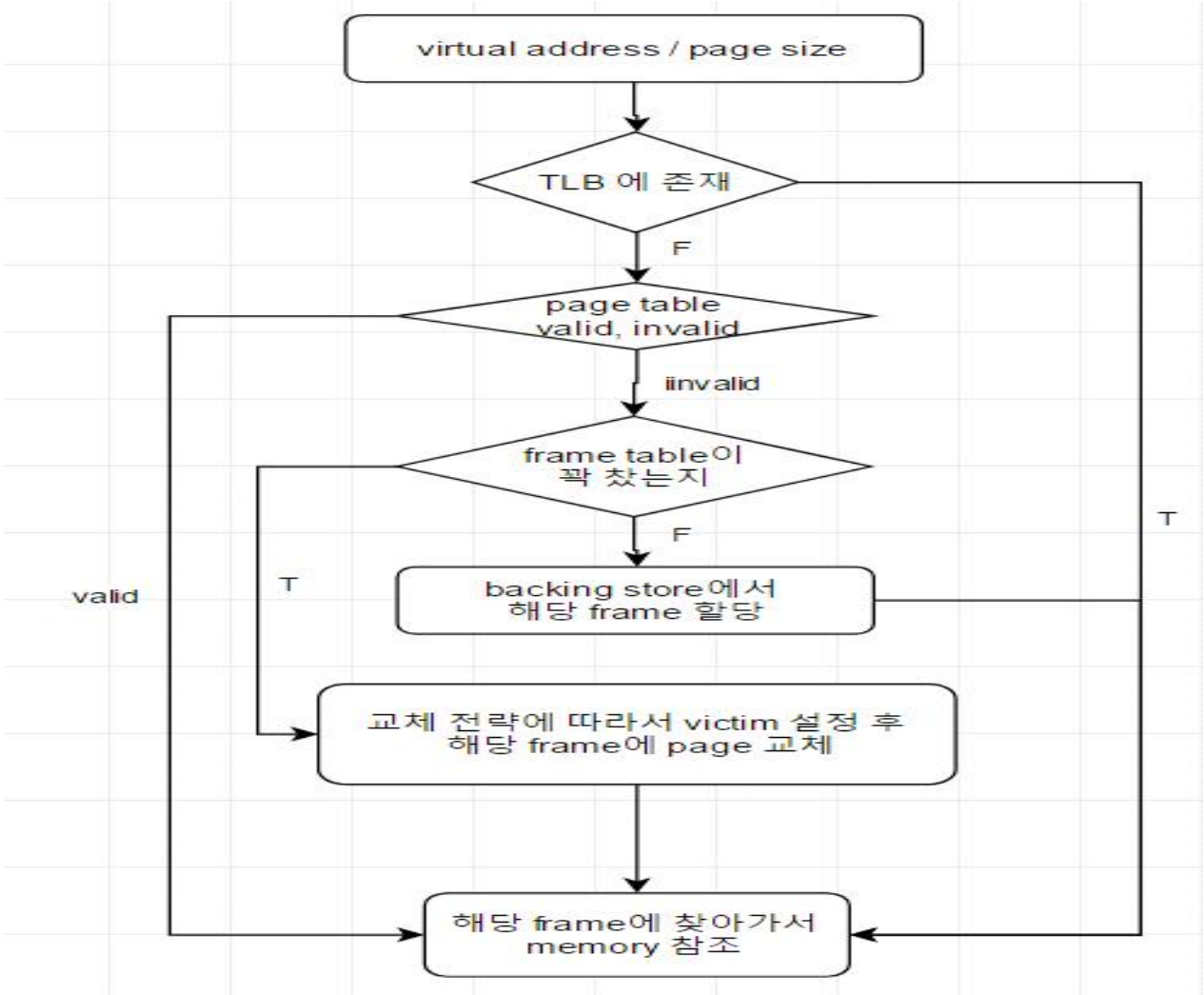
· 동작





설정

- virtual address의 크기가 2^m이고, page의 크기가 2^n일 경우, 상위 m-n비트는 page number를 나타내고 하위 n비트는 offset을 나타낸다.
  여기서는 virtual address의 크기는 16비트, 페이지의 크기는 8비트로 정의.
- page table의 entry는 256개, TLB entry는 16개, frame은 128개로 정의.
- page size = 256bytes, frame size = 256bytes.



1. 파일로 입력받은 virtual address를 page 크기로 나눈 후, 상위 비트는 page number이므로 먼저 TLB에 가서 찾으려고 하는 page가 있는지 확인한다.
2. TLB에 있는 경우 해당하는 frame의 위치에 가서 memory를 참조한다.
3. TLB에 없는 경우 page table에 가서 찾으려고 하는 page의 valid 여부를 파악한다.
4. page가 valid할 경우 해당하는 frame의 위치에 가서 memory를 참조한다.
5. page가 invalid할 경우 frame table이 꽉 찼는지 확인한다.
6. frame table이 꽉 차지 않을 경우 비어 있는 frame을 찾아서 비어 있는 frame에 backing store에서 page를 가져와서 해당 frame에 할당해준다.
7. frame table이 꽉 찼을 경우 해당 교체 알고리즘에 의해 victim을 찾은 후 victim page와 swap을 해준다.

2. 교체 전략

1. FIFO
 선입 선출 알고리즘
 page table에 가장 먼저 들어온 page를 victim으로 설정하고, page를 swap 해준다.
 virtual address가 들어온 순서대로 frame이 할당되므로, frame이 작은 순서가 먼저 page에 들어온 순서이다. 따라서 변수 하나를 선언, 0으로 초기화를 시킨다. swap이 필요할 경우 변수의 값과 같은 frame을 찾아서 그 page와 swap을 하면 된다.
 swap이 끝날 경우, 변수를 1증가시켜주면 된다. 만약 변수 값이 128이 넘어가면 다시 0으로 초기화를 시켜준다.

2. LRU
 각 page 마다 count 값을 가지고 있다. virtual address값이 들어와서 처리 될 때 마다 전체 page의 count값을 증가시켜준다.
 이미 frame이 할당된 page가 reference가 될 경우, 그 page의 count 값을 0으로 초기화시켜준다.
 swap을 해야할 경우, 전체 page 중 count 값이 가장 큰 page가 가장 최근에 사용되지 않은 페이지 이므로, 그 page의 frame을 새로 요청받은 page에 frame을 할당해준다.

3. 프로그램 구현 방법

```
struct page
{
        int frame;
        int count;
};
struct TLB
{
        int page_num;
        int frame_num;
};

struct frame
{
        int page;
        int valid;
        int virtual_address;
};
struct TLB TLB_t[16]={-1,-1};
struct frame frame_table[128]={-1,0,0};
struct page page_table[256]={-1,0};
```

page table : page구조체 배열

TLB : TLB구조체 배열

frame table : frame구조체 배열

<프로그램 수행 과정>

```
for(k=0; k<256; k++)
{
        page_table[k].frame =-1;
        page_table[k].count =0;
}
```

page table의 frame의 값은 –1로 초기화를 해준다.

```
for(k=0; k<=i; k++)
{
    if(TLB_t[k].page_num == num/256)
    {
        TLB_hit++;
        hit++;
        check=1;
        page_table[num/256].count=0;
    }
}
```

파일에서 가져온 virtual address를 256으로 나눈 몫이 page number이고, 그 값이 TLB에 있는 지 TLB배열을 처음부터 확인한다.
TLB배열에 있는 경우 check 값을 1, 없을 경우 0으로 변환한다.

```
if(check==0 && i >=16 && lock==1)   //TLB fault
{
    //tlb change

    if(page_table[num/256].frame == -1) // page_table fault
    {
        page_table[num/256].frame = i;
        frame_table[i].page=num/256;
        frame_table[page_table[num/256].frame].valid = 1;
        page_table[num/256].count=0;
        fprintf(wfp,"Virtual address: %d Physical address: %d\n",num,256*page_table[num/256].frame + num%256);
        fseek(rfp,256*page_table[num/256].frame,SEEK_SET);
        fread(buffer,256,1,rfp);
        for(k=0; k<256; k++)
        {
            phy_memory[256*page_table[num/256].frame+k] = buffer[k];
        }

        for(k=0; k<16; k++)
        {
            if(page_table[TLB_t[k].page_num].count > max)
            {
                max = page_table[TLB_t[k].page_num].count;
                maxpt = k;
            }
        }

        TLB_t[maxpt].page_num = num/256;
        TLB_t[maxpt].frame_num = page_table[num/256].frame;
        i++;
    }
    else//no fault
    {
        for(k=0; k<16; k++)
        {
            if(page_table[TLB_t[k].page_num].count > max)
            {
                max = page_table[TLB_t[k].page_num].count;
                maxpt = k;
            }
        }

        TLB_t[maxpt].page_num = num/256;
        TLB_t[maxpt].frame_num = page_table[num/256].frame;
        fprintf(wfp,"Virtual address: %d Physical address: %d\n",num,256*page_table[num/256].frame + num%256);
        page_table[num/256].count=0;
        hit++;
    }
```

TLB에 존재하지 않고, page table의 frame 값이 –1 이면 fault가 일어났다는 것이므로, page table에 frame을 할당해준다.
page table의 값이 –1이 아닌 경우, page에 해당하는 frame이 존재한다는 말이므로 해당 frame으로 physical address를 변환한다.

frame이 다 찾을 경우, swap을 한다.

-LRU 알고리즘일 경우

```
for(k=0; k<256; k++)
{
        if(page_table[k].count > victim)
        {
                victim = page_table[k].count;
                victimpt = k;
        }
}
page_table[num/256].frame = page_table[victimpt].frame;
frame_table[page_table[num/256].frame].page = num/256;
frame_table[page_table[num/256].frame].valid = 1;
page_table[victimpt].frame = -1;
page_table[victimpt].count = 0;
```

refrence되면 page count값을 0으로 바꾼다.

page fault가 일어날 경우, page table에서 count 값이 가장 큰 page를 victim으로 설정한 후, 그 page와 swap을 한다.

-FIFO 알고리즘일 경우

```
page_table[num/256].frame = j;
frame_table[page_table[num/256].frame].page = num/256;
frame_table[page_table[num/256].frame].valid = 1;
```

가장 먼저 들어온 page를 victim으로 설정하고 그 page와 swap을 한다.

```
FILE *wfp2 = fopen("frame_table_FIFO.txt","w");
for(i=0; i<128; i++)
{
        fprintf(wfp2,"%d %d %d\n",i,frame_table[i].valid,frame_table[i].page * 256);
}
fclose(wfp2);

FILE *wfp3 = fopen("physical_memory_FIFO.txt","w");
for(i=0; i<65536; i++)
{
        fprintf(wfp2,"%d\n",phy_memory[i]);
}
fclose(wfp2);
printf("FIFO hit ratio : total (%d) hit (%d)\n",total,hit);
printf("TLB hit ratio : total (%d) hit (%d)\n",total,TLB_hit);
```

physical memory와 frame table을 파일로 저장하고, hit률을 출력한다.