Chapter 9) 가상 메모리

(1) 요구 페이징

1) 요구 페이징의 개요

• 프로세스의 일부만 메모리로 가져오는 이유

- 메모리를 효율적으로 관리하기 위해서

- 용량이 큰 프로세스를 전부 메모리로 가져와 실행하면 응답이 늦어질 수 있으므로 필요한 모듈만 올려 실행한다.

• 포토샵과 power point 예

- 메모리에는 포토샵의 본프로그램만 올리고 필터는 사용자가 필요할 때마다 메모리로 가져오는 것이 효율적이다.

- power point는 하드디스크에 있는 것을 다 가져올 필요는 없고 본 프로그램만 메모리로 올리고 사용자가 필요한 부분이 있을 때마다 해당 부분을 메모리로 가져오는 것이 효율적이다.

• 요구 페이징(demand paging)

- 사용자가 요구할 때 해당 페이지를 메모리로 가져오는 것

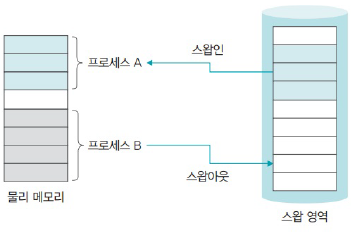
- 사용자가 특정 기능을 요구할 때 해당 모듈을 메모리에 올리면 메모리의 절약, 메모리 효율적 관리, 프로세스의 응답 속도 향상 등의 효과를 볼 수 있다.

• 미리 가져오기(pre-paging)

- 예상되는 페이지를 미리 가져오는 방식이다.

- 초기 발생하는 page fault를 방지하는 기법

-> page fault(페이지 부재): 프로세스가 페이지를 요청했을 때 그 페이지가 메모리에 없는 상황

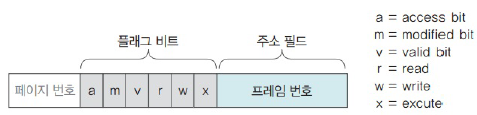
 - 예상되는 부분을 메모리에 가져다 놓아서 시스템의 성능을 향상되지만 미리 가져온 데이터가 사용하지 않는 경우 메모리 낭비가 된다.

2) 페이지 테이블 엔트리의 구조

• 페이지가 스왑 영역에 있는 경우 크게 2가지

- 요구 페이징으로 인해 처음부터 물리 메모리에 올라가지 못한 경우

- 메모리가 꽉 차서 스왑 영역으로 옮겨온 경우

 • 페이지 테이블 엔트리(PTE)의 구성

- 페이지 번호: 주소 변환 방식 중 직접 매핑에서는 필요 없지만,

연관 매핑에서는 페이지 번호와 프레임

번호 둘 다 필요하다.

- 프레임 번호: 가상 주소의 해당 페이지가 어느 프레임에

있는지 알려 주는 자료 구조이다.

메모리 관리자는 프레임 번호를 이용해

가상주소를 물리주소로 변환한다.

- 플래그 비트

\* 접근 비트(access bit): 페이지가 메모리에 올라온 후 사용한 적이 있는지 알려주는 비트

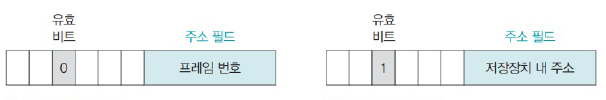
\* 변경 비트(modified bit): 페이지가 메모리에 올라온 후 데이터의 변경이 있었는지 알려주는 비트

\* 유효 비트(valid bit): 페이지가 실제 메모리에 있는지를 나타내는 비트 ex) 0: 물리 메모리 있음

1: 물리 메모리 없음

\* 읽기, 쓰기, 실행 비트: 읽기 비트(read bit), 쓰기 비트(write bit), 실행 비트(execute bit)는 페이지에 대한 읽기 권한, 쓰기 권한, 실행 권한을 나타내는 비트

3) 페이지 부재

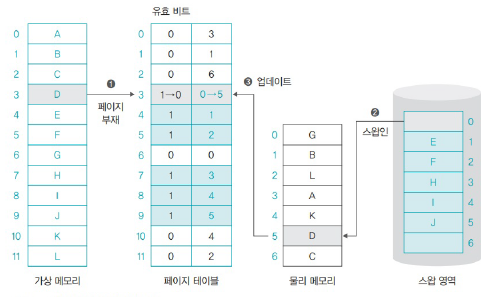
 • 유효 비트

- 유효 비트가 0일 때 페이지가 메모리에 있으므로 주소 필드에 물리 메모리의 프레임 번호가 저장

- 유효 비트가 1일 때 페이지가 스왑 영역에 있으므로, 주소 필드에 스왑 영역 내 페이지의 주소를 저장

• 페이지 부재(page fault)

- 프로세스가 페이지를 요청했을 때 그 페이지가 메모리에 없는 상황

 - 페이지 부재가 발생하면 프로세스가 해당 페이지를 사용할 수 있도록 스왑 영역에서 물리 메모리로 옮겨야 한다.

• 페이지 부재 처리 과정

① 프로세스가 페이지 3을 요청할 때 유효 비트가 1이라서

페이지 부재가 발생

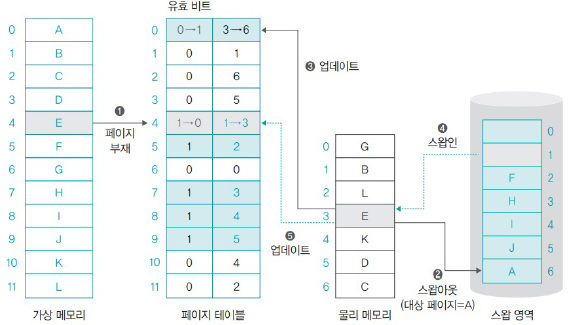
② 메모리 관리자는 스왑 영역의 0번에 있는 페이지를 메모리

의 비어 있는 프레임인 5로 가져온다. (스왑인)

③ PTE 3의 유효 비트를 1에서 0으로 바꾸고 주소 필드 값은 0에서 5로 바꾼 후, 프레임 5로 접근하여, 데이터를 프로세스에 넘긴다.

• 페이지 교체

- 페이지 부재가 발생하면 스왑 영역에 있는 페이지를 메모리의 빈 영역에 올리고 페이지 테이블을 갱신(업데이트)한다.

 - 빈 프레임이 있는 경우 작업이 수월하겠지만, 빈 프레임이 없는 경우 메모리에 있는 프레임 중 하나를 스왑 영역으로 내보낸 후에야 해당 페이지를 가져올 수 있다.

• 페이지 교체 알고리즘(page replacement algorithm)

- 어떤 페이지를 스왑 영역으로 내보낼지를 결정하는 알고리즘

• 대상 페이지(victim page)

- 페이지 교체 알고리즘에 의해 스왑 영역으로 보낼 페이지

• 메모리가 꽉 찬 상태에서 페이지 부재가 발생했을 때 조치하는 과정

① 페이지 4(프로세스 E)를 요청했지만, 해당 페이지의 유효 비트가 1이라서 페이지 부재가 발생함.

② 메모리가 꽉 찬 상태라서 victim page를 골라야 하는 데, 임의로 페이지 0(프로세스 A)을 victim page로 결정한다. 물리 프레임 주소가 3인 A 프로세스를 스왑 영역으로 내보낸다. (스왑 아웃)

③ 프로세스 A 테이블에 있는 유효 비트를 1로 변경하고 주소 필드 값에는 스왑 영역에 있는 주소 값으로 변경한다.

④ 스왑 영역에 있는 프로세스 E를 물리 메모리로 올라가는 데 프레임이 3으로 올라간다. (스왑 인)

⑤ 프로세스 E의 유효 비트를 1에서 0으로 바꾸고, 주소 필드 값을 스왑 주소 1에서 물리 프레임 주소 3으로 바꿔준다.

4) 지역성

• 페이지 교체 알고리즘이 쫓아낼 페이지를 찾을 때 지역성(locality)을 바탕으로 한다.

-> uniform하게 참조하지 않고, 내가 한 부분을 참조했을 때, 그 주변을 살펴본다.

-> 데이터를 살펴볼 때 페이지 혹은 프레임 단위로 보는 근거가 된다.

• 지역성은 기억장치에 접근하는 패턴이 메모리 전체에 고루 분포되는 것이 아니라 특정 영역에 집중되는 성질을 말한다.

• 지역성의 종류

- 공간의 지역성: 가까운 데이터에 접근할 확률이 먼 거리에 있는 데이터에 접근할 확률보다 높다.

- 시간의 지역성: 가장 가까운 시간에 접근한 데이터가 더 먼 시간에 접근한 데이터보다 사용될 확률이 높다.

- 순차적 지역성: 여러 작업이 순서대로 진행되는 경향이 있다는 것을 의미한다.

-> 공간 지역성 부분에 포함된다.

(2) 페이지 교체 알고리즘

1) 페이지 교체 알고리즘의 개요

• 페이지 교체 알고리즘은 스왑 영역으로 보낼 페이지를 결정하는 알고리즘

• 메모리에서 앞으로 사용할 가능성이 적은 페이지를 대상 페이지로 선정하여 페이지 부재를 줄이고 시스템의 성능을 향상한다.

 • 페이지 교체 알고리즘의 종류

• 페이지 교체 알고리즘의 성능평가

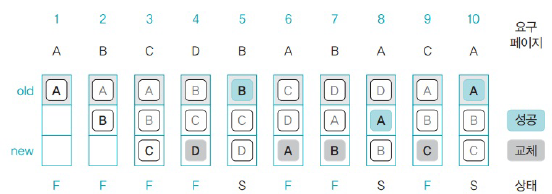
- 페이지 부재 횟수와 페이지 성공 횟수를 비교를 하는 것이다.

2) 무작위 페이지 교체 알고리즘(random page replacement algorithm)

• 스왑 영역으로 쫓아낼 대상 페이지를 특별한 로직 없이 무작위로 선정한다.

• 지역성을 고려하지 않기 때문에 알고리즘의 성능이 좋지 않아 거의 사용하지 않는다.

3) FIFO 페이지 교체 알고리즘(First In First Out page replacement algorithm)

 • 시간 상으로 메모리에 가장 먼저 들어온 페이지를 대상 페이지로 선정

• 메모리의 맨 위에 있는 페이지는 가장 오래된 페이지이고, 새로운 페이지는 항상 맨 아래에 삽입된다.

• FIFO 페이지 교체 알고리즘은 큐로 쉽게 구현할 수 있지만 먼저 들어온 페이지를 항상 스왑 영역으로 옮긴다. -> 시간의 지역성을 고려했기 때문

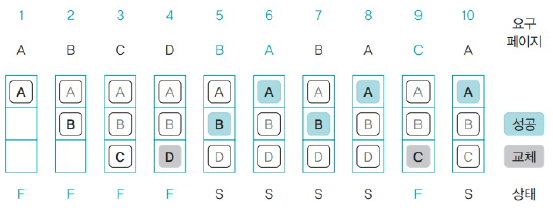
• 메모리에 올라온 시간만 고려하기 때문에 자주 사용되는 페이지가 스왑 영역으로 옮겨지는 문제가 발생

-> 오래 전에 실행한 페이지가 중요한 경우가 될 수 있어 무조건 victim으로 선정하는 것은 옳지 않다.

• 무조건 오래된 페이지를 대상 페이지로 선정하기 때문에 성능이 떨어진다.

• 그림 예제어서 페이지 성공 횟수가 3임을 알 수 있다.

4) 최적 페이지 교체 알고리즘(optimal page replacement algorithm)

 • 메모리가 앞으로 사용할 페이지를 미리 살펴보고 페이지 교체 선정 시점부터 사용 시점까지 가장 멀리 있는 페이지를 대상 페이지로 선정

• 위의 그림을 보면, 4번째 때 어떤 페이지를 스왑 영역으로 보내려고 할 때, A는 6번째에 사용 할 것이고, B는 7번째 사용될 것이고, C는 9번째 사용될 것이기 때문에 C를 스왑 영역으로 내보내고, 그 자리에 D를 넣어주고 있음을 알 수 있다.

• 최적 페이지 교체 알고리즘은 미래의 메모리 접근 패턴을 보고 대상 페이지를 결정하기 때문에 성능이 좋다.

• 미래 접근 패턴을 안다는 것은 불가능하기 때문에 실제로 구현할 수 없다. 최적 페이지 교체 알고리즘은 이상적인 방법이지만 실제로 구현할 수 없다.

• 최적 페이지 교체 알고리즘에 근접한 방법을 연구한 최근 최소 사용 알고리즘(LRU, Least Recently Used), 최소 빈도 사용 알고리즘(LFU, Least Frequently Used)

-> LRU 페이지 교체 알고리즘: 페이지에 접근한 시간을 기준으로 대상 페이지를 선정한다. 즉, 가장 오래 전에 사용한 page를 victim으로 결정한다.

-> LFU 페이지 교체 알고리즘: 페이지가 사용된 횟수를 기준으로 대상 페이지를 선정한다. 즉, 빈번하게 사용하지 않는 page를 victim으로 선정

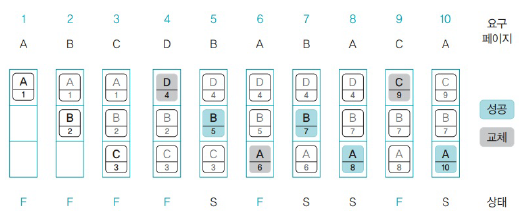
• 위의 그림을 보면 페이지 성공 횟수가 5이다.

5) LRU 페이지 교체 알고리즘

• ‘최근 최소 사용 페이지 교체 알고리즘’이라고 함.

• 메모리에 올라온 후 가장 오랫동안 사용되지 않은 페이지를 스왑 영역으로 옮긴다.

• LRU 페이지 교체 알고리즘은 시간을 기준으로 구현할 수 있으며 카운터나 참조 비트를 이용하는 방법도 있다.

 -> 시간을 page마다 저장하는 것은 오버헤드가 발생하는 것이므로, 카운터나 참조비트를 이용하는 것이다.

• 페이지 접근 시간에 기반한 구현

- 카운터가 제일 낮은 것이 오래 전에

사용한 것이다.

• LRU 페이지 교체 알고리즘의 그림을

보면, 성공횟수가 4이다.

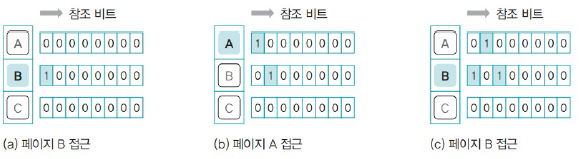
• 카운터에 기반한 구현

- LRU 페이지 교체 알고리즘을 카운터를 사용하여 구현할 수 있지만 추가적인 메모리 공간을 필요하는 것이 단점이다.

• 참조 비트 시프트 방식

- 각 페이지에 일정 크기의 참조 비트를 만들어 사용하는 것

- 참조 비트는 주기적으로 오른쪽으로 한 칸씩 이동한다.(shift)

 - 대상 페이지를 선정할 필요가 있으면 참조 비트 중 가장 작은 값을 대상 페이지로 선정한다.

 - 최근에 사용할 수 있도록 왼쪽에 bit를 있게 된다.

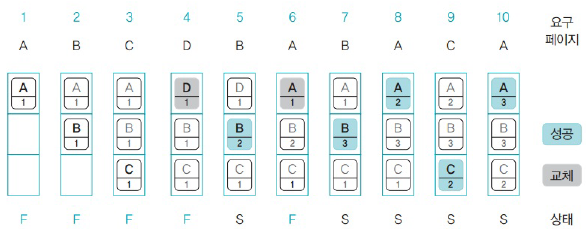
- B > A > C 이기 때문에 C가 victim이 된다.

6) LFU 페이지 알고리즘

• LFU 페이지 교체 알고리즘은 페이지가 몇 번 사용되었는지를 기준으로 대상 페이지를 선정한다.

• 사용된 횟수를 세어 횟수가 가장 적은 페이지를 스왑 영역으로 옮긴다.

• 카운터가 같게 나올 때 FIFO을 기준으로 할 지를 기준을 정해준다.



• LFU 페이지 교체 알고리즘의 그림을 보면 페이지 성공 횟수가 5이다.

-> 일반적으로 LRU 페이지 교체 알고리즘과 LFU 페이지 교체 알고리즘의 성능은 비슷하며, LRU, LFU 페이지 교체 알고리즘은 둘 다 FIFO 페이지 교체 알고리즘보다 성능이 우수하다.

7) NUR 페이지 교체 알고리즘

• LRU, LFU 페이지 교체 알고리즘은 추가 공간이 필요하여, 메모리가 낭비되는 문제가 있어 이를 해결한 알고리즘이 NUR 페이지 교체 알고리즘이다.

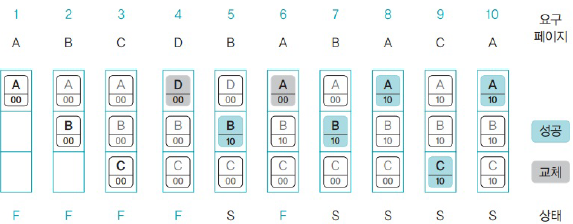
• NUR 페이지 교체 알고리즘에서 페이지마다 참조 비트와 변경 비트를 가진다.

-> 참조 비트: 페이지에 접근(read/execute)하면 1이 된다.

-> 변경 비트: 페이지가 변경(write/append)하면 1이 된다.

• 모든 페이지의 초기 상태는 (0, 0)이다. 모든 페이지 (1, 1)이 되면 초기화 한다.

-> (0, 0) 상태는 사용도지 않았기 때문에 victim으로 선정한다.

 -> 만약 (0, 0)인 페이지가 없다면 (0, 1)인 페이지를, (0, 1)인 페이지가 없다면 (1, 0)인 페이지를, (1, 0)인 페이지도 없다면 최종적으로 (1, 1)인 페이지를 스왑 영역으로 옮긴다.

• NUR 페이지 교체 알고리즘에서 우선 고려 대상은 참조 비트이다. 참조 비트가 0인 페이지를 먼저 찾고, 없으면 변경 비트가 0인 페이지를 찾는다. 만약 같은 비트의 페이지가 여러 개라면 무작위로 대상 페이지를 선정한다.

• 최적 근접 알고리즘인 LRU, LFU, NUR 페이지 교체 알고리즘의 성능은 거의 비슷하며 FIFO 페이지 교체 알고리즘보다 우수하다.

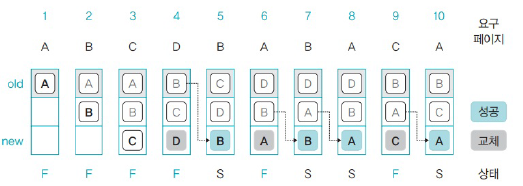
8) FIFO 변형 알고리즘

• 2차 기회 페이지 교체 알고리즘

- FIFO 페이지 교체 알고리즘과 마찬가지로 큐를 사용하지만 차이점은 특정 페이지에 접근하여 페이지 부재 없이 성공할 경우 해당 페이지를 큐의 맨 뒤로 이동하여 대상 페이지에서 제외

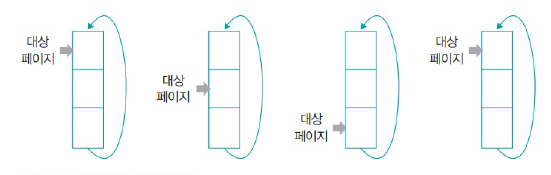
- 가장 오래된 페이지의 참조 비트를 확인

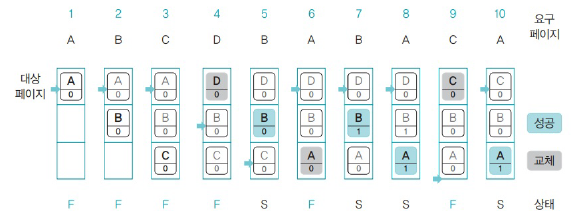
- 참조 비트가 0이면 즉시 교체 대상으로 선정하고 1이면 0으로 바꾸고, FIFO 큐의 맨 뒤로 보낸다.

- 그림에서 보면, FIFO 변형 알고리즘은 성공 횟수가 4이다.

- LRU, LFU, NUR 페이지 교체 알고리즘보다 약간 낮고 FIFO 페이지 교체 알고리즘보다 약간 높은 것으로 알려졌다.

• 시계 알고리즘(clock algorithm)

 - 스왑 영역으로 옮길 대상 페이지를 가리키는 포인터를 사용하는 데 이 포인터가 큐의 맨 바닥으로 내려가면 다음번에는 다시 큐의 처음을 가리키게 된다.

 - 포인터를 시계처럼 한 방향을 돌기 때문에 시계 알고리즘이라고 부른다.

- A 페이지를 대상 페이지로 가리킬 때, 그 자리에 D 페이지가 들어가고, 대상 페이지 포인터 위치를 B로 옮긴다.

- B를 한 번 더 가리켰으므로, 참조비트가 1을 넣어준다.

- B의 참조비트가 1이므로. 대상 페이지 포인터가 B를 건너 띠고, 참조 비트를 0으로 바꾼다.

-> 포인터가 가리키는 페이지는 교체 대상이고, 교체 대상의 참조 비트가 1이면 교체가 되지 않고 건너 띠게 되어 참조비트가 0이 된다.

(3) 스레싱과 프레임 할당

1) 스레싱(threshing)

• 하드디스크의 입출력이 너무 많아져서 잦은 페이지 부재로 작업이 멈춘 것 같은 상태

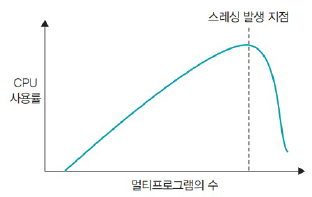
 CPU memory HDD

CPU를 효율적으로 사용하기 위해 다중

 프로그램을 함. 많이 하다 보니, 1 process당

할당되는 frame의 수가 적다. -> 프로그램을

실행하기 위해 메모리의 입출력이 빈번하게 발생하여 멈춰 보임.

 • 물리 메모리의 크기와 스레싱

- CPU가 작업하는 시간보다 스왑 영역으로 페이지를 보내고,

새로운 페이지를 메모리에 가져오는 작업이 빈번해져서 CPU가 작업

할 수 없는 상태 -> 스레싱 발생 시점(threshing point)

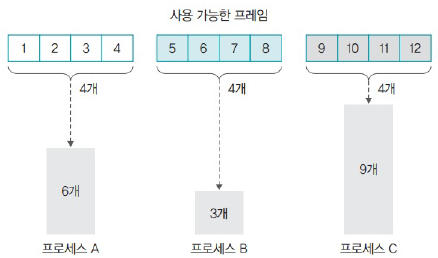
- 다중 프로그래밍 정도가 많아지면 한 프로세스 당 할당 프레임이 작아지면 페이지 부재율이 높아짐.

• 스레싱과 프레임 할당

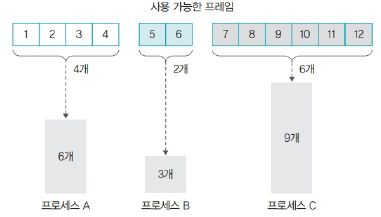
- 어떤 프로세스에는 너무 적은 프레임을 할당하여 페이지 부재가 빈번히 일어나고, 어떤 프로세스에는 너무 많은 프레임을 할당하여 페이지 부재를 줄이는 대신 메모리를 낭비한다.

- 프로세스에 프레임을 할당하는 방식은 크게 정적 할당과 동적 할당으로 나눈다.

2) 정적할당

 • 균등 할당(equal allocation) : 프로세스의 크기와 상관없이 사용 가능한 프레임을 모든 프로세스에 동일하게 할당한다.

- 크기가 큰 프로세스의 경우 필요한 만큼 프레임을 할당 받지 못하기 때문에 페이지 부재가 빈번하게 발생하고, 크기가 작은 프로세스의 경우 메모리가 낭비된다.

 • 비례 할당(proportional allocation): 프로세스의 크기에 비례하여 프레임을 할당하는 방식

- 균등할당보다 나은 방식

- 프로세스가 실행 중에 필요로 하는 프레임을 유동적으로 반영

하지 못한다. -> 작은 프로세스라도 필요 중에 메모리가 필요

한 경우도 있다.

- 사용하지 않을 메모리를 처음부터 미리 확보하여 공간을 낭비

한다.

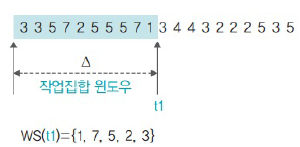
3) 동적 할당(dynamic allocation)

• 시시각각 변하는 요청을 수용하는 방식이 동적 할당이다.

• 작업집합 모델(working set model)

- 지역성 이론을 바탕으로 하며, 가장 최근에 접근한 프레임이 이후에도 또 참조될 가능성이 높다는 가정에서 출발했다. 최근 일정 시간 동안 참조된 페이지들을 집합으로 만들고, 이 집합에 있는 페이지들을 물리 메모리에 유지하여 프로세스의 실행을 돕니다.

- 작업집합 크기(working set size): 작업집합 모델에서 물리 메모리에 유지할 페이지의 크기

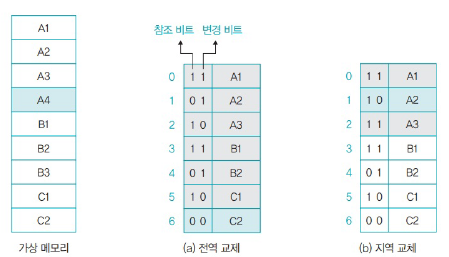
 - 작업집합 윈도우(working set window): 작업집합에 포함되는 페이지의 범위

윈도우를 움직여서 동적으로 물리메모리에 유지할 프로그램을 결정한다.

- 작업집합 윈도우를 너무 크게 잡으면 필요 없는 페이지가 메모리에 남아서 다른 프로세스에 영향을 미친다. 반대로 윈도우를 너무 작게 잡으면 필요한 페이지가 스왑 영역으로 옮겨져서 프로세스의 성능이 떨어진다.

• 페이지 부재 빈도(page fault)

- 프로세스가 필요로 하는 페이지의 양을 동적으로 결정하는 방법

 - 페이지 부재 비율이 상한선을 초과하면 할당한 프레임이 적다는 것을 의미하므로, 프레임을 추가하거나 늘린다. 반대로 페이지 부재 비율이 하한선 밑으로 내려가면 메모리 낭비된다는 의미로 할당한 프레임을 회수한다.

(4) 전역 교체와 지역 교체

• 전역 교체: 전체 프레임을 대상으로 교체 알고리즘을 적용

- 전체 시스템의 입장에서는 전역 교체 방식이 지역 교체 방식보다

효율적이다.

• 지역 교체: 현재 실행 중인 프로세스의 프레임을 대상으로 교체 알고리즘을 적용

- 지역 교체 방식의 장점은 자신에게 할당된 프레임의 전체 개수에 변화가 없기 때문에 페이지 교체가 다른 프로세스에 영향을 미치지 않는다.

- 지역 교체 방식의 단점은 자주 사용하는 페이지가 스왑 영역으로 옮겨져 시스템의 효율이 떨어진다.

Chapter10) 입출력 시스템과 저장장치

(1) 입출력 시스템

1) 입출력장치와 채널

• 다양한 주변장치는 데이터 전송 속도에 따라 저속 주변 장치와 고속 주변 장치로 구분할 수 있다.

• 하나의 버스로 주변 장치들을 묶으면 저속 주변 장치가 고속 주변 장치의 데이터 이동을 방해함.

• 채널: 데이터가 지나다니는 하나의 통로

• 채널 공유와 채널 분리

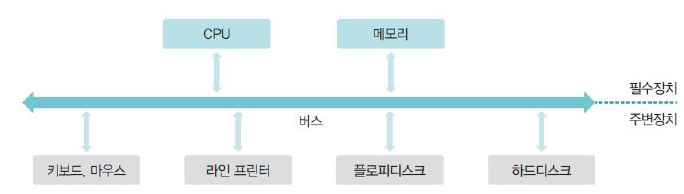
- 주변 장치를 하나의 채널 공유하게 되면 전체적으로 데이터 전송 속도가 느려진다.

- 전송 속도가 비슷한 장치끼리 묶어서 장치별로 채널을 할당하면 전체 데이터 전송 속도를 향상시킬 수 있다.

2) 입출력 버스의 구조

• 초기 버스 구조

- 모든 장치가 하나의 버스로 연결되고, CPU가 작업을 진행하다가 입출력 명령을 만나면 직접 입출력장치에서 데이터를 가져왔는 데 이를 폴링(polling)방식이라고 한다.

 - 버스를 장치에 상관이 없이 공유한다.

• 입출력 제어기를 사용하는 구조

- 메인 버스는 고속으로 작동하는 CPU와 메모리가 사용하고 입출력 버스는 주변장치가 사용

- 입출력 제어기를 사용하면 느린 입출력 장치로 인한 CPU와 메모리의 작업이 느려지는 것을 막을 수 있어 전체 작업 효율이 향상된다.

• 입출력 버스의 분리

- 입출력 제어기를 사용함으로써 작업 효율을 높일 수 있지만, 한편으로 저속 주변장치 때문에 고속 주변장치의 데이터 전송이 느려지는 문제가 있다. -> 고속 입출력 버스와 저속 입출력 버스로 분리 운영

- 주변 장치의 종류에 따라 저속을 선택할 지 고속을 선택할 지의 역할이 필요한 데 이를 채널 선택기(channel selector)가 관리한다.

- 입출력 버스로 감당하기 어려워진 그래픽 카드는 입출력 버스에서 분리하고 메인 버스에 연결되어 사용하며, 현대 컴퓨터에 들어와서는 그래픽 전용 버스를 만들어 그래픽 버스, 고속 입출력 버스, 저속 입출력 버스를 사용한다.

3) 직접 메모리 접근(direct memory access)

- 직접 메모리 접근(DMA)는 CPU 도움 없이도 메모리에 접근할 수 있도록 입출력 제어기에 부여된 권한

- 입출력 제어기는 여러 채널에 연결된 주변 장치로부터 전송된 데이터를 적절히 배분하여 하나의 데이터 흐름을 만든다. ->입출력을 CPU를 걸쳐서 하는 것은 비효율적이기 때문에 바로 I/O 장치에 access 해서 CPU를 덜 사용하는 것이 좋다.

- 채널 선택기는 여러 채널에 전송된 데이터 중 어떤 것을 메모리로 보낼지 결정한다.

• 입출력 장치와 상호작용하는 방법

- 입출력 데이터를 입출력 메모리라는 별도의 메모리에 보관

-> 장점: 메모리 영역을 침범하지 않는다.

-> 단점: 입출력 메모리 사용 시 별도의 명령어를 사용해야 한다.

- 메인 메모리의 공간 중 일부를 입출력 메모리 용도로 넣는다.

-> 장점: 별도의 명령어 필요 없다.

-> 단점: 입출력 메모리를 메모리에 넣으면 메모리 공간이 줄어든다.

4) 인터럽트

• 인터럽트: 주변 장치의 입출력 요구나 하드웨어의 이상 현상을 CPU에 알려주는 역할

• 컴퓨터 시스템의 각 장치마다 IRQ라는 고유 인터럽트 번호를 부여한다.

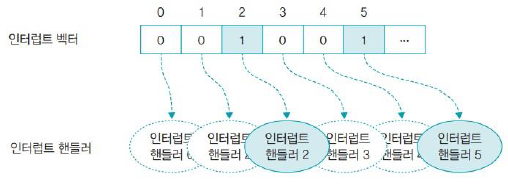
-> CPU는 인터럽트가 발생하라 때 IRQ를 보고 어떤 장치에서 인터럽트가 발생 했는 지를 파악

• 외부 인터럽트: 입출력 장치로부터 오는 인터럽트 뿐 아니라 전원 이상이나 기계적인 오류 때문에 발생하는 오류도 포함한다.

• 내부 인터럽트: 프로세스의 잘못이나 예상치 못한 문제 때문에 발생하는 인터럽트이므로 예외 상황(exception) 인터럽트이다.

• 시그널: 사용자의 의지로 발생시키는 자발적 인터럽트

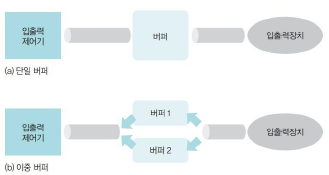
• 인터럽트 벡터: 여러 인터럽트 중 어떤 인터럽트가 발생했는지를 파악하기 위해 사용하는 자료 구조이다.

 -> 인터럽트 벡터 테이블에는 핸들러 주소가 들어있다.

• 인터럽트 핸들러: 인터럽트의 처리 방법을 함수 형태로

만듬. 운영체제는 인터럽트 발생하면

인터럽트 핸들러를 호출한다.

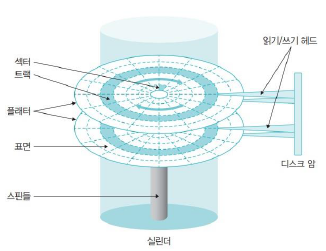


5) 버퍼링

• 버퍼: 속도가 다른 두 장치의 속도 차이를 완화하는 역할

• 이중 버퍼를 사용하면 한 버퍼는 데이터를 담는 용도로 쓰고 또 한 버퍼는 데이터를 가져오는 용도로 쓸 수 있어 유용한다.

• 하드웨어 안전 제거를 사용하면 버퍼가 다 차지 않아도 강제로 버퍼의 내용이 저장장치로 옮겨지는 데 이를 플러시(flush)라고 한다. 플러시가 일어나면 저장장치의 방지하기 위해 입출력 장치의 전원이 차단

(2) 디스크 장치

1) 디스크 장치의 종류

• 하드 디스크

- 플래터: 표면에 자성체가 발려 있어 자기를 이용하여 0과 1의

데이터를 저장

-> 보통 2장으로 구성되며 일정한 속도로 회전

- 섹터(sector): 하드디스크의 가장 작은 저장 단위

- 블록: 하드디스크와 컴퓨터 사이에 데이터를 전송하는 논리적인 저장 단위 중 가장 작은 단위

-> 블록은 여러 개의 섹터로 구성되며, 윈도우 운영체제에서는 블록 대신 클러스터(cluster) 표현

- 트랙(track): 플래터에서 회전축을 중심으로 데이터가 기록되는 동심원

- 실린더(cylinder): 개념적으로 여러 개의 플래터에 있는 같은 트랙의 집합

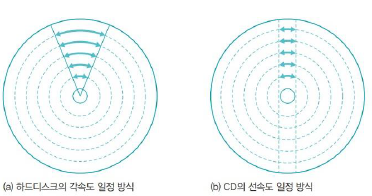
- 읽기/ 쓰기 헤드(read/write head): 하드디스크에서 데이터를 읽거나 쓸 때 사용

- 배드 섹터(bad sector): 플래터의 표면에 생긴 상처는 데이터를 저장할 수 없는 상태

• CD

- 하드디스크와 마찬가지로 트랙과 섹터로 구성

- CD는 표면에 미세한 홈이 파여 있어 헤드에서 발사된 레이저가 홈에 들어가 반사가 되지 않으면 0으로 반사되어 돌아오면 1로 인식한다.

 • 하드디스크와 CD의 비교

- 각속도 일정 회전: 하드디스크의 플래터는 항상 일정한 속도로 회전하여 바깥쪽 트랙의 속도가 안쪽 트랙의 속도보다 빠르다.

-> 장점: 디스크가 일정한 회전 속도로 회전하기 때문에 구동 장치가 단순하다.

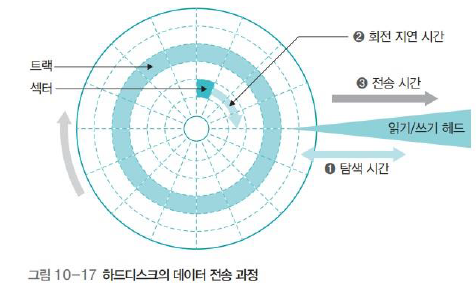
-> 단점: 안쪽 트랙에 비해 바깥쪽 트랙으로 갈수록 낭비되는 공간이 생긴다.

- 선속도 일정 회전: CD에서 사용하는 선속도 일정 방식의 경우 어느 트랙에서나 단위 시간당 디스크의 이동거리가 동일하다.

-> 장점: 한정된 공간에 많은 데이터를 담을 수 있고, 바깥쪽 트랙의 섹터 공간이 낭비되지 않는다.

-> 단점: 모터 제어가 복잡하여 소음이 발생한다.

2) 디스크 장치의 데이터 전송 시간



• 데이터 전송 시간 = 탐색 시간(seek time) + 회전 지연 시간(rotational latency time)

+ 전송 시간(transport time)

-> 탐색 시간의 비율이 크다.

3) 디스크 장치 관리

• 파티션: 논리적으로 분할하는 작업

- 대용량 하드디스크의 경우 하나로 사용하기보다 여러 개로 나누어 사용하면 관리하기가 편리한다.

• 포매팅(formatting): 디스크를 초기화 하는 작업

- 빠른 포매팅은 데이터를 그대로 둔 채 파일 테이블을 초기화하는 방식

- 느린 포매팅은 디스크에 파일 시스템을 탑재하고 디스크 표면을 초기화하는 방식