Debugging Usually Slightly Broken (USB) Devices and Drivers

USB に関する高度なデバッグについてのプレゼン．USB Device のマネジメント，USB Driver の修正についての話．

1. USB basics

USB はデータのやり取りができるバスで，ストレージやプリンタなど様々なレベルのサービスを提供できる．
→ USB Device は USB Host(PC など) に対して機能を提供する．USB Host は同時に様々な USB Device と接続できるが，USB Device は同時に複数の USB Host と接続はできない．1 つの USB Device は最大 31 個の Endpoint [1] を保持している

- Endpoint 0 はどの Device でも必ず使う．Host ←→ Device の通信ができる (コントロール転送)
- 他には，IN と OUT があり，IN は Device → Host，OUT は Host → Device の通信ができる
- IN，OUT それぞれ 15 個の割り当てがされ，$1 + 15 + 15 = 31$ 個

Endpoint には 4 種類の転送方法がある

- Control 転送: 双方向転送可能で Endpoint0 が使える．アプリケーションに利用可能
- Bulk 転送: 画像などの大きいデータや時間に依存しないデータの転送に使われる
- Interrupt 転送: 待ち時間が短いようなデータや時間に依存するデータの転送に使われる
- Isochronous 転送: サイズが大きく時間に依存するデータの転送に使われる

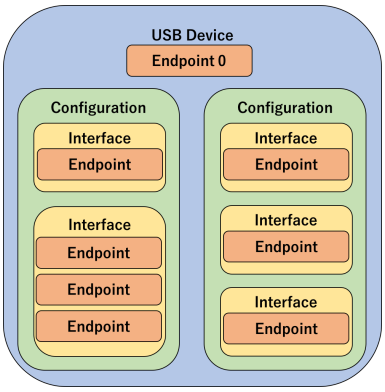

Figure1: USB Device, Configuration, Interface and Endpoint

Endpoint は Interface の中でグループ化され，Interface は Configuration の中でグループ化される．USB デバイスは複数の Configuration を持つ．Interface は機能の提供，Configuration は設定の提供．Configuration と Interface はそれぞれ 1 つずつしか動作させられないので，ある Endpoint を利用できないタイミングがある．Endpoint 0 はグループ化されないので，いつでも利用できる．

このほかに USB Descriptor [2] という重要な要素がある．

- Device Descriptor: Device Driver を選ぶための製品 ID や企業 ID などの情報
- Configuration Descriptor: Device の構成情報 必要な電力
- Interface Descriptor: Device の機能を表す情報
- Endpoint Descriptor: Endpoint の転送種類や方向などの情報

USB と接続するためには，Device Driver で接続，転送する機能を作成する必要がある．しかし，特定機能 (Device Class) は作成する必要がない．

[1] Endpoint:USB Device と Host がデータのやり取りするための FIFO バッファ
[2] 接続された USB Device はどのような機器なのかを Host が把握するためのデータ

00h	Device	Use class information in the Interface Descriptors
01h	Interface	Audio
02h	Both	Communications and CDC Control
03h	Interface	HID (Human Interface Device)
05h	Interface	Physical
06h	Interface	Image
07h	Interface	Printer
08h	Interface	Mass Storage
09h	Device	Hub
0Ah	Interface	CDC-Data
0Bh	Interface	Smart Card
0Dh	Interface	Content Security
0Eh	Interface	Video
0Fh	Interface	Personal Healthcare
10h	Interface	Audio/Video Devices
11h	Device	Billboard Device Class
DCh	Both	Diagnostic Device
E0h	Interface	Wireless Controller
EFh	Both	Miscellaneous
FEh	Interface	Application Specific
FFh	Both	Vendor Specific

Figure2: USB Classes

2. Plug & Play

USB Device を接続してから利用できるまで何が起きているか.

1. Device を接続する
2. Host は接続を検知する
3. アドレスをセットする
   (a) 128 個のアドレスを持つことができ，デフォルトは 0x00．Device ごとに++したアドレスを割り当てる
4. Device の情報 (Descriptors) を入手する
5. Configuration を選択する
   (a) Configuration が一つだけなら問題はない
   (b) 複数ある場合は，選択した Configuration の Interface を利用できるのでそれを利用する
6. Interface のための Driver を選択する (Device ではなく”Interface”のための選択)
   (a) カーネルコードの一部 (モジュール) の struct usb_driver を使う
      i. struct usb_driver は user space に network interface や block device などを提供する
      ii. いくつかの通信プロトコルによって実装され，web browser や ssh client に似ている
   (b) 登録した Driver のリストをカーネルは保持しており，Driver には対応している Device ID が記されている
   (c) この Driver リストにある Device ID と Device Descriptor を比較して一致した Driver を選択する
7. USB Device が使えるようになる

3. Plug & do what I want すべて自動で処理ことは良いが，いつでも自動で処理されるのはよろしくない.
USB に関することがすべて自動で処理されるとセキュアなシステムとは言えない.

- 一部のデバイス機能だけが必要だが，多くのデバイスを許可してしまう
- 間違った Configuration や Driver を選択する可能性がある

USB のシステム構成は 3 種類存在する

- usbX : Host のポートに接続しているもの
- X-A.B.C : 1 つのハブのポートに接続しているもの
- X-A.B.C:Y.Z : 1 つのハブにハブ 2 を繋げ，ハブ 2 のポートに接続しているもの

許可する Device 数の上限を設定する

- USB Device は authorized という属性を持つ
- usbX は authorized_default という属性を持つ
- authorized が 0 であれば，Device は未構成で，承認された時に usbguard が Driver の選択を自動で行う

Linux カーネル v4.4 からは，これが改善された

- USB Interface が authorized 属性を持つ→ Configuration を選択できる

- usbX は interface_authorized_default 属性を持つ
- autorized が 0 であれば，Driver はバインドされず，Driver の選択をする際は手動で行う

カーネルが選択する Configuration を変更することができる

- USB Device が持つ bConfigurationValue を変更すると良い

Device ID を Driver に追加することができる

- 多くの Driver は Vendor ID と Product ID がペアになっている
- しかし，ベンダーによっては Driver からこのペアを削除したり，VID が間違っていたりする
- これらを改善するためには，Driver の Device ID Table を変更する必要がある．

ID は動的に変更させることができ，その形式は 3 通りある．ただし，それぞれ 16 進数で書かないといけない

- VID+PID
- VID+PID+Intf Class
- VID+PID+Intf Class+dev$_info$:
  - $RefVID と$RefPID はデバイステーブルへの登録をするもの

新しい DeviceID を Driver に追加したいときは Driver が持つ new_id ファイルに書き込めば良い
DeviceID を消したいときは，Driver が持つ remove_id ファイルに書き込めば良い
特定の Interface をバインドしたいときは bind ファイル，解除したいときは bind ファイルに書き込めば良い

4. Plug & tell me more ここまで紹介してきた方法だと，予期せぬ問題や振る舞いが起きてしまい，コードをデバッグしようとしても何が問題かわからない
USB はホストを制御するバスで，Device を使うにはホストは USB を初期化しないといけない．USB はポーリング[3]されたバスで，ホストはそれぞれの Devie をポーリングしてデータの要求や送信をする．
USB transport の話をすると Tranfer の中に transaction が存在する．

- transaction では，Endpoint に指定されたパケット上限までのデータを渡す
- transfer は，複数の transaction を持ち，その transaction の長さは可変である
- transaction は Host Controller Driver や USB Device Driver の操作であるため，かなり低いレイヤーである

Linux カーネルでは，transfer は USB Request Block(URB)[4]になっている
典型的な USB Driver は USB に関わる 3 つの関数とユーザ空間で使うような関数から成り立っている

- probe() : Device をチェックしてリソースを割り当て
- disconnect() : リソースを解放する
- complete() : 状態を確認して，データを朱徳したり，再送信したりする

USB Device での典型的なバグ

- Descriptor が間違っていること
- 何かミスがあった時のエラーパスがないこと
- complete() での正しいエラー処理がないこと
- パケットの形式が正しくないこと

[3] ポーリング:連携動作する際に、送信/処理要求がないか、一つ一つの相手に確認する方式
[4] USB バスを通って送られてきたデータのまとまり

HW USB sniffers という USB Driver の性能やプロトコルの性能を評価することができるツールがある．かなり便利であるが高額．Open Hardware なら材料費約 100$で作ることができる．
USB monitor は，sumit() や complete() などの USB に関わるイベントのログを取るのだが，transfer レベルのデータしか取れないず，transaction レベルのデータを取得できない．
URB バッファのデータは常に妥当であるわけではない．妥当性は transfer の結果と Endpoint の方向方向によって決まる．

5. Summary

- USB Descriptor はポートのようなもの
- lsusb を使うことで情報を取得できる
- 各 Device Driver は互換性のある Device リストを宣言する
- USB Device は SysFS を通して管理することができる
- Driver は URB を用いて通信する
- Open HardWare を使えば，USB traffic を監視するのに膨大な金額を使う必要はない