가상메모리

가상 메모리 (Virtual Memory)

- 실제 물리 메모리 개념과 사용자의 논리 메모리 개념을 분리한 것이다.
- 메모리의 공간은 한정적이므로 사용자에게 더 많은 메모리를 제공하기 위해 가상 주소를 사용한다. 메모리 관리 장치는 가상 주소를 이용해 실제 데이터가 담겨 있는 주소로 변환해 준다.
- 여기서 가상 주소 공간은 하나의 프로세스가 메모리에 저장되는 논리적인 모습을 가상 메모리에 구현한 공간이며, 가상 주소는 해당 공간을 가리키는 주소이다.

가상 메모리가 필요한 이유

물리 메모리의 한계

- 모든 프로그램 코드를 물리 메모리에 올릴 수가 없다.
- 그렇다고, 프로그램을 교체하면서 올리면 메모리 교체 성능 문제가 발생한다.

가상 메모리의 장점

- 프로그램 용량이 실제 물리 메모리보다 커도 된다.
- 전체 프로그램이 물리 메모리에 올라와 있지 않아도 된다.
- 더 많은 프로그램을 동시에 실행할 수 있다.
    - 응답 시간은 유지
    - CPU 이용률과 처리율은 증가
- 즉 다중 프로그래밍을 실현하기 위해 물리 메모리의 제약을 보완하고 프로세스 전체를 메모리에 올리지 않고도 실행할 수 있도록 해 준다.

Swapping

- CPU 할당 시간이 끝난 프로세스의 메모리를 보조 기억 장치를 내보내고 (swap-out) 다른 프로세스의 메모리를 불러오는 (swap-in) 작업을 Swap이라고 한다.
- 이러한 Swap 작업에는 디스크 전송 시간이 들기 때문에 메모리 공간이 부족할 때 Swapping이 이루어 진다.

연속 메모리 관리

프로그램 전체가 하나의 커다란 공간에 연속적으로 할당되어야 한다.

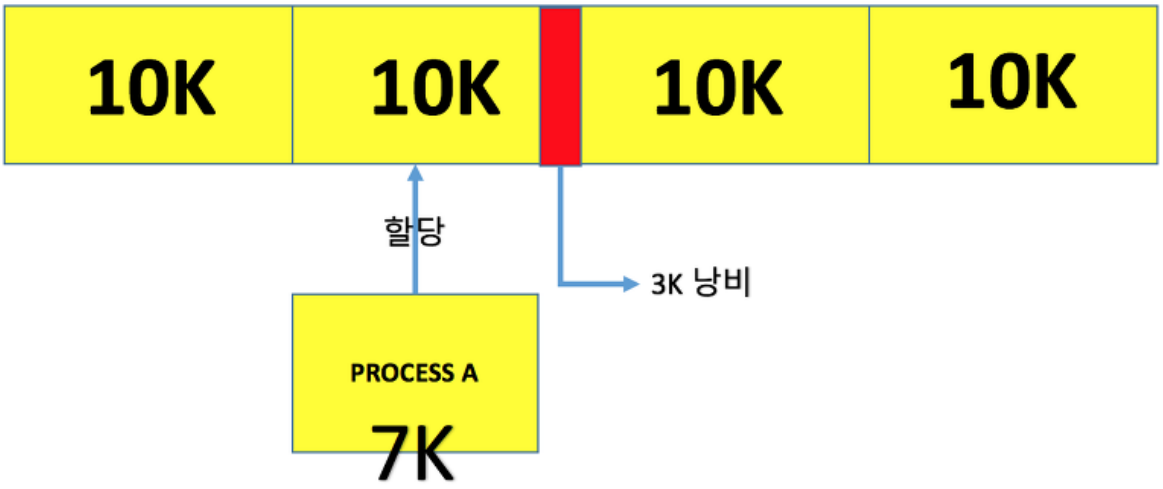
- 고정 분할 기법: 주 기억 장치가 고정된 파티션으로 분할 → 내부 단편화 발생
- 동적 분할 기법: 파티션들이 동적 생성되며 자신의 크기와 같은 파티션에 적재 → 외부 단편화 발생

이렇게 연속 메모리 관리 기법을 사용할 경우, 단편화 현상이 발생한다.

단편화 (Fragmentation)

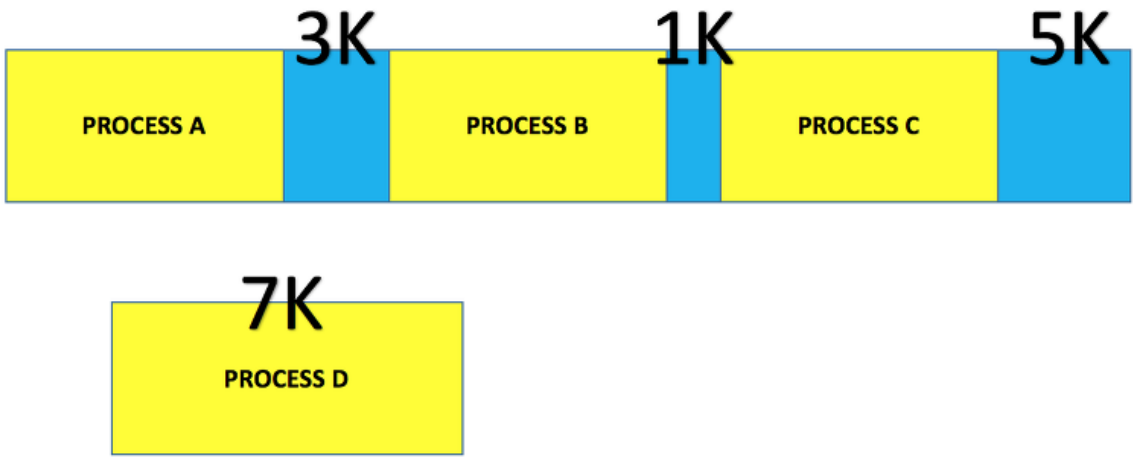
내부 단편화

- 프로세스가 사용하는 메모리 공간에 남는 부분
- 프로세스가 요청한 양보다 더 많은 메모리를 할당할 때 발생하며, 메모리 분할 자유 공간과 프로세스가 사용하는 공간의 크기 차이를 의미한다.



외부 단편화

- 메모리 공간 중 사용하지 못하게 되는 부분
- 메모리 할당 및 해제 작업의 반복으로 작은 메모리가 중간 중간 존재할 수 있다. 이렇게 사용하지 않는 메모리가 존재해서 총 메모리 공간은 충분하지만 실제로 할당할 수 없는 상황이다.
- 외부 단편화를 해결하기 위해 압축을 이용하여 프로세스가 사용하는 공간을 한쪽으로 몰 수 있지만, 작업 효율이 좋지는 않다.



불연속 메모리 관리

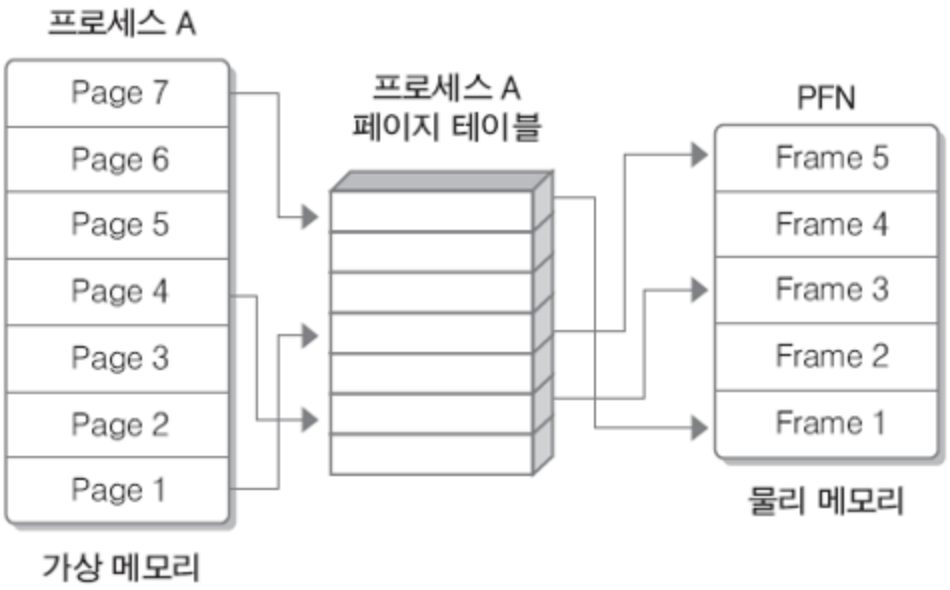
프로그램의 일부가 서로 다른 주소 공간에 할당될 수 있는 기법이다. 앞서 봤던 단편화 문제를 해결하기 위해 제시된 기법으로, 외부 단편화 해소를 위한 페이징과 내부 단편화 해소를 위한 세그멘테이션으로 나뉜다.

페이징 (Paging)

프로세스를 일정한 크기의 페이지로 분할해서 메모리에 적재하는 방식이다.

- 페이지: 고정 사이즈의 가상 메모리 내 프로세스 조각
- 프레임: 페이지 크기와 같은 주 기억 장치의 메모리 조각

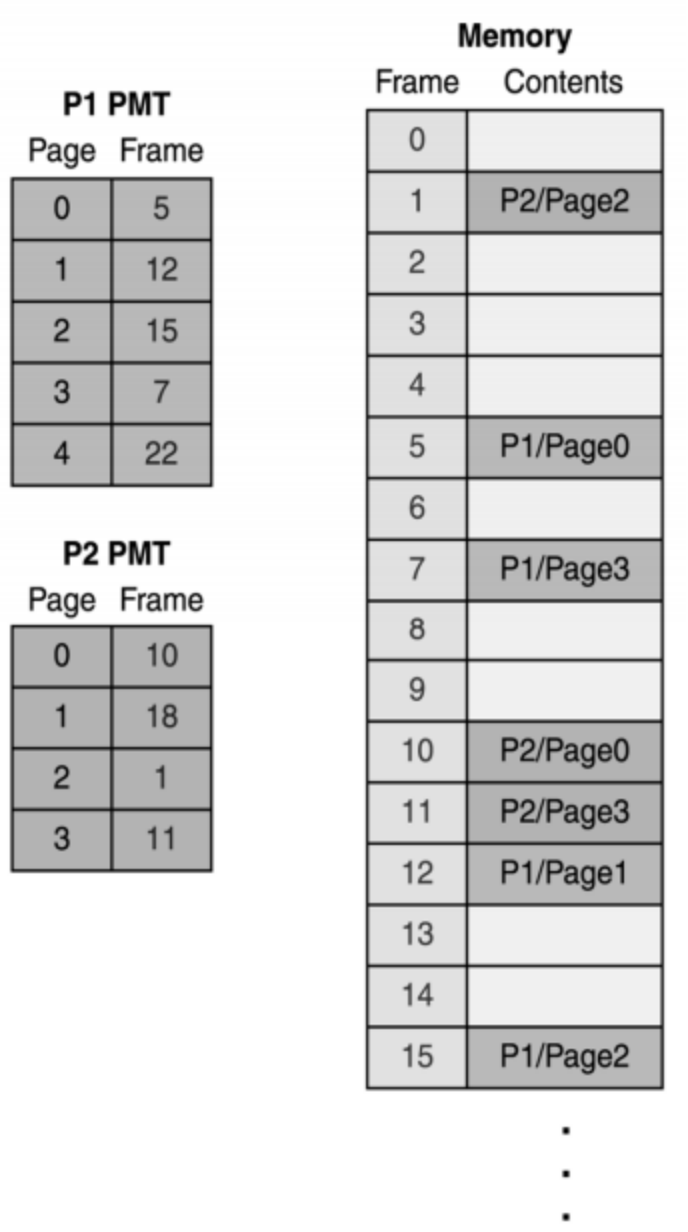
페이징 테이블 (Paging Table)



위 사진은 페이징 테이블의 매핑을 나타낸다. 앞서 봤듯이, 물리 메모리는 고정 크기의 프레임으로, 가상 메모리는 고정 크기의 페이지로 분리되어 있다. 개별 페이지는 순서에 상관 없이 물리 메모리에 있는 프레임에 매핑되어 저장된다.

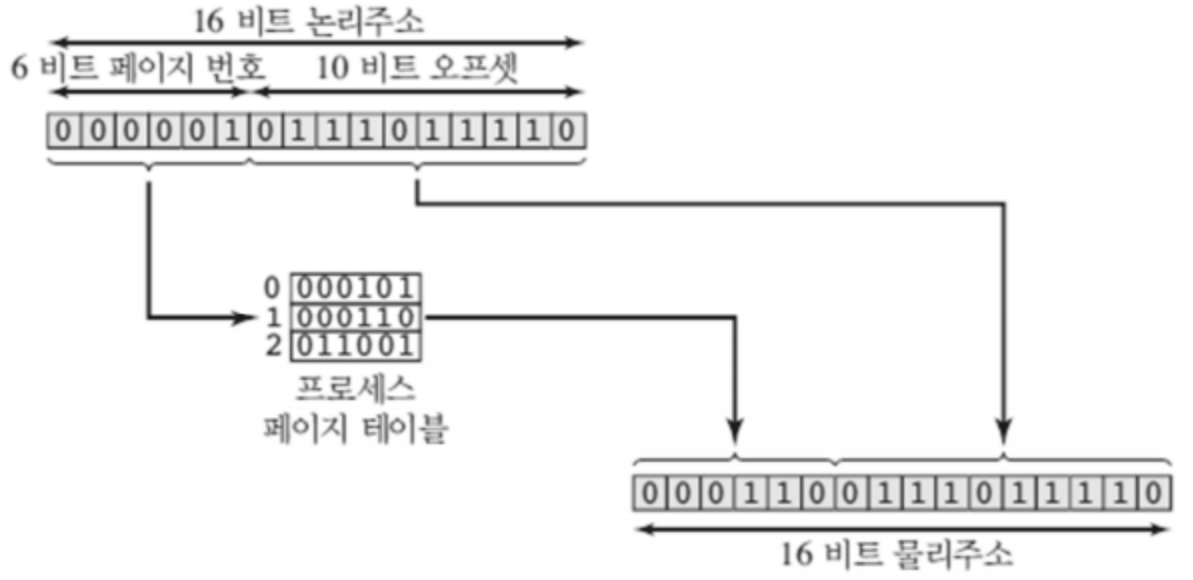
즉, 모든 프로세스는 하나의 페이징 테이블을 가지고 있으며, 여기에는 메인 메모리에 적재되어 있는 페이지 번호와 해당 페이지가 위치한 메인 메모리의 시작 주소가 있다. 이를 통해 하나의

프로세스를 나눈 가상 메모리 페이지들이 각각 실제 메인 메모리의 어디 프레임에 적재되어 있는지 알아낼 수 있다.

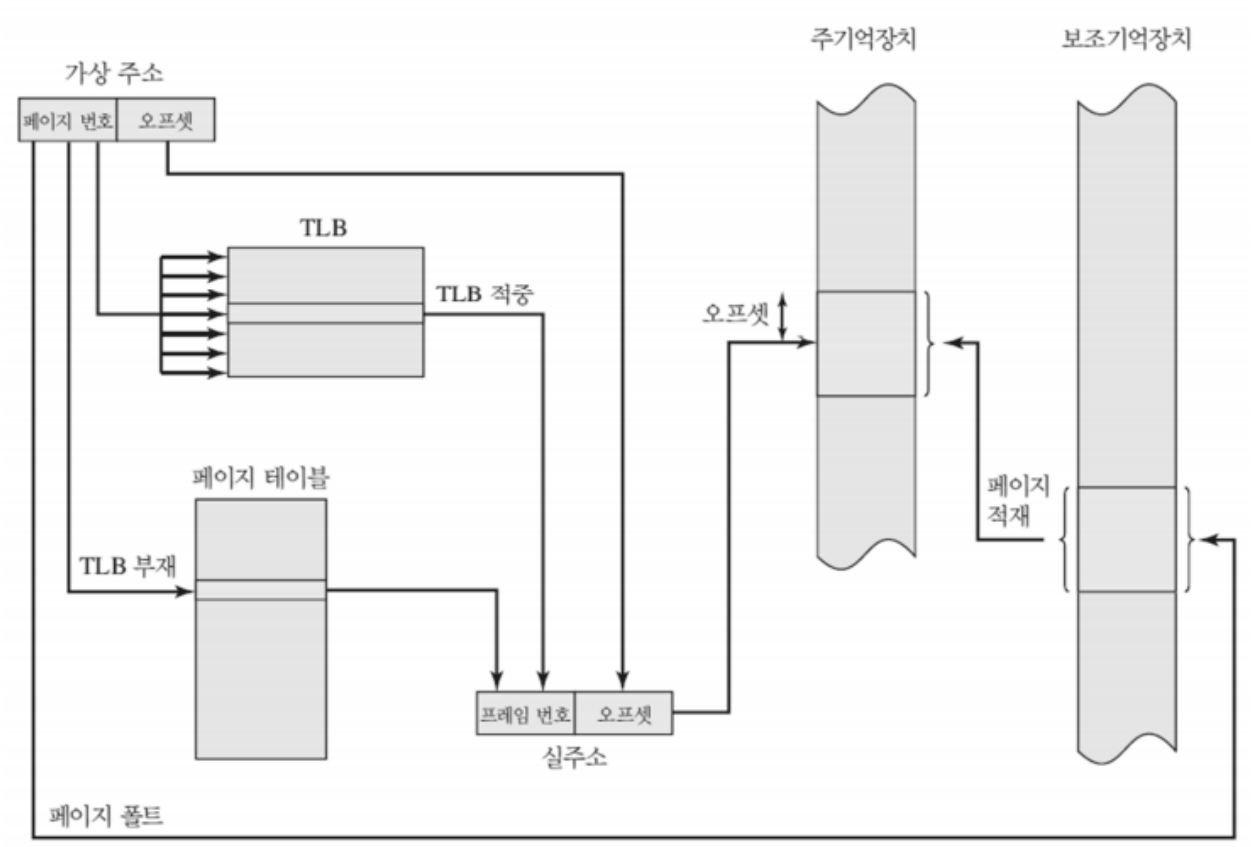


논리 주소와 페이지 테이블

앞서 메모리 관리 장치 (MMU, Memory Management Unit)는 가상 주소 (논리 주소)를 이용해 실제 데이터가 담겨 있는 주소로 변환해준다고 하였다.



논리 주소 (Logical Address)는 <page, offset>과 같은 형태로 구성되는데, 이를 이용해 물리 주소로 변환해 주는 것이다.



가상 주소를 물리 주소로 변환하는 전체 과정이다.

Page 교체 알고리즘

page fault가 발생하면, 요청된 page를 디스크에서 메모리로 가져옵니다. 이 때 물리적 메모리 공간이 부족한 상황이 발생할 수 있습니다.

이럴 때 메모리에 올라와 있는 page를 디스크로 옮겨서 메모리 공간을 확보해야 합니다. 이를 페이지 교체라고 하고, 어떤 page를 교체할지 결정하는 방법은 Page 교체 알고리즘에 따라 달라집니다.

이 교체 알고리즘은 최대한 page fault가 적게 일어나도록 도와줘야 합니다. 그래서 앞으로 사용될 일이 적은 page를 선택하여 교체하는 것이 성능을 향상시킬 수 있습니다.

알고리즘	설명
OPT(Optimal)	앞으로 가장 오랫동안 사용하지 않을 page를 찾아 교체한다.하지만 실제로 구현하기 거의 불가능한 알고리즘 이다.

FIFO(First in First Out)	메모리에 올라온지 가장 오래된 page를 교체하는 알고리즘.
LRU(Least Recently Used)	가장 오랫동안 사용하지 않은 page를 교체하는 알고리즘.
LFU(Least Frequently Used)	가장 참조횟수가 적은 page를 교체하는 알고리즘.

페이징의 장단점

장점

- 논리 메모리는 물리 메모리에 저장될 때 연속되어 저장될 필요가 없고, 물리 메모리의 남는 프레임에 적절히 배치되기 때문에 외부 단편화가 생기지 않는다.

단점

- 내부 단편화 문제가 발생할 수 있다. 페이지 단위를 작게하면 해결할 수 있지만, 페이지 매핑 과정이 복잡해져 오히려 비효율적이다.

세그멘테이션 (Segmentation)

세그먼트는 가상 메모리를 서로 크기가 다른 논리적 단위로 분할한 것을 의미한다. 세그멘테이션은 프로세스를 물리적 단위인 페이지가 아닌 논리적 단위인 세그먼트로 분할해서 메모리에 적재하는 방식이다.

돼지를 도축할 때, 페이징은 돼지를 같은 크기로 잘라서 보관하는 것이라면 세그멘테이션은 부위 별로 잘라서 보관한다고 이해하면 된다. 이렇듯 세그먼트는 의미가 같지 않는 논리적 내용을 기준으로 프로그램을 분할하기 때문에 크기가 같지 않다.

세그먼트 테이블

분할 방식을 제외하면, 페이징과 세그멘테이션이 동일하기 때문에 매핑 테이블의 동작 방식도 동일하다. 다만, 논리 주소의 앞 비트들은 페이징 번호가 아니라 세그먼트 번호가 될 것이다. 즉, <segment, offser> 형태로 구성되며, 세그먼트 번호를 통해 세그먼트의 기준 (세그먼트의 시작 물리 주소)와 한계 (세그먼트의 길이)를 파악할 수 있다.

세그멘테이션의 장단점

장점

- 내부 단편화 문제가 해소된다.
- 보호와 공유 기능을 수행할 수 있다. 프로그램의 중요한 부분과 중요하지 않은 부분을 분리하여 저장할 수 있고, 같은 코드 영역은 한 번에 저장할 수 있다.

단점

- 외부 단편화 문제가 생길 수 있다.