システムソフトウェアと
オペレーティング・システム 67－16
(1994. 12. 9)

並列システムソフトウェアのためのプログラム開発環境

早川栄一，下山朋彦*，並木美太郎，高橋延匡
東京農工大学工学部
* キヤノン株式会社
東京都小金井市中町 2-24-16
E-mail: hayakawa@cc.tuat.ac.jp

あらまし 本稿では、並列システムのシステムソフトウェアのための開発環境について述べる．アーキテクチャの多様化に伴って、システムソフトウェアを容易に構築できる環境が必要となっている．これに対して、OS などのシステムソフトウェアの開発時に着目すべき対象について考察を行った．特に，開発の連続性や可視化の必要性に着目し，これらをサポートする開発環境の設計を行った．特徴としては，命令レベルシミュレータ，仮想マシン，OS シミュレータの３つのシミュレータから構成する環境を提供する．さらに，いずれの環境下でも可視化環境を提供し，システムプログラミングに対しても直観的な動作の理解を可能にする．

和文キーワード システムソフトウェア，オペレーティング・システム，並列プログラミング

A Programming Environment for System Softwares in Parallel Computing

Eiichi Hayakawa, Tomohiko Shimoyama*, Mitarou Namiki and Nobumasa Takahashi
Tokyo University of Agriculture and Technology
* Canon Inc.
2-24-16, Naka-cho, Koganei, Tokyo 184 JAPAN

Abstract: This paper describes a programming environment for system software in parallel computing. With increasing hardware architectural variation, an environment providing ease of programming for system software is required. From this need, the focus of interest was considered at the stage of developing system software, such as operating system(OS)s, and an environment was designed to support the continuity of development at every developmental phase, as well as supporting the visualization of system programs. This environment consists of simulators on three separate levels—that of instruction, virtual machine(interrupt, privileged instruction) and OS SVC(supervisor call). Also, a visualization environment was developed on these simulators to support the understanding of parallel program behavior and assist in their programming.

英文 key words system software, operating system, parallel programming

1 まえがき

近年の半導体技術や実装技術などのハードウェア技術の進歩は，画像処理用システムなどの目的別の並列アーキテクチャの構築を可能にしつつある．このような時代には，計算機システム自身も，目的に応じて多様化の方向へ向かっている．このような状況の中で，OS などの計算機資源の管理を行う部分であるシステムソフトウェアを並列処理に応じた形で作成する方法が問題となる．

システムソフトウェアは，常にオーバヘッドと抽象化とのトレードオフの上に成立する．特に，並列システムのように，並列化による速度向上を目標としたシステムでは，システムソフトウェアのオーバヘッドをいかに減らすか，そのうえでアプリケーションプログラムに対していかにプログラミングが容易なインタフェースを提供できるか，が重要な点である．

このことから，並列システムのシステムソフトウェアは，通常のシステム以上に計算機アーキテクチャとの相互作用が大きい．そのために，システムの開発環境を作成する場合，この点を支援できることが望まれている．

また，並列アプリケーションに対しては，シミュレータ，デバッガ，可視化ツールをはじめとする開発ツールが整備されつつあるが，システムソフトウェアに対しては，これらのツールは現在未整備な状態であり，開発者の負担が大きい．しかし，並列のアーキテクチャを容易に構築できるような時代になれば，それに合ったシステムも多様化してくることから，アプリケーションと同等，もしくはそれ以上のデバッグ環境の提供が必要になってくる．

東京農工大学工学部電子情報工学科 高橋・並木研究室では，辞書検索，構文解析，囲碁対局，文字認識，ＤＴＰなどでのレイアウト，フォント展開などを並列処理の対象として，これらを実行するためのOSと言語処理系の研究を行ってきた．実際に，四つのOSと三つの言語C処理系を作成してきたが，OSを含む並列システムソフトウェアの開発効率の向上，特にデバッグ効率の点から，プログラム開発環境の必要性を感じた．

並列プログラムやシステムソフトウェアに対する開発環境やツールはあるが [1] が，単体で実現されたものであり，開発フェーズの中で連続的に用いるのは難しい．また，並列プログラミングで重要な役割を果たす可視化についても，デバッガやプロファイラの一部として実装しているものはあるが [2], [3], [4], むしろOSやサーバなどのシステム開発に必要な可視化を行ってほしい．

このような問題に対して，我々は並列システムプログラム，特にOSの開発に着目した開発環境の構築を目的に設定した．

本稿では，我々がこれまで作成してきたOSの開発時の要求の分析，OSをはじめとする並列システムソフトウェアの開発のためのプログラム開発環境の構想と，実現したいくつかの環境について述べる．

2 OS/omicron プロジェクトと並列処理

2.1 対象としているアプリケーション

まず，我々の並列処理に対するスタンスについて述べる．我々の研究ユニットでは，OS/omicron [5] と呼ぶ並列処理の研究プロジェクトを行なっている．そこでは，次のアプリケーションを対象としている．

- 日本語文書作成支援（形態素解析，構文解析）
  並列構文解析，並列辞書検索，
- パターン認識
  並列辞書検索
- 卓上電子出版
  紙面レイアウト，アウトラインフォント処理
- 囲碁対局システム
  複数の評価関数を用いた局面評価，候補手探索

これらは，C言語の関数程度を並列の実行単位とした中粒度の並列性を持っている．中粒度の並列性の特徴は，次のとおりである．

(a) 共有データを持ち，相互に同期を取るタスクが生成される．

(b) 処理フェーズを分割することができる．その間で，数10 kバイトから数 M バイトのデータ転送が発生する．

(c) 並列処理の粒度はC言語の関数程度である．

これらの特徴から，コンパイラなどによる自動化が難しく，むしろ，プログラマに対して並列化を支援する環境を提供することが望まれている．また，資源管理を行うOSの存在が重要になる．並列プログラミングを容易にする

このような問題に対して，OS/omicron プロジェクトでは，まず作業の実行単位であるタスクと，資源を共有し目的のために実行するタスクの集合であるタスクフォース [6] の概念を提案した．そして，このモデルに基づき，我々は OS/omicron V2 [7], V3 [9], OS/礎 [8] という二つのOSを開発してきた．これらはいずれも，実記憶上でのリロケータブル，リエントラントな環境を実現する実行環境を備えている．この実行環境を提供する言語Cコンパイラ CAT も作成した．システムはいずれも，CAT で記述されている．

次に，この二つのOSの開発について述べる．

2.2 OS/omicron V2

OS/omicron V2(以下，V2)は，マトリクススイッチ型のバス結合を持ったアーキテクチャを対象にしたOSである．V2の特徴としては，並列プロセッサで効率よく実行可能にするために，OSやサーバ自体を複数のスレッドで実行するモデルで実現したこと，非同期プログラミングを容易にするために，割込みをタスクの状態遷移で仮想化していることである．

V2作成時にはマトリクススイッチ型のマルチプロセッサマシンがないことから，仮想マシンによるデバッグを主に行った．ハイパOS「江戸」と呼ぶ仮想マシン環境を作成し，その上でCP/M-68KとV2という二つのOSを立ち上げ，CP/M-68Kのデバッグ機能，さらに「江戸」が提供するアクセスブレークポイントなどの機能を用いてデバッグを行った．

2.3 OS/omicron V3

V3では，実際に4台のPEを持つ，共有メモリ型のマルチプロセッサシステムをVMEシステム上に作成し，この上での実装を行った．V2と比較すると，カーネル＋サーバ群という形態で構成されていること，カーネル自身は割り込まれずに実行すること，4台のプロセッサの役割がいずれも均質であることが特徴である．

V3は，まず，単一PEのマシン上で動作確認を行った．次に，共有メモリ型のマルチプロセッサの各PEごとに仮想マシンを実行させた．仮想マシンは，不均質なI/Oの均質化と，デバッグの機能を提供するようにした．

2.4 OS/礎

OS/礎[8]は，システムの構築を柔軟にするためのマイクロカーネルである．ターゲットは，共有メモリ型のマルチプロセッサシステムである．モジュール間インタフェースにはメッセージ通信を用い，特化されたプロセッサの集合としてモデル化した．実際に，プリントサーバにおいて，LBP制御，ページ記述言語の処理，フォント展開の専用プロセッサを制御するOSとして実装した．

3 システムソフトウェア開発での要求

我々は，上記OSの開発を行い，さらにこれらの上で並列ソート，並列辞書検索，Earleyの構文解析アルゴリズムの並列化などを行った．

次に，これらの経験から得た，システムソフトウェア開発への要求について述べる．

(1) ハードウェア設定の容易な変更

プロセッサエレメント(PE)とメモリユニット(MU)との結合状態や，キャッシュの構造などはシステムプログラムの構造，性能に大きな影響を与える．例えば，バスアクセスの競合状態，キャッシュへのアクセスパターン，I/Oの実行時間などがある．特に，共有メモリ型のシステムではメモリへのアクセスパターンを意識する必要がある．

PE数，メモリ数とOSとの関係，さらに，バスの使用率，キャッシュのヒット率などが変更と同時に，計測できる環境が必要になる．

(2) 記憶階層へのアクセスの情報

実機が完成していない，もしくは存在しない環境下で，記憶階層へのアクセスを監視するには，ハードウェアシミュレータによって，ハードウェアの動作をシミュレートするアプローチがある．

このシミュレータには，論理レベルのもの，命令レベルのものが存在する．システムプログラム，特に，中粒度の並列性を対象としたシステムでは，命令レベルでのシミュレータでも，かなり有効なデータを取ることができる．例えば，メモリの競合状態や，OSの同期命令などは命令レベルでも取ることができる．

システムプログラムは資源管理のための表を持つことが多い．この表の実装方法が性能向上の一つの鍵となる．その一方で，内容の一貫性を管理しなければならない．特に，性能をあげるためにマルチスレッド化したシステムの場合，一貫性の管理やデバッグが難しくなる．

(3) 開発対象による着目点の変更

開発のある局面では，メモリアクセスの状態までチェックしたいことがある．その一方で，プログラムが動作，OSのSVCレベルでのチェックですむ場合，ここまでチェックするのは実行速度の低下を招く．開発者が欲しいデータに応じて，開発環境が提供する機能を変えることで開発効率をあげたい．

(4) 環境の連続性

開発の進度，目的に応じた環境を用意したい．その間はできるかぎり連続した環境とするほうが望ましい．これは，プログラムの変更が少なくなり，開発時のバグの混入を避けることができるためである．

(5) システムの可視化機構の提供

並列化プログラムでは，その実行状態をチェックプリントなどで把握することは難しい．より，わかりやすい形で表示する必要がある．プログラムの実行を含めた可視化を行うことで，動作の理解，デバッグ，プログラムの直感的な評価が可能になる．また，定量的な評価を助けるため

に，ログデータなどの統計データのグラフ化なども有効である．

このような環境は，従来，可視化はアプリケーションに対して行われてきた．その一方で，システムプログラムに対しては貧弱な環境しか提供していない．しかし，サーバ構成によるOSの機能分化，システムプログラムの多様化に応じて，システムプログラムもこのような直感的な可視化の機構が必要になる．プログラムに対する直感的な把握は，プログラムの動作を理解するのに必要である．

システムプログラムの実行状態の制御

並列アプリケーションと同様に，OSやサーバ自身も並列に動作する場合，これらについても非決定性が生じることから，次のような要求が生じる．

- バグの再現性の確保
- プローブ効果の排除
- 同期命令でのブレークポイント

4 設計方針

上記の考察から，我々は並列システムプログラムのためのプログラム開発環境の実現が必要であると考えた．この開発環境の設計方針は次のとおりである．

(1) 開発フェーズに応じた複数の環境を用意すること

(2) 複数の環境間の連続性を確保すること

(3) 可視化環境を用意すること

以下，この設計方針に基づいた，開発システムについて述べる．

5 開発環境

5.1 開発環境の特徴

本開発環境の特徴は次のとおりである．

(1) 開発目的に応じたシミュレータを用意する

システムプログラムの場合，実機の開発が間に合わないことがあるために，実機上での開発は難しい．また，開発者側が実機の動作をある程度制御できるほうが，デバッグやテストの段階では望ましい．そこで，本システムでは，3つのシミュレータから開発環境を構築する．デバッグの対象とする粒度に合わせて3つの環境を用意する．

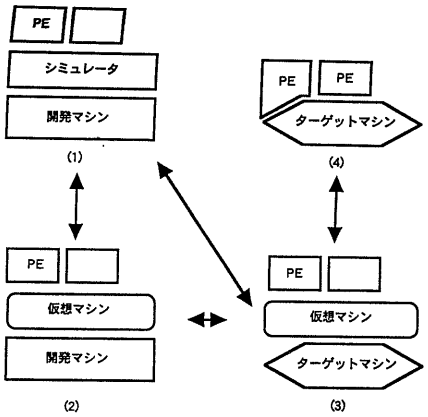

図 1: 開発環境の全体像

- バス使用率や，キャッシュ，メモリアクセスなどの命令レベルでの性能測定，タイミングクリティカルな部分の開発を行う命令レベルシミュレータ
- OS やサーバに関係する特権命令や割込み，MMU，さらにI/O部分をシミュレートする仮想マシン
- OSのSVCレベルの処理をシミュレートするOSレベルシミュレータ

さらに，これらの環境を開発に必要や情報と性能に応じて，使い分けることによって，開発効率を向上させる．

(2) 開発環境において，可視化環境を提供する．

これらの環境のいずれでも，OSやアプリケーションまで含めた可視化環境が備わっていることが重要である．そこで，従来アプリケーションの一つとして実装されることが多かった可視化ツールを，開発環境中に取り込む．さらに，システムソフトウェアで重要となるCPU，MUからタスク，同期命令のレベルまでを可視化可能にする．

5.2 全体構成

この開発環境の全体構成を図1に示す．

開発は4つのステップからなる．これらは，仮想マシン

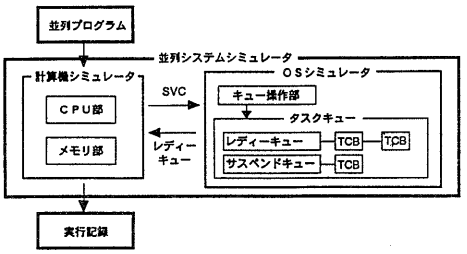

図 2: 並列シミュレータの全体構成

か命令レベルシミュレータか，開発マシンかターゲットマシンかによって，それぞれ分かれる．

開発マシン上でのプログラム開発では，命令レベルシミュレータがある．命令レベルシミュレータ (1) を用いて，システムソフトウェアのデバッグを行う．サーバに関しては，OS シミュレータを持つ命令レベルシミュレータを用いることで，シミュレータの実行速度を上げ，開発効率を向上させることも可能である．タイミングクリティカルな部分については，命令レベルシミュレータを用いてチェックする．

もし，ターゲットマシンと開発マシンの CPU が同じものであれば，開発マシン上に仮想マシンを実現する (2)．

これらによって，OS を開発し，実機上へ仮想マシンを載せて，その上に OS を載せ，開発を行う (3)．仮想マシンを載せているのは，ハードウェアの不備や，ROM モニタのデバッグ機能の不備部分などを補うこと，I/O の不均質性を補うことができる．

このレベルで開発が完了したら，(4) の状態で直接動かして，デバッグ，テストを行う．

5.3 命令レベルシミュレータ環境

命令レベルシミュレータは次の 5 つの機能を備える．

(1) 実行記録の出力

(2) 実行記録に基づいた実行

(3) タスクの状態の監視と変更

(4) タスク生成・同期 SVC のトラップ

(5) ブレークポイントの設定

(6) 並列計算機アーキテクチャの設定

(7) 並列 OS の設定

並列シミュレータの全体構成を図 2に示す．

並列シミュレータは，計算機シミュレータと OS シミュレータという二つの部分から構成する．次にこれらについて述べる．

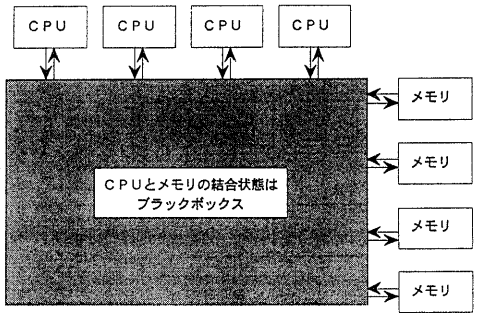

図 3: 計算機シミュレータのモデル

(1) 計算機シミュレータ

計算機シミュレータのモデルを図 3に示す．

計算機アーキテクチャをシミュレートするシミュレータは，実際のハードウェアを機械命令のレベルでシミュレートする．ここで，いくつかの PE-MU 間の結合状態を変えられるように，CPU とメモリとを別の部分に分離して，その間の結合を変更できるようにする．これで，いろいろなバス結合の状態を実験することができる．例えば，マトリクススイッチ型や，共有メモリ型などほとんどの結合状態をこの枠組みに入る．

内部の資源の使用時間管理の表を持ち，資源がどの時間まで使用できるかをシミュレートすることができる．

(2)OS シミュレータ

OS シミュレータは，実現する OS のサブセットを実装する．OS シミュレータを用いる場合，OS の機能は命令のシミュレートではなく，ネイティブコードで実行する．このような，OS シミュレータを実現する理由は，シミュレータの実行速度の向上にある．OS のモデルが決定している場合，その上位に位置するサーバやアプリケーションの開発には，OS 命令まで含めたシミュレートが不必要な場合が多い．むしろ，OS の SVC は仮想マシン命令であると解釈し，OS が提供する資源，例えばタスクやセマフォの単位で操作する方が分かりやすいし，開発効率がよい．システム全体でのチューニングや，OS まで含めたデバッグを行う場合だけ，OS レベルまでのシミュレートを行えばよく，その場合には (1) の計算機シミュレータによって可能である．

また，OS シミュレータは同期命令の実行記録をロギングすることができる．これは，可視化とシミュレータの再実行の二つに用いる．これによって，並列処理の非決定性をある程度制御することができる．また，OS の SVC の

実行順序を入れ換えて再現することで，同期や通信機構などについてのデバッグを容易に行うことができる．

さらに，OS の SVC 命令をネイティブコードで実行する場合，SVC の実行時間が問題となるが，これについても自由に設定が可能である．これで，SVC の実行時間の変化によるプログラム実行への影響を測定できる．

5.4 仮想マシン環境

仮想マシンは，複数の仮想プロセッサや仮想メモリ，仮想時間を提供する．この環境では，メモリ，I/O のインタフェースはすべて仮想化されている．

このような構成で，マルチスレッドの OS やユーザレベルスレッドのシステムの開発を行うことができる．

複数の仮想プロセッサを提供するために，割込み発行，もしくは，一定のタイムクォンタムでスケジューリングする．

この環境では，命令レベルで同時に実行することはできないが，タイムスライスにすることで，複数のプロセッサを同時に実行しているイメージを得ることができる．これによって，上位の OS や AP にとっては，シミュレータより高速にプログラムを実行することができる．

開発マシンとターゲットマシンとが同じ CPU アーキテクチャを備えている場合，仮想マシン環境は，開発マシン，実マシンの二つの上で実現する．プログラミングインタフェースとして同じものを提供することで，プログラム開発の連続性を増すことができる．

この場合，開発マシン上では，メモリマップのインタフェースやソフトウェア割込み，スレッドライブラリなどを用いることで，仮想マシンのインタフェースを実装することができる．

5.5 可視化環境

いずれの環境下でも，システムソフトウェアの内部やアプリケーションの可視化機構を提供できる．

仮想マシン，シミュレータのいずれも可視化環境を提供する．層ごとに，実現すべき可視化の機構は異なる．実際に次のような機構は必要である．また，命令レベルシミュレータや OS シミュレータでは，次のような可視化が必要になる．

- タスク，セマフォ，変数アクセスの相関図

  これらについて，生成，実行，終了，アクセスの間の相互関連を表示する．

- システムプログラム内部のスレッドの実行時間
- タスク，セマフォ，変数アクセスのダイヤグラム

また，これらを統計情報としてヒストグラムとして表示するツールが必要になる．

また，仮想マシンレベルでは，OS やサーバに関連する部分を可視化する．現時点では，次のようなものがある．

- PE モニタ

  プロセッサのマスクレベル，特権状態の遷移を可視化する．

- 割込みモニタ

  割込み状態や割込みレベルの変化を可視化する．

- メモリモニタ

  メモリのどの位置をアクセスしたか，MMU やキャッシュのヒット率などを表示する．

- I/O モニタ

  仮想マシンが提供する I/O をどれだけ利用しているか，待ち時間はどのくらいかを表示する．

6 プロトタイプ

本システムの設計に当たって，我々はいくつかの環境を作成してきた．

我々はすでに命令シミュレータ，OS シミュレータを開発した．このシミュレータは，MC68000 のプロセッサをシミュレートし，同時にとったロギングデータから可視化を行うことが可能である [11]．

可視化の例を図 4, 5, 6に示す．これらは，それぞれ，タスクダイヤグラム，タスクやセマフォの相関図，タスクの粒度に関する統計情報を表示している．

また，仮想マシン環境については，OS デバッグ環境である「忍」を用いる．「忍」では複数の仮想マシンを提供することができる．この仮想マシン環境を用いて，マルチプロセッサの環境を実現する．メモリについても，仮想化した MMU である VMMU を実現している．これによって，PE, MU, I/O については，それぞれ仮想化した環境を提供できる．ハードウェアのデバッグボードのように，別の仮想マシン上でデバッグ機能を実行することができる．デバッグ用の仮想マシンから，他の仮想マシン群の操作や監視を行う機能を実現している．これで，実際の OS のコードを変更せずに，デバッグの操作や可視化が可能になる．

図 7に「忍」の実行画面例を示す．

7 むすび

本稿では，並列システムプログラムの開発時における問題点と要求について考察し，それを解決するプログラム開発環境を提案した．ここでは，命令レベルシミュレータと仮想マシンとを結合し，開発時間の短縮と，効率のよいデバッグ，テストを可能にする環境の設計を行った．また，

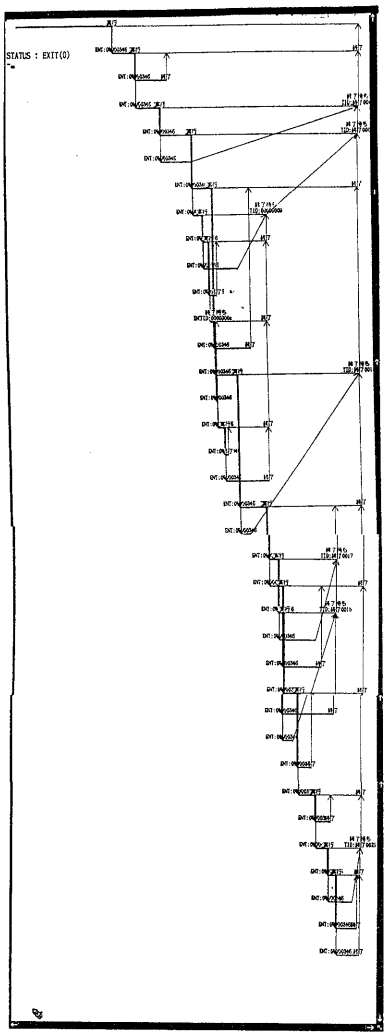

図 4: ダイアグラム表示例

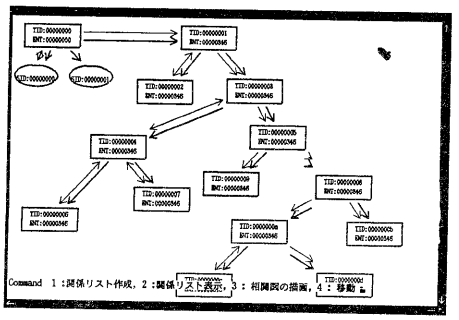

図 5: 相関図

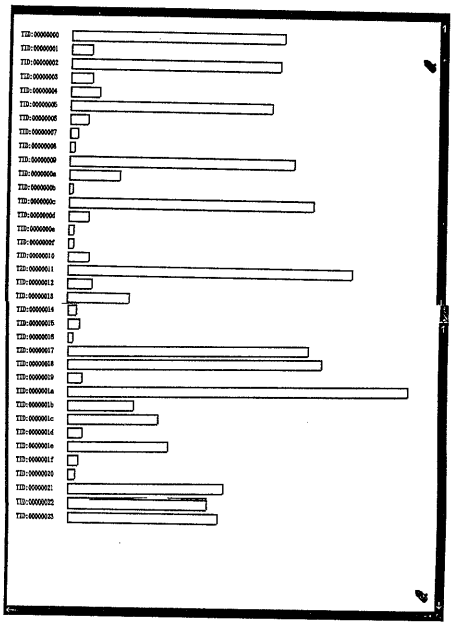

図 6: 統計情報の表示例

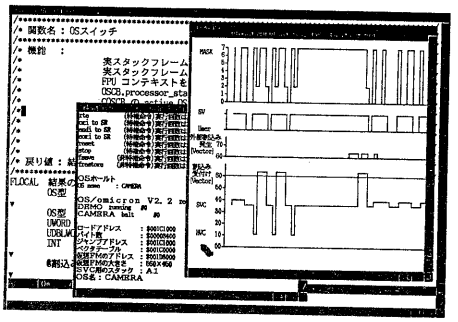

図 7: 「忍」の実行画面

システムプログラムについても可視化が重要であると考え，可視化機構についても考察を行った．

現時点では，開発環境のための要素のうちいくつかが整った段階であり，これらを結合し，実際にOSをはじめとするシステムプログラムの開発を行う必要がある．今後，現実のハードウェアについてシミュレートを行い，動作の比較などを行い，本環境を評価する．

参考文献

[1] 村山他：協調的オペレーティングシステムCOSの開発環境，情報処理学会 JSPP '94, pp.357-364, May. 1994.

[2] McDowell 他：Debugging Concurrent Programs, ACM Computing Surveys, Vol.21, No.4, Dec. 1989.

[3] 舘村他：並列論理型言語 Fleng のマルチウィンドウデバッガ HyperDEBU，情報処理学会論文誌，Vol.33, No.3, 1992.

[4] Hollingsworth 他：The Integration of Application and System Based Metrics in a Parallel Program Performance Tool, Procs. of Third ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, Apr. 1991.

[5] 高橋他：研究プロジェクト総説：OS/omicron 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，39-5, 1988.

[6] 中川他：MC68000 ユニ&マルチ・プロセッサ・システム用記述言語C処理系の開発，情報処理学会 OS 研究会資料，21-7, 1983.

[7] 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, 1990.

[8] 並木他：ビルディング・ブロック・システムのための共有ソフトウェアバス～礎～，情報処理学会 OS 研究会資料，51-2, 1991.

[9] 岡野他：並列処理用 OS カーネル“OMICRON V3”の開発とハイパ OS による共有メモリ型マルチプロセッサへの実装，情報処理学会論文誌，Vol.32 No.5, 1991.

[10] 下山他：アーリー法による並列構文解析プログラムの実現，情報処理学会第 44 回全国大会，2F-8, 1993.

[11] 早川他：並列プログラムの可視化環境の開発，電子情報通信学会 CPSY94-47, Jul. 1994.