システムソフトウェアと
オペレーティング・システム 68－2
（１９９５．３．２４）

マルチバイトコードを基本とするソフトウェア アーキテクチャと OS/omicron

並木美太郎、早川栄一、高橋延匡

東京農工大学　工学部　電子情報工学科
コンピュータサイエンスコース

本報告では、文字コードに対して一貫性を確保することを目的としたマルチバイトコードのソフトウェアアーキテクチャを提案する。本ソフトウェアアーキテクチャでは、応用プログラム、言語処理系、OS に対して、固定長マルチバイトの文字コード系を採用することにより、計算機システムの文字コードに関するアーキテクチャの一貫性を確保し、母語の利用に対して制限のないソフトウェア環境を構築する。マルチバイトコードを基本アーキテクチャとするシステムを実現するため、システム記述言語の言語処理系をマルチバイト化し、さらにマルチバイトのコンパイラを応用プログラムの開発に使用することによって、システムの文字コードに対する一貫性を実現した。本方式を用いることにより、計算機システムを ISO 10646 等のマルチバイトコードに対応できる。本方式に基づいて、漢字を表す２バイトコードをアーキテクチャとする OS を開発し、日本語情報処理のための処理基盤を作成した。

OS/omicron～Software Architecture Based on Fixed Length Multibyte Code～

Mitarou Namiki, Eiichi Hayakawa and Nobumasa Takahashi

Department of Computer Science
Tokyo University of Agriculture and Technology

This paper describes software architecture based on fixed length multibyte code, not with the objective of compatibility with conventional single byte code systems, but aiming to maintain character code consistency in every layer of the computer software. This architecture, by adopting fixed length multibyte character code system for application programs, language processor and operating systems, preserves the consistency of the character codes in the computer systems and implements a software environment that has no restrictions on the characters used.

Character code consistency was archived by converting a type of character in the system description language to use fixed length multibyte code in the development of operating systems. This method can enable computer systems to handle multibyte codes such as ISO 10646. Using this method, we have developed an OS with two byte code architecture to process Kanji characters. This OS has a structure that makes it possible to add resouce management needed to process/input/output native languages.

1．はじめに

英語を母語としない文化圏において計算機の利用が広まったことにより、多国語処理の必要性が生じた。近年では、世界の言語の文字コードを定めようという ISO 10646 文字コード規格や、プログラミング言語においてマルチバイトコードの文字を扱う機能を規格に組み入れるなど、英語以外の言語で表現されたテキストを処理する基盤は整いつつある。

計算機内で、英語の文化圏以外の言語を扱う場合、その計算機が使用される文化の母語に関する文字コードを計算機内で定義しなくてはならない。英語を母語とする文化圏の計算機では、文字を ISO 646(ASCII) や EBCDIC に代表される１バイトコードで表現していた。一方、世界中の言語の文字をコードとして表現したとき、例えば、アジア系の言語である漢字をコード化するためには２バイトを必要とする。漢字の言語圏の共通の文字コードとして提案された C/J/K 文字コードは、２バイトで一つの漢字を表現する。したがって、計算機の母語化を行うための第１ステップとして、計算機システム内でマルチバイトの文字コードを扱える必要がある。

互換性を保ちながらマルチバイトの文字コードを計算機システム内で扱う方法として、従来の１バイトコードにマルチバイトコードを付加した文字コード系を利用する方法が考えられる。例えば、UNIX ではシステム全体を１バイトコードのアーキテクチャを基本とし、プログラミング言語では基本文字コードに加えて各国語を表現するためのマルチバイトコードの文字型を定義している。このような母語化の方法は、システム全体の文字コードの一貫性を阻害し、文字コードを区別してプログラミングを行う必要が生じる。従来の１バイトコードのソフトウェアを利用できるが、母語にあわせてすべてのソフトウェアを変更しなければ、１バイトのコードしか処理できない。

入出力の観点からは、母語の入力や出力を行うための機能をシステムに付加する必要が生じるという問題がある。特殊装置や入出力のための手続きとデータを、母語に合わせ容易にシステムに追加できれば、システム構築の労力を減らすことができる。

本報告では、従来の１バイトシステムに対する互換性を目的とするのではなく、計算機システムのすべての階層において、文字コードに対して一貫性を確保することを目的とし、その解決方法としてマルチバイトコードのソフトウェアアーキテクチャを提案する。本ソフトウェアアーキテクチャでは、応用プログラム、言語処理系、OS に対して、固定長マルチバイトの文字コード系を採用することにより、計算機システムの文字コードに関するアーキテクチャの一貫性を確保し、母語の利用に対して制限のないソフトウェア環境を構築する。マルチバイトコードを基本アーキテクチャとするシステムを実現するため、システム記述言語の言語処理系をマルチバイト化し、さらにマルチバイトのコンパイラを応用プログラムの開発に使用することによって、システムの文字コードに対する一貫性を実現した。本方式を用いることにより、計算機システムを ISO 10646 等のマルチバイトコードに対応できる。本方式に基づいて、漢字を表す２バイトコードをアーキテクチャとする OS を開発した。さらに、本 OS は、母語を処理するための資源管理を OS に追加できる機構を有している。

2．本研究におけるシステムの設計思想

母語を計算機で処理するため、次に示すシステムの設計思想を定めた。

2．1　固定長マルチバイトコードを基本とするソフトウェアアーキテクチャ

計算機システムの歴史を振り返ると、IBM の STRETCH において当時入出力装置や言語処理系でそれぞれ異なった文字コードを採用していたことから、システム設計の段階で全体の整合性をとれるように、システム全体で統一的な文字コードを定めた経緯がある。マル

チバイトコードをシステムに導入するときも、同様の問題が生じる。

まず、第1の問題として、どのようなマルチバイトの文字コード系を選択するかが重要である。母語処理のために、計算機システム内でマルチバイトコードを扱う方法がいくつか考案された。例えば、日本においては、従来の1バイトコード(JIS X0202)と2バイトコード(JIS X0208)を混在した文字コード集合を定義し、互換性を確保しながら日本語を情報処理する計算機システムを構築してきた。1バイトコードと2バイトコードの区別は、エスケープ・シーケンスで区別する方法、文字コード中の未使用ビット(例えば MSB)をコード系の判別に利用する方法が考案された。

いずれの方法にせよ、1バイトコードと2バイトコードの混在した文字コード系で表現されたテキストを処理すると、プログラミングにおいて1バイト文字か2バイト文字かを判別する処理が必要となる。すべてのソフトウェアに対してマルチバイトコードの判別処理を追加する必要が生じ、ソフトウェアの変更が必要となる。

第2の問題として、システム内での文字コードの一貫性という本質的な問題がある。まず、計算機システム内のすべての文字に関するマルチバイトコードが利用可能であることが母語を計算機処理できるようにするための要件である。例えば、母語処理として日本語を追加した計算機システムでは、ある応用プログラムでマルチバイトコードを扱えるが、OS の管理する資源の名前(例えばファイル名)に漢字を扱えない、制限が課せられるといったシステム内で利用できる漢字コードの利用の一貫性を確保できない等の問題が生じた。システム上のすべてのソフトウェアがマルチバイトコード対応となっていてほしいが、現実は一部分のソフトウェアにおいてマルチバイトコード対応となり、利用者に混乱をきたしている。

図1に示すように計算機システムに現れる文字コードのすべてにマルチバイトコードの文字を利用できるかは、本質的にはシステム全体の文字コード系を単一にするか、複数のコード系を許すかに帰着する。

もし、二つ以上の文字コード系を有するならば、異なるコード系を扱うソフトウェアのすべてにおいて、コード系の判別と変換処理を行う必要が生じる。UNIX では、各国語のマルチバイトコードとして EUC という文字コードを標準にしている。OS への文字コードやファイルの内容は1バイトコードとマルチバイトコードの混在コード系を定義し、プログラミング言語においては従来の1バイト文字コード以外に固定長マルチバイトコードの文字型と文字列表記を規定した結果、文字コード系の一貫性は確保されていない。例えば、文字コード変換を行わないと、マルチバイト固定長を表す wchar_t 型の文字列を open システムコールのファイル名にわたせないなどの問題が生じる。ウィンドウシステムのようなサブシステムにおいて、内部に固定長のマルチバイトコードを採用しても、OS のコード系が固定長のマルチバイトコードでなければ、同様の問題が生じる。

別の解決方法としては、プログラミングにおいてマルチバイトコードの処理をプリプロセッサを用いて、1バイトコードの処理に変換する方法も報告されている。しかし、この方式を用いても文字コードの変換処理は必要となる。マルチバイトコードに依存する処理を抽象データ型やオブジェクト指向プログラミングを用いて、追加定義する手法も考えうる。しかし、この場合においても型やメソッドの指定の名前にマルチバイトコードを利用できるかが問題となる。結局、システム内で統一的な文字コードを採用するか、常に文字コードの変換を伴うことを前提とし変換処理をどのようにプログラマやユーザに意識させないようにするかの問題である。

システムアーキテクチャの観点からは、すべての文字コードが同一であることが、文字列操作の観点からは、固定長のマルチバイトコードを採用することが望ましい。そこで、

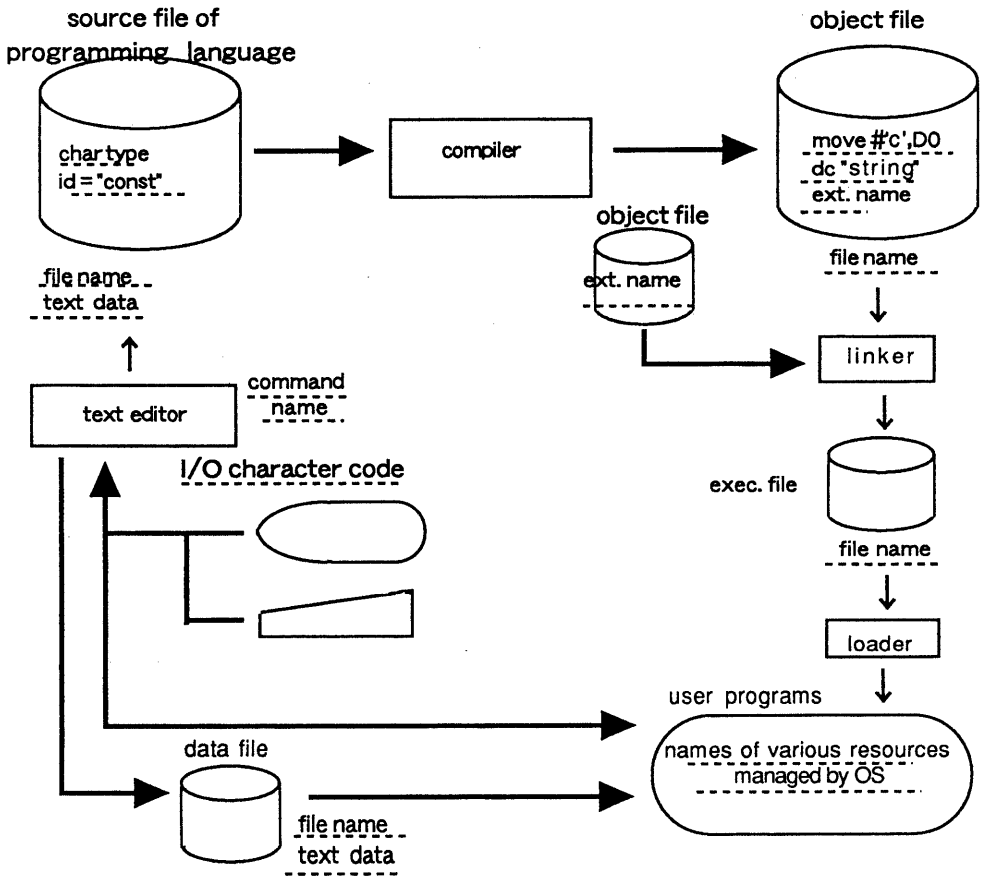

図１．計算機システム内での文字コード

本研究ではシステムの内部文字コードとして、固定長のマルチバイトコードを採用する。特に、システムアーキテクチャとして、OS を固定長マルチバイトコードとすることにより、文字コードの一貫性を確保することが本研究の特色である。

OS を固定長マルチバイトコードにすることにより、バイナリレベルの互換性は達成できない。筆者らの立場は、互換性に価値を見いだすのではなく、システムアーキテクチャとして固定長マルチバイトコードを採用することの有効性と限界を確認することが重要と考える。

２．２　拡張可能な OS 核とすること

母語処理では、文字の入出力が問題となる。英語のアルファベットは、字種が少ないので QWERTY に代表されるようなキーボードを入力装置として利用できた。しかし、字種の多い言語、例えば漢字では数千文字におよぶ文字をキーボードで入力するには困難をともなう。日本語では仮名漢字変換と呼ぶ方法を用い、仮名という表音文字を辞書を索引しながら漢字という表意文字に変換する。出力に対しても、数千文字の文字フォントを管理し、描画しなくてはならない。

このように、母語処理では入出力などに特殊な処理が必要となる。計算機システムを日本語化したとき、入力に対しては、OS を直接変更し OS 内部に変換処理を組込んだ方法に

より仮名漢字変換を組込んだ例もある。しかし、この方法は、OS のバージョンアップごとに変更を施さなくてはならない、ファイルなど OS の管理する資源の利用が制限されるなどの問題がある。

多くのシステムで locale と呼ばれる文化固有の処理を指定する方法が採用された。この指定により、通貨記号、時刻表示の形式、文字コードに対応する文字フォントを切換えることはできる。しかし、文字の入出力のように、新たな資源管理を行う必要のある処理では、その手続きをどのようにシステムへ登録し、プログラマに提供するかが問題となる。

本システムでは、拡張可能な基本的な OS を用意し、母語の処理に関連する資源管理を拡張された OS 機能として提供する。特に、OS の他の資源管理方式と一貫性を維持しながら、母語固有の処理を OS のソースコードを直接変更することなく追加する拡張方式を考察した。

3．母語を処理するための OS の設計

3．1　内部コード

母語を計算機処理するために、ソフトウェアで用いる文字コードを OS から応用プログラムまですべて同一の文字コードとする。本研究では、例として日本語を対象とする。その結果、文字コードとして日本国内の標準規格である JIS X0208 をシステム内部の文字コードとした。JIS X0208 は、２バイトの文字コード系である。

JIS X0208 をシステム内部のすべての文字コード、ファイル名などの OS の管理する資源名、テキストファイルの内容、プログラミング言語の文字型に採用する。システム内部の文字コードはすべて２バイトの固定長とし、１バイトと２バイトの混在を許さない。改行符号などの制御コードについては、１バイトの NUL(0) を前置することにより固定長の２バイトコードとする。

従来の１バイトコード、例えば、ASCII コードで定義されている文字に対しては、１バイトの ASCII コードに NUL を前置して２バイトにする方法も考えられる。しかし、JIS X0208 は ASCII コード集合中の文字をすべて含み、ASCII コードとは異なる文字コードでそれらの文字を表現していることから、同一文字に対して二つの文字コードが割り当てられてしまい、一貫性を確保できない。そこで、ASCII コードに定義されている文字については、JIS X0208 の文字コードを利用することとした。

以上により、文字コードと文字コードを表す記憶領域の大きさに対して一貫性を確保できる。なお、JIS X0208 は ASCII コードの文字を含んでいることから、英語と日本語の２ヶ国語を処理できる。同じ２バイト規格の C/J/K を用い、同様の方法で内部コードを定義すれば、中国語、日本語、韓国語を処理できる。

3．2　マルチバイトコード OS の構築手法

固定長のマルチバイトコードを内部コードとするシステムを開発する一つの方法は、従来の１バイトコードの計算機上の言語処理系を用いることである。例えば、先の UNIX の例では、目標とする固定長マルチバイトコードの OS や応用プログラムの文字に関する処理を、マルチバイトコードの文字型である wchar_t 型(typedef によって定義されている型)を用いてコーディングする方法がある。だが、この方法では、常に１バイトコードのシステム上で母語処理を行うソフトウェアを開発しなくてはならない。母語を計算機処理するソフトウェアの開発者にとって、母語をソフトウェア開発の全フェーズに利用できた方が便利である。ソフトウェアの文書化だけでなく、コーディングに対して、例えばプログラミング言語の識別子等に母語を用いる効果は大きいと考える。

このようなことから、単に OS だけをマルチバイトコード化するのではなく、プログラ

ミング環境全体で固定長マルチバイトコードを扱う。特に、言語処理系の基本的な文字型(例えば言語Cでは char 型)をマルチバイトコードとすることによって、OS 開発、マルチバイトコード OS 上のプログラミング環境におけるマルチバイトコードの使用を標準とする。OS の記述言語、その OS 上での応用プログラムの記述言語が同じであれば、その言語処理系をマルチバイトコードにするだけで、文字コードの一貫性を確保できる。

本研究のマルチバイトコードシステムでは、OS 、応用プログラムを言語Cで記述する。言語C処理系も言語Cで記述する。1バイトコードのシステムにおいて、言語Cにおける文字型 char は通常1バイトの記憶領域が割り当てられる。本研究では、char 型と文字定数を2バイトとし、2バイト固定長のソースコードを入力とするような言語Cコンパイラを、従来の1バイトコードシステム上に開発し、このコンパイラにより OS をコンパイルすることにより2バイトコードの OS を開発する。OS 完成後、このコンパイラを OS 上へ移植することにより、2バイトコードのプログラミング環境を実現した。

固定長マルチバイトコードの OS と計算機システムを構築する手順を次に示す。

手順1：1バイトコードのシステム上でシステム記述言語を作成する

1バイトコードのシステム上で、言語Cコンパイラを言語Cで作成する。このコンパイラは、char 型の大きさ、ソースコードともに1バイトコードとなっている。

手順2：1バイトコードのシステム上でマルチバイトコードのクロスコンパイラを作成する

手順1のコンパイラの型定義表などを変更することにより、固定長マルチバイトコードのクロスコンパイラを作成する。このクロスコンパイラの char 型とソースコードは、マルチバイトになっている。

手順3：マルチバイトのクロスコンパイラにより、OS をコンパイルする

手順2：のクロスコンパイラにより、OS をコンパイルする(クロスコンパイラの入力はマルチバイトなので、1バイトから固定長マルチバイトへのコード変換を行いながらコンパイルを行う)。この結果、文字コードがマルチバイトの OS が作成される。

同様に、このクロスコンパイラで、クロスコンパイラをコンパイルすることにより、マルチバイトのセルフコンパイラが作成される。マルチバイトの OS とマルチバイトのセルフコンパイラにより、固定長マルチバイトコードのプログラミング環境が実現される。こうして作成されたシステムにおける、入出力、ファイルの内容、OS へのインタフェースなどの文字コードは、すべて固定長のマルチバイトコードとなっている。

手順4：マルチバイトの OS 上でマルチバイトの言語処理系を実行し、応用プログラムを開発する

1バイトコード上にある必要なプログラムのソースコードを、コード変換しながら、マルチバイトコードのシステムへ転送し、コンパイルすることにより1バイトコードの環境下で作成したソフトウェアをマルチバイトコード環境下で実行できる。

本方式により、OS から応用プログラムまで、システムを体系的にマルチバイト化できる。従来のシステムにおけるマルチバイトコード化は、個々のプログラムごとにマルチバイトコードの処理を追加・変更しなくてはならなかった。その結果、プログラムに対してプログラマが変更を加えるために労力を必要とし、すべてのプログラムでマルチバイトコードを扱えない等の問題が発生する。本方式を用いれば、プログラムが型のサイズに強く依存

していないという条件の下に、プログラマが変更を加えることなしに、ソースコードレベルの互換性を確保しながらシステムをマルチバイトコード化できる。なお、本方式により、ISO 10646 の4バイトコードのコンパイラも実現されている。

4．実現

本研究における OS の実現は、マルチバイトコード化に対して1バイトコードに起因するエラーを排除するため、OS、言語Cコンパイラ等のシステムソフトウェアをすべて独自に開発した。

本研究における OS は、1981年より設計を開始し、1984年に初版が稼動した。本報告で述べた特長を有する版は、1987年に稼動し、日立製作所製の2050ワークステーション、VME バスシステム上で、自然言語処理、パタン認識の研究・開発に利用している。

固定長マルチバイトコードのシステム上で、母語を扱う30万行のソフトウェアが開発された。本システム上で開発したことにより、文字コードの違いを区別しないでテキスト処理を行えた。また、母語を用いて識別子を記述したことにより、ソースコードの読みやすさが、50 % 向上したなどの効果を得ることができた。

5．議論

本システムについての議論を次に示す。

質問1．「システム全体を固定長マルチバイトコードにするより、混在コードのシステムの上で、従来の1バイトコードで動いているソフトウェアを母語化した方が品質のよいソフトウェアができ、コストを低くできる」

混在コード系のシステムで、既存の1バイトコード系で作成されたソフトウェアをマルチバイトコードに変更することは、多くの手間を必要とする。ソフトウェア全体を読み、文字に関係する部分において1バイトと2バイトコードの区別を行うように処理を変更することは、膨大な変更量におよび、変更の過程で新たなバグが混入する。また、必ずしも既存のソフトウェアを母語化することでソフトウェアが早く安く開発できるわけではない。卓上電子出版における組版ソフトウェアのように、文化に深く起因する処理は単なる文字コードを判別するような処理ではない。既存のソフトウェアを変更するよりは、母語を処理しやすいシステム上で新規に開発した方が良質のソフトウェアを開発できるケースもある。

質問2．「固定長マルチバイトコードを内部コードとすることにより、2倍の記憶領域を必要とするのではないか」

確かに、従来の1バイトコードで表現されていたデータを固定長2バイドコードにすれば、2倍の記憶容量を必要とする。しかし、1バイトコードで表現された内容を母語で表現しても、必ずしも2倍になるとは限らない。例えば、英語で表現された文書を日本語に翻訳し、固定長2バイトコードで表現したところファイルサイズは、約 10 % 増加しただけである。したがって、母語を中心に処理するシステムにおいて、母語に対応する固定長の文字コードを採用しても、記憶領域の浪費は起こらない。もし、1バイトコードを中心に処理する必要があるのならば、1バイトコードを文字コードのアーキテクチャとすべきである。

力説したいことは、1バイトコードとの互換性・親和性を目的に母語を処理するシステム構築を不利にするような設計をするべきでない、個々の文化に応じて最適のシステムを構築できるようにすることが重要、というである。

質問３．「このシステムは母語を処理するが、日本人だけに有利なのではないのか」

本論文では、具体的に日本語を扱うシステムを実現した。しかし、固定長マルチバイトコードを構築する手法は、日本語だけでなく他の言語においても適用できる。また、本システムは、２バイトコードであることから、文字フォントさえ準備できれば、C/J/K コードを用いることにより、中国語や韓国語にも利用できる。ISO 10646 を文字コード系とし本方式を用いてシステムを構築すれば、全世界中の言語をシステム上で処理できる。実際、筆者らは、その準備として ISO 10646 用の OS を開発するための固定長４バイトコードのコンパイラを作成した。

６．おわりに

本論文では、固定長マルチバイトコードを文字コードとし、システム全体の文字に関する一貫性を有する計算機システムのアークテクチャとその実現方法について述べた。本システムでは、OS で扱う文字コードを固定長のマルチバイトコードにすることによって、システム内の文字コードに対する一貫性を確保した。固定長のマルチバイトコードを文字型とするコンパイラによって本システムを開発した結果、体系的に固定長のマルチバイトコードのシステムを実現することができた。

本研究における OS を母語を用いたテキスト処理を行う応用プログラム開発に利用したところ、文字コードの種類を意識することのないコーディングを行えただけでなく、プログラミング言語の識別子等に母語を利用することによりソフトウェアの読みやすさを向上させる利点を得ることができた。

参考文献

[1] “Universal Multiple-octet Coded Character Set(UCS)”, ISO/IEC 10646, 1992.

[2] W.Buchholz et al., “Planning a Computer System”, Mcgraw-Hill, 1962.

[3] “ATT UNIX System V R4, Internationalization”, ATT, 1990.

[4] K.Lunde, “Understanding Japanese Information Processing”, O'Reilly & Assoc., 1993.

[5] 中田育男, “プログラム言語における日本語化の現状と今後の方向”, 情報処理学会 PL研究会資料, 16-6, 1988.

[6] 中村他, “日本語 CLU システムの試作”, 情報処理学会第30回全国大会, 1R-7, pp.461-462, 1985.

[7] 国枝他, “PASCAL をベースとした日本語データ及び漢字を用いるプログラミング”, 情報処理学会第21回全国大会, 5I-3, pp.1015-1016, 1980.

[8] 高橋延匡, “研究プロジェクト総説: OS/omicron の開発”, 情報処理学会OS研究会資料, 39-5, 1988.

[9] 並木他, “OS/omicron 用システム記述言語C処理系Catのソフトウェア工学的見地からの方式設計”, 電子情報通信学会論文誌, Vol.J71-D, No.4, pp.652-660, 1988.

[10] 並木他, “マルチプロセッサシステム向けの OS/omicron タスク管理の設計と実現”, 情報処理学会論文誌, Vol.31, No.6, pp.894-905, 1990.

[11] 中川他, “母語プログラミングの理念, 実現, 実践とその効果”, 電子情報通信学会論文誌 D-I, Vol.J-77-D-I, No.5, pp.364-374, 1994.

[12] 並木他, “言語Ｃコンパイラのマルチバイト化の実現方式, 情報処理学会論文誌, Vol.33, No.11, pp.1331-1340, 1992.