オペレーティング・システム 32－3
（１９８６．９．１２）

ＯＳ／ｏ第２版とシステム記述言語Ｃ

屋代寛，中川正樹，高橋延匡

東京農工大学　工学部　数理情報工学科

本報告は、我々が開発を行なっているオペレーティング・システムＯＳ／ｏ第２版のアーキテクチャについて、システム記述言語Ｃ処理系ｃａｔとの関連を中心に述べている。

本報告では、特に、アーキテクチャの路線という観点から、

（１）日本語文字コード体系

（２）言語処理系、デバッグ環境、文書化などのソフトウェア開発環境

について述べている。この二つは、ＯＳ／ｏ第２版のアーキテクチャにおける重要な部分であり、いずれも自作の言語処理系ｃａｔを基盤として構成される。ＯＳ／ｏ第２版のアーキテクチャは、ｃａｔと密接な関連を持っている。

"The Architecture of OS/omicron V2.0 and Its Implementation Language C"(in Japanese)

Hiroshi YASHIRO, Masaki NAKAGAWA and Nobumasa TAKAHASHI

Department of Information Science, Faculty of Technology, Tokyo University of Agriculture and Technology,
2-24-16, Nakamachi, Koganei, Tokyo, 184 Japan

This paper describes the architecture of OS/omicron V2.0 with a translator CAT of the implementation language C for OS/omicron. OS/omicron is a real-time operating system for a super personal system of MC68000 base and CAT has been designed for OS/omicron.

In this paper, main emphasis is laid on how CAT interacts with OS/omicron in the construction of:

(1) double-byte code system for Japanese character set

(2) programming environment for debugging and documentation.

１．はじめに

計算機システムを取り巻く環境は常に変化している。特にハードウェア環境は、この十年間に目覚ましい発展を遂げている。マイクロプロセッサや半導体メモリ、 周辺 LSI のコストパフォーマンスは半導体技術の進歩によって著しく向上してきた。また、光ディスクや光ファイバなどの次世代技術も着々と実用化されてきている。同時に、計算機システムを利用するアプリケーションの多様化も顕著である。計算機システムの歴史は汎用大型化の道を歩んできたが、ハードウェア環境の革新、そしてアプリケーションの多様化によって、計算機システムは専用小型化の方向性を明確にしている。過去のしがらみにとらわれることなく成長を続けてきたワードプロセッサやパーソナルコンピュータなどの分野では、この傾向が顕著である。今こそ、時代のニーズに応じた新しい計算機システムのアーキテクチャを研究する時期を迎えている。この転換が遅れれば遅れるだけ計算機システムの社会への浸透は妨げられるであろう。

我々は、以上の認識のもとに、コストパフォーマンスの良いハードウェア資源を豊富に利用し、アプリケーション（特に日本語情報処理、パターン認識、人工知能）を意識した計算機システムの研究・開発を行なってきた。このような計算機システムを構築するため、我々は「自前の」オペレーティング・システム（以下ＯＳと記す）の開発を決意し、このＯＳをＯＳ／ｏ(omicron)と名付けた。

ＯＳ／ｏはアプリケーション指向のＯＳである。ここでいうアプリケーション指向とは、ＯＳをブラックボックス化して必要な機能だけをユーザに開放することではない。近年のアプリケーションの多様化を見れば、これら全てのアプリケーションに都合の良いようにシステムを構築することは不可能に近い。真のアプリケーション指向のＯＳでは、ＯＳをも含めた問題指向のシステム作りを目指すべきであろう。

ところが、既存のＯＳでは、「ユーザは何をするかわからない」という思想によって硬いシステムを作り上げている。しかし、何かしようとするとこのシステムの硬さゆえに、手も足もでないという現状に直面することがある。アプリケーションに少しでも本格的に取り組んだことがあるならば、こういった経験を持つことになる。そこでは、「はたして計算機がユーザのためのものか」という疑問を持つに違いない。一方、我々の思想はこれと全く逆の立場をとる。ＯＳ／ｏではパーソナル・ユースを前提としており、「ユーザが得心して行なうならば何をしても良い」という思想に基づいて設計を行なっている。もはや、ＯＳはシステムのためのシステムとしてだけでなく、アプリケーションをも含めたシステムを構築するための環境として考えられるべきではないだろうか。

２．開発方針と概要

前章で述べた設計思想から、我々はＯＳ／ｏの開発方針を以下のように定めた。

（１）人工知能の研究、文字認識の研究などへの応用を前提とする
（２）日本語情報処理に対して、標準的なアーキテクチャを考慮する
（３）時間的制御を容易にするため実記憶方式を前提とする
（４）必要最小限のＯＳの機能を実現する
（５）特殊入出力機器の接続やシステム構成を拡張・縮退した専用ＯＳの開発が容易にできるようにする
（６）ＯＳ自身の開発はシステム記述言語を用いて行ない、将来のハードウェアの拡張に備える

また、実記憶方式ではアドレス空間が広いことが有利となる。そこで、プロセッサには広大なアドレス空間（16M～4 GB)を持つモトローラ社のマイクロプロセッサM68000ファミリを採用し、現在はMC68000を使用している。 本ＯＳの機能的な特徴は次のとおりである。

（ｉ）並列処理を実現するためにマルチタスクの機能を実現すること
（ii）リロケータブル・リエントラントなプログラムを標準とすること
（iii）日本語情報処理のため、16ビットのJIS漢字コードを標準とすること
（iv）システム記述言語として言語Ｃを用いること
（ｖ）情報の統一的管理を目的としたファイル・システムを開発すること
（vi）各種浮動小数点方式を実現し、その評価を行なう環境を提供すること[11]
（vii）プログラムのROM化を考慮した制御方式を実現すること

３．言語処理系の開発[９]

先にも述べたとおり、我々はＯＳ／ｏ自身の開発をシステム記述言語Ｃで行なう方針をとった。言語Ｃを採用した理由は、システムの記述に対して十分な記述力と柔軟性を持ち、ハードウェアの提供するデータ構造をほとんどそのままアクセスできるからである。

我々は開発に使用するコンパイラを自作し、この言語Ｃコンパイラをｃａｔ（C compiler developed at Tokyo University of Agriculture and Technology）と呼んでいる。ｃａｔの開発は1982年にはじまり、ｃａｔ第１版は1985年３月に東大大型計算機センタの VAX/UNIX 上でクロスコンパイラとして稼働している[４]。しかし、ｃａｔ第１版は図１のような構成であり、以下の問題点を抱えていた[10]。

（１）保守性の問題
（２）拡張性の問題
（３）オブジェクトコードの効率

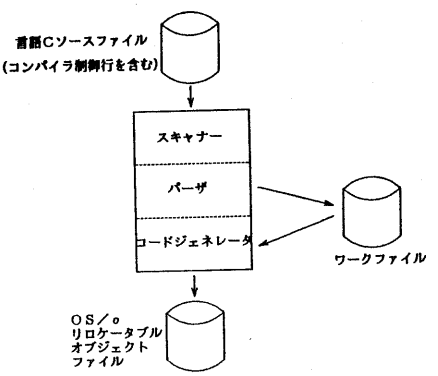

図１．ｃａｔ第１版の構成

特に、(3)のオブジェクトコードの効率の問題は、ＯＳの記述を行なう我々にとっては深刻な問題であった。ＯＳはソフトウェアの基礎を成すソフトウェアである。ＯＳの効率が悪いということは、その上に構築するソフトウェアの効率、ひいてはシステム全体の効率を落とすことにもなりかねない。

そこで、我々は上記の問題を克服するために第２版としてｃａｔを再構成（フェーズ分割化）することにした[10]。ｃａｔ第２版の構成を図２に示す。ｃａｔ第２版の方式設計においては、以下の事項を考慮した。

（ｉ）ソフトウェアの開発支援系としての位置付け

（ii）最適化の強化

本章ではＯＳの記述言語として必要不可欠な（ii）の最適化について述べ、（ｉ）の事項については後述する。

ｃａｔ第２版の最適化処理では以下の事項を考慮した。

（ａ）ソース・プログラムのレベルで最適化できない部分を最適化の処理対象とする

言語Ｃではプログラムの制御について、かなり細かく表現できる。プログラマが意識してプログラムを記述すれば、ある程度、人手による最適化が可能となる。これは、言語Ｃのシステム記述言語としての性格を示すものであり、オブジェクト・コードの生成効率が良いと言われる理由でもある。一方、言語処理系の最適化処理は FORTRAN を中心として発展してきた。 FORTRAN 流の最適化処理は、共通部分の削除、不動変数のループ外への移動等を行なうことに重点が置かれている。したがって、言語Ｃの設計思想と FORTRAN 流の最適化処理の思想には多少の隔たりがある。そこで、ｃａｔ第２版における最適化処理の対象を、プログラマが手を加えることのできない部分にすることとした。

（ｂ）最適化のレベルを分け、プロセッサ依存部分とプロセッサ独立部分の最適化を行なう。

最適化処理の種類は、プロセッサのアーキテクチャに依存しないプロセッサ独立部分と依存するプロセッサ依存部分に大きく分類される。プロセッサ独立部分の最適化処理とは、定数演算の最適化など、ターゲット・プロセッサのアーキテクチャに依存せず、必ず効果が現われる最適化処理のことを指す。一方、プロセッサ依存部分の最適化処理とはターゲット・プロセッサの性能を引き出すための最適化処理であって、他のプロセッサではその効果が保証されないものである。たとえば、MC68000 では定数を含む算術演算において、定数の値が１～８である場合、普通の演算に用いる add 、sub 命令から addi 、subi 命令にすることで、実行時間の短縮を図ることができる。しかし、インテル社の8086ではこのような命令が用意されていないため、MC68000 のような最適化の処理を施しても意味がない。これらの最適化処理を分離しておけば、他のプロセッサにコンパイラを移植する場合、プロセッサに依存しない部分を再構成しなくてすむ。すなわち、コンパイラの移植性が高くなる。

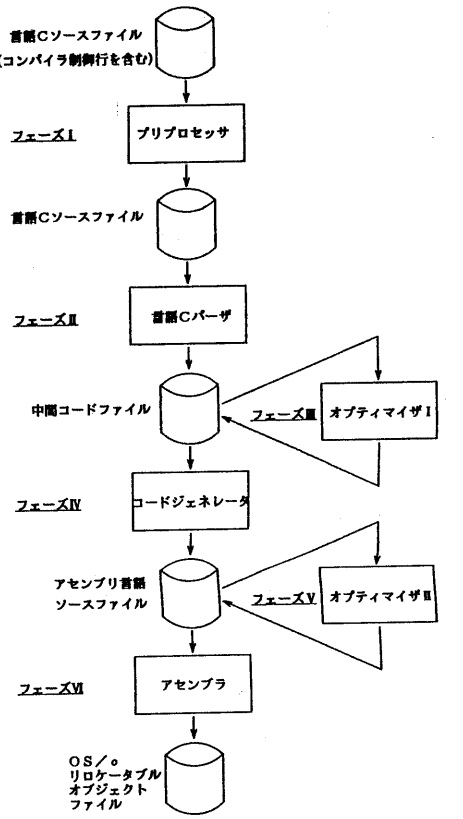

図２．ｃａｔ第２版の構成

最適化の具体的な項目を以下に示す。

①無駄な命令の削除
　（例： &a[0]　　→　a ）

②定数式の計算
　（例： #define MAX 2　, a = MAX +2 ;　　→　a = 4 ; ）

③演算の強さの軽減
　（例： a = c * 8 ;　　→　a = c << 3 ; ）

④定数の畳込み計算
　（例： #define MAX 2　, a = MAX + c + 2 ;　→　a = 4 + c ; ）

⑤MC68000 における効率の良い命令の使用

⑥MC68000 アドレッシングモードの活用

⑦分岐命令の最適化

上記のうち、①～④はプロセッサ依存部分での最適化であり、図２に示すオプティマイザＩで実現している。また、⑤～⑦についてはプロセッサ依存部分の最適化であり、⑤と⑥はコードジェネレータで実現し、⑦はオプティマイザIIで実現している。以上の最適化を行なうことによって、ｃａｔ第１版と比較して、約３分の２のオブジェクト効率（手続き部のオブジェクト・コードの大きさ）を得ることが可能となった[６][７]。

４．ＯＳ／ｏ第２版のアーキテクチャ

ＯＳ／ｏ第１版と第２版の違いは、システムの日本語化にある。したがって、本章ではアーキテクチャの路線という意味からコード体系について述べる。

ＯＳ／ｏでは日本語情報処理のための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。日本語情報処理のための標準的なアーキテクチャは文字コードの体系に集約されると言っても過言ではない。日本語の大きな特徴は英語に代表される欧米系の言語に比べて字種が非常に多いことである。既存のＯＳはそのほとんどが欧米からの輸入品であり、英字を基本とするものがほとんどである。英字をコードで表現するためには、１バイトで表現できる。しかし、日本語の文字をコード化した場合には１バイトで表現することはできない。したがって、既存のＯＳで日本語を扱えるようにするためには、モード変換やエスケープ文字を使用して１バイト・コードと２バイト・コードを混在させることを行なわなければならない。しかし、アプリケーション層ではこの方法で実現できるにせよ、システムの核部分では依然として１バイト・コードを基本としているため、システムとしての整合性は非常に悪いものとなっている。このため、日本語情報処理を行なうアプリケーションは非常に大きな負担を強いられている。ＯＳ／ｏではこれらの非本質的な問題からプログラマを解放すべく、文字コードを全２バイト・コード化する。

4.1 ＯＳ／ｏコード体系の詳細

ＯＳ／ｏでは具体的に以下のコード体系に従う。

（１）文字実体(英字、数字、ひらがな、カタカナ、漢字、記号)のコードは JIS C 6226 のコード体系を採用する

パーソナルコンピュータのレベルで広く用いられているシフトJIS は、英字や数字の半角文字は１バイト、全角文字は２バイトに割り当てている。しかし、これはJIS に反するだけでなく意味的に同じ文字を違うコードに割り当てていることになる。ＯＳ／ｏではコード体系の一貫性を重視し、一つの文字には一つのコードを割り当てる。

（２）制御コードも１バイト目に NULL $(00)_{16}$ を付加して２バイトにする

これによって、２バイト・コードのみのコード体系を構築する。１バイト、２バイト・コードの混在はコード体系の簡潔さを損うだけでなく、プログラム上における文字・文字列の扱いを困難にする。たとえば、文字列へのポインタのインクリメントは、言語Ｃでは、

```
char *p ;
++p ;
```

と記述するが、増分のバイト数は一意に決定することができなくなる。また、文字コードの語長は、文字コードの意味を調べなければならなくなる。つまり、制御コードが１バイトで表現されていると言語Ｃで

```
static char *p = "文字列\n"
```

と記述した場合には、改行符号(\n)は１バイトであるために、物理的には７バイト、論理的には４文字となり、実際には文字コードの意味を調べなければならない。

4.2 文字属性について

日本語を扱う場合には半角文字などの文字属性をどのように扱うかが問題となる。特に、全角や半角等の文字の大きさに対する情報は、美観上必要となる。また、全角、半角の違いは英語における小文字、大文字の違いとは基本的に異なる。日本語において全角や半角の違いは、あくまで外観上の違いだけである。これに対して英字における小文字と大文字の違いは、意味的な違いをも含んでいることに注意しなければならない。角の違いは、英語で言えばむしろフォントに相当する。

これら文字属性の扱いはアプリケーションに大きく依存することに注意すべきである。したがって、拡張性に富んだ方法を考慮しなければならない。

これに対処するには、次の方法が考えられる。

（１）エスケープシーケンスを用いる

JIS で規格化されている文字属性 (JIS C 6220, 6237) はエスケープコードによる状態遷移で行なわれる。たとえば、1B 5B 31 30 30 3B 35 30 20 42 というエスケープシーケンスは以降の文字を半角文字にする。この方式ではコードの２重定義が起こることはない。しかし、この方式の問題点は複雑なエスケープコードがテキスト中に混在してしまうことにある。アプリケーション・ソフトウェアは、この複雑なコードの存在を意識しなければならない。また、ＯＳやコンパイラ等のシステム・ソフトウェアでは文字コード(すなわち、意味的な部分)のみを識別できれば十分であるのにも関わらず、これらのコードを解析しなければならない。

（２）文字コードに属性を表現するコードを付加する

文字コードと文字に対する属性のコードを１セットとして扱う方式である。この方式では、文字の持つ意味的な情報と書体やサイズ等の外観上の情報を一度にコード化できる。文字の属性を使用しない場合には文字コードのみを処理し、文字の属性が必要な場合には付加した文字の属性の情報を同時に処理すれば良い。しかし、文字を４バイトで表現すると文字の属性コードを必要としない場合にも常に４バイトごとにアクセスするために無駄が多くなる。また、文字属性の表現をコード化することは、かえって表現の拡張に制限を与える。つまり、文字の属性の表現を固定長（前の例では２バイト）にすると、表現能力に限界が生じる。表現できる文字コードの数の拡張は２バイトで収まるが、文字の属性の表現では不足する可能性が高い。文字の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大きく依存している。コード体系のアーキテクチャは、非常に長いライサイクルを持つ必要がある。特に、アプリケーションや出力デバイスが多様化する傾向にあることを考慮すれば、拡張性の乏しいコード体系では満足なものとは言えない。

（３）属性情報を別ファイルで持つ

文字コードだけのファイルと文字の属性情報や書式情報等を別ファイルの形で表現する。このため、属性が必要なソフトウェアでは２つ以上のファイルを読み、その必要のないものは文字コードのファイルだけを読みこめばよい。この方式を用いれば、文字や文書の外観に関する情報から文字や文書のもつ意味的な情報を分離することが容易にできる。また、この方式では多様化するアプリケーションやハードウェアに対する拡張も容易である。たとえば、ＯＳやコンパイラ等のシステムソフトウェアでは半角文字等の煩わしい文字属性を意識せずにすみ、エディタやフォーマッタ等のアプリケーションソフトウェアでは豊富な文字属性を用いることが可能になる。

ＯＳ／ｏでは文字属性に対し、（３）の方針をとる。これによって、コード体系の拡張性や一貫性を保つことを目標としている。しかし、（３）の方法ではファイルの管理に注意を払わないと、文字コードのファイルと属性のファイルの統一性が保たれなくなる。そこで、ＯＳ／ｏ第２版では、これらのファイルを統一的に扱えるように設計を行なっている。

4.3 ＯＳの日本語化

ＯＳの核部分（メモリ管理、タスク管理、ファイル管理、入出力管理）で文字コードを扱う場合、ファイル名に代表される識別子が問題となる。ＯＳ／ｏでは、この識別子に対しては、文字の意味的な部分である文字コードのみを用いる方針である。たとえば、半角の “abc” と全角の “ａｂｃ” は同じファイル名を表現する。前にも述べたとおり、全角や半角等の違いは、あくまでも文字の外観上の違いと考える。ＯＳの核部分でこれらの外観上の情報をもつことは、かえって煩わしい。ＯＳの核部分では文字の意味的な情報を持てば良いと考えた。

前節でも述べたとおり、ＯＳ／ｏ第２版では全２バイトの文字コード体系を採用する。このため、１バイトから２バイトのＯＳを構築する場合には、その語長が変わるだけですむ。そこで、ＯＳを高水準言語で記述することによって、文字を扱う部分を抽象化すれば、コード体系の語長の影響を少なくすることが可能となる。たとえば、言語Ｃでは文字を表現する型は char 型である。文字を表現する場合に、この型を用いてシステムの記述を行なえば、言語処理系の語長を変換するだけですむ。

ここで、１バイト・コード体系のＯＳを２バイト・コード体系のＯＳに移行することを考える。

（ｉ）まず、１バイト・コードのＯＳを１バイト・コードの言語処理系で開発する。

（ii）次に、１バイト・コードの言語処理系を２バイト・コードの処理系に改造する。この改造は文字を表現する型の語長を１バイトから２バイトにし、文字定数や文字列定数も同様に２バイト・コードとして扱えるようにする。

（iii）記述したＯＳのソース・コードを１バイト・コードから２バイト・コードに変換する。

（iv）上記（３）で得られたコードを（２）で得られた言語処理系でコンパイルする。

以上で１バイト・コードＯＳから２バイト・コードＯＳ、ひいてはＮバイト・コードのＯＳを作成することが可能となる。

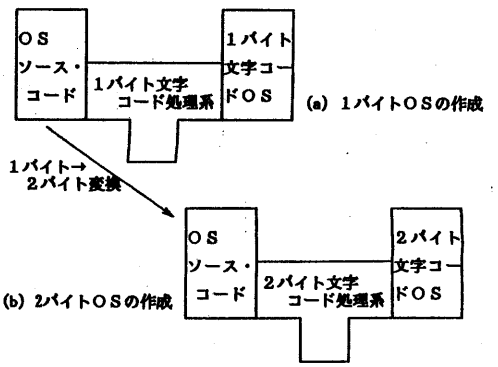

図３．ＯＳの日本語化

しかし、この場合には、プログラミングをする際に十分な注意を要する。たとえば、言語Ｃでは文字を表現する型が char 型であることは前にも述べたが、既存の計算機システムでは文字コードの語長が１バイトとなるために、１バイトを表現する型としても用いることがある。このため、１バイト型と文字型を混用させて用いているプログラムでは１バイト・コードから２バイト・コードへの変換は容易ではなくなる。これに対処するためには、プリプロセッサによるマクロ機能が有効となる。言語Ｃでは、マクロ展開のためのプリプロセッサが前提にされており、このマクロ機能を利用することで移植性の高いプログラムを記述することが可能となる。たとえば、文字を扱う型は CHAR とし、１バイトの語長を持つ型は BYTE としておく。そして、１バイト文字コードを扱う処理系に対しては、

```
#define CHAR char
#define BYTE char
```

と定義し、２バイトコードを扱う処理系に対しては、１バイトの語長を表現する型を short とすると、

```
#define CHAR char
#define BYTE short
```

と定義するのである。すなわち、ＯＳをはじめとするソフトウェアの開発者が文字という論理的な概念と語長などの物理的な概念を認識していることが必要となる。

５．ＯＳ／ｏの開発環境

ＯＳの開発では記述言語の問題も重要ではあるが、それ以上に開発環境の問題が重要なものとなる。　本章では、ＯＳ／ｏにおけるソフトウェア開発環境について、ＯＳ／ｏ第１版で生じた問題点と現在開発中であるＯＳ／ｏ第２版の開発環境について述べる。

5.1 ＯＳ／ｏ第１版の開発環境

ＯＳ／ｏ第１版は1984年８月に試作を終えている[２]。ＯＳ／ｏ第１版の開発は研究室初のＯＳ開発であり、同時開発中であった言語処理系等のＯＳ／ｏのユーティリティを動かす環境作りにその開発目的を置いていた。そのため、いくつかの問題点が露呈した。以下にその問題点を記す。

ＯＳ／ｏ第１版の開発では、開発環境としてデジタル・リサーチ社のCP/M-68Kを利用し、言語処理系もCP/M-68Kの言語Ｃコンパイラ(以下、DR-Cと記す)を利用した。しかし、このことはDR-Cのオブジェクト・コードの実行環境もサポートすることを意味していた。 つまり、ＯＳ／ｏ第１版ではＯＳ／ｏとCP/M-68Kの両者のプログラム実行環境をサポートしなければならなかった。しかし、ＯＳ／ｏとCP/M-68Kではプログラム実行環境が大きく異なっていた。具体的には、ＯＳ／ｏがアドレス・レジスタを基底とする相対アドレスでアクセスするのに対して、CP/M-68Kが絶対アドレスのアクセスを基本としていることである。プログラム実行環境はアーキテクチャをささえる基盤ともいうべきものである。ＯＳ／ｏ第１版はこの異なる両者のアーキテクチャをサポートしなければならなかったために、設計の一貫性を失うこととなった。

一方、開発環境に関しては以下の問題があった。

（１）デバッグ環境の不備

ＯＳはシステムの基盤を支えるソフトウェアである。したがって、ＯＳの開発においては、既存のＯＳ上のユーティリティのサポートを得ることが困難な場合がよく起こる。ＯＳ／ｏ第１版の開発でも、このことは大きな問題となった。具体的には、ＯＳまたはユーティリティの虫によってシステム・ダウンが引き起こされた場合、ハードウェア・リセットによってデバッガ・ボードの簡易デバッガに戻る。そして、メモリ・ダンプによってチェックを行なうという極めて原始的な方法をとらざるを得なかった。

（２）文書の不備

ソフトウェア開発においては外部仕様書、内部仕様書等の文書の占める役割が重要であることは言うまでもない。ＯＳ／ｏ第１版開発では、外部仕様書を中心とする文書作成に日本語ワードプロセッサ（以下、ワープロと略す）を利用した。しかし、プログラムの開発はCP/M-68K上で行ない、文書はワープロで作成するという環境のため、通常良く起こるプログラムと文書の対応がとれないという情況が発生した。

5.2 ＯＳ／ｏ第２版の開発環境

言語Ｃコンパイラの開発によって、ＯＳ／ｏ第１版の設計で起きたアーキテクチャ上の問題は解決された。つまり、ＯＳ／ｏのプログラム実行環境に統一することが可能となった。また、開発環境の重要性という教訓をＯＳ／ｏ第１版を開発で得たことから、我々は開発環境の整備を重視してきた。特に、言語Ｃコンパイラｃａｔ第２版の方式設計では、ソフトウェアの開発環境を考慮して設計を重視した。具体的には、

（１）ソース・コードレベルのインタプリティブ・デバッガの開発[12]

（２）機械語レベルのデバッガDEBU の開発[５]

（３）ワープロの Programmer's Work Bench(以下、PWBと記す)化[８]

（４）文書出力システム「浄書」の開発[13]

があげられる。

これら開発環境は開発工程と密接な関連を持つ。まず（１）のデバッガによって言語Ｃで記述した部分をデバッグする。このデバッガは、単体デバッグが可能である。ＯＳの機能は単機能をとってみれば単純なものである。したがって、このデバッガを使用することによって機能ごとの虫は排除できる。次に、（２）の機械語レベルのデバッガによって、

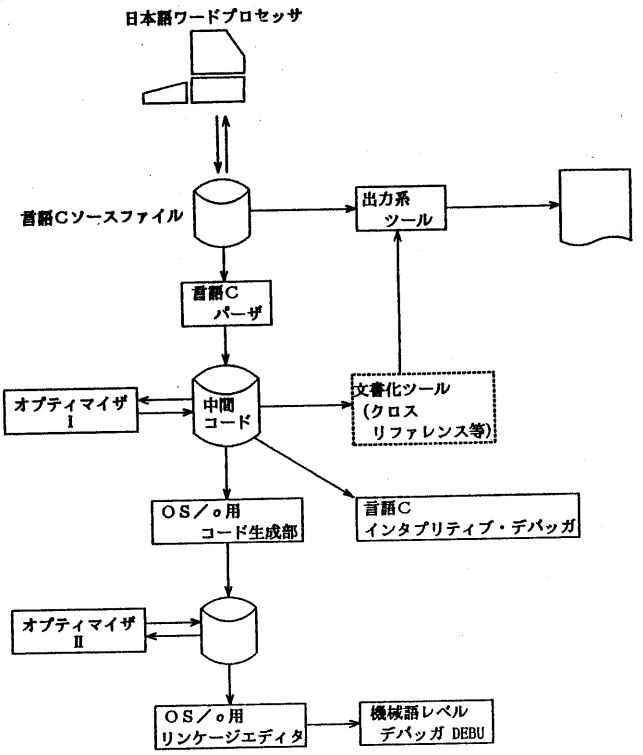

図４．ＯＳ／ｏ第２版の開発環境

全体デバッグを行なう。なお、このデバッガはＯＳをモニタするツールとして改良してある。そして文書化では、(3)のPWB化したワープロを用いる。すなわち、ワープロを用いてコーディングを行ない、コメントに日本語を入れることで、仕様書とプログラムの一体化を目指す。その他、コーリングストラクチャ・ジェネレータ等の文書化ツールも現在開発中である。そして最後に、（4）の文書出力システムを用いることによってレーザビーム・プリンタから印字品質の高い出力を得ることができる。その出力例を図５に示す。

我々は、第１版の教訓から、ＯＳのアーキテクチャがその開発環境をも含んでいることを認識した。ＯＳ／ｏではＯＳのソースコードからユーザに公開し、提供する方針である。そのため、前述した（ａ）ＯＳの開発をシステム記述言語Ｃを用いて行ない、（ｂ）最小限のＯＳの機能を提供することが重要となる。

６．おわりに

ＯＳ／ｏ第２版の文字コード体系と開発環境について述べた。この二つはＯＳ／ｏ第２版のアーキテクチャの重要な核となり、いずれもシステム記述言語Ｃ処理系ｃａｔを基盤とするものである。まさに、システム記述言語とＯＳは密接な関係にあると言える。現在、本稿で述べたアーキテクチャに基いて、ＯＳ／ｏ第２版の開発を行なっている。

参考文献

[1] 中川正樹，篠田佳博，藤森英明，高橋延匡：“MC68000ユニ＆マルチ・プロセッサ・システム用システム記述言語Ｃ処理系の開発”，情報処理学会計算機システムの制御と評価研究会資料21-7，1983.12

[2] 高橋延匡，並木美太郎，武山潤一郎，中川正樹：“ＯＳ／ｏのアーキテクチャと第１版の実現”情報処理学会オペレーティングシステム研究会資料24-11，1984.9

[3] 高橋延匡：“ＯＳ／omicron の設計思想”，情報処理学会第29回全国大会予稿集，1985.9

[4] 篠田佳博，藤森英明，中川正樹，高橋延匡：“ＯＳ／ｏのツールセット（４） ── 言語Ｃコンパイラ ──”，情報処理学会第30回全国大会予稿集，1985.3

[5] 並木美太郎，中川正樹，高橋延匡：“ＯＳ／ｏのツールセット（５） ── デバッガ DEBU ──”，情報処理学会第30回全国大会予稿集，1985.3

[6] 施清池，並木美太郎，中川正樹，高橋延匡：“ｃａｔのオプティマイザ（１） ── プロセッサ独立部分での最適化 ──”，情報処理学会第32回全国大会予稿集，1986.3

[7] 屋代寛，森岳志，並木美太郎，中川正樹，高橋延匡：“ｃａｔのオプティマイザ（２） ── プロセッサ依存部分での最適化 ──”，情報処理学会第32回全国大会予稿集，1986.3

[8] 並木美太郎，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研究会資料47-1，1986.6

[9] 屋代寛，森岳志，並木美太郎，中川正樹，高橋延匡：“ＯＳ／omicron 用言語Ｃコンパイラｃａｔの開発 ── VAX/UNIXクロスシステムからの移行 ──”，情報処理学会ソフトウェア工学研究会資料48-1，1986.6

[10] 並木美太郎，中川正樹，高橋延匡：“言語Ｃコンパイラｃａｔの方式設計”，情報処理学会ソフトウェア工学研究会資料48-2，1986.6

[11] 森岳志，中川正樹，中森眞理雄，高橋延匡：“ＯＳ／omicron における浮動小数点方式のサポート”，情報処理学会第33回全国大会予稿集，1986.10

[12] 田中泰夫，中川正樹，高橋延匡：“ＯＳ／omicron におけるデバッギング・ツール ── 言語Ｃインタプリティブ・デバッガ ──”，情報処理学会第33回全国大会予稿集，1986.10

[13] 里山元章，中川正樹，高橋延匡：“ＯＳ／omicron における文書出力システム 浄書(JOSHO)”，情報処理学会第33回全国大会予稿集，1986.10