

CH 12 - 1 [ch1 12(P.20~71)]

Ch 12 : I/O Systems

TODO : PDF 파일 보고 재정리 필요(수업 들었던 거 정리해둔 내용)

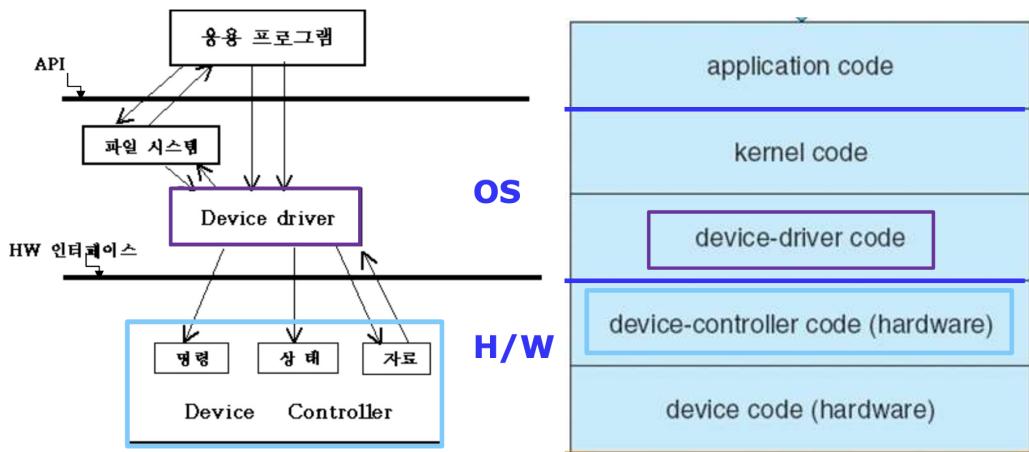
Objectives

- 운영체제의 I/O 하위 시스템 구조 탐색
 - I/O 하드웨어의 원리와 복잡성에 대한 논의
 - I/O 하드웨어 및 소프트웨어의 성능 측면 설명
-

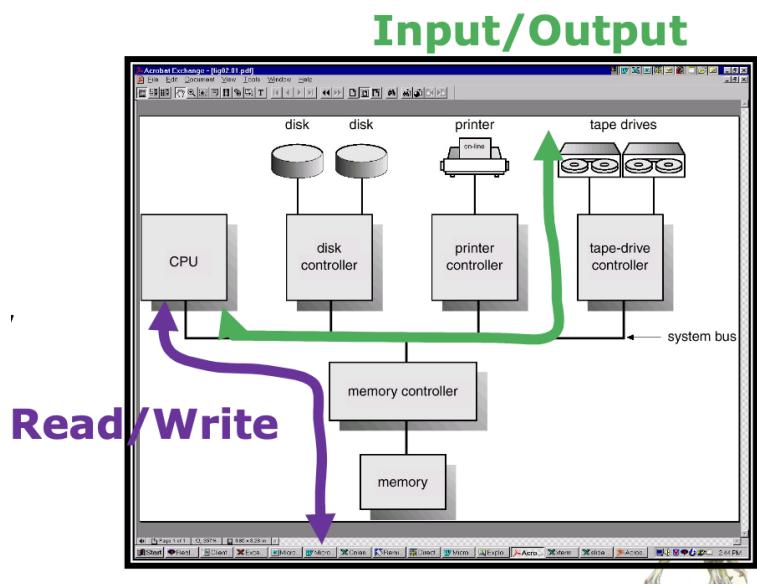
I/O Hardware

- I/O 디바이스의 다양성
 - 저장공간
 - 전달
 - 휴먼 인터페이스
 - 공통 개념
 - Port - 장치를 위한 연결지점
 - Bus - daisy Chain
 - Controller(host adapter) - 포트, 버스, 장치를 작동하는 전자 장치
-

Computer System Operation



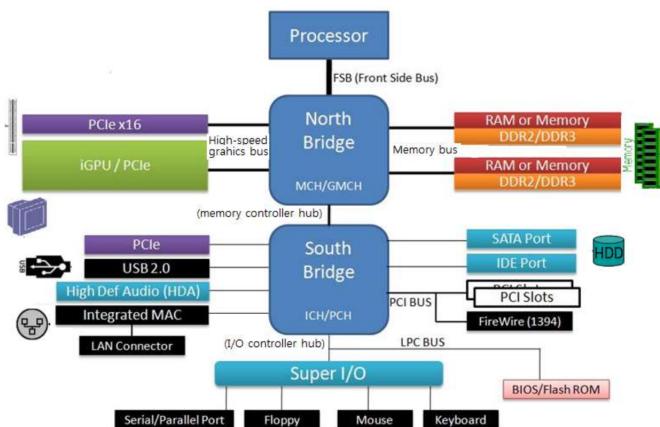
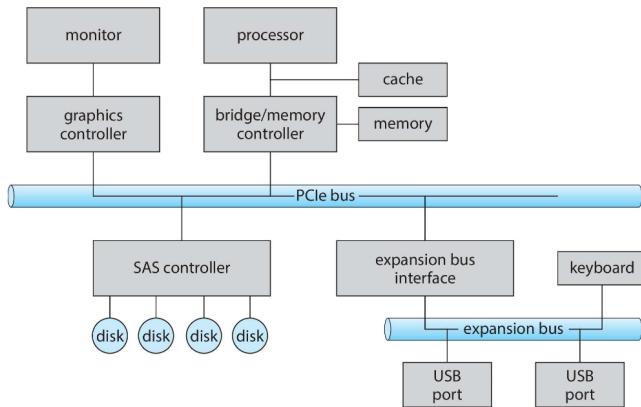
- **Device Driver** - 특정 하드웨어 및 디바이스를 제어하기 위한 프로그램 (블루투스 드라이버, 마이크 드라이버)
- **Device Controller** - 해당 입력/출력 장치를 관리하는 작은 CPU
 - 제어를 위해 State Register, Control Register를 가지며 Data 정보 저장을 위해 Local Buffer를 가지고 있기 때문이다.
 - I/O 작업이 끝났을 경우 인터럽트를 발생시켜 해당 사실을 CPU에 알리는 역할도 한다.



- 입출력 장치와 CPU는 동시 실행이 가능하다.
- 각각의 디바이스 컨트롤러는 특정 디바이스 타입에 종속되며 레지스터와 지역 버퍼를 가진다.

- CPU는 데이터를 디바이스 컨트롤러에 있는 지역 버퍼로 보내고, 그 이후 입출력 장치로 보내진다. (Input/Output)
- CPU는 데이터를 메인 메모리로 보내고 받는다. (Read / Write)

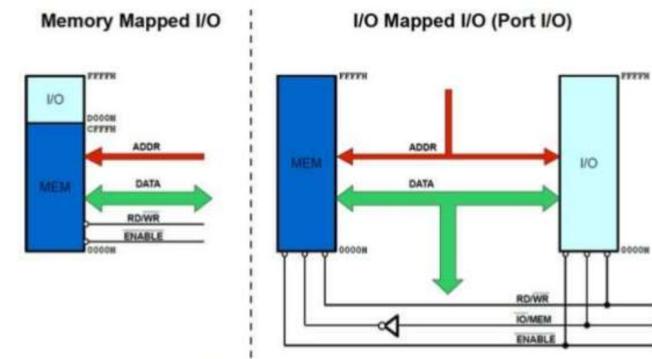
A Typical PC Bus Structure



I/O Hardware(Cont.)

- 입출력 장치의 명령어들이 디바이스들을 제어한다.
- 디바이스들은 보통 디바이스 드라이버가 명령들, 주소들, 그리고 기록할 데이터를 배치하거나, 명령 실행 후에 레지스터들로부터 데이터를 판독하는 레지스터들(또는 로컬 버퍼들)을 갖는다.
 - Data-In Register, Data-Out Register, 상태 레지스터, 제어 레지스터
- 디바이스들은 주소를 가진다.

I/O Mapping



- Direct I/O Instructions :** CPU가 장치의 레지스터에 직접 명령을 작성해서 I/O 명령을 내린다. 데이터 전송 속도가 느리기 때문에 큰 데이터가 필요한 디바이스에 선 사용되지 않는다.
- Memory-mapped I/O :** CPU가 입출력 장치를 액세스할 때, 입출력과 메모리의 주소 공간을 분리하지 않고 하나의 메모리 공간에 취급하여 배치하는 방식이다. 큰 주소 공간에서 유리하다.
- I/O mapped I/O :** 메모리와 입출력의 주소 공간을 분리하여 접근 하는 방식이다. 입출력 장치를 실행시키기 위해 특별히 고안된 CPU 명령어들을 사용한다.

CPU와 컨트롤러 사이 상호 소통을 위한 프로토콜

Q. 호스트(즉, 호스트에서 프로세스를 실행하는 CPU)는 I/O 컨트롤러가 유휴 상태가 된 시점을 어떻게 알 수 있는가?

A1. Polling 혹은 Busy-Waiting (CPU → I/O Device)

- 상태 레지스터에 Busy bit가 존재한다. 컨트롤러는 작업을 수행 중일 때 비지 비트를 1로 설정하고, 작업 수행이 끝나고 다음 명령을 기다리고 있을 때 비트를 0으로 설정한다. CPU는 반복적으로 해당 비트가 0이 될 때까지 읽는다.
- 만약 컨트롤러와 장치가 빠르다면, 해당 방법은 합리적일 수 있다. 하지만 대기 시간이 길어지는 경우 CPU는 다른 작업으로 전환해야 하기 때문에 이 방법은 비효율적이게 된다.

A2. Interrupt (I/O Device → CPU)

- 디바이스 컨트롤러가 CPU에 통지할 수 있도록 하는 하드웨어 메커니즘

Polling

I/O의 각 바이트에 대하여

1. 상태 레지스터가 0이 될 때까지 Busy bit를 읽는다.
2. CPU가 Read 혹은 Write 비트를 설정하고 만약 쓰기인 경우 data-out 레지스터에 데이터를 복사한다.
3. Host가 Command-Ready 비트를 설정한다.
4. 컨트롤러가 busy bit를 설정하고 전달을 수행한다.
5. 전달이 끝나면 controller가 busy bit, error bit, command-ready bit를 0으로 설정한다.

해당 작업은 busy bit 방법이며 장치가 빠른 경우에만 합리적이다. 하지만 장치가 느리다면 비효율적이다.

Interrupt Handling

- 디바이스 컨트롤러가 인터럽트를 발생시킴으로써 CPU에게 동작이 끝났음을 알린다.
 - CPU Interrupt-request line은 I/O Device Controller에 의해 작동된다.
- CPU
 - 실행 중인 작업의 처리를 중지한다.
 - 인터럽트를 처리한다.
 - Interrupt service routine(ISR)
 - Interrupt Handler
 - 중단되었던 작업을 재개한다.

Interrupt Service Routine(ISR)

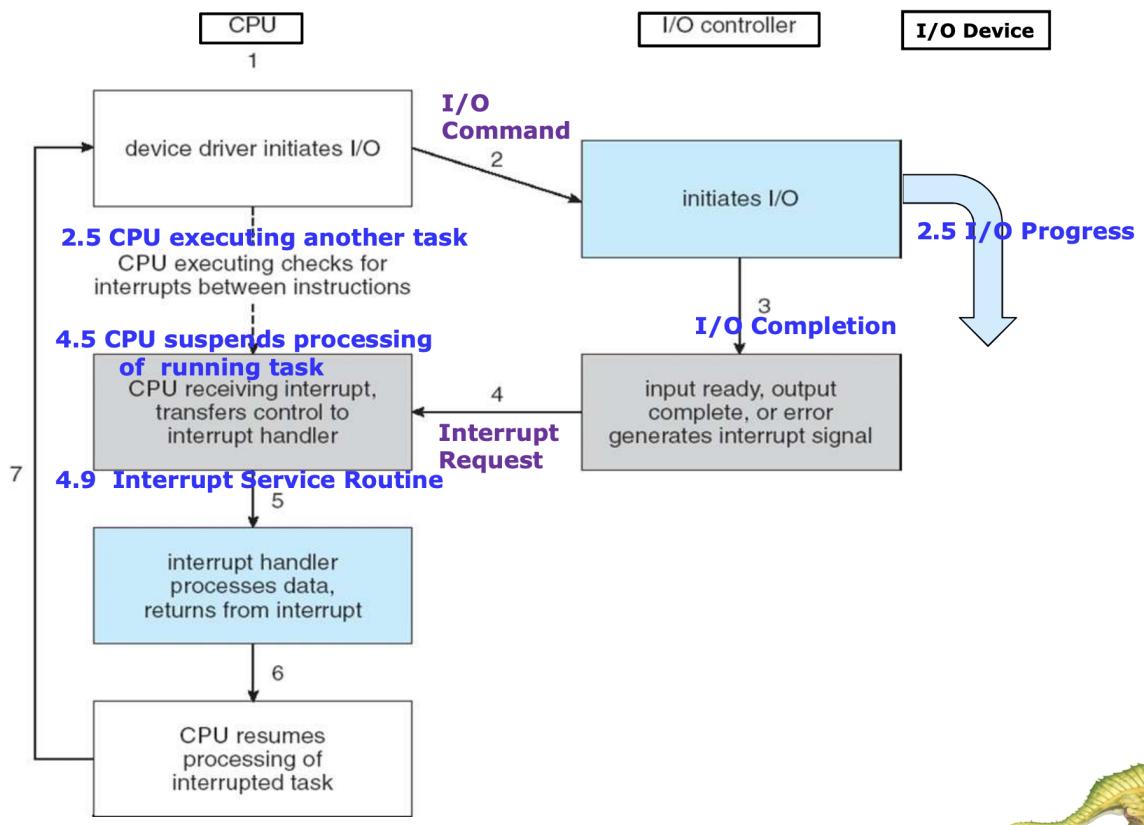
- 레지스터와 프로그램 카운터(중단된 명령어의 주소)를 저장하여 CPU의 상태를 저장한다.
- 어떤 타입의 Interrupt가 발생했는지 결정한다.
 - Polling
 - Polling이란 Interrupt의 발생 여부만 확인한 후 어디서 Interrupt가 발생했는지 하나씩 확인하고 처리하는 방식이다.
 - Vectored Interrupt
 - Vectored Interrupt란 각각의 Interrupt를 미리 만들어두고 해당 Interrupt가 발생했을 때 미리 만들어 둔 Interrupt로 처리하는 방식이다.

- = 인터럽트 벡터를 사용하여 우선 순위에 따라 올바른 핸들러에 인터럽트를 디스패치한다.

Interrupt Handler

- 각각의 인터럽트 유형에 대해 어떤 조치를 취해야 하는지를 결정하는 코드의 별도 세그먼트

Interrupt-Driven I/O Cycle



- CPU와 입출력 장치가 동시에 동작한다는 것이 매우 중요하다.
- 프로세서가 입출력 작업을 요청하면, 그 작업이 완료되기 전까지 기다리지 않고 다른 작업을 수행할 수 있다. 대신, 입출력 장치는 작업이 완료되면 인터럽트를 발생시켜 프로세서에게 알린다.
- Interrupt Service Routine : 인터럽트가 발생했을 시 수행되는 작업

인터럽트의 공통된 기능들

- 인터럽트는 일반적으로 모든 서비스 루틴의 주소를 포함하는 인터럽트 벡터를 통해 인터럽트 서비스 루틴에 제어권을 전달합니다.

- 인터럽트 아키텍처는 인터럽트된 명령의 주소를 저장해야 합니다.
 - 트랩 또는 예외는 오류 또는 사용자 요청으로 인해 발생하는 소프트웨어에서 생성된 인터럽트입니다.
-

I/O 전달의 Overview

Polling

- CPU가 입출력장치의 변화를 지속적으로 읽어들이며 이벤트의 수행 여부를 주기적으로 검사하여 해당 신호를 받았을 때 이벤트를 실행하는 방식이다.

Interrupt

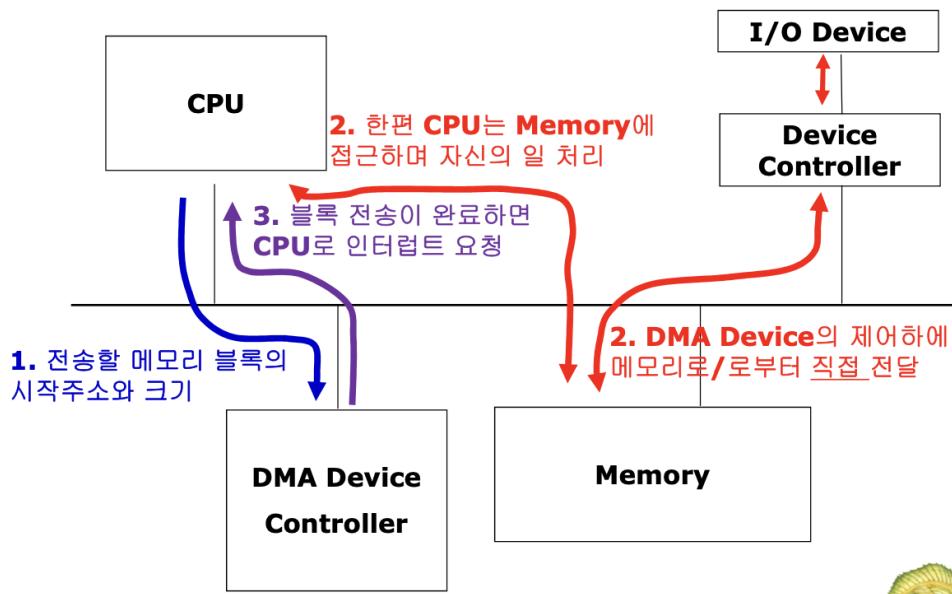
- 디바이스 컨트롤러가 인터럽트를 발생시킴으로써 CPU에게 동작이 끝났음을 알린다.

Direct Memory Access (DMA)

- Block transfer
 - 메모리와 입출력 장치 컨트롤러의 직접적인 전달
-

Direct Memory Access(DMA) Structure

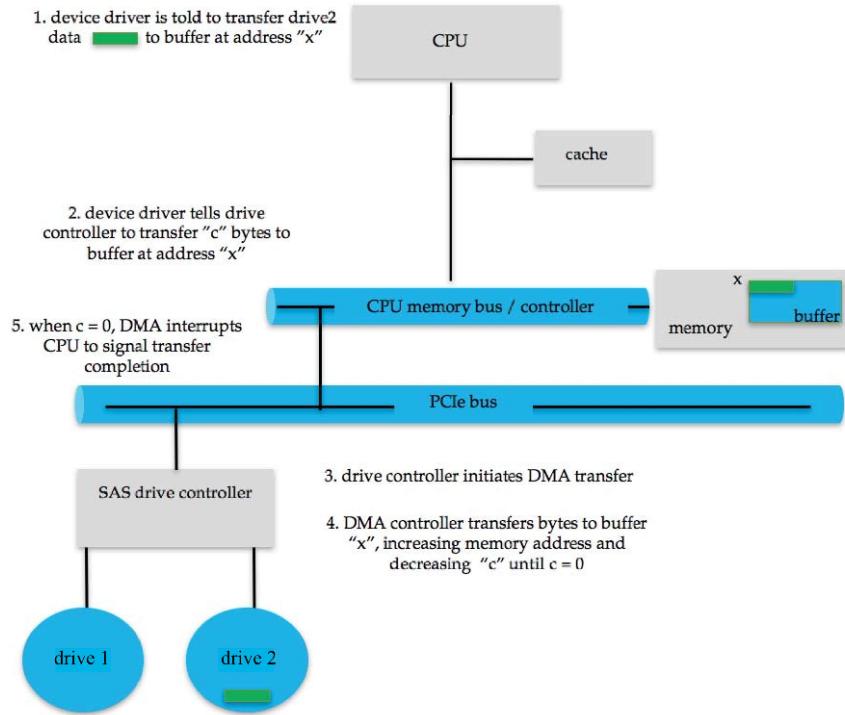
- CPU의 개입 없이 입출력 장치와 기억 장치 사이의 데이터를 전송하는 접근 방식
 - CPU의 개입이 없다 = 할 일이 줄어든다, CPU는 매우 비싼 자원이기 때문에 효율을 높일 수 있는 좋은 방안이 될 수 있다.
 - 장치 컨트롤러는 CPU 개입 없이 버퍼 스토리지의 데이터 블록을 메인 메모리로 직접 전송한다.
 - **블록 단위로 전달한다는 것이 중요하다.**
 - 바이트당 인터럽트 1개가 아니라 블록당 인터럽트 1개만 생성된다.
 - DMA 컨트롤러를 요구한다.
-



CPU STEAL

- CPU와 DMA Device가 희박한 확률로 충돌할 수 있는데, 충돌하게 되면 CPU가 양보 한다.
- 운영체제는 메모리에 DMA 명령어 블록을 작성한다.
 - 자원과 도착 주소
 - mode (Read or Write)
 - 바이트의 수
 - DMA Controller에 명령어 블록의 주소를 작성

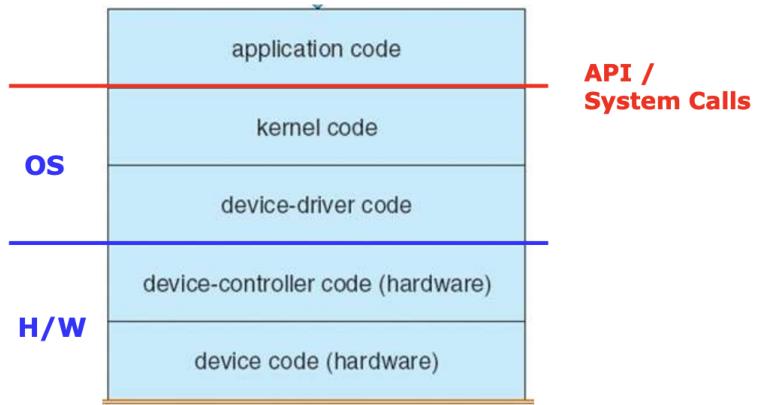
DMA 전달을 수행하기 위한 여섯 단계



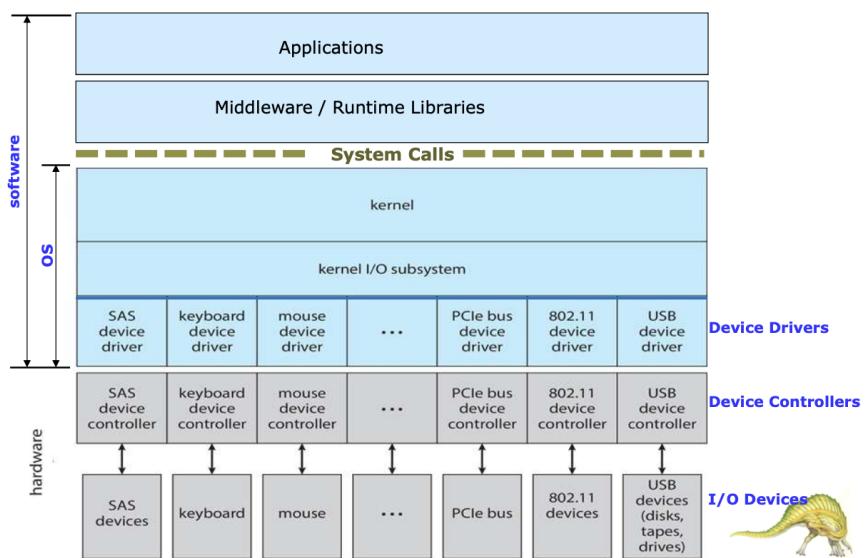
1. 디바이스 드라이버가 디스크 데이터를 주소 X의 버퍼로 전송하도록 지시한다.
2. 디바이스 드라이버가 디스크 컨트롤러에게 디스크에서 주소 X의 버퍼로 C 바이트를 전송하도록 지시한다.
3. 디스크 컨트롤러가 DMA 전송을 시작한다.
4. 디스크 컨트롤러는 각 바이트를 DMA 컨트롤러에게 보낸다.
5. DMA 컨트롤러는 버퍼 X로 바이트를 전송하고 메모리 주소를 증가시키고 C를 감소시킨다. C가 0이 될 때까지 이 과정을 반복한다.
6. C가 0이 되면 DMA는 전송 완료를 알리기 위해 CPU에 인터럽트를 보낸다.

Application I/O Interface

- I/O 시스템 호출은 디바이스 동작을 일반 클래스로 캡슐화하고 애플리케이션과의 하드웨어 차이를 숨깁니다.
- 장치-드라이버 계층은 커널에서 I/O 컨트롤러 간의 차이점을 숨깁니다.
- 이미 구현된 프로토콜을 말하는 새로운 장치는 추가 작업이 필요하지 않습니다.
- 각 OS에는 고유한 I/O 서브시스템 구조와 장치 드라이버 프레임워크가 있습니다.



- 계층의 구분 잘 확인하기

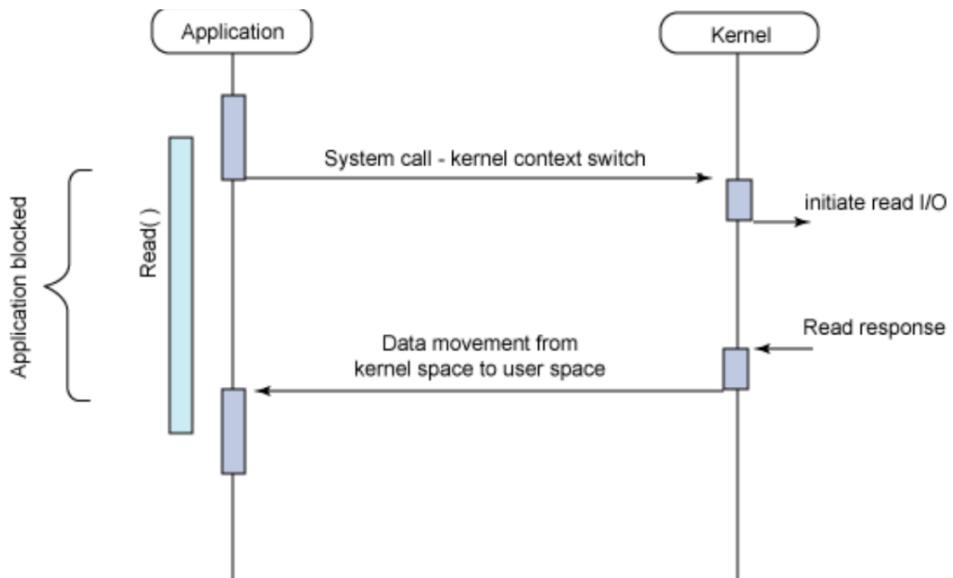


- Device Driver : Device Controller = 1 : 1
- Device Controller : I/O Device = 1 : 1

I/O의 분류

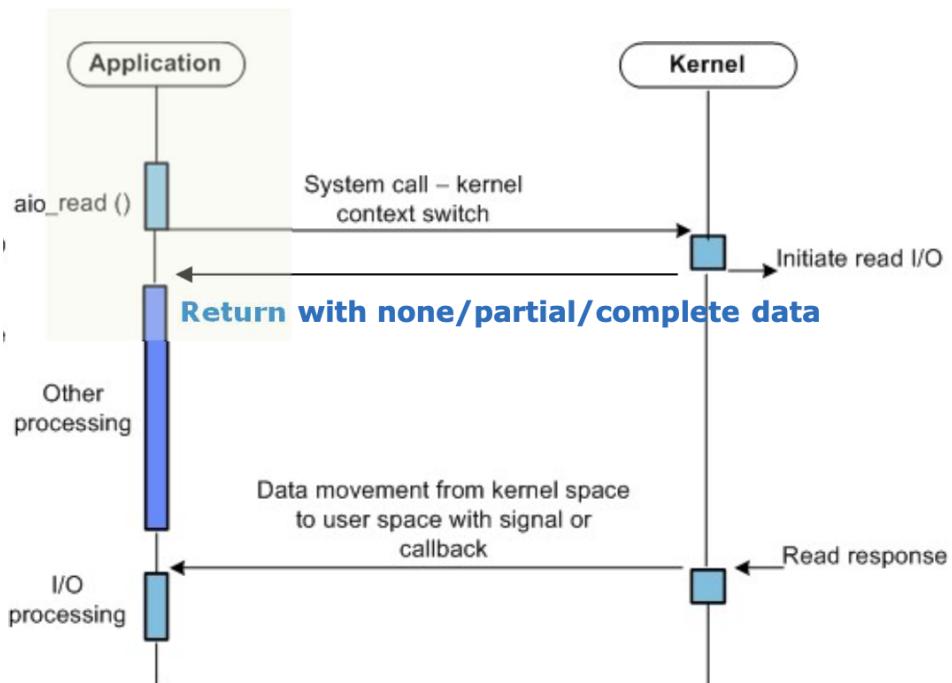
분류기준 1: Blocking I/O vs Non-Blocking I/O

Blocking I/O는 I/O 작업이 진행되는 동안 **User Process**가 자신의 작업을 중단한채, I/O가 끝날때까지 대기하는 방식을 말한다.



1. Process(Thread)가 Kernel에게 I/O를 요청하는 함수를 호출
2. Process(Thread)는 작업 결과를 반환 받을 때까지 대기
3. Kernel이 작업을 완료하면 작업 결과를 반환 받음

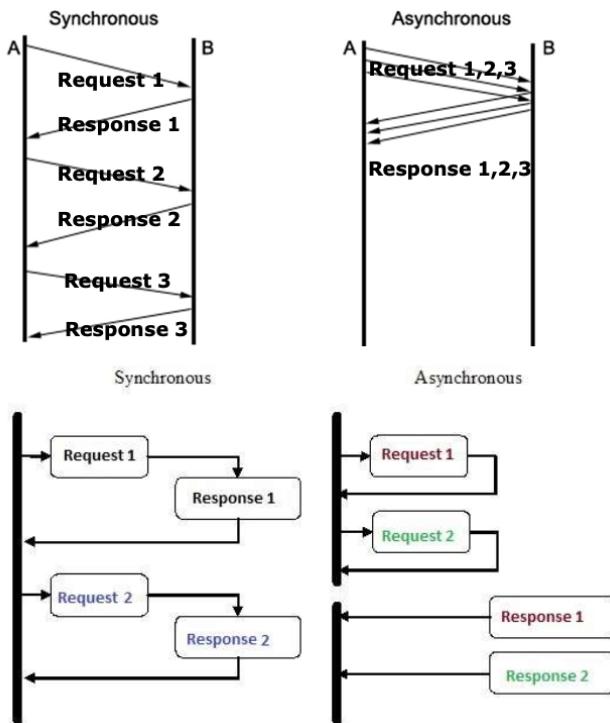
Non-Blocking I/O는 작업이 진행되는 동안 User Process의 작업을 중단하지 않고 I/O 호출에 대해 즉시 리턴하고, User Process가 이어서 다른 일을 수행할 수 있도록 하는 방식을 의미한다.



1. read I/O를 하기 위해 system call
2. 커널의 I/O 작업 완료 여부와 관계없이 즉시 응답a. 이는 커널이 system call을 받자마자 CPU 제어권을 다시 어플리케이션에 넘겨주는 작업
3. 어플리케이션은 I/O 작업이 완료되기 전에 다른 작업을 수행 가능
4. 어플리케이션은 다른 작업 수행 중간중간에 system call을 보내 I/O가 완료되었는지 커널에 요청하고, 완료되면 I/O 작업을 완료

분류기준 2 : Synchronous I/O vs Asynchronous I/O

→ 호출되는 함수의 작업 완료 여부를 누가 신경쓰느냐



- **Synchronous I/O :** 어떤 프로그램이 입출력 요청을 했을 때 디스크 입출력이 완료된 후에야 그 프로그램이 후속 작업을 수행할 수 있는 방식
- **Asynchronous I/O :** 입출력 연산을 요청한 후에 연산이 끝나기를 기다리는 것이 아닌 CPU의 제어권을 입출력 연산을 호출한 그 프로그램에게 곧바로 다시 부여하는 방식을 의미한다.

★ 보통 **Synchronous Blocking I/O**와 **Asynchronous Non-Blocking I/O** 방식을 주로 사용한다.

Nonblocking and Asynchronous I/O

Blocking - 입출력 작업이 끝날때까지 프로세스가 일시중지

- 사용과 이해가 쉽다.
- 요청에 대해 불충분하다.

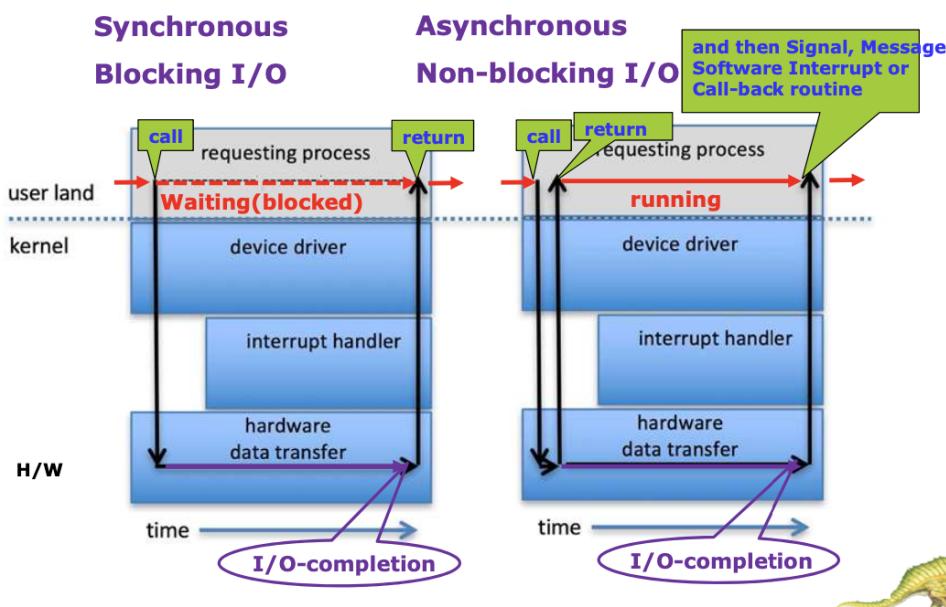
Nonblocking - 입출력 요청은 가능한 한 많이 리턴된다.

- 사용자 인터페이스, 데이터 복사(버퍼화된 입출력)
- 멀티 스레딩을 통해 구현된다.
- 읽거나 쓴 바이트 수로 빠르게 반환
- 데이터가 준비되었을 경우 이를 찾기 위해 select()를 사용하고, 그 다음 전달을 위해 read(), write()를 사용

Asynchronous - 입출력장치가 실행되는 동안 실행되는 프로세스

- 사용하기 어렵다
- 입출력 장치 서브시스템이 입출력장치가 끝날 때 프로세스에게 신호를 보낸다.

Two I/O Methods



Synchronous Blocking I/O

- 입출력장치가 끝날 때까지 스레드에서 다른 작업을 하지 못하며, 다른 작업으로부터 결과 반환시 곧바로 처리된다. (=입출력이 완료될 때까지 대기하며, 작업이 완료되면 결과

를 반환하거나 다음 단계로 진행한다.)

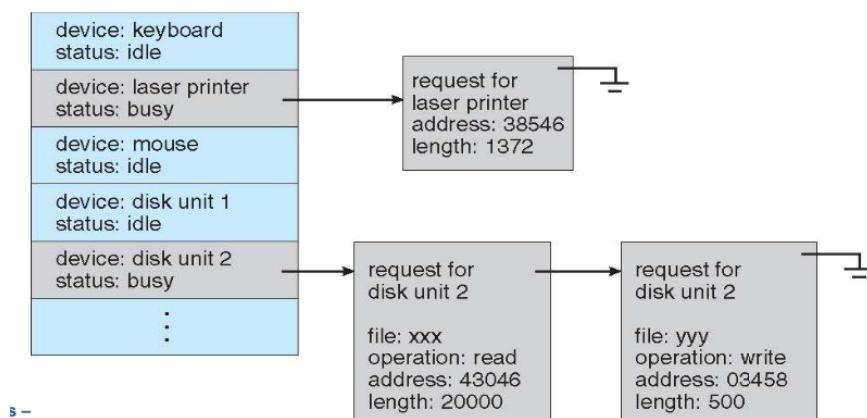
Asynchronous Non-blocking I/O

- 입출력 작업이 진행되는 동안 스레드는 다른 작업을 하며, 결과에 대한 처리는 할 수도 안 할 수도 있다.

Kernel I/O Subsystem

I/O Scheduling

- 입출력장치의 스케줄링은 그들을 실행시킬 때 좋은 순서를 결정하기 위한 요청이다.
 - 전체적인 시스템 성능의 향상
 - 프로세스들 사이에서 공정하게 장치 허가를 공유
 - 입출력장치가 끝날 때까지의 전체적인 대기 시간 줄임
- 몇몇의 입출력장치 요청은 per-device status queue에 의해 정렬된다.

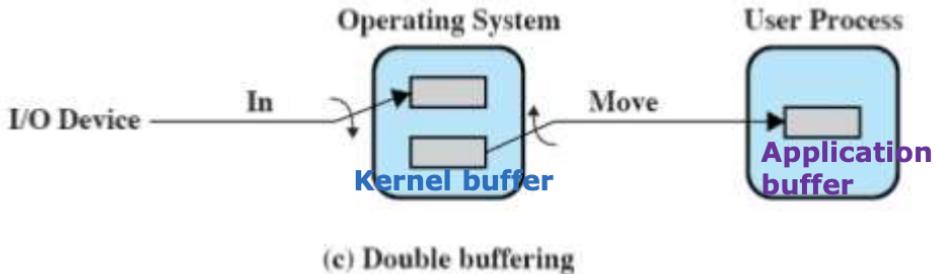


- 버퍼링 : 두 장치 간 또는 장치와 응용 프로그램 간에 전송하는 동안 데이터를 메모리에 저장
 - 장치 간 속도 차이에 대응
 - 파일을 네트워크로 읽어들여서 하드디스크에 저장하는데 네트워크가 너무 느릴 때
 - 버퍼가 생성되어서 네트워크가 읽어오는 데이터들을 버퍼에 저장함. 어느 정도 저장이 되었을 때 디스크에 쓰기 시작함
 - 장치간 전달 사이즈 차이에 대응
 - 보내는 쪽에서 : 큰 데이터를 작은 네트워크 패킷으로 쪼갠 후에 send buffer에 저장함 → 패킷이 네트워크를 통해서 전달 → 받는 쪽에서 : 넘어오는

패킷들을 reassembly buffer에 저장해서 오리지널 데이터로 복구함

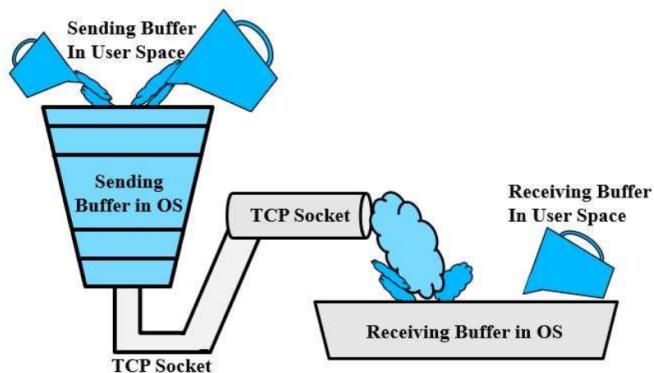
- 데이터 생산과 소비의 시차에 대응하기 위해

Double Buffering



- 버퍼를 두 개 사용한다.
- OS가 다른 버퍼를 비우거나 채우는 동안 사용자 프로세스가 하나의 버퍼로 데이터를 전송할 수 있다.
- 속도 향상의 효과를 확인할 수 있다.

Application Buffer vs Kernel Buffer



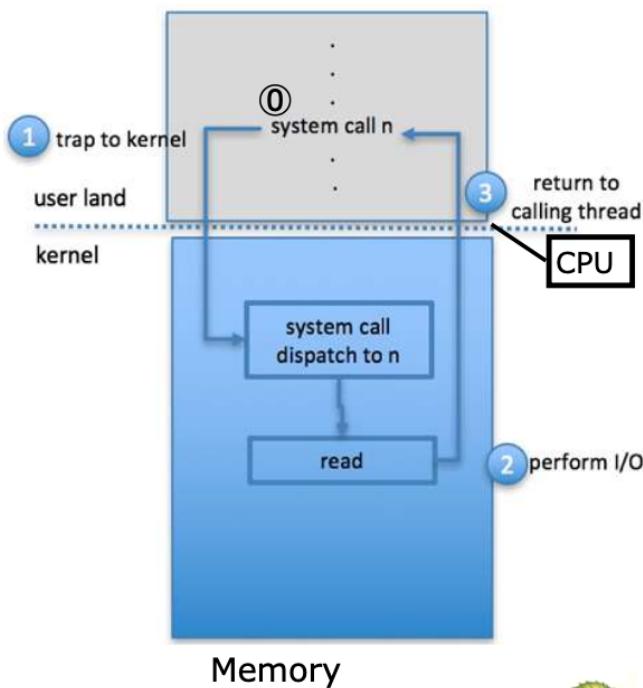
- Application buffer : 프로그램 내 버퍼
- Kernel buffer : OS 내 버퍼
- 애플리케이션 버퍼에서 데이터를 커널 버퍼로 보내 쌓아두고 있다가, read 요청이 들어오면 어플리케이션 버퍼에서 데이터를 가져간다.

I/O Protection

- 사용자 프로세스가 실수로 또는 의도적으로 불법 I/O 지침을 통해 정상 작동을 방해하려고 시도할 수 있다.

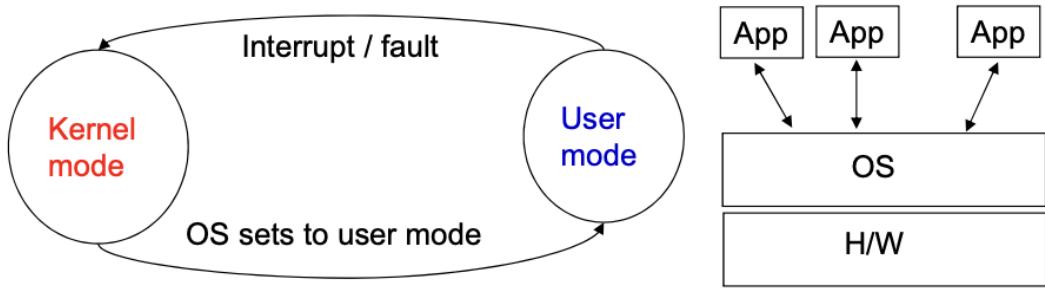
- 권한이 부여되도록 정의된 모든 I/O 명령
 - 입출력장치는 무조건 시스템 콜을 통해 수행되어야 한다.
 - 애플리케이션 프로세스는 자체적으로 입출력을 수행할 자격이 없다.
 - 메모리와 연관되거나 입출력장치 포트 메모리 위치들도 반드시 보호되어야 한다.
-

입출력을 수행하기 위한 시스템 콜의 사용



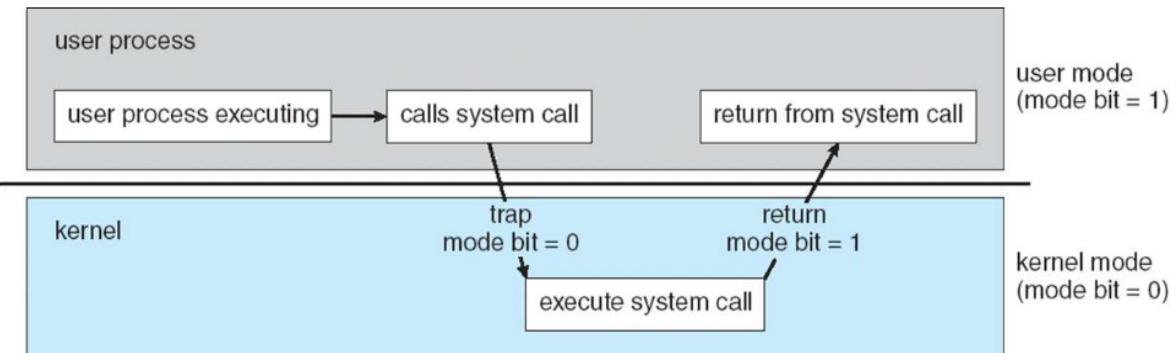
- 사용자 프로그램이 운영체제에 입출력 작업을 요청한다.
 - 일반적으로 인터럽트 벡터의 특정 위치로 트랩의 형태를 취한다.
 - 제어권이 인터럽트 벡터에서 운영체제의 서비스 루틴으로 이동한다.
 - mode bit가 커널 모드로 설정된다.
 - 운영체제는 매개변수가 옳은지 판단하고 입출력 요청을 실행한다.
 - 제어권이 시스템 콜 이후 명령어로 돌아간다.
-

Operating System Operations



- 듀얼 모드 동작은 운영체제 자신과 다른 시스템 컴포넌트들을 보호한다.
 - 유저모드와 커널모드
 - 모드 비트는 하드웨어로부터 제공된다.
 - 시스템이 사용자 코드 혹은 커널 코드를 동작시키고 있음을 구별하는 데 도움을 준다.
 - 권한이 있는 소수의 명령어들은 오직 커널 모드에서만 실행 가능하다.
 - 시스템 콜은 커널로 모드를 바꾸었다가 다시 유저로 돌려준다,

사용자 모드에서 커널 모드로의 변환

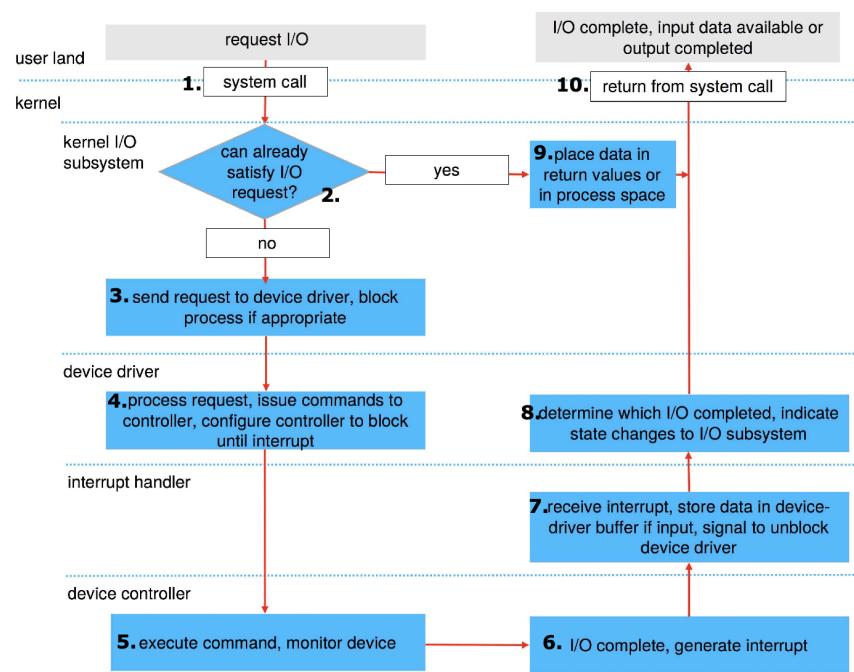


- 무한 루프를 방지하는 타이머
 - 일정 기간 이후 인터럽트를 설정한다.
 - 운영체제가 카운터를 감소시킨다.
 - 카운터 = 0일 경우 인터럽트를 발생시킨다.
 - 일정 프로세스 전에 제어권을 되찾거나 할당된 시간을 초과한 프로그램을 종료하도록 설정한다.

하드웨어 작업에서의 입출력 요청

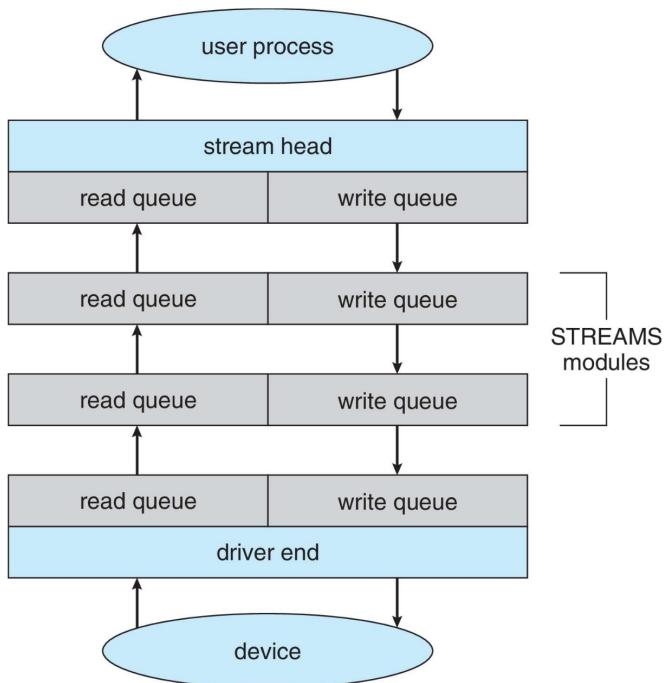
- 프로세스를 위해 디스크로부터 파일을 읽어오는 작업을 고려한다면
 - 장치가 가지고 있을 파일 결정
 - 이름을 장치 표현으로 변경
 - 디스크에서 버퍼로 데이터를 물리적으로 읽어옴
 - 요청한 프로세스가 사용 가능하도록 데이터를 변환
 - 프로세스에게 다시 제어권을 돌려줌

입출력 요청의 라이프 사이클



1. 프로세스가 시스템콜을 통해 IO를 OS에게 요청(인터럽트랑 비슷한 시스템콜)
2. OS는 디바이스 컨트롤러에게 디바이스 드라이버를 통해 일을 시킨다
3. OS는 CPU제어권을 다른 프로세스에게 넘긴다(I/O요청은 오래 걸리니까)
4. 다른 프로세스 수행 중 디바이스가 I/O작업을 끝내면 인터럽트 신호를 보낸다(하드웨어 인터럽트)

STREAMS



- 데이터의 흐름
- 프로그램은 외부에서 데이터를 읽거나 외부로 데이터를 출력하는 작업이 빈번하게 일어나는데, 이때 데이터는 어떠한 통로를 통해서 데이터가 이동된다. 이 통로를 Stream 이라고 한다.
- 각 모듈은 read queue와 write queue를 포함한다.

Performance

- 입출력장치는 시스템 성능에 주된 영향을 미친다.
 - CPU에게 디바이 드라이버, 커널 입출력 코드를 실행하도록 요구한다.
 - 인터럽트 동안 문맥 교환이 발생한다.
 - 컨트롤러 ↔ 메모리, 커널 버퍼 ↔ 애플리케이션 데이터 공간 사이의 데이터 복사 중 메모리 버스를 로드한다.
- 성능을 향상시키기 위해서는
 - 문맥 교환의 횟수를 감소
 - 데이터 카피를 감소
 - 인터럽트를 감소
 - DMA 컨트롤러의 사용 빈도 증가

Device-Functionality Progression

