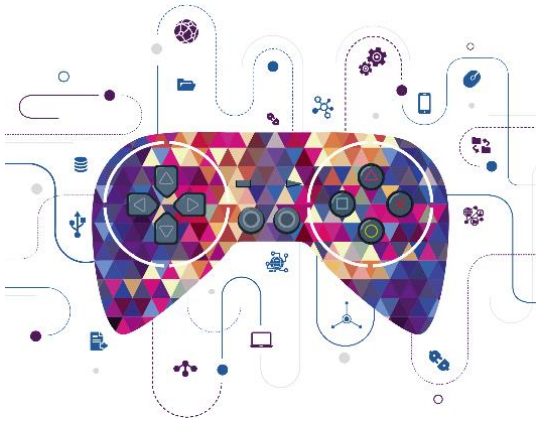


그림으로 배우는 구조와 원리

운영체제 개정 3판

Chatpter 09
입출력 시스템과 디스크 관리

- 입출력 시스템과 모듈을 이해한다.
- 디스크의 구조와 스케줄링을 살펴본다.
- RAID의 제안 방법을 알아본다.

Section 01 입출력 관리(1. 입출력 시스템과 입출력 모듈)

■ 입출력 시스템과 입출력 모듈의 개념

- 모니터나 프린터 같은 하드웨어 장치 뿐만 아니라 입출력 모듈까지 포함
- 물리적 입출력장치가 실제로 입출력을 수행하고, 입출력 모듈은 메모리나 프로세서, 레지스터 등 내부 저장장치와 물리적 입출력 장치 사이의 이진 정보를 전송 방법 제공
- 입출력 모듈이 프로세서를 대신하여 입출력과 관련된 복잡한 일을 처리하면 **입출력 채널** 또는 입출력 프로세서가 되고, 단순히 프로세서의 입출력과 관련된 일을 담당하면 **입출력 제어기** 또는 장치 제어기가 됨

입출력장치를 직접 시스템 버스에 연결하지 않고 입출력 모듈을 이용하여 연결하는 이유는 입출력장치에 따라 제어하고 운용하는 방법이 다르기 때문

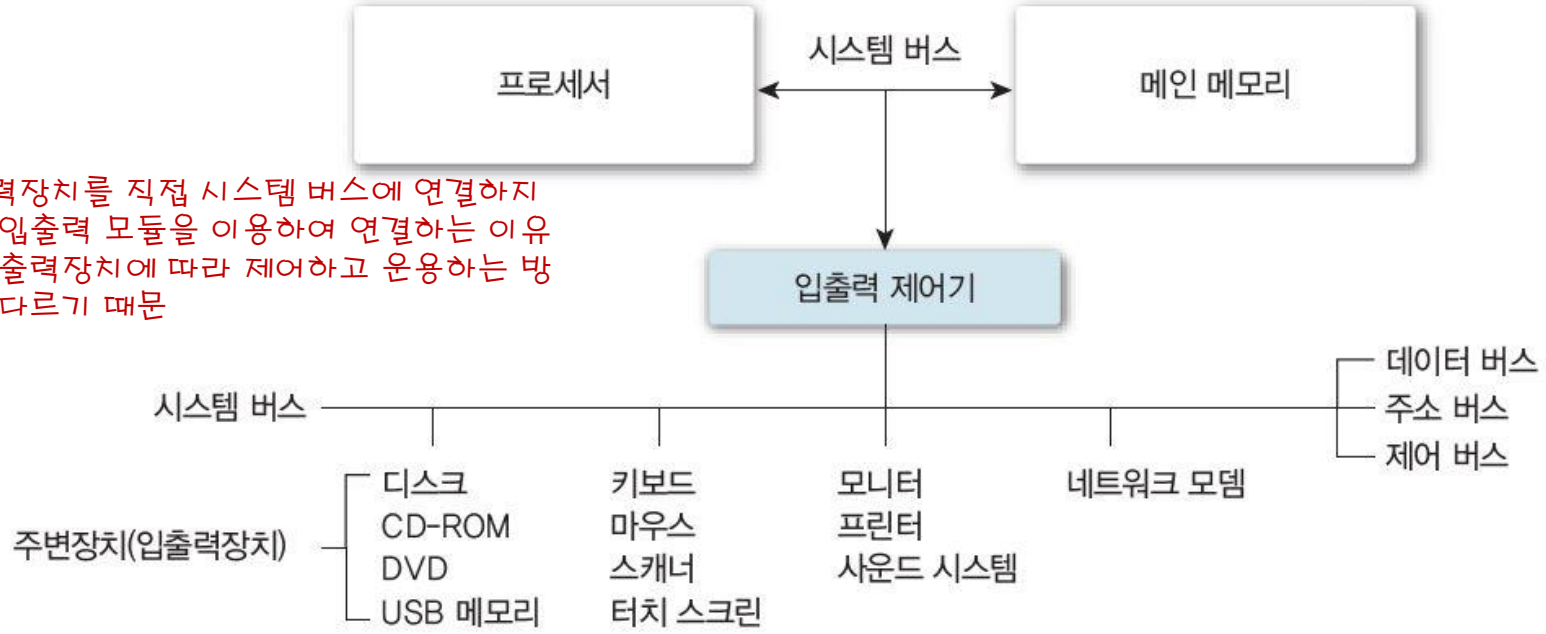


그림 9-1 **컴퓨터 하드웨어 구성과 입출력 모듈**

2. 입출력 모듈의 구성

입출력 모듈의 구성

- 입출력장치의 수와 구조에 따라 다양하게 구성 가능
- 일반적인 구성

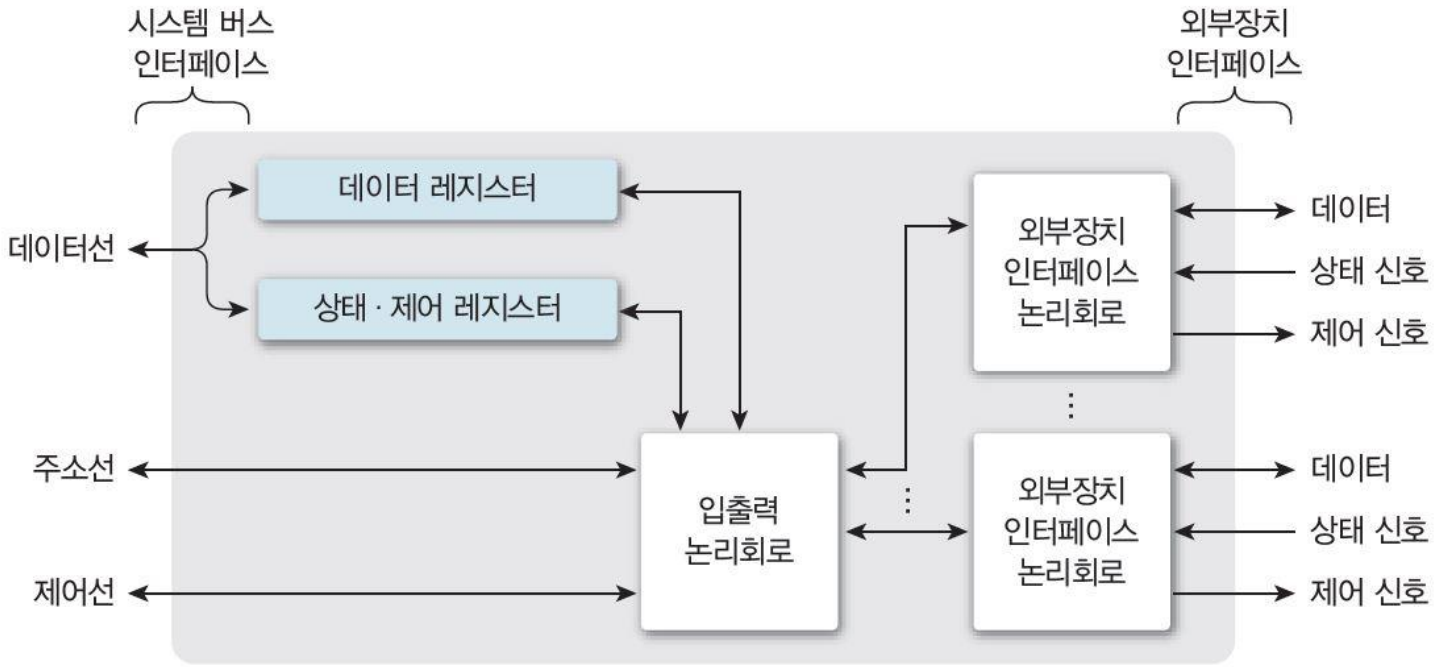


그림 9-2 입출력 모듈의 내부 조직

입출력 모듈로 들어가거나 나오는 데이터는 **데이터 레지스터**에 일시 저장
현재의 상태 정보를 저장하는 **상태 레지스터는 제어 레지스터**로 동작
상태 레지스터에는 프로세서의 명령에 따라 입출력 모듈을 제어하는 장치와 인터페이스를 제어하는 입출력 논리회로가 있음
입출력 논리회로는 상태 레지스터가 제어하는 장치의 주소를 인식하고 주소도 발생 할 수 있어야 하므로 연결된 각 장치와 인터페이스를 제어하려고 장치 인터페이스를 포함한다, 그리고 제어 버스로 프로세서와 교신

3. 입출력 모듈의 기능

입출력 모듈의 기능

- 내부 자원과 데이터 입·출력등 다양한 동작을 제어하고 타이밍 기능을 제공
- **입출력 모듈**이 외부장치의 타이밍과 데이터 형식, 기계적인 세부 사항들을 처리하므로 **프로세서**는 단순히 파일 열기나 닫기 명령만으로도 장치를 제어 가능
- 프로세서가 입출력장치에 명령을 보내려면 입출력 장치의 식별자를 주소로 지정 해야 함
- 주소를 지정하는 방법
  - 메모리 주소 공간과 입출력 주소 공간이 별도로 있는 전용 입출력 주소 방법
  - 메모리 주소 공간의 일부를 입출력 주소 공간으로 공유하는 메모리 매핑 주소 방법
    - 메모리 매핑 주소 방법은 임의의 기계 명령과 주소를 입출력장치를 지정하는데 사용하는등 다양한 유연성, 데이터 전송 완료를 통보하려고 인터럽트 신호 생성
- 프로세서에서 명령을 전달받고, 관련된 메시지를 인식하는 기능 제공
- 입출력 모듈은 명령 해독 → 데이터 교환 → 상태 보고 → 주소 인식 과정을 거쳐 메시지 인식
  - 명령 해독 : 프로세서에서 명령들을 받아 해독한다.
  - 데이터 교환 : 데이터 버스로 프로세서와 데이터를 교환한다.
  - 상태 보고 : 저속 주변장치의 상태를 확인하여 프로세서에 보고한다.
  - 주소 인식 : 모듈에 연결된 여러 장치를 구분할 주소를 인식한다.
- 버퍼링을 이용하여 전송 속도 조절
- 오류를 검출

4. 프로세서 역할에 따른 입출력 방법

- **입출력 방법**
  - 프로세서 제어 입출력(프로그램 제어 입출력, 인터럽트 기반 입출력)
  - DMA 입출력
  - 입출력 채널

4. 프로세서 역할에 따른 입출력 방법

■ 프로그램 제어 입출력 방법

- 프로세서 내부의 입출력 데이터와 주소 레지스터를 입출력 모듈과 연결한 형태
- 주소 레지스터와 버스 사이에서 데이터를 직접 전송할 수 있는 가장 단순한 형태
- 데이터를 **입력**할 때는 입출력 모듈을 거쳐 한 번에 한 워드씩만 데이터 레지스터로 전송, 입출력 데이터 레지스터에서는 프로그램을 이용하여 산술 논리연산장치로 전송
- 데이터를 **출력**할 때는 산술논리연산장치에서 입출력 데이터 레지스터로 이동, 프로그램을 이용하여 입출력 모듈로 전송
- 구성

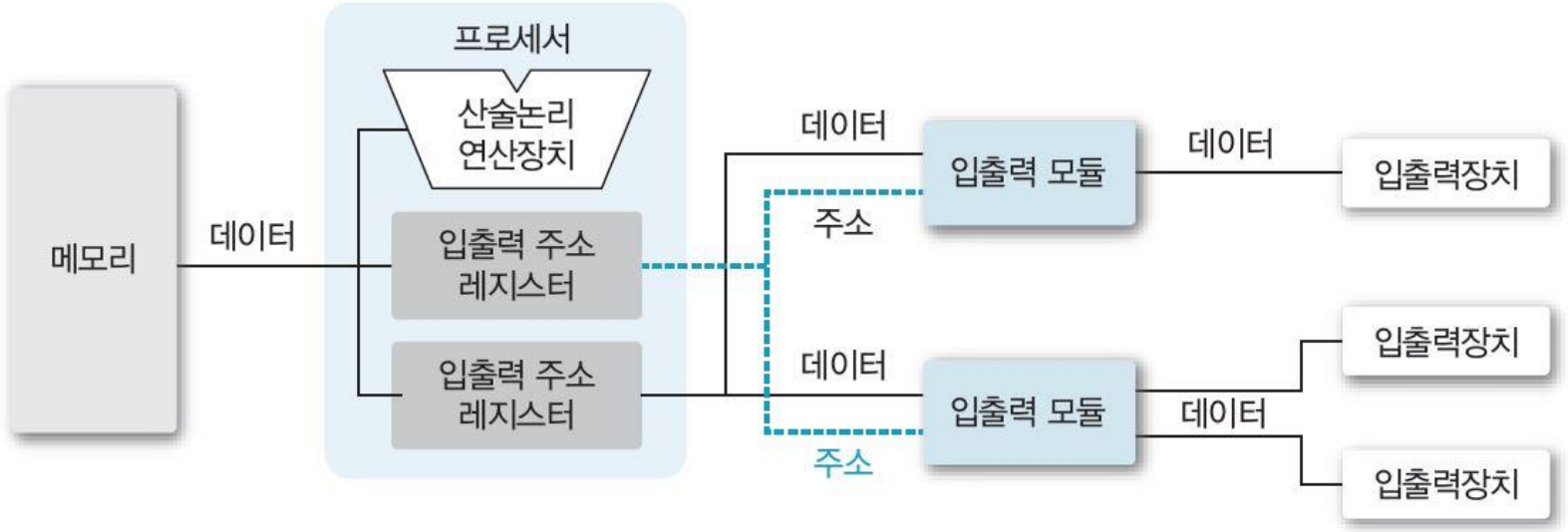


그림 9-3 프로그램 제어 입출력 방법의 구성

4. 프로세서 역할에 따른 입출력 방법

- 프로그램 제어 입출력 방법을 폴링polling 방법이라고도 함
  - **폴링** : 상태 비트를 주기적으로 검사하여 프로세서보다 상대적으로 느린 입출력장치의 상태를 확인
  - 폴링 방법으로 데이터를 전송할 때는 폴링 순환 수행
  - 폴링 방법에서는 순환 횟수가 데이터 전송 성능에 영향. 너무 빈번하게 순환하면 입출력 동작의 진행 여부를 검사하는데 시간 많이 낭비. 반대로 폴링 횟수가 너무 많으면 입출력장치가 오래 쉼

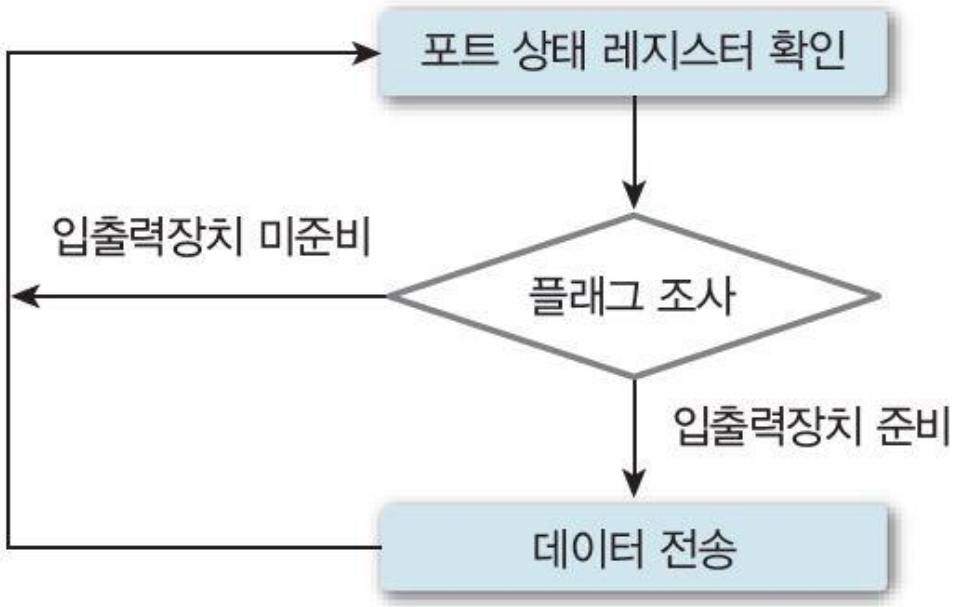


그림 9-4 폴링 순환

4. 프로세서 역할에 따른 입출력 방법

■ 인터럽트 기반 입출력 방법

- 입출력 장치가 작업을 완료한 후에 작업과 관련된 상태와 결과를 메모리에 저장하고 인터럽트를 발생시켜 프로세서에 알림
- 인터럽트를 받은 프로세서는 입출력 명령을 전송하고 입출력 작업 중에 다른 명령 시작
- 프로세서가 프로세서의 도움이 필요한지 입출력장치에 확인하는 대신 입출력장치가 프로세서에 직접 신호를 주는 방법
- 불규칙적이고 빠른 응답성을 요구할 때 적합
- 기본 구성

그림 9-5 인터럽트 기반 입출력 방법의 구성

4. 프로세서 역할에 따른 입출력 방법

- 인터럽트 기반 입출력 방법 과정

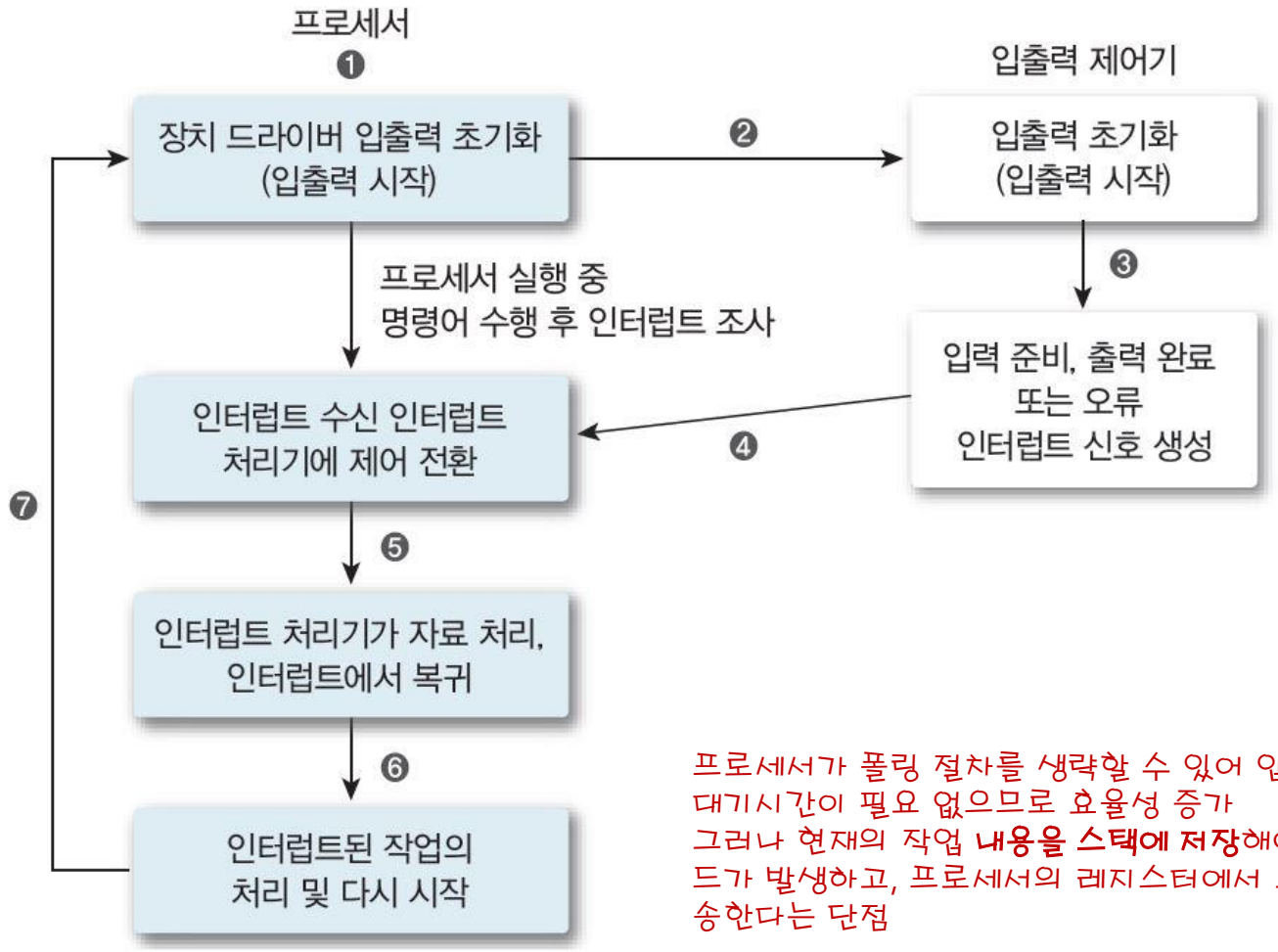


그림 9-6 인터럽트 기반 입출력 방법 과정

프로세서가 폴링 절차를 생략할 수 있어 입출력 연산의 대기시간이 필요 없으므로 효율성 증가
그러나 현재의 작업 **내용을 스택에 저장**해야 하는 오버헤드가 발생하고, 프로세서의 레지스터에서 모든 데이터 전송한다는 단점

4. 프로세서 역할에 따른 입출력 방법

DMA Direct Memory Access 입출력 방법

- 프로세서의 도움 없이도 메인 메모리를 직접 제어하여 데이터를 전송하는 형태
- 직접 메모리 액세스라고 함
- 기본 구성

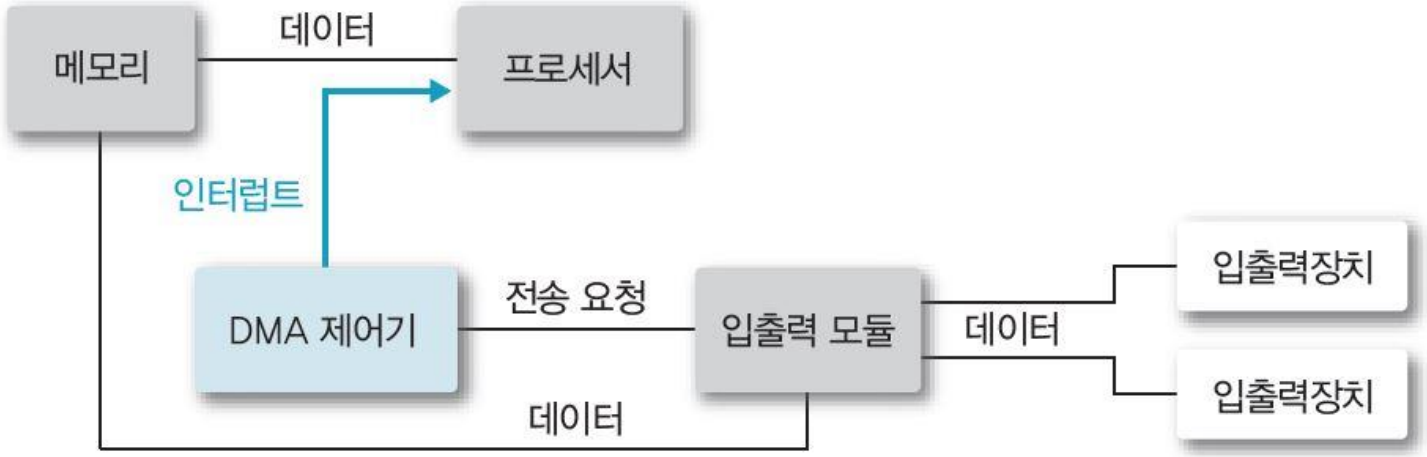
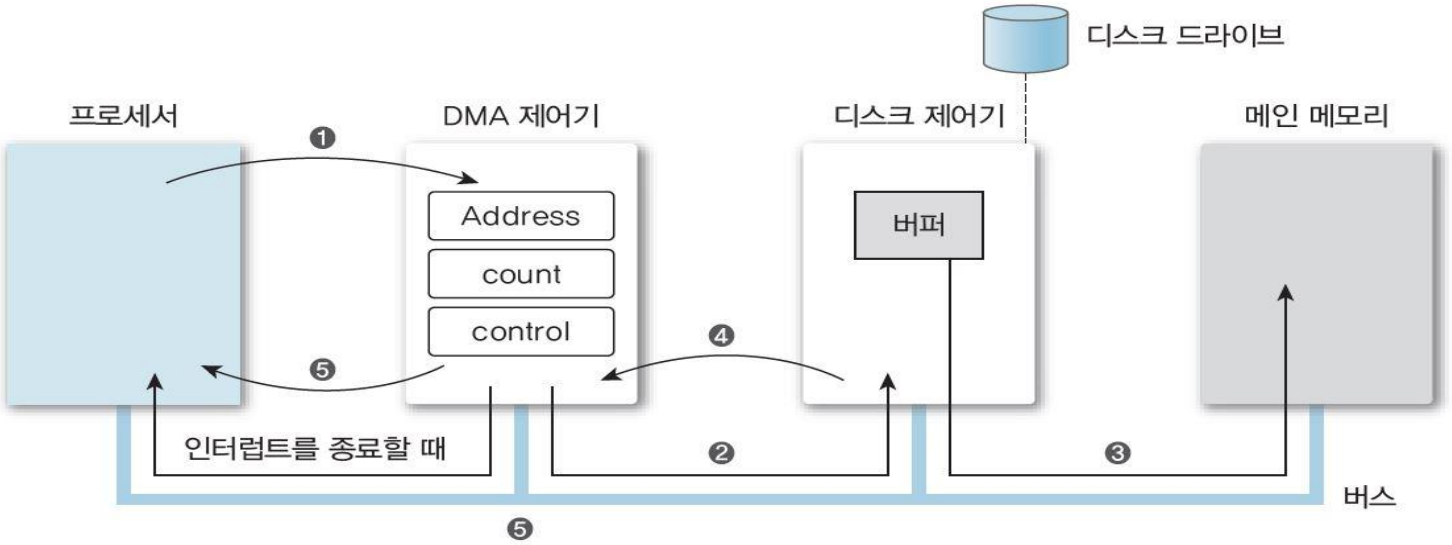


그림 9-7 DMA 입출력 방법의 구성

- 프로세서가 읽기와 쓰기 정보, 입출력 주소와 메모리 주소, 길이를 DMA 제어기에 전달하여 입출력을 요청하면 DMA 제어기는 이것을 직접 처리
- 입출력장치에서 메모리로 **데이터를 블록 단위로 전송** 가능

4. 프로세서 역할에 따른 입출력 방법

- 디스크의 데이터 전송과 멀티미디어 같은 대용량 데이터 전송에 적합
- DMA 전송 동작



❶ 프로세서가 전송 방향, 전송 바이트 수, 데이터 블록의 메모리 주소 등을 DMA 제어기에 보낸다.

❷ DMA 제어기는 디스크 제어기에 데이터를 메인 메모리로 전송하라고 요청한다.

❸ 디스크 제어기가 메인 메모리에 데이터를 전송한다.

❹ 데이터 전송을 완료하면 디스크 제어기는 DMA 제어기에 완료 메시지를 전달한다.

❺ DMA 제어기가 프로세서에 인터럽트 신호를 보낸다.

그림 9-8 DMA 전송 동작 프로세서는 데이터 전송을 시작하고 종료 할 때만 관여함

CPU를 통하지 않고 주변기기의 인터페이스 장치에 제어권을 주어 직접 주기억장치와 데이터를 주고 받는 방식으로 액세스하여 데이터를 전송하는 방법
▶ 입 · 출력 속도가 향상
▶ DMA가 메모리를 접근하기 위해서는 Cycle stealing(사이클 도용)을 한다.
▶ CPU의 계속적인 관여 없이 데이터를 메모리와 주변장치 사이에 전송 한다.
▶ 전송이 끝나면 DMA 제어기는 CPU를 인터럽트 한다.

4. 프로세서 역할에 따른 입출력 방법

■ 입출력 채널을 이용한 입출력

- **입출력 채널** : 프로세서와 메인 메모리를 입출력장치에 결합하여 프로세서의 명령으로 입출력을 제어하는 장치
- 채널 서브시스템은 여러 개로 구성 가능, 프로세서와 메인 메모리의 액세스 조정
- 채널은 사이클 스틸을 사용하여 메모리에 직접 액세스하며, 각 입출력장치는 제어장치를 사용하여 채널과 연결
- 제어장치는 채널과 통신하고 장치를 제어하는 기능을 수행. 유사한 장치에서는 제어 장치를 하나만 사용하므로 한 순간에는 장치 하나만 활성화.
- 입출력 채널에는 DMA 개념을 확장하여 입출력 명령어들을 실행할 수 있는 기능이 있으므로 입출력 동작을 완전히 제어하는 권한 있음
- 컴퓨터 시스템에서는 프로세서가 입출력 명령어들을 실행하지 않아도 입출력 데이터를 메인 메모리에 저장
- **프로세서**는 입출력 채널이 메인 메모리에 있는 프로그램을 실행하도록 지시하여 입출력 전송 실행. **입출력 채널**은 메인 메모리에 저장된 명령어들을 실행하면서 데이터 전송 제어

사이클 스틸 :
DMA 제어기와 CPU가 동시에 주기억장치를 접근하려고 할 때 한 사이클을 데이터 채널에게 양보
즉, 입출력 자료 전송을 빠르게 해줌

4. 프로세서 역할에 따른 입출력 방법

- 입출력 채널을 사용하는 컴퓨터 구조의 예

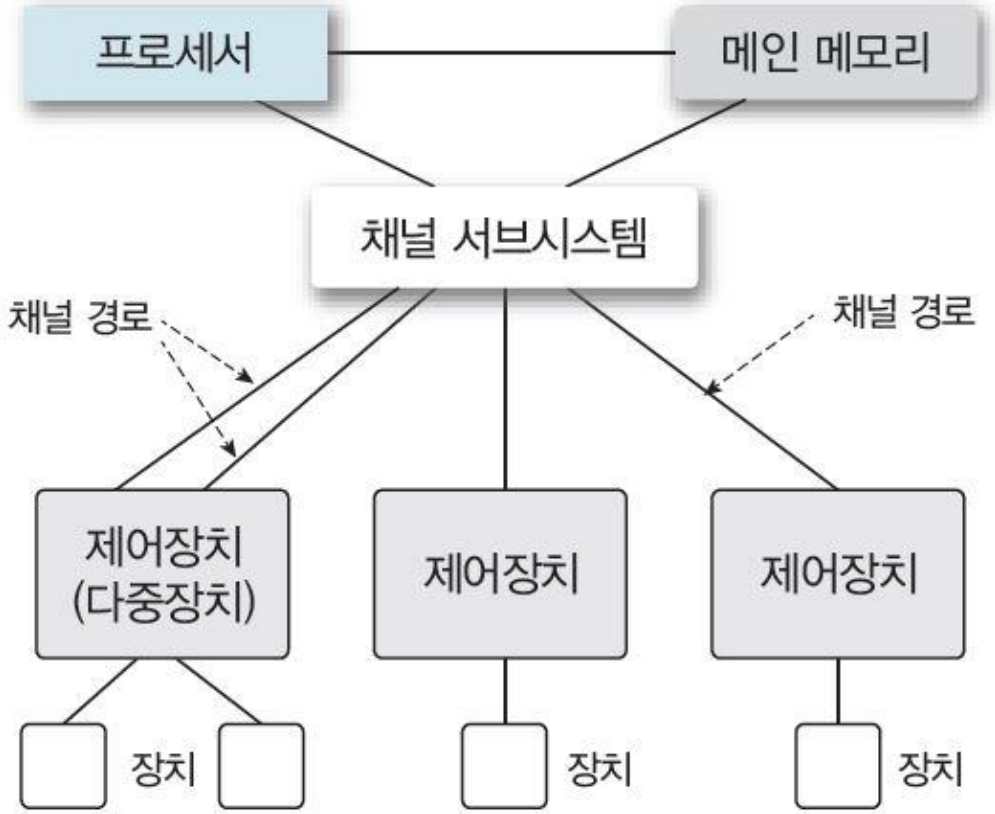
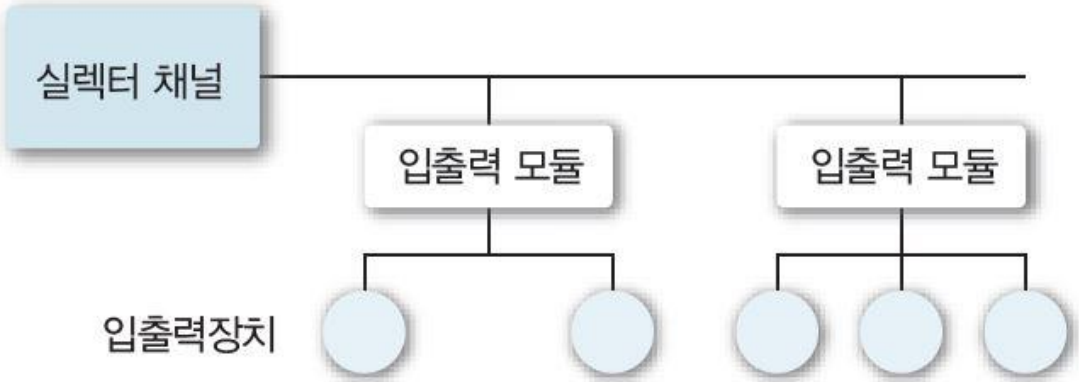


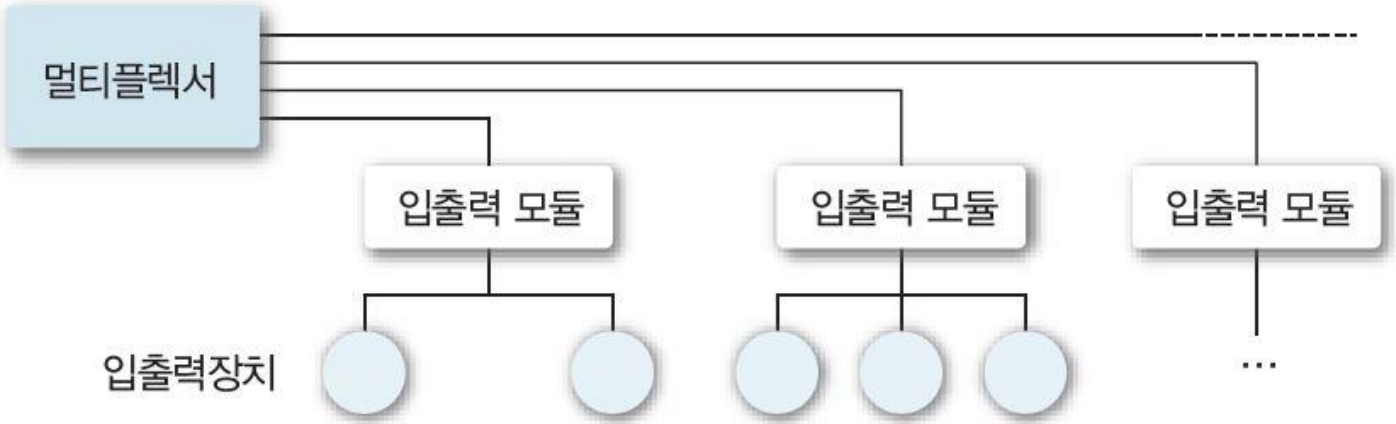
그림 9-9 입출력 채널 구조

4. 프로세서 역할에 따른 입출력 방법

- 입출력 채널의 여러 형태



(a) 실렉터 채널



(b) 멀티플렉서 채널

그림 9-10 입출력 채널의 여러 형태

4. 프로세서 역할에 따른 입출력 방법

- 채널의 종류
  - **실렉터 채널** : 어떤 장치의 입출력을 종료할 때까지 다른 장치를 실행하지 않도록 하는 것. 자기디스크, 자기테이프, 드럼과 같은 고속 장치들은 멀티플렉서 채널을 사용하여 데이터를 고속으로 전송하기 때문에 다른 장치들과 다중화 곤란. 이때 한 번에 하나씩 처리하는 전용 채널을 이용하여 다중화 가능. 어느 한순간에 입출력 장치 중 한 장치와 하는 데이터 전송만 제어하므로 입출력 채널은 장치를 한 개 선택하여 데이터 전송
  - **멀티플렉서 채널** : 바이트 단위로 시분할하여 여러 장치의 출력 처리. 다수의 저속·중속 장치(카드 리더, 프린터)들을 연결할 때 사용. 채널-메모리 연결은 장치와 채널 사이의 데이터 전달 속도보다 더 빠른 속도로 데이터를 전달할 수 있기 때문에 속도가 느린 다수의 입출력장치를 동시에 동작
  - **블록 멀티플렉서 채널** : 실렉터 채널과 멀티플렉서 채널의 장점을 결합한 것, 여러 대의 고속 입출력 장치들을 블록 단위로 처리. 동일한 채널에서 여러 고속장치를 활성화 가능. 채널 명령어는 장치 하나와 입출력 명령어 수행 후 별도의 지시가 없어도 자동으로 다른 장치와 명령을 수행하도록 변환되는 특징. 한 채널이 여러 입출력장치를 시분할 방법으로 사용할 수 있지만, 어떤 때는 장치 하나에만 서비스하므로 실렉터 채널과 비슷. 그러나 멀티플렉서 채널처럼 다른 장치 서비스를 완료할 때까지 기다리지 않아도 됨

5. 커널 입출력 서브 시스템

■ 커널 입출력 구조

- 커널(운영체제)이 제공하는 입출력 서비스와 입출력 인터페이스를 입출력 장치들이 정의한 방법에 따라 실행하려면 먼저 인터페이스를 구성해야 함
- 커널 입출력 구조

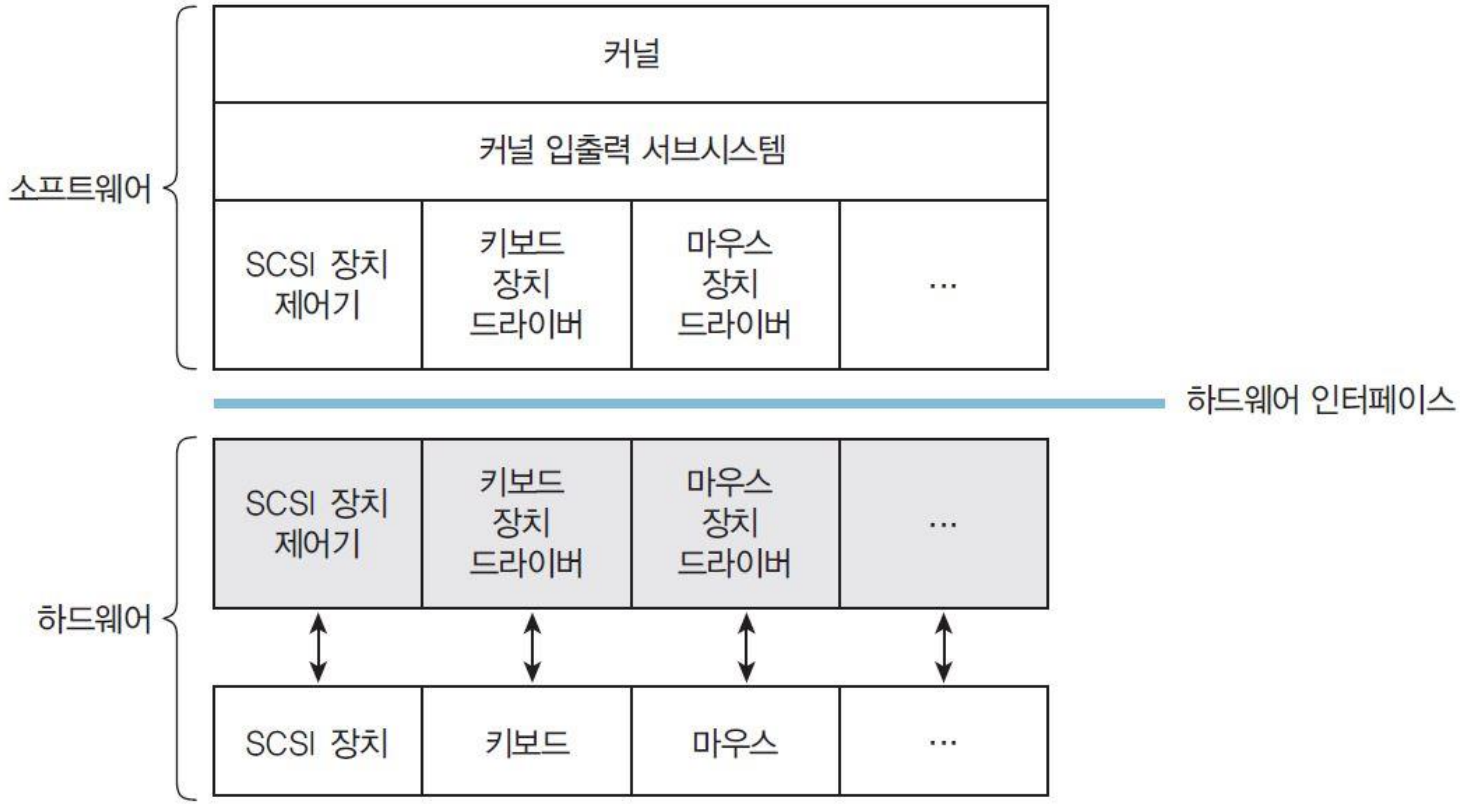


그림 9-11 커널 입출력 구조

5. 커널 입출력 서브 시스템

■ 커널이 제공하는 입출력 관련 서비스(컴퓨터의 효율성 증진)

- 입출력 스케줄링
  - 입출력 요구를 스케줄링 한다는 것은 요구들의 실행 순서를 결정한다는 의미.
- 버퍼링
  - 버퍼링은 입출력 장치와 응용 프로그램 사이에 전송되는 데이터를 버퍼에 임시로 저장하는 방법
  - 버퍼링은 송신자와 수신자의 전송속도 차이로 발생하는 시스템의 데이터 전송 문제를 해결

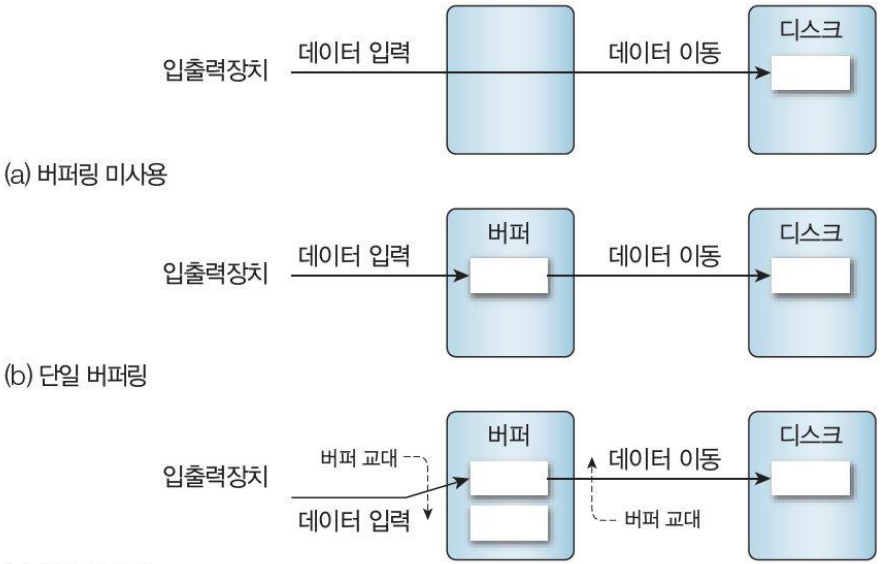


그림 9-12 버퍼링 사용 예

5. 커널 입출력 서브 시스템

- 캐싱
  - 캐싱은 명령어와 데이터를 캐시에 일시적으로 저장하여 프로세서와 메모리 간의 액세스 속도 차이를 줄여서 컴퓨터 성능 향상 방법
  - 캐시는 버퍼와 달리 자주 사용할 자료를 미리 복사하여 저장하는 빠른 메모리 영역. 데이터가 위치하는 유일한 장소가 버퍼라고 한다면, 다른 곳에 저장된 데이터의 복사본을 저장하는 장소는 캐시
  - 캐싱과 버퍼링은 서로 기능이 다르지만, 때로는 한 메모리 영역을 두 가지 목적에서 사용 가능.
  - 데이터가 캐시에 있으면 디스크 입출력을 생략하여 입출력의 효율성 높임
  - 메모리 내에 버퍼를 사용하여 디스크 입출력 스케줄링을 효율적으로 수행하거나 프로세스 간에 공유해야 하는 파일들을 위한 캐시로도 사용 가능

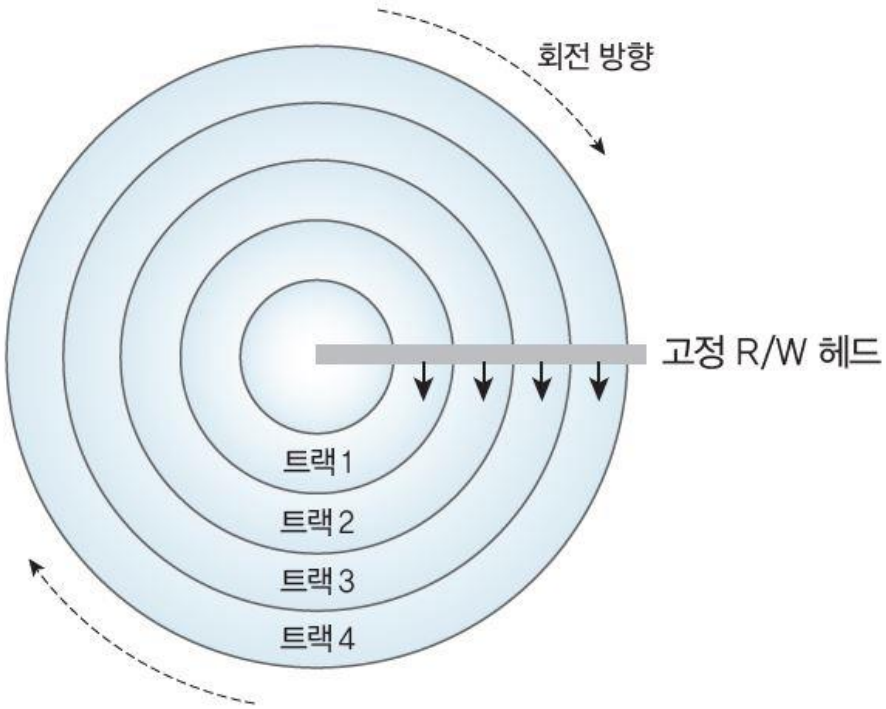
- 스풀링
  - 많은 응용 프로그램이 동시에 출력 데이터를 프린터로 전송하면 커널(운영체제)은 각 출력이 다른 프로그램의 출력과 섞이지 않고 프린터로 출력할 수 있도록 관리. 각 응용 프로그램의 출력은 각각 대응되는 디스크 파일에 스풀링한다. 스풀링 시스템은 응용 프로그램이 출력 데이터 작업을 종료하면 그때까지 모아 놓은 출력 데이터를 프린터 출력용 준비 큐에 삽입하여 스풀 파일을 한 번에 하나씩 프린터에 내보낸다. 이런 스풀링은 커널 스레드로 처리

5. 커널 입출력 서브 시스템

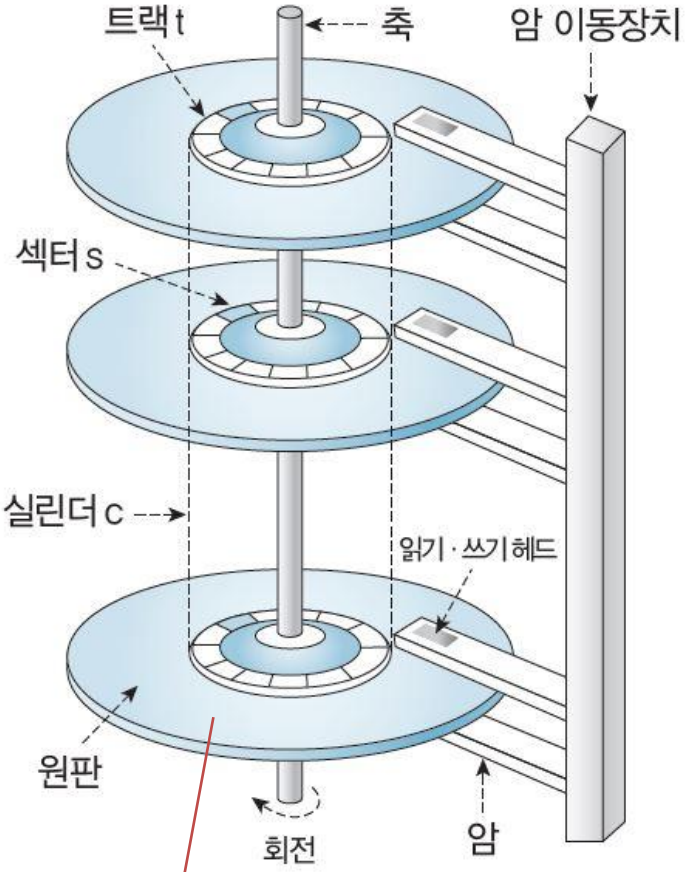
- 오류 처리
  - 커널은 입출력 장치의 고장이나 네트워크 전송 오류로 발생하는 일시적인 고장을 효과적으로 해결, 디스크 읽기 실패나 망의 전송 오류는 재실행하여 문제 해결. 그러나 중요한 구성 요소가 영구적인 고장을 일으키면 커널이 문제를 완전히 극복하기는 곤란

- 자료 관리
  - 커널은 입출력 구성에서 상태 정보를 유지. 입출력 서비스를 커널이 아닌 독립 프로그램이 담당하면 장치 드라이버와 커널의 정보 공유로 메시지를 교환하려는 오버헤드가 증가하는 단점. 장점은 입출력 시스템의 구조와 설계가 간단하고, 운영체제 커널의 크기가 작고, 유연성이 좋다는 것.
  - 입출력 서브시스템은 광범위한 서비스를 조정하며, 다음 사항들을 관리
    - 장치 이름과 장치 액세스 제어 관리
    - 장치 할당과 입출력 스케줄링 관리
    - 버퍼링, 캐싱, 스풀링 관리
    - 장치 상태 모니터링과 오류 처리, 고장 복구 관리

디스크의 구조



(a) 고정 헤드 디스크



(b) 이동 헤드 디스크장치

그림 9-13 디스크의 종류와 구조

디스크를 교체할 수 있어 필요 에 따라 다른 디스크를 장착하여 사용

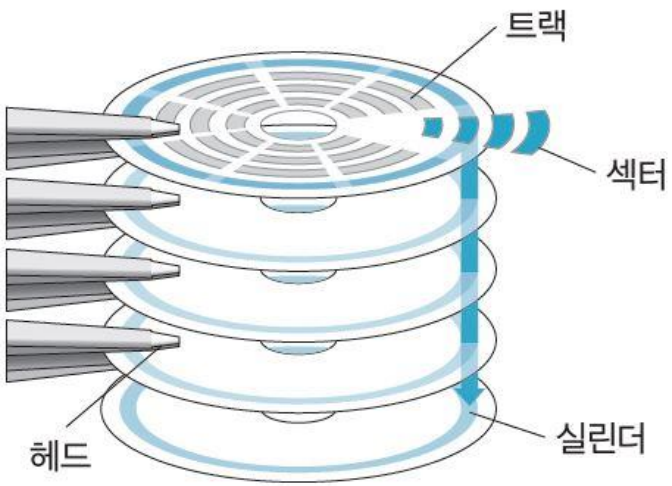
1. 디스크의 구조

디스크 시스템

- 디스크 드라이버, 프로세서, 디스크 제어기로 구분
  - **디스크 드라이버** : 구동 모터, 액세스 암 이동장치, 입출력 헤드 부분의 기계적인 부분 담당
  - **프로세서** : 컴퓨터의 논리적인 상호작용, 즉 데이터의 위치(디스크 주소)와 버퍼, 판독, 기록 등 관리
  - **디스크 제어기**
    - 디스크 드라이버의 인터페이스 역할
    - 프로세서에서 명령을 받아 디스크 드라이버 동작, 디스크 드라이버는 탐색seek, 기록, 판독 등 명령 수행
  - 디스크의 정보는 드라이버 번호, 표면 번호, 트랙 번호 등으로 나누는 디스크 주소로 참조
    - **트랙(track)** : 원형 평판 표면에 데이터를 저장할 수 있는 동심원을 가리킨다. 자기장의 간섭을 줄 이거나 헤드를 정렬하려고 트랙 사이에 일정한 공간을 두어 트랙 구분
    - **실린더(cylinder)** : 동일한 동심원으로 구성된 모든 트랙, 즉 동일한 위치에 있는 모든 트랙의 집합을 의미. 헤드의 움직임 없이 액세스할 수 있는 드라이브의 모든 트랙에 해당
    - **섹터(secter)** : 트랙을 부채꼴 모양으로 나눈 조각을 의미. 트랙 내의 정보는 블록을 구성하고, 이 블록이 하드웨어적으로 크기가 고정되었을 때가 바로 섹터. 섹터는 데이터 기록이 나 전송의 기본 단위로, 일반적으로 512Byte 데이터 영역으로 구성. 섹터는 고유 번호가 있어 디스크에 저장된 데이터의 위치 식별 가능

1. 디스크의 구조

- 디스크의 논리적 구조



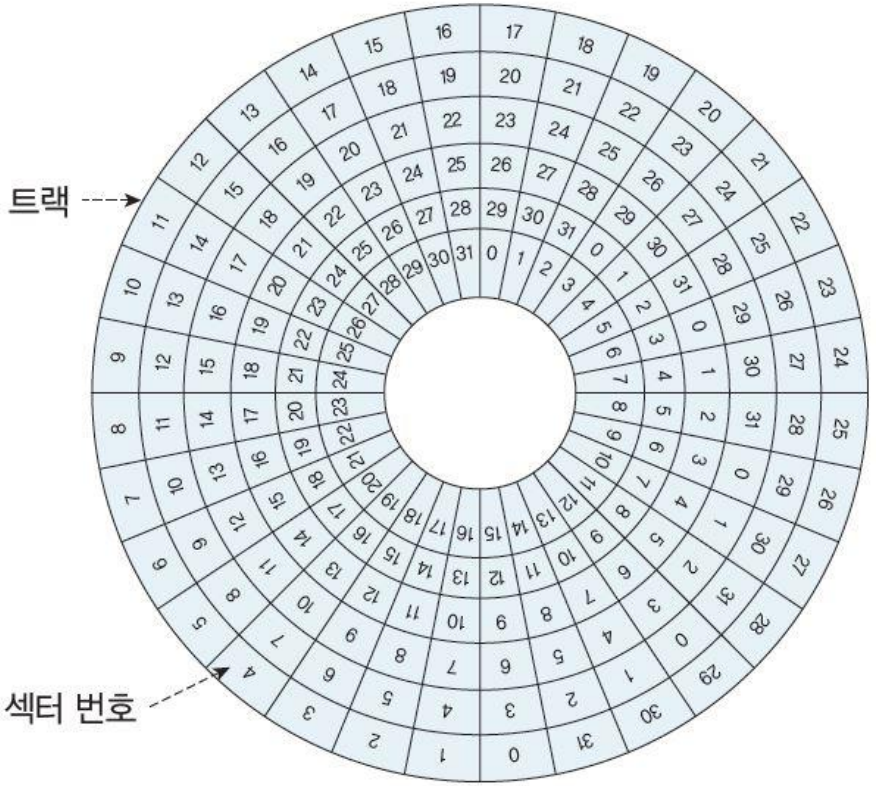
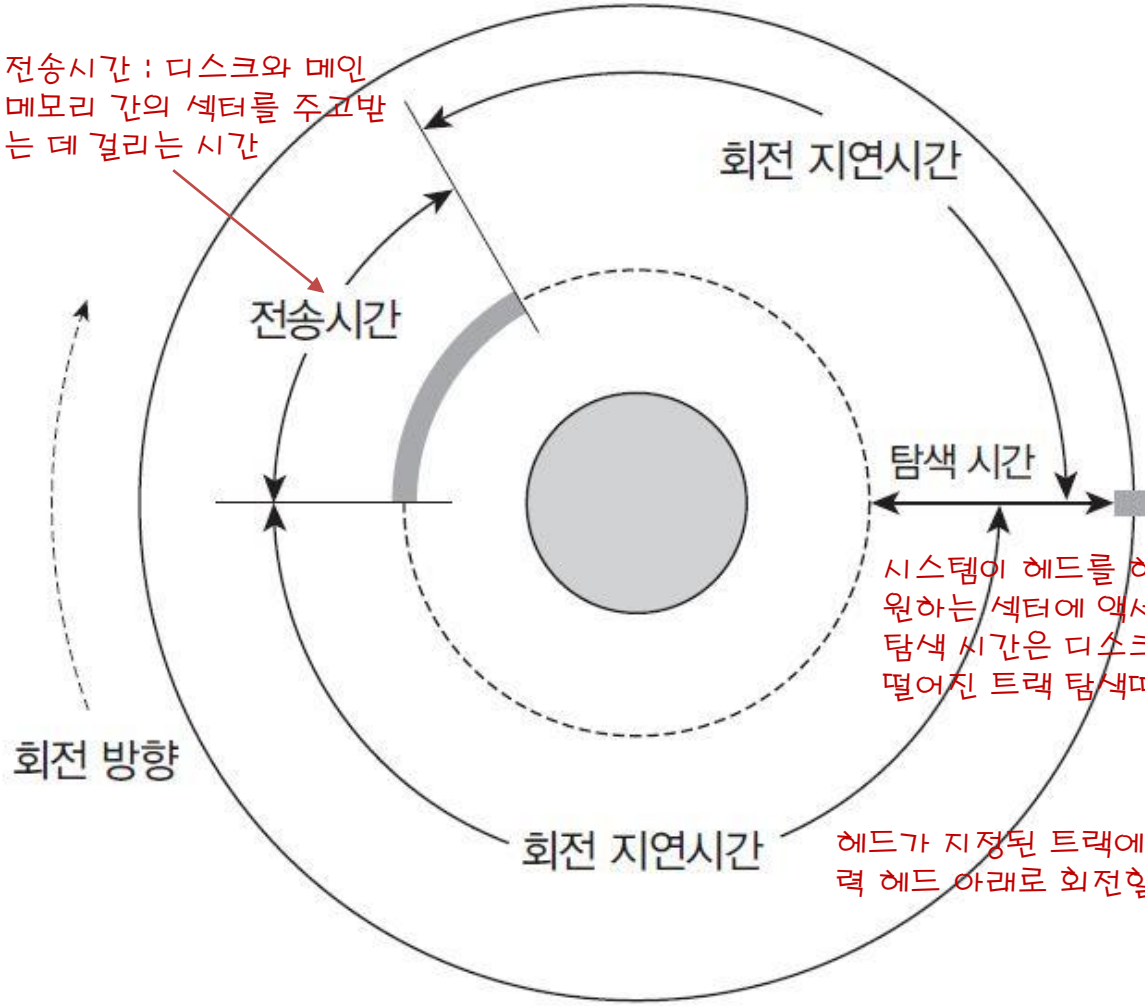


그림 9-14 디스크의 논리적 구조

2. 디스크의 액세스 시간

■ 디스크 액세스 시간의 개념



전송시간 : 디스크와 메인 메모리 간의 섹터를 주고받는 데 걸리는 시간

시스템이 헤드를 해당 트랙이나 실린더에 위치시켜 디스크의 원하는 섹터에 액세스에 걸리는 시간
탐색 시간은 디스크의 헤드가 움직이는 시간에 좌우되어 멀리 떨어진 트랙 탐색때 탐색 시간 김

헤드가 지정된 트랙에 위치해도 원하는 섹터가 입출력 헤드 아래로 회전할 때까지 기다리는 시간

그림 9-15 디스크 액세스 시간 회전하는 디스크의 섹터를 액세스할 수 있는 시간

2. 디스크의 액세스 시간

디스크 액세스 시간의 계산

- 이동 디스크 : 탐색 시간+회전 지연시간+전송시간
- 고정 헤드 디스크 : 회전 지연시간+전송시간

- 탐색 시간 : 50밀리초
- 회전 지연시간 : 16.8밀리초
- 전송시간 : 0.00094밀리초/바이트[1KB 전송시간 : 0.96256(= 0.00094×1,024)]

(a) 탐색 시간, 회전 지연시간, 전송시간 예

- 이동 헤드 디스크의 데이터 액세스 시간 = 50 + 16.8 + 0.96256
- 고정 헤드 디스크의 데이터 액세스 시간 = 16.8 + 0.96256

(b) 디스크 종류에 따른 데이터 액세스 시간

그림 9-16 디스크 액세스 시간을 계산하는 예

2. 디스크의 액세스 시간

- **섹터 번호 k의 디스크 주소(b)**

$$b = k + s \times (j + i \times t)$$

s : 트랙당 섹터 수
t : 실린더당 트랙 수
i : 실린더 번호
j : 표면 번호

그림 9-17 섹터 번호 k의 디스크 주소(b)

네트워크 저장장치

DAS Direct Attached Storage

- 서버와 같은 컴퓨터에 직접 연결된 저장장치를 사용하는 방식
- HAS Host Attached Storage라고도 부름
- [예] 윈도우의 파일 공유(여러 컴퓨터 중 하나를 파일 공유 서버로 지정하고 나머지 컴퓨터에서 서버로 지정된 컴퓨터에 접근하여 파일을 이용하는 방식으로 운영)
- 단점
  - 컴퓨터에 직접 연결된 저장장치를 사용하기 때문에 다른 운영체제가 쓰는 파일 시스템을 사용할 수 없음
  - 데이터의 관리나 백업을 사용자가 직접 해야 함

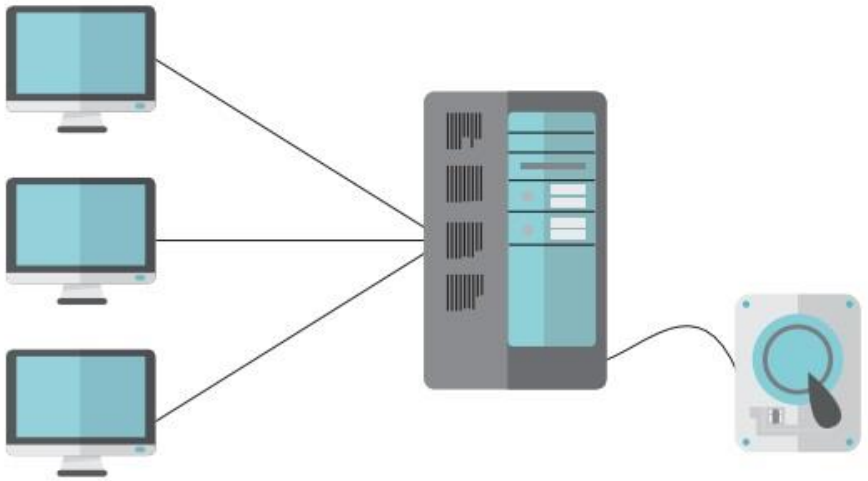


그림 10-21 DAS의 구조

네트워크 저장장치

NAS Network Attached Storage

- 기존의 저장장치를 LAN이나 WAN에 붙여서 사용하는 방식
- NAS 전용 운영체제를 가진 독립적인 장치로 새로운 하드디스크를 추가하거나 뺄 수 있음
- 저장장치를 네트워크상에 두고 여러 클라이언트가 네트워크를 통해 접근하게 함으로써 공유 데이터의 관리 및 데이터의 중복 회피가 가능

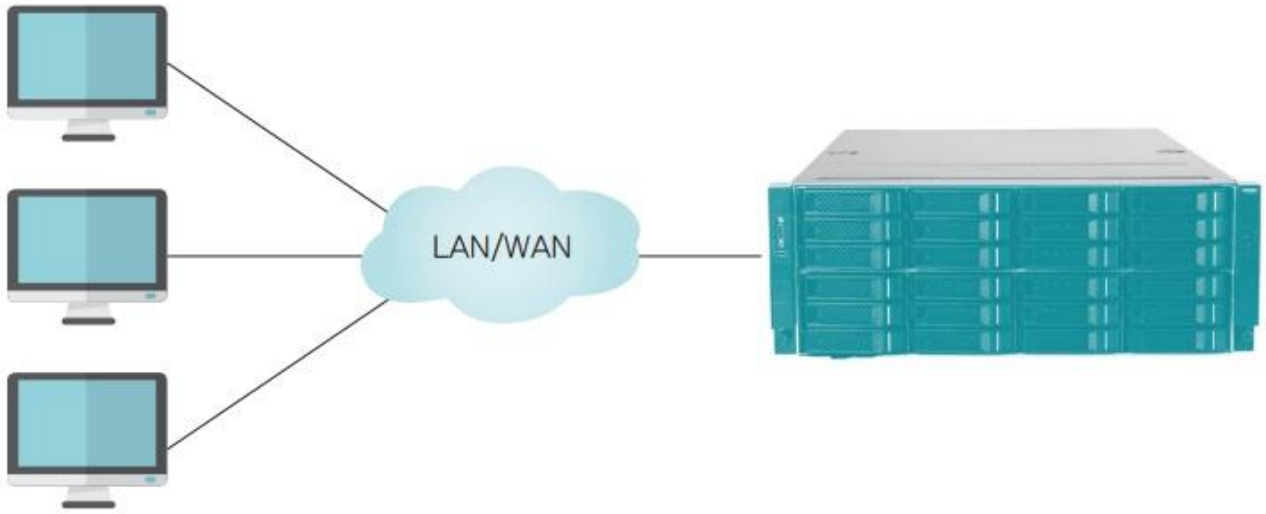


그림 10-22 NAS의 구조

네트워크 저장장치

SAN Storage Area Network

- 데이터 서버, 백업 서버, RAID 등의 장치를 네트워크로 묶고 데이터 접근을 위한 서버를 두는 형태
- 시스템이 제공하는 인터페이스를 통해 데이터에 접근
- 저장장치에 필요한 장치를 네트워크로 묶어 하나의 시스템을 구성하기 때문에 다양한 서비스를 제공
- 데이터의 공유, 백업, 보안등이 서버를 통해 자동으로 이루어짐
- 데이터 서버나 백업 서버를 같이 구축하여 NAS보다 구축 비용이 많이 듬

그림 10-23 SAN의 구조

3. 디스크 스케줄링의 개념과 종류

입출력장치(디스크 드라이버)의 요청 큐가 포함하는 정보

- 입력 동작인지, 출력 동작인지 하는 정보
- 디스크 주소(구동기, 실린더, 표면, 블록)
- 메모리 주소
- 전송할 정보의 총량(바이트나 단어의 수)

하나의 장치를 제어하거나 조절하는 하드웨어 장치 또는 프로그램

디스크 스케줄링의 평가 기준

- 처리량 : 시간당 처리한 서비스 요청 수
- 탐색 시간 : 디스크 헤드(암) 이동 시간
- 평균 반응시간 : 요청 후 서비스할 때까지 대기시간
- 반응(응답)시간 변화 : 반응시간 예측 정도. 즉, 적절한 시간 안에 서비스하여 요청이 무기한 연기되지 않도록 방지

운영체제

- 디스크 액세스 요청을 스케줄하여 디스크 처리하는 평균 시간을 향상시키는데 처리량을 최대화하고 평균 반응시간을 최소화 하면서 탐색 시간을 최소화하도록 해야 함. 시스템 성능은 처리량과 평균 반응시간을 최적화하여 향상시킬 수 있음. 그러나 개별 요청에 지연이 발생할 수 있으므로 시스템 자원을 효과적으로 사용하려면 스케줄링이 필요

3. 디스크 스케줄링의 개념과 종류

■ 디스크 스케줄링의 종류

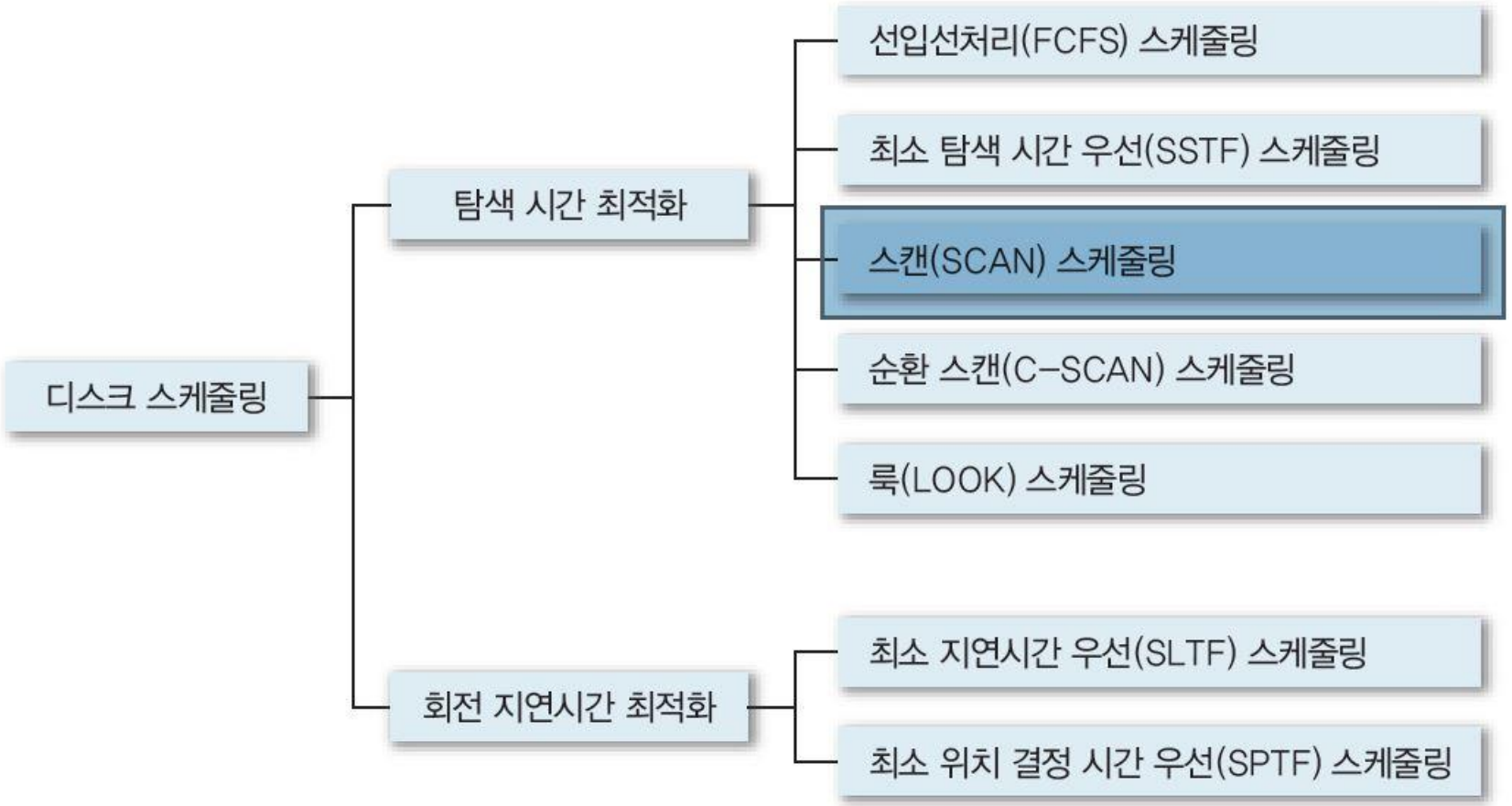


그림 9-18 디스크 스케줄링의 종류

4. 선입선처리 스케줄링FCFS, First Come First Served

선입선처리 스케줄링의 개념

- 요청이 도착한 순서에 따라 처리하는 가장 간단한 스케줄링 알고리즘
- 프로그래밍 쉽고, 어떤 요청도 무기한 연기하지 않고, 본질적으로 공평성(공정성) 유지
- 디스크 요청이 흩어질 때는 실행 시간 오버헤드 적다. 그러나 서비스 지연을 감소시키는 요청을 재정렬하지 않아서 일반적인 임의의 탐색 패턴 결과로 탐색 시간이 증가하면서 처리량이 감소한다는 단점. 헤드 이동 거리 : 454개

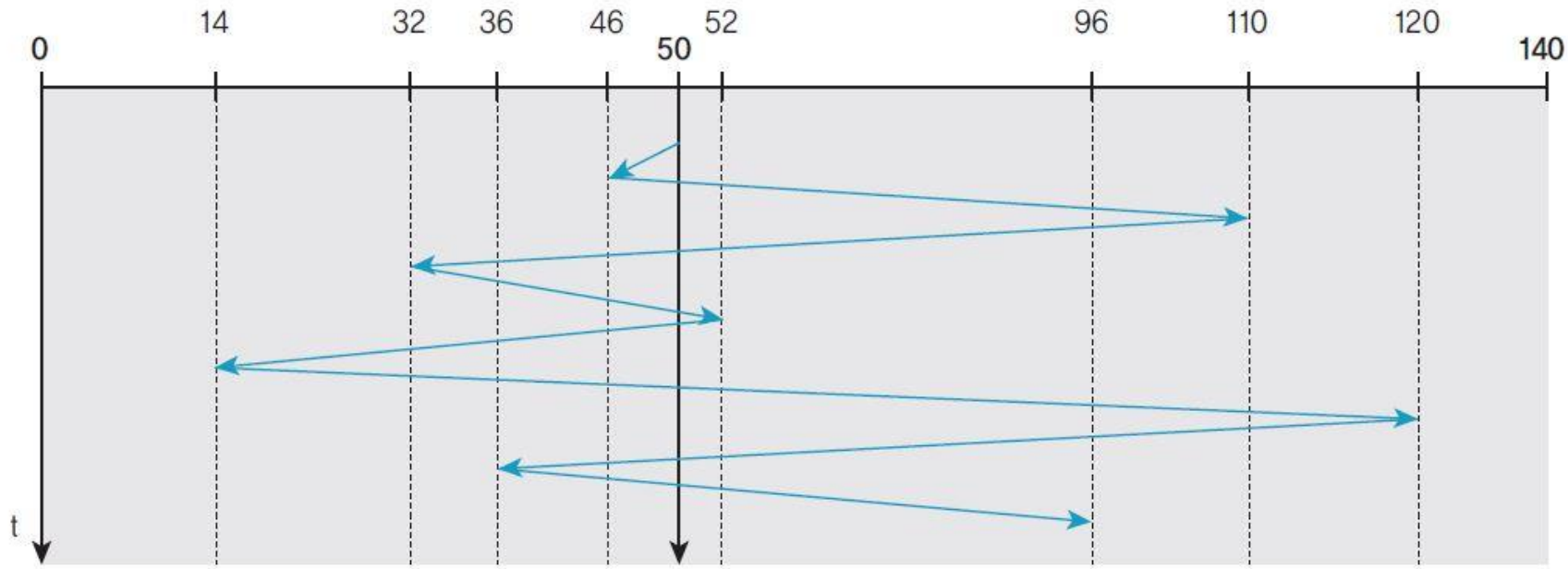


그림 9-19 선입선처리 스케줄링

큐 : 46, 110, 32, 52, 14, 120, 36, 96
헤드 시작 위치 : 50

5. 최소 탐색 시간 우선 SSTF, Shortest Seek Time First 스케줄링

■ 최소 탐색 시간 우선 스케줄링의 개념

- 디스크 요청을 처리하려고 헤드가 먼 곳까지 이동하기 전에 현재 헤드 위치에 가까운 모든 요구를 먼저 처리하는 방법
- 선입선 처리와 비교하면 헤드의 이동 거리가 1/3 정도로 트랙 146개를 이동하므로 디스크 서비스 시간을 실질적으로 줄일 수 있음
- 최소작업을 우선 수행하므로 디스크 요구의 기아 상태 발생 가능.(트랙 14 ~ 트랙 120)
- 공정성을 보장할 수 없고, 서비스를 무기한 연기할 수 있으며, 응답시간의 높은 분산은 대화형 시스템에서는 받아 들일 수 없는 단점, 일괄 처리 시스템에 합리적임.

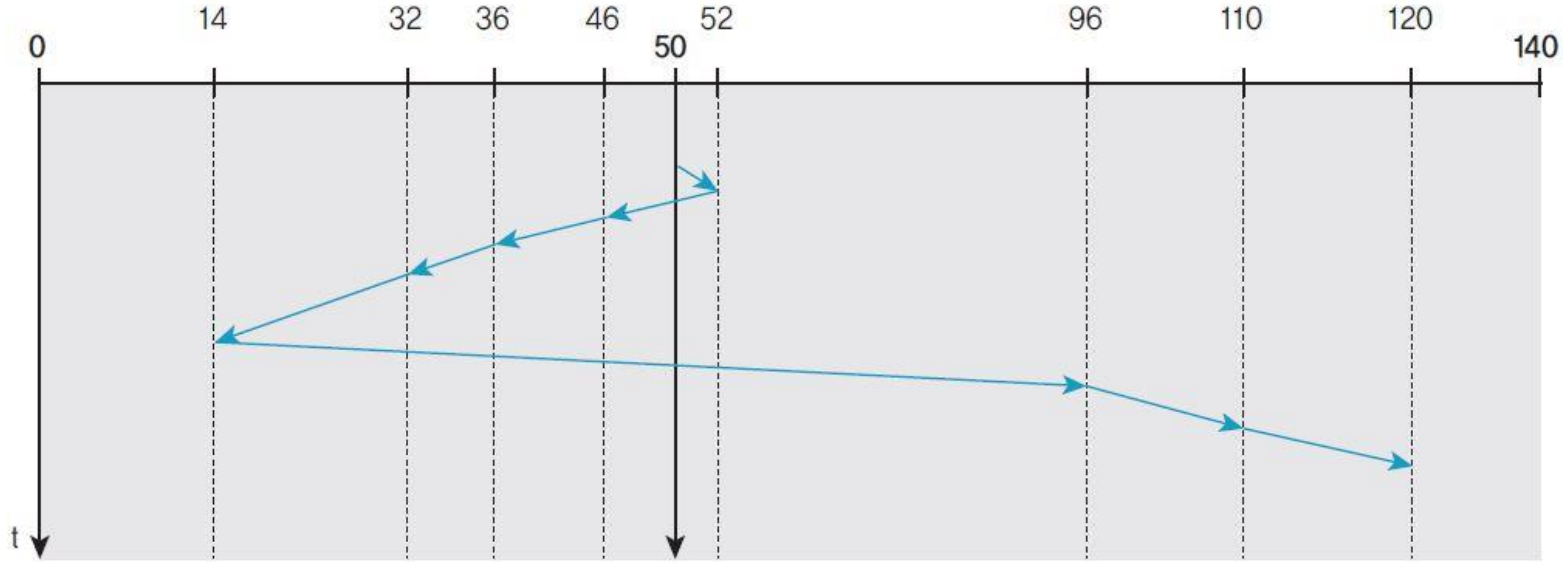


그림 9-20 최소 탐색 시간 우선 스케줄링

큐 : 46, 110, 32, 52, 14, 120, 36, 96
헤드 시작 위치 : 50

6. 스캔 스케줄링

■ 스캔 스케줄링의 개념

- 요청 큐의 동적 특성을 반영하여 기아 상태 해결
- 입출력 헤드가 디스크의 한쪽 끝에서 다른 끝으로 이동하며, 한쪽 끝에 도달했을 때는 역방향으로 이동하면서 요청한 트랙을 처리.

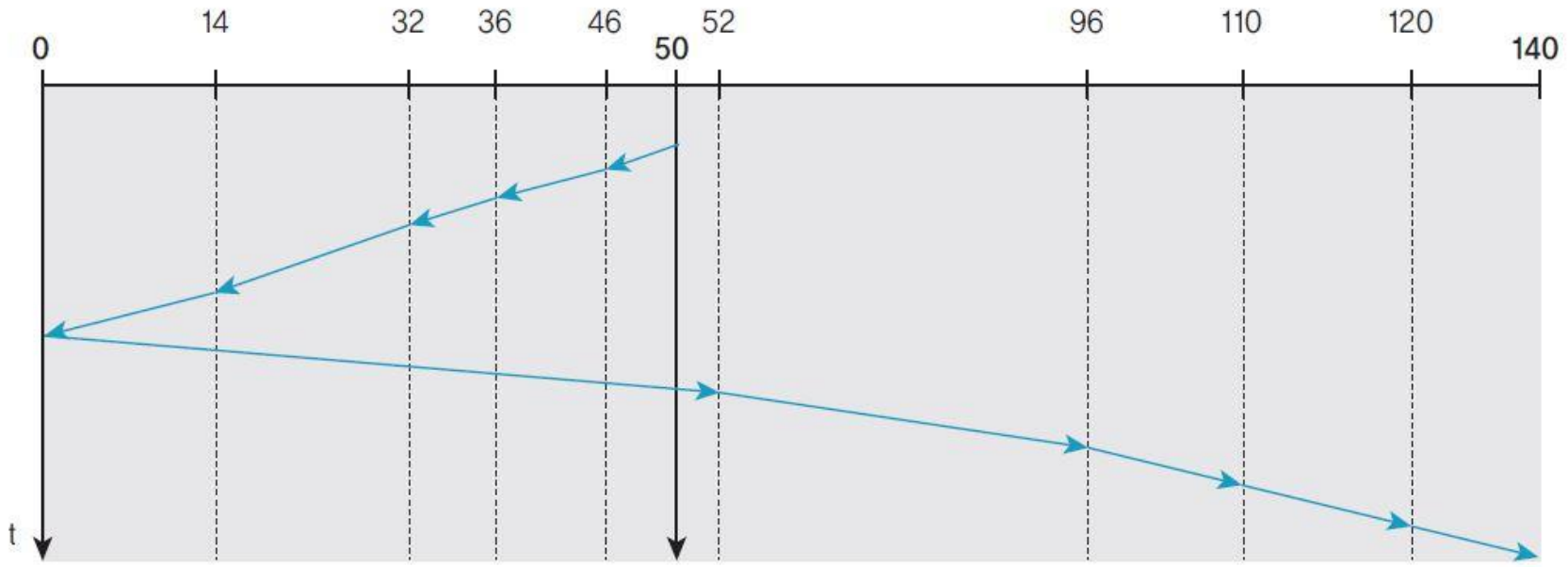


그림 9-21 스캔 스케줄링(안쪽(0) 트랙으로 이동)

큐 : 46, 110, 32, 52, 14, 120, 36, 96
헤드 시작 위치 : 50
헤드 이동 방향 : 0 ↔ 140

7. 순환 스캔C-SCAN, Circular-SCAN 스케줄링

순환 스캔 스케줄링의 개념

- 스캔 스케줄링을 변형하여 대기시간을 좀 더 균등하게 처리하는 방법
- 스캔 스케줄링처럼 헤드는 한쪽 방향으로 이동하면서 요청을 처리 하지만, 한쪽 끝에 다다르면 역방향으로 헤드를 이동하는 것이 아니라 **다시 처음부터 요청 처리**
- 처음과 마지막 트랙을 서로 인접시킨 원형처럼 디스크 처리 처리량 향상
- 바깥쪽 트랙과 안쪽 트랙을 차별하지 않아 반응시간의 변화 줄임
- 동일한 실린더(트랙) 요청이 연속적으로 발생하면 처리가 무기한 연기

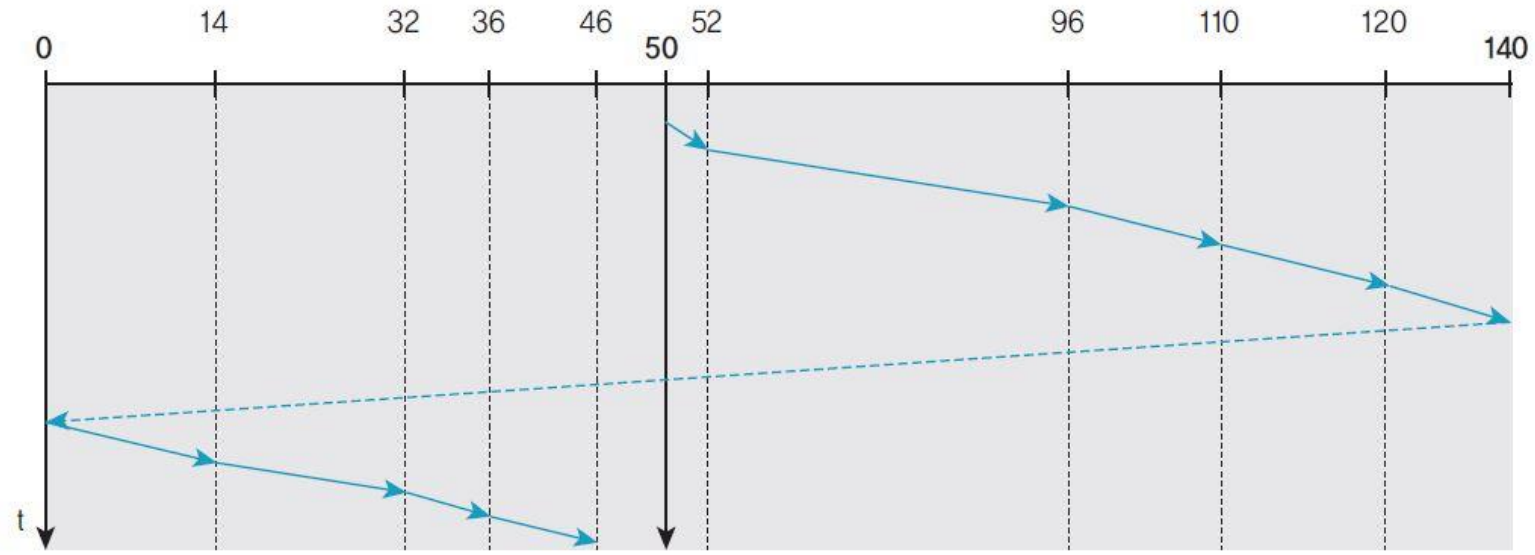


그림 9-23 순환 스캔 스케줄링

큐 : 46, 110, 32, 52, 14, 120, 36, 96
헤드 시작 위치 : 50

8. 룩 스케줄링

룩 스케줄링의 개념

- 스캔 스케줄링이나 순환 스캔 스케줄링은 헤드를 디스크의 끝에서 끝으로 이동하는 원리
- 보통 헤드는 요청에 따라 각 방향으로 이동하지만, **현재 방향에 더는 요청이 없을 때 이동 방향을 바꿔 서비스를 처리**
- 스캔 스케줄링과 순환 스캔 스케줄링의 이런 형태를 룩 또는 순환룩이라고 함
- 룩은 진행 방향으로 움직이기 전에 먼저 요청이 있는지 검사한다는 의미
- 스캔 스케줄링이나 순환 스캔 스케줄링은 헤드를 디스크의 끝에서 끝으로 이동하는 원리 반면에 룩 스케줄링은 헤드를 각 방향으로 요청에 따르는 거리 만큼만 이동하는 원리
- 현재 방향에서 더는 요청이 없다면 헤드의 이동 방향을 바꾸는 방법

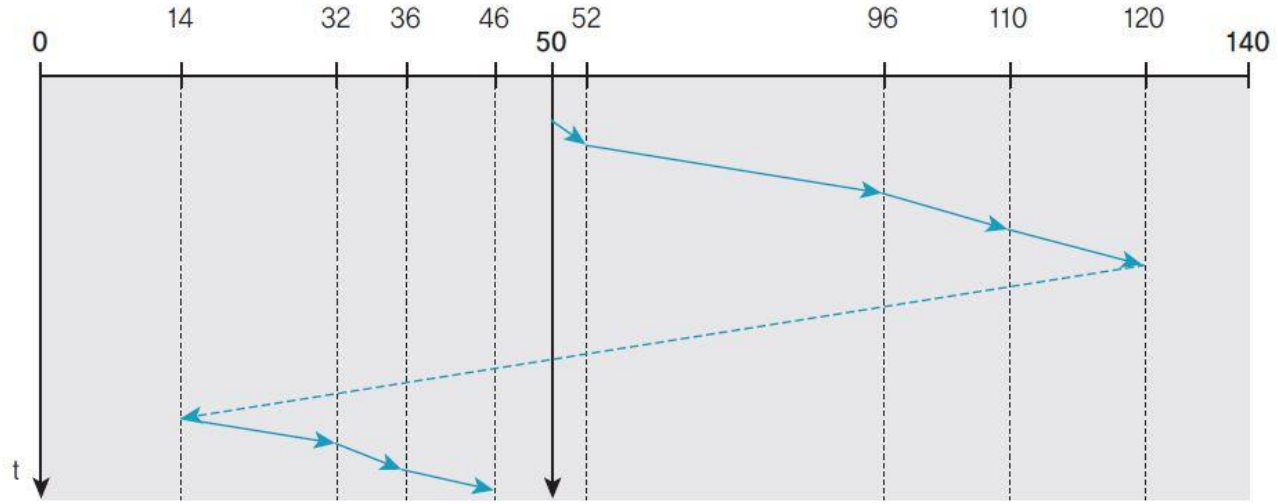


그림 9-24 룩 스케줄링

8-1. C-LOOK 디스크 스케줄링

C-LOOK 디스크 스케줄링Circular LOOK disk scheduling

- C-SCAN 디스크 스케줄링의 LOOK 버전
- 한쪽 방향으로만 서비스하는 C-SCAN 디스크 스케줄링과 유사한데, 차이점은 더 이상 서비스할 트랙이 없으면 헤드가 중간에서 방향을 바꿀 수 있다는 것
- 헤드가 이동한 총거리 : 1+3+3+5+20+3+1+2=38

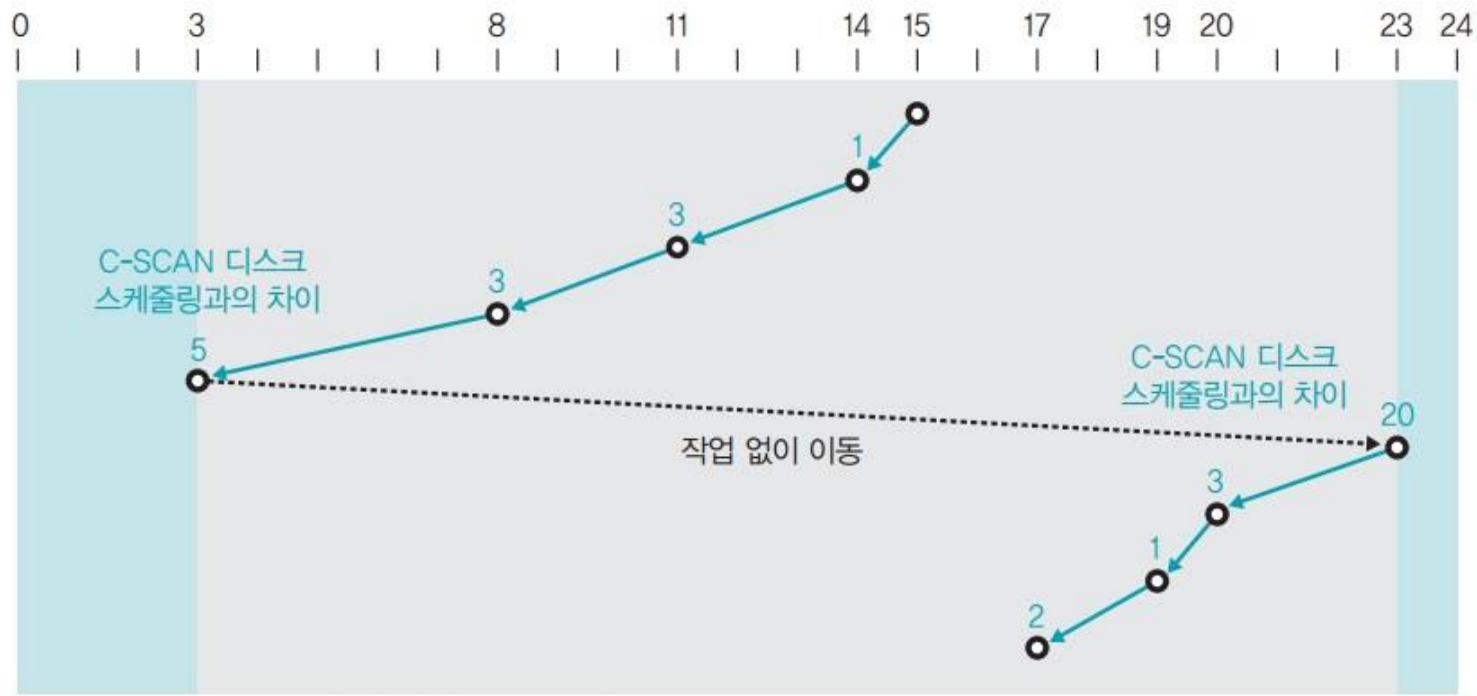


그림 10-32 C-LOOK 디스크 스케줄링의 동작

9. 최소 지연시간 우선 SLTF, Shortest Latency Time First 스케줄링

최소 지연시간 우선 스케줄링

- 모든 요청중 회전 지연시간이 가장 짧은 요청 먼저 처리
- 디스크 헤드가 특정 실린더에 도달했을 때 해당 실린더의 트랙 요청들이 대기하고 있다면, 헤드는 더 이상 움직이지 않고 도착 순서와 관계없이 모든 요청 우선 처리
- 트랙을 일정한 수의 블록으로 나눈 섹터를 토대로 요청들을 섹터 위치에 따라 큐에 넣은 후 가장 가까운 섹터 요청 먼저 처리
- 고정 헤드에서는 탐색을 하지 않으므로 탐색 시간이 없음. 따라서 회전 지연시간만 지연시간이 되므로 드럼처럼 고정 헤드를 사용하면 효과적인 알고리즘
- 섹터 큐잉sector queuing 알고리즘이라고 표현하기도 함

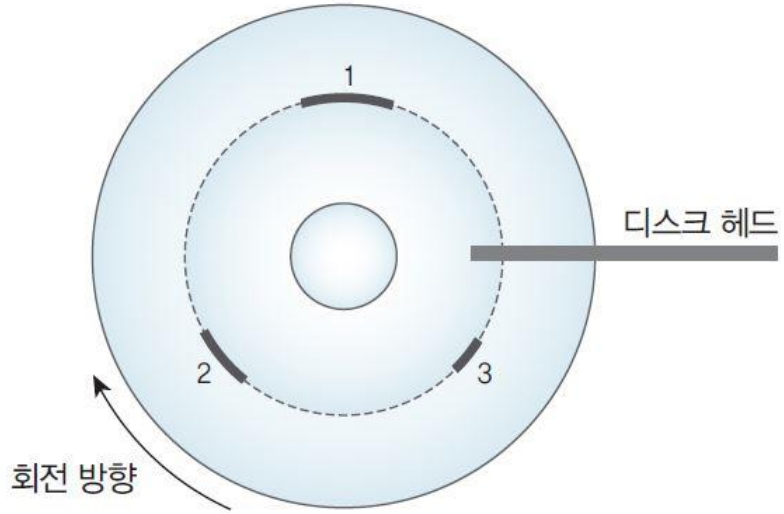


그림 9-25 최소 지연시간 우선 스케줄링

9. 최소 지연시간 우선 SLTF, Shortest Latency Time First 스케줄링

- 선입선처리 스케줄링에서 요청이 모든 섹터에 골고루 분포되어 있다면 예상되는 지연은 0.5 회전
- 섹터 큐의 예

요청이 대기 큐waiting queue의 앞부분에 있지 않더라도 헤드가 지나는 섹터 요청을 서비스하여 처리율 향상 가능

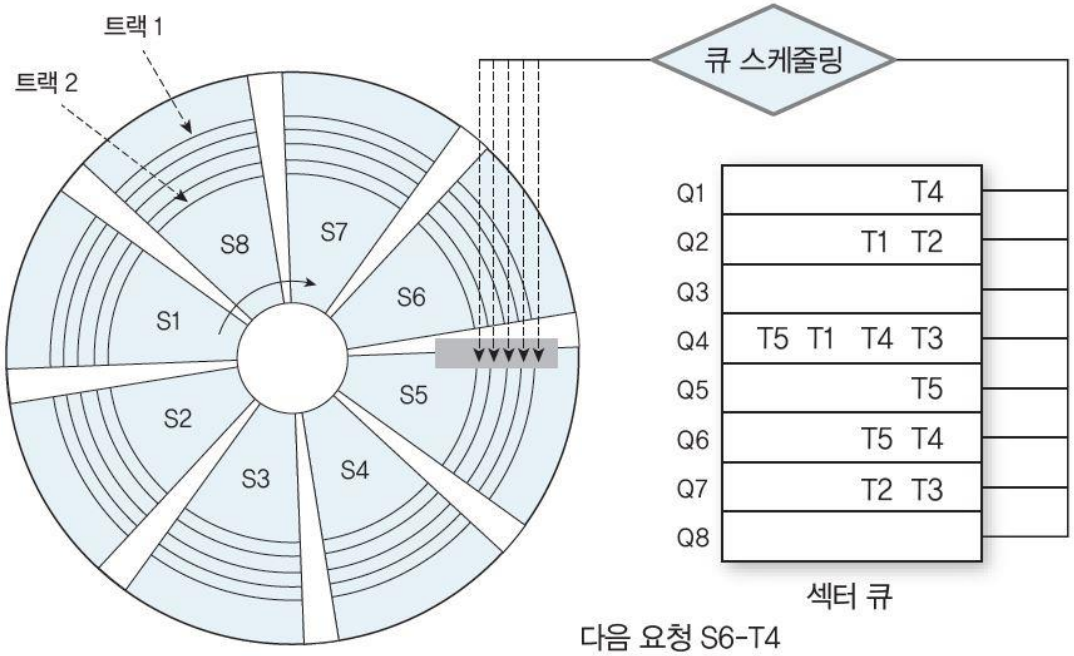


그림 9-26 고정 헤드 디스크를 위한 큐 최적화

섹터 큐잉은 고정헤드장치에서도 사용하지만 특별한 트랙마다 실린더 내에 처리 요청이 하나 이상일 때는 이동헤드장치에서도 쓸 수 있음
헤드가 특정한 실린더에 도착하면 헤드를 더 이상 움직이지 않고 모든 실린더 요청 처리하므로 섹터 큐잉은 동일한 실린더 내에서 다중 요청을 정렬하는 데 사용 가능, 다른 스케줄링 알고리즘과 마찬가지로 섹터 큐잉도 운영체제가 하나 이상의 요청 중에서 헤드 밑에 위치한 첫 번째 요청 선택해야 효과 있음

10. 최소 위치 결정 시간 우선SPTF, Shortest Positioning Time First 스케줄링

최소 위치 결정 시간 우선 스케줄링의 개념

- 가장 짧은 위치 결정 시간, 즉 탐색 시간과 회전 지연시간의 합이 가장 짧은 요청을 다음 서비스 대상으로 선택
- 최소 탐색 시간 우선 스케줄링처럼 처리량이 많고 평균 반응시간 짧음
- 가장 안쪽과 바깥쪽 실린더 요청이 무기한 연기될 가능성
- 에센바흐 방법 : 탐색 시간과 회전 지연시간을 최적화하려고 한 것
  - 헤드는 순환 스캔 스케줄링처럼 진행하나, 요청과 관계없이 트랙이 한 바퀴 회전할 동안 요청을 처리하도록 재배열하는 알고리즘

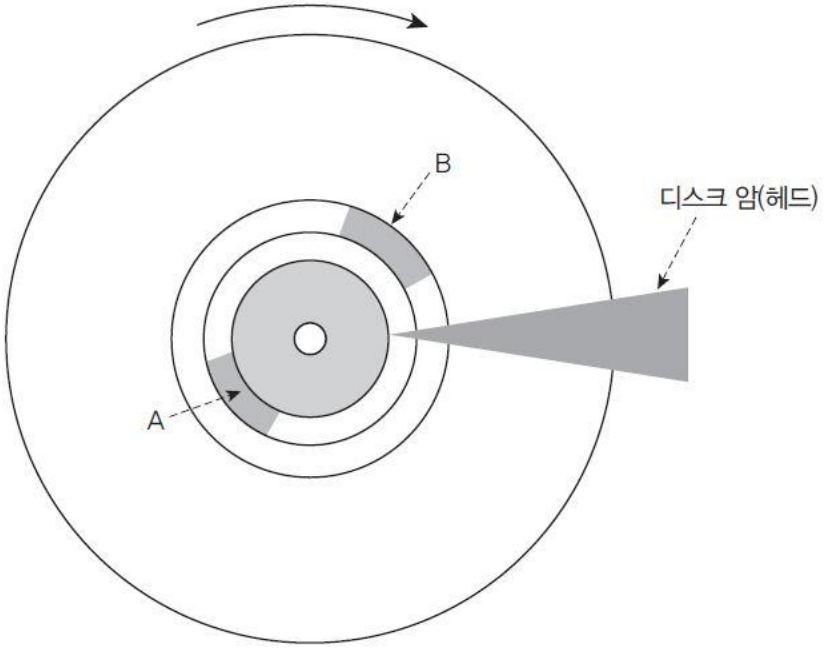


그림 9-27 최소 위치 결정 시간 우선 스케줄링

11. 디스크 스케줄링 알고리즘의 선택

디스크 스케줄링 알고리즘의 선택

- 최소 탐색 시간 우선 스케줄링이 일반적이고 자연스러운 선택이라면, 스캔 스케줄링이나 순환 스캔 스케줄링은 디스크를 많이 사용하는 시스템에 적당
- 최적 알고리즘을 결정하는 것은 가능, 최적 스케줄링에 필요한 계산량 때문에 스캔 스케줄링이나 최소 탐색 시간 우선 스케줄링 이상의 처리 효율 얻을 수 있을지 평가하기 곤란
- 스케줄링 알고리즘 성능은 요청의 형태와 수에 따라 좌우. 큐가 하나 정도밖에 요청을 하지 않는다면 모든 스케줄링 알고리즘 효과 거의 동일. 이때는 선입선처리 스케줄링이 적당함
- 디스크 서비스의 요청은 파일 할당 방법에 많은 영향. 연속적으로 할당된 파일을 읽는 프로그램은 디스크의 인접한 범위 내에서 많은 요청이 발생하여 헤드 이동 제한
- 링크 파일이나 색인 파일은 블록들이 디스크에 흩어져 헤드의 이동 거리는 길지만, 디스크 상용 효율 높음
- 모든 파일은 열어야 사용할 수 있고, 파일을 열려면 디렉터리 구조를 조사해야 하기 때문에 디렉터리를 자주 호출. 따라서 디렉터리 위치에 따라 이동 거리 다름. 디렉터리를 디스크의 양 끝에 두는 것보다는 중간 부분에 두는 것이 디스크 헤드 이동 줄일 수 있음
- 디스크는 컴퓨터의 장치 중에서 가장 속도 느린 장치. 메모리와 제어기 캐시 같은 기술이 성능 증가에 도움이 되지만, 전반적으로 시스템 성능은 디스크의 속도와 신뢰성에 좌우

Section 03 RAID(1. RAID의 소개)

- **RAID Redundant Array of Inexpensive Disks(또는 Redundant Array of Independent Disks 의 개념**
  - 운영체제로 여러 대의 물리적 디스크를 하나의 논리적 디스크로 인식하는 기술
  - 초기의 RAID는 프로세서와 디스크의 속도 차이 극복하려고 디스크 여러 대를 이용 다수의 드라이버에 있는 데이터를 동시 액세스할 수 있도록 데이터를 분산·저장, 이런 RAID 제안은 데이터 중복의 필요성을 효과적으로 해결

2. RAID 계층

RAID 계층

- 하드디스크 여러 개를 논리적 가상 디스크 하나로 구성, 대용량 저장장치로 사용 방법
- 데이터를 하드디스크 여러 개에 분할·저장하여 전송속도를 향상시키고, 시스템 가동 중 생길 수 있는 디스크 오류를 시스템을 정지하지 않고도 교체 가능
- RAID 방법
  - 6계층으로 분류되며, 서로 다른 용도로 사용. 초기 RAID 제안자들이 6개로 구분했던 RAID 각 계층을 좀 더 다양화했으며, 구성과 기능 면에서 다르게 변했기 때문에 사용자 목적에 맞는 RAID 계층 구성이 필요
- RAID의 구성 방법
  - 초기 6계층 외에 RAID 6, RAID 7, RAID 0+1, RAID 50, RAID 53 등 지속적으로 새로운 방법 개발
  - RAID 0, RAID 1, RAID 0+1, RAID 5를 제외한 나머지 방법은 거의 사용하지 않음

2. RAID 계층

RAID 0(스트라이핑)

- 일련의 데이터를 논리적 디스크 배열 하나에 일정한 크기로 나눠서 분산 저장 방법
- 사용자와 시스템 데이터는 논리 디스크 하나에 저장되어 있는 것으로 인식
- 디스크는 크기가 일정한 섹터 또는 물리적 블록 단위인 스트립으로 나눠 연속적인 배열 첨자와 대응되도록 순환 할당
- 스트라이프 : 스트립 하나와 각 배열의 구성 요소가 대응하는 논리적, 연속적인 스트립의 집합
- 대기 중인 다수의 입출력 요구를 병렬적으로 처리하여 높은 전송률 유지 가능
- RAID 0 계층에는 스트라이프가 있지만 데이터를 중복해서 기록하지 않으므로 장애 발생에 대비한 여분의 저장 공간 없음
- 데이터를 입출력할 때 RAID 컨트롤러에서 디스크 여러 개로 나눠서 쓰고 읽어 들임, 중요하지 않은 데이터에서 빠른 데이터 입출력 성능을 요구하는 동영상 편집 등에 적합

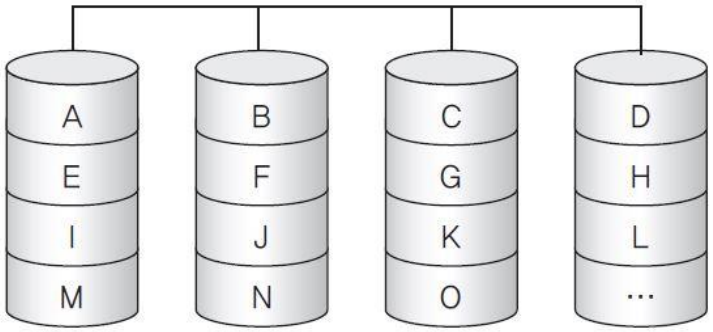


그림 9-28 RAID 0(스트라이핑)

2. RAID 계층

RAID 1(미러링)

- RAID 0처럼 데이터 스트라이핑을 사용하면서 배열 내의 모든 디스크에 동일한 데이터가 있는 미러 디스크를 가짐
- 각 논리적 스트립은 별도의 디스크 2개에 대응하므로 미러링 이라고도 함
- 중복 저장된 데이터가 적어도 2개가 있는 드라이브로 구성
- 읽기 요구는 요구 데이터가 있는 디스크 2개 중 어떤 디스크에서도 서비스를 받을 수 있기 때문에 성능 향상
- 데이터를 분할 저장하지 않기 때문에 두 스트립을 갱신해야 함
- 쓰기 요구는 단일 디스크 드라이브와 같음(병렬적으로 갱신)
- 시스템 드라이브와 같은 중요한 파일에 적합

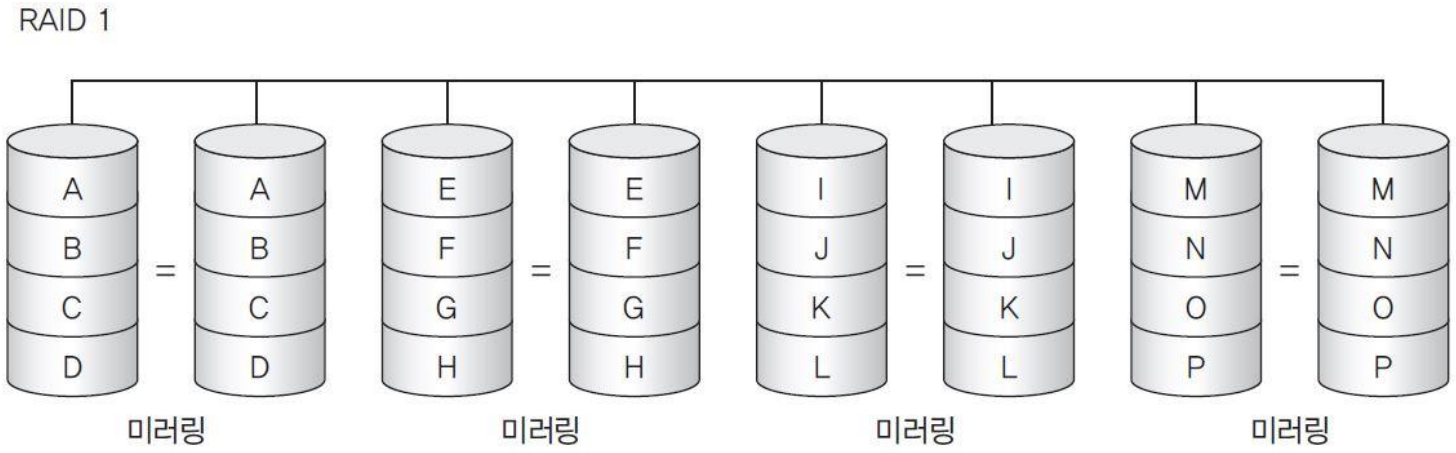


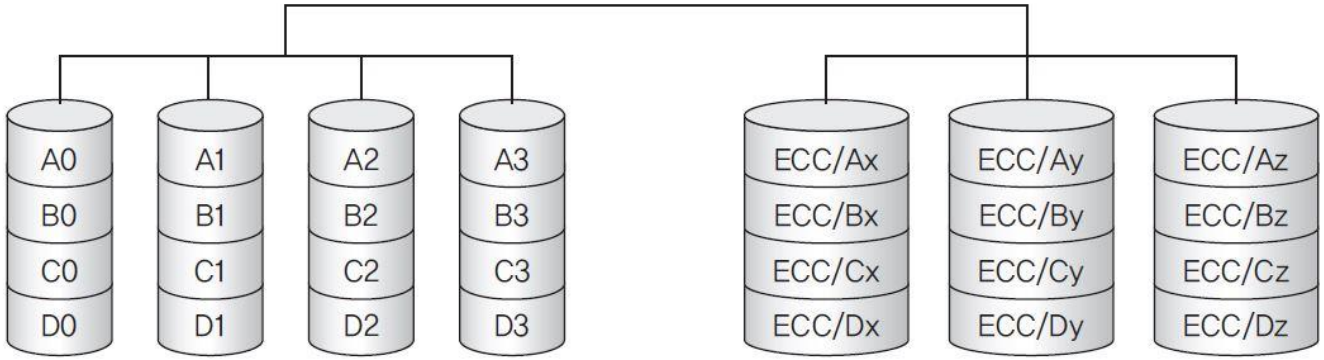
그림 9-29 RAID 1(미러링)

2. RAID 계층

RAID 2(허밍 코드를 이용한 중복)

- 병렬 액세스 방법을 사용하므로 모든 멤버 디스크는 모든 입출력 요구에 참여
- 디스크 간에 데이터 스트라이핑을 사용, 오류 검출 능력이 없는 드라이브는 허밍코드 방법 사용
  - 허밍 코드
    - 패리티 비트를 필요한 수만큼 정해진 위치에 두어 오류가 발생했을 때 오류 발생 비트 알아내어 정정이 가능
    - 허밍 코드의 비트 수 : 정보 비트의 수가 m이면 패리티 비트 수 p는 $2p >= m + p + 1$
    - 패리티 비트의 위치 : 허밍 코드의 왼쪽부터 1, 2, 4, 8, 2n-1에 위치
- 허밍 오류 정정 코드는 패리티 비트를 사용하여 디스크에서 전송된 데이터에 오류가 있는지 확인하여 오류 정정

2. RAID 계층



A0 to A3 = Word A; B0 to B3 = Word B;
C0 to C3 = Word C; D0 to D3 = Word D;

ECC/Ax to Az Word A ECC; ECC/Bx to Bz Word B ECC;
ECC/Cx to Cz Word C ECC; ECC/Dx to Dz Word D ECC;

그림 9-30 RAID 2(허밍 코드를 이용한 중복)

데이터 A를 A0 · A1 · A2 · A3비트로 분할하여 기록하고, 패리트 비트(허밍 코드)들 은 ESS(패리티) 디스크의 대응하는 위치에 ECC/Ax · Ay · Az로 저장, 각 데이터는 ESS 디스크에 저장된 허밍 코드를 이용하여 오류 검출, 정정

- 실시간으로 오류 수정, 현재의 SCSI 드라이브에는 자체적으로 오류 검출 능력이 있어 별로 사용 않음
- 디스크에 오류 발생이 많을 때는 효과 있지만, 디스크의 신뢰성이 높으면 낭비가 크므로 구현 자체가 제외될 때 많음
- 빠른 기록 속도와 함께 장애 복구 능력 요구할 때 사용, 드라이브 두 대만으로 구성 할 수 있기 때문에 작은 시스템에 적합

2. RAID 계층

RAID 3(비트 인터리브된 패리티)

- 데이터를 분산 저장할 때는 스트라이프를 사용, 오류 검출과 수정할 때는 별도의 드라이브 한 대를 패리티 드라이브로 사용
- 내장된 오류 정정 코드 정보 오류 감지하는 데 사용, 데이터 복구는 다른 드라이브에 기록된 정보 XOR 계산 수행, 각 데이터 디스크에 데이터를 비트 단위로 스트라이핑하여 기록
- 디스크에 입출력할 데이터 크기가 K바이트라면, 디스크 N개에 나눠(K/N) 분산 저장하고, N+1 디스크에는 패리티 비트 저장. 바이트 단위의 분산 저장을 경제적 수행하려면 하드웨어적인 지원 필요
- 데이터는 스트립들로 나누므로 높은 데이터 전송률 제공하나(병렬적 데이터 전송 가능), 입출력 작업은 드라이브에서 모두 처리하므로 파일 서버등 입출력이 빈번한 곳에서 사용 부적합(입출력 요청을 한 번에 한 요청만 실행하기 때문)
- 대형 레코드를 많이 사용하는 단일 사용자 시스템과 다량의 데이터 전송을 요구하는 CAD나 이미지 작업에는 적합

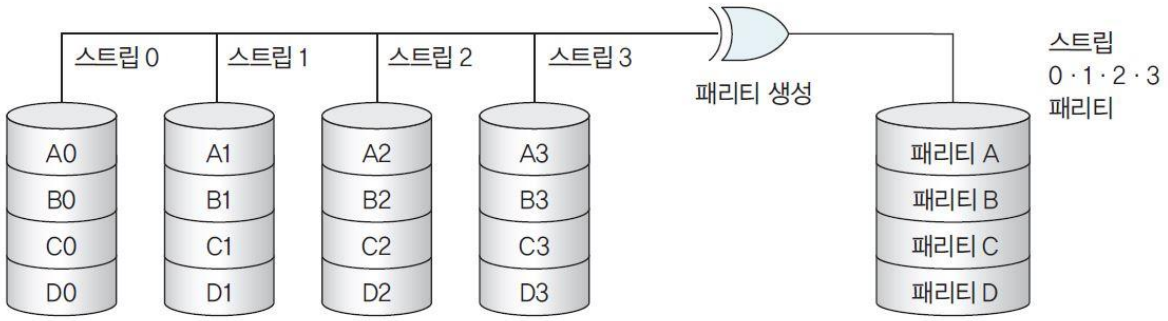


그림 9-31 RAID 3(비트 인터리브된 패리티)

2. RAID 계층

RAID 4(블록 인터리브된 패리티)

- RAID 4와 RAID 5는 독립된 액세스 방법 사용.
- 패리티 드라이브를 사용한다는 점에서 RAID 3과 비슷하지만, 각 드라이브에 데이터를 블록 단위로 분산 저장한다는 것이 다름
- 다른 방법처럼 데이터를 분산 저장 위해 스트라이핑 사용
- 독립된 액세스 방법 사용하여 디스크의 각 멤버는 독립적으로 동작, 각각 분리된 입출력 요구들을 병렬로 처리 가능, 많은 입출력 요청이 필요한 업무에 적합, 높은 데이터 전송률 필요 업무에서는 성능 떨어짐
- 크기가 작은 입출력을 요청할 때 쓰기 성능 떨어짐. 쓰기 요청이 있을 때마다 사용자 데이터와 함께 패리티 정보를 갱신해야 하기 때문에 추가 시간 소요(입출력의 병목현상)
- 데이터 읽기 요청은 RAID 0과 비슷한 성능. 여러 드라이브 중에서 드라이브 한 대만 여분의 패리티 정보를 기록하는데 사용하기 때문에 용량당 비용 높지 않음. 저렴한 가격으로 장애 복구 능력을 요구하거나 빠른 판독 속도 필요할 때 사용

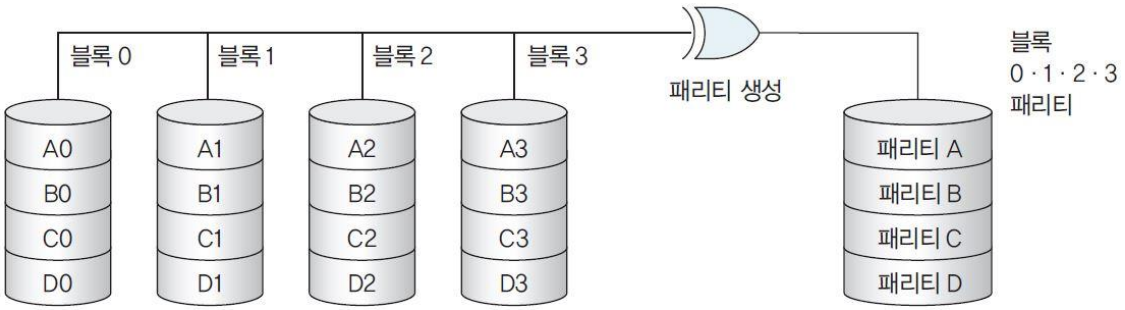


그림 9-32 RAID 4(블록 인터리브된 패리티)

2. RAID 계층

RAID 5(블록 인터리브된 분산 패리티 블록)

- RAID 4 구성과 비슷, 별도의 패리티 드라이브 대신 모든 드라이브에 패리티 정보 나눠 저장하기 때문에 패리티를 담당하는 디스크의 병목 현상 일으키지 않음
- 다중 프로세스 시스템처럼 작고 잦은 데이터 기록이 있을 때 더 빠름
- 읽기 요청은 각 드라이브에서 패리티 정보를 건너 뛰어야 하기 때문에 RAID 4보다 느림
- 최소한 세대, 보통은 다섯 대 이상의 드라이브 필요, 데이터는 각 데이터 영역에 블록 단위로 스트라이핑하여 저장
- 병렬 입출력이 가능하기 때문에 기록과 읽기가 동시에 가능, 데이터 입출력 성능이 아주 빠르면서도 안전성 또한 높은 편으로 파일 서버 등 입출력이 빈번한 업무에 적합
- 현재 가장 널리 사용

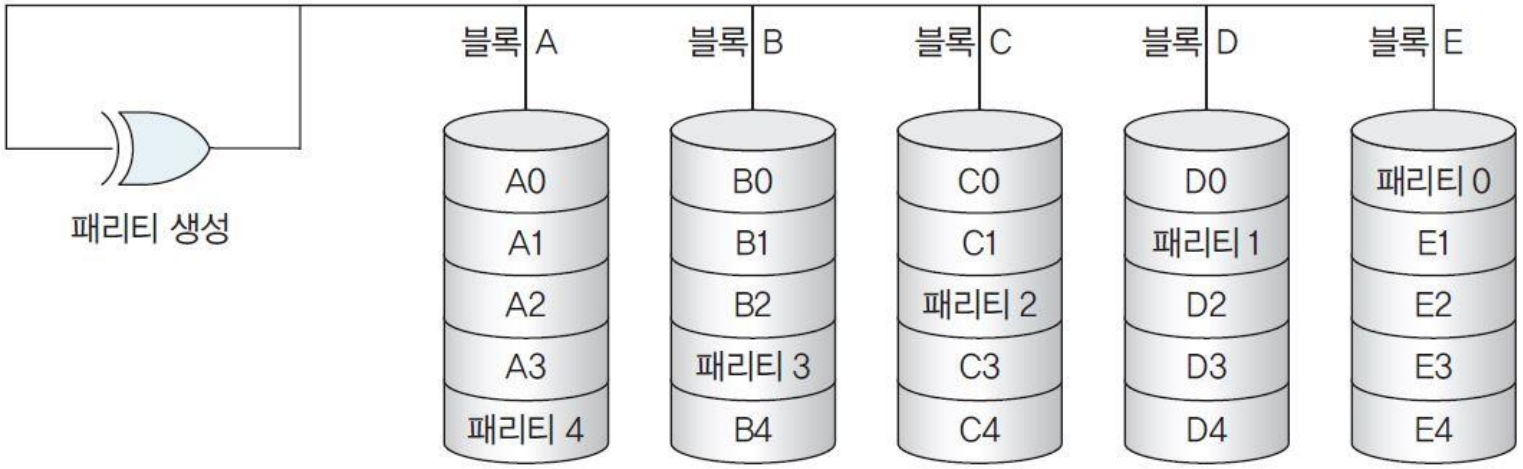


그림 9-33 RAID 5(블록 인터리브된 분산 패리티 블록)

2. RAID 계층

RAID 6

- RAID 5와 같은 방식이지만 패리티 비트가 2개여서 디스크 2개의 장애를 복구할 수 있음
- 패리티 비트를 2개씩 운영하기 때문에 RAID 5보다 계산량이 많고 4개의 디스크당 2개의 추가 디스크가 필요하다는 단점이 있음

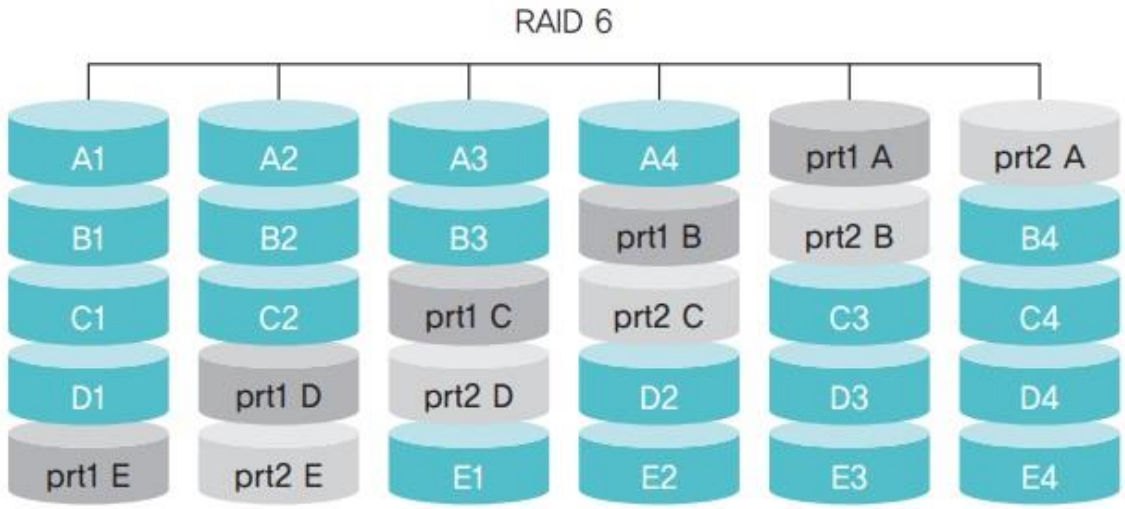


그림 10-41 RAID 6의 구조

2. RAID 계층

RAID 0+1

- 스트라이핑 방법(RAID 0의 빠른 속도)과 미러링 방법(RAID 1의 안정적인 복구 기능) 혼합
- 미러링은 반드시 디스크 수가 똑같아야 하기 때문에 디스크 2개를 사용하여 스트라이핑하려면 최소 디스크가 네 개 필요
- 쓰기 속도는 디스크 2개로 스트라이핑할 때와 동일, 디스크 4개에서 나눠서 읽어 오기 때문에 읽기 속도 빠름.
- 미러링으로 동일한 디스크 복사본 가지고 있어 디스크에 오류 발생하면 복구 가능
- 디스크가 여럿 필요하여 안정성과 빠른 속도가 모두 필요한 중대형 서버에서 많이 사용

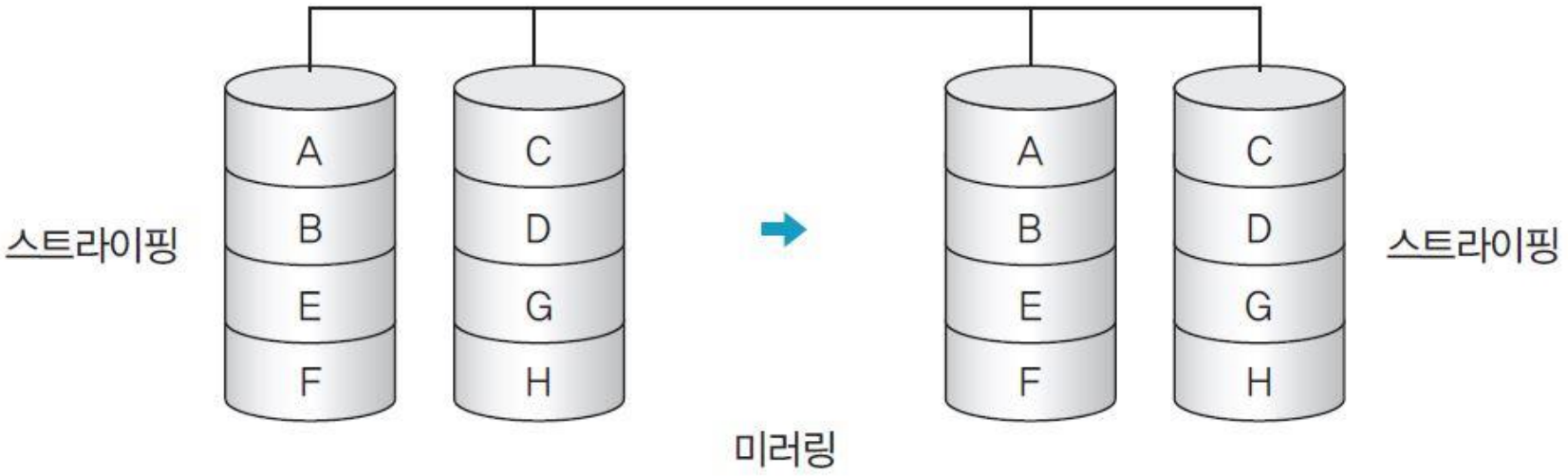


그림 9-34 RAID 0+1

2. RAID 10

RAID 10

- 미러링 기능을 가진 RAID 1과 빠른 데이터 전송이 가능한 RAID 0을 결합한 형태
- 디스크를 RAID 0으로 먼저 묶으면 RAID 0+1이 되고, RAID 1로 먼저 묶으면 RAID 10이 됨
- RAID 0+1의 경우 장애가 발생 했을때 복구하기 위해 모든 디스크를 중단해야 하지만 RAID 10은 일부 디스크만 중단하여 복구할 수 있음

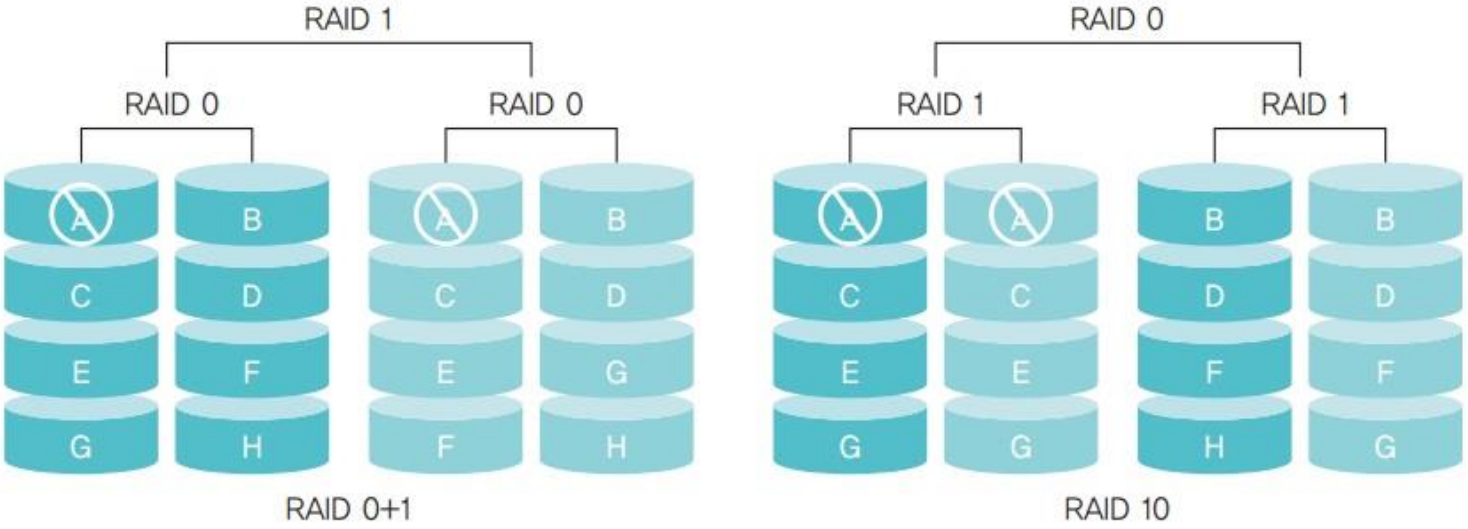


그림 10-42 RAID 0+1과 RAID 10의 구조

2. RAID 50과 RAID 60

RAID 50과 RAID 60

- RAID 50 : RAID 5로 묶은 두 쌍을 다시 RAID 0으로 묶어 사용
- RAID 60 : RAID 6으로 묶은 두 쌍을 다시 RAID 0으로 묶어 사용

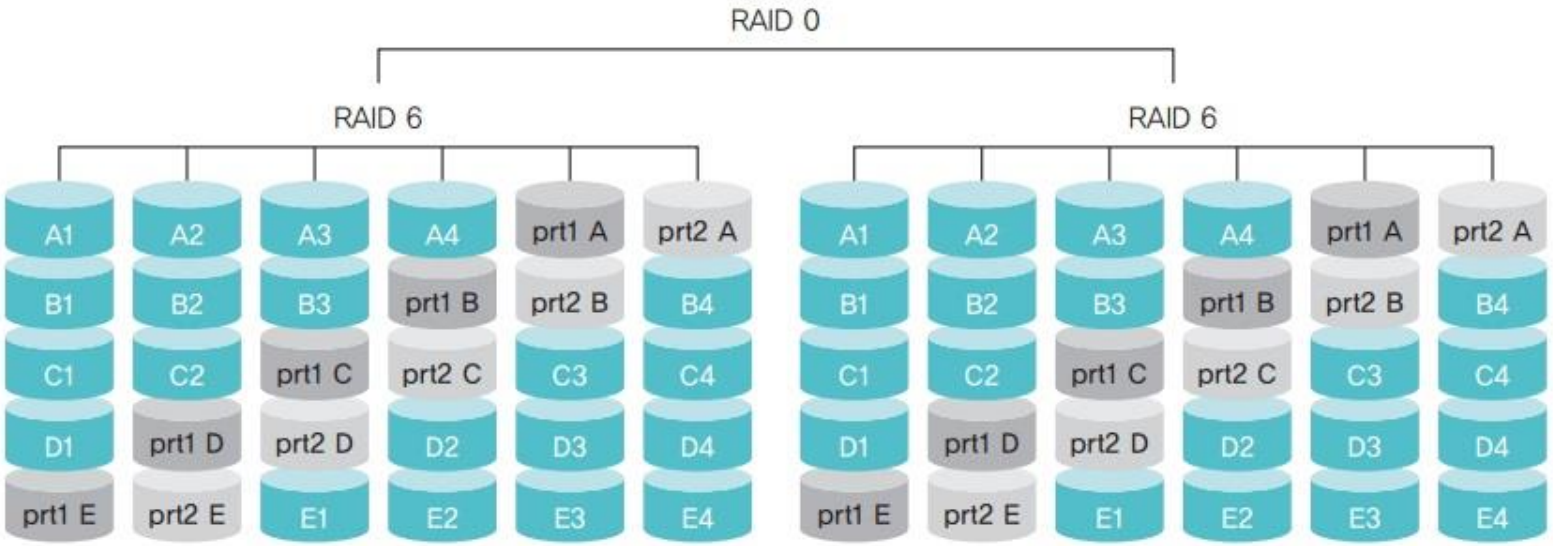


그림 10-43 RAID 60의 구조