Linux の USB デバイスドライバの抽象化と デバイスドライバ生成支援システムに関する考察

水川 晶太　　片山 徹郎

宮崎大学 工学部 情報システム工学科

1　はじめに

デバイスドライバの作成には，オペレーティングシステム (OS) やデバイスに関する多大な知識を必要とする．またデバイスドライバは各 OS，各デバイスごとに作成しなければならない為，デバイスドライバの開発者にかかる負担は大きいものとなっている．本研究では，デバイスドライバの開発にかかる負担を軽減し，デバイスドライバ開発者の作業の分担化を目的として，デバイスドライバのソースコードの抽象化を行う．

デバイスドライバの性質からデバイスドライバ仕様 (OS とデバイスの間で入出力されるデータの受け渡しを行う部分)，OS 依存仕様 (OS とデータを入出力するためのインターフェース)，デバイス依存仕様 (デバイスとデータを入出力するためのインターフェース) の 3 つの仕様に抽象化できると考えている．この 3 つの仕様に抽象化できると，デバイスドライバを他の OS へ移植する際には，OS 依存仕様のみを変更するだけでよい．また機能は同じだが，ハードウェアの構造が異なる製品のデバイスドライバは，デバイス依存仕様のみを書き換えるだけでよい．このように，この 3 つの仕様に抽象化することにより，デバイスドライバ開発の手間と時間を大幅に削減できる [1].

本研究では，それを実現するための事前研究として Linux の USB デバイスを対象とし，複数のデバイスドライバのソースコードを解析し，デバイスドライバの一関数である PROBE 関数のアルゴリズムを定義し，そのアルゴリズムに従ってコードを分割する [2]．また，コード分割による抽象化の有効性を検証するためのツールを試作する．

2　PROBE 関数のコード分割

本研究では，USB デバイスドライバの PROBE 関数のコードを機能ごとに分割する．デバイスドライバのコードを分割するためには，どのような基準で分割するかが重要である．本稿では，複数の USB デバイスドライバの PROBE 関数を観察，分析することによって，USB デバイスドライバの PROBE 関数のアルゴリズムを定義し，それに基づいてコードを分割した．

図 1 に，今回定義した PROBE 関数のアルゴリズムを示す．以下，このアルゴリズムについて説明する．まず，PROBE 関数で使う変数を宣言する．次に，このデバイスドライバで扱えるデバイスかどうかの判定を複数回行う．その判定に必要な値を代入するなどの操作は，判定する前などに適時実施する．判定において不適切な場合は，NULL を返して終了し，適切な場合は次の判定へと進む．全ての判定において適切と判断された場合は，そのデバイスを登録するために必要な値であるデバイス構造体についての操作を行い，デバイス構造体を返して終了する．

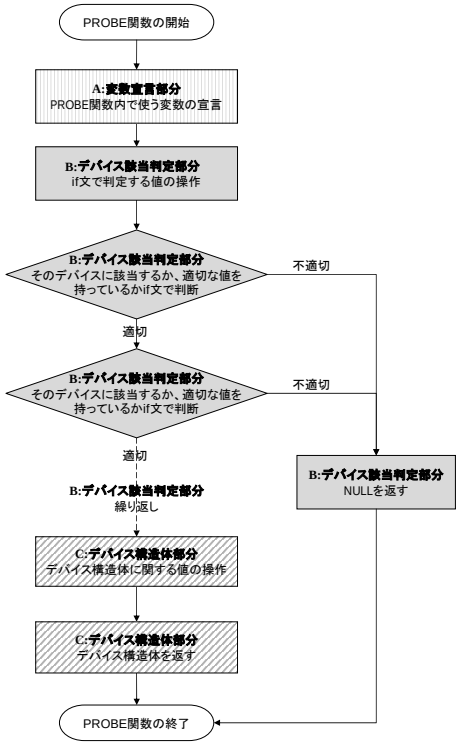

図 1: PROBE 関数のアルゴリズム

このアルゴリズムに基づいて，**A 変数宣言部分**，**B デバイス該当判定部分**，**C デバイス構造体部分**を定義する．また図 1 のアルゴリズム中に記述できない部分は **D その他の部分**として定義する．

既存のデバイスドライバのソースコードを，定義した 4 つの部分に分割し抽象化する．例として USB マウスとハブ，プリンターのソースコードを分割した．その結果，既存のコードのまま，あるいはコードの一部分に変更を加えることによって，定義したアルゴリズムに従うように分割することができた．

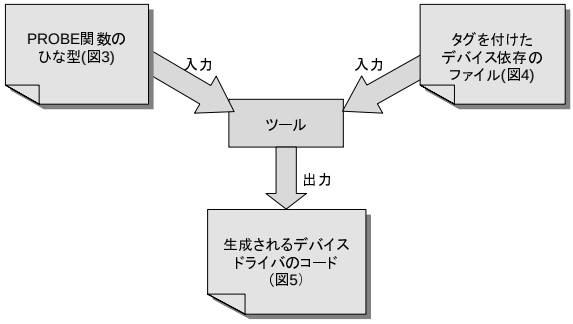

図 2: ツールの概要

```
/*---The model of PROBE function---*/
/*---A part---*/
static void *##DEVICE_NAME_1##_probe
(struct usb_device *dev,
unsigned int i,const struct usb_device_id *id)
/*---B part---*/
if(##IF_CONDITION##){
return NULL;}
/*---C part---*/
return ##DEVICE_NAME_2##;
```

図 3: PROBE 関数のひな型ファイル

3 デバイスドライバ生成支援ツールの試作

図 1 のアルゴリズムに従うようにコードを分割することによって，デバイスドライバを抽象化することが，有益であるかどうかを確認するためのツールを現在作成中である．簡単にツールの紹介を行う．

まず，USB デバイスドライバの PROBE 関数に共通な部分をひな型として用意する．次に，各行に，図 1 のアルゴリズムの A,B,C,D のどの部分に分割されるかを示すタグを付けたファイルを用意する．このファイルのコードには，ひな型に追加する部分が分かるようにする為のタグもつけておく．なおこのファイルには，ひな型に記述されている部分は記述しておく必要がない (記述してもよい)．このファイルは，デバイスごとに異なるファイルとなるのでデバイス依存のファイルであると言える．

デバイスドライバ生成支援ツールに，この 2 つのファイルを入力すると，デバイスドライバのソースコードが出力される．図 2 にツールの概要，図 3 に PROBE 関数のひな型，図 4 にタグを付けたデバイス依存のファイル，図 5 に生成されたデバイスドライバのコードを，それぞれ示す．

4 考察

デバイスドライバの開発にかかる負担の軽減と，デバイスドライバ開発者の作業の分担化を目的として，Linux の USB デバイスドライバの PROBE 関数のアルゴリズムを定義し，そのアルゴリムに従って既存のデバイスドライバのソースコードを，機能ごとに分割した．この分割結果により，デバイスドライバの構造が分かりやすくなり，デバイスドライバのひな型のようなものができたと捉えることができる．

また，コード分割による抽象化の有効性を検証するためのツールの試作を行った．その結果，デバイスドライバのひな型が出来て，ある程度，コード分割の有効性を示すことが出来た．今後の研究で，ひな型の部分を，さらに増やす必要があると考えている．

1 章で述べたように，本研究は，デバイスドライバ仕様，OS 依存仕様，デバイス依存仕様の 3 つの仕様へ抽象化することを目標としている．他の OS や，他のデバイスのデバイスドライバを分析していき，抽象度を高め，3 つの仕様への抽象化を行っていきたい．

```
#A#static void *##DEVICE_NAME_1usb_mouse##_probe
(struct usb_device *dev,
unsigned int ifnum,const struct usb_device_id *id)
#A#{
#A# struct usb_interface *iface;
#A# struct usb_interface_descriptor *interface;
                        省略
#B# endpoint = interface->endpoint + 0;
#B#
if ##IFBEGIN(!(endpoint->bEndpointAddress & 0x80))##
##IFEND##return NULL;
#B# if ##IFBEGIN((endpoint->bmAttributes & 3) != 3)##
##IFEND##return NULL;
#B# /* wacom tablets match... */
#B#
if ##IFBEGIN(dev->descriptor.idVendor == 0x056a)##
##IFEND##return NULL;
                        省略
```

図 4: タグを付けたデバイス依存のファイル (一部分)

```
static void *usb_mouse_probe(struct usb_device *dev,
{
struct usb_interface *iface;
struct usb_interface_descriptor *interface;
                        省略
endpoint = interface->endpoint + 0;
if(!(endpoint->bEndpointAddress & 0x80)){
return NULL;}
if((endpoint->bmAttributes & 3) != 3){
return NULL;}
/* wacom tablets match... */
if(dev->descriptor.idVendor == 0x056a){
return NULL;}
                        省略
```

図 5: 生成されたデバイスドライバのコード (一部分)

参考文献

[1] 奥野幹也, 片山徹郎, 最所圭三, 福田晃:“ UNIX 系 OS におけるデバイスドライバの抽象化と生成システムの実現 ”, 情報処理学会論文誌, Vol.41, No.6, pp.1755-1765 (2000).

[2] 水川晶太, 片山徹郎: “ Linux の USB デバイスドライバの抽象化に関する考察 ”, 情報処理学会 OS 研究会, 2004-OS-97(SWoPP2004), pp.9-16 (2004).