CA6장 I/O

Introduction

I/O 디바이스는 다음과 같은 특성을 가진다..

-Behaviour : input, output, strage -Partner : human or machine -Data rage: bytes/sec, transfers/sec

I/O System Characteristics

Dependability가 중요. Fault: failure of a component

Performance는 Latency, Throughput, Desktops & embedded systems & Servers로 측정

Dependability Measures

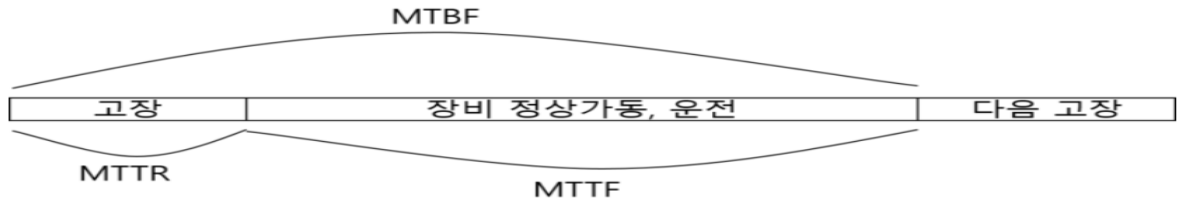
Reliability: mean time to failure (MTTF), Service interruption: mean time to repair (MTTR),

Mean time between failures(시스템의 고장 발생 평균 시간)

MTBF = MTTF + MTTR, MTBF는 오류가 발생할 때까지 응용 프로그램이 실행되는 평균 시간

가용성은 두 성취 상태와 중단 상태 사이의 교차에 관한 서비스 성취의 척도다.

Availability = MTTF / (MTTF + MTTR)



Improving Availability

1. Increase MTTF

-fault avoidance(피함) : construction에 의한 fault 발생 방지

-fault tolerance(자제) : 이중화를 사용하여 고장이 발생하더라도 서비스가 서비스 사양을 준수할 수 있도록 하며, 이는 주로 하드웨어 고장에 적용된다.

-fault forecasting(예측) : 하드웨어 및 소프트웨어 고장에 적용되는 결함의 존재 및 생성 예측, 고장 발생 전에 구성 요소를 교체할 수 있도록 함.

2. Reduce MTTR: improved tools and processes for diagnosis and repair

Disk Sectors and Access

섹터 : 디스크의 트랙의 일부. 각 섹터는 고정된 양의 사용자 접근이 가능한 데이터 저장. 하드 드라이브의 최소 기억단위.(Sector ID, Data, ECC, Synchronization fields and gaps)

실린더 : 트랙의 모임, 트랙 : 동심원

seek time: head를 원하는 선로로 옮길 시간

Rotational Latency : 헤드가 트랙 내에서 원하는 블록까지 가는 시간

Transfer Time : 블록의 섹터들과 이들 사이에 있는 갭들을 통과하는데 걸리는 시간.

Access to a sector involves

- Queuing delay if other accesses are pending
- Seek: move the heads
- Rotational latency
- Data transfer
- Controller overhead

Disk Access Example – 중요!(pdf 10쪽 example)

Disk Read Time

What is the average time to read or write a 512-byte sector for a typical disk rotating at 15,000 RPM? The advertised average seek time is 4 ms, the transfer rate is 100 MB/sec, and the controller overhead is 0.2 ms. Assume that the disk is idle so that there is no waiting time.

Average disk access time is equal to average seek time + average rotational delay + transfer time + controller overhead. Using the advertised average seek time, the answer is

$$4.0\text{ ms} + \frac{0.5\text{ rotation}}{15{,}000\text{ RPM}} + \frac{0.5\text{ KB}}{100\text{ MB/sec}} + 0.2\text{ ms} = 4.0 + 2.0 + 0.005 + 0.2 = 6.2\text{ ms}$$

If the measured average seek time is 25% of the advertised average time, the answer is

$$1.0\text{ ms} + 2.0\text{ ms} + 0.005\text{ ms} + 0.2\text{ ms} = 3.2\text{ ms}$$

Notice that when we consider measured average seek time, as opposed to advertised average seek time, the rotational latency can be the largest component of the access time.

RPM(분당회전수) 계산시에는 15000에서 60나눠서 해야한다. 0.5KB 대신 512 넣고 밑에를 바꾸는 것이 더 편할듯 하다. rotatinal latency는 원하는 정보에 대한 평균 대기 시간은 디스크의 절반이다. 1M가 10^6이다. 1ms=0.001s

access time= seek time + ratency latency + data transfer + controller overhead

Flash Storage

디스크보다 빠르고 작고 lower poewr, more robust(강건한), 하지만 더 비싸다.

Ramdom access라서 seektime과 rotation time이 없다.

Flash type

-NOR flash : Random read/write access

Used for instruction memory in embedded systems

-NAND flash : Denser(밀집한) (bits/area), but block-at-a-time access

Cheaper per GB

Used for USB keys, media storage, …

Bus Types

-Processor-Memory buses : 짧고, 스피드가 빠름. 메모리 구조에 적합

-I/O buses : 길고 ,mutiple 연결 허락, Connect to processor-memory bus through a bridge

Bus Signals and Synchronization

- Data lines : Carry address and data, Multiplexed or separate

- Control lines : Indicate data type, synchronize 처리

- Synchronous : Uses a bus clock

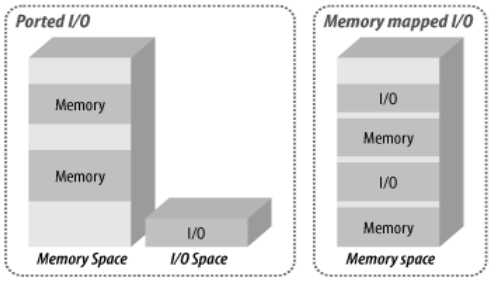
- Asynchronous : Uses request/acknowledge control lines for handshaking

I/O Management

I/O는 OS에의해 조정된다. I/O는 비동기화 interrupt을 발생키신다. I/O프로그램은 fiddly(가짜)다

I/O Register Mapping

I/O 레지스터 접근하는 몇가지 명령어들은 커널모드에서만 실행가능하다.



Polling

I/O 상태 레지스터를 정기적으로 점검하십시오.

폴링(polling)이란 하나의 장치(또는 프로그램)가 충돌 회피 또는 동기화 처리 등을 목적으로 다른 장치(또는 프로그램)의 상태를 주기적으로 검사하여 일정한 조건을 만족할 때 송수신 등의 자료처리를 하는 방식을 말한다.

If device ready, do operation If error, take action

작고 낮은 성능인 real-time embedded systems에서 보통 사용.

다른 시스템에서는 CPU 타임을 낭비한다.

사용 중인 대기 루프는 디바이스가 매우 빠르지 않는 한 CPU를 효율적으로 사용하는 방법이 아님! 그러나 I/O 완료에 대한 점검은 계산 집약적인 코드 사이에서 분산될 수 있다.

Interrupts

입출력 장치에서 자신들이 필요할 때마다 CPU에게 인터럽트 신호를 전송해서 순간 순간 사용하는 방식으로 폴링과 대비되는 용어 입니다. CPU 입장에서는 필요할 때만 입출력 장치와 통신하고 남는 시간동안 다른 일을 할 수 있으므로 폴링 방식에 비해 성능이 증가했습니다. [그림2]는 인터럽트가 발생했을 때, CPU가 어떻게 처리를 하는지 간략하게 보여주고 있습니다.

① 현재 CPU의 각종 레지스터와 상태를 저장

② 인터럽트 핸들링을 위해 ISR (Interrupt Service Routine)을 수행

③ 다시 원래 수행하던 프로그램으로 돌아가 정상적으로 동작

1. Logically AND the pending interrupt field and the interrupt mask field to see which enabled interrupts could be the culprit. Copies are made of these two registers using the mfc0 instruction.

2. Select the higher priority of these interrupts. The software convention is that the leftmost is the highest priority.

3. Save the interrupt mask field of the Status register.

4. Change the interrupt mask field to disable all interrupts of equal or lower priority.

5. Save the processor state needed to handle the interrupt.

6. To allow higher-priority interrupts, set the interrupt enable bit of the Cause register to 1.

7. Call the appropriate interrupt routine.

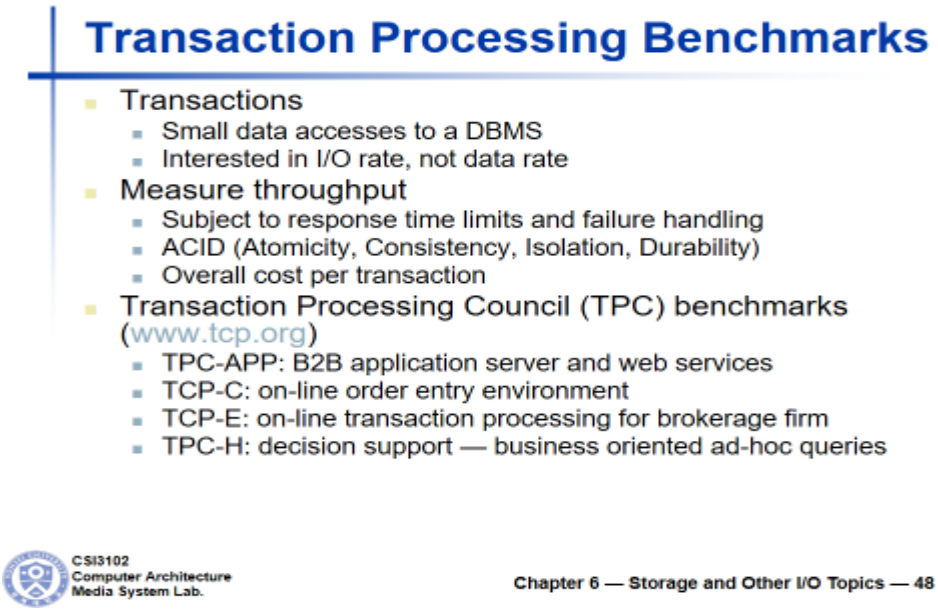
8. Before restoring state, set the interrupt enable bit of the Cause register to 0. This allows you to restore the interrupt mask field.

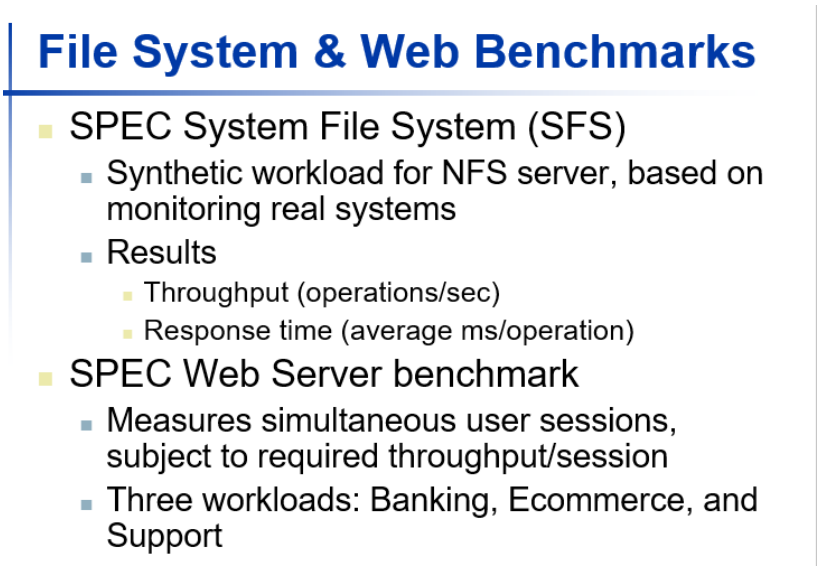
인터럽트 구동 I/O는 프로세서가 모든 I/O 이벤트를 기다려야 하는 것을 완화시켜 주지만, 하드 디스크에서 데이터를 전송하거나 하드 디스크로 전송하는 데 이 방법을 사용한다면 오버헤드는 여전히 견딜 수 없는 것이 될 수 있다. 디스크가 전송될 때 프로세서의 큰 fraction을 소비할 수 있기 때문이다.

우리는 장치가 프로세서와 통신 할 수있게하는 두 가지 다른 방법을 보았습니다. 폴링과 I / O 인터럽트의 두 가지 기술은 I / O 장치와 메모리간에 데이터 전송을 구현하는 두 가지 방법의 기초를 형성합니다. 이 두 기술은 모두 고 대역폭 전송을 제공하는 것보다 장치 컨트롤러 및 인터페이스의 비용을 절감하는 데 더 관심이있는 낮은 대역폭 장치에서 가장 잘 작동합니다. 폴링 및 인터럽트 구동 전송은 모두 데이터 이동 및 전송 관리를 담당합니다. 이 두 가지 계획을 살펴본 후에 고성능 장치 나 장치 모음에 더 적합한 체계를 검토 할 것입니다. 우리는 프로세서를 사용하여 폴링을 기반으로 디바이스와 메모리간에 데이터를 전송할 수 있습니다. 실시간 응용 프로그램에서 프로세서는 I / O 장치 레지스터에서 데이터를로드하고이를 메모리에 저장합니다. 또 다른 메커니즘은 데이터 전송을 중단시키는 것입니다. 이 경우 OS는 여전히 데이터를 장치에서 또는 장치로 작은 바이트 수로 전송합니다. 그러나 I / O 작업은 인터럽트 구동이므로 OS는 데이터를 장치에서 읽거나 장치에 쓰는 동안 다른 작업을 수행하기 만합니다. OS가 장치로부터 인터럽트를 인식하면 상태를 읽어 오류를 확인합니다. 아무 것도없는 경우, OS는 예를 들어 일련의 메모리 매핑 된 쓰기와 같은 다음 데이터 조각을 제공 할 수 있습니다. I / O 요청의 마지막 바이트가 전송되고 I / O 작업이 완료되면 OS는 프로그램에이를 알릴 수 있습니다. 프로세서와 OS는이 프로세스에서 모든 작업을 수행하여 전송 된 각 데이터 항목에 대한 장치와 메모리에 액세스합니다. 인터럽트 구동 I / O는 프로세서가 모든 I / O 이벤트를 기다릴 필요가 없도록 해줍니다. 비록 하드 디스크에서 또는 하드 디스크로 데이터를 전송하는 데이 방법을 사용하더라도 오버 헤드는 여전히 용인 될 수 있습니다. 디스크가 옮겨 졌을 때 프로세서의 상태. 하드 디스크와 같은 고 대역폭 장치의 경우 전송은 주로 상대적으로 큰 데이터 블록 (수백 ~ 수천 바이트)으로 구성됩니다. 따라서 컴퓨터 설계자는 프로세서를 오프 로딩하고 프로세서를 포함하지 않고 장치 컨트롤러가 메모리로 직접 또는 메모리에서 데이터를 직접 전송하도록하는 메카니즘을 고안했습니다. 이 메커니즘을 직접 메모리 액세스 (DMA)라고합니다. 인터럽트 메커니즘은 여전히 I / O 전송이 완료되거나 오류가 발생할 때만 프로세서와 통신하기 위해 장치에서 사용됩니다. DMA가 구현 됨

I/O benchmarks

- TPC, SPECSFS, SPECWeb





DMA(Direct memory access)

- OS provides starting address in memory

- I/O controller transfers to/from memory autonomously

- Controller interrupts on completion or error

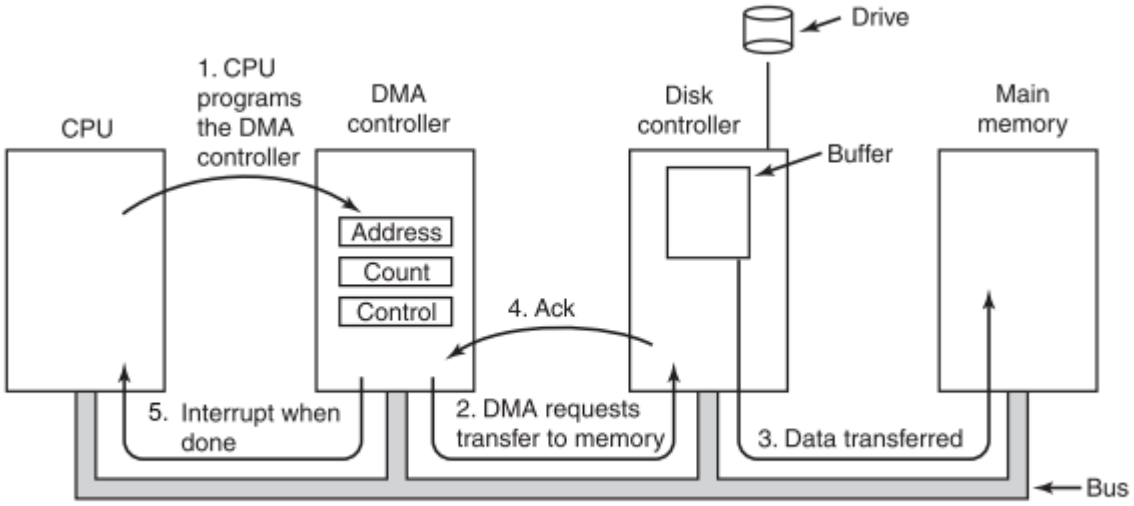


Figure 5-4. Operation of a DMA transfer.

동작 과정을 보면 CPU는 전송할 자료와 장소에 대한 위치를 나타내는 포인터와 전송 될 바이트 크기를 DMA에 전송합니다. 그러면 DMA는 해당 정보를 바탕으로 자신과 연결되어 있는 입출력 장치의 요구사항을 처리할 수 있어, CPU의 가용시간이 증가하는 장점이 있습니다.

DMA를 이용하면 입출력 장치간 데이터를 주고 받는 과정이 끝났을 때에만 인터럽트가 발생하므로 CPU의 역할을 현저히 줄일 수 있습니다.

위키피디아 정의

- 주변장치들(하드디스크, 그래픽 카드, 네트워크 카드, 사운드 카드 등)이 메모리에 직접 접근하여 읽거나 쓸 수 있도록 하는 기능으로서, 컴퓨터 내부의 버스가 지원하는 기능이다. 대개의 경우에 메모리의 일정 부분이 DMA에 사용될 영역으로 지정되며, DMA가 지원되면 CPU가 데이터 전송에 관여하지 않아도 되므로 컴퓨터 성능이 좋아진다.

DMA/Cache Interaction 책 번역

세 가지 주요 기술 중 하나를 사용하면 I / O 데이터의 일관성 문제를 피할 수 있습니다. 한 가지 방법은 I / O 활동을 캐시를 통해 라우팅하는 것입니다. 이렇게하면 쓰기가 캐시의 모든 데이터를 업데이트하는 동안 읽기가 최신 값을 보도록합니다. I / O 데이터가 거의 사용되지 않고 실행중인 프로그램에 필요한 유용한 데이터를 대체 할 수 있기 때문에 캐시를 통해 모든 I / O를 라우팅하는 것은 비용이 많이 들고 잠재적으로 프로세서에 큰 부정적인 성능 영향을 미칩니다. 두 번째 방법은 OS가 I / O 읽기를 위해 캐시를 선택적으로 무효로하거나 I / O 쓰기 (종종 캐시 플러싱이라고 함)를 위해 쓰기 되돌림을 강제 실행하는 것입니다. 이 접근법은 하드웨어 지원이 약간 필요하며 소프트웨어가 쉽고 효율적으로 기능을 수행 할 수 있다면 더 효율적입니다. 캐시의 많은 부분을 플러시하는 것은 DMA 블록 액세스에서만 발생하기 때문에 상대적으로 자주 사용되지 않습니다. 세 번째 방법은 캐시 항목을 선택적으로 플러시 (또는 무효화)하는 하드웨어 메커니즘을 제공하는 것입니다. 캐시 일관성을 보장하기위한 하드웨어 무효화는 다중 프로세서 시스템에서 일반적이며 I / O에도 동일한 기술을 사용할 수 있습니다. 5 장에서는이 주제에 대해 자세히 설명합니다.

DMA/VM Interaction 책 번역

가상 메모리가있는 시스템에서 DMA는 가상 주소 또는 실제 주소로 작동해야합니까? 가상 주소의 명백한 문제는 DMA 장치가 가상 주소를 실제 주소로 변환해야한다는 것입니다. DMA 전송에서 실제 주소를 사용할 때의 주요 문제점은 전송이 페이지 경계를 쉽게 넘을 수 없다는 것입니다. I / O 요청이 페이지 경계를 넘으면 전송 된 메모리 위치가 반드시 가상 메모리에서 연속적 일 필요는 없습니다. 따라서 실제 주소를 사용하는 경우 모든 DMA 전송이 한 페이지 내에 머물러야합니다. 시스템이 페이지 경계를 넘나 드는 DMA 전송을 시작하도록 허용하는 한 가지 방법은 DMA를 가상 주소에서 작동시키는 것입니다. 이러한 시스템에서, DMA 유닛은 전송을위한 가상 - 물리적 매핑을 제공하는 소수의 맵 엔트리를 갖는다. 운영 체제는 I / O가 시작될 때 매핑을 제공합니다. 이 맵핑을 사용함으로써, DMA 유닛은 전송과 관련된 가상 페이지의 위치를 염려 할 필요가 없다. 또 다른 기법은 운영 체제가 DMA 전송을 일련의 전송으로 분할하는 것입니다. 각 전송은 단일 물리적 페이지 내에 있습니다. 그런 다음 전송을 함께 연결하고 전체 전송 시퀀스를 실행하는 I / O 프로세서 또는 지능형 DMA 장치에 전달합니다. 또는 운영 체제가 개별적으로 전송을 요청할 수 있습니다. 어떤 방법을 사용하든 운영 체제는 해당 페이지와 관련된 DMA 전송이 진행되는 동안 페이지를 다시 매핑하지 않아도 협조해야합니다.

Measuring I/O Performance

I/O 성능에 영향가는 것들

- Hardware: CPU, memory, controllers, buses

- Software: operating system, database management system, application

- Workload: request rates and patterns

throughput 측정은 종종 제한된 response time으로 수행됩니다. 둘이 trade-off 관계이다.

Transaction Processing Benchmarks

트랜잭션 처리 (TP) 어플리케이션에는 응답 시간 요구 사항과 처리량을 기반으로 한 성능 측정이 모두 포함됩니다. 또한 대부분의 I / O 액세스는 작습니다. 이 때문에 TP 어플리케이션은 주로 초당 데이터 바이트로 측정되는 데이터 속도와 달리 초당 액세스 수로 측정되는 I / O 속도와 관련됩니다. TP 어플리케이션은 일반적으로 시스템이 응답 시간 요구 사항을 충족시킬뿐만 아니라 특정 유형의 실패를 정상적으로 처리하는 대형 데이터베이스에 대한 변경을 포함합니다. 이러한 애플리케이션은 매우 중요하며 비용에 민감합니다. 예를 들어, 은행은 일반적으로 TP 시스템을 사용하는데, 그 이유는 은행이 특성의 범위를 염려하기 때문입니다. 여기에는 트랜잭션이 손실되지 않았는지 확인하고 트랜잭션을 신속하게 처리하며 각 트랜잭션을 처리하는 비용을 최소화하는 작업이 포함됩니다. 이러한 시스템에서 장애 발생시의 신뢰성은 절대적인 요구 사항이지만 응답 시간과 처리량은 모두 비용 효율적인 시스템을 구축하는 데 중요합니다.

Transactions

- Small data accesses to a DBMS

- Interested in I/O rate, not data rate

Measure throughput

- Subject to response time limits and failure handling

- ACID (Atomicity, Consistency, Isolation, Durability)

- Overall cost per transaction

먼소리지...

File System & Web Benchmarks

- SPEC System File System (SFS)
  - Synthetic workload for NFS server, based on monitoring real systems
  - Results
    - Throughput (operations/sec)
    - Response time (average ms/operation)
- SPEC Web Server benchmark
  - Measures simultaneous user sessions, subject to required throughput/session
  - Three workloads: Banking, Ecommerce, and Support

I/O vs. CPU Performance

Amdahl's Law : 병렬 처리가 컴퓨팅 성능을 높이기 때문에 I / O 성능을 무시하지 마십시오.

$$\frac{1}{(1-P)+\frac{P}{S}}$$

(개선 후 실행시간 = 개선에 의해 영향을 받는 실행 시간 / 성능 향상 비율 + 영향을 받지 않는 실행 시간)

예를 들어서 어떤 작업의 40%에 해당하는 부분의 속도를 2배로 늘릴 수 있다면, P는 0.4이고 S는 2이고 최대 성능 향상은

$$\frac{1}{(1-0.4)+\frac{0.4}{2}} = 1.25$$

Year	CPU time	I/O time	Elapsed time	% I/O time
now	90s	10s	100s	10%
+2	45s	10s	55s	18%
+4	23s	10s	33s	31%
+6	11s	10s	21s	47%

이렇게 CPU성능이 매년 2배씩 좋아져서 I/O 비율을 무시할수 없다.

RAID

- Redundant Array of Inexpensive (Independent) Disks (외우기)

Use multiple smaller disks (c.f. one large disk)

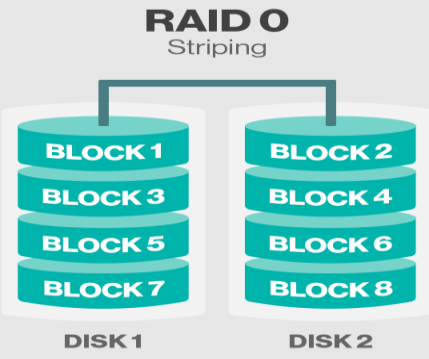
Parallelism improves performance

Plus extra disk(s) for redundant data storage

Provides fault tolerant storage system

- Especially if failed disks can be "hot swapped"

RAID 0

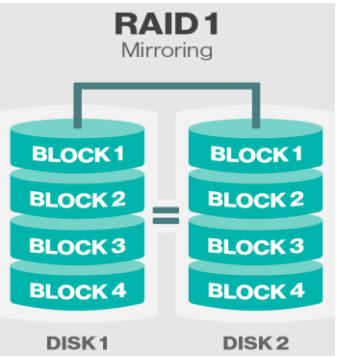


- No redudancy : 데이터를 여러 디스크에 stripe하면됩니다.

- 성능 향상은 됨

하나의 파일을 두개의 디스크에게 나눠쓴다. 읽을적에 A1과 A2을 동시에 읽을수 있어서 속도면에서 괜찮다.(장점) 하지만 문제가 disk0나 1중에 하나가 삭제되면 파일 recovery가 안된다.(단점)

RAID 1 : mirroring



disk 1이 disk2의 거울이다. 이것은 속도면에서 큰 이득이 없다. write 할적에 두군데 복사해야한다. 이득은 백프로 reduance한다. On disk failure, read from mirror

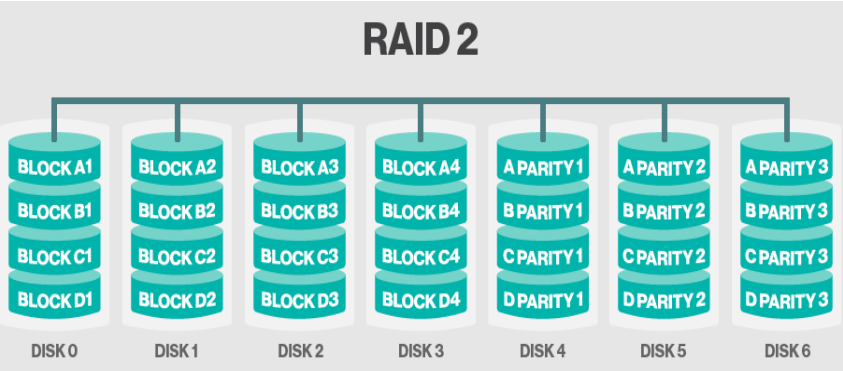
그리고 하나의 디스크는 아예못쓴다. RAID1은 스토리지의 안정성이 중요한곳에는 많이 쓴다.(ex) 은행)

N + N disks, replicate(복제하다) data

* 장점 : 드라이브 하나가 고장 나면 똑같은 내용의 다른 드라이브가 하나 더 있기 때문에 매우 안전하다. RAID 1은 읽기 성능이 단일 드라이브에서의 성능과 같거나 훨씬 좋다.

* 단점 : 각 드라이브는 미러링되기 때문에 전체 용량의 절반밖에 사용하지 못한다. 드라이브 두 개에 동일한 데이터를 써야 하기 때문에 쓰기 성능이 나쁘다. 하지만 아직 다른 RAID 레벨의 쓰기 성능보다는 훨씬 낫다.

RAID 2: Error correcting code (ECC)



-N + E disks

- Split data at bit level across N disks

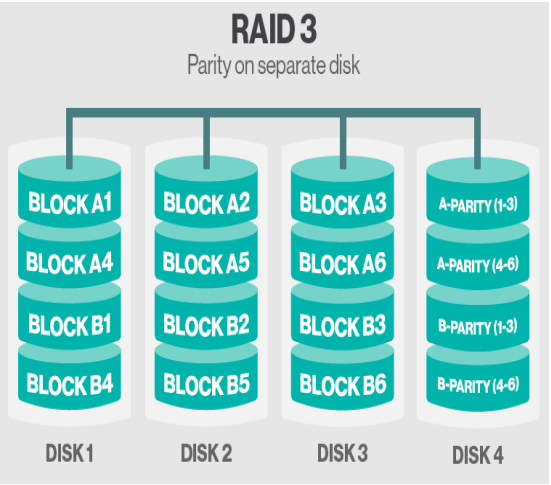
- Generate E-bit ECC

- Too complex, not used in practice(안쓰임)

오류정정부호(ECC)를 기록하는 전용의 하드디스크를 이용해서 안정성을 확보한다. RAID 2는 비트 단위에 Hamming code를 적용한다. 때문에 하나의 멤버 디스크가 고장나도 ECC를 이용하여 정상적으로 작동할 수 있지만, 추가적인 연산이 필요하여 입출력 속도가 매우 떨어진다.

모든 I/O에서 ECC 계산이 필요하므로 입출력 병목 현상이 발생하며, ECC 기록용으로 쓰이는 디스크의 수명이 다른 디스크들에 비해 짧아지는 문제가 있어 현재는 사용하지 않는다.

RAID 3: Bit-Interleaved Parity



- N + 1 disks

- Redundant disk stores parity

- Read access :Read all disks

- Write access : Generate new parity and update all disks

On failure -Use parity to reconstruct(복원하다) missing data

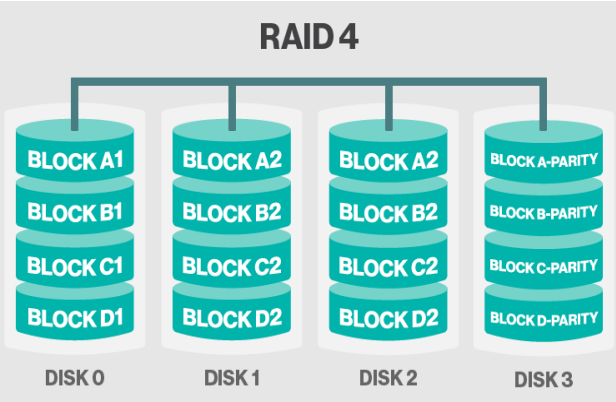
Not widely used

설명 : 데이터는 바이트 단위로 쪼개져서 모든 디스크에 균등하게 나뉘어 저장되고 패리티 정보는 별도의 전용 디스크에 저장된다.

*장점 : 한 개의 드라이브가 고장 나는 것을 허용하며 순차적 쓰기 성능과 순차적 읽기 성능이 우수하다.

* 단점 : 잘 사용되지 않고 문제를 해결하는 것이 어려울 수 있다. 패리티 디스크가 Fail 되면 복구가 불가능하기 때문이다. 하드웨어 RAID가 되어야 실제로 쓸만하다. RAID 3은 보통 매우 효율적이지만 임의 쓰기 성능이 나쁘고 임의 읽기 성능은 꽤 좋다. paritiy disk는 모든 read/wirte operation에 관여해야하니깐 편중적으로 디스크를 써야해서 bottleneck(병목)된다..

RAID 4: Block-Interleaved Parity



- N + 1 disks

RAID 3와 비슷-

모든 파일을 블록으로 쪼개고 각 블럭은 여러 디스크에 저장되지만 균등하진 않다. RAID 3처럼 RAID 4도 패리티를 처리하기 위해 별도의 디스크를 사용한다. 동시 트랜잭션 사용량이 많은 시스템에서 읽기 속도는 매우 중요한데 이런 시스템에 적합하다.

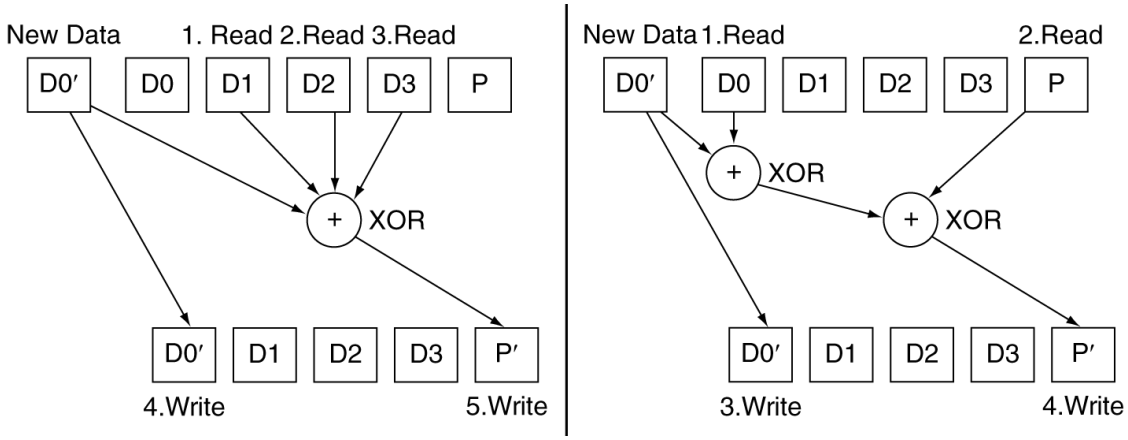
* 장점 : 드라이브 하나가 고장 나는 것을 허용하고 읽기 성능이 매우 좋다.

* 단점 : 쓰기 성능이 나쁘지만 블럭 읽기(block read) 성능은 괜찮다.

RAID 3 vs RAID 4

Raid 3은 Byte단위로 데이터를 저장하는 반면 Raid 4는 Block단위로 저장합니다.

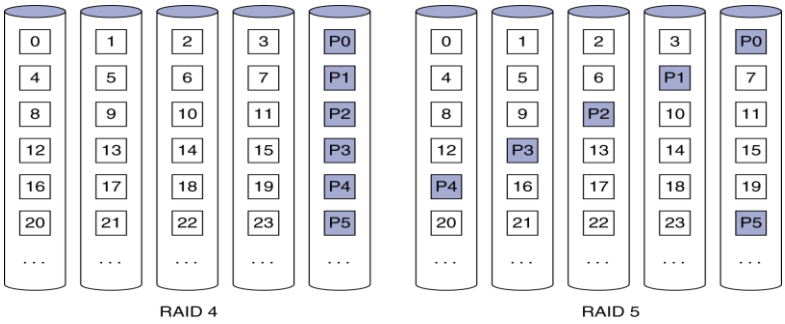
Block단위로 저장을 할 경우 작은 파일의 경우는 한번의 작업으로 데이터를 읽을 수 있기 때문에 성능상의 장점이 있습니다. 레벨 3은 동기화를 거쳐야 하기 때문에 3보다는 레벨 4를 많이 사용합니다



small write problem : 어떤 디스크의 내용이 업데이트가 되면 엄청난 오버헤드가 발생한다.

패러리 비트를 업데이트해야해서 저렇게 된다.

RAID 5: Distributed Parity



- N+1 disks.

- Like RAID 4, but parity blocks distributed across disks

- Avoids parity disk being a bottleneck

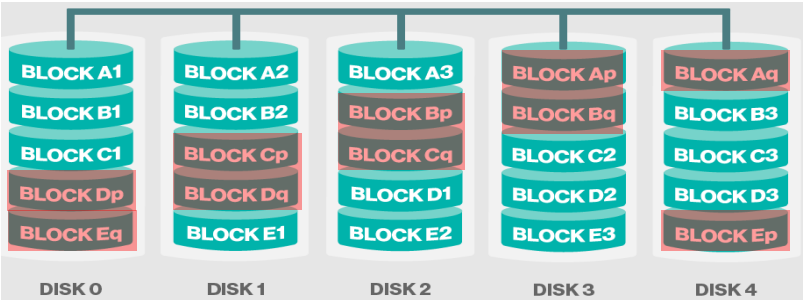
설명 : RAID 4처럼 데이터의 블럭은 모든 디스크에 나뉘어 저장되지만 항상 균등하진 않고 패리티 정보도 모든 디스크에 나뉘어 저장된다.

* 장점 : 지원하는 회사가 많고 한 개의 드라이브가 고장 나는 것을 허용한다.

Raid 3과 Raid 4의 단점을 해결한 형태이며 고급 Raid Controller에서 현재 많이 쓰이고 있는 형태입니다. 이는 별도의 ECC 드라이브를 두지 않고, 각 드라이브에 분산되어 ECC 드라이브에 대한 병목현상을 막아주며 ECC를 위한 알고리즘 때문에 성능 면에서는 여전히 Raid 0 보다 떨어지지만 성능, 안정성, 용량의 세 부분을 고려한 형태이다.

* 단점 : 디스크 재구성(rebuild)이 매우 느리고 쓰기 성능은 패리티 정보를 끊임없이 갱신해야 하기 때문에 우수하다고 할 수는 없다.

RAID 6: P + Q Redundancy



-N+2 disks

- Like RAID 5, but two lots of parity

- Greater fault tolerance through more redundancy

- Multiple RAID : 보다 발전된 시스템이 유사한 fault tolerance(관대)를 제공하므로 성능이 향상됨

* 설명 : RAID 6은 RAID 5와 비슷한 구성이지만 다른 드라이브들 간에 분포되어 있는 2차 패리티 정보를 넣어 2개의 하드에 문제가 생겨도 복구할 수 있게 설계되었으므로 RAID 5보다 더욱 데이터의 안전성을 보장 할 수 있다. 하지만 이러한 추가적 보호 기능은 비용 부담으로 이어진다.

* 장점 : 두 개의 드라이브까지 고장 나는 것을 허용하고 읽기 성능이 우수하고 매우 중요한 경우에 적합하다.

* 단점 : 쓰기 성능은 패리티를 여러 번 갱신해야 하기 때문에 RAID 5보다 매우 나쁘다. 디스크를 재구성하는 동안에 성능이 매우 나빠질 수 있다.

마지막 요약

Concluding Remarks

- I/O performance measures
  - Throughput, response time
  - Dependability and cost also important
- Buses used to connect CPU, memory, I/O controllers
  - Polling, interrupts, DMA
- I/O benchmarks
  - TPC, SPECSFS, SPECWeb
- RAID
  - Improves performance and dependability