# OS/omicron における日本語プログラミング環境†

鈴木茂夫†† 小林伸行†† 田中泰夫††
中川正樹†† 髙橋延匡††

本論文では，日本語情報処理を前提とした研究用計算機システム OS/o における日本語プログラミング環境について報告する．OS/o は，システム全体で一貫して日本語が使用できるプログラム開発システムを目標としている．そのため，フル2バイトの文字コード体系を採用し，ファイル名，プログラミング言語の識別子に至るすべての文字に日本語を使用可能とした．これは，OS/o を含めたすべてのシステムプログラムの記述言語として開発した言語Cの処理系 CAT の文字型を2バイト化することにより実現した．文字の書体，大きさなどの文字属性の表現方式として，属性情報を文字コードの実体から切り離し，独立した別のファイルに持つ方式を採用した．文字コードの実体ファイルと属性ファイルの統一的管理はファイルシステムにより実現している．これにより，属性情報を必要としない OS やコンパイラは，文字コードの実体だけを扱うことで，文字属性を意識する必要がなくなる．また，各アプリケーションの要求に応じて OS の機能を動的に拡張する機構を用意し，これを利用して日本語変換入力機能を実現した．そして，OS/o の一応用として，レーザビームプリンタを用いた日本語文書出力システムを開発した．以上のように，OS/o では，日本語の入力，処理，そして出力を含めたトータルな日本語プログラミング環境を実現している．

## 1. はじめに

計算機科学の発達にともない，我が国では母国語である日本語の処理を，計算機を利用して行いたいという要求が次第に強まってきた．英語文化圏においては，英語の入力，文書処理，清書出力といった一貫した環境をすでに計算機上に確立していることを考えれば，これは当然の流れと言える．

日本語情報処理の研究として，我々はオンライン手書き文字認識による日本文入力，レーザビームプリンタを用いた日本語文書出力などの研究を行っている．こうした研究のためには，日本語を扱う機能を備えた開発システムが必要である．しかし，既存のオペレーティングシステム (OS) は，そのほとんどが欧米で開発されたものであるため，その上で日本語処理を行うことには，多くの場合困難が伴う．

例えば，これらのシステムでは，日本語処理機能の実現のために，1バイトコード体系から1バイト2バイト混在のコード体系への変更が一般に行われている．しかし，そのために文字列操作のアルゴリズムが複雑になるなどの副作用が生じている．また，日本語が使用可能となっても，その仕様に一貫性を持ったシステムは少ない．例えば，テキストの入出力，言語処理系のコメントや文字列に日本語が使用できても，ファイル名やプログラミング言語の識別子には許されないといったシステムがある．これらの問題は，システムの日本語化を，欧米で開発された1バイト系のシステムに日本語処理機能を付加する形で行った結果生じた問題である．さらに，こうしたシステムの多くはブラックボックス化されており，そのために一般ユーザばかりか研究者でさえも，OS に対して機能の追加・削除といった改造を行うことが事実上できないという問題もある．

そこで我々は，日本語情報処理を前提とし，研究者が自由に手を加えることのできる透明性 (transparency) を持った OS の必要性を認識し，OS/omicron (OS/o) として実現した．さらに，OS/o を含めたすべてのシステムの記述言語として言語Cを採用し，言語処理系 CAT (C compiler developed at Tokyo University of Agriculture and Technology) の開発を行った．

OS/o では，システム全体で一貫して日本語を使用できるプログラム開発システムを目標としている．このためには，文字コード体系および文字属性の問題，言語処理系の日本語化，そして日本語入出力におけるユーザインタフェースの問題などの解決が必要である．そこで本論文では，こうした諸問題を含めた言語処理系 CAT の日本語化，OS/o の構成，および日本語入力環境としての日本語変換入力機能の実現方法について述べる．

† The OS/omicron Programming Environment in the Japanese Language by SHIGEO SUZUKI, NOBUYUKI KOBAYASHI, YASUO TANAKA, MASAKI NAKAGAWA and NOBUMASA TAKAHASHI (Department of Information Science, Faculty of Technology, Tokyo University of Agriculture and Technology).

†† 東京農工大学工学部数理情報工学科

## 2. OS/o の開発方針

前述したように，OS/o は日本語情報処理を念頭に置いて，研究者が自由に手を加えることのできる OS として設計された．以下にその開発方針について述べる．

### 2.1 フル2バイトコードの採用

OS/o では日本語情報処理に対して標準的なアーキテクチャを提供することを前提としている．そこで第一の問題としてあげられるのが，文字コード体系の問題である．既存の OS では，その文字コードとして通常1バイトコード体系を採用している．これは，既存の OS のほとんどが英語文化圏で開発されたためであり，英語は1バイトで表現できるからである．しかし，日本語処理のためには，1バイトコードでは当然不十分である．そこで，一般には1バイト2バイト混在のコードで日本語の表現を行っているが，そのために文字列操作などのアルゴリズムは文字列の内容に依存することになる．その結果，文字コード表現の状態遷移に伴う非本質的処理を要することになる．

この問題を重視し，OS/o では自然な形で日本語情報処理を行うことができるよう，制御文字も含めて，文字コードをすべて2バイトで表現することにした(これをフル2バイトコードと呼ぶ)．これは OS/o を含めたすべてのアプリケーションの記述言語である言語Cの文字型をフル2バイト化することにより実現した．

### 2.2 文字属性情報を属性ファイルとして分離

日本語を扱う場合，次に問題となるのは文字属性の扱いである．特に，全角と半角の違いは英語における小文字，大文字とは基本的に異なり，あくまでも外観上の違いである．こうした文字属性に対処する方法として，現在のところ以下の2つの方法が一般に用いられている．

(1) エスケープシーケンスによる方法

JIS で規格化されている文字属性はエスケープシーケンスによる状態遷移で行っている．この方式では，拡張性はあるものの複雑なエスケープコードがテキスト中に混在するという問題がある．アプリケーションは，このシーケンスの解析処理を行わなければならず，プログラムの開発効率，そしてその実行効率の両方で不利となる．文字属性情報を必要としない OS やコンパイラ等のシステムソフトウェアは，不要な文字属性情報をスキップする処理を要する．

(2) (文字コード)+(属性コード) で文字を表す方法

文字コードと文字に対する属性のコードを固定長の1セットとして扱う方式である．この方式では，文字コードのみを扱う場合，属性情報も必要とする場合，ともに容易に処理を行うことができるという利点がある．しかし，属性コード長の制限により，豊富な文字属性を必要とするアプリケーションの今後の発展に障害となる危険性がある．

以上のことから，OS/o では属性情報を文字コードの実体から切り離し，拡張性を失うことのないように，別のファイルに持つ方式を採用した．これにより，文字コードのみを扱うソフトウェアは文字コードの実体ファイルのみを用い，属性情報が必要なソフトウェアだけが属性ファイルを利用することとなる．しかし，この方式で問題となるのが，文字コードの実体ファイルと属性ファイルの管理を統一的に行わなければならないという点である．OS/o では，後述するファイルシステムによりその管理を実現する．

文字に対する属性は，出力系のアプリケーションや出力デバイスの仕様に依存している．したがって，ファイルシステムは，その機構を提供するだけで，属性情報の内容については各アプリケーションに任せることにする．

### 2.3 アプリケーション指向の OS

OS/o は必要最小限の機能を提供する．そして，アプリケーションソフトウェアを構築しようとするユーザが OS に対して機能の拡張を望めば，各アプリケーションにとって必要な機能を自由に付け加えていけるようにする．OS/o は，アプリケーションを含めた専用計算機システムを構築する際に，自由な形で利用できるソフトウェア，つまりアプリケーション指向の OS を目標としている．

### 2.4 スケジューリングの開放されたマルチタスク機能

並列処理の実現と，プログラミング環境におけるマルチタスクの有用性を考慮して，OS/o ではマルチタスク機能を実現する．パーソナルユースを前提とする OS/o にとって，この機能の実現はマルチユーザ・ジョブの実行のためというより，単一ジョブに対する並列処理の手段を提供することが主目的である．したがって，タスクの優先順位の変更などの操作により，自由にスケジューリングの変更を行えるように設計した．

### 2.5 システム記述言語として言語 C を採用

ソフトウェアの生産性，保守性を考慮し，またその記述力の高さから OS/o ではシステム記述言語として言語 C を採用した．そして，OS/o 自身の開発も言語 C を用いて行った．この目的は，アプリケーションソフトウェアを開発するユーザが OS のソースコードを容易に改造できること，そして OS 自身の生産性を上げることである．

### 2.6 リロケータブル・リエントラントな実行環境

オンライン手書き文字認識のようなリアルタイム処理を必要とするアプリケーションを可能とする必要がある．こうしたリアルタイムアプリケーションの時間的制御を容易にするため，OS/o では実記憶方式を採用している．このため，ハードウェアとしてアドレス空間の広い MC 68000 シリーズを CPU に選んだ．その上でマルチタスク機能を実現するためには，プログラムをリロケータブルにすることが望ましい．これは，将来マトリックス型などのマルチプロセッサ構成[3]に移行した場合に，オブジェクトモジュールを動的にリロケーションする必要からも要請された．また，同一プログラムを並列実行させる場合のためには，プログラムがリエントラントであることが望ましい．以上のことから，OS/o では，リロケータブル，リエントラントな実行環境を標準とする．これは，言語 C コンパイラ CAT によって実現する．

### 2.7 プログラムの ROM 化

OS の核や，頻繁に使用されるコンパイラ，エディタなどのシステムプログラムを ROM 化することにより，ディスクからのプログラムロード時間を省き，またプログラムの誤動作からもシステムプログラムを保護することができる．また，OS/o では後述する日本語辞書についても ROM 化を実現している．これにより，システムの起動時間の短縮，という点からプログラミング環境におけるユーザインタフェースの向上を図る．

## 3. 言語 C コンパイラ CAT の日本語化

OS/o は日本語文化に根ざした計算機システムを目標としている．これを実現するためには，まず OS/o 自身の根底からの日本語化を行わなければならない．それには，システム記述言語である言語 C の処理系 CAT の日本語化が必要である．逆に考えれば，CAT を日本語化することにより，OS/o の日本語化が可能であり，ひいてはシステム全体の日本語化につながる．この前提には，OS/o が 2 章で述べた方針でフル 2 バイトコード，属性ファイルを採用したことがある．

例えば，OS の核部分で文字を扱う場合，文字属性を属性ファイルとして分離したことにより，半角の “ABC” と全角の “ＡＢＣ” を文字コード列として処理することで，同じ識別名として扱うことができる．すなわち，OS の日本語化は，文字コードをフル 2 バイト化することで可能であり，複雑なシーケンスコードの解析といった新たな機能を付け加える必要などない．また，OS で扱う文字コードをフル 2 バイト化するためには，OS の記述言語である言語 C コンパイラ CAT の文字型の語長を変換するだけですむ．

このように，OS そしてシステム全体の日本語化は，言語 C の処理系 CAT のフル 2 バイト化により，その基底となる部分を実現できる．

### 3.1 フル 2 バイトの文字コード

OS/o では，フル 2 バイトの文字コードとして JIS X 0208 (旧 JIS C 6226)[8] のコード体系を採用した．制御コードについては，その上位バイトに NULL $(00)_{16}$ を付加することによりフル 2 バイトを実現している．JIS X 0202 (旧 JIS C 6228)[8] では，JIS X 0201 (旧 JIS C 6220)[8] の 1 バイトコード体系の中に JIS X 0208 の 2 バイト文字コードを組み込む方式を規定している．また，JIS X 0201 において，NULL コードは情報内容に影響を与えず，どこにでも挿入できる規約になっている．したがって，フル 2 バイトコード体系は JIS X 0201, JIS X 0202, JIS X 0208 の規定に従ったものである．

フル 2 バイトコード体系の利点は，今までの ASCII コード上での文字列比較などのアルゴリズムをそのまま使用できることにもある．これにより，ASCII コード上で開発された CAT の日本語化においても，ソースプログラムにおけるアルゴリズムの変更をほとんど行うことなく実現することができた．

### 3.2 識別名の日本語化

計算機のアプリケーションが，科学的問題から社会的文化的活動のシステム化に拡大されればされるほど，開発されるソフトウェアのなかには，英語に直しにくい用語が多くなる．例えば，日本語フォーマッタなどでは，組版規則の用語が日本語であるため，関数名，変数名といった識別名に漢字を含む日本語を使用したいという要求がでてくる．そこで，CAT では文字列，文字定数だけでなく，識別名にも日本語を許すことにした．

日本語の識別名を許すと，ASCII コードを採用している従来のコンパイラへの移植性の低下という問題が生まれてくる．しかし，日本語識別名を使用することにより得られる可読性，保守性の高さ，そして日本語フォーマッタなどのソフトウェアの生産性の向上という利点を重視した．

識別名として許される文字セットは，英字（'_' も含める），平仮名，片仮名，第一水準漢字，第二水準漢字であり，2文字以降に数字を許す．

### 3.3 実現方式

言語Cコンパイラ CAT は，ASCII コード体系のシステム上で開発された．その関係から，ソースコードは ASCII 文字集合でプログラミングされていた．これを ASCII 版 CAT とする．CAT の日本語化は，これをもとに文字コードをフル2バイトとする版（日本語 CAT）を生成する方針で臨んだ．ASCII 版 CAT は，プリプロセッサ，パーザ，コードジェネレータ，アセンブラ，リンカから構成されている．日本語 CAT の生成過程では，ソースプログラムの変更を最小限に抑える方針を採った．以下に実現方式を示す．

（1） 文字型が2バイトで識別名にフル2バイトコードの英数字を許すメタコンパイラ（meta CAT）を作成する

そのために，ASCII 版 CAT のパーザのソースコードに対して，

① 文字型に関する記述を符号付き2バイトに変更する

② 文字型の変数を符号付き2バイト変数にする．

③ 文字定数を符号付き2バイト定数に変更する．また，文字列を符号付き2バイト変数の配列に変更してフル2バイトコード列で初期化する．

④ 出力する中間コードの識別名を，フル2バイトコードから ASCII コードに変換するようにする．この理由は，meta CAT 作成の際に，アセンブラ，リンカの変更は行わず，従来の ASCII コード系のものをそのまま使用するためである．

以上の変更を施したパーザのソースコードを，ASCII 版 CAT でコンパイルし，ASCII 版 CAT の他のフェーズと組み合わせて meta CAT とする．

（2） meta CAT を用いて日本語 CAT を作成する

ASCII 版 CAT のプリプロセッサ，パーザ，アセンブラのソースコードに対して，日本語の識別名を許すように識別子を切り出す部分を変更する．これらのソースコードをフル2バイトコードに変換し，（1）で作成した meta CAT でコンパイルする．フル2バイトコードへの変換は，プリプロセッサとパーザのフェーズ間で行う．これは，meta CAT のプリプロセッサとして，ASCII 版 CAT のものを無修正で用いたためである

以上により，ソースコードの変更をほぼ最小限に留めて CAT の日本語化が完了した．この日本語 CAT を使用した言語Cプログラムの例を図1に示す．

```
/***    外部関数の定義          ***/
/*-------------------------------------------------------*/
/*〈関数名〉: main :                                     */
/*〈引数  〉 INT     argc;    : コマンド行の引数の数 :   */
/*          CHAR    *argv[]; : 引数を含む文字列の配列を */
/*                             指すポインタ :            */
/*〈機能  〉: プログラムの制御を行う。:                  */
/*-------------------------------------------------------*/
ENTRY   main(argc, argv)
INT     argc;
CHAR    *argv[];
{
        if (argc != 2)  {
                puts("使用方法 : [demo] [入力ファイル名]");
                exit(0);
        }
        if ((fp = fopenc(++argv, 0)) == NIL)    {
                puts("入力ファイルがオープンできません！");
                exit(0);
        }
        分類処理関数();
        表示処理関数();
        if (fclose(fp) < 0)     {
                puts("入力ファイルがクローズできません！");
                exit(0);
        }
}

/***    静的関数の定義          ***/
/*-------------------------------------------------------*/
/*〈関数名〉: 分類処理関数 :                             */
/*〈機能  〉: 文字を分類して各クラスの文字数を           */
/*          カウントする。:                              */
/*-------------------------------------------------------*/
FLOCAL  AVOID   分類処理関数()
{
        CHAR    文字;

        while ((文字 = getc(fp)) != EOF)
                if (漢字判定(文字))     {
                        漢字文字数++;
                } else if (片仮名判定(文字))    {
                        片仮名文字数++;
                } else if (平仮名判定(文字))    {
                        平仮名文字数++;
```

図1 日本語プログラムの例

Fig. 1 An example of the C programming in Japanese.

## 4. OS/o の構成

OS/o は，図2に示すように，ファイル管理，タスク管理，メモリ管理を行う OS の核の部分，ユーザアプリケーション層，そしてその間に位置するユーザ拡張部層の3つの層から構成されている．ユーザ拡張部は，ユーザが動的にタスクを登録して OS を拡張していくためのものであり，この機能の実現は2.3節で述

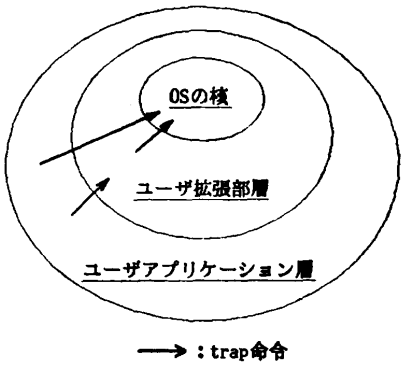



図 2 OS/o の階層
Fig. 2 Hierarchy of the OS/o system

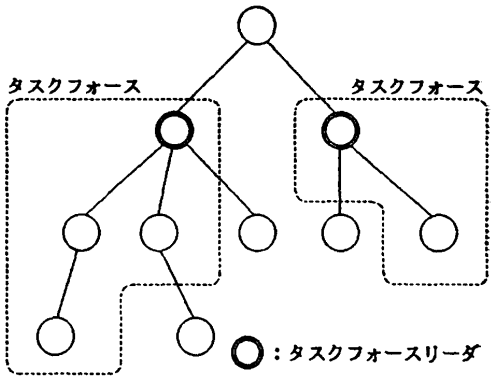



図 3 タスクの全体像
Fig. 3 Task structure of OS/o.

べた方針に基づいている．これにより，標準的または保護の強い機能を OS の機能として自由に付け加えていくことができる．したがって，ユーザアプリケーション層からは，より拡張された OS が見えることになる．具体的な例としては，後述する日本語入力環境における日本語変換タスクなどが，このユーザ拡張部層に登録される．

### 4.1 タスク管理部

本節では，OS に対して透明性を与え，また OS の機能の拡張性を提供するための OS/o のタスク管理部について述べる．

#### 4.1.1 タスクとタスクフォース

OS/o では，すべての処理プログラムをタスクという単位で扱う．タスクはプログラムによって実行手順を記述された OS から見た1つの制御単位であり，

（1） 手続き領域
（2） 静的データ領域
（3） ヒープ領域
（4） スタック領域

の4つの領域を主記憶上に持つ．タスクは動的に生成可能であり，その生成関係は図3のような木構造の親子関係となる．そして，これらのタスクはタスク管理の下で並列に実行される．

マルチタスクの環境のもとでは，1つの仕事を複数のタスクが共同して行う形態が考えられる．この場合，各タスク間で多くの情報交換が行われる．OS/o ではこうしたタスク群のために，一部の実行環境を共有し，情報交換を容易に行えるタスクフォースと呼ばれる形態を採り入れた．タスクフォースでは，実行環境として，手続き，静的データ，ヒープの3領域，そしてオープンされているファイル，セマフォを共有する．また，タスクフォース内の各タスクをタスクフォースメンバと呼び，タスクフォース内でルートに相当するタスクをタスクフォースリーダと呼ぶことにする．

タスクフォースで共有する実行環境はタスクフォース管理表（TFCB：Task Force Control Block）によって，そしてスタック領域のような各タスク固有の実行環境はタスク管理表（TCB：Task Control Block）によってそれぞれ管理される．

#### 4.1.2 ポインタ通信

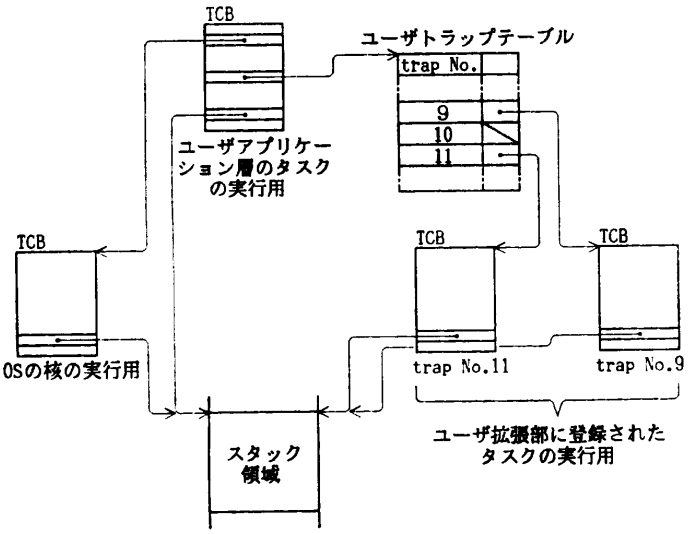
タスクフォースでない普通のタスクの場合でも，親タスクと子タスクの間では多くの情報交換が必要な場合がある．そこで OS/o では，メッセージ通信のほかに，親子タスク間で行うポインタ通信を用意した．これは，通信相手からポインタ値を受け取り，相手のデータ領域を直接アクセスすることにより，情報交換を行う方法である．

ポインタ通信は，実記憶方式の利点を活かした通信方法であるが，通信相手のデータ領域を操作するということから，プロテクションの問題がでてくる．したがって，親子タスク間で，しかも双方のタスクがその使用を認めた場合にのみポインタ通信を許している．

#### 4.1.3 タスクの実行

タスクは running 状態，ready 状態，suspended 状態の3状態をとる．OS/o では同期，排他制御の手段として PV セマフォを採用している．そして，割込み信号を基本同期手段である V 命令に抽象化することにより，割込みハンドラを事象（割込み）待ちで suspended 状態になっているタスクとして扱っている．つまり，OS/o では，割込み処理，例外処理にいたるすべての処理をタスクという単位で統一的に管理している．

ready リスト，suspended リストにリンクされるタ




OS の核およびユーザ拡張部タスクの実行のための TCB およびスタック領域は，ユーザアプリケーション層の各タスクに，タスク生成時点で用意される．

図 4 OS の核およびユーザ拡張部タスクの実行環境
Fig. 4 Run-time support by OS kernel and user extension tasks.

スクは，ともに優先順位の高い順にソートされる．そして，2.4 節で述べた方針に基づき，ユーザはこの優先順位を変更することにより，自由なスケジューリングを行うことができる．

#### 4.1.4 OS のタスク化

OS/o では，OS 自身もタスクとして扱う．これは，OS をタスクの実行環境で動作させ，スケジューリングの対象にするということである．これにより，OS の処理を実行中にも外部割込みを許すことができ，リアルタイム処理などに対処可能となる．しかし，ready リスト，suspended リストの操作や，context switching の処理は critical region として扱われるため，割込み禁止となりスケジューリングの対象外となる．

マルチタスクの環境では，並列に動作中の各タスクが同時に OS を呼び出すことがある．そこで，OS/o では，OS をすべてのタスクから同時に利用できる無限個（存在するタスクの数）のタスクとして扱う．これを実現するために，OS を利用できるすべてのタスクに対して，OS の実行環境としてのスタック領域と TCB を，タスク生成時点で用意する（図 4 参照）．スタック領域については，OS の呼び出しを同期式に限定することにより，呼び出しタスクのスタック領域を引き継いだ形で使用する．これらの機構により，各タスクから呼び出されるそれぞれの OS は，手続き，静的データ，ヒープの 3 領域を共有し，各呼び出しタスクにスタック領域が用意された，一種のタスクフォースメンバとなる．

#### 4.1.5 ユーザ拡張部の実現

ユーザ拡張部に登録されたタスクは，OS の一拡張機能であるため，OS の核と同様の方法（トラップ命令）でユーザアプリケーション層のタスクから呼び出される．したがって，ユーザ拡張部のタスクも，複数のタスクから同時に呼び出されることを考えなければならない．そこで，OS の核の場合と同様に，スタック領域，TCB をユーザアプリケーション層の各タスクに用意する方法で実現する（図 4 参照）．

また，ユーザ拡張部への登録は動的に行われるため，登録を行ったタスク，そしてそのタスクから生成されていった子孫のタスクに，登録されたタスクのための実行環境が用意されることになる．したがって，ユーザ拡張部のタスクを利用できるのは，そのタスクを登録したタスクとその子孫にあたるタスクだけである．すべてのタスクから利用可能なタスクを登録するには，すべてのタスクのルートにあたるタスクで登録を行えばよい．

### 4.2 ファイルシステム

#### 4.2.1 設 計 方 針

日本語プログラミング環境を実現するうえで，言語処理系とタスク管理を動の側面とすれば，ファイルシステムは静的環境としてユーザをサポートしなければならない．以下に，OS/o のファイルシステムの設計方針について述べる．

（1） 日本語を標準とすること

記述言語の識別名やコマンド名に日本語を許すならば，当然ファイル名やディレクトリ名にも漢字を含めた日本語を許すべきである．また，ファイルの入出力単位においても，文字については 2 バイトコードを標準とする．

（2） 実体ファイルと属性ファイルの統一的管理を指向すること

OS/o では，先にも述べたように，文字コードとその文字属性を分離し別ファイルとして表現する．しかし，属性の処理を必要とするアプリケーションは，実体と属性を同時処理する必要がある．そこで，ファイルシステムとしては，それらを統一的に管理する．実体と属性の関係は，文字に限らず有効であり，その意

味と処理はアプリケーションに任される．

（3） ファイルの世代管理を行うこと

OS/o ではプログラム，文書に対して，大容量の追記型光ディスクを用いた世代管理を行う[7]．

（4） 半導体技術の進歩に見合った記憶階層を設計すること

現在大容量化・低価格化が著しい半導体メモリ，特に ROM をファイルとして扱い有効活用できることを目指す．OS/o では，特にメモリのアドレス空間上におかれたファイルを LSI ファイルと名付ける．

#### 4.2.2 特　　徴

我々は，上記の方針をもとに以下の特徴を持つファイルシステムを構築した．

（1） 属性ファイルのサポート

属性ファイルをサポートするために，OS/o ではファイルの構造化を行った．ファイルは図5に示すようにいくつかのセグメントから構成される．これらのセグメントはファイルという単位で統一して管理される．例えば，クローズ，子タスクへの引継ぎなどをファイル単位でまとめて行うことができる．個々のセグメントは，すでにオープンされているファイルのファイル記述子とセグメントの名前をもってオープンされる．その結果としてセグメント記述子が得られ，ファイル記述子とペアで各セグメントへのアクセスが行われる．

このセグメントの1つを文字コードの実体，そしてそれ以外の任意個のセグメントを属性とすることにより，文字の実体と属性をファイルという単位で1つの資源として管理することができる．

文字属性の構成および内容の管理は，上記の方針に基づき，属性情報を必要とする各アプリケーションが行う．

例えば，後述する日本語ワープロ PWB を考慮した文字属性セグメントは，文字サイズ，網かけなどの属性情報を固定長で持ち，文字コードの実体と一対一に対応する構成になっている．一方，言語Cコンパイラ CAT のように文字属性を必要としないソフトウェアは文字コードの実体セグメントだけを対象に処理を行う．

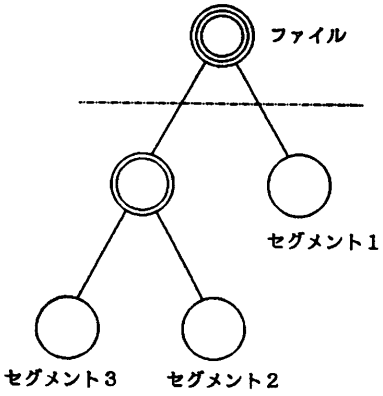



図5 OS/o ファイル構造
Fig. 5 The OS/o file structure.

（2） ROM ディスク

一般に，ワークステーション，ミニコンといった高機能な計算機になるにつれ，計算機の立ち上げに時間を要する．確かに高機能なものほどハードウェアの診断などに時間がかかるのであろうが，個人の道具となるワークステーション，パソコンにとってはスイッチを入れるだけですぐに立ち上がることがヒューマンインタフェースの面で重要なことである．

そこで，我々は安価になった ROM を使ってスイッチを入れただけですぐに立ち上がるホットスタートを実現する．

（3） 仮想フロッピディスク

OS/o では，高速で大容量のハードディスクをベースにしたファイルシステムを標準としている．しかし，ハードディスクが大容量といっても無制限というわけではない．そこで OS/o では，フロッピディスク装置を実装し，仮想フロッピディスクという方式を採用した[1]．仮想フロッピディスク方式は，ハードディスク（高速で容量制限あり）とフロッピディスク（低速だが交換可能）という2つの記憶階層を一元化する．機構としては，ハードディスク内の特定の領域をフロッピディスクのステージングエリアとして使用する．これにより，ディスク容量の論理的な制限をなくし，さらにフロッピディスクアクセスの高速化を図る．

## 5. 日本語入出力環境

OS/o，言語 C コンパイラ CAT の日本語化，属性ファイル等の実現，そして後述する日本語辞書により，日本語プログラミングにおける基本環境は実現できた．次に問題となるのは，ユーザと計算機とのインタフェースであり，具体的には日本語入出力の問題である．言語処理系がその識別名にまで日本語を使用できるようになっても，日本語プログラムを入力そして出力できる環境が整っていなければ，なんの意味もない．本章では，OS/o における日本語入出力環境の実現について述べる．

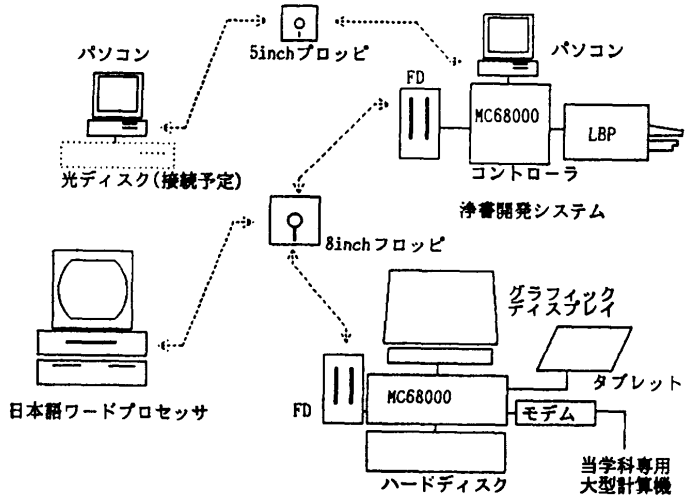



図 6 ワープロ PWB を用いた開発環境
Fig. 6 Development environment with Japanese word-processors as PWB's.

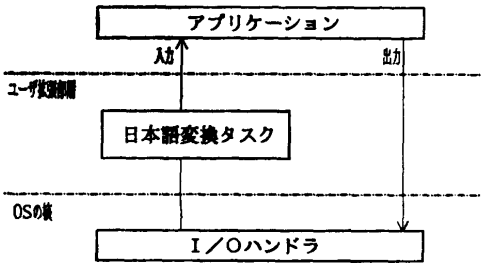



図 7 通常アプリケーションにおける日本語入力
Fig. 7 Japanese input by default transformation.

### 5.1 日本語入力環境

#### 5.1.1 日本語ワープロのオフライン PWB 化

我々は開発システムとして CP/M-68K を使用してきた．ところが，CP/M-68K には，日本語入力の機能が備わっていない．また当研究室では，最近まで日本語入力に必要な辞書が備わっていなかった．このため，スタンドアロンな日本語ワードプロセッサを用いて，**図 6** のような日本語入力の手段を開発した．これをワープロのオフライン PWB (Programmer's Work Bench) 化と呼ぶ[5]．これによりオフラインではあるが日本語入力を可能とした．

#### 5.1.2 OS/o の日本語変換機能

OS/o における日本語の入力は，OS/o のユーザ拡張部に日本語変換の機能を置くことにより実現する．日本語変換は，次の 2 つの方式で行う．

(1) フロントプロセッサ方式

アプリケーションが入力要求を行うと，**図 7** に示すようにキーボードから入力されたデータは，日本語変換タスクで日本語に変換されアプリケーションに入力される．この場合，アプリケーションは，変換がどこで行われるかということや漢字が混じるということを，まったく意識しないですむ．

しかし，この方式では日本語変換タスクが行う変換操作や候補の選択操作などにより，ユーザインタフェースが固定化してしまう．このため，エディタやコマンドインタプリタなどの入力として用いると，使用するキーが重なるなど不便なことが多い．

(2) 変換機能を直接呼び出す方式

方式(1)に対し，エディタやコマンドインタプリタなど，独自のユーザインタフェースを持つアプリケーションは，**図 8** の方式をとる．アプリケーションはキーボードからタイプされた生のデータを入力し，日本語に変換したい場合にだけ，日本語変換タスクに文字列を渡して変換する．このようにすると，エディタやコマンドインタプリタは変換操作，候補の選択操作などの

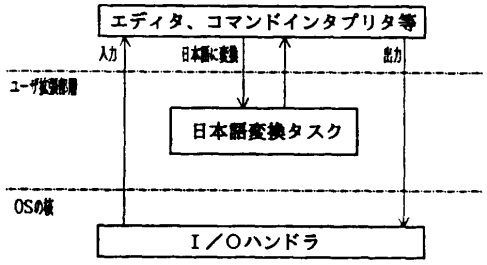



図 8 エディタ等における日本語入力
Fig. 8 Japanese input by on-demand transformation.

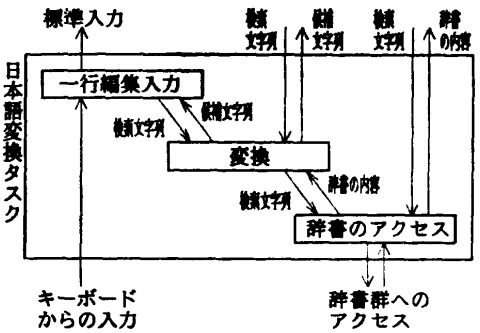



図 9 日本語変換タスクの内部構成
Fig. 9 Structure of the keycode to Japanese transformation task.

ユーザインタフェースをそれぞれ自由に設計することができる.

日本語変換タスクは図9に示す内部構成をとる. 辞書には, ユーザ辞書, 自立語辞書, 付属語辞書, 単漢字辞書などがある. このうち, 自立語辞書, 付属語辞書, 単漢字辞書を, OS/o のシステム辞書としてユーザに提供する. なお, 自立語辞書, および付属語辞書は,「九州芸工大自立語辞書 KID-J 82」を原辞書として生成したものである[4]. これらの辞書はロード時間の省略やアクセス速度を上げるため, ROM 化されている. 自立語辞書は, 約9万語の自立語と, その品詞, 活用などの付属情報から成る. また, その容量は約 2.7 Mbytes である. これと付属語辞書を組み合わせることにより, 日本語変換だけでなく日本語の解析などの日本語処理にも利用できる.

ユーザ辞書は, ユーザが自由にその内容を編集できる辞書である. このユーザ辞書を複数用いることによって, 目的別に使い分けることができる. ユーザ辞書の検索キーとして使用できる文字は, 仮名に限らずあらゆる文字が可能である. 情報処理用語など, 仮名の綴りでなく英字の綴りで登録したい場合や, エディタでプログラムに日本語を使用する場合などを考えると, アルファベットの並びから直接日本語に変換できると便利である.

変換や辞書のアクセスの機能はアプリケーションから呼び出すことができる. このとき, 検索する辞書や検索方法を変えるなど豊富なインタフェースを用意し, ユーザが自由に組み合わせて変換を行えるようになっている. この結果, パーソナル日本語ワープロなどが自由に設計できることを期待している.

### 5.2 日本語文書出力環境

我々は, 日本語文書出力システムの研究も行っている. このシステムでは, レーザビームプリンタをインテリジェント化し, 各種ソフトウェアおよびハードウェアの開発を行っている. 例えば, 研究室でのミーティング議事録の出力, プログラムリストの出力, さらには日本語フォーマッタを備えている.

このシステムによる美しいプログラムリスト, 文書の出力は, プログラマの創作意欲を高めることにもなる.

## 6. おわりに

本研究における成果を以下に述べる.

(1) ASCII コード体系のシステム上で開発した言語処理系をもとにフル2バイトコードの日本語言語Cコンパイラ CAT を作成した.

(2) 上記の処理系を用いて, 日本語オペレーティングシステム OS/o の開発を行った.

(3) フル2バイトコードの採用により, 自然な形で日本語処理を行うことが可能となった.

(4) 文字属性を属性ファイルとして文字コードの実体から分離したことにより, OS, コンパイラなどのシステムプログラムは文字属性を意識する必要がなくなった. このことは, CAT そして OS/o の日本語化において実証された.

(5) 日本語プログラミング環境における静的環境として, ファイル名, ディレクトリ名に日本語を許し, 文字コードの実体ファイルと属性ファイルを統一して管理できるファイルシステムを構築した.

(6) 日本語入力手段として, 日本語変換入力機能を実現した.

(7) 日本語変換入力機能の実現は, OS/o のタスク管理機構の透明性と拡張性に基づいている.

(8) 日本語入力, 処理, そして出力まで含めたトータルな日本語プログラミング環境を提示した.

そして, 現在 OS/o の日本語変換入力機能を利用した日本語プログラムエディタ等の開発を進めている. また, 言語Cコンパイラの識別子に漢字を含めた日本語を使用した場合の使い勝手や, ソフトウェアの生産性に与えるインパクトなどについても今後考察を加えていく予定である.

**謝辞** 日本語辞書の生成および整備を行った宮内忠信氏, 下村秀樹氏, ワープロ PWB の開発を行った並木美太郎氏, 日本語文書出力環境を作成した曽谷俊男氏, 狩野敦氏, 鈴木未来子女史に感謝する. また, 日本語辞書 KID-J 82 を提供していただいた九州工業大学の吉田将教授に深謝する.

## 参考文献

1) 林 努, 高橋延匡: MC 68000 用 OS の基本設計(1)—ファイルシステムの設計, 情報処理学会マイクロコンピュータ研究会資料, 25-3(1982).
2) 高橋延匡, 武山潤一郎, 並木美太郎, 中川正樹: MC 68000 用小型 OS: OS/o の開発, 情報処理学会計算機システムの制御と評価研究会資料, 21-6 (1983).
3) 高橋延匡, 阿刀田央一, 鵜澤繁行, 中川正樹, 小谷善行, 高田正之, 清水敬子, 中森眞理雄, 斎藤延男: マルチ・マイクロプロセッサ・システムの基本設計, 第26回情報処理学会全国大会論文集,

1 N-5, pp. 99-100 (1983).
4) 「日本語単語機械辞書」について，文部省科学研究費特定研究「情報化社会における言語の標準化」総括班 (1985).
5) 並木美太郎，関口 修，里山元章，中川正樹，高橋延匡：日本語ワードプロセッサのソフトウェア生産への応用，情報処理学会ソフトウェア工学，47-1 (1986).
6) 屋代 寛，中川正樹，高橋延匡：OS/o 第2版とシステム記述言語C，情報処理学会オペレーティング・システム研究会資料，32-3 (1986).
7) 横関 隆，中川正樹，高橋延匡：追記型光ディスクを用いた世代管理ファイルシステムの開発，情報処理学会オペレーティング・システム研究会資料，36-1 (1987).
8) JIS ハンドブック 情報処理—1987，日本規格協会 (1987).

(昭和63年 4 月 8 日受付)
(昭和63年11月14日採録)

**鈴木 茂夫** (正会員)

昭和62年東京農工大学工学部数理情報卒業．同年4月，同大大学院修士課程入学，現在に至る．オペレーティングシステムなどのシステムソフトウェアの研究に従事．

**小林 伸行** (正会員)

昭和62年東京農工大学工学部数理情報卒業．同年4月，同大大学院修士課程入学，現在に至る．プログラミング環境などの研究に従事．

**田中 泰夫** (正会員)

昭和61年東京農工大学工学部数理情報卒業．昭和63年同大大学院修士課程修了．在学中，システムソフトウェアの開発に従事．

**中川 正樹** (正会員)

昭和52年東京大学理学部卒業．昭和54年同大学院修士課程修了．同大学院在学中英国 Essex 大学留学(M. Sc. in Computer Studies)．昭和54年より東京農工大学工学部数理情報助手．日本語手書き入力，計算機システムソフトウェアなどの研究に従事．

**高橋 延匡** (正会員)

昭和8年生．昭和32年早稲田大学第一理工学部数学科卒業．同年(株)日立製作所中央研究所入社．HITAC 5020 モニタ，TSS の開発に従事．昭和52年より東京農工大学工学部数理情報工学科教授．オペレーティング・システム，日本語情報処理，パターン認識の研究に従事．電子情報通信学会，ソフトウェア科学会，計量国語学会，ACM 各会員．理学博士．