22023 운영 체제

I/O 관리 및 디스크 스케줄링

제11장

심재홍 님,
컴퓨터 공학부,
조선대학교 CHOSUN UNIVERSITY

I/O 수행

프로그래밍된 I/O

프로세스가 작업이 완료될 때까지 바쁘게 대기 중입니다.

인터럽트 기반 I/O

I/O 명령이 발행됨

프로세서는 계속해서 명령을 실행합니다.

I/O 모듈은 완료되면 인터럽트를 보냅니다.

I/O 수행

직접 메모리 액세스(DMA)

DMA 모듈은 데이터 교환을 제어합니다.
주기억장치와 I/O 장치 사이

전체 블록이 전송된 후에만 프로세서가 중단됩니다.

기술 간의 관계

Table 11.1 I/O Techniques

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

직접 메모리 액세스

프로세서는 I/O 작업을 프로세서에 위임합니다.
DMA 모듈

read: device에 데이터가 들어오면 주소 xxxx 번지에 4KB까지 저장하라.

write: 주소 xxxx번지에 4KB의 데이타를 device로 보내라.

DMA 모듈은 메모리로 또는 메모리에서 직접 데이터를 전송합니다.

완료되면 DMA 모듈은 프로세서에 인터럽트 신호를 보냅니다.

DMA

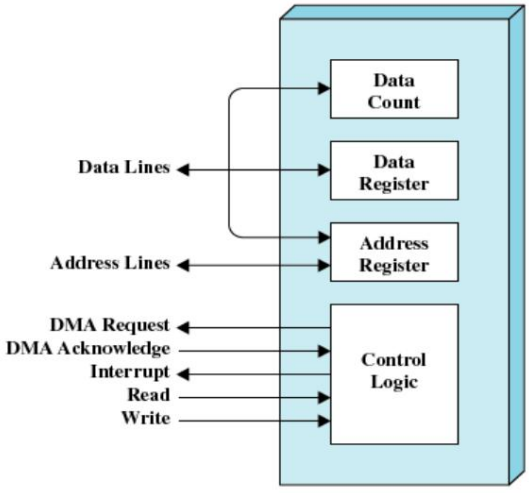
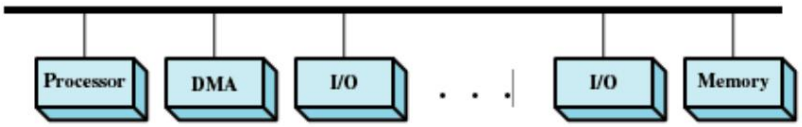
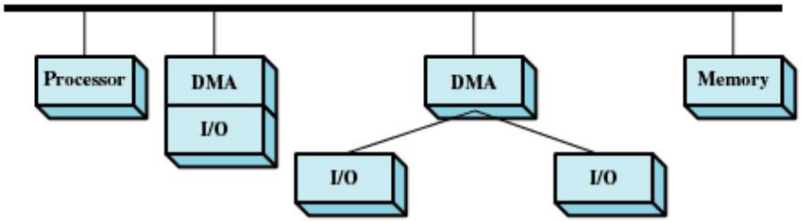


Figure 11.2 Typical DMA Block Diagram

DMA 구성

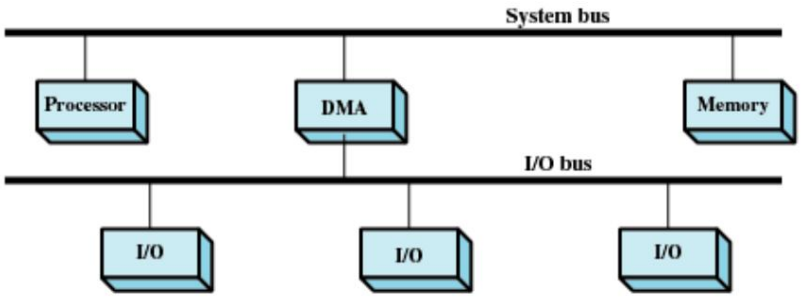


(a) Single-bus, detached DMA



(b) Single-bus, Integrated DMA-I/O

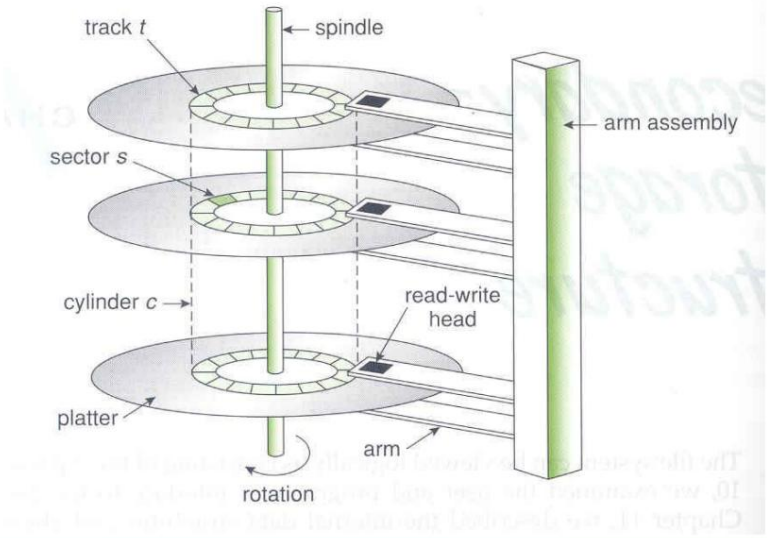
DMA 구성



(c) I/O bus

Figure 11.3 Alternative DMA Configurations

섹터, 트랙, 실린더



섹터, 트랙, 실린더

Sector: 가장 작은 데이터 전송 단위 (page 크기와 동일)

512B, 1KB, 4KB, 8KB 단위 (크기는 OS마다 다름)

Track : sector들의 집합(디스크 상의 하나의 원)

Cylinder : 동일한 위치에 있는 track들의 집합

디스크 판(platter)이 4개일 경우 위/아래 면에 각각 1개씩의 track이 있으므로 8개의 tracks으로 하나의 cylinder를 구성

디스크 판에 track이 200개 있다면 200개의 cylinder가 존재, 각 cylinder에는 8개의 tracks이 존재

하나의 디스크를 두개의 partitions(C:, D:)으로 나눌 때 cylinder 단위로 나눔, C: (0 ~ 50 cylinders), D: (51 ~ 199 cylinders)

Cylinder를 구성하는 각 track에 read/write head가 따로 존재 -> 8개의 트랙에 동시에 읽고 쓰는 것이 가능

디스크 성능 매개변수

읽거나 쓰려면 디스크 헤드가
원하는 트랙과 원하는 섹터의 시작 부분에 위치

디스크 I/O 시간

대기열 시간 (장치 대기) + 채널 대기 시간 (채널 대기) + 검색 시간 + 회전 지연(대기 시간) + 데이터 전송 시간

디스크 I/O 전송 타이밍

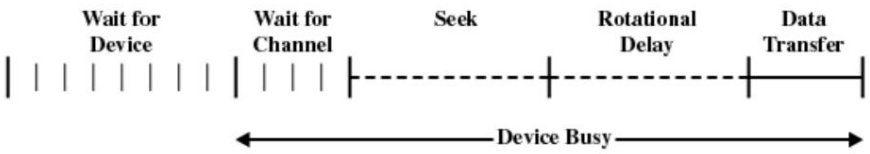


Figure 11.6 Timing of a Disk I/O Transfer

디스크 성능 매개변수

탐색 시간

원하는 트랙에 헤드를 위치시키는 데 걸리는 시간(헤드
가 해당 트랙 찾아가는 시간)

Ts = m n + s

— Ts = 예상 탐색 시간, n = 통과한
트랙 수, m = 디스크 드라이브에 따라 달라
지는 상수, s = 시작 시간

Ø 저렴한 디스크 : m = 0.3, s = 20ms

— 비싼 디스크 : m = 0.1, s = 3ms

파일 데이터가 연속된 tracks에 저장되어 있을 경우, 첫
트랙을 제외한 나머지 track들의 seek time은 0라 가정

디스크 성능 매개변수

회전 지연 또는 회전 대기 시간

섹터의 시작이 다음 단계에 도달하는 데 걸리는 시간
머리

Tr = 1 / (2r) = 평균시간(디스크 반 바퀴 회전시간)

Ø Tr = 평균 회전 지연 시간, r = 초당 회전수로 표
시되는 회전 속도

= 초당 디스크 회전 수

1/r = time to rotate once (한 바퀴 회전 시간)

= 한 트랙 전체를 읽는(쓰는) 시간과 동일

Disk : 3600 rpm(분당회전수), 16.7 ms/rot, Tr = 8.3 ms

파일 데이터가 연속된 tracks에 저장되어 있을 경우에도 각
트랙마다 rotation delay를 적용해야 함

디스크 성능 매개변수

데이터의 전송 시간

Time its takes while desired sector moves under the head (디스크의 회전 시간과 동일)

Tt = b / (rN) = 1/r * (b/N) = (한 바퀴 회전 시간) * (읽어낸 데이터 용량 b / 한 트랙 용량 N)

한 트랙을 읽는 시간 = 한 바퀴 회전 시간

한 트랙이 32 sectors라면 16 sectors 읽는 시간은?
반 바퀴 회전 시간과 동일함

한 트랙의 데이터 용량이 1MB라면 1KB를 읽는 시간은? 한 바퀴 회전 시간의 1/10

Tt = 전송 시간, b = 전송될 바이트 수
N = 트랙의 바이트 수, r = 초당 회전수로 표시되는 회전 속도

디스크 성능 매개변수

입장 시간

탐색 시간과 회전 지연의 합

책을 읽거나 읽을 수 있는 자세를 취하는 데 걸리는 시간 쓰다

총 접속 시간

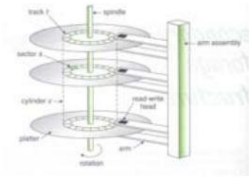
Ta = Ts + Tr + Tt = Ts + 1/(2r) + b/(rN)

타이밍 비교

섹터가 헤드 아래로 이동할 때 데이터 전송이 발생합니다.

파일이 동일한 실린더 및 인접 섹터 에 저장되면 전체 파일 의 데이터 전송 속도 가 더 빨라집니다.

평균에 의존하는 것의 위험성을 설명합니다.
가치



타이밍 비교

디스크 사양

평균 탐색 시간 : 20 ms

전송속도 : 960KB/s

섹터당 512바이트, 트랙당 32섹터 : 16KB

A transfer rate = 960KB/s = 한 트랙 용량 * 회전 수(r) =
960 * 32/1024한 트랙 읽는 시간 r = 60 (초당 회전 수)

간 = 한 바퀴 회전시간 = 1/60 sec = 16.7ms

Rotational delay = 1/(2r) = 1/(2*60) = 8.3ms (반 바퀴 회전시간)

1 MB 크기의 파일을 읽는다고 가정

1MB / 512(sector크기) = 2048 sectors

Ø 2048개 섹터 / 32(섹터/트랙) = 64개 트랙

64개의 tracks상의 2048 sectors로 구성되어 있음

타이밍 비교

순차적 구성(단일 표면)

해당 파일이 한 platter 표면의 64개의 tracks (2048 sectors)에 연속으로 저장되어 있는 경우임

첫 번째 트랙

45 ms = 20 ms (average seek) + 8.3 ms (rotational delay) + 16.7 ms (한 트랙 읽기)

한 트랙을 읽는 시간 = 한 바퀴 회전시간

후속 각 63개 트랙

(연속된 track이므로 seek time이 필요 없음)

25 ms = 8.3 ms (rotational delay) + 16.7 ms (한 트랙 읽기)

총 접속 시간

45ms + 63개 트랙 * 25ms = 1620ms = 1.6초

타이밍 비교

랜덤 액세스(단일 표면)

해당 파일의 2048 sectors가 한 platter 표면에 랜덤으로 흩어져 저장되어 있는 경우

각 부문

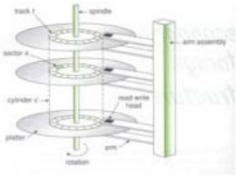
28.8ms = 20ms(평균 탐색) + 8.3ms(회전 지연) + 0.5ms(1개 섹터 읽기)

한 섹터를 읽는 시간 =
1sector / 32sectors(한 트랙) *
(한 바퀴 회전시간 16.7ms)

총 접속 시간

2048개 섹터 * 28.8ms = 58982ms = 59초

타이밍 비교



순차적 병렬 액세스

해당 파일이 디스크의 연속된 cylinders에 연속된 tracks에 저장된 경우 (가장 최적의 조건임)

한 cylinder는 4개의 platters, 즉 8 surfaces에 각각 한 개씩 있는 track으로 총 8개의 tracks 구성됨

파일은 총 64 tracks으로 구성되므로, 한 cylinder에 8 개 tracks씩 저장되면 총 8개의 연속 cylinder에 저장됨

First cylinder(8 tracks) = 45 ms = 20 ms (average seek) + 8.3 ms (rotational delay) + 16.7 ms (한 실린더 읽기)

Each succeeding 7 cylinders = 25ms = 8.3ms (rotational delay) + 16.7 ms (한 실린더 읽기)

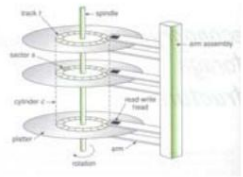
한 트랙 또는 한 실린더를 읽는 시간 = 한 바퀴 회전시간

(연속된 cylinder(track)이므로 seek time이 필요 없음)

Total access time = 45ms + 7 동시에 25ms = 220ms

8개의 tracks을

타이밍 비교



디스크 사양/순차병렬접속

평균 탐색 시간 : 10 ms

디스크는 6초 동안에 1000번 회전

한 바퀴 회전시간: 6000ms/1000번 = 6ms

4096 bytes per sector, 32 sectors per track

8 surfaces를 가짐(한 cylinder가 8개의 tracks으로 구성되었음을 의미)

10 MB 크기의 파일을 읽는다고 가정 (연속된 실린더에 저장되어 있음)

Rotational delay = 반 바퀴 회전시간 = 3ms

한 트랙의 transfer time = 한 cylinder transfer time = 디스크 한 바퀴 회전 시간 = 6ms

한 트랙 저장 용량 = 4096(212) x 32(25)=217bytes = 27KB = 128KB

한 cylinder 저장용량 = 128KB(217bytes) x 8(23tracks) = 220bytes = 1MB

따라서 위 파일은 총 10개의 cylinder에 연속으로 저장되어 있음

첫 cylinder를 읽는 total access time은? 나머지 9개 cylinder 를 읽는 total access time은? 파일을 읽는 total access time?

디스크 스케줄링 정책

Seek time이 성능을 좌우함

각 track 내의 특정 sector를 읽으려고 할 때, 어떤 track이든 평균적인 rotational delay와 그 sector를 읽는 시간은 동일함

따라서 여러 tracks에 대한 요청이 들어 왔을 때, 각 트랙의 seek time을 최소화하는 disk scheduling 알고리즘이 필요함

단일 디스크의 경우 여러 I/O 요청이 있습니다.

200개의 트랙이 있는 디스크를 가정합니다.

맨 바깥쪽 트랙이 0, 제일 안쪽 트랙이 199

현재 트랙 100 에서 시작하여 증가하는 방향 으로 track number (헤드가 들어가는 중)

요청된 트랙은 수신된 순서대로 : 55, 58, 39, 18, 90, 160, 150, 38, 184

디스크 스케줄링 정책

무작위 스케줄링

대기열에서 무작위 순서로 요청을 선택합니다.

최악의 성능

다른 기술을 평가하기 위한 벤치마크로 유용함

디스크 스케줄링 정책

우선순위

목표는 디스크 사용을 최적화하는 것이 아니라 다른 목표를 달성하는 것입니다.

단기 배치 작업의 우선순위가 더 높을 수 있음

좋은 대화식 응답 시간 제공

디스크 스케줄링 정책

후입선출

거래 처리 시스템에 적합

- 가장 최근 사용자에게 장치를 제공하므로 팔의 움직임이 거의 없어야 함

직업을 가질 수 없기 때문에 기아 가능성

선두를 되찾다

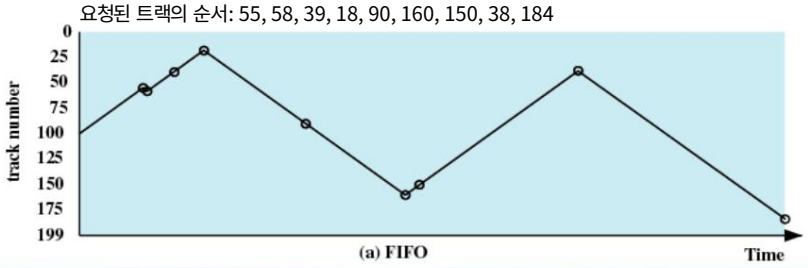
디스크 스케줄링 정책

선입선출(FIFO)

요청을 순차적으로 처리

모든 프로세스에 공정함

무작위 스케줄링에 접근합니다.
프로세스가 많은 경우 성능

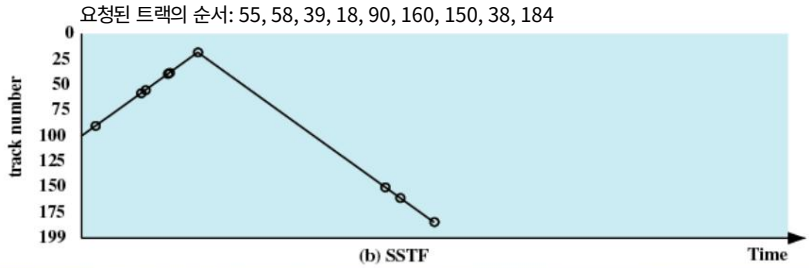


디스크 스케줄링 정책

최단 서비스 시간 우선

현재의 헤드 위치와 가장 가까운 track들을 먼저 서비스함

현재 위치에서 디스크 암의 움직임이 가장 적은 디스크 I/O 요청을 선택합니다. 항상 최소 검색 시간을 선택합니다.



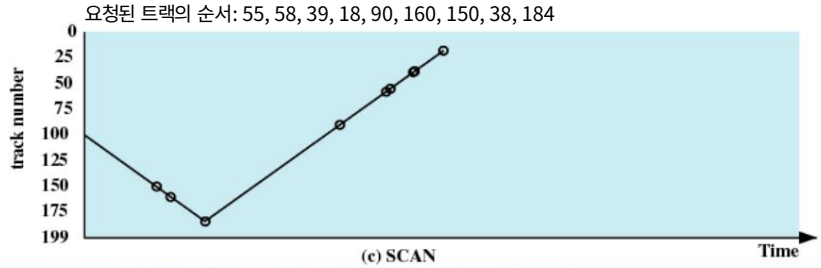
디스크 스케줄링 정책

스캔

헤드가 들어가면서 순서적으로 track 들을 서비스하고
나오면서도 순서적으로 track 들을 서비스함

암은 한 방향으로만 움직이며 해당 방향의 마지막 트랙에 도달할 때까지 모든 미해결 요청을 충족한 다음

방향이 반대이다



디스크 스케줄링 정책

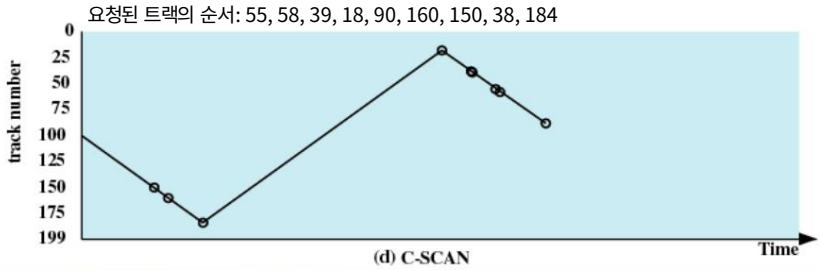
C-스캔

헤드가 들어가면서 순서적으로 track들을 서비스하고,
나올 때는 서비스하지 않음. 나온 후 다시 들어가면서 서비스 함

한 방향으로만 스캔을 제한합니다.

한 트랙에서 마지막 트랙을 방문한 경우

방향으로 팔이 디스크의 반대쪽 끝으로 돌아가고 스캔이 다시 시작됩니다.



디스크 스케줄링 정책

SSTF, SCAN 및 C-SCAN 팔
은 상당 기간 동안 움직이지 않을 수 있습니다.
시간

왜요? 80, 80, 80, 81, 150, 81, 82, 180 , 80, 80, 155, 80, 81,79,82,81, 솔루션

Ø FSCAN, N-스텝-SCAN

앞의 서비스 요청 tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184
이들 중 주어진 알고리즘에 의해 첫 번째 track이 서비스되는 도중에 다음의 새로운 track들의 요청이 들어 왔다 가정하자. 85, 100, 155, 165, 80

디스크 스케줄링 정책

FSCAN

Has two Queues
1st Queue: 55, 58, 39, 18, 90, 160, 150, 38, 184 2nd Queue: 85, 100, 155, 165, 80 첫 번째 큐
가 SCAN에 의해 서비스 시작한다. 서비스 중 새로 도착한 request들은 모두 두 번째 큐에 저장 첫 번째 큐 서비스가 끝나면 이때의
마지막 track번호와 헤드 의 움직이는 방향을 고려하여 두 번째 큐가 SCAN에 의해 서 비스 된다.

두 번째 큐가 서비스 되는 도중에 도착하는 새로운 request는 다시 첫 번째 큐에 저장된다. 두 번째 큐 서비
스가 끝나면 다시 첫 번째 큐가 SCAN에 의해 서비스 된다.

디스크 스케줄링 정책

FSCAN

1차 테일 : 55, 58, 39, 18, 90, 160, 150, 38, 184

2차 테일 : 85, 100, 155, 165, 80

1st Queue가 SCAN에 의해 서비스

- Ø 150, 160, 184, 90, 58, 55, 39, 38, 18

이후 2nd Queue가 SCAN에 의해 서비스

1st Queue의 마지막 서비스 트랙번호 18, 헤드는 나오는 중 (즉, 트랙번호가 감소하는 방향)

- Ø 80, 85, 100, 155, 165

디스크 스케줄링 정책

N-스텝-스캔

디스크 요청 대기열을 길이 N 의 하위 대기열로 분할합니다.

하위 대기열은 다음을 사용하여 한 번에 하나씩 처리됩니다.
주사

하위 대기열이 처리되는 동안 다른 하위 대기열에 새 요청을 추가해야 합니다.

디스크 스케줄링 정책

4-step-SCAN

큐에 도착한 tracks들을 4개씩 나눈 후 SCAN방식으로 service함

[55, 58, 39, 18], [90, 160, 150, 38], [184] SCAN

에 의해 첫 번째 sub-queue의 첫 번째 track(58)을 서 비스하는 도중, 새로 requests(85, 100, 155, 165, 80)가 도착 했으므로,

sub-queue들은 [55, 58, 39, 18], [90, 160, 150, 38], [184, 85, 100, 155], [165, 80]로 변한다. 각

서브 큐를 SCAN으로 서비스함 서비스 순서

[58, 55, 39, 18], [38, 90, 150, 160], 헤드는 들어 가는 중 [184, 155, 100, 85], 헤드 나오는 중 [80, 165]

디스크 스케줄링 정책

첫 번째 requests: 55, 58, 39, 18, 90, 160, 150, 38, 184 디

스크가 90, 100 트랙 순으로 서비스를 막 끝냈음(헤드 들어가는 중) 각 알고리즘

에 의해 첫 requests의 첫 번째 선택된 track이 서비스 되는 도중에 아래의 두 번째 requests가 들어 왔다.

두 번째 requests: 85, 100, 155, 165, 80

각 알고리즘의 첫 번째 서비스되는 track은 첫 번째 requests 에서 선택해야 함

FIFO

55, 58, 39, 18, 90, 160, 150, 38, 184, 85, 100, 155, 165, 80 SSTF

90, 85, 80, 100, 58, 55, 39, 38, 18, 150, 155, 160, 165, 184

스캔

150, 155, 160, 165, 184, 100, 90, 85, 80, 58, 55, 39, 38, 18

C-스캔

150, 155, 160, 165, 184, 18, 38, 39, 55, 58, 80, 85, 90, 100

RAID

독립 디스크 의 중복 배열

운영 체제에서 단일 논리 드라이브로 간주되는 물리적 디스크 드라이브 세트

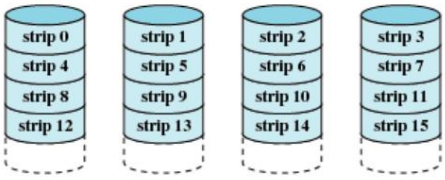
가격이 싼 여러 개의 디스크를 모아 서로 연결하여 용량이 큰 하나의 디스크인 것처럼 사용함 (하나의 C: 드라이브)

데이터는 물리적 드라이브에 분산되어 있습니다.

an array (한 파일의 블록들은 여러 개의 디스크에 순서적으로 돌아가면서 나누어 저장됨)

중복 디스크 용량은 패리티 정보를 저장하는 데 사용됩니다.

RAID 0(비중복)



(a) RAID 0 (non-redundant)

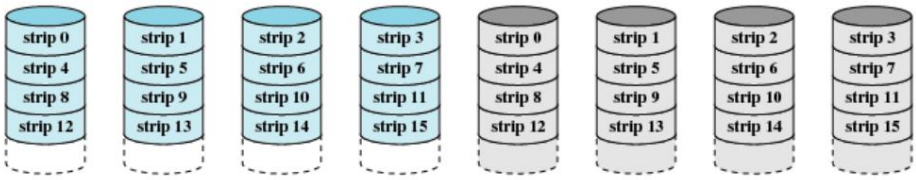
논리 디스크는 스트립으로 구분됩니다.

스트립 : 물리적 블록, 섹터 또는 기타 단위

스트립은 라운드 로빈으로 연속적인 배열 구성원에 매핑됩니다.

스트라이프(Stripe): 정확히 하나의 스트립을 각 배열 구성원에 매핑하는 논리적으로 연속적인 스트립 세트

RAID 1(미러링)

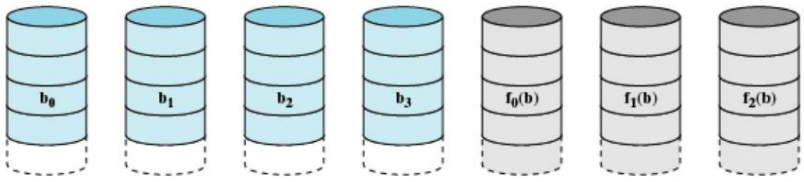


(b) RAID 1 (mirrored)

디스크 하나가 고장 날 때를 대비해 여분(redundant)의 디스크에 백업 데이터를 저장하는 방법

1. 미러링
2. 해밍 코드
3. 패리티 체크

RAID 2(해밍 코드를 통한 중복성)



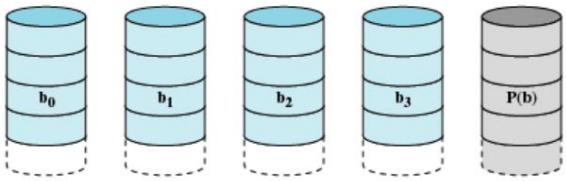
(c) RAID 2 (redundancy through Hamming code)

병렬 액세스 기술을 사용합니다. 스트립은 매우 작습니다(단일 바이트 또는 단어).

해밍 코드 사용 : 단일 비트 오류를 수정하고 이중 비트 오류를 감지할 수 있음

중복 디스크 필요 : 데이터 디스크 개수의 로그에 비례: 2P >= I + P + 1

RAID 3(비트 인터리브 패리티)



(d) RAID 3 (bit-interleaved parity)

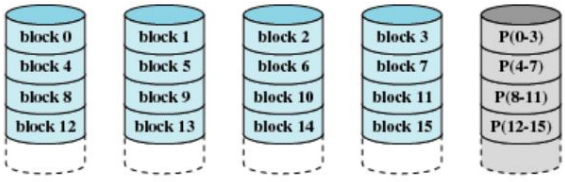
RAID 2와 유사: 단일 중복만 필요
디스크

각 데이터 디스크의 해당 비트에 대한 배타적 논리합에 의해 패리티 비트가 생성됩니다.

$X4(i) = X3(i) + X2(i) + X1(i) + X0(i) \bigcirc X1(i)$

$= X4(i) + X3(i) + X2(i) + X0(i) \bigcirc$

RAID 4(블록 수준 패리티)



(e) RAID 4 (block-level parity)

독립적인 접근 기술을 사용한다

$X4(i) = X3(i) + X2(i) + X1(i) + X0(i) \bigcirc X1(i)$

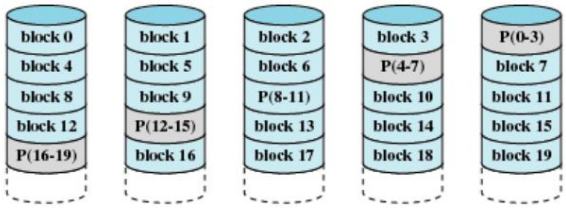
가 새로 수정되어 X1'(i)가 된 경우, X4'(i)는 아래처럼 기존의 X1(i), X1'(i), X4(i)로 구할 수 있음 (두 번의 read와 write)

$X4'(i) = X3(i) + X2(i) + X1'(i) + X0(i) \bigcirc$

$= X3(i) + X2(i) + X1'(i) + X0(i) + X1(i) + X1(i) \bigcirc$

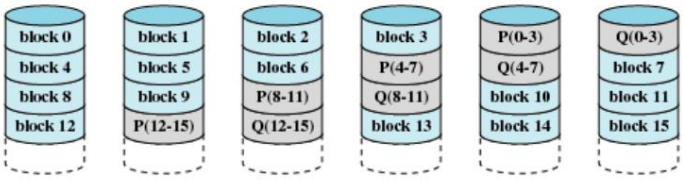
$= X4(i) + X1(i) + X1'(i)$

RAID 5(블록 수준 분산 패리티)



(f) RAID 5 (block-level distributed parity)

RAID 6(이중 중복)



(g) RAID 6 (dual redundancy)