オペレーティング・システム 58－5
（1993. 3. 19）

# マイクロ・カーネル・アーキテクチャにおける I/O driver 実装方法に関する一考察

加藤 意之　乾 和志　人見 潔

オムロン（株） システム総合研究所

マイクロカーネルアーキテクチャのＯＳにとって、ＯＳのサーバとデバイスドライバのメモリ空間が違うことはパフォーマンスの面から見て問題となる。そこで、ユーザレベルのデバイスドライバが考えられているが、我々は実際に二通りのデバイスドライバをインプリメントし、パフォーマンスの違いについて測定を行なった。本発表ではその結果について考察を行ない、今後の問題点についてまとめた。なお、測定の対象は LUNA88K で稼働している MACH3.0 のイーサネットドライバである。

# Implementation of the I/O driver for Microkernel based Operating System

Motoyuki KATOH　Kazushi INUI　Kiyoshi HITOMI

OMRON Corporation Advanced Systems Research Center

On microkernel based operating system, the OS server has different address space from I/O driver. It causes disadvantage in point of performance. We implemented the two types of I/O driver and measured the performance of both. In this paper we'll show the results of the measurement and discuss about the future problems. The target is ethernet driver of MACH3.0 on LUNA88K.

# 1 はじめに

近年、並列、分散処理を有効に行なうオペレーティングシステムが求められているが、その基本思想としてマイクロカーネルアーキテクチャのＯＳが増えてきている。これは必要最低限の機能を持つマイクロカーネル部と、その上に構築されユーザレベルで実現されるＯＳサーバから構成されている。

このようなＯＳのＩ／Ｏシステムでは一般的にはＯＳサーバとデバイスドライバのタスクの空間が違うため両者の間で何らか手段を用いてデータのやりとりが必要となり、パフォーマンスがよくないという問題がある。その問題を解消するために、ＯＳサーバにデバイスのメモリ空間をマッピングしデバイスドライバをユーザレベルで動かすという方法がある。この方法ではＯＳサーバとデバイスドライバ間でのデータ通信のオーバヘッドが減るためパフォーマンスの向上がはかれる。

我々はマイクロカーネルアーキテクチャのＯＳにおける、デバイスドライバのインプリメント方法について考察を行なった。実際に二つの方法（デバイスドライバがマイクロカーネル内で動作する場合とデバイスのメモリをマッピングすることによってＯＳサーバ内でデバイスドライバが動作する場合）でインプリメントを行ない、それらのパフォーマンスの比較を行なった。

本発表では

- Mach3.0 におけるデバイスドライバの概要
- Mach3.0 におけるイーサネットドライバのインプリメントの詳細
- デバイスドライバのパフォーマンス測定
- デバイスドライバの実装に関する考察

について述べる。

# 2 Mach3.0 におけるデバイスドライバ

## 2.1 Mach3.0 オペレーティングシステム

Mach3.0 はカーネギーメロン大学で研究開発されているマイクロカーネルベースのＯＳである。その上に 4.3BSD サーバを動かすことで、4.3BSD UNIX と互換性を持った環境を提供している。

Mach3.0 マイクロカーネルは下記のような機能を提供する。

- タスク、スレッドの管理
- プロセス間通信の管理
- スケジューリング
- 物理資源の管理

## 2.2 Mach3.0 のデバイスドライバ

Mach3.0 のデバイスドライバは次の三つの特徴を持っている。

1. デバイス・インディペンデント・ドライバ
2. ユーザレベル・デバイスドライバ
3. ロケーション・トランスペアレンシ

### 2.2.1 デバイスインディペンデントドライバ

Mach3.0 より以前のバージョンの Mach では BSD UNIX のデバイスドライバを、Mach の仮想メモリ管理、スレッド管理機構に合うように多少の変更を加えて使用していた。これらのドライバは一般にマシンベンダーから供給されるため、同様なハードウェアのコントローラチップを使っているマシンでもベンダーが異なれば、ドライバのコードも大きく異なっていた。

しかし、同じ種類のデバイスでは基本的な機能は同じである。これらの共通の機能をまとめて階層化を行ない、チップに依存する部分を階層の一番下に隠すことで、システムのチップに依存するコードを他のチップのコードに変更す

ることが楽になる。また移植性の向上だけでなく、チップのバージョンアップなどに対応することも容易になる。

このようにデバイスの種類ごとに階層化を行ないそれぞれの階層にデバイスの種類に依存するインタフェイスを決め、コントローラチップ依存部をその最下層に隠すことで、大きくチップ非依存のコードを書くことができる。

Machでは、BSD UNIXのようなキャラクタデバイス、ブロックデバイスを持たずに、スクリーン、コンソール、ディスク、テープ、シリアルラインそしてイーサネットなど機能的にまとめられたデバイスを持っている。それぞれのドライバは一つまたはそれ以上のチップ非依存の階層を持っている。それらは外部インタフェイスをもち、デバイスドライバの論理的な部分がインプリメントされる。最下層の核となる関数だけがハードウェアを直接扱う。この構造のおかげで、移植性とコードの共有率が大きく向上した。

このデバイスインディペンデントドライバはMachのマシン非依存VMインタフェイス(pmapモジュール)に似ている。このpmap階層はVMシステムの下層にあり、MMU(Memory Management Units)依存の部分を隠している。これによって新しいマシンに移植する場合にはpmap層のみを変更すれば良く、またpmap層のほとんどは同類のMMUでは同じであるので、移植性、コードの共有性が向上した。

### 2.2.2 ユーザレベルデバイスドライバ

デバイスをユーザレベルで動かすことには二つの利点がある。

- パフォーマンスに優れる。
- 高優先度で走る時間（インタラプトコンテキストで走る時間）が少ない

デバイスドライバがマイクロカーネル内にある場合、データバッファもマイクロカーネル内にある。これはマイクロカーネル、ユーザの境界を越えたデータの受渡しが必要なことを意味する。これに対して、ユーザレベルのドライバではデータバッファをアプリケーション側にマッピングしているため、このデータの受渡しが不要であり、パフォーマンスが向上する。

また、デバイスドライバの処理がほとんどカーネルモード（特権モード）で実行される場合、一般にその間は高優先度で実行されているので、リアルタイム処理を行なう場合には不利である。これに対して、ユーザレベルでデバイスドライバが動くことができれば、カーネルの特権モードで動く時間は短くすることができる。これはスケジューリングができない時間が予測可能になるので、リアルタイムOSに対応するために有効である。

このようなユーザレベルでのデバイスドライバを実現するには次のような機能がマイクロカーネルに必要である。

- マイクロカーネルのアドレス空間（デバイスのコントロールレジスタ、データバッファなど）をユーザタスクのアドレス空間にマップする機構
- カーネルのインタラプト処理ルーチンとユーザレベルのデバイスドライバとの間の同期機構

ユーザレベルのタスクからデバイスを操作するには、デバイスの制御レジスタをユーザタスクのメモリ空間にマップすることが必要である。また、マイクロカーネルとの間で制御用の情報共有のための領域なども必要となる。そのために、マッピングの機構が必要となる。

デバイスからの割り込みが発生した場合、マイクロカーネル内でそれを受けとる。ユーザレベルのデバイスドライバの場合はその割り込みをユーザレベルの割り込み処理ルーチンに知らせる必要がある。

また、マイクロカーネル内では割り込みレベルの制御ができるのでそれを用いてクリティカルリージョンの排他制御を行なっていたところを、一般にユーザレベルでは割り込みレベルの制御はできないのでロックなどを用いて排他制御をエミュレーションする必要がある。

### 2.2.3 ロケーショントランスペアレンシ

デバイスが直接つながっていないリモートなマシンから他のマシンのデバイスをアクセスすることは、重要なことである。このために、カーネルのトラップ機構を使ったインタフェイスだけでなく、IPC を使ったデバイスインタフェイスが用意されている。これは NORMA 型のマルチプロセッサマシンにとっては重要なインタフェイスである。

## 2.3 Mach3.0 の I/O インタフェイス

Mach3.0 の I/O インタフェイスは、MIG(Mach Interface Generator) を用いて定義されており以下のようなインタフェイスがある。

**device_open** (master_device_port, mode, name, device)
オープンルーチン。name に対応するデバイスとの接続を確立しそのデバイスのポートを返す。

**device_close** (device)
クローズルーチン。device に対応するデバイスとの接続を断つ。

**device_write** (device, mode, recnum, data, num_bytes, bytes_written)
ライトルーチン。device に対応するデバイスに対して、num_bytes のデータを書き込む。実際に書き込んだバイト数を返す。

**device_read** (device, mode, recnum, bytes_wanted, data, bytes_read)
リードルーチン。device に対応するデバイスからデータを読み込む。読み込んだデータとそのバイト数を返す。

**device_map** (device, protection, offset, size, pager, unmap)
マップルーチン。デバイスに相当するメモリマネージャとの接続を確立し、そのページャのポートを返す。そのポートを vm_map() に渡してマイクロカーネルのメモリ空間をユーザタスクのメモリ空間にマップすることができる。

**device_set_status** (device, flavor, status)
デバイスのステータスを変更する。flavor の値の意味はデバイスに依存する。

**device_get_status** (device, flavor, status)
デバイスステータスを得る。flavor の値の意味はデバイスに依存する。

デバイスの名前は Mach3.0 ではアルファベットと数字からなる文字列で、数字によって同じ種類のデバイスにつながった異なるデバイスを区別する。レコードナンバ (recnum) はデバイスによってその意味は違う。ディスクでは物理ブロックの指標として用い、シリアルラインではこれを無視する。リード、ライトルーチンではデータをインライン、アウトラインデータとして戻すことができる。イーサネットのように非同期にデータを返してくるようなデバイスのためにリードルーチンはリクエストとリプライの二つに分割することができる。

このインタフェイスを守るものはデバイスドライバがカーネルの内部か外部にインプリメントされているかどうかに関わらず、Mach のデバイスとして認められる。ユーザレベルのデバイスドライバでも同じインタフェイスがカーネルによってエキスポートされているのでユーザアプリケーションにとってドライバがカーネルの内部または外部にインプリメントされているかどうかは関係ない。また、ユーザレベルのドライバは他のインタフェイスをシェアードメモリやＲＰＣなどによって他のタスクに対して提供することもできる。

# 3 イーサネットドライバの内部詳細

## 3.1 概要

Mach3.0 におけるイーサネットドライバは大きく二つの部分に分けることができる。

(a) チップからの割り込みを受けて、データをバッファからデバイスドライバのメモリ空間にコピーする部分

(b) デバイスドライバからのデータを受けてUNIX のネットワークにおけるメモリ管理方式である mbuf 構造体にデータを入れ、上位のプロトコル層に渡す役割の部分である。

イーサネットから送られてくるデータはディスクなどの場合と違い非同期に上がってくるので、常に監視しておく必要がある。上記の二つの部分のうち (a) の部分は割り込みによって起動され、(b) の部分は UNIX サーバ側で thread として常に動いている。

通常のデバイスドライバの場合、(a) の部分はマイクロカーネルの内部で、割り込みコンテキスト内で動作している。UNIX サーバとの同期およびデータの送信の機構として Mach のメッセージ通信を用いている。

ユーザレベルで動くデバイスドライバの場合、(a) の部分は割り込みを受けたマイクロカーネルによって evc_signal 機構を使って起動され、UNIX サーバ側で動く。

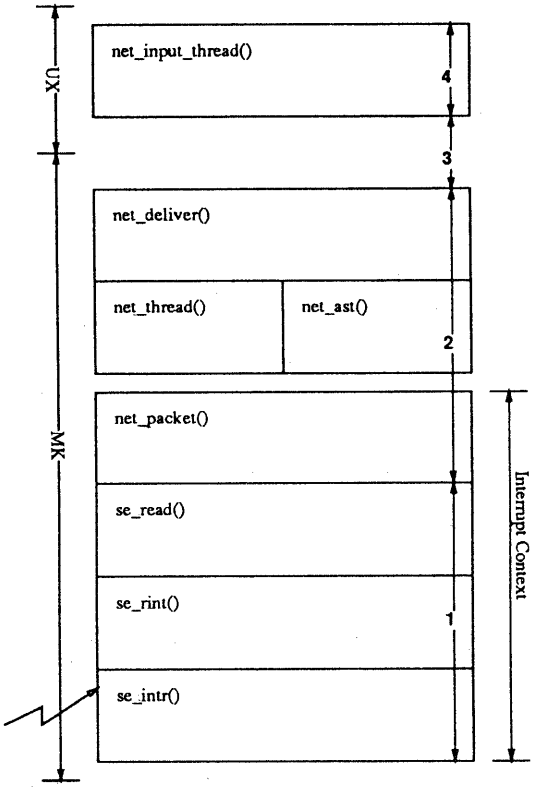


図 1: 通常の場合の内部構成（受信）

## 3.2 データ受信の処理内容

### 3.2.1 通常のデバイスドライバ

通常のデバイスドライバの構成を図 1 に示す。マイクロカーネル側で動くルーチン

- se_intr()
  イーサネットのハードウェアからの割り込みがあった後呼ばれる。エラーチェックおよびその割り込みがパケットの送信によるものか受信によるものかによってそれぞれのルーチンを呼び出す。

- se_rint()
  パケットを受信した場合に呼ばれる。イーサネットのコントロールチップとデバイスドライバとで共有しているバッファの管理を行なう。

- se_read()
  チップと共有しているバッファからデータをデバイスドライバのメモリ空間に取り出す。そのための領域の確保も行なう。

- net_packet()
  データをキューにつなぐ。もし net_thread() が起きていなければ、ast をオンにする。

- net_thread()
  キューを見てデータがあれば net_deliver() を使って、UNIX サーバにデータを送る。データがなければスリープしている。

- net_ast()
  net_thread() が起きていなければ、ast によって起動され、net_deliver() を使って、UNIX サーバにデータを送る。

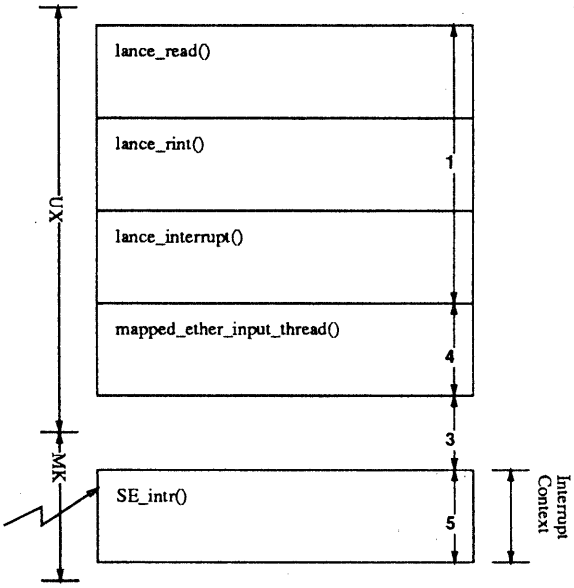


図 2: ユーザレベルの場合の内部構成（受信）

- net_deliver()
  キューからデータをとりだし、ipc_mqueue_send() を使って UNIX サーバにデータを送る。

UNIX サーバ側で動くルーチン

- net_input_thread()
  mach_msg() システムコールで、net_deliver() から送られてくるデータを待っている。そのデータを mbuf 構造体にいれて IP 層にあげる。

### 3.2.2 ユーザレベルのデバイスドライバ

ユーザレベルでデバイスドライバが動く場合の構成を図 2 に示す。

マイクロカーネル側で動くルーチン

- SE_intr()
  イーサネットのハードウェアからの割り込みがあった後呼ばれる。evc_signal() を使って UNIX サーバに割り込みを通知する。

UNIX サーバ側で動くルーチン

- lance_interrupt()
  マイクロカーネルからの evc_signal が届くと呼ばれる。エラーチェックおよびその割り込みがパケットの送信によるものか受信によるものかによってそれぞれのルーチンを呼び出す。

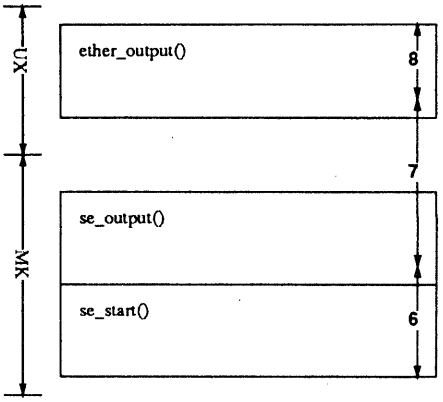


図 3: 通常の場合の内部構成（送信）

- lance_rint()
  パケットを受信する場合に呼ばれる。イーサネットのコントロールチップとデバイスドライバとで共有しているバッファの管理を行なう。

- lance_read()
  チップと共有しているバッファからデータをデバイスドライバのメモリ空間に取り出す。そのための領域の確保も行なう。

- mapped_ether_input_thread()
  UNIX サーバ側で常に動いており、evc_wait() で割り込みの通知を待っている。割り込みの通知が来たら lance_interrupt() を呼びだし、データを受けとる。そのデータを mbuf 構造体にいれて IP 層にあげる。

## 3.3 データ送信の処理内容

### 3.3.1 通常のデバイスドライバ

通常のデバイスドライバの構成を図 3 に示す。

UNIX サーバ側で動くルーチン

- ether_output()
  上位のプロトコル層から mbuf 構造体に入

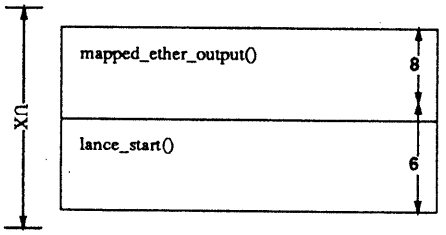


図 4: ユーザレベルの場合の内部構成（送信）

ったデータを受け、device_write_request() を使ってマイクロカーネル側にデータを渡す。device_write_request() は MIG で書かれたルーチンで、実際には mach_msg() システムコールを用いてマイクロカーネル側にデータを渡している。

マイクロカーネル側で動くルーチン

- se_output()
  UNIX サーバから受けたデータをキューにつないで se_start() を呼び出す。

- se_start()
  キューを見てデータがあればとりだし、デバイス側のバッファにデータをコピーしてチップをスタートさせる。

### 3.3.2 ユーザレベルのデバイスドライバ

ユーザレベルでデバイスドライバが動く場合の構成を図 4 に示す。

UNIX サーバ側で動くルーチン

- mapped_ether_output()
  上位のプロトコル層から mbuf 構造体に入ったデータを受け、そのデータをキューにつなぐ。

- lance_start()
  キューを見てデータがあればとりだし、デバイス側のバッファにデータをコピーしてチップをスタートさせる。

マイクロカーネル側で動くルーチン

なし

| 処理部 | 通常 (read) | 通常 (write) | ユーザ (read) | ユーザ (write) |
|---|---|---|---|---|
| 1 | 13.7 | 35.6 | 26.9 | 60.3 |
| 2 | 6.3 | 4.6 | — | — |
| 3 | 27.1 | 29.3 | 0 | 0 |
| 4 | 52.9 | 30.5 | 72.1 | 39.3 |
| 5 | — | — | 1.0 | 0.4 |

表 1: 処理内容の内訳（受信）単位は %

## 4 測定結果

以下に、二つの方法でインプリメントしたデバイスドライバのパフォーマンスの測定結果を示す。

### 4.1 測定 1

測定方法

あるマシン（Mach3.0 以外のＯＳ）とターゲットのマシン（Mach3.0）との間で、socket(TCP/IP) を用いてデータ転送を行なう。一方のマシンからはデータを 500Mbytes 書き続け (write システムコール)、もう一方ではそれを読み出す (read システムコール)。読んだデータは捨ててしまう。これを両方向について行なった。

測定結果

read システムコールの場合
通常のデバイスドライバ
85 Kbytes/sec
ユーザレベルのデバイスドライバ
212 Kbytes/sec
write システムコールの場合
通常のデバイスドライバ
105 Kbytes/sec
ユーザレベルのデバイスドライバ
203 Kbytes/sec

処理内容の内訳の表の処理部 (1 – 8) は図 1 – 4 のそれぞれの領域に対応している。同じ処理部は動作する環境がマイクロカーネル、UNIX サーバの違いはあっても内容的にはほぼ同じコードを用いてある。

| 処理部 | 通常 (read) | 通常 (write) | ユーザ (read) | ユーザ (write) |
|---|---|---|---|---|
| **6** | 7.7 | 8.4 | 48.8 | 33.1 |
| **7** | 89.2 | 81.9 | — | — |
| **8** | 3.1 | 9.7 | 51.2 | 66.9 |

表 2: 処理内容の内訳（送信）単位は %

### 4.2 測定 2

測定方法

あるマシン（Mach3.0 以外のＯＳ）とターゲットのマシン（Mach3.0）との間で、ftp を使って約 5Mbytes のデータの転送を行なう。

測定結果

通常のデバイスドライバ
59 Kbytes/sec

ユーザレベルのデバイスドライバ
100 Kbytes/sec

## 5 考察

測定結果より、Mach3.0 の通常のデバイスドライバよりも、ユーザレベルのデバイスドライバの方が、50 % ほどパフォーマンスが改善された。

これは標準のデバイスドライバの場合マイクロカーネルと UNIX サーバとの間の通信が余計にかかっている（表 1 における **2, 3** の部分および表 2 における **7** の部分。）ために、この差が現れたと考えられる。

ftp によるデータ転送の場合は、40 % 程の改善が見られる。これはディスクのアクセスを伴うため測定 1 の場合と差が見られると思われる。

マイクロカーネルのメモリ空間をＯＳサーバにマップすることで、両者の間の通信量を減らすことができパフォーマンスの向上がはかれる。しかし、これは複数のＯＳサーバを考えると個々のサーバがデバイスドライバを持つ必要があるなど、マイクロカーネルの特徴である柔軟性の面で問題がある。また、マイクロカーネル内にデバイスドライバが存在すると、新しいドライバの追加や、ドライバの改良などを行なうことが困難となり、拡張性の面で問題がある。

これらを回避するためにデバイスサーバとしてデバイスドライバをユーザレベルでインプリメントする方法が考えられる。この方法であれば、複数のＯＳサーバやその他のプロセスからもデバイスがアクセス可能であり、拡張性の面から見てもマイクロカーネルの変更なしにデバイスドライバを追加、改良することも可能なインプリメントである。この方法においてデバイスサーバとマイクロカーネルの間を本発表によるインプリメントにすることで、パフォーマンスを損なわずにデバイスドライバをサーバー化することが可能となる。

## 6 おわりに

Mach3.0 のイーサネットドライバを例に、マッピングを用いてユーザレベルで動作するデバイスドライバのパフォーマンスの有効性を示した。これは今後考えられるデバイスサーバのインプリメント方法として有効であると思われる。

## 参考文献


[1] Alessandro Forin, Davis Golub, Bryan Bershad:“An I/O System for Mach 3.0”, Mach Symposium, Autumn 1991

[2] 前川 守, 所 真理雄, 清水 謙多郎 編: “分散オペレーティングシステム UNIX の次に来るもの”, 共立出版, 1991

[3] S.J.Leffler, M.K.Mckusick, M.J.Karels, J.S. Quarterman: “UNIX 4.3BSD の設計と実装”, 丸善, 1991

[4] 乾 和志, 菅原 圭資:“分散ＯＳ Mach がわかる本”, 日刊工業新聞社, 1992