Describe how you implement page allocation, deallocation, fork, copy-on-write, and TLB

우선 page allocation, deallocation, fork, copy-on-write 이 4개를 먼저 구현했다.
우선 이 모든 과정을 하기 전에 해당 메모리 시스템의 구조를 잘 파악해야 한다. 특히 페이지 테이블을 잘 파악해야 한다. 여기서는 2-level 페이지 테이블을 사용하였다. (Outer layer, inner layer) 과제에 있는 변수명으로 설명하면 첫 번째 페이지 테이블에서 pdes directory, 두 번째 페이지 테이블에서 ptes을 가지고 있다. 이는 모두 같은 크기, 즉 하나의 페이지 단위로 관리된다. 그 말인즉 4개의 pdes directory 와 16개의 ptes의 크기는 같다는 것이다. 하나의 프로세스, 하나의 페이지 테이블 당 최대 64개의 ptes을 처리해줄 수 있다는 뜻이다. Vpn 을 이용해서 directory, ptes 찾는 법은 다음과 같다. Pdes directory는 4개이므로, 이는 해당 vpn을 16으로 나눈 몫과 같다. (총 4개니까) Ptes는 해당 vpn을 16과 나눈 나머지이다. 실제 physical memory는 128개의 페이지로 이루어져 있다. 프로세스 당 하나의 페이지 테이블이 존재하며, 이 페이지 테이블은 ptbr로 관리한다. 또한 프로세스가 바뀌면 페이지 테이블을 flush 해줘야 한다.

우선 alloc은 그냥 머리 속에 들어오는 순서대로 짜주면 된다. 주의할 점은 페이지 테이블을 동적으로 만들어줘야 한다는 점이다. 해당 vpn이 실제 메모리의 프레임으로 매칭 되려면, 실제 128개의 프레임에 대해 빈 것에 매칭시켜주면 된다. 이는 mapcount을 활용해서 찾으면 된다. 파라미터로 들어온 값을 pte에 넣어주고, valid로 바꿔준다. 여기서 조심해야 하는 부분은 private에 rw을 저장해야 한다는 것이다. 이는 나중에 COW을 할 때 사용된다.

```
/// 일단 명령이 write 인데 해당 페이지는 read 권한만 있는데, private 이 read write 임
```

이는 내 코드의 주석이다. 이 상황을 처리하기 위함이다. 이때는 페이지 폴트로 새로운 프레임을 할당해주면 된다.

Free 부분 또한 굉장히 심플하다. 그냥 해당 페이지 테이블에서 해당 PTE을 찾은 뒤 초기화 해주고, 해당 VPN이 매칭되는 FRAME의 mapcount을 줄여주면 된다. 당연히 TLB도 이에 맞게 값을 업데이트 해줘야 한다

함수 description에서 알 수 있듯이, 총 3개의 경우에서 페이지 폴트 함수가 호출된다. 근데 0,1은 무슨 수를 쓰더라도 해당 오류를 고칠 수 없다. 허나, 세번째 경우, 즉 COW가 발생할 때는 페이지 폴트가 발생한다. 우선 COW가 발생하는 경우를 설명하자면 하나의 실제 프레임에 대해 여러 개의 VPN이 매칭될 때, 해당 프레임에 write 명령을 사용 할 수 없다. 왜냐면 여러 페이지가 참조하고 있기 때문에, 값이 바뀌면 당연히 이상할 것이다. 그러므로 write을 하면 오류가 발생하니까, 페이지 폴트가 불린다. 이때 private에 write기능이 있었다면, 해당 VPN은 새로운 프레임을 할당 받고, 기존의 프레임과의 연결을 끊는다. 이를 그냥 코드로 작성하면 페이지 폴트 함수를 풀 수 있다.

마지막은 스위치 프로세스입니다. 스위치 프로세스란 프로세스를 다른 프로세스로 바꾸는 것이다. 우선은 레디 큐(processes list)에 해당 pid번호를 가진 프로세스가 있는지 look up한다. 만약 있다면 해당 프로세스를 current와 바꾸고 tlb도 리셋 해야 한다. 수업시간에서 설명하실 때는 프로세스 번호를 파라미터로 가지고, 프로세스가 바뀔 때 PLB가 안 바뀐다고 하셨지만, 이번 과제에서는 PLB을 리셋 해줘야 했다. 만약 해당 프로세스가 레디 큐에 없다면, current을 fork 해줘야 한다. 당연히 여기서는 fork을 하면 pa3의 복사본이 생기니까 실제로 fork을 하면 안되고,

current의 복사본인 프로세스와 페이지 테이블을 만들면 된다. 이를 구현하는 방식은 새로운 프로세스를 malloc 해주고, current의 페이지 테이블을 그대로 복사하면 된다. 이때 새로 생기는 페이지 테이블들은 기존의 current의 페이지 테이블이 가리키는 프레임을 같이 가리키고 있다. 이 상황에서는 current와 새로 생긴 프로세스의 퍼미션을 access read로 바꿔줘야 한다. 그리고 해당 프레임의 mapcount을 증가시켜 주면 된다. 이를 코드로 구현하면 된다.

TLB 관리는 간단하다. 최근 사용된 vpn, pfn을 배열에 넣어서 기억하고 있으면 된다. 만약 해당 vpn이 있으면 이를 업데이트 해주면 된다. 이 파트는 그냥 간단해서 넘어가도록 하겠다. Insert 부분도 마찬가지다.

정말 이 순서대로 코드를 작성하였다. 그리고 alternative 대신에 이 디자인을 고른 이유를 설명하라고 하셨는데, 딱히 대안을 생각해 본적이 없고, 그냥 정말 논리적인 순서에 따라 코드를 작성하였고, 대부분 몇차례 안에 통과하였다. COW가 메인 파트였던 것 같은데, 이부분은 그냥 한 번에 통과해서 딱히 대안을 생각해 본적이 없다.

그럼에도 불구하고 실패를 한 부분이 있었다. 처음에는 페이지 테이블을 따로 동적으로 malloc해줘야 하는지 몰랐었다. 그래서 그냥 계속 페이지 테이블을 생성안하고 접근해서 세그먼트 오류가 계속 발생했었다. 생각을 해보면 페이지 테이블의 크기가 크기 때문에, 동적으로 필요할 때 생성해주는 게 메모리 관련 차원에서 효율적일 것 같다.

우선 처음에 VPN 0~3을 alloc 해주었다. 이렇게 되면 프로세스 0의 페이지 테이블에는 4개의 pte가 들어가게 된다. 이때 switch 1을 한다. 당연히 레디 큐에 프로세스 1은 없고, 새로 만들어준다. 이때 프로세스1 은 프로세스 0과 동일한 페이지 테이블을 갖는다. 실제 프레임 0~3은 이제 mapcount가 2이다. Vpn 0~3은 이제 read만 가능하고, Private에 read, write가 들어있다. 이 때 프로세스 1은 read1, write2 명령을 수행하는데, write2 에서 cow가 발생한다. 이제 프로세스 VPN(2)는 새로운 프레임을 할당 받고, rw을 원래대로, 즉 private값을 대입해준다. 그러므로 이제 실제 프레임은 0~4까지 할당 받은 상태이다.
다시 switch 2 명령어를 실행한다. 레디 큐에 없고 새로 프로세스, 페이지 테이블을 만들어주고 프로세스 1의 페이지 테이블을 복사한 뒤, rw을 read로, private을 세팅해준다. 하지면 여기서도 write 3에서 문제가 발생한다. VPN(3)에서 COW가 발생한다. 새로운 실제 프레임을 할당해준다. 이 경우에는 프레임 5를 할당해준다. 마지막으로는 다시 처음의 0번 프로세스로 전환한다. Read2m write2 에서는 문제가 발생하지 않는다. 왜냐면 프로세스 1 빼고 이를 참조하고 있는 프로세스는 0번 자기 자신 밖에 없기 때문이다. 그러나 VPN(3)에 대응하는 프레임은 프로세스 1과 공유 중이기 때문에, 새로운 프레임을 할당 받고, 이는 6이. 된다.
COW에 대한 설명은 위에서 했고
프레임은 순서대로 받는다고 리드미에 나와있다.

배운 점: 나름 재밌었고, 계층적 페이지 테이블을 좀 더 잘 이해하게 된 것 같다. 그러나 TLB와 같이 수업시간에 배운 내용들과 상반되는 내용이 나와 중간에 조금 돌아갔다.