LG전자 BS팀

『4일차』: 오전

◈ 훈련과정명 : OS 기본

◈ 훈 련 기 간 : 2023.05.30-2[illegible]3.06.02

목차

『11과목』

1-2교시 :
I/O Systems

Syllabus

학습목표

- 이 워크샵에서는 운영 체제의 I/O 하위 시스템 구조 탐색을 할 수 있다.
- I/O 하드웨어의 원리와 복잡성 논의를 할 수 있다.
- I/O 하드웨어 및 소프트웨어의 성능 측면 설명을 할 수 있다.

Syllabus

눈높이 체크

- 입출력 하드웨어를 알고 계신가요?
- 애플리케이션 I/O 인터페이스를 알고 계신가요?
- 커널 I/O 하위 시스템을 알고 계신가요?
- I/O 요청을 하드웨어 작업으로 변환을 알고 계신가요?
- 스트림을 알고 계신가요?

1. I/O Hardware

개관

- I/O 관리는 운영 체제 설계 및 운영의 주요 구성 요소.
  - 컴퓨터 작동의 중요한 측면
  - I/O 장치는 매우 다양.
  - 그들을 제어하는 다양한 방법
  - 성과 관리
  - 자주 사용되는 새로운 유형의 장치
- 다양한 장치에 연결되는 포트, 버스, 장치 컨트롤러
- 장치 드라이버는 장치 세부 정보를 캡슐화합니다.
  - I/O 하위 시스템에 대한 균일한 장치 액세스 인터페이스 제공

1. I/O Hardware

I/O Hardware

- 놀랍도록 다양한 I/O 장치
  Storage
  Transmission
  Human-interface
- 일반적인 개념 – 컴퓨터와 I/O 장치 인터페이스의 신호
- 포트 Port – 장치의 연결 지점
- 버스 Bus - 데이지 체인 daisy chain 또는 공유 직접 액세스
- PC와 서버에서 공통적으로 사용되는 PCI 버스, PCI Express(PCIe)
- 확장 버스 expansion bus 는 상대적으로 느린 장치를 연결.
- Serial-attached SCSI (SAS) 공통 디스크 인터페이스
- 컨트롤러 Controller(호스트 어댑터) - 포트, 버스, 장치를 작동하는 전자 장치
- 때때로 통합, 경우에 따라 별도의 회로 기판(host adapter)
- 프로세서, 마이크로코드, 전용 메모리, 버스 컨트롤러 등을 포함.
- 일부는 버스 컨트롤러, 마이크로코드, 메모리 등을 사용하여 장치별 컨트롤러와 통신.

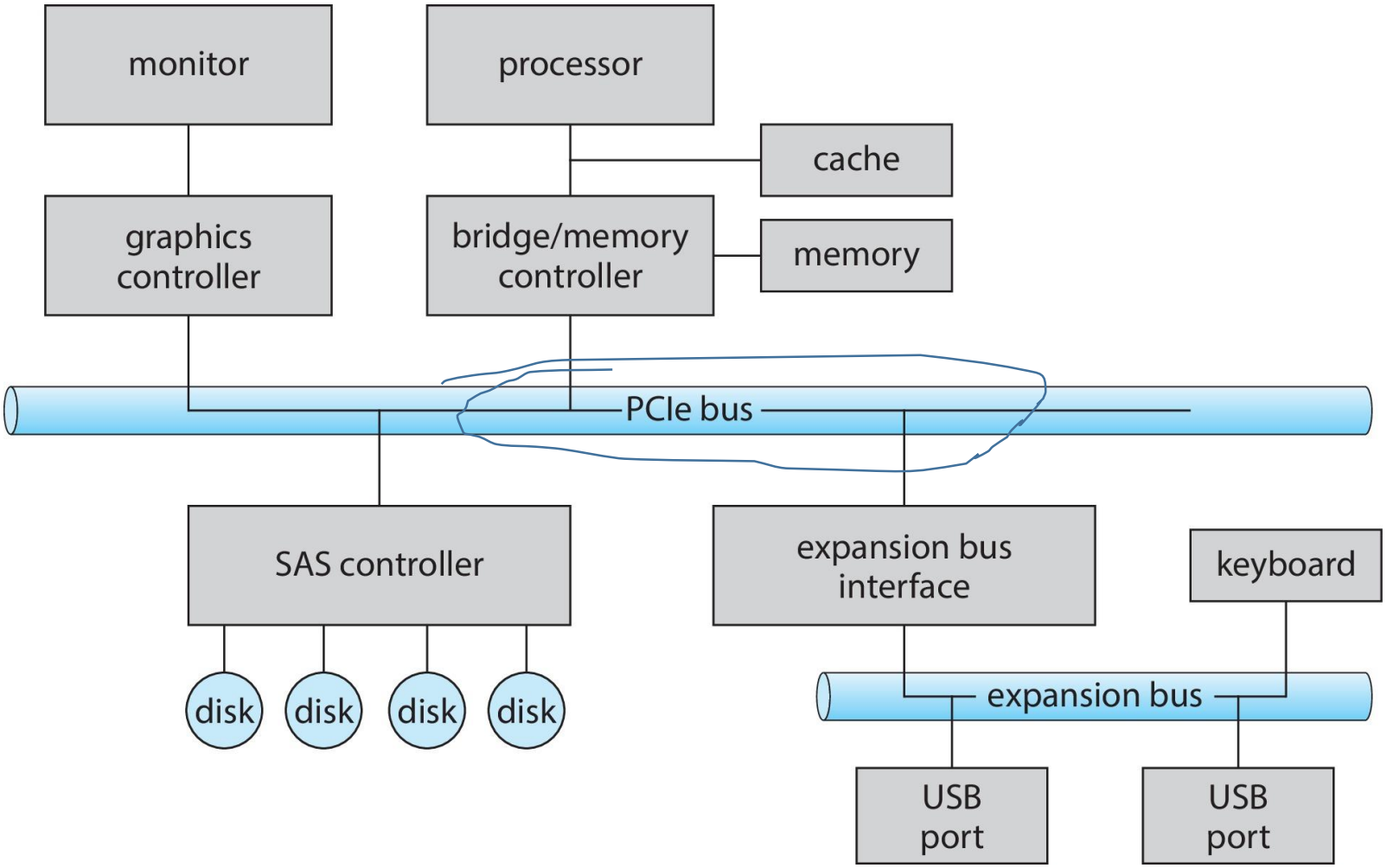
1. I/O Hardware

I/O Hardware

- Fibre channel(FC)은 복잡한 컨트롤러이며 일반적으로 버스에 연결되는 별도의 회로 기판(host-bus adapter, HBA).
- I/O 명령 제어 장치
- 장치에는 일반적으로 장치 드라이버가 명령 실행 후 레지스터에서 데이터를 쓰거나 읽을 명령, 주소 및 데이터를 배치하는 레지스터가 있다.
- Data-in register, data-out register, status register, control register
- 일반적으로 1-4바이트 또는 FIFO 버퍼
- 장치에는 다음에서 사용하는 주소가 있다.
- 직접 I/O 명령어Direct I/O instructions
- Memory-mapped I/O
  - ✓ 프로세서 주소 공간에 매핑된 장치 데이터 및 명령 레지스터
  - ✓ 특히 큰 주소 공간(그래픽)의 경우

1. I/O Hardware

일반적인 PC 버스 구조



1. I/O Hardware

PC의 장치 I/O 포트 위치(일부)

- **Memory-Mapped I/O**

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

1. I/O Hardware

Polling

- I/O의 각 바이트에 대해

1. 상태 레지스터에서 0까지 비지 비트 읽기
2. 호스트는 읽기 또는 쓰기 비트를 설정하고 쓰기가 데이터 출력 레지스터에 데이터를 복사하는 경우
3. 호스트는 명령 준비 비트를 설정.
4. 컨트롤러는 사용 중 비트를 설정하고 전송을 실행.
5. 컨트롤러는 전송이 완료되면 사용 중 비트, 오류 비트, 명령 준비 비트를 지웁니다.

- 1단계는 디바이스에서 I/O를 기다리는 busy-wait 사이클.

- 장치가 빠른 경우 합리적인
- 그러나 장치가 느리면 비효율적.
- CPU가 다른 작업으로 전환할까?
- 그러나 주기 데이터를 놓치면 덮어쓰거나 손실.

1. I/O Hardware

Interrupts

- 폴링은 3개의 명령어 주기로 발생할 수 있다.
  - 상태 읽기, 논리 및 상태 비트 추출, 0이 아닌 경우 분기
  - 드물게 0이 아닌 경우 더 효율적인 방법은 무엇일까?
- I/O 장치에 의해 트리거된 CPU 인터럽트 요청 라인
  - 각 명령 후 프로세서에서 확인
- 인터럽트 처리기 Interrupt handler 가 인터럽트를 수신.
  - 일부 인터럽트를 무시하거나 지연하도록 마스크 가능
- 올바른 처리기로 인터럽트를 디스패치하는 인터럽트 벡터 Interrupt vector
  - 시작 및 종료 시 컨텍스트 전환
  - 우선 순위 기반
  - 가려지지 않는 일부nonmaskable
  - 동일한 인터럽트 번호에 둘 이상의 장치가 있는 경우 인터럽트 체이닝

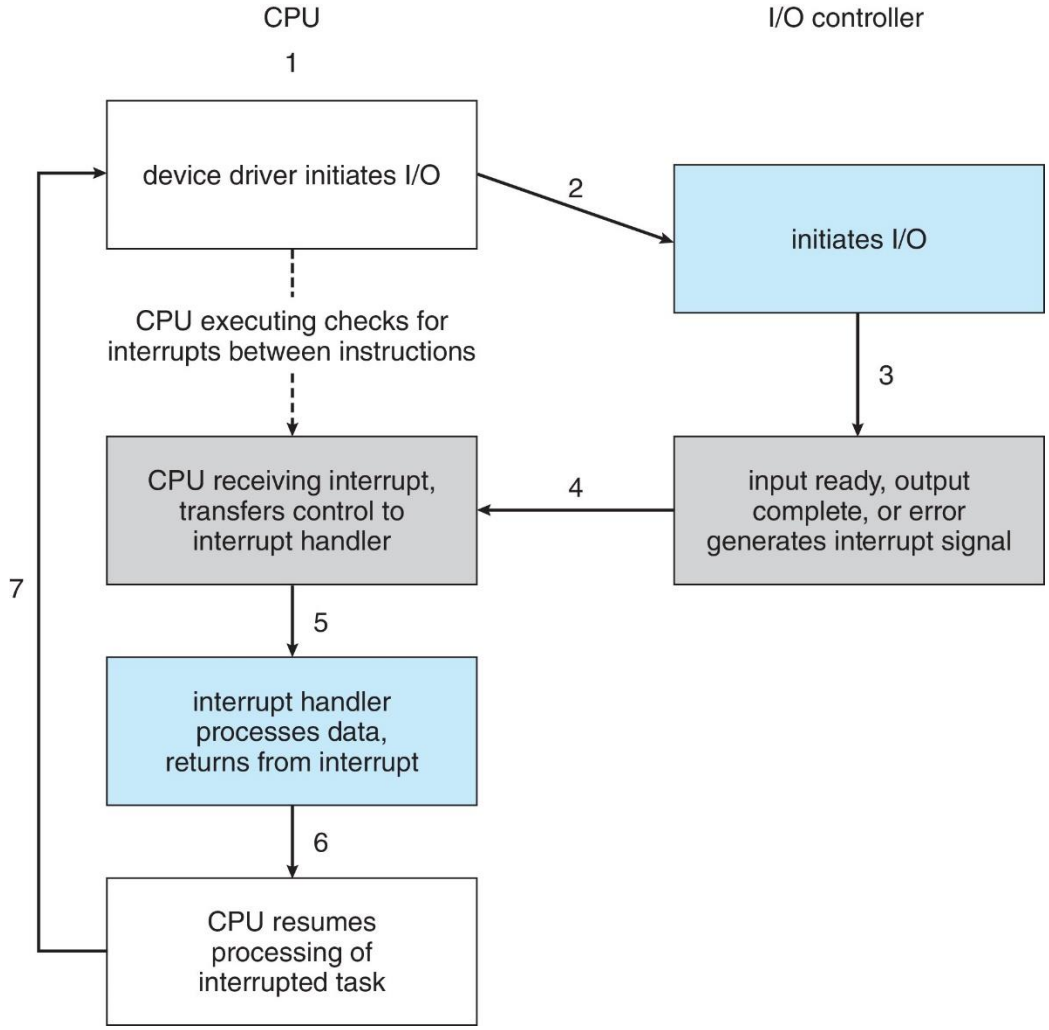
1. I/O Hardware

Interrupts

- 예외에도 사용되는 인터럽트 메커니즘
  - 프로세스 종료, 하드웨어 오류로 인한 시스템 충돌
- 메모리 액세스 오류가 발생하면 페이지 오류가 실행.
- 트랩을 통해 시스템 호출이 실행되어 커널이 요청을 실행하도록 트리거.
- 다중 CPU 시스템은 인터럽트를 동시에 처리할 수 있다.
  - 운영 체제가 이를 처리하도록 설계된 경우
- 시간에 민감한 처리에 자주 사용, 빨라야 함

1. I/O Hardware

인터럽트 구동 I/O 주기



1. I/O Hardware

지연 시간 Latency

- 단일 사용자 시스템조차도 초당 수백 또는 초당 인터럽트를 관리하고 수십만 개의 서버를 관리하기 때문에 인터럽트 관리에 스트레스를 줍니다.
- 예를 들어 조용한 macOS 데스크톱은 10초 동안 23,000개의 인터럽트를 생성.

인텔 펜티엄 프로세서 이벤트 벡터 테이블

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

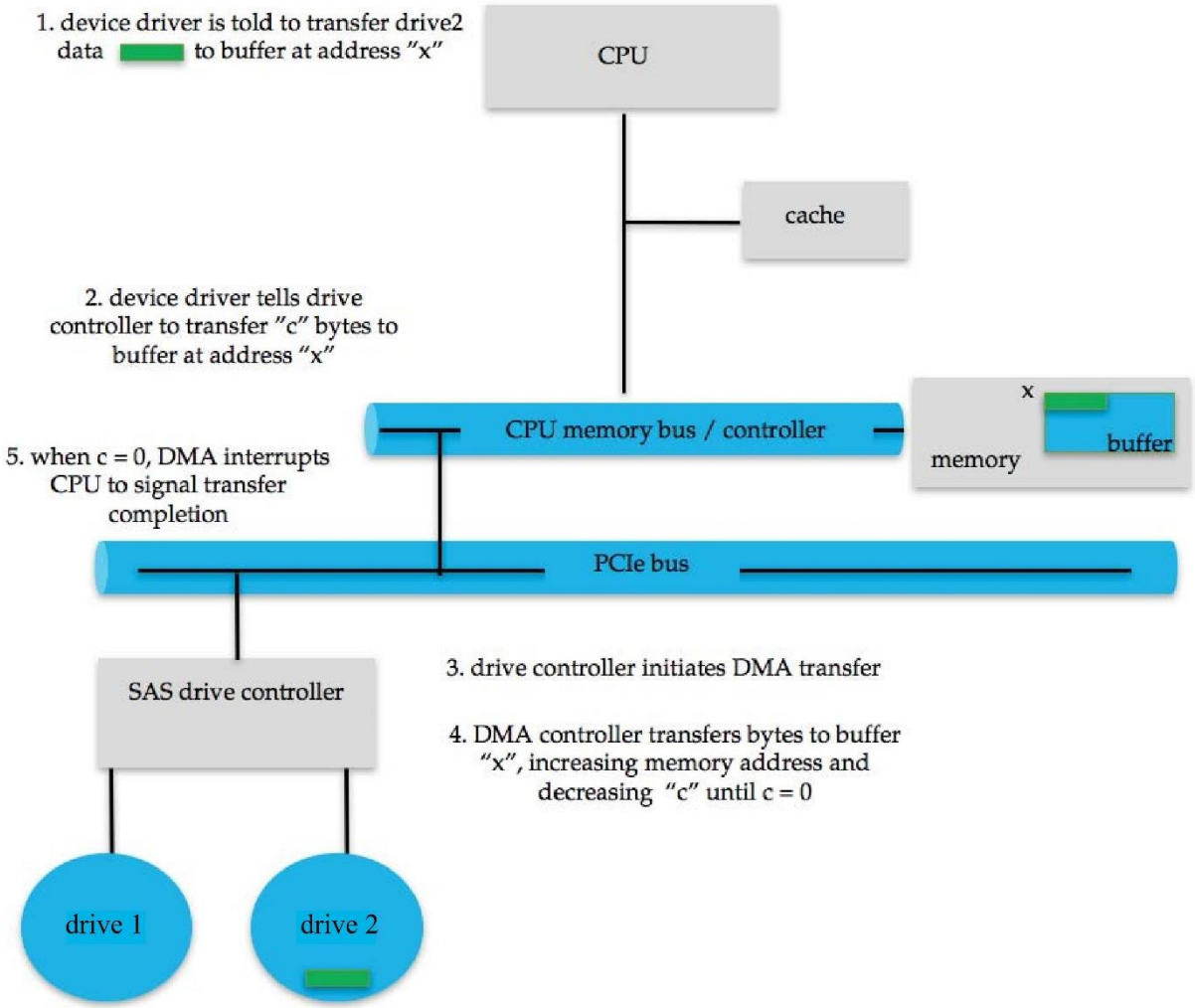
1. I/O Hardware

직접 메모리 액세스 Direct Memory Access

- 대규모 데이터 이동을 위해 프로그래밍된 I/O(한 번에 1바이트)를 방지하는 데 사용.
- DMA 컨트롤러 필요
- I/O 장치와 메모리 간에 데이터를 직접 전송하기 위해 CPU를 우회.
- OS가 DMA 명령 블록을 메모리에 씁니다.
  - 소스 및 대상 주소
  - 읽기 또는 쓰기 모드
  - 바이트 수
  - 명령 블록의 위치를 DMA 컨트롤러에 씁니다.
  - DMA 컨트롤러의 버스 마스터링 – CPU에서 버스 가져오기
  - CPU에서 주기를 훔치지만 Cycle stealing 훨씬 더 효율적.
  - 완료되면 신호 완료에 대한 인터럽트
- 가상 주소를 인식하는 버전은 훨씬 더 효율적일 수 있다 - DVMA

1. I/O Hardware

DMA 전송을 수행하기 위한 6단계 프로세스



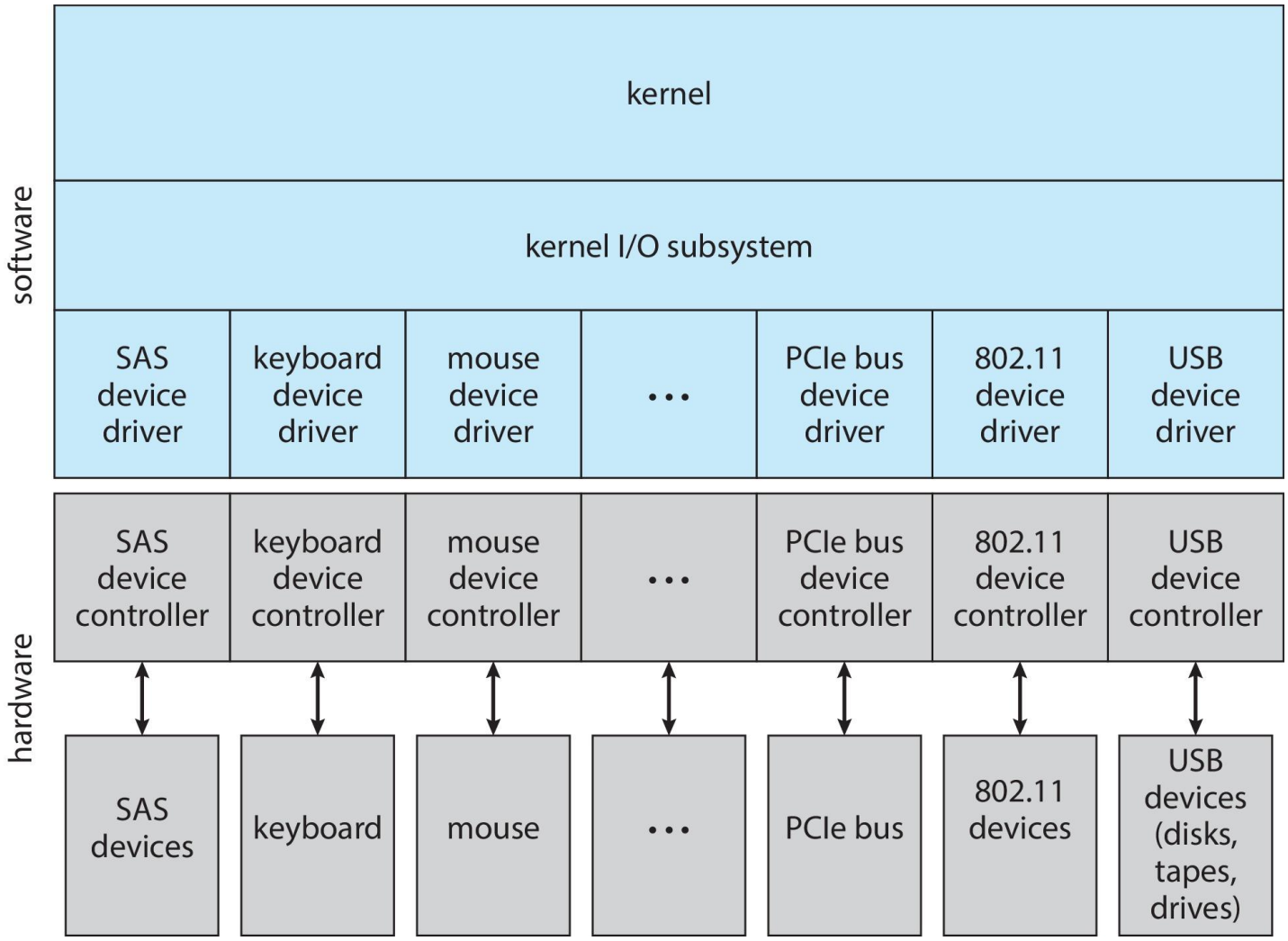
2. Application I/O Interface

일반적인 PC 버스 구조

- I/O 시스템 호출은 장치 동작을 일반 클래스로 캡슐화다.
- 장치 드라이버 계층은 커널에서 I/O 컨트롤러 간의 차이점을 숨깁니다.
- 이미 구현된 프로토콜을 말하는 새로운 장치는 추가 작업이 필요하지 않다.
- 각 OS에는 고유한 I/O 하위 시스템 구조와 장치 드라이버 프레임워크가 있다.
- 장치는 여러 차원에서 다양.
  - Character-stream or block
  - Sequential or random-access
  - Synchronous or asynchronous (or both)
  - Sharable or dedicated
  - Speed of operation
  - read-write, read only, or write only

2. Application I/O Interface

커널 I/O 구조



2. Application I/O Interface

I/O 장치의 특성

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

2. Application I/O Interface

I/O 장치의 특성

- 장치 드라이버가 처리하는 장치의 미묘함
- 대체로 I/O 장치는 OS별로 다음과 같이 그룹화할 수 있다.
  - Block I/O
  - Character I/O (Stream)
  - Memory-mapped file access
  - Network sockets
- I/O 장치 고유의 특성을 직접 조작하기 위해 일반적으로 탈출/백도어
  - 임의의 비트를 장치 제어 레지스터로 보내고 데이터를 장치 데이터 레지스터로 보내는 Unix ioctl() 호출
- UNIX 및 Linux는 "major" 및 "minor" 장치 번호의 튜플을 사용하여 장치의 유형 및 인스턴스를 식별합니다(여기서는 주 8 및 부 0-4)

```
% ls -l /dev/sda*
```

```
brw-rw---- 1 root disk 8, 0 Mar 16 09:18 /dev/sda
brw-rw---- 1 root disk 8, 1 Mar 16 09:18 /dev/sda1
brw-rw---- 1 root disk 8, 2 Mar 16 09:18 /dev/sda2
brw-rw---- 1 root disk 8, 3 Mar 16 09:18 /dev/sda3
```

2. Application I/O Interface

블록 및 문자 장치

- 블록 장치에는 디스크 드라이브가 포함.
  - 명령에는 읽기, 쓰기, 검색이 포함.
  - Raw I/O, direct I/O 또는 파일 시스템 액세스
  - Memory-mapped file access 가능
  - 요구 페이징을 통해 가져온 가상 메모리 및 클러스터에 매핑된 파일
  - DMA
- 문자 장치에는 키보드, 마우스, 직렬 포트가 포함.
  - 명령에는 get(), put()이 포함.
  - 상단에 계층화된 라이브러리는 라인 편집을 허용.

2. Application I/O Interface

Network Devices

- 자체 인터페이스를 가질 수 있도록 블록과 캐릭터에서 충분히 다양
- Linux, Unix, Windows 및 기타 많은 소켓 인터페이스 포함
  - 네트워크 작동에서 네트워크 프로토콜을 분리.
  - select() 기능 포함
- 접근 방식은 매우 다양(pipes, FIFOs, streams, queues, mailboxes).

2. Application I/O Interface

Clocks and Timers

- 현재시간, 경과시간, 타이머 제공
- 일반 해상도 약 1/60초
- 일부 시스템은 고해상도 타이머를 제공.
- 타이밍, 주기적 인터럽트에 사용되는 프로그래밍 가능한 간격 타이머 Programmable interval timer
- ioctl()(UNIX에서)은 시계 및 타이머와 같은 I/O의 이상한 측면을 다룹니다.

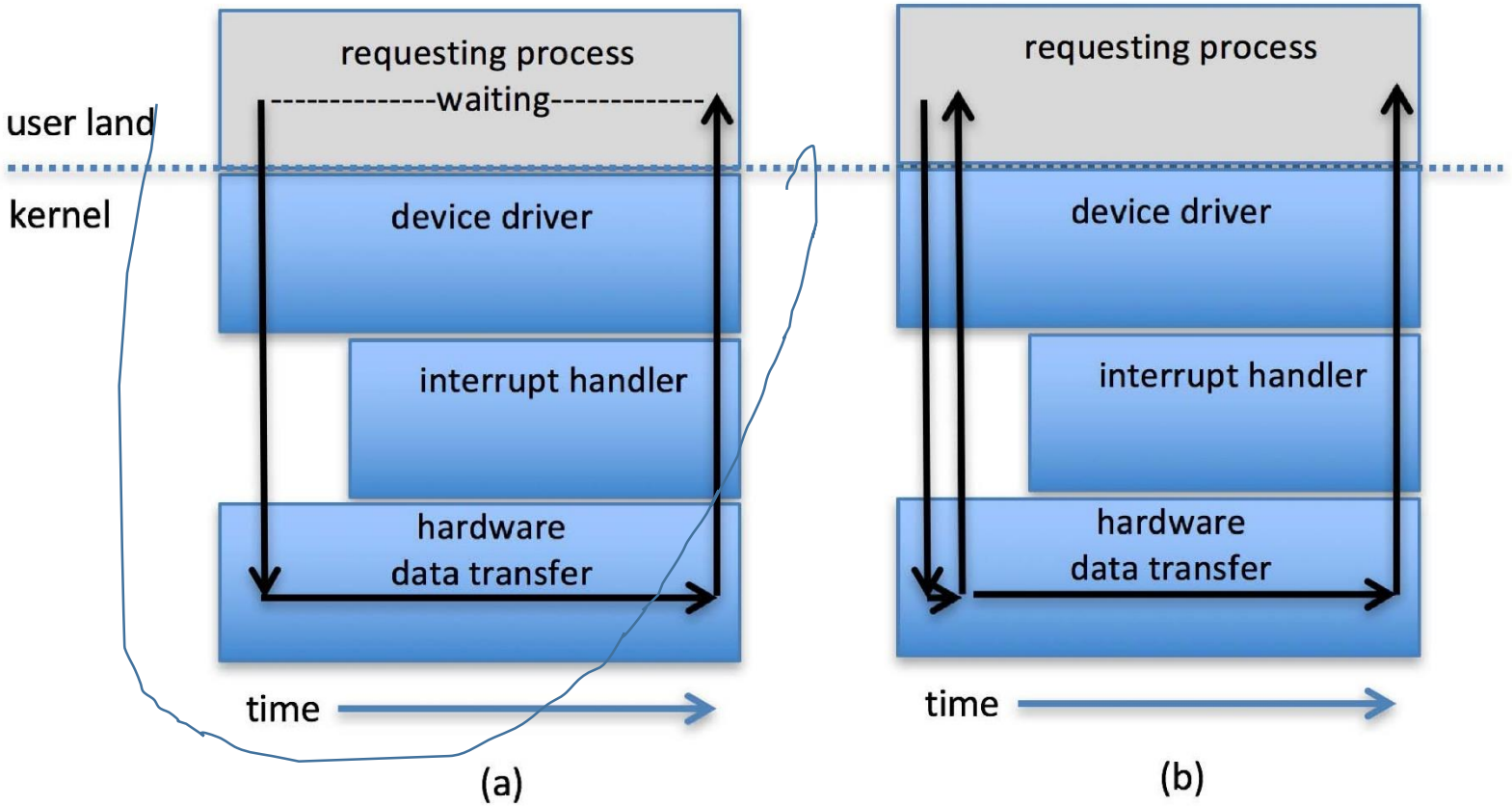
2. Application I/O Interface

Nonblocking and Asynchronous I/O

- 차단 Blocking - I/O가 완료될 때까지 프로세스가 일시 중단됨
  - 사용하기 쉽고 이해하기 쉽다.
  - 일부 요구사항에는 불충분
- Nonblocking - I/O 호출이 가능한 한 많이 반환
  - 사용자 인터페이스, 데이터 복사(버퍼 I/O)
  - 멀티스레딩을 통해 구현
  - 읽거나 쓴 바이트 수와 함께 빠르게 반환
  - select() 데이터가 준비되었는지 확인한 다음 read() 또는 write() 전송
- 비동기식 Asynchronous - I/O가 실행되는 동안 프로세스 실행
  - 사용하기 어려움
  - I/O 완료 시 I/O 하위 시스템 신호 처리

2. Application I/O Interface

두 가지 I/O 방법



2. Application I/O Interface

두 가지 I/O 방법

- 벡터화된 I/O는 하나의 시스템 호출이 여러 I/O 작업을 수행하도록 허용.
- 예를 들어 Unix readve()는 읽거나 쓸 여러 버퍼의 벡터를 허용.
- 이 분산 수집 방법은 여러 개별 I/O 호출보다 낫다.
  - 컨텍스트 전환 및 시스템 호출 오버헤드 감소
  - 일부 버전은 원자성을 제공.
  - 예를 들어 읽기/쓰기가 발생할 때 여러 스레드가 데이터를 변경하는 것에 대해 걱정하지 않아도 된다.

3. Kernel I/O Subsystem

기본 개념

- **Scheduling**
  - 장치별 대기열을 통한 일부 I/O 요청 순서 지정
  - 일부 OS는 공정성을 시도합니다.
  - 일부는 서비스 품질 구현(예: IPQOS)
- 버퍼링 Buffering - 장치 간에 전송하는 동안 메모리에 데이터 저장
  - 장치 속도 불일치에 대처하기 위해
  - 장치 전송 크기 불일치에 대처하기 위해
  - "복사 시맨틱"을 유지하려면
  - 이중 버퍼링 Double buffering – 두 개의 데이터 사본

① 커널 및 사용자
② 다양한 크기
③ 전체 / 처리 중 및 전체 아님 / 사용 중
④ 경우에 따라 효율성을 위해 Copy-on-Write를 사용할 수 있다.

3. Kernel I/O Subsystem

기본 개념

- **캐싱 Caching - 데이터 사본을 보관하는 더 빠른 장치**
  - 항상 사본만
  - 성능의 핵심
  - 때때로 버퍼링과 결합
- **스풀링 Spooling - 장치에 대한 출력 보류**
  - 기기가 한 번에 하나의 요청만 처리할 수 있는 경우
  - 인쇄Printing
- **장치 예약 Device reservation - 장치에 대한 독점 액세스 제공**
  - 할당 및 할당 해제를 위한 시스템 호출
  - 교착 상태에 주의

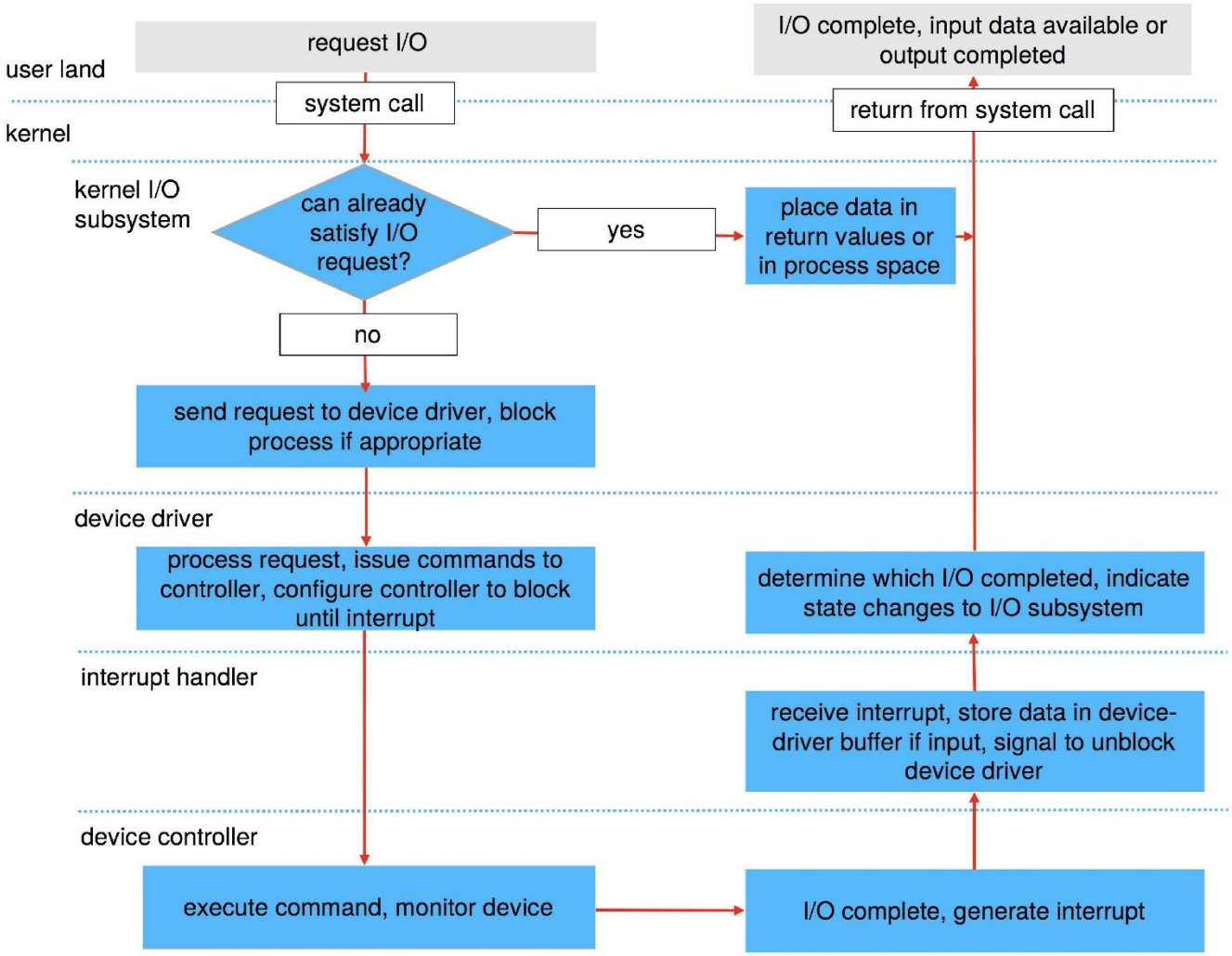
4. I/O 요청을 하드웨어 작업으로 변환

기본 개념

- 프로세스를 위해 디스크에서 파일을 읽는 것을 고려.
- 장치 보관 파일 결정
- 이름을 장치 표현으로 번역
- 물리적으로 디스크에서 버퍼로 데이터 읽기
- 요청 프로세스에서 데이터를 사용할 수 있도록 합니다.
- 제어를 프로세스로 되돌리기

4. I/O 요청을 하드웨어 작업으로 변환

I/O 요청의 수명 주기



5. STREAMS

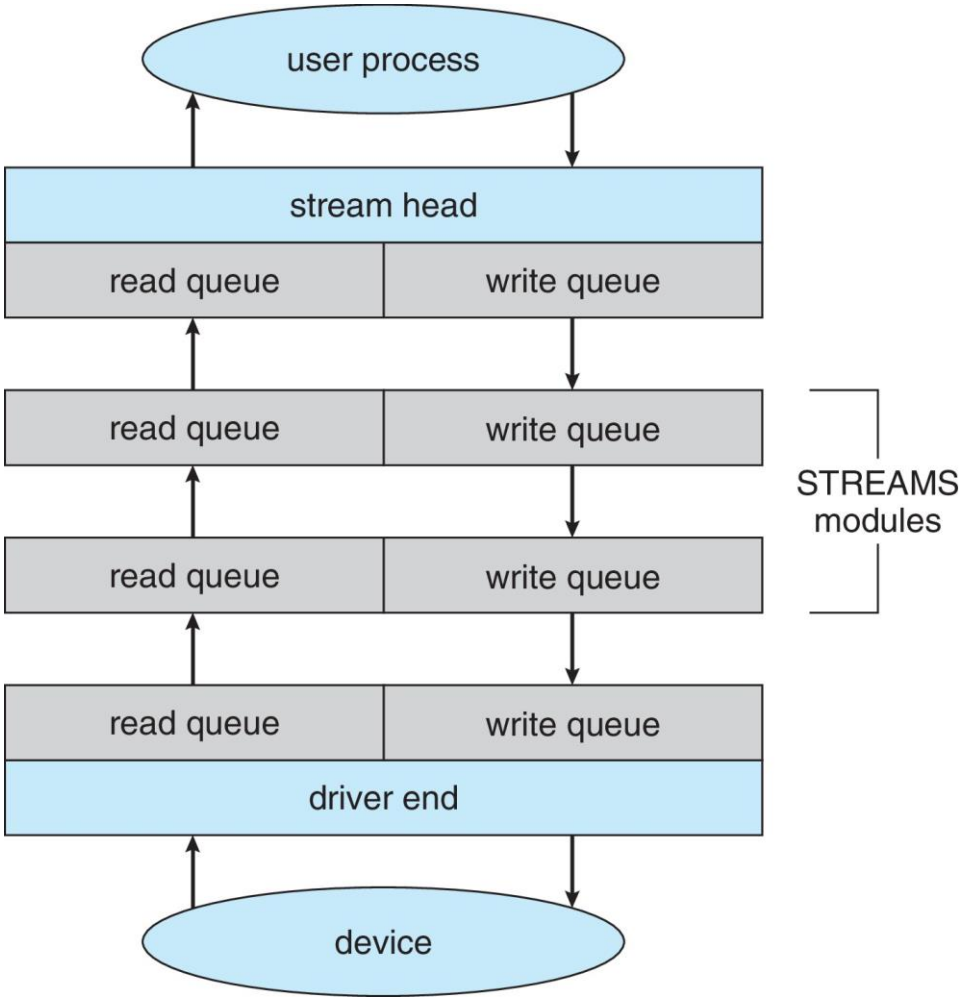
스트림?

- STREAM – Unix System V 이상에서 사용자 수준 프로세스와 장치 간의 전이중 통신 채널
- 스트림은 다음으로 구성.
  - 사용자 프로세스와 STREAM 헤드 인터페이스
  - 디바이스와 드라이버 엔드 인터페이스
  - 그들 사이에 o개 이상의 STREAM 모듈
- 각 모듈에는 읽기 대기열과 쓰기 대기열이 포함되어 있다.

- 메시지 전달은 대기열 간 통신에 사용됩니다.
  - 사용 가능 또는 사용 중을 나타내는 흐름 제어 옵션
- 내부적으로 비동기식, 사용자 프로세스가 스트림 헤드와 통신하는 동기식

5. STREAMS

The STREAMS Structure?



『12과목』

3-4교시 : File-System

Syllabus

학습목표

- 이 워크샵에서는 파일 시스템의 기능 설명 할 수 있다.
- 파일 시스템에 대한 인터페이스 설명 할 수 있다.
- 액세스 방법, 파일 공유, 파일 잠금 및 디렉터리 구조를 포함하여 파일 시스템 설계 장단점 논의 할 수 있다.
- 파일 시스템 보호를 탐색 할 수 있다.

Syllabus

눈높이 체크

- 파일 개념을 알고 계신가요?
- 액세스 방법을 알고 계신가요?
- 디스크 및 디렉토리 구조를 알고 계신가요?
- 보호Protection를 알고 계신가요?
- 메모리 매핑된 파일을 알고 계신가요?

1. File Concept

File Concept

- 연속적인 논리 주소 공간
- 유형:
  - 데이터
    - ✓ 숫자
    - ✓ 성격
    - ✓ 바이너리
  - 프로그램
- 파일 작성자가 정의한 내용
  - 텍스트 파일,
  - 소스 파일,
  - 실행 가능 파일

1. File Concept

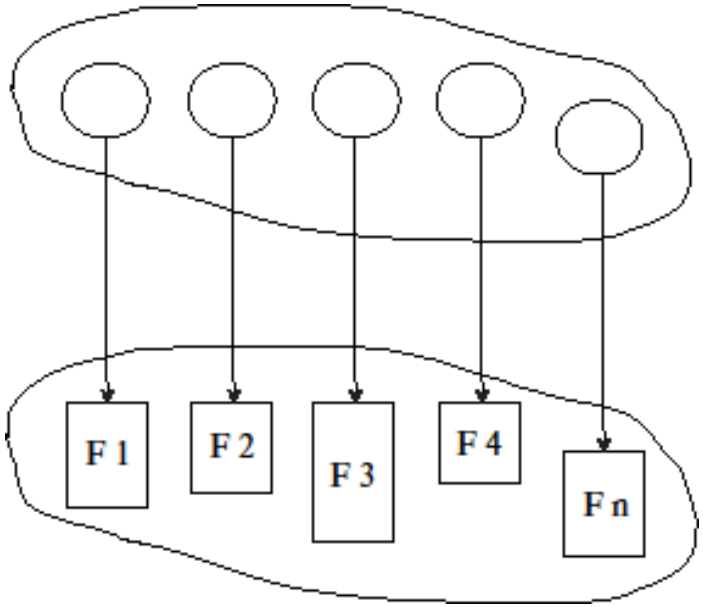
File Attributes

- 이름 Name – 사람이 읽을 수 있는 형식으로 유지되는 유일한 정보
- 식별자 Identifier – 고유한 태그(숫자)는 파일 시스템 내에서 파일을 식별합니다.
- 유형 Type – 다양한 유형을 지원하는 시스템에 필요
- 위치 Location – 장치의 파일 위치에 대한 포인터
- 크기 Size – 현재 파일 크기
- 보호 Protection – 읽기, 쓰기, 실행을 할 수 있는 사람을 제어합니다.
- 시간, 날짜 및 사용자 식별 Time, date, and user identification - 보호, 보안 및 사용 모니터링을 위한 데이터
- 파일에 대한 정보는 디스크에 유지되는 디렉토리 구조에 보관됩니다.
- 파일 체크섬과 같은 확장된 파일 속성을 포함한 다양한 변형
- 디렉토리 구조에 보관된 정보

1. File Concept

디렉토리 구조

- 모든 파일에 대한 정보를 포함하는 노드 모음



- 디렉토리 구조와 파일 모두 디스크에 상주

1. File Concept

File Operations

- Create
- Write – 쓰기 포인터 위치에서
- Read – 읽기 포인터 위치에서
- Reposition within file - seek
- Delete
- Truncate
- Open(Fi) – 항목 Fi에 대한 디스크의 디렉토리 구조를 검색하고 항목 내용을 메모리로 이동합니다.
- Close(Fi) – 메모리에 있는 항목 Fi의 내용을 디스크의 디렉토리 구조로 이동

1. File Concept

파일 열기

- 열린 파일을 관리하려면 몇 가지 데이터가 필요.
- 열린 파일 테이블 **Open-file table** : 열린 파일 추적
- 파일 포인터: 파일을 연 프로세스당 마지막 읽기/쓰기 위치에 대한 포인터
- **File-open count**: 파일이 열린 횟수 카운터 – 마지막 프로세스가 파일을 닫을 때 열린 파일 테이블에서 데이터 제거 허용
- 파일의 디스크 위치: 데이터 액세스 정보의 캐시
- 액세스 권한: 프로세스별 액세스 모드 정보

1. File Concept

파일 잠금

- 일부 운영 체제 및 파일 시스템에서 제공
  - reader-writer locks과 유사
  - 리더 잠금과 유사한 Shared lock - 여러 프로세스가 동시에 획득할 수 있음
  - 라이터 잠금과 유사한 Exclusive lock
- 파일에 대한 액세스 중재
- 필수 또는 권고:
  - 필수 Mandatory – 보유 및 요청된 잠금에 따라 액세스가 거부.
  - 조언 Advisory – 프로세스는 잠금 상태를 찾고 수행할 작업을 결정할 수 있다.

1. File Concept

File Structure

- None - sequence of words, bytes
- 간단한 레코드 구조
  - Lines
  - Fixed length
  - Variable length

- 복잡한 구조
  - Formatted document
  - Relocatable load file
- 적절한 제어 문자를 삽입하여 첫 번째 방법으로 마지막 두 개를 시뮬레이션할 수 있다.
- 결정하는 사람:
  - Operating system
  - Program

2. Access Methods

기본 개념

- 파일은 고정 길이 논리 레코드.
- 순차적 액세스Sequential Access

- 바로 연결Direct Access

- 기타 액세스 방법

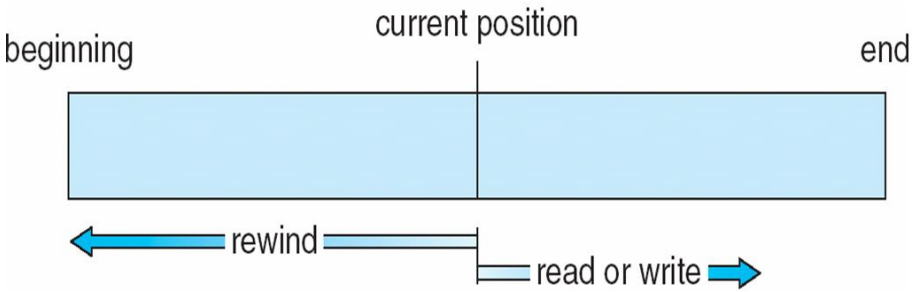
2. Access Methods

Sequential Access

Operations

- **read next**
- **write next**
- **Reset**
- no read after last write (rewrite)

Figure



2. Access Methods

Direct Access

Operations

- read n
- write n
- position to n
  - read next
  - write next
  - rewrite n
- n = relative block number

● 상대 블록 번호를 통해 OS는 파일을 배치할 위치를 결정할 수 있다.

2. Access Methods

직접 액세스 파일에 대한 순차 액세스 시뮬레이션

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp*; *cp* = *cp* + 1;
write next	*write cp*; *cp* = *cp* + 1;

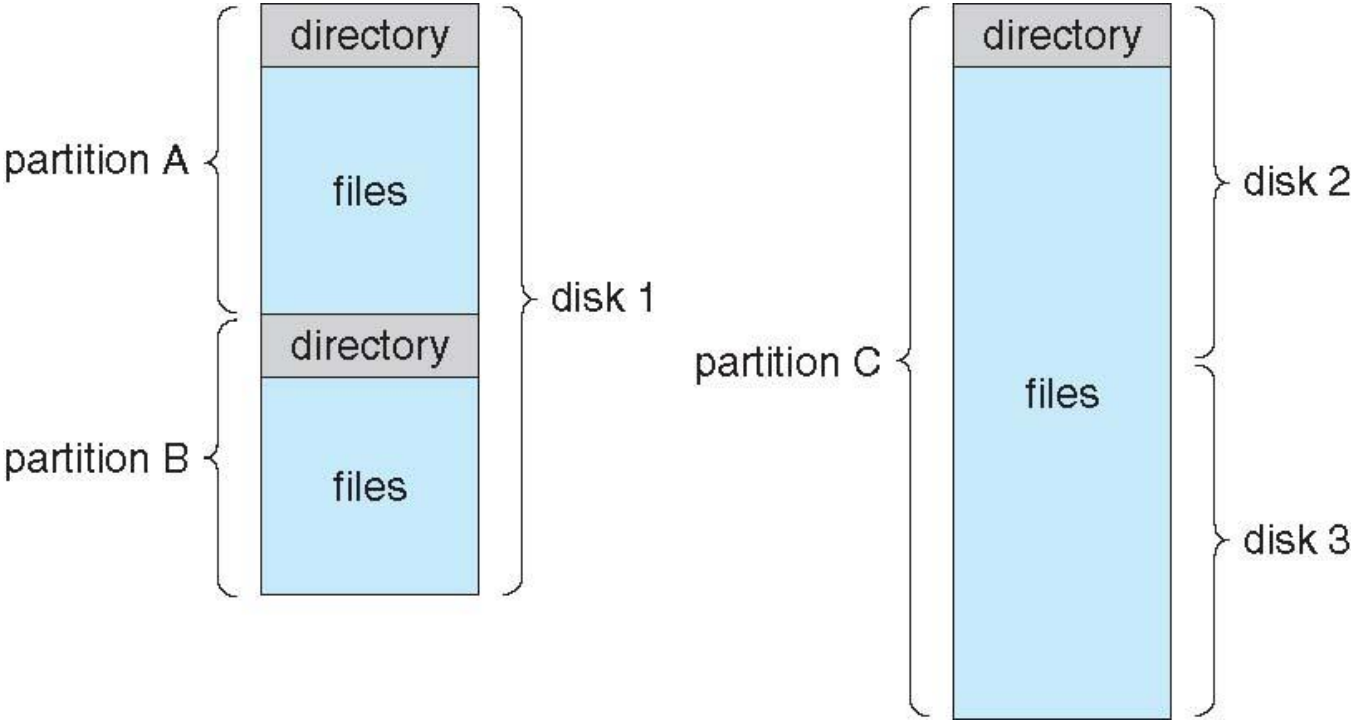
3. Disk and Directory Structure

기본 개념

- 디스크를 파티션 partitions 으로 세분할 수 있음
- 디스크 또는 파티션을 장애로부터 RAID로 보호할 수 있다.
- 디스크 또는 파티션은 파일 시스템 없이 원시 raw 로 사용하거나 파일 시스템으로 포맷할 수 있다.
- 미니디스크 minidisks, 슬라이스 slices 라고도 하는 파티션
- 파일 시스템을 포함하는 엔티티를 볼륨이라고 합니다.
- 파일 시스템을 포함하는 각 볼륨은 장치 디렉토리 device directory 또는 볼륨 목차or volume table of contents에서 해당 파일 시스템의 정보도 추적.
- 범용 파일 시스템 general-purpose file systems 외에도 많은 특수 목적 파일 시스템 special-purpose file systems 이 있으며, 종종 모두 동일한 운영 체제 또는 컴퓨터 내에 있다.

3. Disk and Directory Structure

일반적인 파일 시스템 구성



3. Disk and Directory Structure

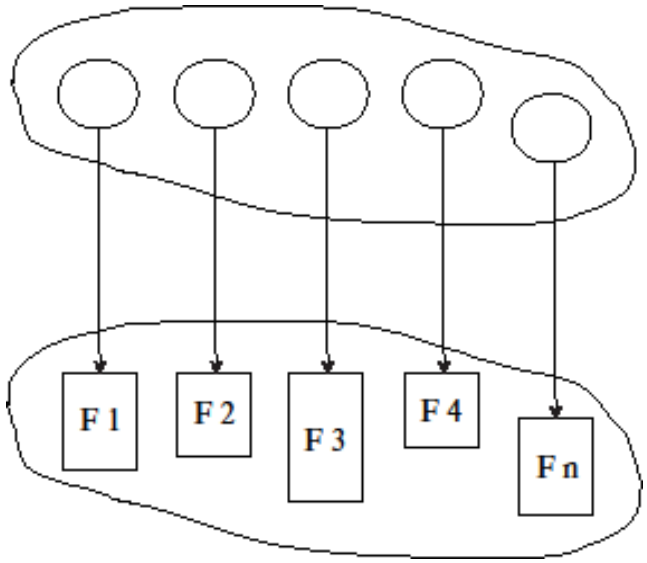
파일 시스템 유형

- 우리는 주로 범용 파일 시스템에 대해 이야기한다.
- 그러나 시스템에는 종종 파일 시스템이 있을 수 있다. 일부는 범용 및 일부는 특수 목적
- **Solaris**
- **tmpfs – 빠른 임시 I/O를 위한 메모리 기반 휘발성 FS**
- **objfs – 디버깅을 위한 커널 기호를 얻기 위해 커널 메모리에 인터페이스**
- **ctfs – 데몬 관리를 위한 계약 파일 시스템**
- **lofs - 루프백 파일 시스템을 사용하면 하나의 FS를 다른 FS 대신 액세스할 수 있다.**
- **procfs – 구조를 처리하기 위한 커널 인터페이스**
- **ufs, zfs – 범용 파일 시스템**

3. Disk and Directory Structure

Directory Structure

- 모든 파일에 대한 정보를 포함하는 노드 모음



- 디렉토리 구조와 파일 모두 디스크에 상주

3. Disk and Directory Structure

디렉터리에서 수행되는 작업

- **Search for a file**
- **Create a file**
- **Delete a file**
- **List a directory**
- **Rename a file**
- **Traverse the file system**

3. Disk and Directory Structure

디렉토리 조직

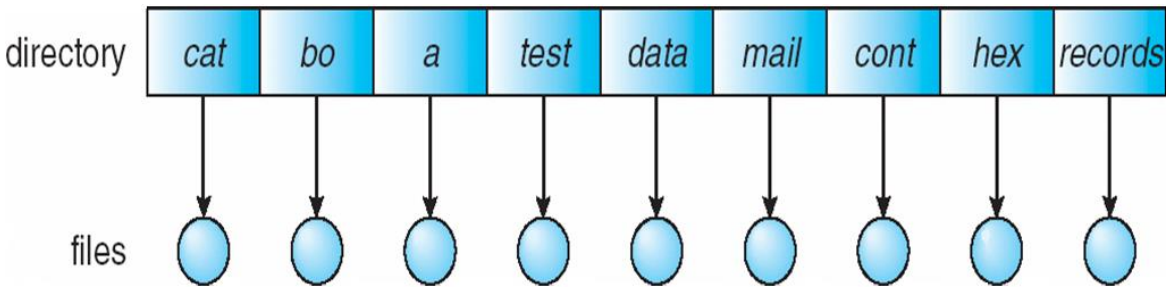
- 디렉토리는 논리적으로 구성되어 있다.

- 효율성 Efficiency – 신속하게 파일 찾기
- 이름 지정 Naming - 사용자에게 편리함
  - 두 명의 사용자가 서로 다른 파일에 대해 동일한 이름을 가질 수 있다.
  - 동일한 파일이 여러 다른 이름을 가질 수 있음
- 그룹화 Grouping – 속성별로 파일을 논리적으로 그룹화(예: 모든 Java 프로그램, 모든 게임 등)

3. Disk and Directory Structure

Single-Level Directory

- 모든 사용자를 위한 단일 디렉토리

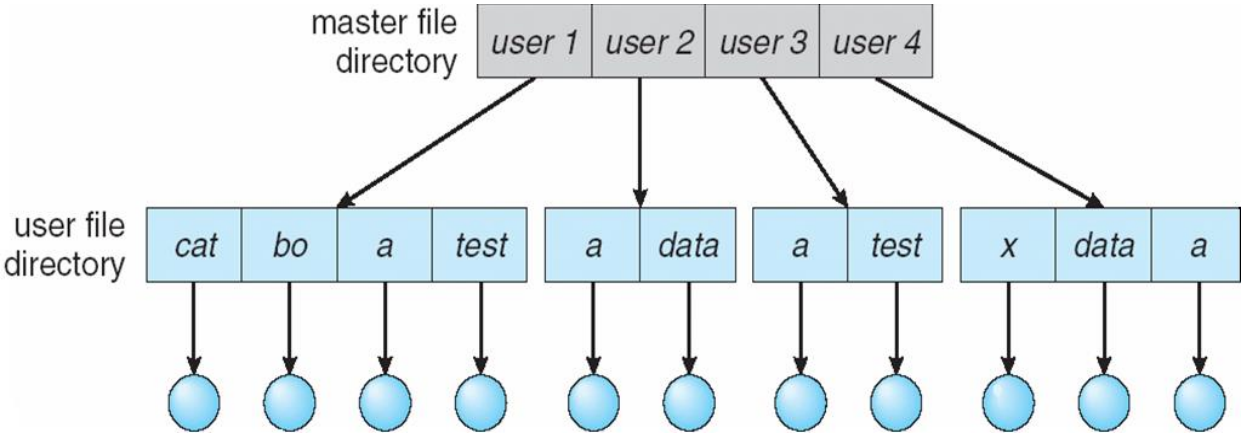


- 명명 문제
- 그룹화 문제

3. Disk and Directory Structure

Two-Level Directory

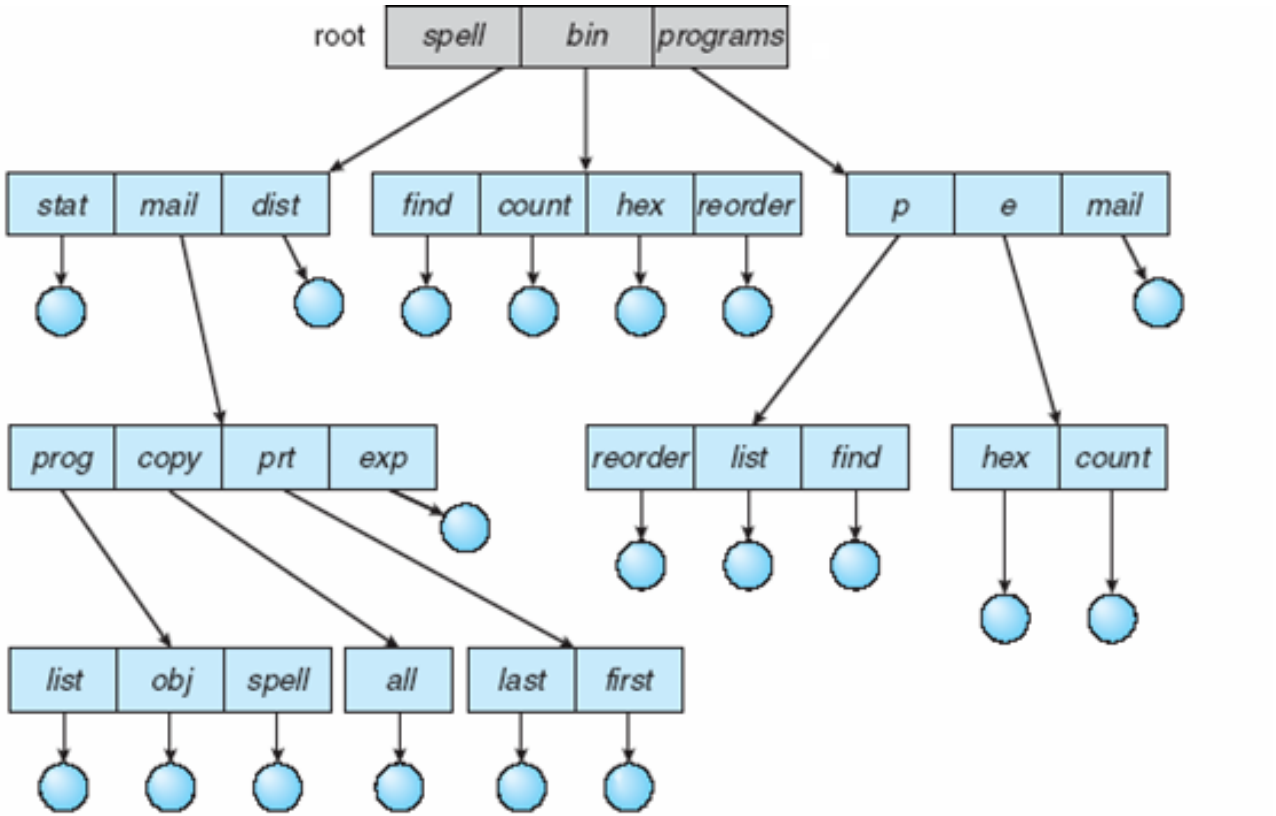
- 각 사용자에 대한 별도의 디렉토리



- 경로 이름
- 다른 사용자에 대해 동일한 파일 이름을 가질 수 있습니다.
- 효율적인 검색
- 그룹화 기능 없음

3. Disk and Directory Structure

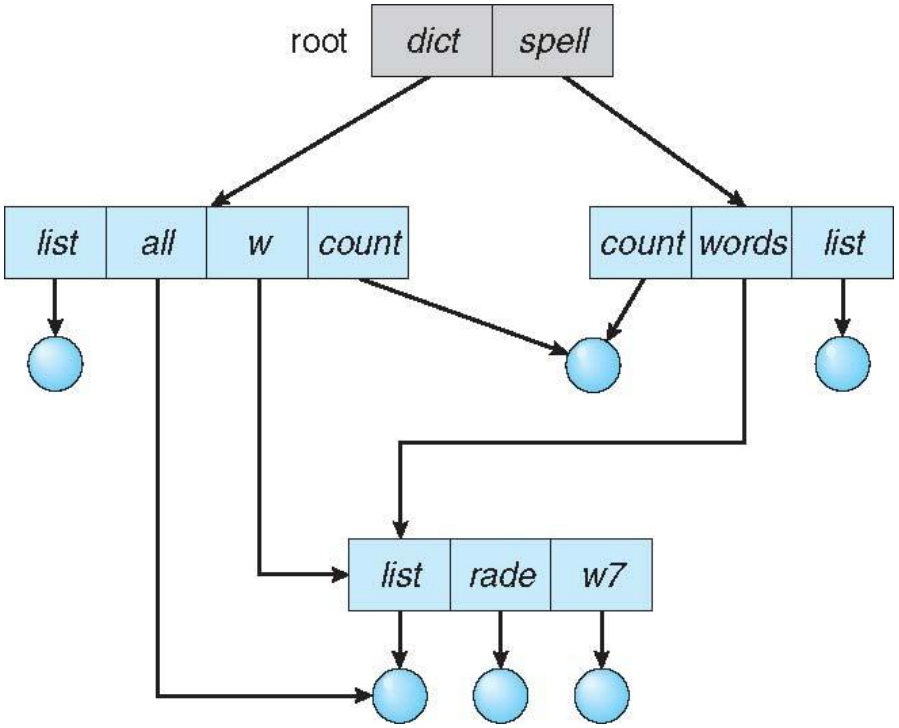
Tree-Structured Directories



3. Disk and Directory Structure

비순환 그래프 디렉터리 Acyclic-Graph Directories

- 공유 하위 디렉토리 및 파일 보유



3. Disk and Directory Structure

비순환 그래프 디렉터리 Acyclic-Graph Directories

- 두 개의 다른 이름(앨리어싱 aliasing)
- dict가 w/list를 삭제하는 경우 ⇒ dangling pointer

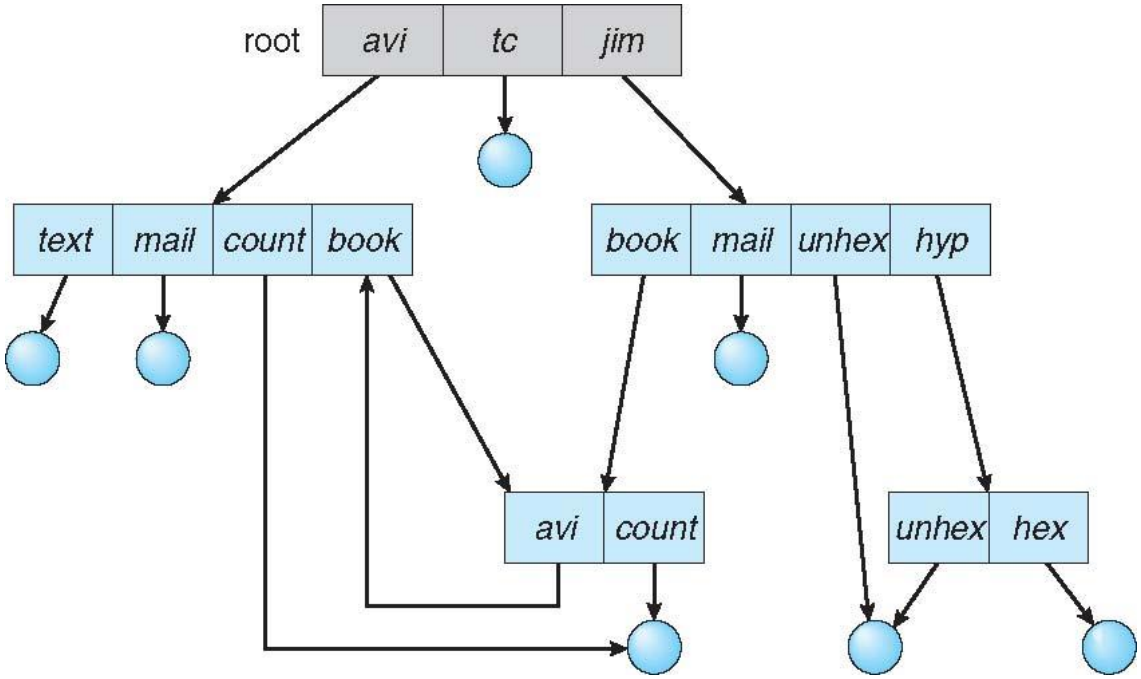
솔루션:

- 백포인터 Backpointers, : 모든 포인터를 삭제할 수 있다.
- 가변 크기는 문제를 기록합니다.
- 데이지 체인 구성을 사용하는 백포인터
- 엔트리 홀드 카운트 솔루션
- 새 디렉토리 항목 유형
- 링크 Link – 기존 파일에 대한 다른 이름(포인터)
- 링크 해결 Resolve the link – 포인터를 따라 파일 찾기

3. Disk and Directory Structure

일반 그래프 디렉토리

- 하위 디렉토리가 아닌 파일에 대한 링크만 허용
  Garbage collection
- 새 링크가 추가될 때마다 주기 감지 알고리즘을 사용하여 정상인지 확인



3. Disk and Directory Structure

Current Directory

- 디렉토리 중 하나를 현재(작업) 디렉토리로 지정할 수 있다.

```
cd /spell/mail/prog
    type list
```

- 파일 생성 및 삭제는 현재 디렉토리에서 수행.
- 새 파일 생성 예

If in current directory is **/mail**
The command
mkdir <dir-name>
Results in:

Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

4. Protection

기본 개념

- 파일 소유자/작성자는 다음을 제어할 수 있어야 한다.
  - 할 수 있는 일
  - 누구에 의해

Types of access

Read
Write
Execute
Append
Delete
List

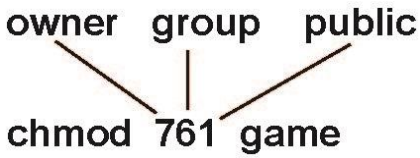
4. Protection

Unix의 액세스 목록 및 그룹

- 액세스 모드: 읽기, 쓰기, 실행
- Unix / Linux의 세 가지 사용자 클래스

			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

- 관리자에게 그룹(고유 이름)을 만들고 G라고 말하고 일부 사용자를 그룹에 추가하도록 요청.
- 파일(예: 게임) 또는 하위 디렉터리에 대해 적절한 액세스를 정의.
- 파일에 그룹 첨부




```
chgrp      G      game
```

4. Protection

샘플 UNIX 디렉토리 목록

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 pbg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2003	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2003	program
drwx--x--x	4 pbg	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

1. 파일과 디렉토리

현재 작업 디렉터리 출력: pwd(print working directory)

- 현재 작업 디렉터리의 절대 경로명을 출력한다.

```
k8s@DESKTOP-RoEQ2U6:~/protection$ cd ~
k8s@DESKTOP-RoEQ2U6:~$ pwd
/home/k8s
k8s@DESKTOP-RoEQ2U6:~$
```

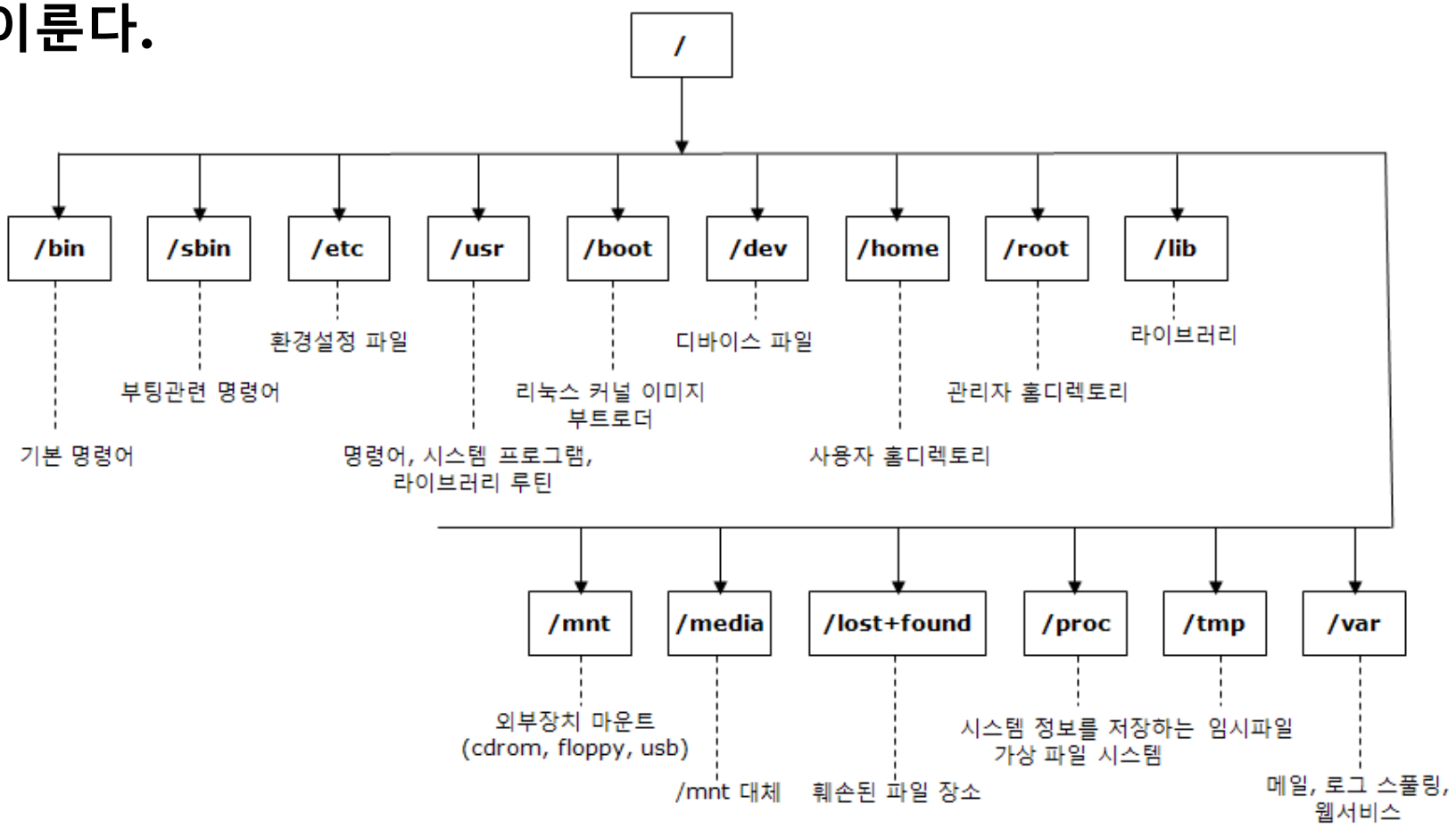
디렉터리 이동: cd(change directory)

- 현재 작업 디렉터리를 지정된 디렉터리로 이동한다. 디렉터리를 지정하지 않으면 홈 디렉터리로 이동한다.

1. 파일과 디렉토리

디렉터리 계층구조

- 리눅스의 디렉터리는 루트로부터 시작하여 트리 형태의 계층구조를 이룬다.



1. 파일과 디렉토리

명령어의 경로 확인: which

- 명령어의 절대경로를 보여준다.

```
k8s@DESKTOP-RoEQ2U6:~$ which ls
/usr/bin/ls
k8s@DESKTOP-RoEQ2U6:~$ which pwd
/usr/bin/pwd
k8s@DESKTOP-RoEQ2U6:~$
```

1. 파일과 디렉토리

디렉터리 리스트

- **ls(혹은 dir) [-aslFR] 디렉터리* 파일***
- 지정된 디렉터리의 내용을 리스트 한다. 디렉터리를 지정하지 않으면 현재 디렉터리 내용을 리스트 한다. 또한 파일을 지정하면 해당 파일만을 리스트 한다.

옵션	설명
-a	경로안의모든파일을나열한다.‘.’으로시작하는파일들도포함된다.
-c	파일최근변경시간에따라정열해서보여준다.
-d	경로안의파일목록을나열하지않고,그경로를보여준다.(이것은쉘스크립트에서유용하게쓰인다.)
-f	경로내용을정열하지않는다:이것은디스크에저장된순으로보여준다.-a와-U옵션과같은뜻이며,-l,-s,-t.옵션과반대의뜻이다.
-i	파일왼쪽에inode번호를보여준다.
-l	파일의모든정보출력
-t	파일시간순으로정열한다.최근파일이제일먼저
-s	파일크기를1Kb단위로나타낸다.POSIXLY_CORRECT환경변수가지정되면,512b단위로지정된다.
-F	파일형식을알리는문자를각파일뒤에추가한다.일반적으로실행파일은"*",경로는"/",심블릭링크는"@",FIFO는"|",소켓은"=",일반적인 파일은없다.

1. 파일과 디렉토리

디렉터리 리스트

- **ls -s**
- -s(size) 옵션
- 디렉터리 내에 있는 모든 파일의 크기를 K 바이트 단위로 출력
- **ls -a**
- -a(all) 옵션
- 숨겨진 파일들을 포함하여 모든 파일과 디렉터리를 리스트
- “.”은 현재 디렉터리, “..”은 부모 디렉터리
- **ls -l**
- -l(long) 옵션
- 파일 속성(file attribute) 출력
- 파일 이름, 파일 종류, 접근권한, 소유자, 크기, 수정 시간 등

```
k8s@DESKTOP-RoEQ2U6:~$ ls -s
total 0
0 c_workspaces  0 java_workspaces  0 protection
k8s@DESKTOP-RoEQ2U6:~$ ls -a
.  .bash_history  .bashrc  .motd_shown  .python_history          c_workspaces     protection
.. .bash_logout   .local  .profile     .sudo_as_admin_successful  java_workspaces
k8s@DESKTOP-RoEQ2U6:~$ ls -l
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 13:05 c_workspaces
drwxr-xr-x 1 k8s k8s 512 May  5 16:36 java_workspaces
drwxr-xr-x 1 k8s k8s 512 May 28 21:28 protection
k8s@DESKTOP-RoEQ2U6:~$
```

1. 파일과 디렉토리

디렉터리 생성: mkdir(make directory)

- **mkdir [-p] 디렉터리+**
- 디렉터리(들)을 새로 만든다.
- 중간 디렉터리 자동 생성 옵션 -p
- 필요한 경우에 중간 디렉터리를 자동으로 만들어 준다.

```
k8s@DESKTOP-RoEQ2U6:~$ mkdir protection
k8s@DESKTOP-RoEQ2U6:~$ ls
c_workspaces  java_workspaces  protection
k8s@DESKTOP-RoEQ2U6:~$ mkdir -p ~/mkdirtest/dir1
k8s@DESKTOP-RoEQ2U6:~$ tree
Command 'tree' not found, but can be installed with:
sudo apt install tree
k8s@DESKTOP-RoEQ2U6:~$ sudo apt install tree
[sudo] password for k8s:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
...
Unpacking tree (2.0.2-1) ...
Setting up tree (2.0.2-1) ...
Processing triggers for man-db (2.10.2-1) ...
k8s@DESKTOP-RoEQ2U6:~$ tree
```

디렉터리 삭제 : rmdir(remove directory)

- **rmdir 디렉터리+**

```
k8s@DESKTOP-RoEQ2U6:~$ tree mkdirtest/
mkdirtest/
└── dir1

1 directory, 0 files
k8s@DESKTOP-RoEQ2U6:~$ rm -rf mkdirtest/
k8s@DESKTOP-RoEQ2U6:~$ ls
c_workspaces  java_workspaces  protection
k8s@DESKTOP-RoEQ2U6:~$
```

1. Protection

간단한 파일 만들기

cat 명령어 사용

- 표준입력 내용을 모두 파일에 저장한다. 파일이 없으면 새로 만든다.

```
k8s@DESKTOP-RoEQ2U6:~/protection$ cat > cs1.txt
Unix is a multitasking, multi-user computer operating system originally
developed in 1969 by a group of AT&T employees at Bell Labs, including
Ken Thompson, Dennis Ritchie, Brian Kernighan, Douglas McIlroy,
and Joe Ossanna.
...

^D
```

touch

- 파일 크기가 0인 이름만 있는 빈 파일을 만들어 준다.

```
k8s@DESKTOP-RoEQ2U6:~/protection$ touch cs2.txt
```

1. Protection

파일 속성

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -asl
total 0
0 drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
0 drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
0 -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
0 -rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.tx
k8s@DESKTOP-RoEQ2U6:~/protection$
```

파일 속성	의미
파일 크기	파일의 크기(K 바이트 단위)
파일 종류	일반 파일(-), 디렉터리(d), 링크(l), 파이프(p), 소켓(s), 디바이스(b 혹은 c) 등의 파일 종류를 나타낸다.
접근 권한	파일에 대한 소유자, 그룹, 기타 사용자의 읽기(r)/쓰기(w)/실행(x) 권한
하드 링크 수	파일에 대한 하드 링크 개수
소유자 및 그룹	파일의 소유자 ID 및 소유자가 속한 그룹
파일 크기	파일의 크기(바이트 단위)
최종 수정 시간	파일을 생성 혹은 최후로 수정한 시간

1. Protection

파일 속성

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -asl
total 0
0 drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
0 drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
0 -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
0 -rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.tx
k8s@DESKTOP-RoEQ2U6:~/protection$
```

파일 종류	표시	설명
일반 파일	-	데이터를 갖고 있는 텍스트 파일 또는 이진 파일
디렉터리 파일	d	디렉터리 내의 파일들의 이름들과 파일 정보를 관리하는 파일
문자 장치 파일	c	문자 단위로 데이터를 전송하는 장치를 나타내는 파일
블록 장치 파일	b	블록 단위로 데이터를 전송하는 장치를 나타내는 파일
FIFO 파일	p	프로세스 간 통신에 사용되는 이름 있는 파이프
소켓	s	네트워크를 통한 프로세스 간 통신에 사용되는 파일
심볼릭 링크	l	다른 파일을 가리키는 포인터와 같은 역할을 하는 파일

1. Protection

접근권한(permission mode)

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -asl
total 0
0 drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
0 drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
0 -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
0 -rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.tx
k8s@DESKTOP-RoEQ2U6:~/protection$
```

- 파일에 대한 읽기(r), 쓰기(w), 실행(x) 권한

권한	파일	디렉터리
r	파일에 대한 읽기 권한	디렉터리 내에 있는 파일명을 읽을 수 있는 권한
w	파일에 대한 쓰기 권한	디렉터리 내에 파일을 생성하거나 삭제할 수 있는 권한
x	파일에 대한 실행 권한	디렉터리 내로 탐색을 위해 이동할 수 있는 권한

1. Protection

접근권한(permission mode)

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -asl
total 0
0 drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
0 drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
0 -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
0 -rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.tx
k8s@DESKTOP-RoEQ2U6:~/protection$
```

- **소유자(owner)/그룹(group)/기타(others)로 구분하여 관리**
- **예: rwx r-x r-x**

1. Protection

접근권한(permission mode)

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -asl
total 0
0 drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
0 drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
0 -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
0 -rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.tx
k8s@DESKTOP-RoEQ2U6:~/protection$
```

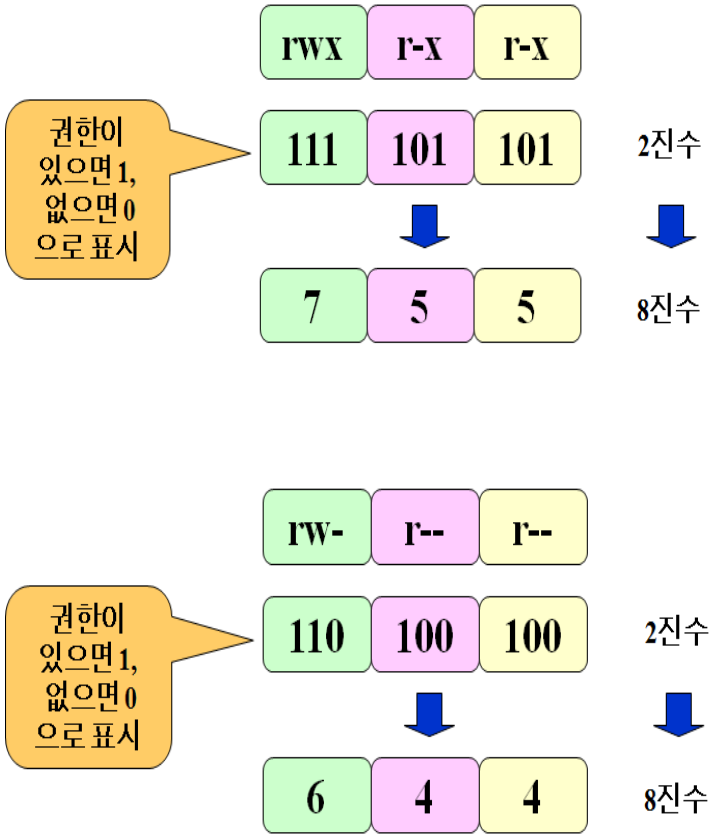
- **rwxrwxrwx 3 칸씩 구분하여 소유자 (owner), 그룹 (group), 그 외 (other) 로 나눈다 . 이 권한은 숫자로 표시할 수 있다 .**

접근권한	의미
rwxrwxrwx	소유자, 그룹, 기타 사용자 모두 읽기,쓰기,실행 가능
rwxr-xr-x	소유자만 읽기,쓰기,실행 가능, 그룹, 기타 사용자는 읽기,실행 가능
rw-rw-r--	소유자와 그룹만 읽기,쓰기 가능, 기타 사용자는 읽기만 가능
rw-r--r--	소유자만 읽기,쓰기 가능, 그룹과 기타 사용자는 읽기만 가능
rw-r-----	소유자만 읽기,쓰기 가능 그룹은 읽기만 가능
rwx------	소유자만 읽기,쓰기,실행 가능

1. Protection

접근권한 변경

- 접근권한 표현: 8진수



접근권한	8진수
rwxrwxrwx	777
rwxr-xr-x	755
rw-rw-r--	664
rw-r--r--	644
rw-r-----	640
rwx------	700

1. Protection

접근권한 변경

```
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ chmod 777 cs1.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rwxrwxrwx 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ chmod 644 cs1.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s   0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$
```

1. Protection

접근권한 표현: 기호

- 기호를 이용한 접근권한 변경

사용자범위 연산자 권한
[u|g|o|a]+ [+|-|=] [r|w|x]+

구분	기호와 의미
사용자 범위	u(user:소유자), g(group:그룹), o(others:기타 사용자), a(all:모든 사용자)
연산자	+(권한 추가), -(권한 제거), =(권한 설정)
권한	r(읽기 권한), w(쓰기 권한), x(실행 권한)
권한	r(읽기 권한), w(쓰기 권한), x(실행 권한)

1. Protection

접근권한 변경

k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s 0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ chmod o-r cs1.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-r----- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s 0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ chmod g+w,o+rw cs1.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-rw-rw- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s 0 May 28 20:24 cs2.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ chmod g-w,o-w cs1.txt
k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al
total 0
drwxr-xr-x 1 k8s k8s 512 May 28 20:24 .
drwxr-x--- 1 k8s k8s 512 May 28 20:17 ..
-rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt
-rw-r--r-- 1 k8s k8s 0 May 28 20:24 cs2.txt

Ubuntu에 python 설치하기

k8s@DESKTOP-RoEQ2U6:~/protection$ whereis python

python:
k8s@DESKTOP-RoEQ2U6:~/protection$ python
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3

k8s@DESKTOP-RoEQ2U6:~/protection$ pip3 --version

Command 'pip3' not found, but can be installed with:
sudo apt install python3-pip

k8s@DESKTOP-RoEQ2U6:~/protection$ sudo apt install python3-pip

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  build-essential bzip2 dpkg-dev fakeroot javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libjs-jquery libjs-sphinxdoc
  libjs-underscore libpython3-dev libpython3.10-dev lto-disabled-list python3-dev python3-distutils python3-lib2to3
  python3-setuptools python3-wheel python3.10-dev zlib1g-dev
Suggested packages:

...

k8s@DESKTOP-RoEQ2U6:~/protection$ pip3 --version

pip 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10)

k8s@DESKTOP-RoEQ2U6:~/protection$ python3

Python 3.10.6 (main, Mar 10 2023, 10:55:28) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>

2. 파일입출력

텍스트 파일 만들고 읽기

- 데이터는 거의 예외 없이 파일에 저장된다. 파일은 텍스트 파일, CSV 파일, Excel 파일 또는 기타 유형의 파일일 수 있다. 이러한 파일에 액세스하고 데이터를 읽는 방법을 알게 되면 Python에서 데이터를 처리, 조작 및 분석하는 도구를 얻을 수 있다. 초당 많은 파일을 처리할 수 있는 프로그램이 있는 경우, 각 작업을 일회성으로 수행하는 것보다 프로그램을 작성하는 것의 결과를 실제로 보게 된다.스크립트가 처리하는 파일을 Python에 알려야 한다. 파일 이름을 프로그램에 하드코딩할 수 있지만 그렇게 하면 다양한 파일에서 프로그램을 사용하기가 어렵다. 파일에서 읽는 다양한 방법은 명령 프롬프트 또는 터미널 창의 명령줄에서 Python 스크립트 이름 뒤에 파일 경로를 포함하는 것이다.
- 이 방법을 사용하려면 스크립트 상단에 내장된 sys 모듈을 가져와야 한다. 스크립트에서 sys 모듈이 제공하는 모든 기능을 사용할 수 있도록 하려면 스크립트 맨 위에 import sys를 추가한다.

2. 파일입출력

단일 텍스트 파일 읽기

- 다음 처럼 텍스트 파일을 만들고 다음 6줄을 작성합니다.

```
k8s@DESKTOP-ROEQ2U6:~/protection$ cat > first_text.txt
I'm
already
much
better
at
Python.
k8s@DESKTOP-ROEQ2U6:~/protection$
```

파일	소스코드
실습환경	Tf38_cpu
소스코드	I'm already much better at Python.
결과값1	
비고	

2. 파일입출력

단일 텍스트 파일 읽기

- sys 모듈을 가져오면 argv 목록 변수를 마음대로 사용할 수 있다. 이 변수는 명령줄 인수 목록(스크립트 이름을 포함하여 명령줄에 입력한 모든 것)을 캡처하여 Python 스크립트로 전달한다. 모든 목록과 마찬가지로 argv에는 인덱스가 있다. argv[0]은 스크립트 이름이다. argv[1]은 명령줄에서 스크립트에 전달된 첫 번째 추가 인수이며, 이 경우 first_script.py가 읽을 파일의 경로가 된다.

파일	소스코드
실습환경	first_script.py
소스코드	#!/usr/bin/env python3 import sys
결과값1	
비고	

2. 파일입출력

단일 텍스트 파일 읽기

파일	소스코드
실습환경	**Tf38_cpu**
소스코드	**print("Output ###: ")** **filereader = open("data/first_text.txt", 'r')** **for row in filereader:** ** print(row.strip())** **filereader.close()**
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 first_script.py Output ###: I'm already much better at Python. k8s@DESKTOP-RoEQ2U6:~/protection$
비고	

2. 파일입출력

readline() 함수 이용하기

- open(“*.txt", 'r')로 파일을 읽기 모드로 연 후 readline()을 사용해서 파일을 읽어 출력한다. readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려준다.

파일	소스코드
실습환경	second_script.py
소스코드	k8s@DESKTOP-RoEQ2U6:~/protection$ cat > second_script.py #!/usr/bin/env python3 import sys print("Output ###: ") F1=open("first_text.txt",'r') line=F1.readline() print(line) F1.close()
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 second_script.py Output ###: I'm
비고	

2. 파일입출력

readline() 함수 이용하기

- open(“*.txt", 'r')로 파일을 읽기 모드로 연 후 readline()을 사용해서 파일을 읽어 출력한다. readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려준다.

파일	소스코드
실습환경	third_script.py

소스코드	k8s@DESKTOP-RoEQ2U6:~/protection$ cat > third_script.py
	#!/usr/bin/env python3
	import sys
	
	print("Output ###: ")
	F1=open("first_text.txt",'r')
	while True:
	  line=F1.readline()
	  if not line:
	  break
	  print(line)
	F1.close()

결과값1	
비고	

2. 파일입출력

readline() 함수 이용하기

- open(“*.txt", 'r')로 파일을 읽기 모드로 연 후 readline()을 사용해서 파일을 읽어 출력한다. readlines 함수는 파일의 모든 줄을 읽어서 각각의 줄을 요소로 갖는 리스트로 돌려준다.

파일	소스코드
실습환경	third_script.py
소스코드	
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 third_script.py Output ###: I'm already much better at Python.
비고	

2. 파일입출력

read 함수 사용하기

- **read()는 파일의 내용 전체를 문자열로 돌려준다.**

파일	소스코드
실습환경	four_script.py
소스코드	#!/usr/bin/env python3 import sys print("Output ###: ") F2=open("first_text.txt",'r') line=F2.read() print(line) F2.close()
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 four_script.py Output ###: I'm already much better at Python.
비고	

2. 파일입출력

추가 모드('a')

- 쓰기 모드('w')로 파일을 열 때 이미 존재하는 파일을 열면 그 파일의 내용이 모두 사라지게 된다. 하지만 원래 있던 값을 유지하면서 단지 새로운 값만 추가해야 할 경우도 있다.
- 이런 경우에는 파일을 추가 모드('a')로 열면 된다. *.txt 파일을 추가 모드('a')로 열고 write를 사용해서 결괏값을 기존 파일에 추가해 적는 예이다. 여기에서 추가 모드로 파일을 열었기 때문에 *.txt 파일이 원래 가지고 있던 내용 바로 다음부터 결과값을 적기 시작한다.

파일	소스코드
실습환경	five_script.py

소스코드	k8s@DESKTOP-R0EQ2U6:~/protection$ cat > five_script.py
	#!/usr/bin/env python3
	import sys
	
	print("Output ###: ")
	F3=open("first_text.txt",'a')
	for i in range(11,20):
	  line = "%d번째 줄입니다. \n" % i
	  F3.write(line)
	F3.close()

결과값1	
비고	

2. 파일입출력

추가 모드('a')

파일	소스코드
실습환경	**five_script.py**
소스코드	
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 five_script.py Output ###: k8s@DESKTOP-RoEQ2U6:~/protection$ cat first_text.txt I'm already much better at Python. 11번째 줄입니다. 12번째 줄입니다. 13번째 줄입니다. 14번째 줄입니다. 15번째 줄입니다. 16번째 줄입니다. 17번째 줄입니다. 18번째 줄입니다. 19번째 줄입니다. k8s@DESKTOP-RoEQ2U6:~/protection$
비고	

2. 파일입출력

추가 모드('a')

- 이런 경우에는 파일을 추가 모드('a')로 열면 된다. *.txt 파일을 추가 모드('a')로 열고 write를 사용해서 결괏값을 기존 파일에 추가해 적는 예이다. 여기에서 추가 모드로 파일을 열었기 때문에 *.txt 파일이 원래 가지고 있던 내용 바로 다음부터 결과값을 적기 시작한다.

파일	소스코드
실습환경	six_script.py
소스코드	k8s@DESKTOP-RoEQ2U6:~/protection$ cat > six_script.py #!/usr/bin/env python3 import sys print("Output ###: ") F4=open("first_text.txt",'r') line=F4.read() print(line) F4.close()
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 six_script.py Output ###: I'm already much better at Python. 11번째 줄입니다. 12번째 줄입니다. 13번째 줄입니다. 14번째 줄입니다. 15번째 줄입니다. 16번째 줄입니다. 17번째 줄입니다. 18번째 줄입니다. 19번째 줄입니다.

2. 파일입출력

with문과 함께 사용하기

- 파일을 열면 위와 같이 항상 close해 주는 것이 좋다. 하지만 이렇게 파일을 열고 닫는 것을 자동으로 처리할 수 있다면 편리하지 않을까? 파이썬의 with문이 바로 이런 역할을 해준다.

파일	소스코드
실습환경	seven_script.py
소스코드	k8s@DESKTOP-R0EQ2U6:~/protection$ cat > seven_script.py #!/usr/bin/env python3 import sys print("Output ###: ") F5=open("noWith.txt",'w') F5.write("Life is too short, you need python!!!") F5.close()
결과값1	
비고	

2. 파일입출력

with문과 함께 사용하기

- 파일을 열면 위와 같이 항상 close해 주는 것이 좋다. 하지만 이렇게 파일을 열고 닫는 것을 자동으로 처리할 수 있다면 편리하지 않을까? 파이썬의 with문이 바로 이런 역할을 해준다.

파일	소스코드
실습환경	seven_script.py
소스코드	
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 seven_script.py Output ###: k8s@DESKTOP-RoEQ2U6:~/protection$ ls -al total 0 drwxr-xr-x 1 k8s k8s 512 May 28 21:23 . drwxr-x--- 1 k8s k8s 512 May 28 21:06 .. -rw-r--r-- 1 k8s k8s 228 May 28 20:22 cs1.txt -rw-r--r-- 1 k8s k8s 0 May 28 20:24 cs2.txt -rw-r--r-- 1 k8s k8s 290 May 28 21:07 first_script.py -rw-r--r-- 1 k8s k8s 255 May 28 21:19 first_text.txt -rw-r--r-- 1 k8s k8s 179 May 28 21:19 five_script.py -rw-r--r-- 1 k8s k8s 125 May 28 21:16 four_script.py -rw-r--r-- 1 k8s k8s 37 May 28 21:23 noWith.txt -rw-r--r-- 1 k8s k8s 256 May 28 21:11 second_script.py -rw-r--r-- 1 k8s k8s 144 May 28 21:23 seven_script.py -rw-r--r-- 1 k8s k8s 125 May 28 21:21 six_script.py -rw-r--r-- 1 k8s k8s 297 May 28 21:12 third_script.py k8s@DESKTOP-RoEQ2U6:~/protection$ cat noWith.txt Life is too short, you need python!!!k8s@DESKTOP-RoEQ2U6:~/protection$
비고	

2. 파일입출력

with문과 함께 사용하기

- 파일을 열면 위와 같이 항상 close해 주는 것이 좋다. 하지만 이렇게 파일을 열고 닫는 것을 자동으로 처리할 수 있다면 편리하지 않을까? 파이썬의 with문이 바로 이런 역할을 해준다.

파일	소스코드
실습환경	eight_script.py
소스코드	k8s@DESKTOP-RoEQ2U6:~/protection$ cat > eight_script.py #!/usr/bin/env python3 import sys print("Output ###: ") with open("With.txt",'w') as F6: F6.write("Life is too short, you need python!!!")
결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 eight_script.py Output ###: k8s@DESKTOP-RoEQ2U6:~/protection$ cat With.txt Life is too short, you need python!!!
비고	

2. 파일입출력

텍스트 파일 쓰기

파일	소스코드
실습환경	nine_script.py

소스코드	#!/usr/bin/env python3
	import sys
	
	print("Output ###: ")
	my_letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
	max_index = len(my_letters)
	filewriter = open("second_text.txt", 'w')
	for index_value in range(len(my_letters)):
	  if index_value < (max_index-1):
	  filewriter.write(my_letters[index_value]+'\t')
	  else:
	  filewriter.write(my_letters[index_value]+'\n')
	filewriter.close()
	print("Output ###: Output written to file")

결과값1	k8s@DESKTOP-RoEQ2U6:~/protection$ python3 nine_script.py Output ###: Output ###: Output written to file k8s@DESKTOP-RoEQ2U6:~/protection$ cat second_text.txt a b c d e f g h i j k8s@DESKTOP-RoEQ2U6:~/protection$
비고	