OS/omicron 第4版におけるデバイス管理とウィンドウシステムの実現

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

マルチメディアで扱うデータは、音声や動画のような大規模なものが多く、データの形式は多種多様である。従来のようなサーバによるシステム構成の場合、容易にモジュールを追加できるので新しいデータ形式に対する拡張性が確保できるが、サーバ間のコンテクストスイッチやデータコピーのオーバヘッドが問題となる。また、デバイスのインタフェースをファイルモデルに限定してしまうと、上位の資源管理がマルチメディアにおける多様なデータへの対応が困難になる。

筆者の所属する東京農工大学大学院工学研究科高橋・並木研究室では、手書きインタフェースに着目し、パタンデータ処理を指向したOS/omicron第4版(V4)[1]の研究を行っている。このOSは当研究室の研究基盤であり、現在V4上でCSCWの研究[2]や構造を持ったファイルシステムの研究などが行われている。

そこで、筆者らはデバイスの応答性を確保でき、パタンデータ処理に適したデバイス管理の研究を行っている。今回、このデバイス管理上にOS/omicron第3版(V3)用に開発された"未"ウィンドウシステム[3]を移植した。本稿では、V4のデバイス管理とそれを利用したウィンドウシステムの実現について述べる。

2. OS/omicron 第4版の概要

V4はパタンデータ処理を指向したOSである。パタンデータの中でも手書き入力に着目し、紙を仮想化した「電紙」をシステムで提供する。電紙は実際の紙と同様に、文字や図、表などを自由に書くことができ、紙同士を切り貼りすることもできる。

V4では上位の資源管理機構を柔軟に構築できるようにマイクロカーネル構成を採用し、名前によるダイナミックリンク機構を導入する。この名前は識別子名だけではなく、システムの管理する資源すべてを名前で管理し、その実体をすべてポインタで表すことができるようにワンレベルストアを採用する。また、V4では64bitのアドレス空間を各タスクで共有する二次元アドレスを実行環境として想定している。電紙などのデータの実体はこの二次元アドレスの一つにマップされ、ユニークなオブジェクトIDが与えられる。

3. デバイス管理の設計方針

デバイス管理の設計方針を次のように設定した。

(1) メモリ同様のインタフェースの提供

デバイスを任意のタイミングで、任意の位置から、任意のサイズでアクセスできるメモリとして抽象化してユーザに提供する。

(2) 関数呼出しによるドライバの構築

デバイスドライバをタスクとして実装した場合、割込みの発生時やデバイス利用時にコンテクストスイッチが発生し、そのオーバヘッドが問題となる。そこで、呼び出したタスクのコンテクストのまま、関数呼出しでドライバの呼出しを行う。

4. デバイス管理の設計

4.1 セグメントマッピング方式のモデル

方針にもあるように、V4ではデバイスをメモリとして仮想化して提供する。つまりデバイスとデータをやり取りするバッファを、直接ポインタとしてユーザに提供する。しかし、シーケンシャルデバイスのようなバッファリングが必要なデバイスでは、ポインタを切り替えなければならなくなるので、二重間接のポインタでデバイスにアクセスすることにする。この様子を図1に示す。

4.2 プログラミングインタフェース

V4では、名前により資源を管理することを先に述べた。デバイスもその例外ではなく、外部変数名として表される。V4の実行環境では、外部変数の参照はリンクテーブルを介して行われ、名前のバインドは実行時にダイナミックリンカにより決定

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

される。デバイスを利用する際は巨大な配列が外部変数として存在するものとして記述すればよい。ただし、デバイスへの書込みはユーザが明示的に指示する必要がある。実際の利用例を図2に示す。

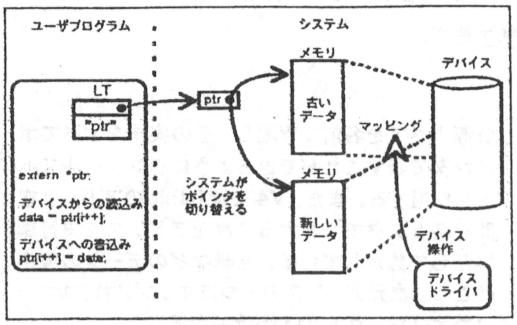

図1 セグメントマッピング方式のモデル

```
extern  UBYTE   *KeyBoard;
イベント        event;
CHAR            key;
UDBLWORD        i = 0L;

while( 1 ){
  key = KeyBoard[i++];
  event.種類 = @キーボード;
  event.コード = key;
  イベントをキューに積む( &event );
}
```

図2 デバイスの利用例

4.3 デバイスアクセス方式の設計

上述のインタフェースを実現するためには、ユーザのメモリアクセスを監視しなければならない。I/O処理の粒度はデバイスごとに異なるので、ランダムアクセスデバイスではページングの機構を利用し、ページ単位のアクセス監視を行う。シーケンシャルデバイスでは、I/Oの処理はバイト単位で行われるので、メモリの境界違反保護を利用することでバイト単位のアクセス監視を行う。デバイスドライバはこれらのフォールトやデバイスからの割込みをタイミングとして呼び出される。

5. デバイス管理の実現

実現はPC/AT互換機用に開発されたV4用マイクロカーネル上で行った。記述言語は当研究室で開発した言語C処理系であるCAT386である。Intel386系プロセッサで実現するにあたって、デバイスはセグメントにマップし、ページングフォールトとセグメンテーションフォールトを利用することでメモリアクセスを監視した。実現したドライバは、キーボード、マウス、タブレット(RS-232C)、IDE ハードディスク、SVGAなどである。

6. ウィンドウシステムの実現

デバイス管理を実際に利用したタスクとして、V3用に開発されたペンインタフェース指向のウィンドウシステム、“未”を移植した。V3版の“未”はデバイスドライバとして記述されている。V3のデバイスドライバは、ファイルインタフェースを持つサーバのようなものであり、モジュライティが高く拡張性があるという特徴があった。

V4ではV3のようなデバイスドライバのインタフェースを提供しないので、関数呼出しにより呼び出されるサーバとして実現した。これまでのサーバ構成ではウィンドウをIDで管理していたので、“未”内部でID表を管理し、ID表を引くという操作が頻繁に行われていた。V4ではフレームメモリをポインタで管理し、IDからフレームメモリへの変換や表引きをする必要をなくした。

7. おわりに

本稿では、OS/omicron第4版におけるデバイス管理と、それを利用したウィンドウシステムの実現について述べた。本研究で得られた成果を次に示す。

・デバイスをメモリのインタフェースで扱える環境を提供した

・デバイス利用時のコンテクストスイッチを削減することができた

また、今後の課題として次のようなものが挙げられる。

・デバイスをマップされたメモリの保護の問題

・上位の資源管理を含めた定量的な性能評価

参考文献

[1] Hayakawa, et.al: “Basic Design of SHOSHI Operating System that Supports Handwriting Interface”, 情報処理学会論文誌, Vol.35 No.12

[2] 中島, 早川, 並木, 高橋:“分散協調作業における手書きのためのカード作業の一環境”, 情報処理学会第52回全国大会, pp.253-254, 1996

[3] 早川, 河又, 宮島, 加藤, 並木, 高橋:“ペンインタフェース研究・開発のためのウィンドウシステム“未”(HITSUJI)の設計と開発”, 情報処理学会論文誌, Vol.36 No.4, 1995