システムソフトウェアと
オペレーティング・システム
(1997. 8. 21)
76－8

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–

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
〒630-01 奈良県生駒市高山町 8916-5
E-mail:{kat, sai, fukuda}@is.aist-nara.ac.jp

本研究では、オペレーティング・システム (OS) の自動生成を追求する。OS は種々の要素から構成される。その中でデバイスドライバは、ハードウェアごとに用意しなければならない、また、動作のタイミングなどのハードウェアに関する知識が必要である、などの理由により、他の部分に比べ記述に最も時間と労力を必要とする。本稿では、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について検討する。デバイスの例として、プリンタデバイスを取り上げ、既存のデバイスドライバから、デバイスドライバの基本機能を洗い出し、その抽象化を行なう。同時に、デバイスに固有の値を抽出し、仕様記述言語 VDM-SL を用いて、プリンタデバイスの仕様を記述する。デバイスの仕様を記述することにより、デバイスドライバを作成する際にかかる時間および手間を削減する。

Toward Automatic Generation of Device Drivers
–Method for Generating Printer Devices–

Tetsuro Katayama　　Keizo Saisho　　Akira Fukuda

Graduate School of Information Science,
Nara Instiitute of Science and Technology
8916-5, Takayama-cho, Ikoma-shi, Nara 630-01, Japan

This research investigates the possibility of operating systems(OS)' automatically generating. Writing device drivers is the most difficult one among processes to develop or port OS. In this paper, an automatically device driver generation system is proposed, and its input is examined. Fundamental functions of device drivers are extracted and peculiar values of devices are abstracted from existing device drivers. As an example, a specification for a printer device is written in VDM-SL, which is one of specification languages. By describing a specification for devices clearly, device drivers can be generated more easily. Consequently, time and labor in generating the device drivers decrease.

1 はじめに

オペレーティング・システム (以下 OS) の研究の多くは、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の設計や性能に大きな影響を与える部分に集中しており、OSそのものを生成する方法はほとんど研究されていない [1]。OSプログラムは大規模プログラムであるので、作成、修正、移植などが困難である。このため、OSプログラムの自動生成はOS研究者の夢であると言っていい。我々は、OSの自動生成の可能性を追求している [2][3]。

OSの核にあたるカーネル部分は、デバイスドライバや割り込み処理などを含んでおり、ハードウェアを直接操作する。このため、OSを異なるアーキテクチャに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部分の中でも、最も時間と労力がかかるのはデバイスドライバの部分である。新しいデバイスに対応するために、OS内のデバイスドライバを修正・追加するには、デバイスのハードウェアの知識に加えて、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ため、多大な労力を要する。また、同じサービスを提供するデバイスでも、使用されているチップ (コントローラ) が異なれば、それに合った新たなデバイスドライバを記述しなければならない。マルチメディアやインターネット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。

本研究は、コーディングが最も困難であるデバイスドライバの自動生成に対象を絞り、デバイスの仕様記述から、その仕様を満たすデバイスドライバのソースコードの自動生成を目的とする。本稿では、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について述べる。以後、2章では、デバイスドライバ生成システムの概要について述べる。3章では、デバイスの例として、プリンタデバイスを取り上げる。デバイスドライバの基本機能を洗い出し、抽象化を行なうことにより、デバイスの仕様の記述例を示す。記述には、近年、ヨーロッパで注目されている仕様記述言語 VDM-SL[4](The Vienna Development Method Specification Language) を用いる。4章では、議論および考察を行なう。

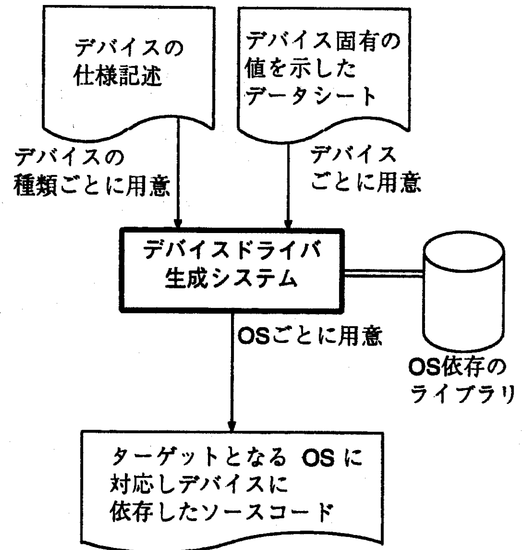

図 1: デバイスドライバ生成システムの概要

2 システムの概要

我々は、OSの自動生成の可能性を追求している [2][3]。OSの生成に対する入力として、OSの仕様記述を考えている。これは、仕様記述言語を用いることにより、生成過程におけるバグの混入を防ぐことができるためである。ハードウェアに依存しないファイルシステムやスケジューリングなどは、入力となる仕様を一旦記述すれば、OSを異なるアーキテクチャに移植する場合でも、記述した仕様自体を特に変更する必要はない。しかしながら、ハードウェアに依存するデバイスドライバは、デバイスに合わせて、仕様を記述しなければならないため、移植する場合には変更しなければならない。そこで、デバイスドライバを生成する処理系として、図1の処理系を提案する。

デバイスの仕様記述とデバイスに固有の値を示したデータシートとを、デバイスドライバ生成シ

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。ここで、デバイスの仕様記述はデバイスの種類に依存し、データシートはードウェアに依存する。また、生成システムはターゲットとするOSごとに用意する。システムのデータとしてOSに依存したライブラリが存在するものとし、これを利用してデバイスドライバを生成する。ここで、ライブラリは、OSによってデバイスドライバの呼び出し方が異なるので、引数の与え方などの情報を与える。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、デバイスの仕様記述とデバイスに固有の値とを入力として与えればよいため、デバイスドライバを作成する際、および、異なるアーキテクチャに移植する際の、時間および手間を大きく削減できる。

次章では、生成システムの入力となる、デバイスの仕様記述について検討する。

3 仕様記述言語によるデバイスの記述

3.1 仕様記述言語 VDM-SL

デバイスの仕様記述に使用する言語として、今回はVDM-SL[4]を用いた。VDM-SLはVDMの仕様記述言語部分であり、データ型から構成したモデルの状態を変化させる仕様が記述可能な、モデル指向仕様記述言語である。データ型は数値、文字列などの基本型や、集合、マップなどの合成型を持ち、このデータ型により生成した値を、関数と操作で変化させる記述を行なう。また、VDM-SLで記述された仕様は、プログラミング言語C++のソースコードに変換可能であるため、容易に実行が可能となる。

次節では、デバイスの仕様の簡単な記述例として、プリンタデバイスを取り上げ、VDM-SLを用いて記述した。

3.2 仕様記述言語によるデバイスの仕様の記述例

今回は一例として、デバイスを記述する仕様の量が比較的少ない、プリンタデバイスを例にとる。MINIX[5], FreeBSD[6], NetBSD[7]を参考にして、プリンタドライバを生成するために必要なプリンタデバイスでの基本的な操作を抽出した。プリンタデバイスにおける操作には、表1に示す項目がある。

表1: プリンタデバイスにおける操作

操作名	操作内容
init	デバイスの初期化を行なう
oepn	デバイスを使用可能にする
close	デバイスを使用不可能にする
write	デバイスに値を書き込む
cancel	書き込み中の処理を途中で取り消す
error	エラー処理を行なう

また、同様に、プリンタデバイスに固有の値としては表2に示す項目を抽出した。

VDM-SLでの記述において、状態は大域変数を用いて記述し、状態を変化させるような仕様を操作として記述した。これは、ポート、ステータス、コマンドなどの値を大域変数に代入しておくことにより、どの操作からも利用できるからである。プロセスの実行は関数・操作を呼び出すことによって行なうため、プロセスの時間的な順序関係は呼び出された関数・操作の順番によって記述した。

表1, 2をもとにして、プリンタデバイスの仕様を記述した。図2はVDM-SLを用いたプリンタデバイスの記述例である(言語の詳細については文献[8]を参照)。図2の仕様には、一部にVDM-SLの構文構造からは曖昧になっている表記が存在するが、ここでは詳しく記述しない。

図3は、図2の仕様をC++に変換したものの一部(Init操作部)である。

デバイスドライバ生成システムでは、このあと、仕様から生成したC++のコードに対して、OSに対応したライブラリを参考にして、ターゲットとなるOSのデバイスドライバのソースコードを生成する。現段階において、一部分のコードを自動的に生成することはできている。しかしながら、引数や変数名の整合性などの細かい箇所は依

表 2: プリンタデバイスにおけるデバイス固有の値

名前	働き
データレジスタポート	印刷するデータなどをこのポートに出力する
ステータスレジスタポート	プリンタの状態をこのポートを通して読み書きする
コントロールレジスタポート	コントロールコードをこのポートに出力する
コントロールコード	プリンタを制御するコマンドを表す
ステータスコード	プリンタの状態に応じた値を持つ
IRQ	割り込み番号を表す

```
values
  Data_Port     = 0x3BC;
  Status_Port   = 0x3BD;
  Control_Port  = 0x3BE;

  assert_strobe_Command = 0x1D;
  negate_strobe_Command = 0x1C;
  select_Command        = 0x0C;
  init_printer_Command  = 0x08;

  busy_Status    = 0x10;
  nopaper_Status = 0x20;
  normal_Status  = 0x90;
  online_Status  = 0x10;

  printer_IRQ = 7

operations
  Print_drv() ==
  (
    Init();
    while true do
      ||(Open(),Close(),Write(),
          Cancel(),Error())
  );

  Init() ==
  (
    out_byte(Control_Port,
             init_printer_Command);
    out_byte(Control_Port,
             select_Command);
    put_irq_handler(printer_IRQ,
                    Print_handler)
  )
  ext rw status
  pre status = online_Status;
```

```
Open() ==
ext rw status
post status = online_Status;

Close() ==
ext rw status
post status = not online_Status;

Write() ==
(
  if (data_count = 0) then
    out_byte(Control_Port,
             select_Command)

  out_byte(Data_Port, data);
  out_byte(Control_Port,
           assert_strobe_Command);
  out_byte(Control_Port,
           negate_strobe_Command);
)
ext rw status;
    ro data_count
pre data_count >= 0 and
    status = normal_Status;

Cancel() ==
ext rw data_count
post data_count = 0;

Error() ==
  case status :
    busy_Status -> retry(),
    not online_Status and nopaper_Status
      -> paper_empty(),
    not online_Status
      -> out_byte(Control_Port,
                  select_Command)
```

図 2: 仕様記述言語 VDM-SL によるプリンタデバイスの記述例

然手作業で行なっており、今後、自動化しなければならない。

4 議論および考察

この章では、本研究のアプローチに対する議論および考察を行なう。

4.1 仕様記述言語について

デバイスドライバ生成システムの入力として、仕様記述言語を用いることにより、デバイスの生成過程におけるバグの混入を防ぐことができる。しかしながら、仕様記述言語は、一般に言語として記述の自由度が少ない。VDM-SL はデータの型は扱いやすいが、プロセスの実行は関数・操作

```
void vdm_DefaultMod_Init() {
  PushFile("printer.vdm");
  {
    PushPosInfo(34, 3);
    {
      PushPosInfo(35, 13);
      Int tmpVar_14;
      tmpVar_14 =
        vdm_DefaultMod_Control__Port;
      Int tmpVar_15;
      tmpVar_15 =
        vdm_DefaultMod_init__printer__Command;
      vdm_DefaultMod_out__byte
        (tmpVar_14, tmpVar_15);
      PopPosInfo();
    }
    {
      PushPosInfo(37, 13);
      Int tmpVar_16;
      tmpVar_16 =
        vdm_DefaultMod_Control__Port;
      Int tmpVar_17;
      tmpVar_17 =
        vdm_DefaultMod_select__Command;
      vdm_DefaultMod_out__byte
        (tmpVar_16, tmpVar_17);
      PopPosInfo();
    }
    {
      PushPosInfo(39, 20);
      Int tmpVar_18;
      tmpVar_18 =
        vdm_DefaultMod_IRQ;
      Int tmpVar_19;
      tmpVar_19 =
        vdm_DefaultMod_print__handler;
      vdm_DefaultMod_put__irq__handler
        (tmpVar_18, tmpVar_19);
      PopPosInfo();
    }
    PopPosInfo();
  }
  PopFile();
}
Bool vdm_DefaultMod_pre__Init() {
  PushFile("printer.vdm");
  PopFile();
  return (Bool)
    (vdm_DefaultMod_status ==
     vdm_DefaultMod_online__Status);
}
```

図 3: VDM-SL によるプリンタデバイスを C++ に変換したもの (一部)

を呼び出すことによって行なうため、プロセスの時間的な順序関係は呼び出された関数・操作の順番により、記述しなければならない。

OSI(Open System Interconnection) 標準の仕様を記述するための形式記述技法として開発された言語の一つに LOTOS がある [9]。LOTOS には抽象データ型しか存在しないので、定数は演算から作らなければならなくなり、値の引渡しはプロセス間の相互作用として行なわれる。従って、LOTOS でプログラミング言語の状態変数に相当するものを作ることは難しい。

今回は、仕様記述言語として VDM-SL を用いた。しかしながら、ポート番号などの定数を宣言できる機能と、プロセスの時間的な順序関係を記述できる機能とを共に備えている言語があれば、デバイスの仕様を記述する作業はかなり容易になると考えられる。

4.2 I_2O について

I_2O (Intelligent Input Output) SIG[10] において、OS と デバイスドライバとの間に標準のインターフェイス I_2O を定めている。I_2O の仕様では、デバイスドライバを、次の 3 つの階層に分ける。OS に依存した OSM (OS Specific Module)、ハードウェアに依存した HDM (Hardware Device Module)、OSM と HDM との間で情報を送受する Messenger である。OSM と HDM 間の通信に使うパケットの形式を明確に定義することで、OS が異なっても一つのデバイスに対して、HDM 自体を書き変えることなく通信が可能となる。我々の生成システムで I_2O を利用した場合、ターゲットとなる OS ごとに、デバイスドライバ生成システムを用意する必要がなくなるので、システム自体の開発が非常に容易になる。しかしながら、I_2O 自体の問題として、デバイスドライバを階層構造にしたことによるオーバーヘッドがある。特に高速なデバイスにおいてこの問題が致命的になることも考えられる。このため、I_2O を採用するかどうかについては、まだ決定していない。

4.3 チップの仕様について

本稿では、生成システムの入力を作成する手段として、デバイス固有の値に関しても、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して作成した。すなわち、リバース・エンジニアリング的な手法をとった。本来ならば、チップの仕様を、生成システムの入力として用いるべきである。これは、現状において、チップの

仕様を入手することが困難であることが多く、また、特許などの関係で入手が不可能な場合もあるためである。

4.4 HDL への対応について

我々の生成システムでは、デバイス固有の値、および、デバイスの仕様を入力として明示的に与える必要がある。ハードウェア記述言語 HDL (Hardware Description Language) で記述・設計されたデバイスに対しては、その記述自体から、自動的に、生成システムの入力を抽出することが可能かもしれない。その場合には、HDL の記述からデバイスドライバを自動的に生成するシステムの構築が可能になると考えられる。

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、システムの入力について述べた。既存のデバイスドライバから、基本機能を洗い出し、抽象化を行なった。生成システムの入力となる、デバイスの仕様には、仕様記述言語 VDM-SL を用いた。例として、プリンタデバイスの仕様を記述した。

今後の課題として、以下の項目がある。

- 標準インターフェイス I_2O の採用の検討

  I_2O SIG において、OS と デバイスドライバとの間に標準のインターフェイス I_2O を定めている。I_2O の仕様では、OS が異なっても通信が可能となる。しかしながら、性能の問題があるため、我々の生成システムで利用するかどうかについては、今後検討しなければならない。

- チップの仕様の入手

  今回の例では、既存のソースコードからデータや関数を参考にすることによって、システムの入力を作成するという、リバース・エンジニアリング的な手法をとった。最近では、ネットワーク関連のチップなどデータシートを公開する所が増えてきており、今後は、入力としてそれらを利用する予定である。

- HDL による記述への対応

  生成システムでは、デバイス固有の値、および、デバイスの仕様を入力として明示的に与える必要がある。将来的には、HDL で記述・設計されたデバイスに対して、その記述から自動的にデバイスドライバを生成可能なシステムの構築を目指す。

- デバイスドライバ生成システムの自動化

  現段階のシステムでは、最終的なコード生成の段階を、手作業に頼っている。今後、なんらかの方法によって、自動化する必要がある。また今回は、プリンタデバイスという比較的仕様の記述量が少ないデバイスを例にとった。今後、様々なデバイスに適用する予定である。

参考文献

[1] Xiaohua Jia and Mamoru Maekawa: "Operating System Kernel Automatic Construction," *Operating Systems Review*, Vol.29, No3, pp.91–96, 1995.

[2] 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.

[3] 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.

[4] Cliff B. Jones: "Systematic Software Development using VDM," *Prentice Hall*, 1990.

[5] Andrew S. Tanenbaum: "MINIX オペレーティングシステム," アスキー出版局, 1989.

[6] FreeBSD ハンドブック: http://www.freebsd.org/ja_JP.EUC/handbook/

[7] NetBSD Project : http://www.netbsd.org/

[8] The VDM–SL Tool Group and The Institute of Applied Computer Science: "The IFAD VDM-SL Language," 1996.

[9] 高橋薫, 神長裕明: "仕様記述言語 LOTOS," カットシステム, 1995.

[10] I_2O Special Interesting Group(SIG): http://www.i2osig.org/