**USB & USB/IP**

날짜 : 2012.03.08(목)

지도교수 : 이민석 교수님

|  |  |  |
| --- | --- | --- |
| Lemonade | 학번 | 이름 |
| 0592073 | 이상현(조장) |
| 0592051 | 김종욱 |
| 0792047 | 강인구 |
| 0792078 | 이윤재 |

**문서 정보 수정 내역**

|  |  |
| --- | --- |
| 파일명 | USB&USBoverIP |
| 원본 작성자 | 강인구 |
| 수정자 | 강인구, 이윤재 |

**버전 정보**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 날짜 | 수정자 | 버전 | 수정/추가 | 내용 |
| 2012.02.20(월) | 강인구 | 1.0 | 추가 |  |
| 2012.02.21(화) | 강인구 | 1.1 | 추가 | ‘USB 전송’ 추가 |
| 2012.02.22(수) | 이윤재 | 1.2 | 추가 | ‘USB/IP’ 추가 |
| 2012.03.08(목) | 강인구 | 1.3 | 수정 | ‘Protocol’ 추가, Enumeration 수정 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**목차**

1. **USB 개요**
   1. **개요**
   2. **속도**
   3. **전송방식**
   4. **End-Point**
2. **USB 구조**
   1. **접속 토폴로지**
   2. **데이터 전송구조**
3. **USB 전송**
   1. **케이블, 신호처리**
   2. **Protocol**
   3. **Enumeration**
4. **USB/IP**
   1. **개요**
   2. **구조 및 동작원리**
   3. **IP Encapsulation Strategy**
5. **USB 개요**
   1. **개요**

**USB** (Universal Serial Bus / 범용 직렬 버스)

컴퓨터와 주변 기기를 연결하는 데 쓰이는 입출력 표준 가운데 하나이다. 대표적인 버전으로는 USB 1.0, 1.1, 2.0, 3.0 등이 있다.

USB는 다양한 기존의 직렬, 병렬 방식의 연결을 대체하기 위하여 만들어졌다.



키보드, 마우스, 게임패드, 조이스틱, 스캐너, 디지털 카메라, 프린터, PDA, 저장장치와 같은 다양한 기기를 연결하는 데 사용되고 있다. 이러한 기기 연결의 대부분은 표준 연결 방식을 이용하여 이루어지고 있다. USB는 PC를 위하여 개발되었지만 지금은 PDA나 게임콘솔 등에서도 채택되어 사용되고 있고, USB의 전원 공급 기능을 이용하여 충전 용도로도 많이 사용되고 있다. 2008년에는 전 세계적으로 약 20억 개의 USB 장치가 있다.

* 1. **속도**

USB 표준의 이론상 최고 전송 속도는 다음과 같다.

* Low speed(1.5 Mbps)
* Full speed(12 Mbps)
* High speed(480 Mbps)
* Super speed(5 Gbps)

USB 초기에 Low speed로 연결되는 키보드, 마우스 같은 제품들이 있었지만 이제는 거의 쓰이지 않는다. Full speed의 USB 1.0도 1.1 규격으로 업데이트된 다음 USB 2.0으로 전환되었다. 2010년 초 USB 3.0을 적용한 제품들이 나오기 전에는 대부분 USB 2.0을 사용했다. USB 2.0 제품의 이론상 최대 전송속도는 480Mbps이지만 실제 속도는 절반 정도인 240Mbps가 나온다.

USB 3.0 방식을 사용하는 외장 저장장치 시제품의 속도를 측정해보면 연속 읽기 속도가 960Mbps에 달하고 쓰기 속도도 824Mbps가 나온다. 이는 현재 사용되는 하드디스크의 최대 전송 속도를 초과한다.

그러나 USB의 속도에는 치명적인 제약이 있는데 주 컨트롤러(host controller)에 연결된 기기들 간에는 대역폭을 나누어 쓰게 되므로 장치가 늘어날수록 속도는 현저히 떨어진다.

* 1. **전송방식**

- 제어(Control)전송 : 비 동기통신. 디바이스 기본 제어에 사용. Device설정, Address할당 등에 사용.

- 벌크(Bulk)전송 : 비 동기통신. 인터럽트전송이나 등시성전송이 쉬고 있을 때 일반적인 데이터통신에 사용. 정확성이 보장되는 프린터나 디스크 등에 사용.

- 인터럽트(Interrupt)전송 : 비 동기통신. 소량의 데이터 통신을 하는 이벤트 통지 같은 처리에 적합. 마우스, 키보드 등에 사용.

- 등시성(Isochronous)전송 : 동기통신. Real time성이 요구되는 전송방식. 미리 정해진 대역을 확보하여 설계자가 의도한 시간에 전송이 가능. USB 스피커나 헤드폰에 사용.

* 1. **End-Point**

End-Point는 데이터를 저장할 수 있는 버퍼, 데이터의 메모리 블록 또는 USB 칩의 레지스터, End-Point에 저장 된 데이터는 송신 또는 수신하기 위하여 대기중인 데이터로 설명될 수 있다. 하나의 End-Point는 통신의 흐름의 종단이며 각 고유의 ID를 갖고 USB 디바이스 내에 존재하게 된다. 각 USB 논리적 디바이스는 End-Point의 집합체이다.

각 USB 논리적 디바이스는 시스템에 연결될 때 시스템에 의하여 독립적인 주소를 할당 받는다. 그리고 각 End-Point는 설정된 시간 내에 번호를 할당 받게 된다. Endpoint는 데이터 전송의 방향의 정보 (input device – host, output host – device)를 갖는다. 그리고 디바이스의 주소, 번호, 방향의 구성체라 할 수 있다. 또한 전송의 형태를 정의한다.

End-Point는 다음의 특징을 기술하고 있다.

- 버스의 접근 주파수/지연시의 요구사항들

- 대역폭에 대한 요구사항들

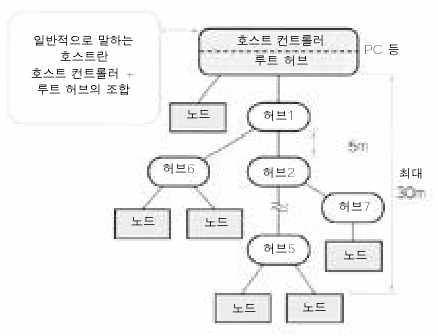
- End-Point들의 수

- 오류 핸들링 행동에서의 요구사항들

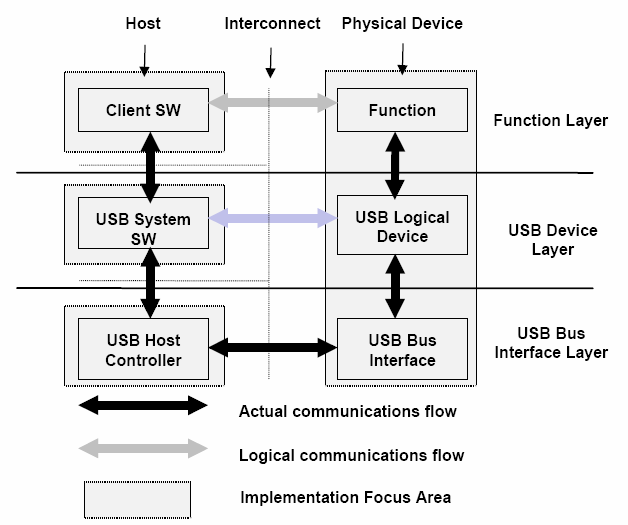
- End-Point가 수용할 수 있는 최대 패킷의 크기

- 전송형태, 전송 방향

1. **USB 구조**
   1. **접속 토폴로지**

****

* 1. **데이터 전송 구조**



- USB의 물리적 디바이스 : USB 케이블 종단에 물려있는 사용자에 있어서 특별한 하드웨어 구조에 따른 Function을 나타낸다.

- 클라이언트 S/W : USB 디바이스에 대응하는 호스트에 대한 소프트웨어를 실행하며 실제의 디바이스에 의해서 결정되는 소프트웨어이며 USB와 함께 제공되는 특정 OS와 타겟 디바이스에 따라서 제공된다.

- USB 시스템 S/W : 특정 OS 에서 USB를 지원하는 소프트웨어이며 OS에는 의존적이지만 특정 클라이언트 소프트웨어 또는 디바이스에 대해서는 독립적이다.

- USB호스트 컨트롤러 : 호스트 측에서 버스 인터페이스를 나타내며, USB 디바이스를 호스트에 결합할 수 있도록 해주는 모든 S/W 또는 H/W를 의미 한다.

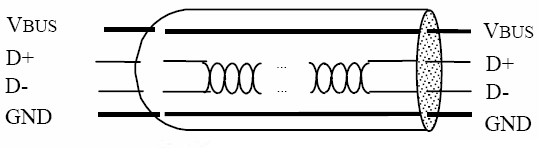
USB 호스트란 USB 호스트 컨트롤러와 USB 시스템 소프트웨어 그리고 마지막으로 클라이언트를 가진다. USB시스템 소프트웨어에서는 USB드라이버(USBD), 호스트 컨트롤러 드라이버(HCD) 호스트 소프트웨어로 구성되어 있다. 호스트는 결합된 디바이스에 대하여 모든 책임과 권한을 가지고 있으며 USB 디바이스는 버스로 접근할 수 있는 사용권을 호스트에 의해 부여 받는다. 또한 호스트는 USB의 토폴로지를 항상 모니터하고 있어야 하고 모든 접근을 통제해야 한다.

USB 디바이스란 USB 인터페이스, USB 논리적 디바이스(Endpoint), Function으로 구성된다. USB 디바이스는 다양한 성능을 가지고 있으나 호스트와의 송수신을 위해 동일한 기본 인터페이스를 가지고 있다. 즉, 호스트가 각종 디바이스를 동일한 방법으로 관리할 수 있다. 확인과 설정을 위해 디바이스는 자신의 정보를 호스트로 보고하며 포맷은 모든 디바이스에 대해 공통적인 부분과 각 디바이스에 특별한 부분으로 구분될 수 있다.

디바이스는 마치 호스트와 직접적으로 연결된 논리적 디바이스로 데이터를 송/수신한다. 클라이언트 S/W와 Function과의 관계를 살펴보면 클라이언트 소프트웨어는 USB에 같이 연결된 다른 디바이스와는 독립적으로 관계된 Function을 직접적으로 다뤄야 한다. Function 들을 처리하기 위해서는 USB 소프트웨어 프로그래밍 인터페이스들을 사용해야 한다. 클라이언트 소프트웨어는 USB에 연결되어 있는 다른 디바이스들과는 독립적이 되며, 이는 설계자들이 디바이스의 설계와 클라이언트 소프트웨어를 하드웨어와 소프트웨어간의 상호 동작에 대해서만 생각 할 수 있도록 해주는 것이다.

1. **USB 전송**
   1. **케이블, 신호처리**

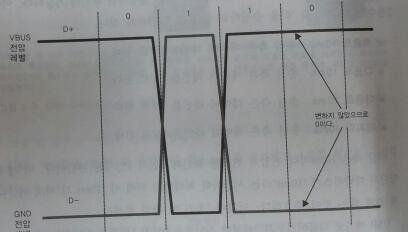
USB 케이블은 아래 그림과 같이 차동형인 데이터 선(D+,D-)과 전원(Vbus), 그라운드(GND)의 4개의 선으로 구성된다.



데이터 신호는 차동 신호이므로 D+와 D- 신호 선은 트위스트 페어 선으로 하고 외부의 노이즈를 줄이기 위하여 케이블에 실드 하였다. 로우 스피드용 케이블은 트위스트 페어나 실드가 필요하지 않다(케이블 길이 3m 제한됨). 전원 핀은 +5v에서 최대 500mA의 전류를 흘린다. 그리고 각 선은 색을 다르게 지정되어있다. USB커넥터의 치수도를 나타낼 수 있는데 시리즈 A 플러그는 상류 측에서 접속하고 시리즈B 플러그는 하류 측에 접속한다.

다음으로 케이블의 핀 배치를 보면 전원용인 1핀과 4핀의 길이는 신호용에 비해 길고 2핀과 3핀은 짧다. 이는 삽입 시 전원이 먼저 공급된 다음 신호 선이 접속되도록 하고 반대로 뺄 경우 신호 선이 먼저 절단된 다음 전원이 절단되게 하여 디바이스가 파괴되지 않도록 하였다.

2개의 데이터라인으로 1비트를 표현한다. 이 2개의 데이터라인(D+와 D-)의 전압변화로 0인지 1인지를 판단한다. 연속으로 0을 보낼 때는 전압을 변화하지 않는다. 하지만 동기 시키기 위해 6비트 연속 0이면 무조건 1로 만들고, 이 1은 동기 시키는 데만 사용하고 데이터로 다루지 않는다(비트 스태핑). 1비트별 시간간격은 로우/풀/하이 스피드 별로 다르다.



* 1. **Protocol**

USB는 Frame 단위로 전송되는데 하나의 Frame은 여러 개의 Transaction으로 구성이 되고 이 Transaction은 USB 상에서 돌아다니는 여러 개의 Packet으로 구성이 됨. (Frame > Transaction > Packet)

* **Frame :** 1ms마다 발생하는 전송단위 (SOF와 Transaction 들로 이루어짐)

**<SOF>** (Start Of Frame) Packet – 24bit

|  |  |  |
| --- | --- | --- |
| PID  (8 bit) | Frame Number  (11 bit) | CRC5  (5bit) |

PID : Packet ID Frame Number : Frame 번호 CRC5 : Error 유무 확인

* **Transaction :** Host와 Device간의 전송단위 (3개의 Packet으로 이루어짐)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Token | | | | Data | | Handshake | | |
| IN | OUT | SOF | SETUP | Data0 | Data1 | ACK | NAK | STALL |

* **Packet :** 전송의 최소단위

<Token Packet>

|  |  |  |  |
| --- | --- | --- | --- |
| PID  (8 bit) | ADDR  (7 bit) | ENDP  (4 bit) | CRC5  (5 bit) |

<Data Packet>

|  |  |  |
| --- | --- | --- |
| PID  (8 bit) | Data  (1~1023 byte) | CRC16  (16 bit) |

<Handshake Packet>

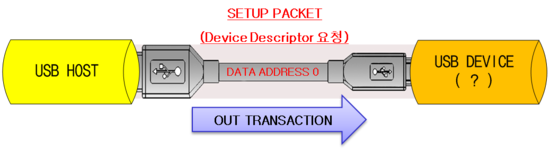
|  |
| --- |
| PID  (8 bit) |

* 1. **Enumeration**

어플리케이션이 디바이스와 통신하기 위해 디바이스에 관한 정보를 얻고 적합한 디바이스 드라이버를 결정하는 정보 교환 과정.

디바이스 주소할당, 디바이스로부터 descriptor 읽기, 적합한 디바이스 드라이버 선정과 로딩, 디바이스 전원 요구사항과 인터페이스 지정하는 컨피규레이션 선택 등이 있다.

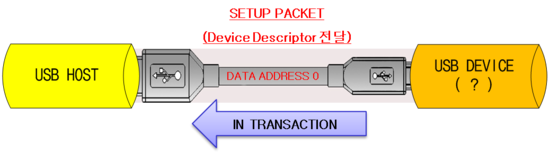
* + - 1. 사용자가 디바이스를 USB 포트에 연결하고, 허브는 디바이스를 감지한다.
      2. 호스트 PC 가 Data Address0를 통해 Device Descriptor 요청 (SETUP Packet)



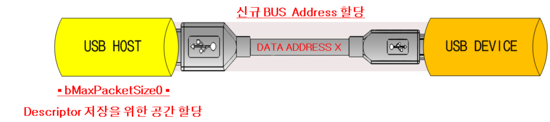
* + - 1. 호스트 명령을 디코드하고 프로그램 메모리에서 Descriptor 검색.



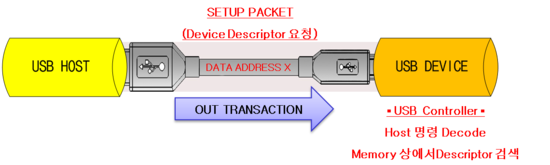
* + - 1. Device Descriptor 전송.

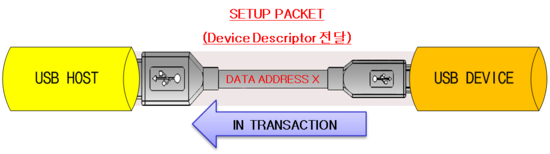


* + - 1. HOST가 신규 Bus Address 할당.

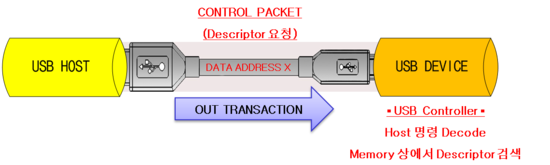


* + - 1. 새 Data Address를 통해 Device Descriptor 요청(SETUP Packet) 및 수신





* + - 1. Device Descriptor 제외한 나머지 Descriptor (컨피규레이션 Descriptor와 리포트 Descriptor) 요청 및 수신

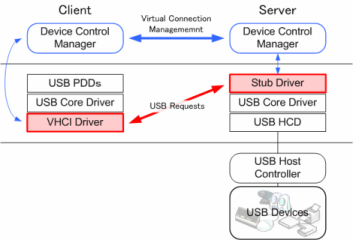


1. **USB/IP**
   1. **개요**

IP네트워크를 통해 일반적인 USB디바이스를 공유하는 시스템. 원격 USB디바이스로는 기존의 USB디바이스 드라이버와 애플리케이션을 수정 없이 사용 가능하다. Host 컴퓨터는 직접 USB디바이스가 연결된 것처럼 원격 USB디바이스를 사용 할 수 있다.

USB/IP는 현재 리눅스 디바이스 드라이버와 오픈 소스 라이선스 GPL하에 사용할 수 있게 구현 된다. USB/IP의 능력은 로컬 네트워크 내에서 동기 / 비 동기가 모두 가능한 장비를 포함한 모든 종류의 기존 리눅스 디바이스들을 수정 없이 사용하기에 실용적이다. 현재 USB/IP는 리눅스 커널 2.6.28 이후 버전의 지원이 필요하다.

* 1. **구조 및 동작원리**



VHCI driver는 Client host에서 USB HCD역할을 하고, Stub driver는 Server host에서 USB PDD의 역할을 한다. VHCI driver는 원격 USB장치가 IP네트워크를통해 Client host에 연결된 경우 USB Core Driver의 포트 상태 변경을 통지하고, USB Root Hub의 동작을 에뮬레이트한다. USB/IP는 USB device 번호가 client device 번호와 server device 번호 사이에서 번역되는 것을 보장한다. 추가적으로, SETADDRESS와 CLEAR\_HALT와 USB요청은 USB Core driver가 관리 데이터를 올바르게 업데이트 할 수 있도록 인터셉트하고 있다.

IP 네트워크를 통한 모든 URB들의 전송은 TCP 프로토콜을 통해서 간다. 그러나 버퍼링 지연을 방지하고, 가능한 빨리 TCP/IP 패킷을 전송하기 위해 Nagle algorithm은 비활성화 돼있다. 현재 USB/IP는 통신하기 위해 UDP를 사용하지 않게 구현돼있다. 이것은 USB와 UDP/IP의 전송 오류의 특성이 매우 다르기 때문이다. host controller는 isochronous 트랜잭션 실패 시 재전송 하지 않지만, USB PDD들과 devices는 거의 트랜잭션 실패가 발생하지 않는다. 또한 devices에 어떤 물리적 문제가 발생하지 않는 한 트랜잭션 실패는 거의 USB에서 발생하지 않는다. 그러므로 일반적으로 transport layer는 URB들이 올바르게 도착하거나 손실된 패킷 재전송을 보장해야 한다.

device 공유 시스템의 핵심디자인은 운영체제에서 멀티 가상 버스 지원이다. 가상버스는 각각의 복잡한 주변 장치 버스로 구현된다.

* 1. **Encapsulation Strategy**

VHCI dirver 는 USB HCD와 동등하고, URB들을 큐에 넣는 작업을 하고 있다. URB는 VHCI driver에서 USB/IP요청 블록으로 변환되고 원격 컴퓨터로 전송된다. Stub driver는 USB PDD들의 새로운 유형으로 추가된다. Stub driver는 원격 머신으로부터 들어오는 USB/IP 패킷 디코딩을 하고, URB들을 뽑아낸다. 그리고 local USB devices로 보내는 역할을 한다.

이 전략을 사용해, 직접 연결된 USB 장치와 원격 USB 장치 간의 인터페이스 차이를 완전히 HCD 계층으로 숨길 수 있다. USB PDD들, 다른 driver들, 그리고 어플리케이션은 원격 USB device들을 정확히 같은 방식으로 사용할 수 있다. 일단 USB Core driver를 열거하고 local로 연결된 것처럼 USB PDD들 및 어플리케이션이 장치에 액세스 할 수 있도록 수정되지 않은 원격 USB devices를 초기화 한다.