Virtual Memory

유튜브 주소: https://youtu.be/YMKK369d0rQ

IT융합공학부 이준희

<목차>

1. ‘가상 메모리’의 등장 배경

- **메모리 용량 제한**: 초기 컴퓨터의 물리 메모리는 매우 제한적이었다.
  이 때문에 큰 프로그램을 실행하거나, 여러 프로그램을 동시에 실행하는 것이 어려웠음.

- 가상 메모리가 등장한 이유는 많은 존재하지만,
  가장 큰 이유는 **메모리 용량의 한계**에 의해 제약이 따라왔던 것이다.

  이를 해결하기 위해 등장한 것이 **‘가상 메모리(Virtual memory)’** 이다.
  가상 메모리는 물리 메모리보다 훨씬 큰 가상 주소 공간을 프로그램에 제공함으로써,
  실제 메모리의 물리적 한계를 넘어서는 프로그램 실행과 메모리 관리의 유연성을 가능하게 해줌.

2. Virtual Memory

- **물리 메모리 영역**을 **보조저장장치까지 연장**한 것이 '가상 메모리' 이다.
- 주기억장치에 비해 보조기억장치는 가격이 훨씬 저렴하기 때문에
- 보조저장장치까지 메모리 영역을 확장하게 되면,
- 프로세스를 메모리에 적재하기 위해, 메모리의 크기를 신경 쓸 필요가 없다.

2. Virtual Memory

- 정확히는 디스크(보조저장기억장치)에 존재하는 프로그램의 실행을 위해
- 프로세스의 일부만을 메모리에 할당하는 것이다.

- 다중 프로그래밍을 하면서, 실제 물리 메모리가 꽉 차게 되면,
- 메모리의 일부분을 ‘스왑 영역’으로 옮겨 빈 영역을 확보한다.

- **스왑-아웃(swap-out)), 스왑-인(swap-in)**

물리 메모리에서 보조저장장치의 일부 영역인 스왑영역에 빼 내는 것을 ‘스왑 아웃(swap-out)’
반대로, 스왑영역에서 메모리에 적재하는 것을 ‘스왑 인(swap-in)’이라고 한다.

2. Virtual Memory

- 가상 메모리처럼 **일부만 메모리에 적재할 수 있다면?**

=> 프로세스의 크기가 **물리적 메모리의 크기에 의해 제약 받지 않을 수 있음**

- **다중 프로그래밍 정도**가 늘어남

⇒**Turnaround time(총처리 시간), Response time(응답시간)**이 늘어나지 않으면서,
CPU Utilization(이용률)과 **Throughput(처리량)** 향상

- 보다 빠른 실행이 가능

=> **swap을 위한 입출력이 줄어듦** (필요한 일부 page만 swap하면 되므로)

※ 가상 메모리 관련 문제들

- 가상 메모리의 '**페이지 테이블**'의 구성
- **페이지 폴트(page fault)**
- 페이지 할당
- **스왑 영역(swap space)**
- 프레임 할당
- **스레싱(Thrashing)**
- 작업 집합
- **페이지 교체 알고리즘 .. 등**

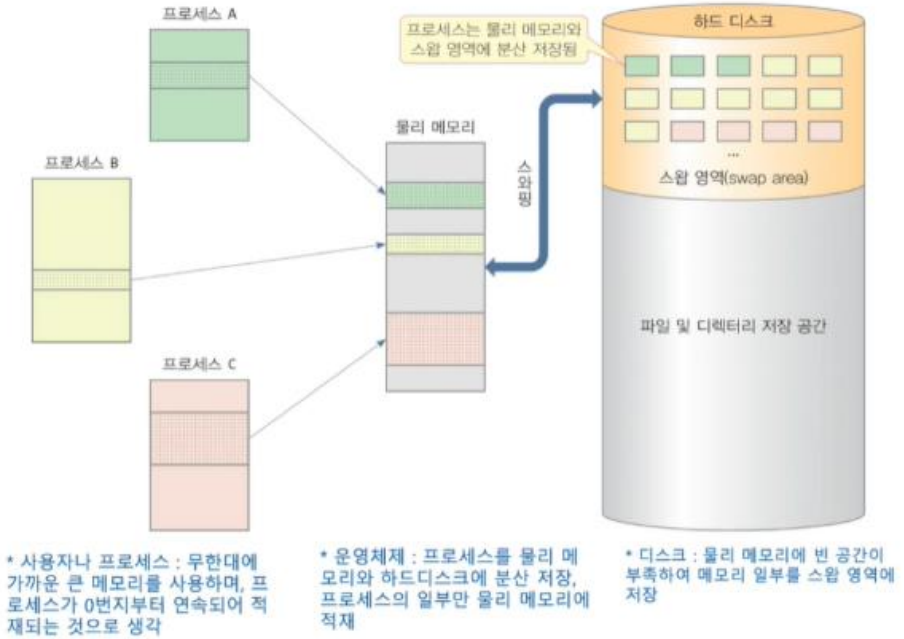
3. 요구 페이징(Demand Paging)

- **‘가상 메모리’**를 가능하게 해주는 기법

- **필요한 page들만 물리 메모리에 적재하는 방법**

=>

  - 실행중에 page들이 **실제 필요할 때에 적재**됨
  - 사용되지 않는 page는 적재되지 않음
  - 페이지를 갖는 **보조기억장치 영역**: ***실행파일 + 스왑***



4. 스왑 영역(Swap Space)

- 메모리가 부족할 때, 메모리를 비우고 페이지를 저장해두는
- **하드 디스크의 일부 영역**

- **스왑인(Swap – in)**

⇒ page-in이라고도 부르며, **스왑 영역에서 한 page를 메모리 frame으로 읽어들이 는 행위**

- **스왑 아웃(Swap – out)**

=> page-out이라고도 부르며,
메모리 frame에 저장된 page를 스왑 영역에 저장하는 행위

※ 가상 메모리에서의 **‘페이지 테이블’**

- **valid/presence bit**
  - 해당 page가 **물리 메모리에 있는지 여부**
    - **1** : 해당 page가 frame 번호의 **메모리에 있음**
    - **0** : 해당 page가 **디스크에 있음**
    - **MMU**는 실시간으로 페이지 테이블의 **valid/presence bit**를 확인하면서
      - **필요한 page가 물리 메모리에 적재되어 있는 지 확인**

- **modified bit(dirty bit)**
  - 해당 page가 **수정되었는지 여부**
    - **1** : 해당 page가 frame에 적재된 이후 **수정되었음**
      - 스왑 아웃될 때 **스왑 영역에 저장 필요**
    - **0** : 해당 page는 **수정된 적이 없음**
      - 스왑 아웃될 때 **스왑 영역에 저장 불필요 => 스와핑 시간 단축**

5. 페이지 부재(Page Fault)

- 메모리에 없는 page가 참조할 때 **페이징 하드웨어(MMU)가 OS에게 보내는 trap**

- **Page fault에 대한 처리 과정**
  1. 프로세스에 대한 **page table**을 통해 해당 참조가 유효한지 검사
  2. **Free frame**을 찾아 디스크로부터 page를 가져옴
  3. **Page table 갱신** (presence bit, frame number)
  4. Page fault에 의해 **중단된 instruction 재실행**

6. 스레싱(Thrashing)

- **프로세스가 실제 실행보다 더 많은 시간을 페이징 처리에 할당하고 있는 상태**

- 과도한 page 교체로 인한 **swap in/out에 몰두**

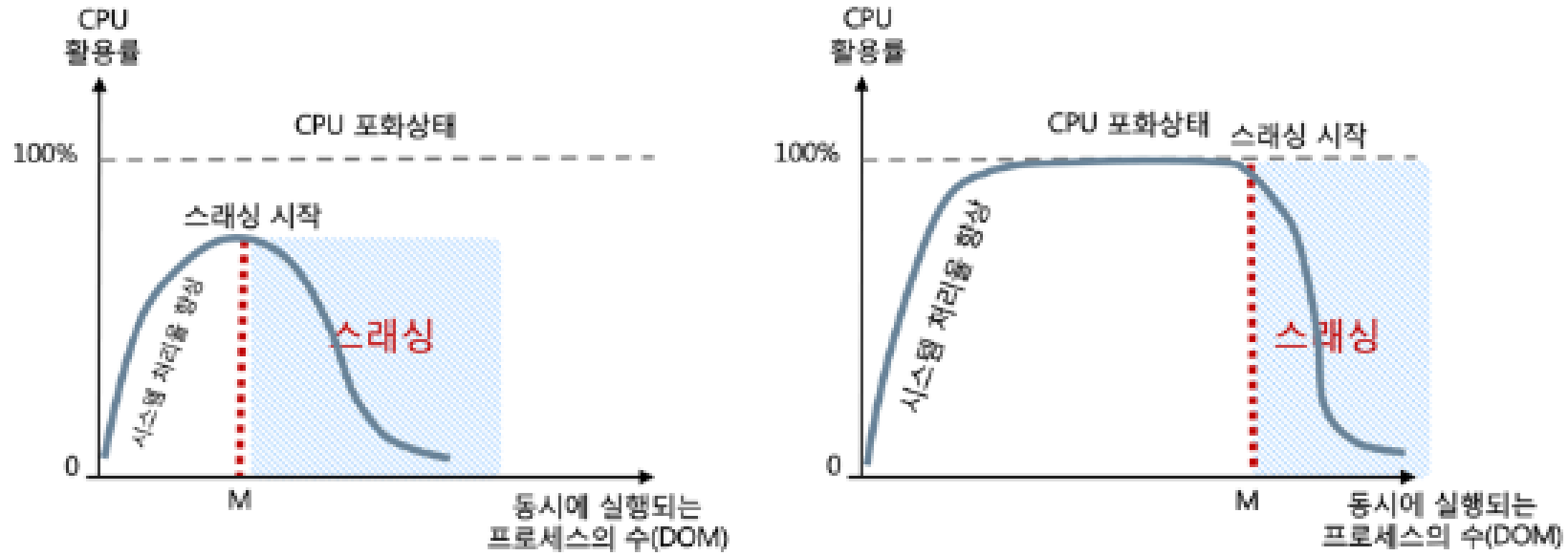
원인

- 프로세스가 **실행에 필요한 충분한 frame을 할당받지 못하면**
  page fault가 매우 높아짐

  - **=> CPU 이용률**이 낮아짐
  - => OS는 **다중 프로그래밍 정도**를 증가시키려 함
  - => **악순환의 반복으로 심각한 성능 저하 초래**

스레싱(Thrashing) 현상

- 다중 프로그래밍 정도가 높아질수록 **CPU 이용률 증가**
- 다중 프로그래밍 정도가 **임계점(M)**을 넘어가면 **스레싱 발생**



(a) 메모리가 작은 경우

(b) 메모리가 많은 경우

참고 : Denning의 1980년 논문 Working Sets Past and Present)

‘스레싱 현상’ **해결 및 예방**

- **다중 프로그래밍 정도 줄이기**
  - *일부 프로세스 종료*

- **빠른 보조저장장치의 사용** ex) SSD
  - 스와핑 시간 감축, CPU 대기시간 감축

- **물리 메모리 증가** ex) Ram 8GB => 16GB

Q & A