OS／omicron 第４版におけるデバイス管理の設計と実現

4M－2

佐藤元信，森永智之，早川栄一，並木美太郎，高橋延匡

（東京農工大学　工学部）

1. はじめに

マウス、タブレットといったデバイスが登場し、計算機が一般に普及するにつれ、GUI の研究が盛んに行われるようになった。筆者らの研究グループは、特に表示一体型タブレットに着目し、手書きインタフェース（以下、I／F）を提供するためのシステム基盤となる、OS／omicron 第４版（以下、V4）を研究･開発している。本稿では、V4 におけるデバイス管理、特にメモリマップト方式の設計と実現について述べる。

2. V4 におけるデバイス管理

V4 では、計算機上で仮想化された紙である電紙という資源を提供する。「電紙」とは、属性を持つ一つのセグメントであり、「文章の上に張り付けられた絵」のように紙同士の関係を表すために、電紙間のリンクを張ることができる。これを実現するために、V4 ではワンレベルストアを採用し、二次記憶装置をメモリと同じ I／F で扱える環境を提供する。V4 においてデバイス管理をする際に問題となるの次の二点である。

(1)　応答性の問題

手書きインタフェースでは、ユーザのペン操作に対するインキングの応答性が重要だが、ストリームを用いてデバイスを扱うのでは、コピーのオーバヘッドのため、十分な応答性を確保できない。

(2)　抽象化の問題

デバイスは個々に差異があり、統一的に扱うのが困難である。しかし、これらの差異を吸収しなければ扱う側に負担がかかる。

そこで上記の問題点を踏まえ、V4 におけるデバイス管理の目的を次のように設定した。

(1)　応答性の確保

デバイスの応答性の悪化につながる、コンテクストスイッチやデータコピーを極力減らす。

(2)　デバイスを統一的に扱える環境の提供

デバイスを抽象化して差異を吸収することで、デバイスを統一的に扱える環境を提供する。

3. デバイス管理の設計方針

前述の目的を達成するために、V4 におけるデバイス管理の設計方針を次のように設定した。

(1)　デバイスドライバは関数コール型とする

デバイスドライバ（以下、DD）をタスクとした場合、割込みやタスク間通信に伴うコンテクストスイッチのオーバヘッドが問題となる。そこで関数コール型とすることで、これらのオーバヘッドを削減する。

(2)　メモリと同じインタフェースを提供する

メモリの I／F には任意の単位でランダムアクセス可能であるという特徴がある。デバイスをメモリとして抽象化することで、デバイスを統一的に扱うことのできる環境を提供する。以下、これをメモリマップト方式と呼ぶ。

4. メモリマップト方式の設計

4.1　メモリマップト方式のモデル

通常、ユーザがメモリを利用するときは、利用の前後に「確保」、「解放」という OS の機能を呼び出すという I／F を取る。デバイスも同様の I／F で利用できるようにする。つまり、メモリを確保する際にデバイスを指定し、デバイスを確保したメモリにマッピングする。以降、デバイスをデータをやり取りするには、メモリにアクセスするだけでよい。この様子を図１に示す。

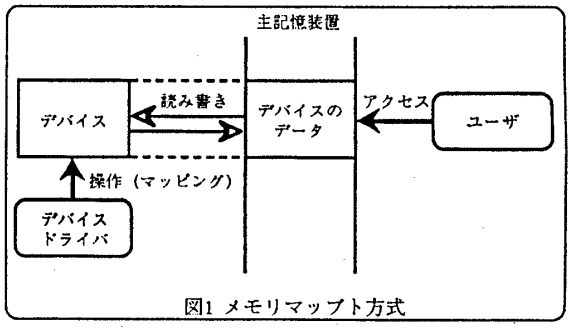

図1 メモリマップト方式

4.2　メモリマップト方式の機構

デバイスをメモリと同様に利用するには、確保時にデータの読込みを、解放時にデータの書出しをすればよい。しかし、このような機構では、(1) 内容を書き換えなくても書き出してしまう、(2) 物理メモリ以上の二次記憶装置をマップすると読込み時にページアウトしてしまう、などの問題がある。

そこで、メモリへのアクセスを監視するために、ページングフォールトを利用する。確保時には仮想アドレスとデバイスのマッピングだけを行う。初めてアクセスした部分ではページングフォールトが発生するので、

Design and Implementation of Device Management On OS／omicron Version 4
SATO Motonobu , MORINAGA Tomoyuki , HAYAKAWA Eiichi , NAMIKI Mitarou and TAKAHASHI Nobumasa
Faculty of Technology , Tokyo University of Agriculture and Technology

そのときにデバイスからの読込みを行う。解放するときはページングテーブルのダーティビットを見て、内容の書換えが行われたか判断し、該当ページだけを書き出せばよい。

5. メモリマップト方式の実現

メモリマップト方式を PC/AT 互換機上で実現されている V4 用マイクロカーネルのもとで実現した。このカーネルはプロセッサコンテキストスイッチ、セグメント管理、割込み管理の機能を提供する。この上にタスク管理機構、タスク間通信管理、割込みの仮想化機構などを実現し、デバイスドライバやページャを動作させるための環境を整備した。

5.1 デバイスドライバの実現

今回の実現では IDE とマウスのデバイスドライバを作成した。各ドライバは初期化と読込み、書込みの機能を提供する。ドライバを利用するのはページャだけであり、ユーザはページャにメモリを要求するという形でデバイスを利用する。つまり、デバイスドライバはユーザからは完全に見えない位置にある。マウスはバイト単位でデータをやり取りするため、ページングフォールト（4K バイト単位）では実用的な応答速度を得ることができない。そこで、マウスのようなデバイスでは、セグメンテーションフォールトを利用し、バイト単位でアクセスを監視することとした。マウスのデータの受信は割込みを利用し、マウスがマップされているメモリにデータを順次格納する。全体構成を図 2 に示す。

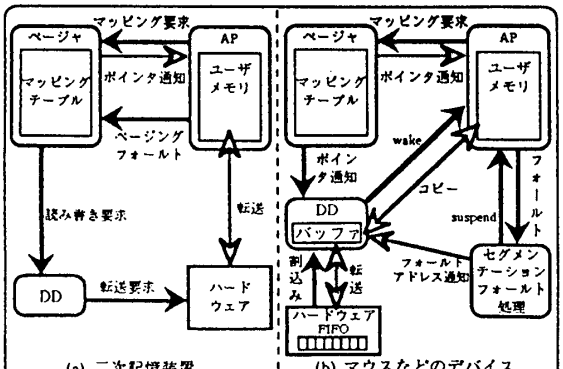

図2 メモリマップト方式の全体構成

5.2 ページャの実現

ページャはタスクとして実現し、プロセス間通信（以下、IPC）により起動する。各ページごとに対応するデバイスを格納するテーブルを作成し、仮想アドレスとディスクのマッピングを管理する。ユーザからのメモリ確保要求に従ってマッピングテーブルを作成し、ページングフォールトによりデバイスドライバを呼び出す。マウスのようなデバイスでは、セグメンテーションフォールトが発生したとき、フォールトしたアドレスをデバイスドライバに通知する。

5.3 MS−DOS ファイルシステムの実現

メモリマップト方式の評価をするために、MS−DOS のファイルを読み込むことができるサーバを実現した。提供する機能は、従来の open／close と read、メモリマップト方式による map／unmap である。このサーバは IPC により利用される。

6. メモリマップト方式の評価

実現したファイルシステムを用いて、性能測定を行った。まず、測定環境を表 1 に示す。

表1 測定環境

CPU	i80486DX4/100MHz
メモリ	15.6 Mバイト
HDD	IDE 200 Mバイト

測定は、23664 バイトのテキストファイルを画面に出力するために要した時間である。実行開始前のハードディスクのヘッドの位置、画面上のカーソルの位置などはまったく同じ環境で測定したので、画面への出力とディスクリードにかかった時間はどちらの場合も同じである。それらを除いた処理時間の詳細を図 3 に示す。

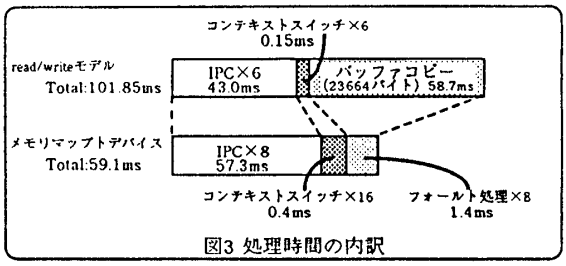

図3 処理時間の内訳

図に示すように、メモリマップト方式のオーバヘッドは read／write モデルのものと比較すると 58.0%であった。read／write モデルの場合、ファイルサイズが大きくなるにつれ、バッファからユーザメモリへのコピーのオーバヘッドが問題となる。メモリマップト方式の場合、ファイルが大きくなるにつれフォールト処理が多くなることが問題となる。1 セクタ当たりのオーバヘッドを計算すると、read／write モデルでは 1.27ms、メモリマップト方式では 0.94ms であり、読み込むセクタが多くなっても、結局メモリマップト方式の方が高速である。

7. おわりに

本稿では、デバイスをメモリと同じインタフェースで扱う環境を提供するデバイス管理の設計と実装について述べた。本方式では、二次記憶装置においてはバッファコピーによるオーバヘッドを削減することができ、この方式が有効であることを示すことができた。マウスなどのデバイスではより頻繁にフォールトが発生するので、処理速度が低下することが予想される。今後はマウスなどのデバイスの性能評価を行う予定である。

参考文献

[1] Maurice J.Bach,「UNIX カーネルの設計」, 共立出版, 1991