

LinuxのUSBデバイスドライバのコード分割による 抽象化に関する研究

水川 晶太¹⁾・片山 徹郎²⁾

Study on Abstraction of USB Device Drivers on Linux with Dividing Source Codes of the Device Drivers

Shota MIZUKAWA, Tetsuro KATAYAMA

Abstract

Writing device drivers spends much time and makes efforts because it needs much knowledge of the target operating system(OS) and device. And, many device drivers must be written. This research aims at reducing the burdens of programmers who write device drivers by abstracting device drivers. As a preparatory research, this paper adopts Linux as a target OS and USB(Universal Serial Bus) device as a target device, and defines an algorithm for PROBE function of USB device drivers on Linux. USB device drivers on Linux are abstracted with dividing source codes of the device drivers by the algorithm. It assists the programmers in understanding structure or behavior of device drivers.

Key Words:

device driver, operating system(OS), USB(Universal Serial Bus) device, Linux, PROBE function

1. はじめに

コンピュータをとりまくデバイスは多種多様であり、新しいデバイスが次々と登場している。このデバイスを、コンピュータのオペレーティングシステム(OS)がアクセスして制御するためのプログラムが「デバイスドライバ」である。

デバイスドライバは、

- 作成する際に OS だけでなく、デバイスのハードウェアに関する多大な知識を必要とする。
- 各 OS、および、各デバイスごとに作成しなければならないため、開発が必要なデバイスドライバの個数は OS の種類とデバイスの個数の積に比例し、その数は膨大である¹⁾。

以上のような理由から、デバイスドライバの開発者

にかかる負担は大きい。OSの開発や移植の際には、デバイスドライバの作成が最も時間と労力を要する。またデバイスは、今後も増え続けることが明確であり、デバイスドライバの開発者にかかる負担はますます大きくなる。よって、開発者の負担を減らし、デバイスドライバの開発を少しでも容易に行えるようにしなければならない。

本研究では、デバイスドライバ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、それを達成するために、デバイスドライバの抽象化を行う。本稿では、LinuxのUSB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、USBマウスとUSBハブを例にとり、既存のコードを、定義したアルゴリズムに従って分割することによって抽象化する。

1) 情報工学専攻大学院生

2) 情報システム工学科助教授

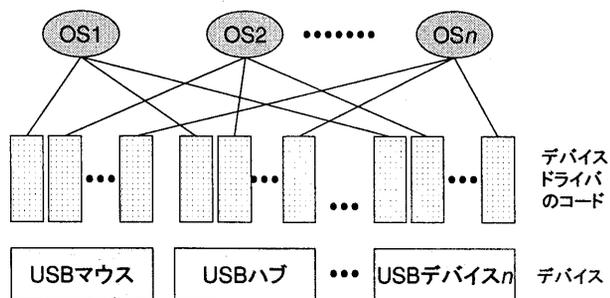


図 1: 現状のデバイスドライバ開発

2. デバイスドライバの抽象化とコード分割

この章では、デバイスドライバの抽象化と、デバイスドライバのコード分割について述べる。

2.1 デバイスドライバの抽象化

デバイスドライバは、OS がデバイスにアクセスして制御するプログラムである。またアプリケーションがデバイスにアクセスする際には、必ず OS を通して行われる。よってデバイスドライバは、以下の共通する特徴を持っている。

- OS やそのデバイスに大きく依存する。
- アプリケーションは、デバイスが何であるかを知る必要がない。

デバイスドライバは図1に示すように、各 OS、および、各デバイスごとに用意しなければならないため、開発効率が非常に悪い。そこで上記のデバイスドライバが持っている特徴を生かして、デバイスドライバの開発効率を上げることができれば、非常に有用である。

デバイスドライバが共通に持つ特徴を生かすためには、その特徴を明確に表さなければならない。その一手段として有効と考えられる手法がデバイスドライバの抽象化である。

既にデバイスドライバは、各機能を関数という形で抽象化している。デバイスドライバのプログラムが同じ種類のデバイス用のものであれば、機能的に同じ処理を行っていることに着目すると、その構造は抽象化できる。またデバイスドライバは、OS とデバイスとの間で、データをやり取りするためのプログラムである。これらのことを利用すると、デバイスドライバは以下の3つの部分に抽象化できる。

- 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 とデータを入出力するためのインターフェース

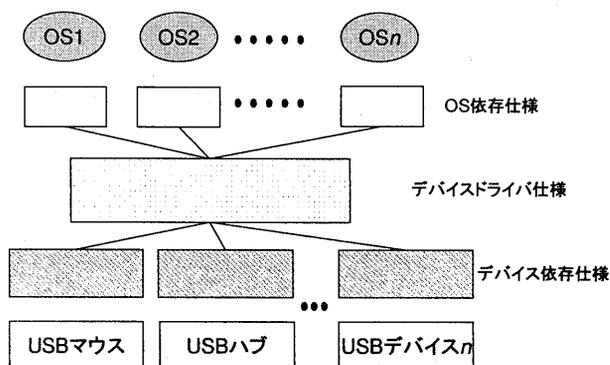


図 2: 3つの仕様を用いたデバイスドライバ開発

- 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

以上のことから、デバイスドライバを複数の OS やデバイスに対応できるように、作成の段階で上述の3つの部分に抽象化することは妥当なことであり、重要なことである。この抽象化した3つの部分を、それぞれ以下のように定義する。

• デバイスドライバ仕様

デバイスが持つ機能を使用するために必要なデータの種類と、そのデータを入出力する大まかな流れを示す。

• OS 依存仕様

OS からのデバイスへのデータの与え方と、デバイスドライバを OS から呼び出す際のインターフェースを記述する。

• デバイス依存仕様

デバイスドライバが、デバイスに対してデータを入出力する際のインターフェースやタイミング、データ型などを記述する。

この3つの仕様を用いると、図2に示すようなデバイスドライバ開発が実現できる。このような環境ができると、デバイスドライバを他の OS へ移植する際、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変更するだけでよい。このように、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。

さらに、3つの仕様のそれぞれは、開発する際に必要とする知識が異なるので、開発に必要な知識が少なくすむという点でも開発者の負担を軽減できる。OS

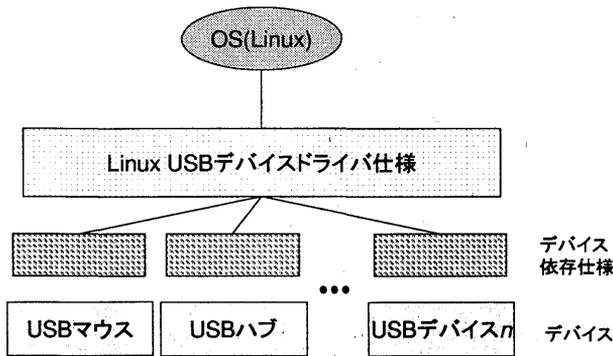


図 3: 本稿で実現するデバイスドライバ開発

依存仕様の開発者は OS に関する知識のみ、デバイス依存仕様の開発者はデバイスに関する知識のみを知っていれば、デバイスドライバの開発に携わることが可能になる。

2.2 コード分割

本研究では、デバイスドライバを前節の3つの仕様に抽象化することを目指す。本稿では、このことを実現するための事前研究として、OSは、Linuxの1つに限定し、LinuxのUSBデバイスドライバ仕様を抽出する。具体的には、USBデバイスとしてUSBマウスとUSBハブを例にとり、これらのデバイスドライバの一関数であるPROBE関数に着目して、PROBE関数のコードを機能ごとに分割することにより、デバイスドライバの抽象化を行う。

この研究によって、USBデバイスのこの関数ではこのようなことを書けばよいというガイドラインがある程度できるようになり、新たなデバイスのデバイスドライバ開発が容易になることが期待できる。本稿で行うデバイスドライバのコード分割によって、図3のようなデバイスドライバ開発が実現できると考える。

3. 研究の対象

この章では、研究の対象としたOSのLinux、デバイスのUSBデバイス、Linux USBデバイスドライバの構造、関数のPROBE関数について、それぞれ詳細を述べる。

3.1 Linux

対象としたOSは、Linux²⁾である。Linuxは、UNIX互換のOSで、学術機関を中心に普及が進んでおり、企業のインターネットサーバ用のOSとしても多く採用されている。Linuxは、デバイスドライバを含めたカーネルのソースコードが公開されているため、本稿で行うデバイスドライバのコード分割に最

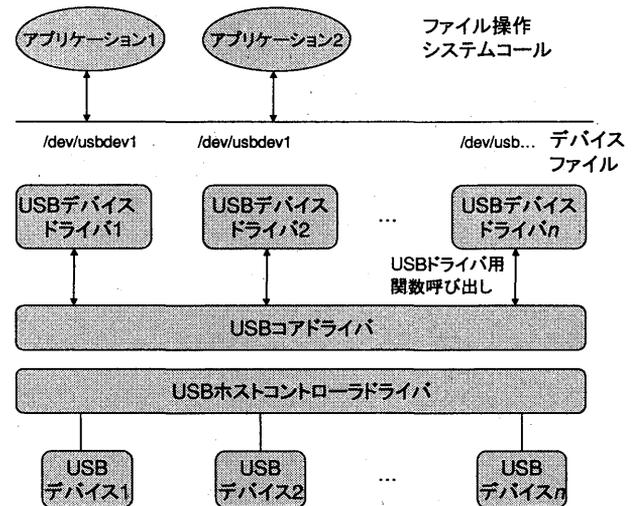


図 4: Linux USB デバイスドライバ

適であると判断したので研究の対象とした。本稿で対象としたLinuxのカーネルのバージョンは、2.4.18であり、ディストリビューションは、Vine Linux 2.5のFTP版を使用した。

3.2 USB デバイス

対象としたデバイスの種類は、USB(Universal Serial Bus)³⁾デバイスである。USBデバイスは、一般のPC(パーソナルコンピュータ)でも広く利用されるようになった。またUSB2.0の規格では、480Mbps(理論値)のスピードで通信できる⁴⁾ため、有益性も増している。USBデバイスは、増え続けるデバイスの中でも特に増加の勢いが激しいため、USBデバイスのためのデバイスドライバの生産性を上げる事が求められている。また、USBデバイスを開発する際のデバイスドライバの作成にかかる時間を減らすことができれば、新たなUSBデバイス開発にも役立つ。以上のような理由から、USBデバイスを今回の研究の対象とした。

3.3 Linux USB デバイスドライバ

Linux USB デバイスドライバの構造を図4に、USB デバイスドライバ部分の構造を図5に示す。

USB デバイスは、デバイスドライバに割り付けられたデバイスファイルに対してシステムコール(open, read, write, close, ioctl)を発行することによって操作できる。USB デバイスドライバは各デバイスごとに存在し、アプリケーションからの要求を処理する。

USB コアドライバは、USB デバイスドライバから呼び出す関数を提供する部分である。USB デバイスドライバは、USB コア関数群を利用してUSB デバイ

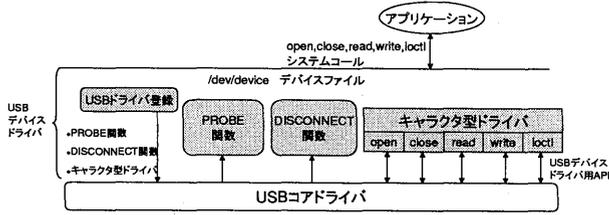


図 5: USB デバイスドライバの構造

スとのデータ転送を行う。USB コアダライバの下位は USB ホストコントローラドライバである。ホストコントローラは USB ホスト側のチップであり、UHCI (Universal Host Controller Interface) と OHCI (Open Host Controller Interface) の 2 種類が存在する。Linux には、それぞれのコントローラに対応するドライバが存在する⁵⁾。

3.4 PROBE 関数

Linux の USB デバイスドライバの中で、今回、研究の対象とした部分が PROBE 関数である。PROBE 関数は USB デバイス接続時のドライバ検出関数である。

USB コアダライバが USB デバイスを検出した際に、この PROBE 関数を呼び出す。ベンダ ID, プロダクト ID, デバイスクラス, エンドポイントなどのデバイス情報が引数に与えられ、デバイス情報から担当するデバイスであるかどうかを判断する。PROBE 関数が対象デバイスであると判断すると、ドライバが使用するメモリなどのリソース確保を行い、USB デバイスドライバが使用可能となる。

4. PROBE 関数のコード分割

本稿では、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。この章では、実際に行ったコード分割について具体的に述べる。USB デバイスの対象として、USB マウスと USB ハブを選ぶ。

4.1 コード分割の基準

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 6 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

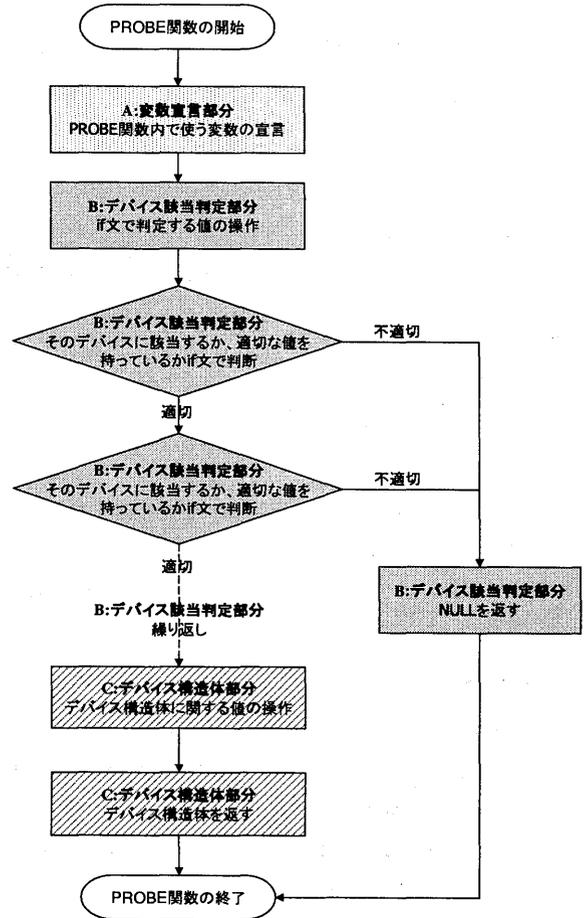


図 6: PROBE 関数のアルゴリズム

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。

このアルゴリズムに基づいて、実際に以下の 4 つの部分にコードを分割する。

A 変数宣言部分

PROBE 関数内で使用する変数を宣言している部分である。

B デバイス該当判定部分

3.4 節でも述べたように、デバイスが接続されて USB コアから呼び出された PROBE 関数は、ベンダ ID, プロダクト ID, デバイスクラス, エンドポイントなどのデバイス情報が引数に与えられ、デバイス情報から担当するデバイスかどうかを判断する。その役割のメイン部分が、この「デバイ

ス該当判定部分」である。つまりこの部分では、if文によってデバイス情報のある値を判定して、担当できるデバイスであるかどうかを判断する。担当できないデバイスであると判断した場合は、NULLを返して終了する。ここで判定するデバイス情報には、エンドポイントに関する値などがある。

C デバイス構造体部分

上述の「デバイス該当判定部分」で、最終的に担当できるデバイスであった場合は、そのデバイスに関する情報の集まり、つまり構造体（構造体変数）を戻り値として返す。この構造体をデバイス構造体と呼ぶ。「デバイス構造体部分」では、デバイス構造体に関係のある部分である。例えば、デバイス構造体の構成要素（メンバ）に必要な値を代入している部分などである。

D その他の部分

PROBE関数の中で、他の関数を呼び出して何らかの操作を行っている場合がある。例えば、デバイスの情報をレジスタに登録する処理などである。また、`printk`文や、`info`文、`err`文などによってカーネルに、デバイスの発見などの情報を出力する場合がある。これらの部分は、「その他の部分」とした。ただしこの部分が、「デバイス該当判定部分」や「デバイス構造体部分」に関係が深いと判断した場合は、そちらの方に分類する。例えば`err`文が、「デバイス該当判定部分」のif文の中にある場合などである。また、この「その他の部分」によって、「デバイス該当判定部分」や「デバイス構造体部分」が分断されることもあり得る。それは、「その他の部分」が、「デバイス該当判定部分」や「デバイス構造体部分」の途中に行わなければならない処理を含むからである。よって、「その他の部分」については、図6のアルゴリズム中には記述していない。

4.2 コード分割の例

この節では、実際のデバイスドライバのコード分割について書く。前節のコード分割の基準に基づいて、実際にUSBマウスとUSBハブの、デバイスドライバのコードを分割する。

4.2.1 USBマウスのコード分割

USBマウスのデバイスドライバのコード分割の結果を、図7に示す。図中の「その他の部分」で行っている処理について説明する。D1の「その他の部分」は、

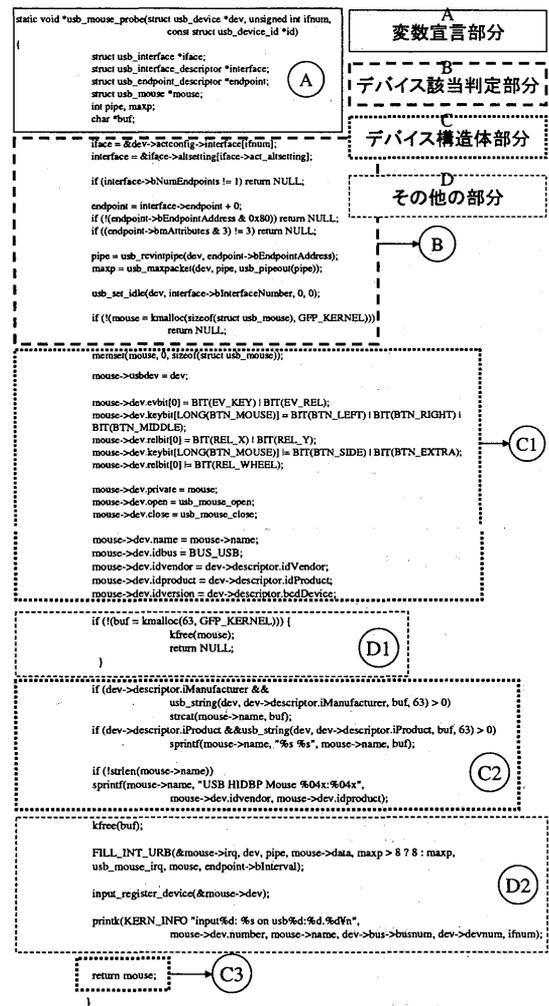


図 7: USBマウスのデバイスドライバコード分割の例

`kmalloc`文によりメモリを `char` 型ポインタ変数 `buf` へ割り当てる。もしメモリ割り当てができない場合は、デバイス構造体変数 `mouse` のメモリを開放し、このデバイスドライバでは扱えないと判断して NULL を返して終了する。

D2の「その他の部分」では、関数の呼び出しによって、USBマウス構造体の情報をレジスタに登録したり、`printk`文により、接続されたUSBマウスの情報をカーネルに出力している。

4.2.2 USBハブのコード分割

次に、USBハブのデバイスドライバのコード分割の例を図8に示す。図中の「その他の部分」のD1で行っている処理を説明する。

D1-1 `info`文によって、USBハブが見つかったことをカーネルにログとして出力する。

D1-2 この処理は、USBマウスのデバイスドライバ(図

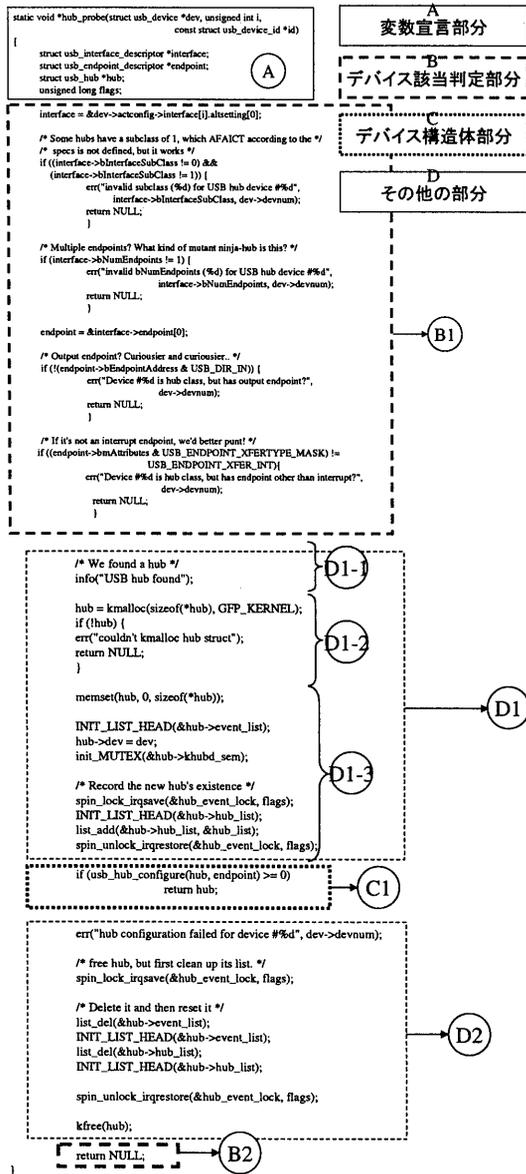


図 8: USB ハブのデバイスドライバコードの分割の例

7) の D1 と同じ処理であり、デバイス構造体 hub へメモリを割り当て、もし割り当てに失敗した場合は、NULL を返して終了する。

D1-3 USB マウスのデバイスドライバ (図 7) の D2 と同じ処理であり、USB ハブのデバイス構造体 hub の情報をレジスタに登録している。

USB ハブのデバイスドライバ内では、USB マウスのデバイスドライバ (図 7) の C1, C2 のようなデバイス構造体に関する値の操作が全く行われていない。これは、C1 で呼び出される usb_hub_configure 関数内や、usb_hub_configure 関数が呼び出す他の関数内で、デバイス構造体に関する値の操作が行われている

からである。

4.1 節で定義した PROBE 関数のアルゴリズム (図 6 参照) では、最後にデバイス構造体を返している。しかし、この USB ハブではまだ途中であるにもかかわらず、C1 でデバイス構造体 hub を返している。

C1 では usb_hub_configure 関数を呼び出す。この usb_hub_configure 関数には、PROBE 関数と同じように、接続された USB ハブがこのデバイスドライバで扱えるかどうかを判定するという機能がある。正常ならば 0 を、そうでなければ -1 を PROBE 関数に返して終了する。-1 が返ってきた場合、PROBE 関数としては NULL を返さなければならない。ここで、usb_hub_configure 関数から -1 が返ってきて直ちに NULL を返すわけにはいかないため、D2 の処理が必要になってくる。なぜならば、usb_hub_configure 関数を呼び出す前に、USB マウスのデバイスドライバ (図 7) の B の部分と同じように、D1-3 で USB ハブのデバイス構造体の情報をレジスタに登録しているため、usb_hub_configure 関数によって不適切と判断されたデバイスの記録を抹消しなければならないからである。この処理が D2 の部分である。この D2 の処理を行った後、NULL を返して終了する。これらの理由から、デバイス構造体 hub を返す位置が C1 の部分になっている。

4.3 コードの変更

第 2 章で述べたように、本稿におけるデバイスドライバのコード分割は、デバイスドライバの抽象化を目的としている。抽象化につなげるために、コード分割を最適化しなければならない。つまり PROBE 関数を機能ごとにコードを分割するためには、同じ機能をできるだけひとまとまりにする必要がある。またコードを図 6 のアルゴリズムと同じにしなければ、定義したアルゴリズムの正当性が実証できない。これらの事を実現するために、コードの変更を行った。

まず USB マウスのデバイスドライバ (図 7 参照) について説明する。D1, D2 の「その他の部分」によって、C の「デバイス構造体部分」が C1, C2, C3 の 3 つの部分に分断されている。4.1 節で述べたように「その他の部分」は、「デバイス該当判定部分」や「デバイス構造体部分」の途中で行わなければならない処理を含む。D1 と D2 はまさにこのような処理であり、D1 と D2 によって C が分断されることはやむを得ない。

また、分断されている箇所は、C の「デバイス構造体部分」のみであり、この C を分断しているのは、D の「その他の部分」であることから、現状のままで、図 6 のアルゴリズムに従っていると判断できる。よっ

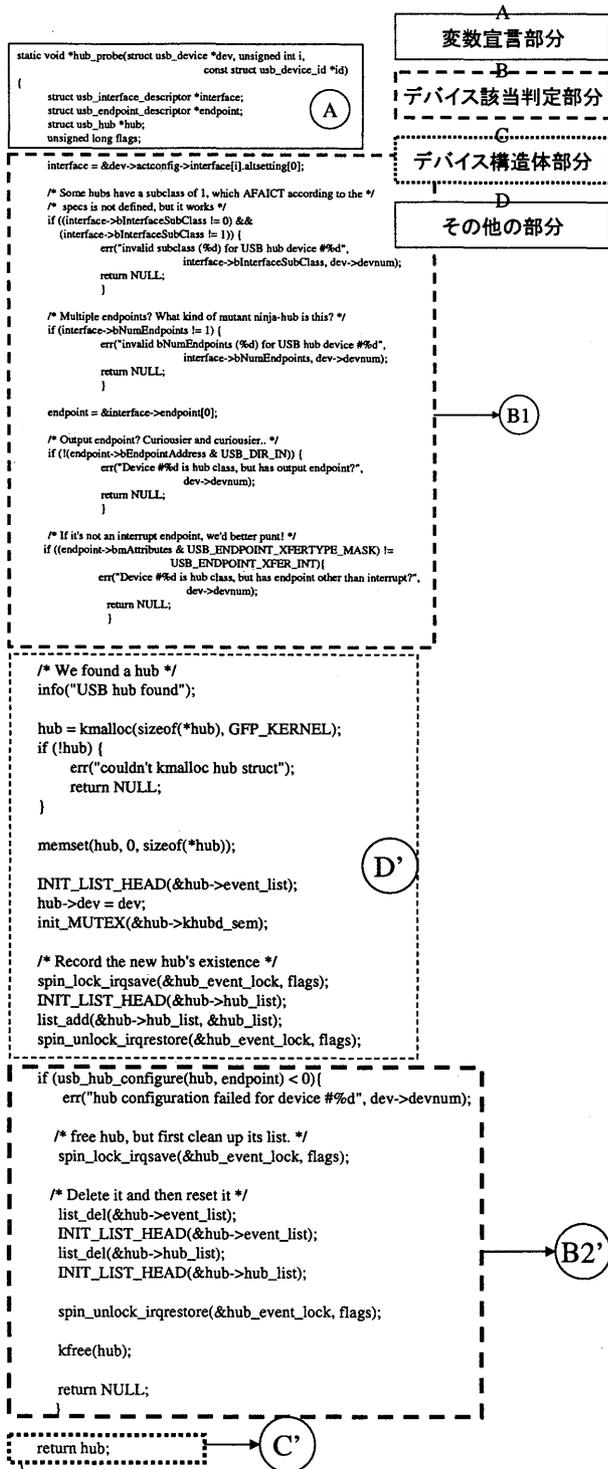


図 9: USB ハブのデバイスドライバコードの変更

て、USB マウスのデバイスドライバのコードは変更する必要がない。

次に USB ハブのデバイスドライバのコードの変更について説明する。図 8 を見ると、USB マウスのデバイスドライバの場合と異なり、「デバイス該当判定部分」である B1 と B2 の間に、D の「その他の部分」以外に加えて、C1 の「デバイス構造体部分」が含まれてしまっている。ここで、図 6 のアルゴリズムに従うように、デバイスドライバのコードを変更した。コード変更後のコード分割の例を図 9 に示す。

変更点は、図 8 の C1 の if 文の変更である。

i) if 文の判定を

```
if(usb_hub_configure(hub,endpoint) >= 0)
から
if(usb_hub_configure(hub,endpoint) < 0)
に変更
```

ii) if 文の中に D2 と B2 を入れ、PROBE 関数の最後でデバイス構造体 hub を返すように変更

以上の変更により、C1 と D2 をひとまとまりの「デバイス該当判定部分」の B2' として分割できた。またデバイス構造体 hub を最後で返すことができ、この部分が C' となった。よって、図 6 のアルゴリズムに従って処理を行うようにコードを変更することができた。

図 8 と図 9 を比べてみると、コード変更後の図 9 の方が PROBE 関数の機能ごとに、ひとつのかたまりにできていることが分かる。なお、コードを変更したデバイスドライバで USB ハブが正常に動くことを確認した。

よって、他の USB デバイスのデバイスドライバの PROBE 関数のコードを 4.1 節で述べた 4 つの部分に分割した後、このようなコードの変更を行えば、USB ハブと同じように、図 6 のアルゴリズムに従うようにすることができると思える。

5. 考察

本稿では、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、既存のデバイスドライバのコードを機能ごとに分割した。デバイスドライバのコードを分割したことによって、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。

このひな型ができたことにより、「この部分はデバイス該当判定部分だからこのような内容を書けばいい」ということが判断できるようになり、デバイスドライ

バの開発にかかる負担を軽減できたと言える。またこのことはデバイスドライバを書いたことがない人についても言えるので、新しいデバイスドライバ開発者の育成にも役立つと考える。

本稿では、Linux USB デバイスドライバの PROBE 関数に対してアルゴリズムを定義し、「変数宣言部分」、「デバイス該当判定部分」、「デバイス構造体部分」、「その他の部分」の4つの部分を考案し、デバイスドライバのコードを実際に分割することができた。しかしこの4つの部分に分割することが最適なのかは、今後の研究で再考の余地がある。

USB デバイスではデータ転送を行う際に、URB(USB Request Block)⁶⁾ という概念を用いる。現在この URB データの取り扱いの流れに着目して、新たな分割基準を考案中である。この URB データに着目したコードの分割では、PROBE 関数に加えて、open, close, DISCONNECT 関数を含めて抽象化できると考えている。

デバイスドライバの作成に関する製品として、エクセルソフト社の WinDriver⁷⁾ などがある。これらの製品は、対象とするデバイスを決定すると、自動的にデバイスドライバのコードのひな型を作成する。しかしながら、ひな型の中身については、人手により埋めなければならない。その部分には OS に関する部分、デバイスに関する部分が両方含まれることになる。

これに対して、本研究では、3つの仕様へ抽象化することを目指している。このことが実現できれば、デバイスドライバ開発者の作業の分担が見込まれ、他の OS や他のデバイスへの移植も容易になると考えられる。

6. まとめ

本稿では、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の作成における作業の分担化を目的として、Linux の USB 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、USB マウスと USB ハブを例にとり、既存のコードを、そのアルゴリズムに従って分割した。その結果、デバイスドライバのひな型を作ることができ、デバイスドライバの開発者にとって、デバイスドライバのコードを書く上での指針とすることができると考える。

今後の課題を以下に述べる。

- 新しい分割基準の考案

デバイスドライバの開発にかかる負担を軽減する上で、本稿で考案した「変数宣言部分」、「デバイス該当判定部分」、「デバイス構造体部分」、「その

他の部分」の4つの部分に分割することが最適かどうかは再考の必要があり、場合によっては新しい分割基準を考案しなければならない。

- 他の関数やデバイス、OS への適応

本稿では、OS は Linux、デバイスは USB デバイス、関数を PROBE 関数に限定して、デバイスドライバのコードを分割することにより、Linux の全 USB デバイスの PROBE 関数をどのように書けばいいかという指針を示すことができた。今後、他の関数や他のデバイス、そして他の OS でもこのような指針を示したい。

- OS 依存仕様、デバイスドライバ仕様、デバイス依存仕様への抽象化

本研究では、デバイスドライバを OS 依存仕様、デバイスドライバ仕様、デバイス依存仕様の3つの仕様へ抽象化することを目指している。本稿では、デバイスドライバのコードを機能ごとに分割することによる抽象化を行った。この抽象度レベルを高めていくことによって、3つの仕様へ抽象化できると考えている。

- デバイスドライバ自動生成システムの実現

上述の3つの仕様へ抽象化できると、それを利用したデバイスドライバ自動生成システムが作成できる¹⁾。このシステムは、3つの仕様を入力とし、デバイスドライバのコードを出力するものである。このシステムが実現できると、デバイスドライバ開発者の負担を大きく減らすことができる。

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) Linux Online!: “<http://www.linux.org/>”.
- 3) USB Implementers Forum: “<http://www.usb.org/>”.
- 4) 鈴木一海, 五十嵐顕寿: “入門 USB”, 技術評論社 (2001).
- 5) CQ 出版社: “TECHI Vol.8, USB ハード&ソフト開発のすべて” (2001).
- 6) Detlef Fliegl: “Programming Guide for Linux USB Device”, <http://usb.cs.tum> (2000).
- 7) エクセルソフト WinDriver: “<http://www.xlsoft.com/jp/products/windriver/>”.