システムソフトウェアと オペレーティング・システム 74−31
マルチメディア通信と分散処理 81−31
（1997. 2. 28）

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 −

長尾 周司　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
e-mail:{syuuzi-n, kat, sai, fukuda}@is.aist-nara.ac.jp

デバイスドライバの作成は，アーキテクチャに大きく依存するために困難な作業である．デバイスドライバを自動的に生成する事ができれば，オペレーティングシステム (OS) を作成または移植する際の手間を大きく削減できる．本研究では，デバイスドライバを自動的に生成するために，まずデバイスの仕様を定式化し，その仕様を満たすデバイスドライバのソースコードを自動的に生成する事を目指している．本稿では，FreeBSD と NetBSD とを参考にし、デバイスの一例として SCSI デバイスを取り上げる．SCSI デバイスを制御するために必要な SCSI プロトコルと SCSI ホストアダプタの二つのデバイスドライバについて，デバイスの仕様を定式化する．

Toward Automatic Generation of Device Drivers
− Formalization for Device Drivers −

Shuuji Nagao　Tetsuro Katayama　Keizo Saisho　Akira Fukuda

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

We would like to generate device drivers automatically, because writing device drivers, which depends on target machine architecture, is one of difficult things in creating an operating system(OS). Automatic generation of device drivers could allow us to easily create and port an OS. In order to automatically generate device drivers, we firstly formalize device specifications, and then source codes for a device driver satisfing them is generated automatically. This paper describes methods of formalizing device specifications. We adopt SCSI devices of FreeBSD and NetBSD as an example, and formalize the device specifications from both device drivers of SCSI protocols and SCSI host adapters which control the SCSI devices.

1 はじめに

現在，多種多様なオペレーティングシステム (OS) が使われている．しかしながら，マルチプラットフォームで使われているOSの数は少ない．異なるアーキテクチャにOSを移植する際に，アーキテクチャに依存している部分のソースコードを書き換えることが必要となり，その作業は現在のところ人がやらなければならないため，もっとも困難な作業だからである．これまでのOSについての研究の主な事柄は，スケジューリングポリシ，メモリ管理，ファイルシステムの機能，OSの構成法といったOS自体の設計や性能に大きな影響を与える部分に集中している．一方，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，変更等が困難であり，OSプログラムの自動生成はOS研究者の夢である．しかし，OS自体を何らかの情報から生成することは極めて困難なために，研究されていなかった [1].

カーネルはデバイスドライバや割り込み処理などを含んでいるため，ハードウェアを直接操作する必要がある．異なるデバイスを使用する際には，同じ機能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．また，異なるアーキテクチャにOSを移植する際には，ソースコードの大幅な書き換えを必要とする．ハードウェアに依存した部分のソースコードの作成は，コーディングだけでなく，デバッグも困難である．

デバイスの仕様から自動的にデバイスドライバを生成するシステムを作成すれば，コーディングとデバッグにかかる時間を大幅に削減する事が可能になる．本論文ではデバイスドライバを自動的に生成するための入力となる，デバイスの仕様を定式化する手法を検討する．

2 システムの概要

我々は，デバイスドライバを生成する処理系として図1に示す処理系を想定している [2]．デバイスの仕様記述をデバイスドライバ生成システムに入力する事により，仕様を満たすソースコードを生成する．このシステムによっ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さえ入力として与えればよい．このため，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と手間の削減を実現できる．

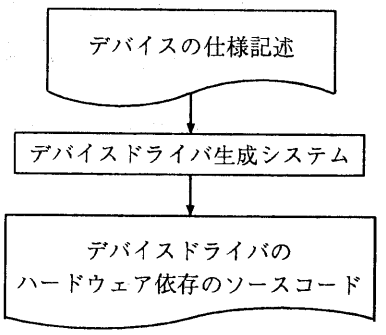


図 1: 生成処理系の概要

そこで次章では，入力として必要なデバイスの仕様記述を決定するために，デバイスの仕様を定式化する事を試みる．

3 デバイスドライバの定式化

3.1 既存のOSのデバイスドライバ

本研究ではFreeBSD，NetBSDの2つのOSを参考にしてデバイスドライバの定式化を試みる．

3.1.1 FreeBSD

カリフォルニア大学バークレイ校のComputer Systems Research Group(CSRG) による4.4BSD-Liteにi386アーキテクチャに必要な部分を加えて作られたOSである [3]．i386のアーキテクチャに限定した開発が行なわれている．このため，アーキテクチャに依存した部分が理解しやすい．また，サポートされているデバイスの数が多い．これらの理由により，FreeBSDを参考にした．具体的にはFreeBSD 2.1.6.1-RELEASEのSCSIホストアダプタのデバイスドライバのソースコードを参考にしている．

3.1.2 NetBSD

FreeBSDと同様に4.4BSD-Liteに各アーキテクチャに必要な部分を加えて作られたOSである．現

表 1: NetBSD でサポートされているアーキテクチャ

DEC Alpha
Acorn RiscPC
Atari
HP 9000/300 series
i386-family PC
m68k-based Macintosh
Motorola m68k MVME boards
PC532
Acer Pica
DECstation
PowerPC systems with OpenFirmware
Sun SPARC series
Sun 3
DEC VAX
Sharp X680x0

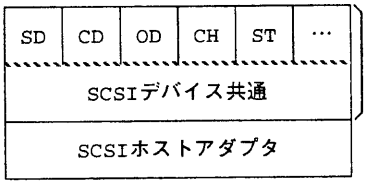
在，表 1に示すアーキテクチャのマシンがサポートされている [4].

NetBSD はマルチプラットフォームで動かす場合を考慮して作られているため，FreeBSD と比べてより一層アーキテクチャに依存しない形でソースコードが記述されている．逆に i386 アーキテクチャ特有のデバイスに関しては，FreeBSD の方が多くサポートされている．SCSI プロトコルのデバイスドライバは，サポートされているデバイスの数は FreeBSD の方が多い．しかしながら，処理の流れは NetBSD の方が統一されているので，SCSI プロトコルのデバイスドライバは NetBSD のものを参考にした．参考にしたのは NetBSD 1.2 の SCSI プロトコルに関する部分のデバイスドライバのソースコードである．

3.2 SCSI ドライバの定式化

今回，SCSI 装置に関するデバイスドライバの定式化を試みた．SCSI 装置に対するデバイスドライバは，次の 2 つの部分に分かれている．

- SCSI ホストアダプタを制御するためのドライバ．
- SCSI プロトコルに従ってデバイスを制御するためのドライバ．



SD: ダイレクトアクセスデバイス
CD: CD-ROMデバイス
OD: オプティカルデバイス
CH: メディアチェンジャデバイス
ST: シーケンシャルアクセスデバイス

図 2: SCSI デバイスドライバの構造

この SCSI デバイスドライバの構造を図 2に示す．SCSI 規格の部分をさらに 2 つに分けているのは，SCSI デバイスに共通な部分と，デバイスの種類によって異なる部分の 2 つに分かれているからである．

SCSI ホストアダプタを制御するためのドライバは，使われているコントローラが異なった場合に，新しいドライバを記述しなければならない．またアーキテクチャの異なるマシンに移植する際には，記述しなおす必要のある部分がでてくる．

一方，SCSI プロトコルに従ってデバイスを制御するためのドライバは，SCSI で決められているデバイスの種類に応じてドライバを記述する必要がある．ベンダ固有の特殊なデータ構造を採用しない限り，一度ドライバのソースコードを記述してしまえば，デバイスを他のメーカー製に取り替えても新たに記述する必要はない．さらに，SCSI プロトコルのデバイスドライバはアーキテクチャにも依存しないため，移植する際にもそのまま使用することができる．

3.2.1 SCSI プロトコルの定式化

SCSI プロトコルに関しては，図 2で示したように SCSI デバイスの種類に応じた処理と，SCSI デバイスに共通な処理の二つに分けられる．ダイレクトアクセスデバイスと CD-ROM デバイスにおいて次に挙げる処理は，表示するディスク情報の内容が違うことを除くと，NetBSD の SCSI プロトコルのドライバが行う処理の内容は一致している．

- match
  接続しているデバイスの種類を検索する.
- attach
  ディスクの構造体の設定等を行なう.
- open
  デバイスをオープンする.
- close
  デバイスをクローズする.
- strategy
  転送するデータをチェックしてstartを呼び出す.
- start
  実際にデータを転送する命令を発行する.
- read
  RAWデバイスの読み込みを行なう.
- write
  RAWデバイスの書き込みを行なう.

ダイレクトアクセスデバイスとメディアチェンジャデバイスにおけるattachの処理の流れを比較する.

- ダイレクトアクセスデバイスのattachの処理の流れは，以下となる.
  1. ドライバに接続するために必要な情報を設定.
  2. ディスクの構造体の初期化.
  3. 古いデバイスかどうかをチェック.
  4. ディスク情報を表示.
- メディアチェンジャデバイスのattachの処理の流れは，以下となる.
  1. ドライバに接続するために必要な情報を設定.
  2. ディスク情報を表示.

ダイレクトアクセスデバイスとメディアチェンジャデバイスにおけるopenの処理の流れを比較する.

- ダイレクトアクセスデバイスのopenの処理の流れは，以下となる.
  1. “test unit ready” SCSIコマンドを発行.
  2. “start stop unit” SCSIコマンドのSTARTを発行.
  3. “prevent allow medium removal” SCSIコマンドをPREVENTで発行.
  4. パーティションをチェック.
  5. デバイスがcharacter型かblock型をチェック.
- メディアチェンジャデバイスのopenの処理の流れは，以下となる.
  1. “test unit ready” SCSIコマンドを発行.
  2. ディスクパラメータを取得.

ダイレクトアクセスデバイスとメディアチャンジャデバイスにおけるattachの処理の流れを比較した場合，いくつか同じコマンドが発行されている．openの処理の流れを比較した場合も同様の事が言える．しかしながら，定式化を行なうためにダイレクトアクセスデバイスとメディアチェンジャデバイスのattachとopenをそれぞれ一つにまとめて抽象化することは，かなり困難である．抽象化をするためには，attachとは何であるか，openとは何であるかという情報を，それぞれ論理式または数式で表す必要がでてくるためである.

3.2.2 SCSIホストアダプタの定式化

SCSIホストアダプタを制御するデバイスドライバを定式化するために各関数の抽象化を試みる．Adaptecのaha154xシリーズとUltrastorのultrastor14fにおける同じ機能の関数を表2に示す.

表2において，registerdev，attachは共にカーネルがデバイスを管理するために呼び出す関数である．initはprobeから呼ばれてアダプタの初期化，IRQ，DMAの設定を行なう．probeはinitに渡すデータを用意しておく関数である．intrはアダプタからの割り込みが起こった際に処理する関数である．scsi_cmdはSCSIコマンドを発行する関数である．それ以外の関数の説明は省略する.

- registerdev，attach
  registerdev，attachはカーネルが使う構造体に値を代入するだけなので，一つにまとめる事ができる.

表 2: SCSI ホストアダプタの同じ機能の関数

ADAPTEC 製 aha154x	Ultrastor 製 ultrastor14f
aha_registerdev	uha_registerdev
ahaattach	uha_attach
ahaprobe	uhaprobe
aha_init	uha_init
aha_scsi_cmd	uha_scsi_cmd
aha_poll	uha_poll
ahaminphys	uhaminphys
aha_done	uha_done
ahaintr	uhaintr
aha_timeout	uha_timeout
aha_free_ccb	uha_free_mscp
aha_get_ccb	uha_get_mscp

- probe

  probe で行なわれている処理の流れは次のように抽象化できる.

  1. init が使うデータ構造を用意する.
  2. init を呼び出す.
  3. init で得られた値のうちカーネルの他の部分が必要とするデータを与える.

- init

  ahainit, uhainit の処理の流れをそれぞれ図 3, 図 4で示す.

  init の処理は

  1. ボードの初期化に必要なデータを設定.
  2. ボードをリセット.
  3. アーキテクチャ依存のデータを取得.
  4. ボードに依存する処理.

  の 4 つの処理からなる.

  図ではボードに依存する処理がいくつかに分割されているが, これらは一つにまとめる事ができる. ボードに依存する処理を一つにまとめた後, 上記の順番で処理を行なえばアダプタに関する初期化をする事が可能である.

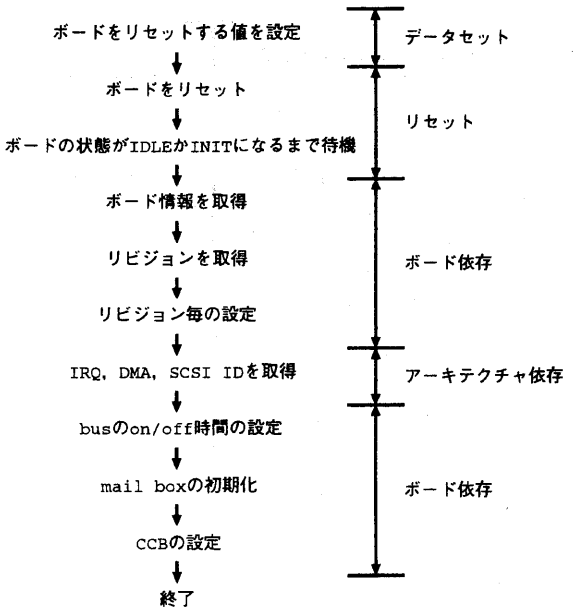


図 3: ahainit の処理の流れ

- scsi_cmd

  aha_scsi_cmd は SCSI コマンドを RESET, TARGET, OTHER に分けてアダプタのデータ構造を展開している. uha_scsi_cmd は RESET, OTHER に分けてアダプタのデータ構造を展開している. aha_scsi_cmd, uha_scsi_cmd の両方でサポートされている RESET と OTHER のデータ構造の展開について考えることで, アダプタに依存するデータ構造の一部を定式化する.

  表 3に RESET の際に必要なコマンドブロックの展開を示す.

表 3: RESET の際に必要なコマンドブロックの展開

	RESET
aha_scsi_cmd	opcode=AHA_RESET_CCB data_addr=0 data_length=0
uha_scsi_cmd	opcode=U14_SDR ca=0x01

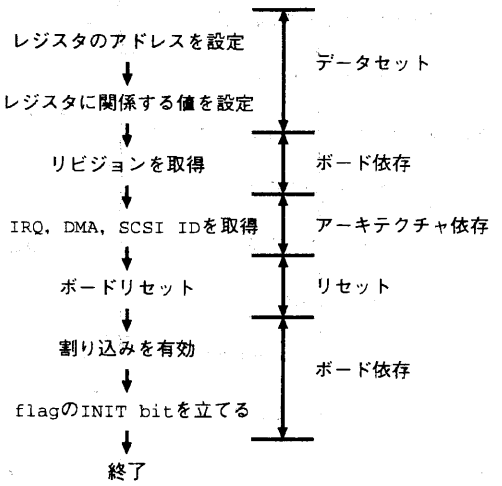


図 4: uhainit の処理の流れ

4 考察

今回，SCSI プロトコルと SCSI ホストアダプタの定式化を試みた．

3 章における，SCSI プロトコルの定式化がうまくいかなかったのは，ドライバに記述されている関数をこれ以上細かく抽象化しても，各デバイス間で共通な処理を見つけ出す事は難しく，また大きくまとめるためにはまとめたい操作の性質を見つけ出す必要があり，これが極めて難しいため，より抽象度の高い表現にすることはあきらめざるをえなかったからである．NetBSD のソースコードにおいて，既に各デバイス毎に記述されているドライバの関数が，SCSI プロトコルの定式化としてもっともふさわしいと考えられる．

一方，SCSI ホストアダプタの定式化に関しては，init を 4 つの処理に抽象化できた．抽象化できた理由としては，

- ボードの初期化に必要なデータはある程度明確に限定されているので，これ以上細分化できない．
- ボードをリセットするコマンドは 1 つか 2 つ程度なので，最初のボードの初期化に必要なデータ同様極めて限定される．
- IRQ，DMA，SCSI ID という項目は最初から概念的に抽象化されている．
- ボードに依存する処理はそれだけでひとまとめにせざるをえなかった．

が考えられる．

scsi_cmd に関しては init と違うアプローチを試みた．init で高かった抽象度を低くする事によって，より具体的なデバイスの仕様記述を定義するためである．この手法をアダプタに特有な項目の全てに適用すれば，アダプタの仕様記述の定義を決定する事ができる．

5 おわりに

今回はデバイスの仕様記述を決定するために，既存の OS のデバイスドライバの一部を抽象化した．デバイスドライバの処理の流れと，SCSI ホストアダプタが持つデータ構造に関しては，今回試みた手法を用いる事によって定式的に取り扱う事が可能になる．

今後，SCSI ホストアダプタの命令と I/O ポートの定式化に取り組む必要がある．

参考文献

[1] Xiaohua Jia and Mamoru Maekawa: *Operating System Kernel Automatic Construction*, Operating Systems Review, Vol.29, No.3, pp.91–96, 1995.

[2] 長尾周司，片山徹郎，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] FreeBSD ハンドブック (http://www.jp.freebsd.org/www.freebsd.org/ja_JP.EUC/handbook/)

[4] Architecture Supported by NetBSD (http://www.netbsd.org/Ports/index.html)