|  |
| --- |
| **USB Host Driver** |
| 구조설계서 |
|  |
|  |
| **2022-01-14** |
| **김성후** |

이 문서는 **USB Host Driver** 개발을 위한 구조설계서이다.

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | 2022-01-14 | 김성후 | 초기 문서 생성 |
| 0.2 | 2022-01-16 | 김성후 | Interim 작성 |
| 0.3 | 2022-01-20 | 김성후 | Interim 1차 수정 |
| 0.4 | 2022-01-22 | 김성후 | Pre-final 작성 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[1. 시스템 개요 3](#_Toc479150191)

[2. 요구사항 4](#_Toc479150192)

[2.1. 기능적 요구사항 4](#_Toc479150193)

[2.2. 비기능적 요구사항 4](#_Toc479150194)

[2.3. 품질 속성 5](#_Toc479150195)

[부록 8](#_Toc479150198)

[A. 도메인 모델 9](#_Toc479150199)

[B. 품질 시나리오 10](#_Toc479150200)

[C. 품질 시나리오 분석 11](#_Toc479150201)

[D. 후보 구조 12](file:///C:\Users\김성후\OneDrive\문서\architect\reference\requirement\구조설계서(배포).docx#_Toc479150202)

[E. 후보 구조 평가 13](file:///C:\Users\김성후\OneDrive\문서\architect\reference\requirement\구조설계서(배포).docx#_Toc479150203)

[F. 최종 구조 설계 14](#_Toc479150204)

# 시스템 개요

* 1. **시스템 개요**

USB란 Universal Serial Bus의 약어로, CTI(Computer Telephony Integration) 산업의 성장을 위해 인텔, 마이크로소프트, COMPACT, IBM, NEC, DEC 및 Nortel 등의 회사가 개발하였다. 플러그 앤 플레이를 위한 PC 주변장치의 Bus 규격으로서, 새로운 주변기기가 접속되었을 때 재부팅이나 셋업 과정 없이 자동인식으로 최대 127개의 장치를 연결할 수 있으며, 기존 흔히 사용되던 시리얼 포트에 비해 데이터 전송속도도 빠르게 향상되었다. 규격이 마련됨과 동시에 인텔과 VIA, SiS, ALi 등과 같은 메인보드용 칩셋 제작사들은 USB 포트를 지원하는 칩셋을 발표하고 그들을 주축으로 지금처럼 PC에 USB 포트가 기본적으로 장착되기 시작됐다.

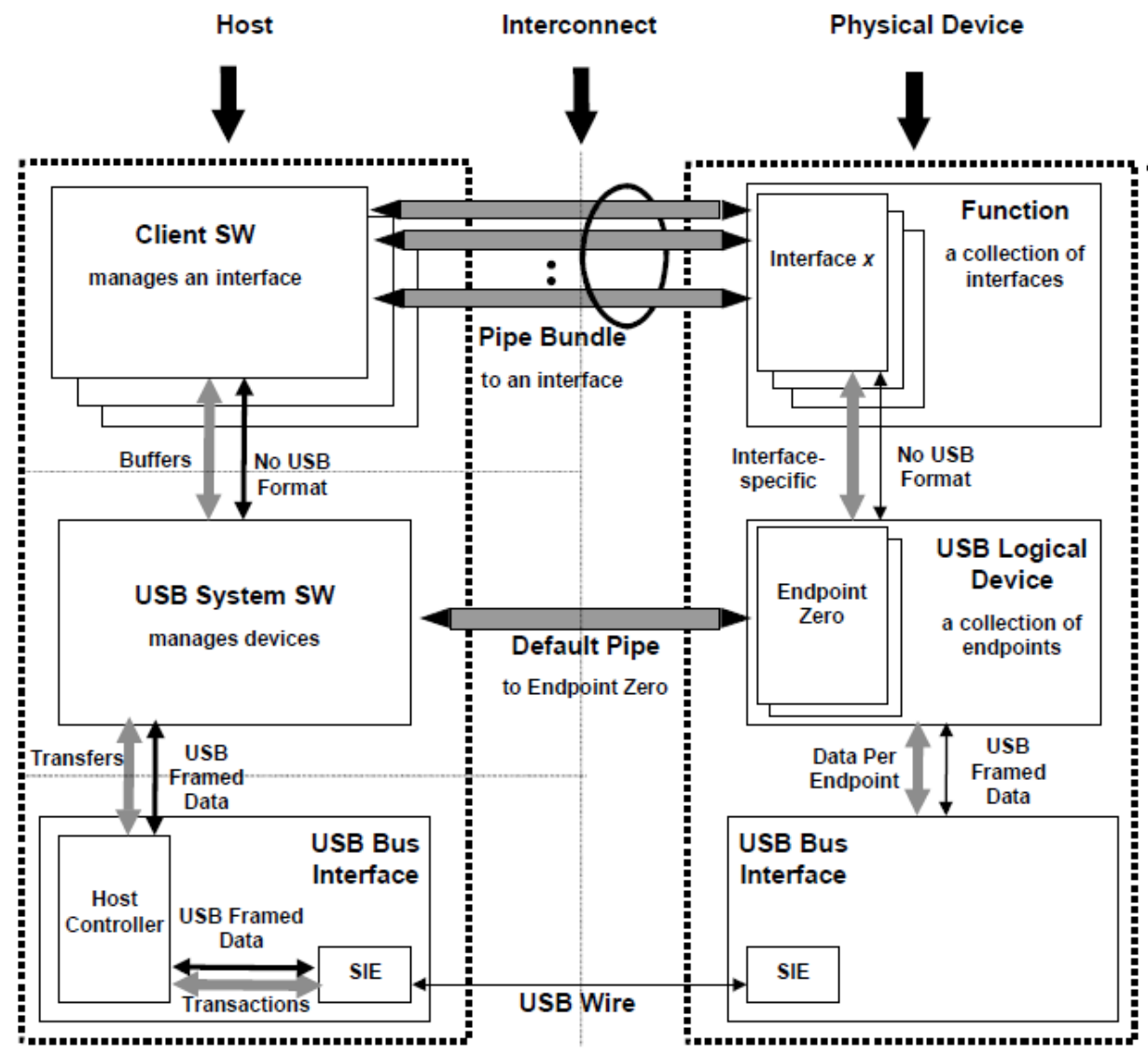


그림 - USB 시스템의 구조

USB는 하나의 Host와 다수의 USB Device를 연결한다. 이들은 물리적인 USB Wire를 통해 연결되어 서로 통신하며 그 위에 논리적인 계층을 가진다. 우선 물리적 계층의 USB Bus Interface의 Host Controller는 통신을 위한 하드웨어 적인 요소를 담당한다. 그리고 그 위의 레이어의 USB System SW는 USB 장치들을 관리하기 위한 Logic을 담당하며 Peer 개념의 USB Logical Driver와 논리적인 연결을 가지고 있다. USB Logical driver의 통신을 위한 접점을 Endpoint Zero라고 부르며, 논리적 개념의 연결을 Default Pipe라고 명명하고 있다. 또 그 상위 레이어는 Client Software와 USB Device의 Function과의 논리적인 연결을 가지게 된다. Client Software는 Host의 운영체제 내의 소프트웨어 혹은 Host에 탑재된 소프트웨어를 의미하며, USB Device의 Function은 해당 장치의 고유한 기능을 구현한 소프트웨어를 의미한다. 각각의 논리적 연결은 실제로는 물리적 계층의 USB Wire를 통해 전기적인 신호로 전달되지만 논리적 개념의 추상화를 통해 복잡도를 낮춰주는 역할을 한다.

* 1. **사업 환경**

USB Device가 PC 업계의 인터페이스 표준으로 확고한 지위를 가지면서, 매해 수십억 개의 새로운 USB Device들이 시장에 출시되고 있다. 그에 따라 USB 이전 시대의 보편적인 연결 방식이었던 시리얼 포트에 대한 PC 제품들의 지원이 점차 감소하고 있으며, PC에서는 다수의 USB Port를 제공하고 있다. 또한 코로나 19의 영향으로 한동안 스마트폰과 태블릿 PC에 밀려 수요가 감소하던 Note PC에 대한 수요는 재택근무와 가정 학습의 증가에 따라 2021년 최고 성장률을 기록하였다. 이러한 환경하에서 본 과제에서는 Note PC에 연결하여 5개의 USB 연결을 지원하는 USB Host Driver를 설계하려 한다.



그림 – Note PC 제품의 USB Port

* + 1. **Stakeholder**

|  |  |
| --- | --- |
| Stakeholder | Interest |
| Note PC 제조사 | 부품사로부터 USB Host Device를 납품 받아 자사의 Note PC에 탑재하려 한다. 얇은 제품을 지향하기 때문에 납품 받는 제품이 최소한의 하드웨어로 충분한 성능을 내기를 기대한다. |
| Operation System  개발사 | 불특정 다수의 USB Device에 대해서 최대한 정보를 파악할 수 있길 바라며, 그에 따른 최적의 데이터 전송을 기대한다. |
| USB Device  개발사 | 개발한 장치가 타겟 운영체제에서 개발의도대로 동작하기를 기대한다. 그러기 위해 제품의 특성에 맞는 최대한의 리소스(전력, 대역폭 등)를 할당 받기를 기대한다. |
| LED Controller  개발사 | 다양한 제품에 범용적으로 사용될 수 있는 LED 제품을 개발하고 납품하고 싶어한다. |
| USB Device  사용자 | 가능한 많은 장치를 연결하여 사용하고 싶어한다. 합리적인 가격에 안정적인 속도와 충전 기능을 제공받고 싶어한다. |

* + 1. **시스템 제약사항**
* 제품은 5개의 USB 슬롯을 제공한다.
* 제품은 외부에 LED가 5개 존재하며 연결된 5개의 USB Device를 표현한다. 파란색 불빛을 낼 수 있으며 점멸할 수 있고 점멸 속도도 조절이 가능하다.
* 제품은 Host PC로부터 동작에 충분한 양의 전력을 공급받아 제품 자체의 전력은 고려사항이 아니다.
* USB Device Driver를 가진 운영체제가 있는 PC 환경을 Host로 가정한다.
* USB의 물리적 통신 프로토콜은 USB Host Controller가 담당한다.
* USB Host Controller는 CRC(Cyclic Redundancy Check) 및 재전송 등을 통해 전송 데이터에 대한 무결성을 보장한다고 가정한다.
* 127 개의 USB Device 연결이 가능하며, 물리적으로 30미터 이내에 연결되어 있다.
  1. **시스템 정의**

본 과제에서 개발하는 USB Hub System의 개요 및 개발 범위는 아래 그림 3에서 빨간색 점선으로 표시한 부분과 같다.

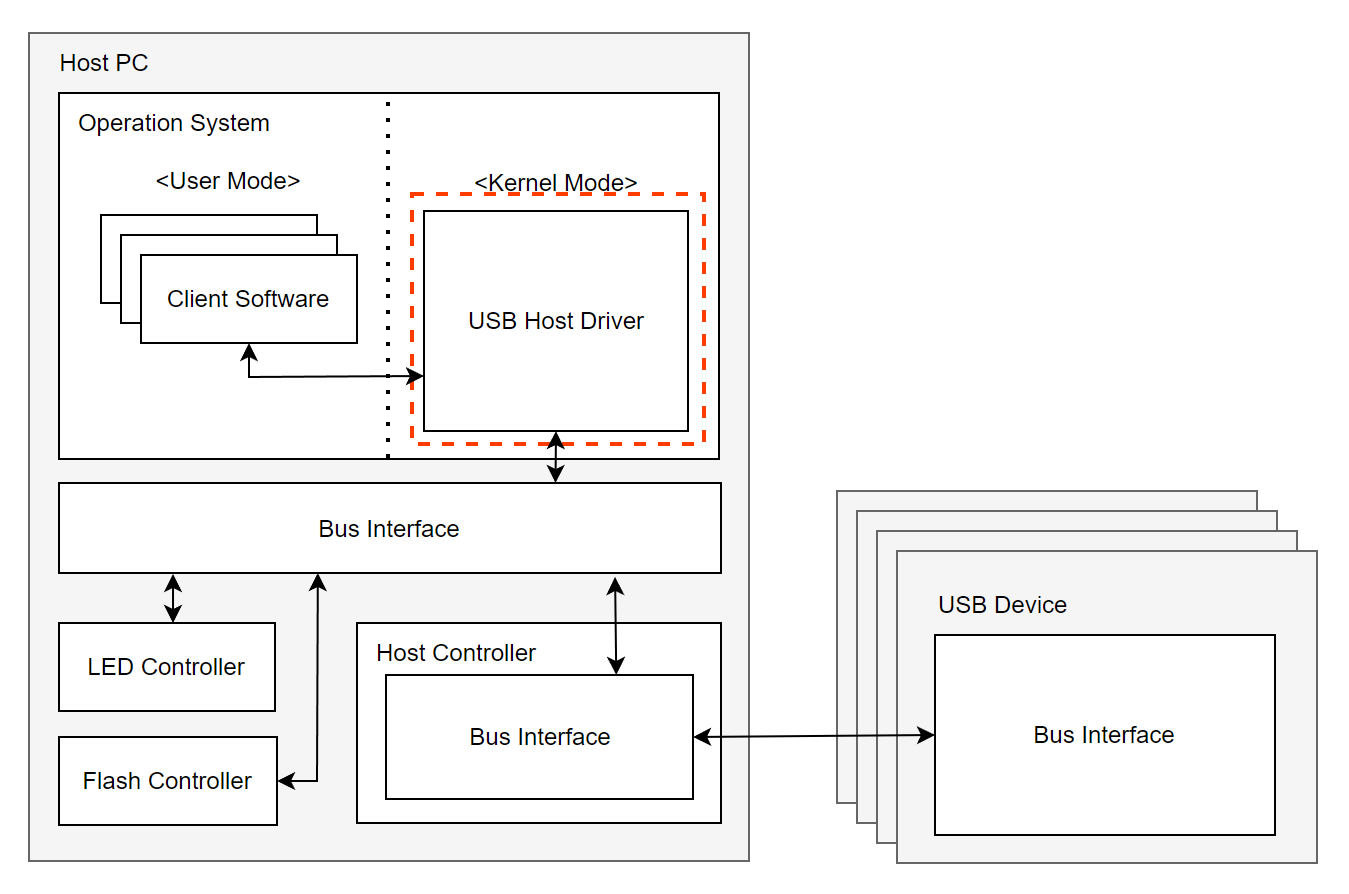


그림 3 - 시스템 전체 구조와 개발 범위

본 과제의 설계 범위는 USB Host Driver까지이다. Host PC 내에는 USB Host System과 LED, Flash 등의 하드웨어가 존재하며 그를 제어하기 위한 각각의 Controller들이 있다. 각각의 Controller들은 Bus Interface를 통해 Operation System과 물리적 통신을 할 수 있으며 소프트웨어 적인 처리는 Device Driver가 담당을 한다. Device Driver는 System Call을 통해 Client Software와 통신한다. 본 과제에서는 USB Host Driver가 Client Software와 System Call을 통해 통신하고 Host Controller에 USB Wire로 연결된 USB Device와 물리적으로 통신하며 USB Device를 관리하고 데이터를 전송하는 기능을 설계한다.

* + 1. **External System**

|  |  |
| --- | --- |
| External System | Interface |
| Operation System | 본 시스템은 Operation System의 System Call을 통해 Client Software와 통신한다. 또한 본 시스템을 통해 Operation System은 USB Device와 제어 정보를 전달하고 데이터를 주고받을 수 있다. |
| USB Device | USB Host Driver는 USB Device와 Host Controller를 통해 통신한다.  USB Device는 본 시스템을 통해 Client Software의 요청에 응답한다. |
| LED | LED 장치는 Third-party 형태로 USB Host System에 제공되는 장치이다. USB Host Driver는 Bus Interface를 통해 인터럽트로 제어 신호를 주고받는다. |

* + 1. **시스템 경계**

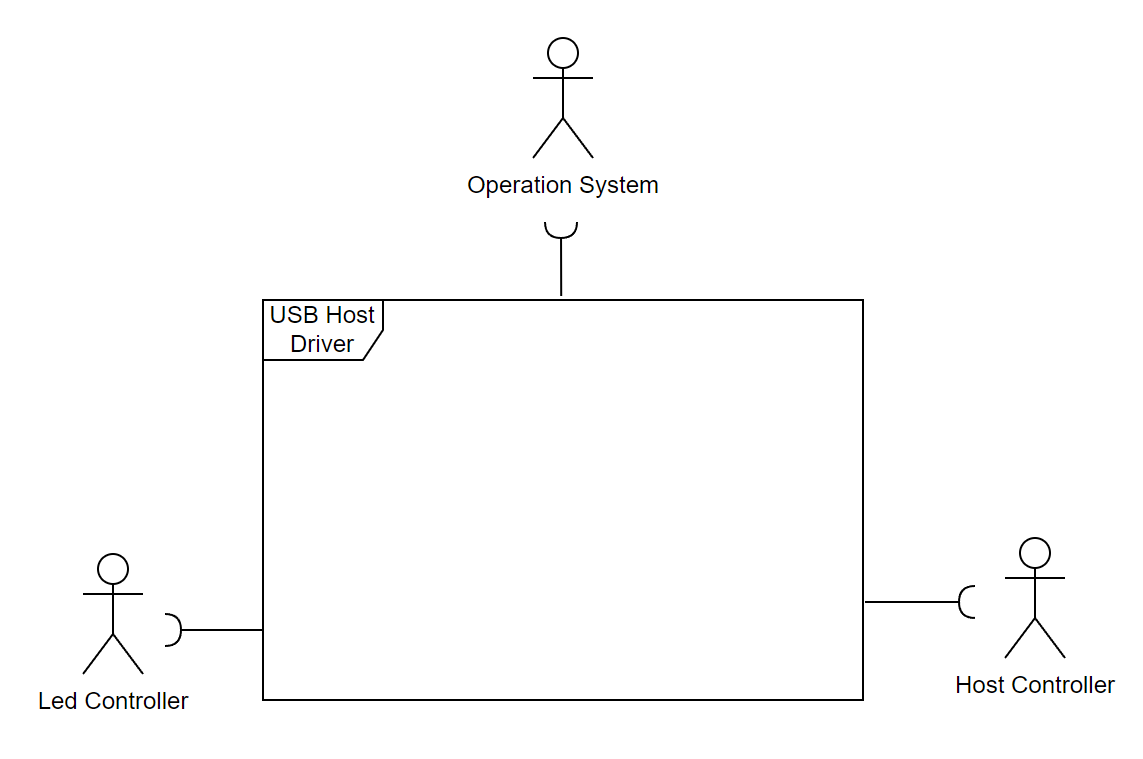


그림 4 - 시스템 경계

본 과제에서 다루고자 하는 USB Host Driver의 경계는 아래 그림 4와 같다. 본 시스템은 Operation System의 요청에 따라 Host Controller와 통신하며 USB Device에 대한 제어 및 데이터 전송을 수행할 수 있다. 또한 LED 하드웨어에 대한 제어를 Led Controller에게 요청할 수 있다.

* + 1. **시스템 동작**
* USB Host System을 시작하고 종료하는 절차를 수행한다.
* USB Device 연결 및 해제에 대한 처리를 수행한다.
* USB Device에 대한 정보를 수집하고 요청에 따라 전달한다.
* USB Device에 대한 읽기 및 쓰기를 수행한다.

# 요구사항

## 기능적 요구사항

* + 1. **Use Case Diagram**

시스템 경계에 따른 Use Case Diagram은 아래 그림 5와 같다.

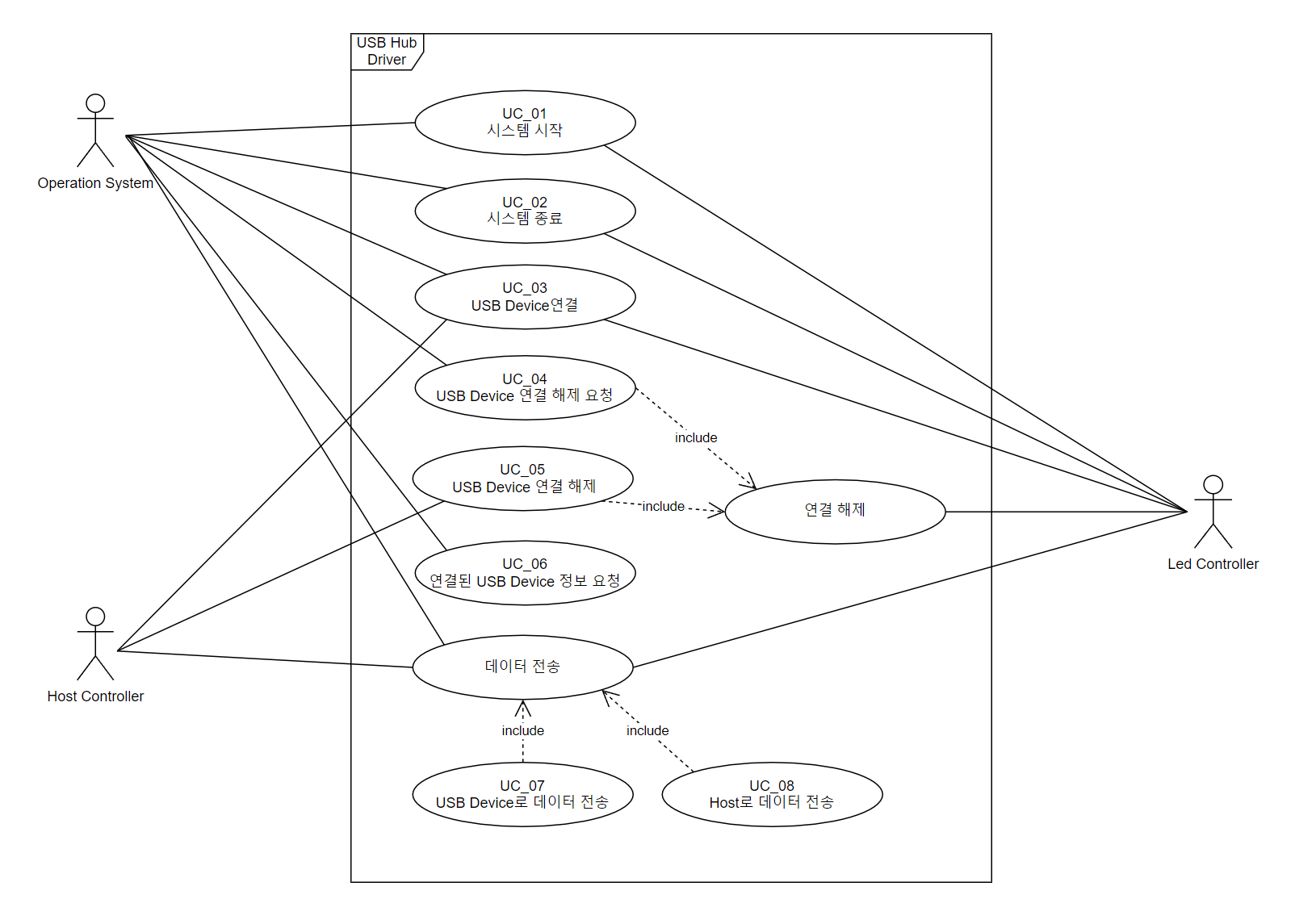


그림 5 – Use Case Diagram

* + 1. **Use Case 명세**
       1. **USB Hub 연결**

|  |  |
| --- | --- |
| **UC\_01** | USB Hub 연결 |
| 설명 | 시스템은 Host와 연결되면 시스템을 초기화한다. Host와 통신이 가능한 통로를 개설하고 Host, Memory 등으로부터 필요한 정보를 요청하고 LED를 점등한다. 또한 시스템은 가용 자원 정보 수집을 시작한다. |
| 행위자 | Host Interface |
| 선행조건 | USB Hub 장치가 물리적으로 Host PC의 USB 슬롯에 연결 |
| 후행조건 | 시스템은 ATTACH 상태 |
| 기본 동작 | 1. Host Interface는 시스템에게 통신할 수 있는 주소를 전달한다. 2. 시스템은 1번에서 받은 주소를 이용해 Message Pipe를 구성한다. 3. 시스템은 Message Pipe를 통해 수용 가능한 USB Class 목록을 요청한다. 4. Host Interface는 수용 가능한 USB Class 목록을 시스템에게 전달한다. 5. 시스템은 수용 가능한 USB Class를 저장한다. 6. 시스템은 Memory Interface에게 Known Device 목록을 요청한다. 7. Memory Interface는 시스템에게 Known Device 목록을 전달한다. 8. 시스템은 Known Device 목록을 저장한다. 9. 시스템은 LED Interface에게 3초간 점멸하도록 명령한다. 10. 시스템은 10ms마다 Device Interface에게 가용 자원 정보를 요청한다. 11. Device Interface는 요청이 올 때마다 시스템에게 가용 자원 정보를 전달한다. 12. 시스템은 가용 자원 정보를 저장한다. |
| 추가 동작 | - |

* + - 1. **USB Hub 연결 해제**

|  |  |
| --- | --- |
| **UC\_02** | USB Hub 연결 해제 |
| 설명 | 시스템은 Host와 연결이 해제되면 일정시간 재연결을 기다린다. 재연결이 되지 않으면 연결된 USB Device와 연결을 해제하고 LED 점등을 해제한다. 또한 시스템은 가용 자원 수집을 중단한다. |
| 행위자 | Host Interface |
| 선행조건 | 시스템은 ATTACH 상태  Host PC의 USB 슬롯에 연결된 USB Hub 장치가 물리적으로 연결 해제 |
| 후행조건 | 시스템은 DETACH 상태 |
| 기본 동작 | 1. Host Interface는 시스템에게 연결 해제를 통보한다. 2. 시스템은 5000ms동안 재연결을 기다린다. 3. 재연결이 되지 않으면 시스템은 모든 연결된 USB Device들에게 연결 해제를 통보하고 해당 USB Device와 Mapping된 Stream PIPE를 제거한다. 4. 시스템은 Message Pipe를 제거한다. 5. 시스템은 Led Interface에게 모든 LED 점등 해제를 명령한다. 6. 시스템은 가용 자원 정보 수집을 중단한다. |
| 추가 동작 | 1. (2에서) USB Hub가 Host PC의 USB 슬롯에 다시 연결되면 시스템에 저장된 정보를 유지하고 Use Case을 종료한다. |

* + - 1. **USB Device 연결**

|  |  |
| --- | --- |
| **UC\_03** | USB Device 연결 |
| 설명 | 시스템은 USB Device와 연결되면 USB Specification에 정의된 열거 절차를 진행한다. 시스템은 USB Device가 요구한 리소스에 대해서 수용가능한만큼의 리소스를 제공한다. 또한 USB Device가 연결된 위치의 LED를 점등하고 Host Interface에게 연결을 통보한다. |
| 행위자 | Device Interface |
| 선행조건 | 시스템은 ATTACH 상태  USB Hub의 USB 슬롯 혹은 Sub-Hub에 USB Device가 물리적으로 연결 |
| 후행조건 | - |
| 기본 동작 | 1. Device Interface는 시스템에게 USB Device 연결을 통보한다. 2. 시스템은 Device Interface에게 최소 전력(100mA)를 공급을 명령한다. 3. Device Interface는 시스템에게 전력 공급을 통보한다. 4. 시스템은 Device Interface에게 USB Device와 시스템간 통신할 수 있는 주소를 할당한다. 5. Device Interface는 시스템에게 VID, PID를 전달한다. 6. 시스템은 연결된 적 있는 장치인지 확인한다.   6-1. 연결된 적 없는 장치라면 Device Interface에게 USB Device descriptor를 요청한다.  6-2. Device Interface는 시스템에게 USB Device descriptor를 전달한다.  6-3. 시스템은 USB Device descriptor에서 USB Class를 확인하고 수용할 수 있는지 확인한다.   1. 시스템은 USB Device의 요구 리소스에 대한 가용 자원이 충분한지 확인한다. 충분하지 않다면 시스템은 요구 사항과 가용 자원에 대한 협상된 값을 도출한다. 2. 시스템은 Device Interface에게 7번에서 계산된 자원만큼의 전력을 공급하도록 명령한다. 3. 시스템은 LED Interface에게 USB Device가 연결된 위치의 LED를 점등하도록 명령한다. 4. 시스템은 Host와 USB Device가 통신할 수 있는 Stream Pipe를 개설한다. 5. 시스템은 USB Device가 연결된 적 없던 장치라면 Memory Interface에게 Known Device로 저장을 요청한다. 6. 시스템은 Host Interface에게 새로운 Stream Pipe가 개설되었음을 통보한다. |
| 추가 동작 | 1. (6-3에서) 수용할 수 없는 장치라면 Host Interface에게 수용할 수 없는 USB Device가 연결되었음을 통보한다. 2. (9에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 절차는 생략한다. |

* + - 1. **USB Device 연결 해제 요청**

|  |  |
| --- | --- |
| **UC\_04** | USB Device 연결 해제 요청 |
| 설명 | Host에서 특정 USB Device의 연결을 해제하는 절차이다. 물리적으로 USB 슬롯에서 USB가 해제되지는 않지만 논리적으로는 연결을 해제한다. Windows 운영체제의 “안전하게 USB 장치제거”에 해당하는 기능이다. |
| 행위자 | Host Interface |
| 선행조건 | 시스템은 ATTACH 상태  USB Device가 시스템에 연결된 상태 |
| 후행조건 | USB Device는 USB Hub의 슬롯에 물리적으로만 연결된 상태 |
| 기본 동작 | 1. Host Interface는 시스템에게 특정 USB Device 연결 해제 요청을 한다. 2. 시스템은 Device Interface에게 특정 USB Device 와의 연결 해제를 요청한다. 3. 시스템은 특정 USB Device와 Mapping된 Stream Pipe를 해제한다. 4. 시스템은 LED Interface에게 특정 USB Device와의 매핑 해제와 점등 해제를 명령한다. 5. 시스템은 특정 USB Device의 컨텍스트 정보를 삭제한다. |
| 추가 동작 | 1. (2에서) Graceful 연결 해제가 요청되는 경우, Device Interface로부터 연결 해제 통보를 받을 때까지 3을 진행하지 않는다. 2. (4에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 해제 절차는 생략한다. |

* + - 1. **USB Device 연결 해제**

|  |  |
| --- | --- |
| **UC\_05** | USB Device 연결 해제 |
| 설명 | 물리적으로 특정 USB Device가 USB 슬롯에서 분리되었을 때 발생하는 Use Case이다. Host에게 연결 해제를 통보하고 Stream Pipe 등 할당된 자원과 저장된 자료를 삭제한다. |
| 행위자 | Device Interface |
| 선행조건 | 시스템은 ATTACH 상태  USB Device가 시스템에 연결된 상태 |
| 후행조건 | - |
| 기본 동작 | 1. Device Interface가 시스템에게 특정 USB Device의 연결 해제를 통보한다. 2. 시스템은 Host Interface에게 특정 USB Device의 연결 해제를 통보한다. 3. Host Interface는 시스템의 Stream Pipe에 남은 데이터를 Read한 뒤, 시스템에게 디바이스 해제 요청을 한다. 4. 시스템은 특정 USB Device와 Mapping된 Stream Pipe를 해제한다. 5. 시스템은 LED Interface에게 특정 USB Device와의 매핑 해제와 점등 해제를 명령한다. 6. 시스템은 특정 USB Device의 컨텍스트 정보를 삭제한다. |
| 추가 동작 | 1. (5에서) USB Device가 루트 허브에 연결된 장치가 아니라면 LED 점등 해제 절차는 생략한다. |

* + - 1. **연결된 USB Device 정보 요청**

|  |  |
| --- | --- |
| **UC\_06** | 연결된 USB Device 정보 요청 |
| 설명 | Host는 연결된 USB Device에 대한 정보를 요청한다. 시스템은 USB Device와 연결될 때 생성한 컨텍스트를 이용하여 Device Interface를 거치지 않고 정보를 모은 뒤 Host Interface에게 전달한다. |
| 행위자 | Host Interface |
| 선행조건 | 시스템은 ATTACH 상태 |
| 후행조건 | - |
| 기본 동작 | 1. Host Interface는 시스템에게 연결된 USB Device들의 정보를 요청한다. 2. 시스템은 모든 연결된 USB Device 정보를 수집한다. 3. 시스템은 Host Interface에게 수집한 정보를 전달한다. |
| 추가 동작 | 1. (2에서) 연결된 USB Device가 없는 경우 바로 3번으로 넘어간다. |

* + - 1. **Host로 데이터 전송**

|  |  |
| --- | --- |
| **UC\_07** | Host로 데이터 전송 |
| 설명 | 특정 USB Device로부터 데이터를 읽는 Use Case이다. |
| 행위자 | Host Interface |
| 선행조건 | 시스템은 ATTACH 상태  한 개 이상의 USB Device가 시스템에 연결된 상태 |
| 후행조건 | - |
| 기본 동작 | 1. Host Interface가 시스템에게 특정 USB Device의 데이터 읽기를 요청한다. 2. 시스템은 내부 USB Device에 대한 상태를 “READING”으로 변경한다. 3. 시스템은 Device Interface에게 특정 USB Device의 데이터 읽기를 요청한다. 4. Device Interface는 요청한 데이터가 남아있는 한 시스템에게 데이터 쓰기를 수행한다.    1. Device Interface는 시스템에게 데이터 쓰기를 수행한다.    2. 시스템은 Host Interface에게 읽을 데이터가 있음을 통보한다.    3. 시스템은 LED Interface에게 데이터 레이트를 전달해서 값에 맞추어 점멸할 수 있도록 명령한다. 이 때 USB Device가 Sub-Hub에 연결된 상태라면 USB Hub에 연결된 조상 장치에 해당하는 LED를 점멸한다. 5. 시스템은 LED Interface에게 점멸 해제를 명령한다. 6. 시스템은 내부 USB Device에 대한 상태를 “SUSPEND”로 변경한다. |
| 추가 동작 | - |

* + - 1. **USB Device로 데이터 전송**

|  |  |
| --- | --- |
| **UC\_08** | USB Device로 데이터 전송 |
| 설명 | 특정 USB Device에게 데이터를 쓰는 Use Case이다. |
| 행위자 | Host Interface |
| 선행조건 | 시스템은 ATTACH 상태  한 개 이상의 USB Device가 시스템에 연결된 상태 |
| 후행조건 | - |
| 기본 동작 | 1. Host Interface는 시스템에게 특정 USB Device에 대한 쓰기를 통보한다. 2. 시스템은 내부 USB Device에 대한 상태를 “WRITING”으로 변경한다. 3. Host Interface는 요청한 데이터가 남아있는 한 시스템에게 데이터 쓰기를 수행한다.   3-1. Host Interface는 시스템에게 데이터 쓰기를 수행한다.  3-2. 시스템은 Device Interface에게 읽을 데이터가 있음을 통보한다.  3-3. 시스템은 LED Interface에게 데이터 레이트를 전달해서 값에 맞추어 점멸할 수 있도록 명령한다. 이 때 USB Device가 Sub-Hub에 연결된 상태라면 USB Hub에 연결된 조상 장치에 해당하는 LED를 점멸한다.   1. 시스템은 LED Interface에게 점멸 해제를 명령한다. 2. 시스템은 내부 USB Device에 대한 상태를 “SUSPEND”으로 변경한다. |
| 추가 동작 | - |

## 비기능적 요구사항

**2.2.1 NFR\_01 지원 가능한 USB 버전**

|  |  |  |
| --- | --- | --- |
| **NFR\_01**  **(QS\_07)** | **확장성** | 지원 가능한 USB 버전 |
| 설명 | 본 시스템을 지원하는 USB 2.0에 해당하는 USB Device가 아니더라도 최소 조건을 충족하면 지원이 가능해야 한다. | |
| 환경 | 시스템이 정상 동작한다. | |
| 자극 | USB Device를 USB Hub의 슬롯에 삽입한다. | |
| 반응 | UC\_03 USB Device 연결 | |
| 측정 | [지원 가능한 USB 버전] = Min(2, 연결된 USB Device 버전) | |
| 제약 | [지원 가능한 USB 버전] >= 2 | |

**2.2.1 NFR\_02 Low-Speed를 사용하는 USB Class 장치의 전송 속도**

|  |  |  |
| --- | --- | --- |
| **NFR\_02**  **(QS\_02)** | **성능** | Low-Speed를 사용하는 USB Class 장치의 전송 속도 |
| 설명 | Low-Speed를 사용하는 USB Class의 경우 고속으로 데이터 전송시 오히려 데이터 에러를 발생시킬 수 있다. 허용된 수치 이하의 속도로 전송되어야 한다. | |
| 환경 | 시스템이 정상 동작한다. | |
| 자극 | USB Device로부터 데이터를 전송한다. | |
| 반응 | UC\_07 Host로 데이터 전송 | |
| 측정 | [전송 속도] = [Host Controller HW를 통해 측정되는 속도] | |
| 제약 | [전송 속도] <= 1.5Mbps | |

## 품질 속성

**2.3.1 QA\_01 High-Speed를 사용하는 USB Class 장치의 전송 속도**

|  |  |  |
| --- | --- | --- |
| **QA\_01**  **(QS\_04)** | **성능** | High-Speed를 사용하는 USB Class 장치의 전송 속도 |
| 설명 | 고속 전송이 필요한 USB Class 장치의 경우 데이터 전송이 빠를수록 좋다. | |
| 환경 | High-Speed를 사용하는 USB Class 장치가 USB Hub의 슬롯에 삽입되어 있다. | |
| 자극 | USB Device로부터 데이터를 전송한다. | |
| 반응 | UC\_07 Host로 데이터 전송 | |
| 측정 | [전송 속도] = [Host Controller HW를 통해 측정되는 속도] | |

**2.3.2 QA\_02 등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성**

|  |  |  |
| --- | --- | --- |
| **QA\_02**  **(QS\_05)** | **성능** | 등시성이 요구되는 USB Class 장치의 전송 속도 일관성 |
| 설명 | 등시성이 요구되는 USB Class 장치의 경우 단위시간동안 전송되는 데이터의 양이 일정할수록 좋은 품질을 나타낸다. | |
| 환경 | Host가 시스템에게 등시성이 요구되는 데이터 읽기를 요구한다. | |
| 자극 | USB Device로부터 데이터를 전송한다. | |
| 반응 | UC\_07 Host로 데이터 전송 | |
| 측정 | [전송 속도의 표준편차] = [단위 시간동안 Host Controller HW를 통해 측정되는 속도의 표준 편차] | |

**2.3.3 QA\_03 지원 가능한 USB Class의 수**

|  |  |  |
| --- | --- | --- |
| **QA\_03**  **(QS\_10)** | **변경용이성** | 지원 가능한 USB Class의 수 |
| 설명 | 지원 가능한 USB Class를 추가하는데 비용이 적어야 한다. | |
| 환경 | 아직 USB Hub의 연결이 이루어지지 않았다. | |
| 자극 | USB Hub 장치를 Host PC의 USB 슬롯에 삽입한다. | |
| 반응 | UC\_01 USB Hub 연결 | |
| 측정 | [변경 모듈 수] = [USB Class 하나를 추가하는데 변경이 필요한 전체 모듈의 수] | |

**2.3.4 QA\_04 High-Speed를 사용하는 USB Class 장치의 전송 속도**

|  |  |  |
| --- | --- | --- |
| **QA\_04** | **안정성** | QS\_08. USB Device가 요구한 전력 공급 비율 |
| 설명 | 최대한 USB Device가 요구하는 만큼의 전력을 공급할 수 있어야 한다. | |
| 환경 | 시스템이 정상 동작한다. | |
| 자극 | USB Device를 USB Hub의 슬롯에 삽입한다. | |
| 반응 | UC\_03 USB Device 연결 | |
| 측정 | [USB Device가 요구한 자원 할당 비율] = [Resource Negotiator에서 요구한만큼 할당하는 경우] / [USB Device가 연결된 횟수] | |

**2.3.5 QA\_05 High-Speed를 사용하는 USB Class 장치의 전송 속도**

|  |  |  |
| --- | --- | --- |
| **QA\_05** | **성능** | QS\_01. 시스템 부팅시간 |
| 설명 | 시스템의 부팅시간은 짧을수록 좋다. | |
| 환경 | 아직 USB Hub의 연결이 이루어지지 않았다. | |
| 자극 | USB Hub 장치를 Host PC의 USB 슬롯에 삽입한다. | |
| 반응 | UC\_01 USB Hub 연결 | |
| 측정 | [부팅 시간] = [LED 장치의 최초 점멸 시각] – [시스템이 Host PC의 USB 슬롯에 삽입되는 시각] | |

**2.3.6 QA\_06 High-Speed를 사용하는 USB Class 장치의 전송 속도**

|  |  |  |
| --- | --- | --- |
| **QA\_06** | **성능** | QS\_11. 연결 이력이 있는 USB Device의 연결 속도 |
| 설명 | 연결 이력이 있는 USB Device를 연결할 때에는 처음 연결할 때보다 더 빠르면 좋다. | |
| 환경 | 시스템이 정상 동작한다. | |
| 자극 | USB Device를 USB Hub의 슬롯에 삽입한다. | |
| 반응 | UC\_03 USB Device 연결 | |
| 측정 | [연결 이력이 있는 USB Device의 연결 속도] = [Host Interface로 새로운 USB Device 연결이 통지되는 시각] - [연결 이력이 있는 USB Device의 USB 슬롯 연결 시각] | |

부록

[A. 도메인 모델 9](#_Toc479150206)

[B. 품질 시나리오 10](#_Toc479150207)

[C. 품질 시나리오 분석 11](#_Toc479150208)

1. 도메인 모델

본 과제에서 USB Hub Driver에 대한 도메인 모델은 그림 6과 같다. 본 도메인 모델은 Boundary/Control/Entity 모델로 작성되었다. Boundary는 외부의 Interface와 상호 작용하고, Control은 Boundary와 Entity를 연결해주며, Entity는 시스템에 필요한 데이터를 저장한다.

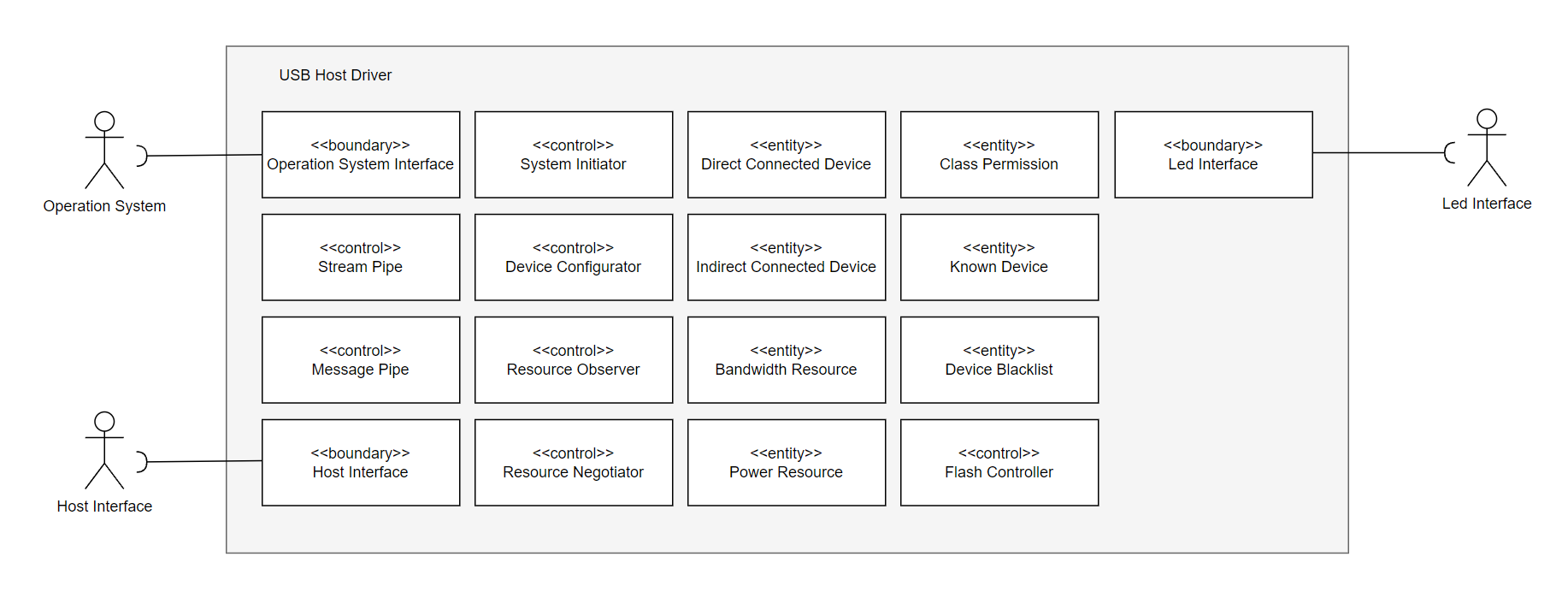


그림 - Domain Model

**A1. Domain Model List**

|  |  |  |
| --- | --- | --- |
| Domain Model | Category | Description |
| Operation System Interface | Boundary | Operation System으로부터 내려오는 System Call을 받고 해석하고 전파하는 역할을 담당하는 도메인 모델이다. |
| Stream Pipe | Control y | Host 시스템과 USB Device 간의 데이터 전송을 담당하는 도메인 모델이다. USB Specification에서 Host의 Client SW와 USB Device의 Function 간의 Endpoint에 해당하는 역할이다. |
| Message Pipe | Control | Host 시스템과 USB Device 간의 제어 신호를 전송하는 도메인 모델이다. USB Specification에서 Endpoint Zero에 해당하는 역할이다. |
| Led Interface | Boundary | LED Controller와 통신하는 LED Interface에 대한 Boundary 도메인 모델이다. LED Interface와 인터럽트를 주고받으며 명령을 해석하거나 명령으로 변환하는 역할을 한다. |
| Host Interface | Boundary | USB Device와 물리적인 통신을 담당하는 Host Controller와의 인터페이스이다. Host Controller 장치에 대한 추상화가 이루어진다. |
| Flash Controller | Control | Host 시스템 내부의 Flash로부터 데이터를 쓰고 읽는 역할을 담당하는 도메인 모델이다. |
| System Initiator | Control | 시스템의 시작, 종료 절차를 관리하는 도메인 모델이다. |
| Device Configurator | Control | USB Device에 대한 열거 절차를 담당하는 도메인 모델이다. Host와 USB Device가 서로 인지할 수 있도록 Configuration 절차를 주도하며 Device Entity를 생성하고 Pipe를 개설한다. |
| Resource Observer | Control | Host PC와 Host Controller의 상태를 체크하여 가용 가능한 자원에 대한 정보를 주기적으로 얻어와 저장하는 역할을 하는 도메인 모델이다. |
| Resource Negotiator | Control | 가용 가능한 자원을 조회하고 USB Device에서 요청하는 자원에 대한 가용 범위를 결정하는 도메인 모델이다. |
| Bandwidth Resource | Entity | 가용 가능한 대역폭 자원에 대한 정보를 저장하는 도메인 모델이다. |
| Power Resource | Entity | 가용 가능한 전력 자원에 대한 정보를 저장하는 도메인 모델이다. |
| Direct Connected Device | Entity | USB 슬롯에 직접 연결된 장치를 추상화한 도메인 모델이다. 직접 연결된 장치는 LED 장치와 Mapping되므로 플로우가 달라져 간접 연결된 장치와 구분하였다. 본 시스템은 직접적으로 USB Device와 연결되지 않기 때문에 USB Device에 대한 정보를 저장하는 역할을 한다. |
| Indirect Connected Device | Entity | USB 슬롯에 연결된 USB Hub를 통해 연결된 장치를 추상화한 도메인 모델이다. 본 시스템은 직접적으로 USB Device와 연결되지 않기 때문에 USB Device에 대한 정보를 저장하는 역할을 한다. |
| Class Permission | Entity | 허용가능한 USB Class를 저장하는 도메인 모델이다. Host 시스템에게 질의하여 수용가능한 USB Class를 얻어와 해당 Entity에 저장한다. |
| Known Device | Entity | 연결된 적 있는 USB Device에 대한 식별자를 저장하는 도메인 모델이다. USB Device의 VID, PID를 조합하여 USB Device의 고유한 값을 만들어 저장한다. |
| Device Blacklist | Entity | 연결 거부된 USB Device의 식별자를 저장하여 향후 연결을 거부하는데 쓰이는 도메인 모델이다. |

**A2. Sequence Diagram**

**A2.1. 시스템 시작**

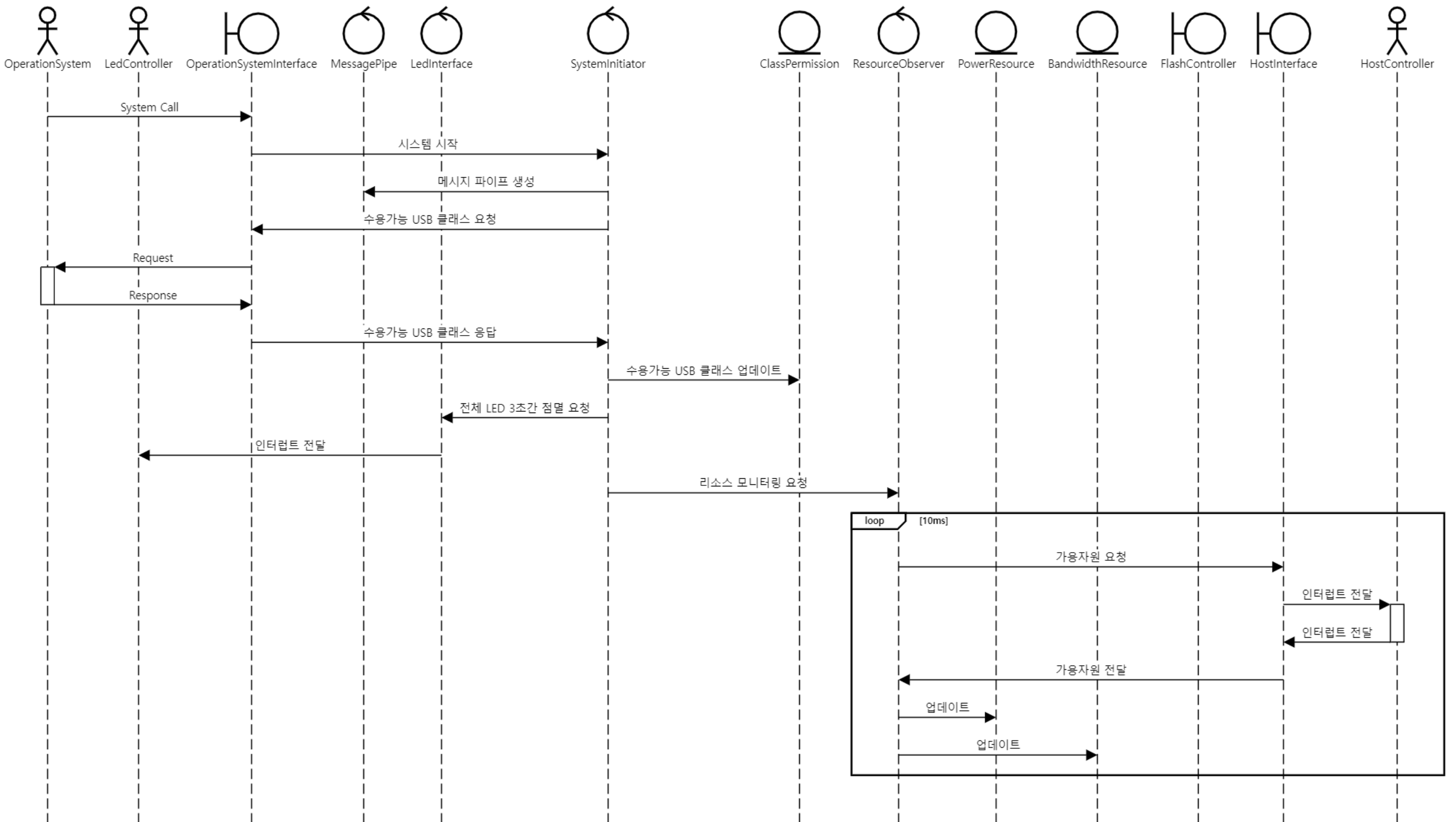


그림 – 시스템 시작

**A2.2. 시스템 종료**

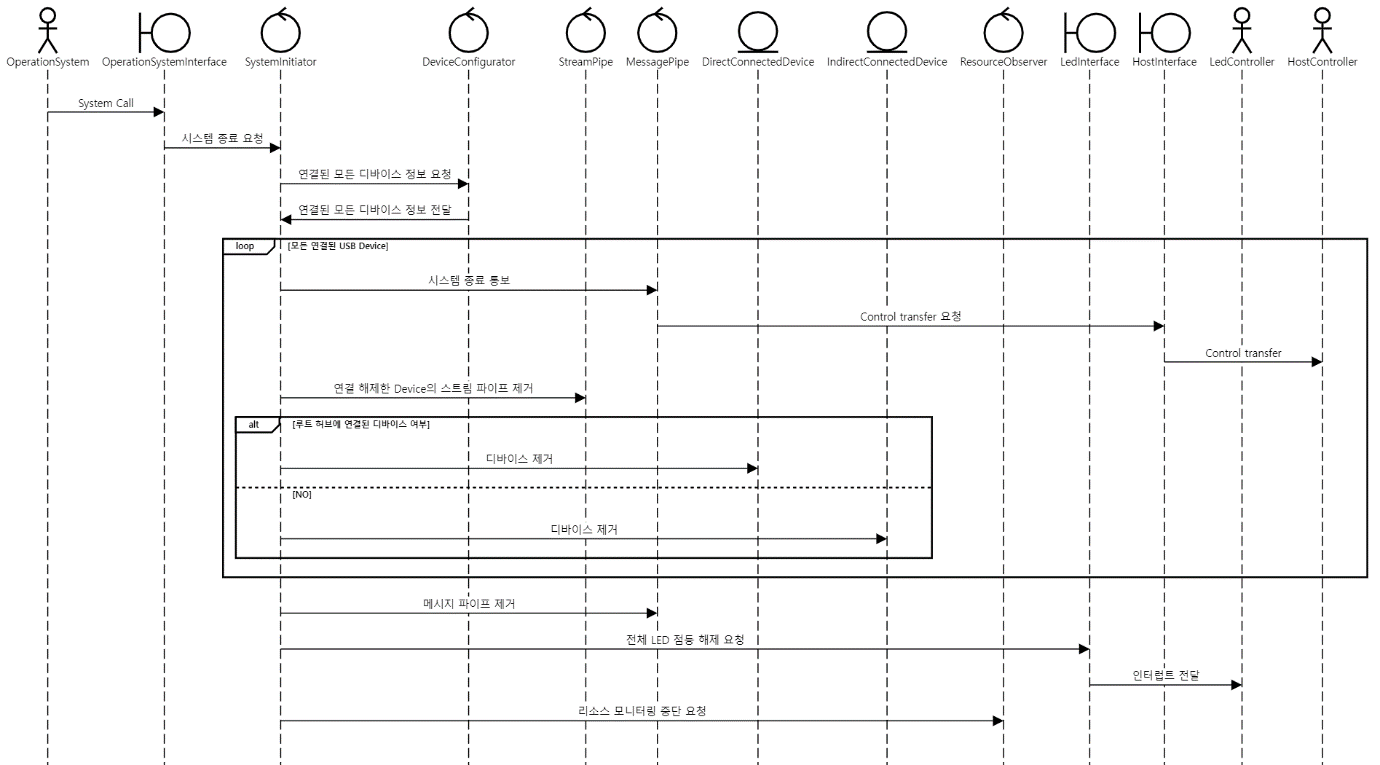
****

그림 – 시스템 종료

**A2.3. USB Device 연결**

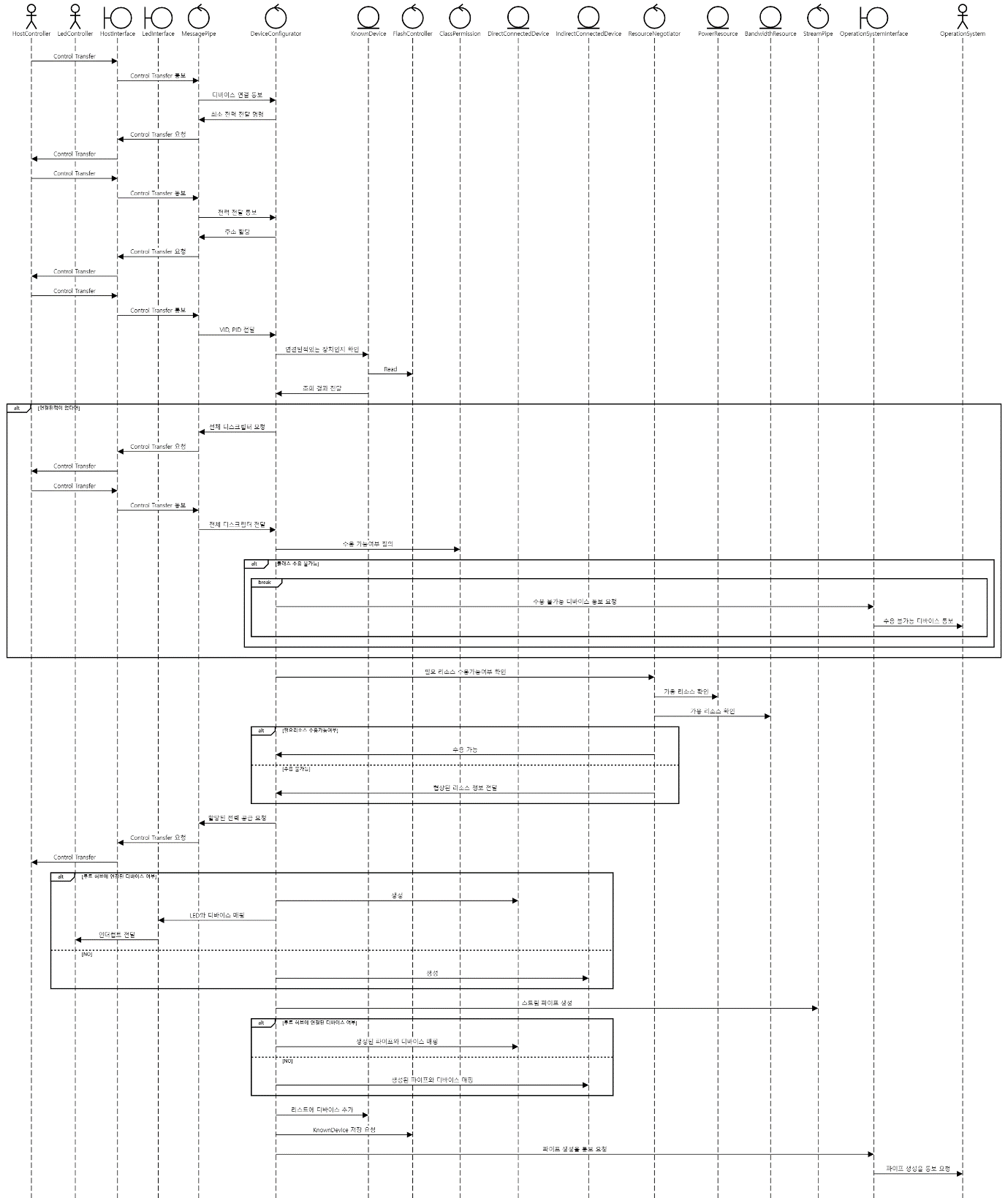
****

그림 - USB Device 연결

**A2.4. USB Device 연결 해제 요청**

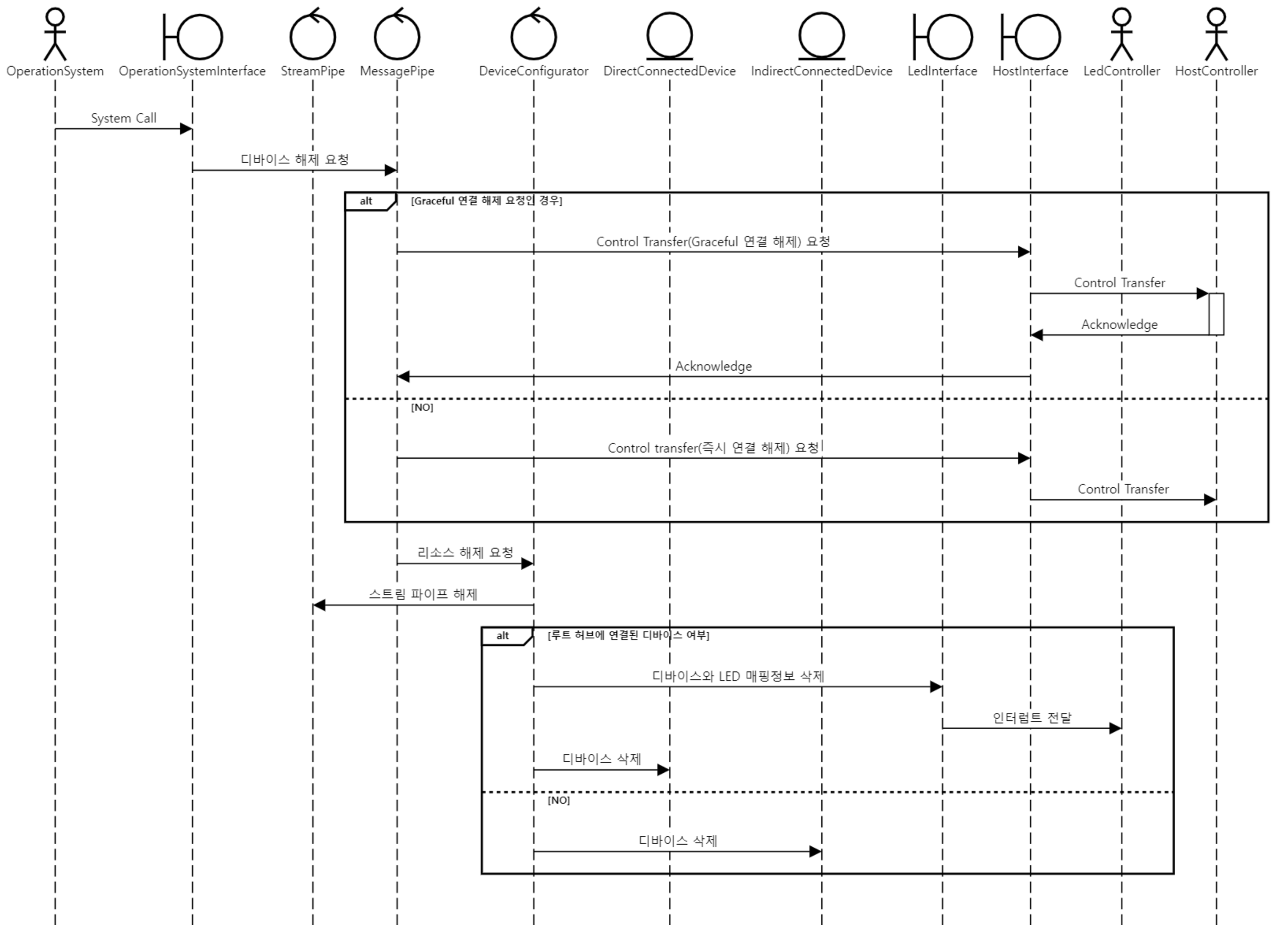


그림 - USB Device 연결 해제 요청

**A2.5. USB Device 연결 해제**

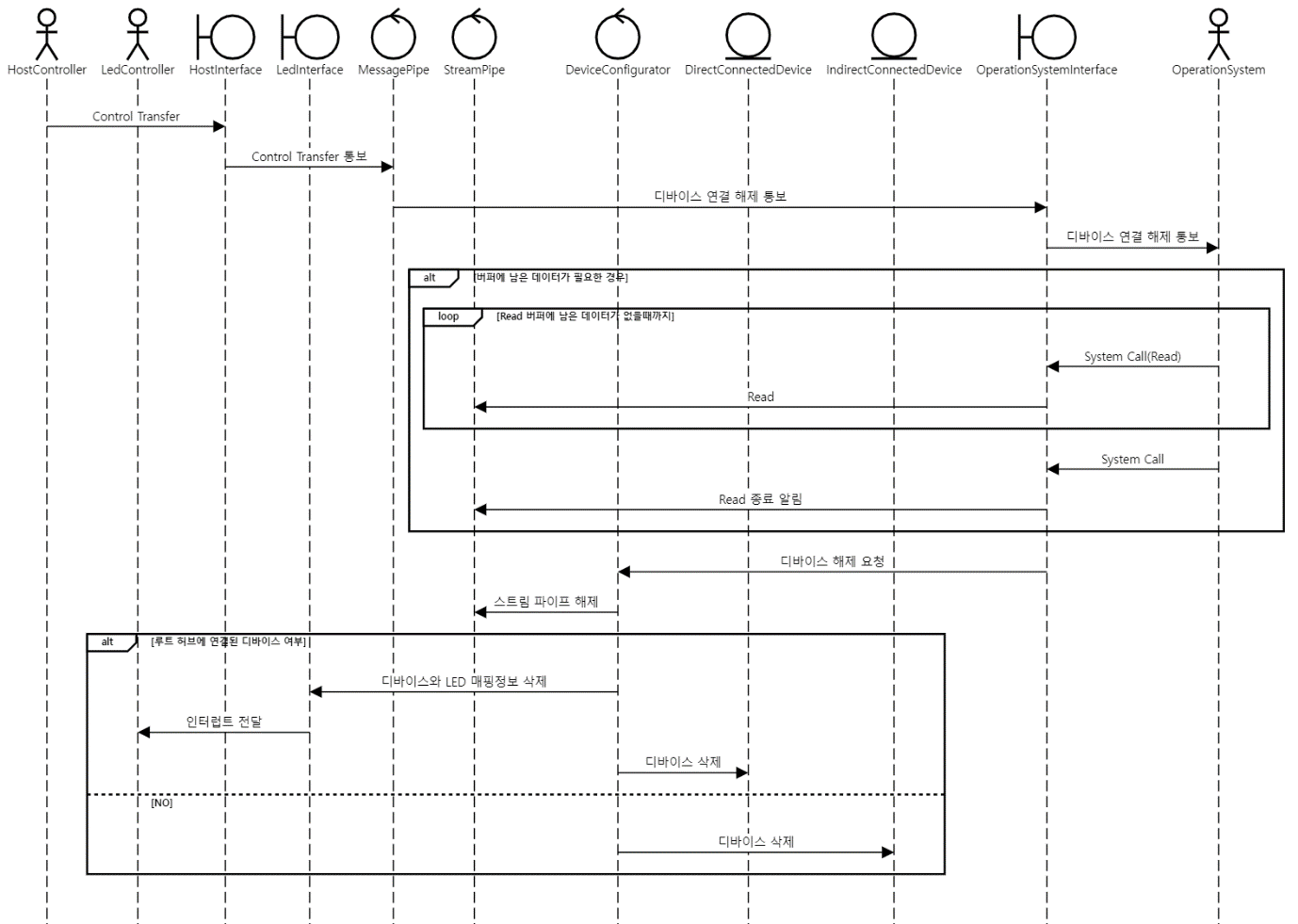


그림 - USB Device 연결 해제

**A2.6. 연결된 USB Device 정보 요청**

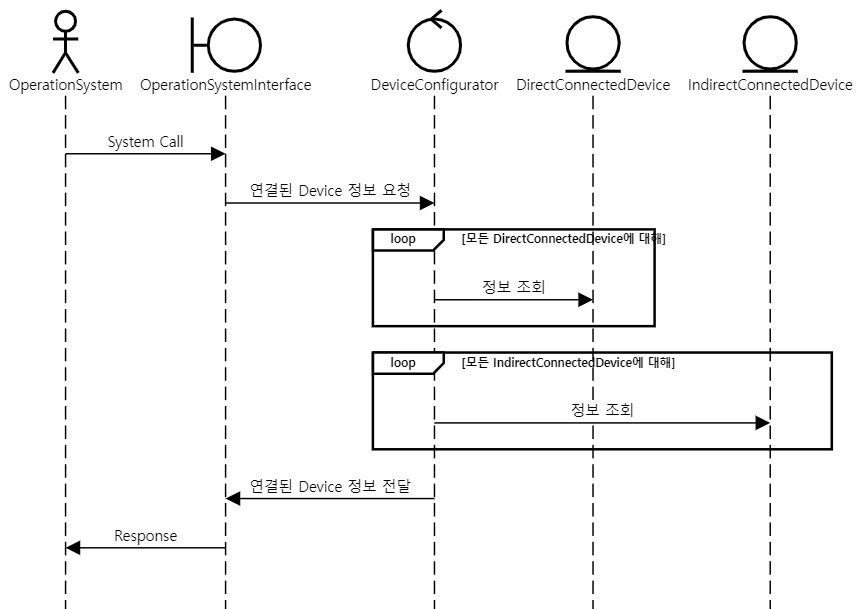


그림 - 연결된 USB Device 정보 요청

**A2.7. Host로 데이터 전송**

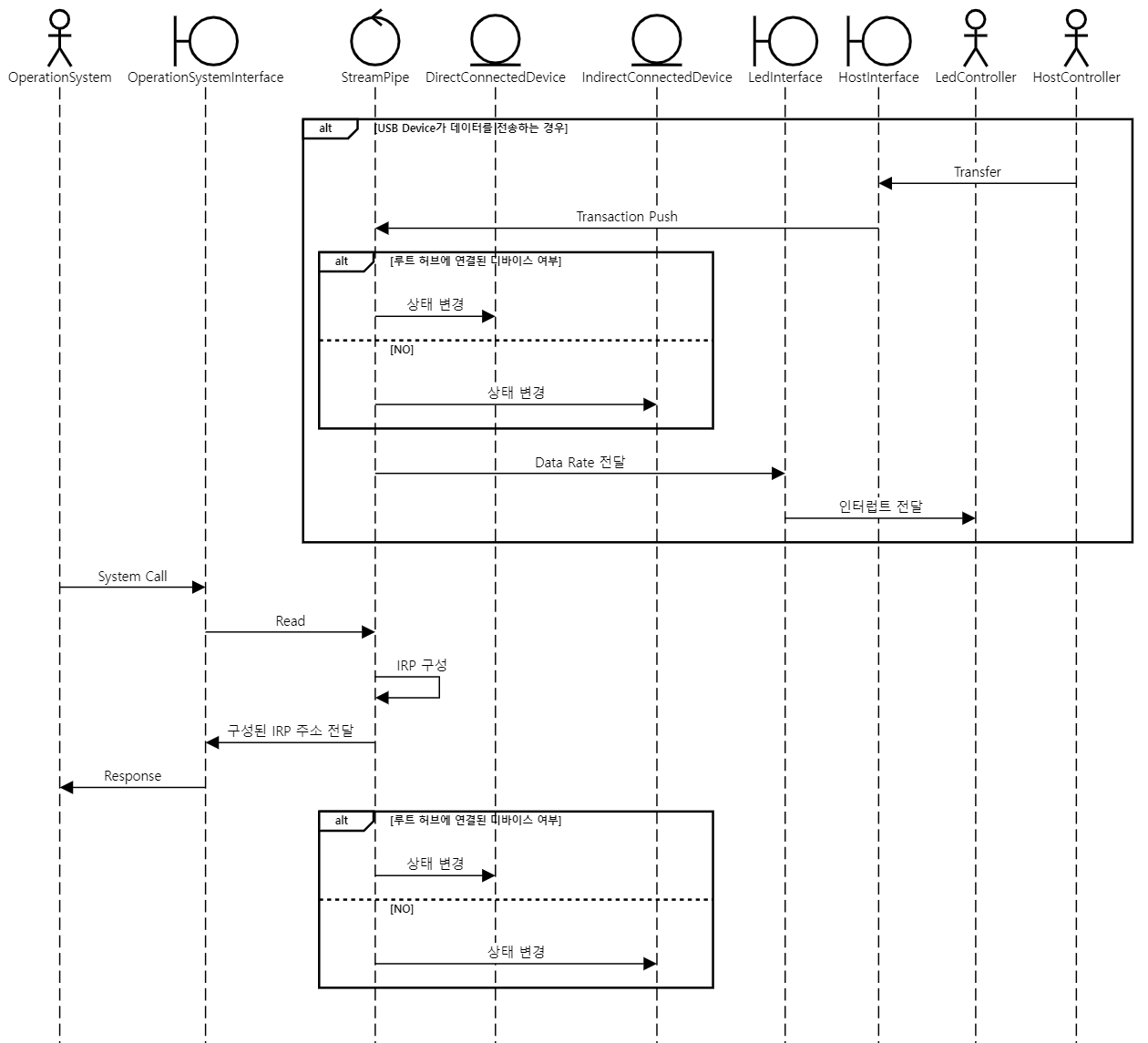
****

그림 - Host로 데이터 전송

**A2.7. USB Device로 데이터 전송**

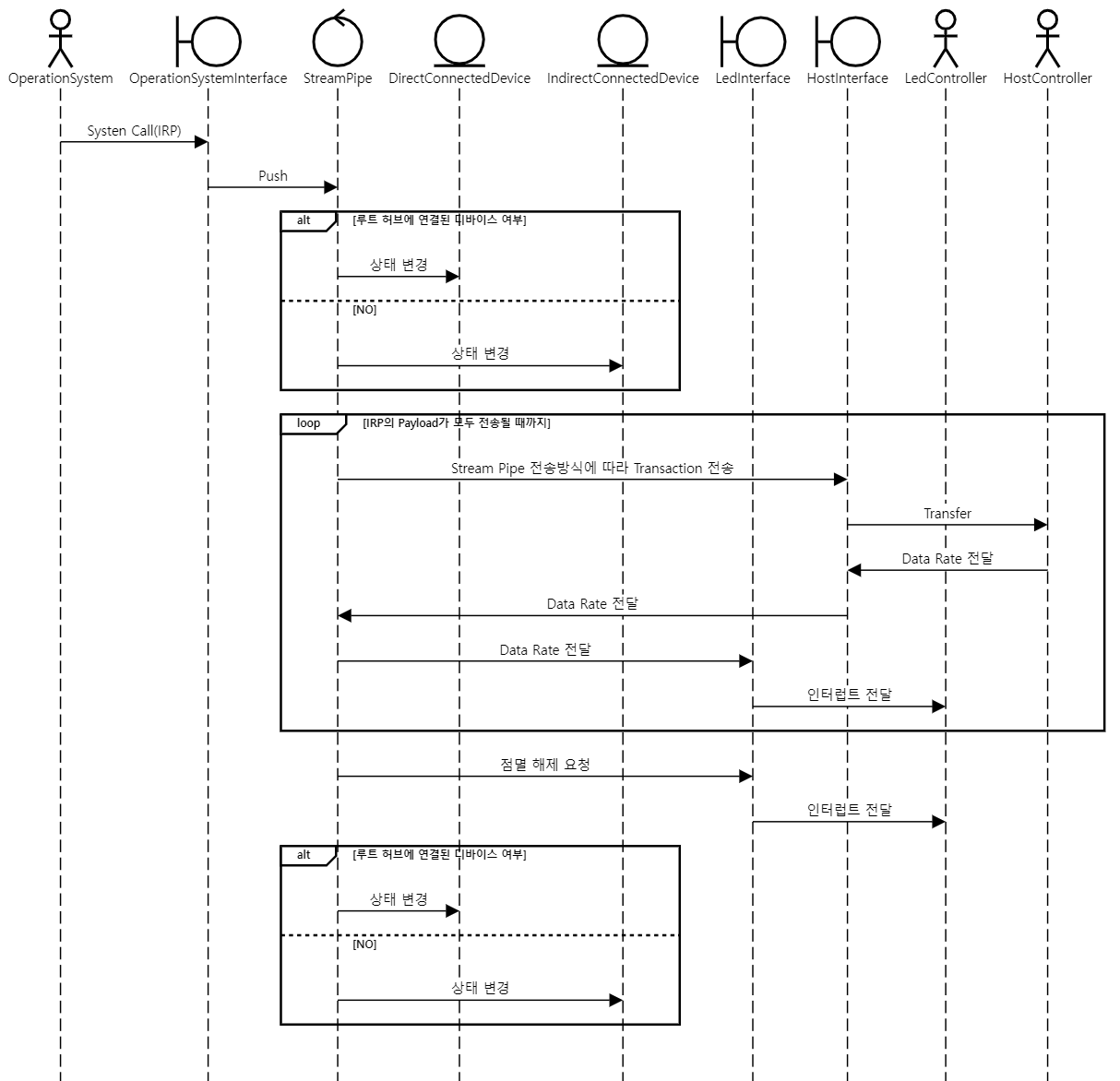
****

그림 - USB Device로 데이터 전송

1. 품질 시나리오

|  |  |  |
| --- | --- | --- |
| QS\_01 | 성능 | 시스템 부팅시간 |
| 측정 방법 | [부팅 시간] = [LED 장치의 최초 점멸 시각] – [시스템이 Host PC의 USB 슬롯에 삽입되는 시각]  [부팅 시간]은 짧을수록 좋다. |
| QS\_02 | 기능 정확성 | Low-Speed를 사용하는 USB Class 장치의 전송 속도 |
| 측정 방법 | [전송 속도] = [Host Controller HW를 통해 측정되는 속도]  [전송 속도] <= 1.5Mbps |
| QS\_03 | 성능 | Full-Speed를 사용하는 USB Class 장치의 전송 속도 |
| 측정방법 | [전송 속도] = [Host Controller HW를 통해 측정되는 속도]  [전송 속도]는 빠를수록 좋다. |
| QS\_04 | 성능 | High-Speed를 사용하는 USB Class 장치의 전송 속도 |
| 측정방법 | [전송 속도] = [Host Controller HW를 통해 측정되는 속도]  [전송 속도]는 빠를수록 좋다. |
| QS\_05 | 성능 | 등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성 |
| 측정방법 | [전송 속도의 표준편차] = [단위 시간동안 Host Controller HW를 통해 측정되는 속도의 표준 편차]  [전송 속도의 표준편차]가 작을수록 좋다. |
| QS\_06 | 안정성 | USB Hub Device의 CPU 점유율 표준편차 |
| 측정방법 | [CPU 점유율 표준편차] = [Host Controller HW를 통해 측정되는 CPU 점유율의 표준편차]  [CPU 점유율 표준편차]는 낮을수록 좋다 |
| QS\_07 | 확장성 | 지원 가능한 USB 버전 |
| 측정방법 | [지원 가능한 USB 버전] = Min(2, 연결된 USB Device 버전)  [지원 가능한 USB 버전] >= 2 |
| QS\_08 | 안정성 | USB Device가 요구한 전력 공급 비율 |
| 측정방법 | [USB Device가 요구한 자원 할당 비율] = [Resource Negotiator에서 요구한만큼 할당하는 경우] / [USB Device가 연결된 횟수]  [USB Device가 요구한 자원 할당 비율]가 높을수록 좋다. |
| QS\_09 | 자원활용성 | USB Hub 자체의 전력 사용량 |
| 측정방법 | [USB Hub 자체의 전력 사용량] = [전체 전력 소비량] – [연결된 USB Device에 할당된 전력량의 총합]  [USB Hub 자체의 전력 사용량]이 작을수록 좋다. |
| QS\_10 | 변경 용이성 | 지원 가능한 USB Class의 수 |
| 측정방법 | [변경 모듈 수] = [USB Class 하나를 추가하는데 변경이 필요한 전체 모듈의 수]  [변경 모듈 수]는 적을수록 좋다. |
| QS\_11 | 성능 | 연결 이력이 있는 USB Device의 연결 속도 |
| 측정방법 | [연결 이력이 있는 USB Device의 연결 속도] = [Host Interface로 새로운 USB Device 연결이 통지되는 시각] - [연결 이력이 있는 USB Device의 USB 슬롯 연결 시각]  [연결 이력이 있는 USB Device의 연결 속도]은 짧을수록 좋다. |

1. 품질 시나리오 분석

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 품질 시나리오 | | 중요도 | 복잡도 | 선정결과 |
| 성능 | QS\_04. High-Speed를 사용하는 USB Class 장치의 전송 속도 | H | H | QA\_01 |
| QS\_05. 등시성(Isochronism)이 요구되는 USB Class 장치의 전송 속도 일관성 | H | H | QA\_02 |
| QS\_02. Low-Speed를 사용하는 USB Class 장치의 전송 속도 | H | L | NFR\_02 |
| QS\_03. Full-Speed를 사용하는 USB Class 장치의 전송 속도 | L | L |  |
| QS\_01. 시스템 부팅시간 | M | M | QA\_05 |
| QS\_11. 연결 이력이 있는 USB Device의 연결 속도 | L | H | QA\_06 |
| 안정성 | QS\_08. USB Device가 요구한 전력 공급 비율 | H | L | QA\_04 |
| QS\_06. USB Hub Device의 CPU 점유율 표준편차 | M | H |  |
| 변경용이성 | QS\_10. 지원 가능한 USB Class의 수 | H | M | QA\_03 |
| 확장성 | QS\_07. 지원 가능한 USB 버전 | H | M | NFR\_01 |
| 자원활용성 | QS\_09. USB Hub 자체의 전력 사용량 | L | H |  |

1. 후보 구조

// 활동6. 후보 구조 설계

// 점검6-1. 품질에 대한 분석과 후보 구조가 적절한가?

// 점검6-2. 성능에 대한 분석과 후보 구조가 적절한가?

// 점검6-3. 변경용이성/확장성에 대한 분석과 후보 구조가 적절한가?

나중에 스스드 문서처럼 카테고리

* 품질이나 논펑션 => 문제 => 후보구조들 이런식으로

소프트웨어 클라이언트는 일반적으로 I/O 요청 패킷(IRP)을 통해 파이프로 데이터 전송을 요청한 다음 대기하거나 완료되면 알림을 받습니다. IRP에 대한 세부 정보는 운영 체제별 방식으로 정의됩니다. 이 명세는 (호스트에서) 자신과 장치의 끝점 사이에서 적절한 방향으로 데이터를 이동하기 위한 소프트웨어 클라이언트의 식별 가능한 요청을 단순히 참조하기 위해 이 용어를 사용합니다. 소프트웨어 클라이언트는 원하는 경우 파이프가 미해결 IRP를 모두 반환하도록 할 수 있습니다. 소프트웨어 클라이언트가 알림을 받습니다.

IRP와 관련된 버스 트랜잭션이 성공적으로 또는 오류로 인해 완료되었을 때 IRP가 완료되었음을 나타냅니다.

파이프에 대해 보류 중이거나 진행 중인 IRP가 없는 경우 파이프는 유휴 상태이며 호스트 컨트롤러는 파이프와 관련하여 아무 조치도 취하지 않습니다. 즉, 그러한 파이프의 끝점은 파이프로 향하는 버스 트랜잭션을 볼 수 없습니다. 파이프에 대한 버스 활동이 있는 유일한 시간은 해당 파이프에 대해 IRP가 보류 중인 경우입니다.

비등시성 파이프가 호스트에 STALL을 보내도록 하는 조건을 만나거나(8장 참조) IRP의 패킷에서 세 개의 버스 오류가 발생하면 IRP가 중단/중단되고 모든

미해결 IRP도 폐기되고 소프트웨어 클라이언트가 (구현에 따라) 상태에서 복구하고 USBD 호출을 통해 중지 또는 오류 상태를 승인할 때까지 더 이상의 IRP는 허용되지 않습니다. 적절한 상태는 소프트웨어 클라이언트에 오류 대 정지에 대한 특정 IRP 결과를 알려줍니다(10장 참조). 등시성 파이프 동작은 섹션 5.6에 설명되어 있습니다.

IRP는 버스를 통해 클라이언트 데이터를 이동하기 위해 여러 데이터 페이로드가 필요할 수 있습니다. 이러한 다중 데이터 페이로드 IRP에 대한 데이터 페이로드는 전체 IRP의 나머지를 포함하는 마지막 데이터 페이로드까지 최대 패킷 크기가 될 것으로 예상됩니다. 자세한 내용은 각 전송 유형에 대한 설명을 참조하세요. 을위한

IRP 데이터 버퍼를 완전히 채우지 않는 입력의 짧은 패킷(즉, 최대 크기의 데이터 페이로드 미만)은 클라이언트의 기대에 따라 두 가지 의미 중 하나를 가질 수 있습니다.

• 클라이언트는 IRP에서 다양한 크기의 데이터를 기대할 수 있습니다. 이 경우 IRP 데이터 버퍼를 채우지 않는 짧은 패킷은 단순히 "데이터 단위의 끝"을 나타내는 대역 내 구분 기호로 사용할 수 있습니다. IRP는 오류 없이 폐기되어야 하며 호스트 컨트롤러는 다음 IRP로 진행해야 합니다.

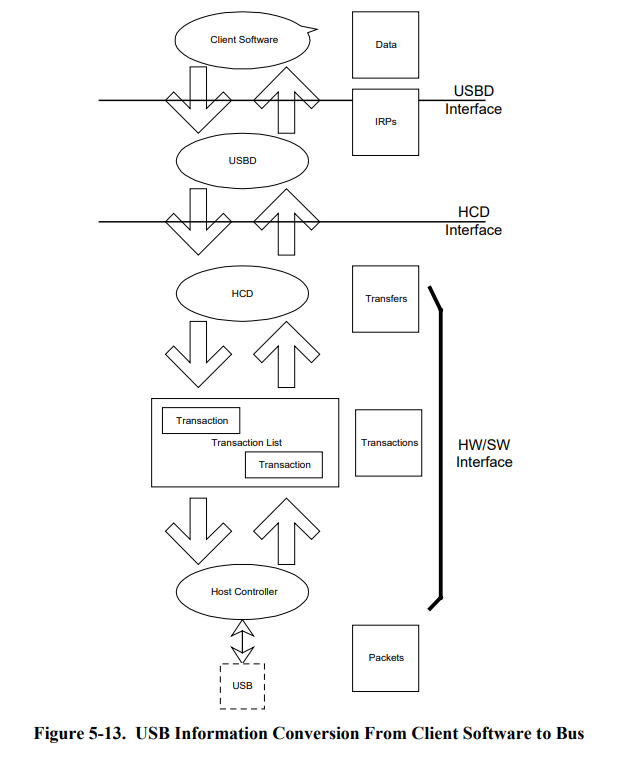
• 클라이언트는 특정 크기의 데이터를 기대할 수 있습니다. 이 경우 IRP 데이터 버퍼를 채우지 않는 짧은 패킷은 오류를 나타냅니다. IRP는 폐기되어야 하고, 파이프는 정지되어야 하며,

파이프와 관련된 보류 중인 IRP도 폐기되어야 합니다.

호스트 컨트롤러는 두 가지 경우에 다르게 동작해야 하고 주어진 IRP에 대해 어떤 방식으로 동작할지 자체적으로 알 수 없기 때문입니다. 클라이언트가 원하는 행동을 IRP별로 표시하는 것이 가능합니다.

끝점은 NAK로 응답하여 호스트에 자신이 사용 중임을 알릴 수 있습니다. NAK는 소프트웨어 클라이언트에 IRP를 반환하기 위한 폐기 조건으로 사용되지 않습니다. NAK는 수에 관계없이 발생할 수 있습니다.

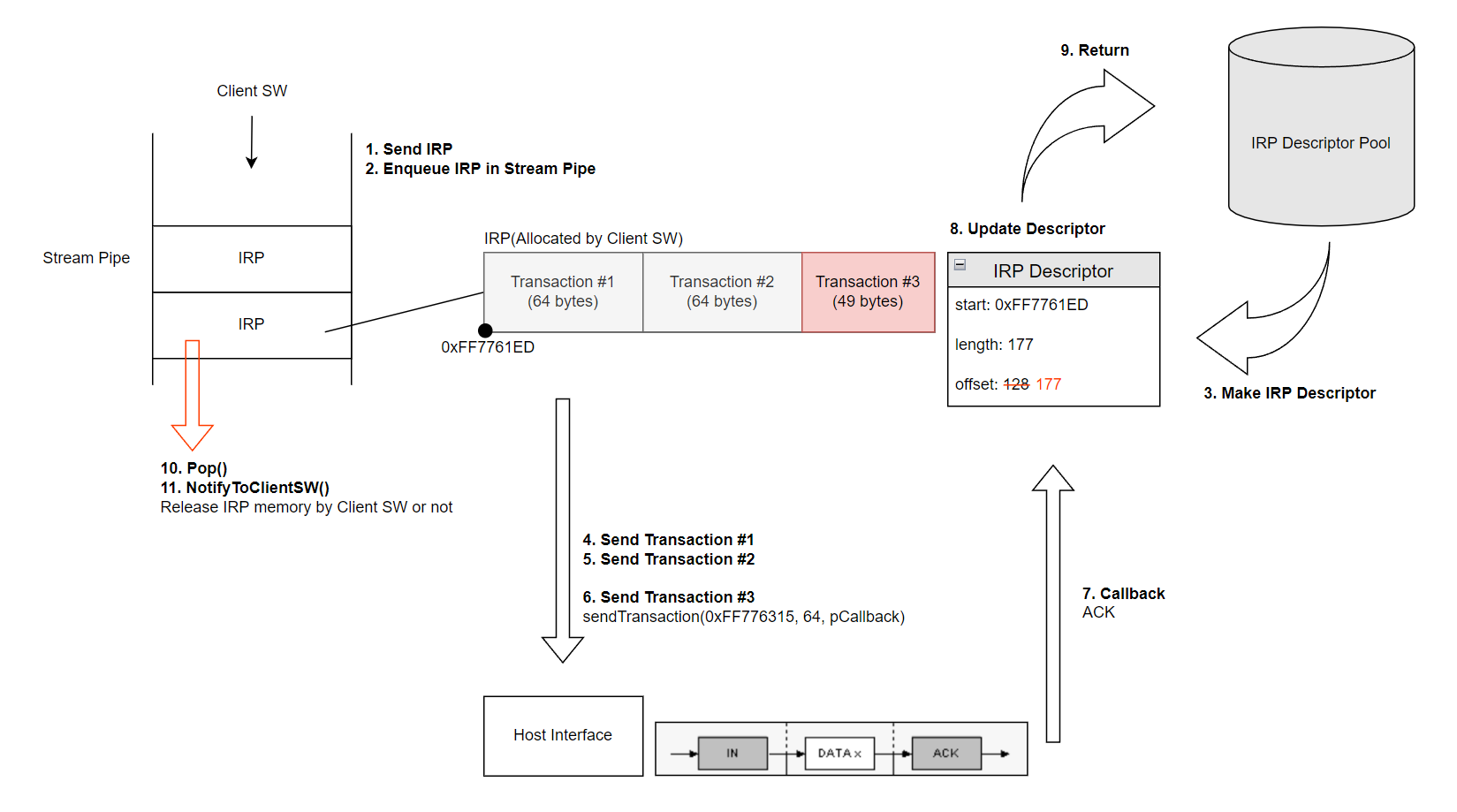
주어진 IRP의 처리. 트랜잭션에 대한 NAK 응답은 오류를 구성하지 않으며 위에서 설명한 세 가지 오류 중 하나로 계산되지 않습니다.



텍스트이(가) 표시된 사진

자동 생성된 설명

CA1. IRP 패킷의 주소를 그대로 사용하여 추가적인 메모리 할당 없이 Transaction을 구성하는 방법



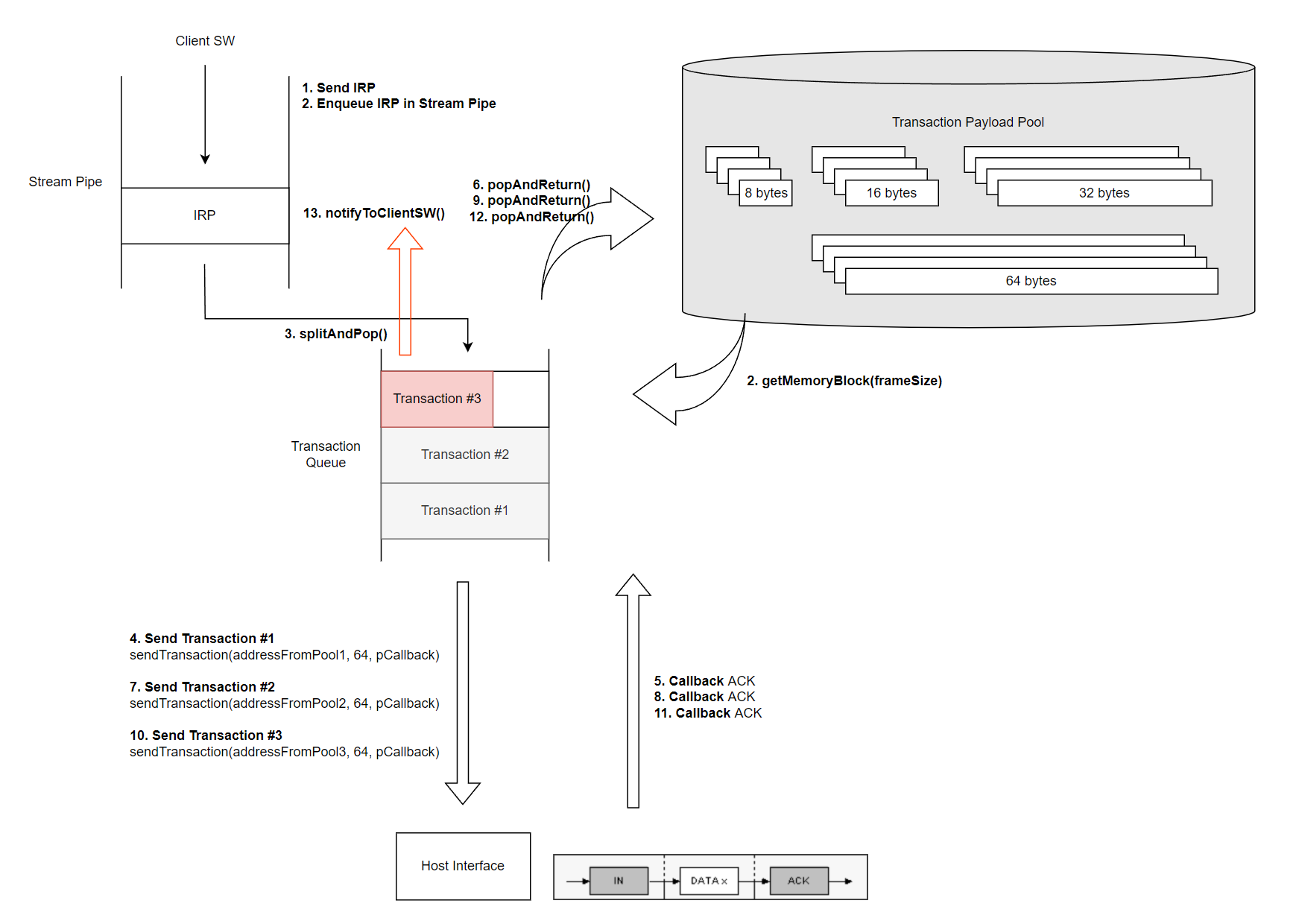
데이터 전송시에 시스템은 Client SW로부터 받은 IRP를 segmentation해서 Transaction으로 보내야 한다. 이때 Transaction의 Payload를 위한 메모리 공간을 할당하지 않고 IRP의 주소 값을 그대로 이용하는 구조가 CA1이다.

이를 위해 시스템은 IRP를 Client SW로부터 수신하면 전송한 지점을 기록하기 위한 IRP Descriptor를 생성하고 Transaction이 이루어질 때마다 offset을 기록한다. 전송길이는 Stream Pipe에 정의된 통신방식에 따라 적절한 크기의 Frame만큼 데이터를 전송한다. IRP를 구성하는 Transaction이 모두 전송되면 Queue형태인 Stream Pipe에서 IRP를 Pop하고 Client SW에게 IRP 전송이 완료되었음을 통지한다.

이 구조는 메모리의 할당 및 해제에 큰 성능 비용이 발생한다는 점에서 착안하였다. 따라서 Transaction의 Payload를 위한 메모리를 할당하지 않는 것이 핵심이며, 할당이 필요한 것은 고정된 크기의 작은 IRP Descriptor 뿐이다. IRP는 Transaction에 비해 개수가 적으므로(Transaction이 IRP의 Segment이므로) 상대적으로 적은 비용이 발생하며, 미리 할당된 Pool을 통해 비용을 더 줄일 수 있다. 또한 시스템 내에서 IRP의 데이터를 Transaction의 Payload로 복사하는 비용을 줄일 수 있다.

(퍼포먼스 ++, 가용성 +)

CA2. 메모리 할당을 하여 새로운 Transaction을 구성하는 방법



Client SW로부터 IRP를 수신하면 시스템은 Stream Pipe가 가지고 있는 전송 타입에 따라 적절한 Frame 크기로 데이터를 Segmentation해서 별도의 Transaction Queue에 넣는 구조가 CA2이다.

CA1에서 언급했듯 메모리의 할당 및 해제에는 큰 성능 비용이 발생하므로 Frame 크기에 따라 여러 memory block을 미리 할당해둔 Transaction Payload Pool을 가진다. 만약 미리 할당된 Element가 모두 소진되면 그때부터는 Heap에 메모리를 할당하여 반환한다. Pool Element의 대여/반납과 Heap에서 할당된 메모리의 일관된 처리를 위해 구현단계에서 interface에 대한 추상화가 요구된다.

Transaction Queue의 요소들은 순차적으로 Host Interface를 통해 전송되며 전송이 완료되면 Pool Element를 반환한다. IRP의 마지막(Max Frame 사이즈가 아닌 Transaction)을 전송한 후에는 반납을 진행한 후 Client SW에게 전송이 완료되었음을 통지한다.

이 구조는 Payload를 복사해야 한다는 단점이 존재한다. 이에 따른 성능 저하를 보완하기 위해 DMA(Direct Memory Access) 장치를 이용한 복사를 활용할 수도 있다.

(퍼포먼스-, 가용성- 메모리 공간 미리 많이 잡으므로, DMA 사용시 특정 하드웨어로 인해 변경용이성 -)

* 업링크는 고민해보자

CA4. Host가 polling 주기가 되면 데이터를 가져가는데(드라이버가 모아둔걸 보내주는데) 메모리 풀에다 데이터 쌓아놓고 그걸로 올려보내는 IRP 패킷 구성

* 리드는 내가 어쨌든 할당된 메모리를 가지고 있어야 한다. 프레임 사이즈에 맞추어 메모리 할당 풀을 가지고 있자
* 즉 프레임 사이즈의 풀에다가 들어오는데로 쌓아놓고 여러 프레임 메모리를 조합해서 패킷만들어 올려보내기
* 프로토콜떄 하던거 생각하면 분산된 메모리로 굳이 들고다니다가 최종적으로 한번 복사

CA5. 업링크시 IRP 패킷 풀을 구성해놓고 패킷에다가 피포로 데이터를 복사해 넣으면서 요청시 올려보내기

CA6. 이것도 DMA로?

CA?. Host Interface에서 재전송 담당

* NAK 패킷을 받으면 알아서 재전송하므로 위에서는 신경안씀

CA??. 위에서 재전송

CA7. Message Pipe에 대해서 요청을 Queueing하는 구조

* 달리 방법이 있나? 다른 후보구조를 생각해보자.

CA8. Host와의 인터페이스를 추상화하는 방식

* 오퍼레이션이 다양하지 않아서 시스템콜을 추상화하면 여러 운영체제에 대해 변경용이성

CA9. 주요 운영체제에 대한 인터페이스 제공

* 스페시픽한 오퍼레이션을 구현할 수 있다. 변경 용이성 떨어짐.

CA10. LED는 우리가 인터페이스를 정하고 맞는 LED를 납품받는 형식

CA11. LED와 USB Device 매핑해서 옵저버 패턴?

CA12. HostController와의 인터페이스

* 셋업보내고 데이터 받고 액보내고 없으면 넥보내고 이러한 동작은 스펙에 정의되어 있으므로 추상화된 인터페이스를 가진다. Send(), receive(), isData()이런걸로 추상화 => 변경용이성

CA13. Flash에서 미리 테이블을 읽고 사용하기(성능) – 하드웨어 접근시간을 아끼지만 부팅속도가 늘고 불러온 정보를 새로운 이벤트에 따라 업데이트하고 플래시에 다시 기록해야 됨

CA14. Flash에서 필요할 때 접근(성능) – 부팅속도는 빠름

CA15. KnownDevice의 자료구조 – Hash

CA16. List

CA17. 시큐리티를 위해 BlackList 등록하기(Revoke Access tactic) – 구조는 15, 16중에 결정된것과 동일하게 한다. 아이디로 관리할 것이므로

CA18. 파이프마다 버퍼를 갖는 방법 – 버퍼 풀을 갖자. 크기가 다양한가? 리소스 환경에 따라 달라질지도

CA19. 큰 버퍼내에 파이프마다 독립된 영역을 갖는 방법 – 경계를 넘지 않도록 안전장치가 있어야 한다.ㄱ 가드

CA20. 3,.0의 풀 듀플렉스 지원가능성을 위한 버퍼 구조(읽기, 쓰기 버퍼 분리) – 공간낭비

리드의 경우에는 트랜잭션을 쌓아두다가 ClientSW에서 읽기를 원할 때

20-1 => 하나로 하는 경우 변경용이성 떨어짐

CA21. 파이프의 스테이트 머신(DeviceAttached, Powered, Bus Reset)

CA22. 파이프 ISP => IDirection, IHalfDuflex, IType

CA23. Default Pipe를 별도의 모듈로 두는 구조(Increase Cohesion)

CA24. Default Pipe도 제어정보를 전송하는 파이프의 상속구조로 두는 구조

CA25. Nego를 위한 모듈을 따로 두는 구조 Reduce size of module

CA26. Configurator에 기능을 몰아넣는 구조 응집성?

CA27. 시큐리티를 위해 허용 가능한 Class를 담당하는 모듈을 두는 구조

CA28. 허용 가능한 클래스를 os로부터 받아와서 비트맵으로 저장하는 방식 => 운영체제와의 별도의 API가 필요하다.

CA29. 모두 허용해놓고 거부되는게 있을때마다 하나씩 줄여나가는 방식 => 플래시에 저장해야되는 문제, 한번은 공격당할 수 있는 가능성 => 운영체제로부터 거부될 때 허용되지 않는 것이라는 리즌이 정의되어 있어야함

CA30. 가용성을 위해 ResourceObserver를 둬서 최적의 리소스 할당 가능하도록

CA31. Watchdog형식으로 주기적으로 가용자원을 확인하는 방식

CA32. 특정 이벤트 발생시마다 가용자원을 확인하는 방식

CA33. 시스템 초기화 및 종료를 위한 모듈( Increase Cohesion)

CA34. System Initialtor가 Manager류라서 지나치게 시나리오 관장? => 여러 모듈간의 메시지 패싱으로 처리

CA35. 안정성을 위한 Graceful 종료 제공(시퀀스 다이어그램)

CA36. Device 연결구조를 트리로 표현하는 구조

CA37. 리스트로 표현하는 구조 => 논리적으로는 뎁스가 달라도 차등이 없다. 트리 구성시 순회를 위한 규칙이 없어 리스트가 더 빠르다. 허브 해제시 그 허브에 해당하는 모듈 다 찾아야함

CA38. 고유 아이디를 가지고 해시 구성. 허브 해제시 모듈 다 찾아야하는데 순회가 어려움

CA39. Device 개수가 127로 정해져있으니 풀로 정보 저장하기. 파이프도 레퍼런스 가져야함.

CA40. Direct 연결의 경우 LED와 매핑되는 다른 책임이 있으므로 구분하기

CA41. 파이프 종류에 따라 다형성으로 표현하는 방법(변경용이성) => 동일한 책임을 가진다. 전송

CA42. 주소 할당을 라운드 로빈으로 하는 방식(포인터와 127 라운드)

CA43. 1부터 빈곳을 찾아서 할당하는 구조

CA44. KnownDevice에 대해서 패스트 트랙으로 연결하는 구조

CA45. 모든 접속에 대해서 디스크립터를 확인하는 구조. 시큐리티

구조를 어떤 구조 내에서 다른 선택지가 또 있으면 34-2 이런식으로 분기할까

1. 후보 구조 평가

// 활동7. 최종 구조 설계

// 점검7-1. 충돌되는 후보 구조의 비교/분석이 적절한가? (근거)

// 점검7-2. 선정된 후보 구조의 단점/RISK 분석이 명확한가?

1. 최종 구조 설계

// 활동7. 최종 구조 설계

// 점검7-3. 최종 구조로의 통합 과정에 대한 설명이 적절한가?

// 점검7-4. 최종 구조의 단점/RISK 관리가 적절한가? (개선)