가상 메모리

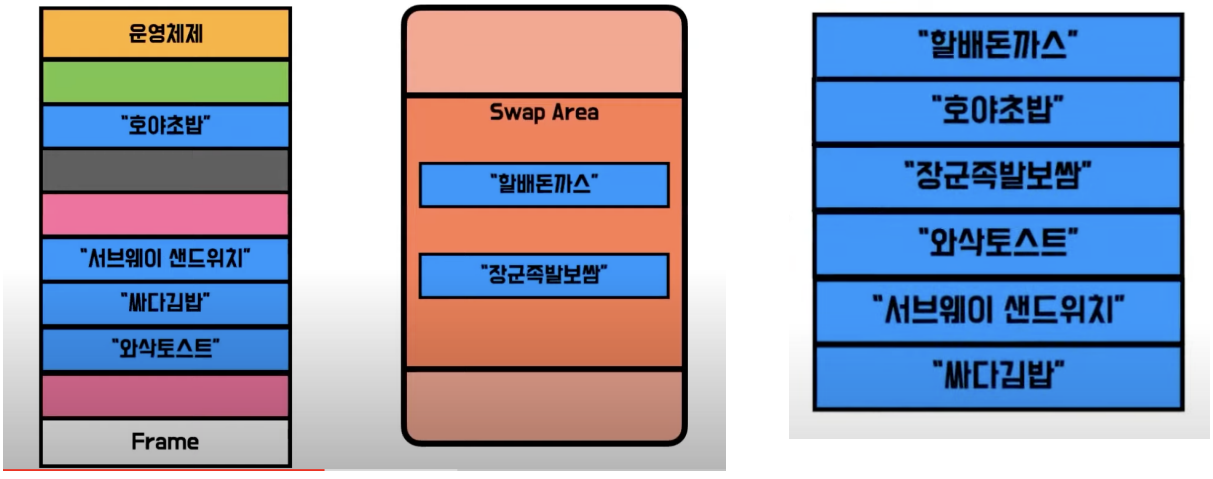
가상메모리란?

메모리 관리 기법중 하나로 프로세스 전체가 메모리 내에 올라오지 않더라도 실행이 가능하도록 하는 기법!

장점

1. 사용자 프로그램이 물리 메모리의 제약에서 벗어남
2. 각 프로그램이 차지하는 메모리가 적으므로 동시에 더 많은 프로그램 수행 가능
3. 프로그램을 메모리에 올리고 swap 하는데 필요한 IO 횟수가 줄어든다.

가상 메모리를 통한 메모리 관리 기법



- **가상 메모리로 사용자 프로세스 속이기**

실행중인 프로세스는 자신이 온전한 형태로 메모리에 올라가있다고 생각하기 때문에 에러를 내지 않고 정상적으로 작동한다. 그러나 실제 프로세스는 필요한 부분만 메모리에 있고 나머지 부분은 swap공간에 위치

메모리 공간과 swap공간에 있는 프로세스 조각을 전부 합쳐 만들어낸 가짜 메모리를 **가상 메모리**라 한다.

이를 통해 실제 메모리의 크기보다 더 많은 양의 메모리를 사용하는 것 처럼 구현을 할 수 있다!

CPU를 통해 요구하던 논리주소는 이 가상 메모리 주소.

- **Page fault**

프로세스가 CPU를 점유하고 작업을 하던 중 TLB와 메모리에 올라와있지 않은 page를 요구할 경우 (이를 **Page fault**라고 한다) swap 공간에서 page를 가져오고 TLB에 등록해야한다. 이 작업을 운영체제가 진행하며 디스크 I/O 작업이기 때문에 시간이 상당히 오래걸린다. Page fault확률이 곧 성능!!! → 다만 컴퓨터 프로그램 특성 상 같은 곳을 계속 참조하기 때문에, TLB를 참조하여 Page fault가 발생할 확률이 낮다.

- **Page Replacement Algorithm**

swap공간에서 page를 가져오다가 frame이 가득 찼을 경우 메모리의 page 하나를 쫓아내야한다. 이를 replacement라 하고 어떤 페이지를 replacement하는지는 운영체제가 결정한다. 이를 위한 여러 알고리즘이 제시되었다.

1. **OPT(Optimal Algorithm)**
    - Optimal Algorithm은 가장 먼 미래에 참조되는 page를 대체하는 방법.
    - 실제로 사용하지는 못하는 이상적인 알고리즘 이며, 다른 알고리즘의 성능에 대한 upper bound를 제공하는 역할을 하는 정도로 사용된다.
2. **FIFO(First In First Out) Algorithm**
    - 제일 먼저 들어온 페이지부터 먼저 내쫓는 방법
    - 미래를 모르는 경우에 모든 page가 평등하게 frame에 머무를 수 있다.
    - 장점: 구현하기 쉽다.
    - 단점: 어떤 page는 항상 필요할 수 있는데, 이러한 경우를 고려하지 않고 replace를 시키기 때문에 효율성이 좋다고 말할 수 없다.
3. **LRU (Least Recently Used) Algorithm**
    - 가장 오래전에 참조된 것을 지우는 방법
    - 연결리스트를 통해, O(1)의 속도로 탐색 및 삽입, 삭제를 진행할 수 있다.
    - 장점: 최적에 가장 가까운 알고리즘
    - 단점: 구현하기가 어렵고 접근 빈도를 고려하지 못한다는 단점이 있다.

4. **LFU(Least Frequently Used) Algorithm**
    - 가장 참조 횟수가 적은 페이지를 지우는 방법
    - 장점: LRU가 고려하지 못하는 빈도를 고려할 수 있다.
    - 단점: 참조 횟수를 전체 탐색해야하기 때문에 O(logN)의 시간 소요

LRU와 LFU를 실제 운영체제에 적용시킬 수 있을까?? → X

> LRU와 LFU의 알고리즘 구현에서 운영체제가 자료구조를 변경하고 유지하는 작업을 수행해야 하는데, 이미 메모리에 page가 올라가 있는 경우에는 CPU가 운영체제에 넘어가지 않는다. page fault가 발생해야 CPU가 운영체제에 넘어가고, 디스크에서 메모리로 page를 로드할 때 해당 page에 대한 정보를 얻고 갱신할 수 있다. 따라서 운영체제가 참조한 지 오래되거나 참조 횟수가 적은 페이지를 정확하게 알 수 없다.

5. Second Chance Algorithm (Clock Algorithm)

주요 알고리즘 및 장단점

FIFO (First In, First Out)작동원리: 가장 먼저 메모리에 들어온 페이지를 가장 먼저 내보냅니다.장점: 구현이 간단합니다.단점: 메모리에 오래된 페이지가 반드시 사용 빈도가 낮다는 보장이 없어, 효율성이 낮을 수 있습니다(부적합페이지 교체 문제).LRU (Least Recently Used)작동 원리: 가장 오랫동안 사용되지 않은 페이지를 교체합니다.장점: 실제 페이지 사용 패턴을 잘 반영합니다.단점: 구현이 복잡하며, 실행 시간에 영향을 줄 수 있습니다.Optimal Page Replacement작동 원리: 미래에 가장 오랫동안 사용되지 않을 페이지를 교체합니다.(이론적 모델로 실제 구현은 불가능)장점: 최소한의 페이지 폴트를 보장합니다.단점: 미래의 호출 정보를 알 수 없기 때문에 실제 시스템에서는 구현이 불가능합니다.

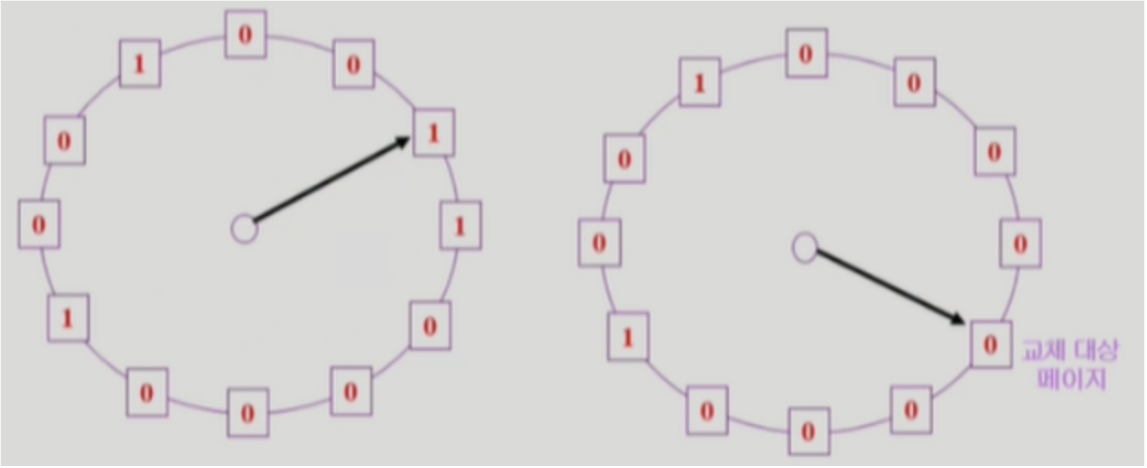
실제 적용

LRU 알고리즘: 웹 브라우저 캐시, Cache Memory 관리, 데이터베이스 관리 시스템 등에서 널리 사용됩니다. 메모리 접근 패턴이 시간에 따라 변하지 않는다는 가정 하에서 비교적 좋은 성능을 보입니다.FIFO 및 다른 알고리즘: FIFO는 그 단순성 때문에 초기 시스템에서 주로 사

용되었으나, 현재는 보다 성능이 좋은 알고리즘들이 개발되어 주로 사용됩니다. 그렇지만 단순한 임베디드 시스템이나 실시간 시스템에서는 여전히 FIFO나 개선된 변형 알고리즘이 사용되곤 합니다.

각 페이지 교체 알고리즘은 사용될 환경과 필요한 성능, 구현의 복잡성 등을 고려하여 선택됩니다.

- Reference bit가 0인 것을 찾을 때까지 시계처럼 한 바퀴씩 포인터를 이동하다가 0인 것을 찾으면 해당 page를 교체하는 방식
- 만약 Reference bit가 1인 page를 만나면 0으로 바꿔주고, 한 바퀴 되돌아와서도 (Second Chance) 여전히 0이면 해당 page를 교체한다. 다시 bit가 1로 바뀌어있다면 그만큼 자주 사용되는 page라는 의미인 것이다.



Trashing

메모리에 프로세스 종류가 많아질 수록, 각 프로세스가 사용할 수 있는 frame의 개수는 줄어들고 page가 적게 올라온 프로세스들은 해당 페이지만큼만 명령을 수행할 수 있고, 다음 작업을 수행하려다 page fault가 자주 발생하여 replacement를 요청하게 된다!

프로세스가 replacement를 진행하는 동안 다른 프로세스로 CPU 점유가 넘어가지만 그 프로세스 또한 replacement를 하게되는 상황 발생.

replacement를 하는 동안 CPU는 할 일이 없음. 운영체제는 이를 보고 프로세스의 개수를 늘림 → 악순환 반복

이 현상을 Trashing이라 한다.

운영체제는 Workin-set, page Frequeny 알고리즘 사용하여 해소

- working-set

대부분 프로세스가 일정 page만 집중적으로 이용한다는 성격 이용!

만약 frame 3개를 자주 이용할 때 최소 3개의 frame이 확보되어야 메모리에 이 프로세스를 올리고 replacement도 마찬가지로 3개의 단위로 page를 쫓아냄

- page Frequeny

pagefault 비율의 상한선 하한선을 두는 기법

상한선을 넘으면 프로세스마다 지급하는 Frame을 늘리고 (프로세스 개수를 줄임)

하한선보다 작으면 프로세스 마다 지급하는 Frame 개수를 줄임 (프로세스 개수를 늘림)