システムソフトウェアと
オペレーティング・システム 79－9
（1998. 8. 6）

デバイスドライバ生成システムにおける入力形式に関する考察

山下勝也　片山徹郎　最所圭三　福田晃

奈良先端科学技術大学院大学
情報科学研究科
〒 630-0101 奈良県生駒市高山町 8916-5
E-mail:{katuya-y,kat,sai,fukuda}@is.aist-nara.ac.jp

デバイスドライバは、対象となるデバイスとオペレーティングシステム (OS) 毎に用意しなければならない。また、作成に関しては、動作のタイミングなどのデバイスのハードウェアに関する知識が必要となるため、OS の他の部分に比べて最も時間と労力を必要とする。本研究では、デバイスドライバの仕様とデバイス依存仕様、OS 依存仕様を入力とし、デバイスドライバのソースコードを出力とするデバイスドライバ生成システムを提案する。入力であるそれぞれの仕様は、中間言語を用いて記述する。デバイスの例として、ネットワークデバイスを取り上げる。提案したシステムを用いて、デバイスドライバの一関数を生成することにより、システムの動作確認を行なった。

Definition of an Input Form for a Device Driver Generating System

Katsuya Yamashita　Tetsuro Katayama　Keizo Saisho　Akira Fukuda

Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5Takayama,Ikoma,Nara 630-0101,Japan

Writing device drivers is the most difficult one among processes to develop or port operating systems(OSs) because of requiring knowledge of device hardware. The device drivers are needded at each device and OS. In this paper, an device driver generating system is proposed, and it generates a source code of the device driver by specifications: the device driver specification, the device dependent specification, and the OS dependent specification. They are written in the intermediate language which we propose. The system generates a function, which behaves correctly, in a device driver of a network device.

1 はじめに

オペレーティングシステム (以下 OS) についての研究は、主にスケジューリングポリシーやメモリ管理、ファイルシステムの機能、構成法といった OS 自体の設計や性能に大きな影響を与える部分に対して行なわれている。これに対して、OS そのものを生成する方法は、 OS プログラムが大規模であり作成すること自体が困難であるため、ほとんど研究されていない。我々は、OS の自動生成の可能性を追求している [1]。

OS の中のカーネル部分は、デバイスドライバや割り込み処理などを含みハードウェアを直接操作する。このため、異なるアーキテクチャに OS を移植する際にはカーネルの大幅な書き換えが必要となる。ハードウェアに依存するカーネル部分の中でも、デバイスドライバの作成はもっとも時間と労力を要する。また、同じサービスを提供するデバイスでも、使用されているコントローラが異なれば、それに適合した新しいデバイスドライバを作成しなければならない。

デバイスドライバの作成は、デバイスのハードウェアや、開発の対象となる OS に関する知識に加えて、タイミング制御などの複雑で注意深いコーディングを必要とすることにより、多大な労力を要する。近年のマルチメディアやインターネットの発展を背景にした多様なデバイスの登場が予測される現在、以上のことが今後ますます深刻な問題となる。

本稿ではデバイスドライバ作成にかかる負担を軽減することを目的とする。デバイスドライバ生成システムを提案し、そのシステムの入力形式について考察する。また、ネットワークデバイスのデバイスドライバの一関数を取り上げて、システムの動作を確認する。以後、2 節では、デバイスドライバ生成システムの概要を述べる。3 節ではデバイスドライバ生成システムの入力形式について述べ、例としてネットワークデバイスを取り上げる。提案した独自の中間言語にしたがってネットワークデバイスの仕様を記述した例を示す。4 節では本稿で提案したデバイスドライバ生成システムとシステムの入力形式に対して考察を行なう。

2 デバイスドライバ生成システムの概要

我々は既にデバイスドライバを仕様から生成する処理系を提案し、その入力となるデバイスの仕様を形式的仕様記述言語である VDM-SL を用いて記述した [3]。文献 [3] では、デバイスドライバを生成するのに必要な仕様を以下の 3 つに分けていた。

- デバイスの仕様記述 (デバイスの種類毎に用意)
- デバイス固有の値を示したデータシート (デバイス毎に用意)
- OS 依存のライブラリ (OS 毎に用意)

ここでは、デバイスの種類が同じでも、デバイス上に搭載されているコントローラが変われば、コントローラの制御の仕方が変わる場合がある。その際はデータシートだけでなく、デバイスの仕様も書き直す必要があった。すなわち、同じ種類のデバイスであっても、デバイスが変われば、デバイスの仕様記述とデータシートは変わることがあり、上記の分け方が意味がない場合があった。

そこで、デバイスドライバを生成する処理系を、図 1のように新たに提案しなおす。

デバイスドライバの仕様とデバイス依存仕様、OS 依存仕様をデバイスドライバ生成

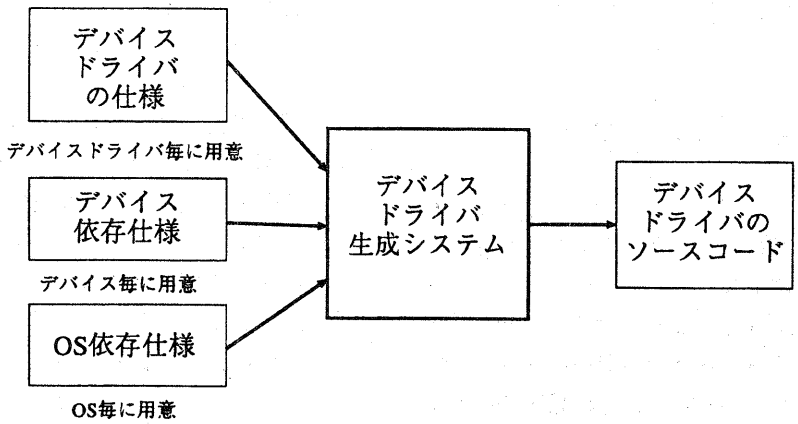

図 1: デバイスドライバ生成システムの概要

システムに入力することにより、仕様を満たすソースコードを生成する。それらの仕様は我々が提案する中間言語を用いて記述する。

- デバイスドライバの仕様

  生成するデバイスドライバがどのような関数を用意して、どんなデバイスを使用するか、デバイスドライバ内で使用するデータ構造や関数内のコードを記述する。

- デバイス依存仕様

  デバイスへの入出力動作やタイミング制御など、デバイスのハードウェアに依存する仕様を記述する。

- OS 依存仕様

  OS のデバイスドライバインターフェイスの定義を記述する。

以上の 3 つを利用してデバイスドライバのソースコードを生成する。これらの仕様記述と中間言語の詳細は、3 節で述べる。

このシステムを用いて、デバイスドライバを生成するためには、デバイスドライバの仕様とデバイスの仕様、OS 依存仕様を入力として与えればよい。図 2に、システムに与える 3 つの入力の関係を示す。また、デバイスドライバの仕様とデバイス依存仕様と切り離すことにより、デバイス依存仕様に関しては再利用が可能である。逆に、新規にデバイスが開発された場合は、そのデバイス依存仕様を中間言語を用いて記述し、それをそのまま用いることによりデバイスドライバを生成することが可能となる。

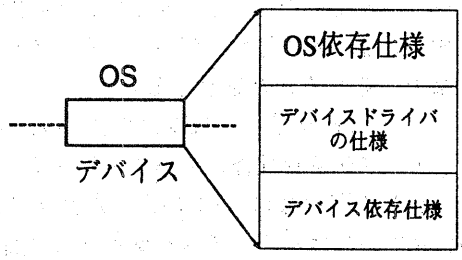

図 2: システムの入力の構造

このことにより、あらかじめデバイスの仕様が用意されているデバイスに対してデバイスドライバを作成する場合や、異なるアーキ

テクチャに移植する場合の時間と手間を大幅に削減することができる。

3 システムの入力形式

我々はデバイスドライバを生成する方法として、LOTOS や VDM-SL などの形式的記述言語を用いてプリンタデバイスの仕様を記述する試みを行なってきた[1][2][3]。デバイスドライバの仕様を記述するためには、ハードウェアを直接扱う機能が記述に使用する言語でサポートされている必要がある。しかし、これらの言語はそもそもデバイスの仕様を記述するために開発された言語ではないため、ハードウェアを直接操作する機能がサポートされていない。したがって、今回独自に中間言語を定義して、デバイスドライバの仕様を記述することを試みる。

デバイスドライバ生成システムの入力には、デバイス依存仕様とデバイスドライバ仕様、OS 依存仕様の 3 つの入力が必要である。中間言語は、デバイスドライバ内の関数の機能を持たせる際の効率の良さを考慮して、C 言語をベースとしている。以下にその概要を述べる。

また、以降 3Com 社の Etherlink III の FreeBSD[4] 用のデバイスドライバで使用する関数として, 構造が比較的簡単な stop 関数を例に挙げる。

3.1 デバイス依存仕様

デバイス依存仕様は、デバイスに依存した部分を記述する箇所である。このため、次の機能を記述する必要がある。

- デバイスとのデータの入出力動作
- タイミング制御
- ハードウェア割り込み
- OS とデータを受渡しするための機構

デバイスとのデータ入出力動作とタイミング制御を中間言語でサポートする。「ハードウェア割り込み」と「データを受渡しするための機構」とについては現段階ではサポートしていない。

デバイス依存仕様には、以下のようにデバイスに対する入出力動作及びタイミング制御を時間軸に沿って記述する。

- out8 /{port/} /{data/}
  port に 8 ビット の data を書き込む。
- out16 /{port/} /{data/}
  port に 16 ビット の data を書き込む。
- in8 /{port/} /{data/}
  port に 8 ビット の data を読み込む。
- in16 /{port/} /{data/}
  port に 16 ビット の data を読み込む。
- DELAY(Z)
  Z μs だけ待つ。

図 3に、デバイスドライバ依存仕様の記述例を示す。

3.2 OS 依存仕様

OS 依存仕様は OS に依存した部分を記述する箇所である。このため、次の機能を記述する必要がある。

- OS 側が要求するデバイスドライバインターフェイスの定義
- デバイスドライバに必要なヘッダファイル及びデータ構造の定義
- ハードウェアを直接制御する関数の定義

```
out16 /{ BASE + EP_COMMAND /} /{ RX_DISABLE /};
out16 /{ BASE + EP_COMMAND /} /{ RX_DISCARD_TOP_PACK /};
while (in16 /{ (BASE + EP_STATUS) & S_COMMAND_IN_PROGRESS /} );
out16 /{ BASE + EP_COMMAND /} /{ TX_DISABLE /}
out16 /{ BASE + EP_COMMAND /} /{ STOP_TRANSCEIVER /};
out16 /{ BASE + EP_COMMAND /} /{ RX_RESET /};
out16 /{ BASE + EP_COMMAND /} /{ TX_RESET /};
while (in16 /{ (BASE + EP_STATUS) & S_COMMAND_IN_PROGRESS /} );
out16 /{ BASE + EP_COMMAND /} /{ C_INTR_LATCH /};
out16 /{ BASE + EP_COMMAND /} /{ SET_RD_0_MASK /};
out16 /{ BASE + EP_COMMAND /} /{ SET_INTR_MASK /};
out16 /{ BASE + EP_COMMAND /} /{ SET_RX_FILTER /};
```

図 3: stop 関数のデバイス依存仕様の記述例

```
static void %<driver.name>_stop __P((struct %<driver.name>_softc *sc))
```

図 4: stop 関数の OS 依存仕様の記述例

上記の定義を以下のように中間言語で定義する。但し、現段階の実装の都合により、「デバイスドライバに必要なヘッダファイル及びデータ構造の定義」と「ハードウェアを直接制御する関数の定義」とについては、完全に記述できていない。

- 関数のプロトタイプを記述する。
- %<driver.name> デバイスドライバの名前 (driver.name) を埋め込む

図 4に、OS 依存仕様の記述例を示す。

3.3 デバイスドライバの仕様

デバイスドライバの仕様はデバイスドライバの構造を記述する。このため、以下の内容を記述する必要がある。

- デバイスドライバの名前の宣言
- OS 側が要求するデバイスドライバインターフェイスの宣言
- デバイスを制御する機能のデバイス依存仕様からの取り出し
- デバイスドライバ内の関数の機能を果たすコード
- デバイスドライバ内で使用するローカル変数及び構造体の宣言と定義

図 5に上記の機能を使ったデバイスドライバの仕様の記述の概略を示す。

図 6に、デバイスドライバ仕様の記述例を示す。

デバイスドライバ生成システムはデバイス依存仕様、OS 依存仕様、デバイスドライバの仕様の 3 つを入力とし、デバイスドライバのソースコードを出力する。図 7に、3 つの仕様 (図 3、4、6) を生成システムに入力した際に出力される stop 関数のコードを示す。

```
driver_name "foo"

/* デバイスドライバの名前を */
/* foo に定義する。*/

%<hogehoge> {

/* OS 側で要求されるデバイスドライバ */
/* の関数を定義する                  */

    if (sc->gone) {
     return;
    }

%<chip1.hogehoge>

/* chip1 というコントローラの */
/* hogehoge という機能を果たす*/
/* 命令シーケンスを呼び出す */

}
```

図 5: デバイスドライバの仕様記述の例

4 考察

この節では、中間言語やデバイスの仕様書、デバイスドライバ生成システムの有効性について考察を行なう。

4.1 中間言語について

デバイスドライバ生成システムの入力として、形式的仕様記述言語を用いることにより、デバイスドライバ生成過程におけるバグの混入を防ぐことができる。しかし、LOTOS や VDM-SL をはじめとする言語は、デバイスの仕様を記述することを念頭において開発されたわけではない。デバイスの仕様を記述するために、より適した言語を用いることによって、デバイスドライバの生成が容易になると考えられる。今回は独自の中間言語を定めて仕様を記述した。

デバイス依存仕様はコントローラを操作

```
driver_name "ep"

%<stop>{
if (sc->gone) {
    return;
}

%<ep.stop>
}
```

図 6: stop 関数のデバイスドライバの仕様の記述例

する命令シーケンスから構成されている。コントローラを操作する命令シーケンスのインターフェイスに、引数及び戻り値を明示的に設定できない。つまり、コントローラの状態を変更する操作しかできない。

現時点では、デバイス依存仕様でコントローラを操作した際に得られた戻り値を、デバイスドライバの仕様で利用できない。たとえば、イーサネットコントローラを操作してイーサネットフレームを読み書きする部分はデバイス依存仕様に該当するが、そこから読み書きしたイーサネットフレームを OS に渡す時に、戻り値を渡せないためにグローバル変数にデータを格納して渡すしかない。これでは、デバイスドライバの仕様の部分がデバイス依存仕様の内容に依存することになる。これでは、デバイスドライバからデバイスに依存する部分を切り離す目的を達成できない。

本稿では、関数内にはデバイスへの命令シーケンスが一つしかない stop 関数を例に取り上げ生成を行なった。しかし、デバイスドライバで定義される関数内でコントローラを操作する一連の命令シーケンスが、いくつか別れていて、かつ独立している場合が考えられる。その場合には我々が定義した中間言語の変更、及び拡張を検討しなければなら

```
static void ep_stop __P((struct ep_softc *sc)){

  if (sc->gone) {
      return;
  }

  outw(BASE + EP_COMMA, RX_DISAB);
  outw(BASE + EP_COMMA, RX_DISCARD_TOP_PA);
  while ( inw((BASE + EP_STATUS) & S_COMMAND_IN_PROGRE) );
  outw(BASE + EP_COMMA, TX_DISAB)
  outw(BASE + EP_COMMA, STOP_TRANSCEIV);
  outw(BASE + EP_COMMA, RX_RES);
  outw(BASE + EP_COMMA, TX_RES);
  while ( inw((BASE + EP_STATUS) & S_COMMAND_IN_PROGRE) );
  outw(BASE + EP_COMMA, C_INTR_LAT);
  outw(BASE + EP_COMMA, SET_RD_0_MA);
  outw(BASE + EP_COMMA, SET_INTR_MA);
  outw(BASE + EP_COMMA, SET_RX_FILT);
}
```

図 7: 仕様からデバイスドライバ生成システムを用いて出力したソースコード (stop 関数)

ない。

4.2 デバイスの仕様書について

現段階のシステムでは、デバイス依存仕様は既存のデバイスドライバのソースコードから、データや関数を参考に手作業で行なっている。このため、新規のデバイスに対して開発する際には、そのデバイスの仕様書からデバイス依存仕様を書くことになる。従来の手法と変わらないので馴染みやすいという利点はあるが、仕様書を参照してデバイス依存仕様を作成するため、従来と同じ程度手間がかかる。

現在メーカーが配布しているコントローラの仕様は、タイミングチャートとコントローラに内蔵されているレジスタマップ及びレジスタにセットする値を載せていることが多い[5][6]。今後は、現在配布されているコントローラの仕様の形式から、直接デバイスドライバ生成システムの入力形式に変換、または入力形式の作成を支援できるようなシステムを検討しなければならない。

4.3 デバイスドライバ生成システムの有効性について

デバイスドライバ生成システムは入力として 3 つの仕様を必要とする。このうち、デバイス開発者がデバイス依存仕様を用意し、OS 開発者が OS 依存仕様を用意すれば、デバイスドライバを作成するためにはデバイスドライバの仕様を書くだけで良い。このように、デバイスドライバ作成を 3 者で分担することで、作成にかかる負担を軽減することが可能である。

また、デバイス依存仕様と OS 依存仕様は再利用が可能である。このため、あらかじめ用意されたデバイス依存仕様を用いて、あるデバイスから別の OS に対するデバイスドライバを作成することができる。更に、OS 依存仕様を用いて、同じ OS 上の異なるデ

バイスのデバイスドライバを作成することができる。以上のことから、従来よりデバイスドライバを作成する際の時間及び手順が、削減できる。

今回例として生成した stop 関数を実際にデバイスドライバに組み込み、正しく動作することを確認した。

5 おわりに

本稿では、デバイスドライバ生成システムを提案し、システムの入力形式について述べた。システムは、デバイスドライバの仕様、デバイス依存仕様、OS 依存仕様の 3 つを入力とし、その 3 つを満たすデバイスドライバのソースコードを出力する。言語には我々が独自に定めた中間言語を用いた。この中間言語を用いて実際に Etherlink III の FreeBSD のデバイスドライバ内の stop 関数の仕様を記述し、そのソースコードを出力した。また、生成されたソースコードを実際にデバイスドライバに組み込んで、正しく動作することを確認した。

今後の課題として以下の項目が考えられる。

- 他のデバイスにおける同一関数への適用

  本稿では、OS を FreeBSD、デバイスを 3Com 社の Etherlink III、デバイスドライバの関数を stop 関数に絞って記述した。他のネットワークデバイスの stop 関数に関して、同様に記述できるか検討する必要がある。

- デバイスの仕様について

  今回示した仕様記述の例は、既存のデバイスドライバを参照して書き出した。これは、デバイスの仕様書を入手することが困難だからである。しかし、Web ページからイーサネットコントローラの仕様を入手できるところもある。このようなデバイスに関しては、その仕様をシステムの入力に用いることを考えている。また、その仕様をデバイスドライバ生成システムの入力形式に変換するシステムも作成可能であるかどうか検討する。

- 多種多様なデバイス及び OS のサポート

  今回、OS は FreeBSD、デバイスは 3Com 社の Etherlink III に絞り、検討を行なった。将来的にはデバイスドライバ生成システムをより多くのデバイス、及び OS に対応させ、本手法の有効性を確認する。

参考文献

[1] 長尾周司, 片山徹郎, 張漢明, 最所圭三, 福田晃: "OS の自動生成に向けて," 情報処理学会研究報告, 96-OS-73, pp.103-108, 1996.

[2] 長尾周司, 片山徹郎, 最所圭三, 福田晃: "デバイスドライバの自動生成に向けて – デバイスドライバの定式化 –," 情報処理学会研究報告, 97-OS-74, pp.177-182, 1997.

[3] 片山徹郎, 最所圭三, 福田晃: "デバイスドライバの自動生成に向けて –プリンタデバイスの生成に関する考察–," 情報処理学会研究報告, 97-OS-76, pp.43-48, 1997.

[4] FreeBSD Inc: http://www.freebsd.org/

[5] Intel 82595TX ISA/PCMCIA High Integration Ethernet Controller: http://www.intel.com/design/network/datashts/281630.htm

[6] TDK Fast Ethernet PHY for 100BASE-TX/10BASE-T, MII, Auto Negot., filters: http://www.tsc.tdk.com/lan/78Q2120.pdf