**Assignment #1**

컴퓨터 공학부 3학년 201711335 이호용

과제를 해결하는데 있어서 우선 메모리 할당량이 중요하다고 생각했습니다. Init()이 한번만 호출된다면 처음 호출시에 모든 page에 대한 directory와 table이 연결되어야 한다고 생각했는데, demanding page의 개념에 의한 page\_fault 구현이라면 그럴 필요가 없다는 사실을 알고 시작했습니다.

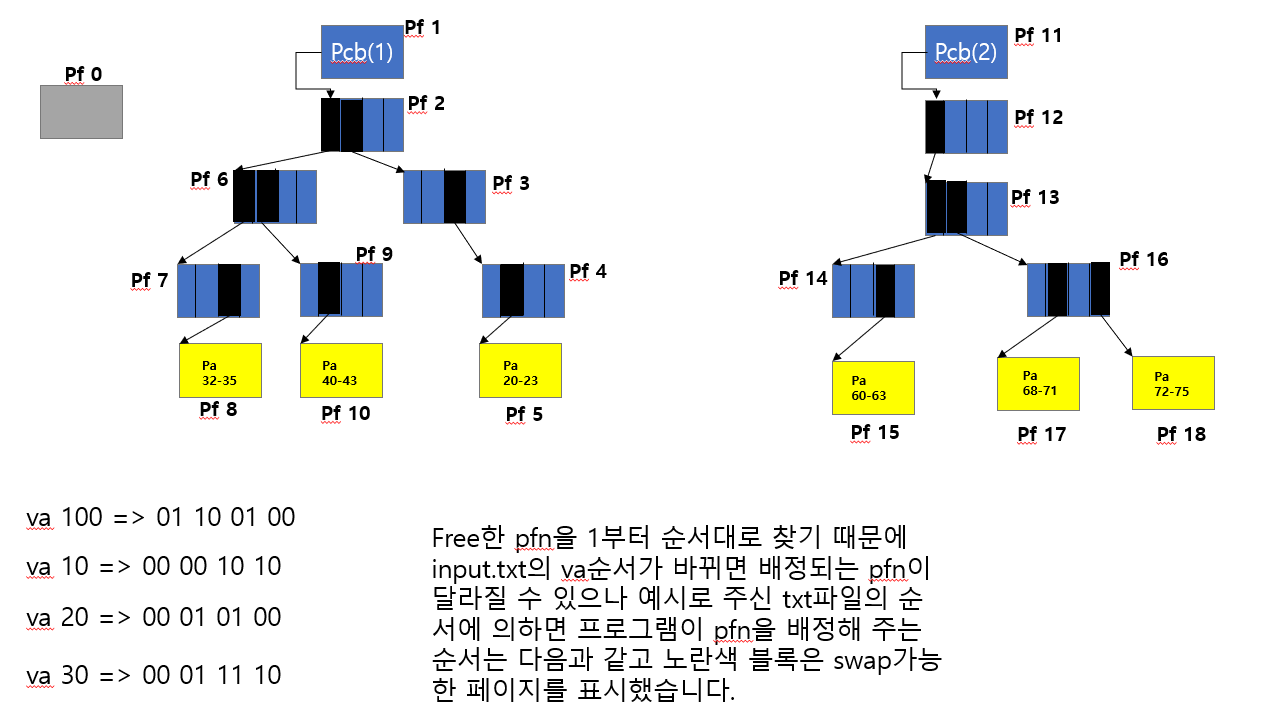
PDE구조를 파악하면서 valid에 집중했습니다. 메모리에 존재한다면 1 , 매핑이 안되어 있거나 swap-out상태라면 0 인데다가 최하위 비트를 차지하고 있다면 이는 짝/홀 수로 구분 할 수 있다고 생각했습니다.

**구현할 메모리는 바이트 단위로 다루는 것을 이용해 unsigned char 자료형을 사용 하였고 page-size가 4 바이트 이므로 메모리를 1차원 배열로 할당하고 index를 4칸씩 자르면 하나의 page-frame을 구성한다고 생각했습니다. 그렇기에 코드에 자주 보이는 \*4 연산은 page frame number(pfn)을 이용해 메모리의 위치를 찾기 위해 사용한 것들 입니다. 만약 pfn 1 이라면 메모리 상에서 4,5,6,7의 인덱스를 가지는 공간을 의미한다고 생각했습니다.**

선언한 전역변수는 free-list를 관리할 배열, pid를 통해 pdbr을 찾을 수 있는 배열, 그리고 FIFO를 구현할 배열을 선언했습니다. 각 프로세스는 pdbr을 pcb에 보관하고 물리메모리에 올려 놓지만 page\_fault에서 pid에 해당하는 pdbr이 메모리상에 어디에 있는지 알고 있어야 fault가 발생한 page를 연결시켜 줄 수 있다고 생각했습니다.

커맨드로 입력하는 pmem\_size를 4로 나눈 수가 maxpfn에 해당한다고 생각했고 그 크기만큼 free-list배열을 할당했고, 프로세스 별 pdbr을 저장할 배열은 프로세스 개수만큼이지만 더 많은 프로세스가 context switch하는 상황을 가정해 길이 30으로 지정했습니다. **FIFO를 구현할 배열은 page 사용에 대한 히스토리가 저장되어야 한다고 생각해서 100정도의 길이로 선언했습니다. 이 두가지 배열은 꽉차는 경우가 없다고 가정하고 진행하였습니다.**

설계 과정



왼쪽의 트리가 pid 1 의 경우이고 오른쪽의 트리가 pid 2일 경우 입니다.

1. Ku\_mmu\_init()

과제 구현 함수 입니다. 전달 받은 값 만큼 메모리를 할당하고 동시에 free-list를 관리할 배열도 할당하면서 두 배열 모두 모든 index에 0을 초기화 했습니다. 아직 어떠한 프로세스도 사용하지 않기 때문에 0 값을 택했습니다.

1. Ku\_run\_proc()

과제 구현 함수 입니다. Context switch를 담당하면서 ku\_cr3의 값을 알맞게 바꿔주는 역할을 하는데, 생성될 프로세스의 pcb, page directory를 위한 page를 할당받고 메모리에 올려 둡니다. 조건문을 삽입한 이유는 pdbr[pid] 값이 이미 존재한다면 새로운 페이지를 할당할 필요가 없기 때문입니다.

1. Ku\_page\_fault()

과제 구현 함수입니다. Pid와 va를 이용해 fault가 발생한 페이지를 찾습니다. 이를 위해 비트연산을 하여 주소 변환을 진행합니다. 3-level 구조 이므로 각 pde, pmde, pte를 계산하면서 이미 값이 존재한다면 새로운 페이지에 매핑할 필요가 없기 때문에 조건문을 삽입했습니다. 예를들어, 어떤 va가 찾는 pa에 대해서 다른 근처의 va가 이미 page\_dir , page\_middle\_dir을 생성했다면 page\_table만 생성해주면 되는 것입니다. Pte의 경우 앞서 말한 valid bit의 특성으로 인해 만약 0이 아닌 짝수의 값이 들어가있다면 swap\_out된 경우를 의미하기 때문에 따로 구현한 함수를 호출해 swap\_in을 해주었습니다.

1. **Int getfreePfn(int type)**

* **반환 값: 현재 사용가능한 pfn 값을 반환해 줍니다.**
* **인자 값: page를 type별로 구분하여 0,1,2 셋중 하나를 인자로 받습니다.**
* **내용 : type이 0이면 사용가능한 free-page, 1이면 swap가능한 매핑된 page, 2이면 swap불가능한 매핑된 page로 구분하고, pfn을 1부터 maxpfn까지 돌면서 free-page를 찾아 반환해 줍니다. 하지만 free-page를 찾지 못했을 경우 swapping()을 호출하게 됩니다.**

1. Unsigned char getPte(int k)

-반환 값: PTE의 형식으로 변환된 값

-인자 값: pfn

-내용: PTE를 구성하고 있는 구조는 사우이 6비트가 pfn이고 하위 2비트는 00 혹은 01 을 가지게 됩니다. 하지만 PTE가 생성되었다는 것은 매핑이 되었다는 것을 의미하므로 01을 가지게 됩니다. 따라서 pfn을 2만큼 bit shift 시키고( 4 곱하기) 1을 더해줍니다.

1. Int pteTopfn(unsigned char k)

-반환 값: pfn

-인자 값: PTE의 형식을 가진 값

- 내용: 5번 함수의 연산 과정을 반대로 수행하는 함수입니다. (PTE-1)/4

1. Int swapping()

-반환 값: FIFO알고리즘에 의해 선택된 pfn

-인자 값: void

-내용: FIFO알고리즘을 수행해 swap대상으로 선택된 pfn을 가지고 swap\_out을 호출해 줍니다.

1. Void swap\_out(int pfn)

-반환 값: void

-인자 값: eviction이 될 pfn

-내용: eviction이 되어야 하는 page의 swap\_out을 진행합니다. 메모리상에서 해당 pfn의 정보를 swap-space로 옮기고 메모리를 0으로 바꿔주고 free-page를 관리하는 배열상에서 pfn에 해당하는 값을 0으로 바꿔 free한 상태로 바꿉니다. 하지만 FIFO알고리즘이 0을 인자로 전달해 주게 되면 eviction이 될수있는 페이지가 존재 하지 않는 다는 것을 의미하기 때문에 exit()으로 물리메모리 공간이 부족하다는 메시지를 출력하면서 종료합니다.

1. Void swap\_in(int target, int pmde, int pte\_index)

-반환 값: void

-인자 값: swap-space에서 가져와야할 page , 가져온 페이지를 연결해줄 pmde와 pte\_index

-내용: swap\_in을 호출했다는 것은 짝수의 PTE값을 가진 PTE를 접근하려 했다는 뜻이고 그러한 PTE의 구조는 6비트 pfn에 하위 2비트는 00 일것입니다. 따라서 PTE를 4로 나누게 되면 pfn을 얻을 수있는데 이것이 swap-space상의 swap\_out되었던 page를 가리키게 됩니다. 새로운 page하나를 getfreePfn을 통해 받아오고 해당 받아온 page를 사용해 구조를 연결 시켜 줍니다.

실행화면

텍스트이(가) 표시된 사진

자동 생성된 설명

Pmem이 충분할 경우 swap을 요청하지 않습니다. 위의 설계그림에서 알 수 있듯이 최소 필요한 page-frame의 개수는 19개입니다. 따라서 76 바이트 만큼의 물리메모리까지는 swap이 발생하지 않습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Swap이 한번 호출될 경우 맨 처음 swap가능한 page를 할당받았던 PA: 20(pfn 5)에 pid 2 va 30 이 할당되면서 PA: 22로 주소변환이 이루어 집니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Swap을 2번 호출하면서 pfn 5 , 8 이 스왑되고 pfn 5에는 pid:2의 pcb가 저장되기 때문에 pfn 8에 pid 2 va 10이 할당되면서 주소변환이 PA:34로 이루어 집니다. 다시 swap이 2번 호출되면서 pfn 10 ,8 이 스왑 됩니다. pid 2 va 20을 위해 page-table을 하나를 새로 할당하면서 pfn 10에 저장됩니다. 따라서 다음 swap이 가능한 pfn은 8번이 되고 pid 2 va 20에 pfn 8을 할당합니다. 그 후에 pid 2 va 30에 대해서도 pfn 8번이 할당되다가 pid 1 va 10을 접근하려고 하자 pfn 8번에 swap\_in 시킨 모습입니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

설계 그림에서 노란색을 제외한 pf의갯수는 13개 이고 최소한 하나의 노란색 블록은 물리 메모리상에 존재하여 실행되어야 하기 때문에 14개의 페이지를 만족하지 못하면 프로그램이 종료됩니다. 따라서 실행화면 같이 52 바이트만큼의 메모리를 할당하면 물리메모리 공간이 부족하여 종료된 모습입니다.