プロセス切り換え機構の高速化に関する提案†

田胡和哉†† 板野肯三††† 益田隆司†††

筆者らは，オペレーティング・システムの設計，実現，理解等をより容易に行うことを目的として，資源管理を行うプロセスを通信で結合することによりシステムを記述する方式を提案し，UNIX* システムを例にして，提案した方式によってオペレーティング・システムを汎用の小型計算機上で実現した．その結果，とくに，通信によるプロセス切り換えのオーバヘッドが大きいことが明らかになった．本論文では，プロセッサ・バンク方式と呼ぶ，プロセス切り換えの高速化を図るためのハードウェア機構を提案する．基本的には，複数のプロセッサをプールし，このプロセッサを切り換えることによりプロセスを切り換えることを考える．あるプロセッサがプロセスを実行するのと並列に，別のプロセッサがプロセスの状態の退避，実行可能なプロセスの選択，その状態の回復を行って次に実行するプロセスの準備を行う．そして，プロセスが停止すると同時に次のプロセスの実行を開始することにより，プロセス切り換えのオーバヘッドの大半を吸収する．また，プロセッサ・バンク方式によるシステムを，マイクロ・プロセッサを 2～4 台用いて設計し，その性能を先の OS の動作解析から得られた実測データにより推定した．

1. はじめに

オペレーティング・システム (Operating Systems：以下，OS と略記する) は，他のソフトウェア・システムに比較して，設計，開発が難しいシステムであるといわれている．一方，ハードウェア価格の急速な低下により，小型，超小型機の普及は著しく，それに伴って，新しい OS を設計したり，あるいは，既存の OS に手を加えて，それぞれの問題向きの OS を作成したりするような必要性が増大している．

このような背景から，OS の設計，開発，機能の追加，変更をより容易に行うための技術が種々検討されている[6]．筆者らは，その一つの方法として，OS を資源管理機能の集合体としてとらえ，各資源管理機能を互いに通信で結合されたプロセスによって実現する方式を提案した．そして，提案した方式によって，米国ベル研究所で開発され現在広く利用されている UNIX システム[3]を再設計し，それを実現することを試みている[4]．現在，UNIX バージョン 6 の再開発がほぼ完了した段階であるが，これまでの開発の過程を通して，システムのモジュラリティが向上し，設計，開発，デバッギング等の効率が大幅に改善されたことが確かめられた．また，再設計後のシステムでは，システムの理解が従来システムに比較してはるかに容易になっている．

提案したシステムの設計方式では，上記のような利点がある一方，モジュール間の結合がすべて通信によって行われるために，従来の手続き呼び出しを利用する方法に比較すると性能の低下がかなり大きくなると考えられる．我々は，今後，再設計されたシステムの性能に関する各種の実測を行う予定であるが，再設計されたシステムによるこれまでの第一次近似的な測定では，当初から予想されたように，OS の CPU 時間が大きく，かつ，その半分近くが通信によるプロセスの切り換えによって費やされていることが明らかになった[5]．

提案した OS の設計方式の実用化を図るためにはまずこの問題を解決することが重要であり，本論文では，その一つの方策として，プロセス切り換えを高速に行うためのハードウェア機構を提案する．

プロセス切り換えの高速化を図る方法として，レジスタ・バンク方式がしばしば利用される．実際に，数組の汎用レジスタや記憶管理レジスタを用いて，環境の切り換え時のレジスタの内容の退避，回復に要するオーバヘッドを軽減する機構が，多くの計算機システムで用意されている．しかしながら，これは，数種類の環境の切り換え，たとえば，ユーザ・プログラムと OS のカーネル間での切り換え時には効果を発揮するが，小規模のプロセスが頻繁に切り換わる場合には必ずしも有効に動作しない．

この理由は，第一に，レジスタ・バンクの数がプロセスの数と同程度に多く必要であること，第二に，レ

† Design of a High-Speed Process Switching Mechanism by KAZUYA TAGO (Doctoral Program in Engineering, University of Tsukuba), KOZO ITANO and TAKASHI MASUDA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系

* UNIX is a trademark of Bell Laboratories.

ジスタ・バンクは受動的であり能動的な処理機構は単一であるために，プロセスのスケジュールに要するオーバヘッドが吸収できないことによる．

そこで，受動的な要素であるレジスタだけでなく能動的な要素である処理機構をも含んでいるプロセッサをバンクとしてプールして利用することを考案した．以下では，このような方法を，プロセッサ・バンク方式と呼ぶことにする．プロセッサ・バンクの中のアクティブなプロセッサは，アクティブなプロセスに割りあてられ，そのプロセスを実行する．バンク中のアクティブでないプロセッサは，汎用レジスタや記憶管理レジスタ中にあるプロセスの状態を退避し，次の実行可能なプロセスを選択し，そのプロセスの状態をレジスタに回復して，次の実行に備える．アクティブなプロセスが停止すると，プロセッサ・バンク中に待機している実行可能なプロセスに割りあてられたプロセッサがただちに実行を開始する．

プロセッサ・バンク方式の利点は，第一に，最近の半導体技術の進歩により，低価格の1チップ化されたマイクロ・プロセッサが利用できる点であり，経済的にも技術的にも容易に実現できることがあげられる．第二の利点は，プロセスのスケジュールは，プロセッサ・バンク中のアクティブでないプロセッサによって排他的に行うことができるので，制御が簡単化されることである．第三の利点は，プロセッサ・バンク中のアクティブなプロセッサは複数個でもよく，一般のマルチ・プロセッサを管理する一つの方式としても考えることができ，広範囲の応用が可能な点にある．

本論文では，以下に，プロセッサ・バンク方式の基本概念と，そのハードウェア機構の構成について述べ，小規模のプロセスの集合として構成されているOSがプロセッサ・バンク方式によるアーキテクチャ上で動作することを想定して，その論理的適用法，および，システムの性能の推定を行う．

2. プロセッサ・バンク方式の基本概念

通信で結合されたプロセスは制御においてもアクセスする変数においても互いに独立であるために，それを単位としてOS等の並行動作システムを記述することにより，高いモジュラリティが得られる．その一方において，通信に由来するプロセス切り換えが頻繁に行われることになり，性能が低下しやすい．実際に，この方式に基づいて試作されたOSでは，1回のプロセス切り換えに450 μsecかかるのに対し，システムの動作中約1.3 msecに1回プロセス切り換えが行われ，全体としてプロセス切り換えに費やされるCPU時間はきわめて大きい．

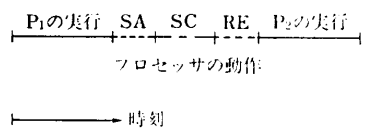

図1 並行動作プログラムの実行過程
Fig. 1 Execution process of concurrent program.
SA：状態の退避に要する時間，SC：プロセスのスケジュールに要する時間，RE：状態の回復に要する時間

プロセッサ・バンク方式の狙いは，複数のプロセッサを切り換えて使用することにより，このようなオーバヘッドの軽減を図ることにある．

2.1 プロセス切り換え処理とその高速化

複数のプロセスの集合からなる並行動作システムは，単一プロセッサ上では，図1のように実行される．実行中のプロセス P_1 がある事象により停止すると，P_1 の状態をスタックあるいはプロセス制御ブロック(process control block：以下，pcbと略記する)に退避し(SA)，次に実行すべき実行可能なプロセス P_2 を選択し(SC)，P_2 の状態を回復して(RE)，その後，P_2 の実行にはいる．SA，SC，REの和が，1回のプロセス切り換えに要する時間である．

プロセス切り換えを高速に行うために，プロセッサ・バンク方式では，複数のプロセッサを交互に切り換えることによりプロセスの切り換えとプロセスの実行を並列に行う．すなわち，プロセッサ PS_1 がプロセス P_1 を実行しているあいだに，他のプロセッサ PS_2 は，他のプロセスのSA，SC，REを行って次の実行に備え待機する．P_1 が通信待ち等の理由により停止した場合には，PS_2 がただちにプロセス P_2 の実行を開始する．

実際のシステム運用にあたっては，あるプロセスが停止した時点で，つねに他のプロセッサにおけるプロセス切り換え処理が終了しているとは限らない．そこで，さらに，多数のプロセッサを用いて，プロセスの切り換えが終了して待機状態にあるプロセッサをプールしておくことにより，それが存在する確率をふやす．このとき，以下では，プロセッサの切り換えのスケジュール方法により，a)単一プロセッサ・モデル，b)マルチ・プロセッサ・モデル，c)マルチ・バンク・モデル，に分けて考えることにする．単一プロセッサ・モデルによるシステムでは，プロセスの実行および切り換えを行うプロセッサはつねにたかだか1台

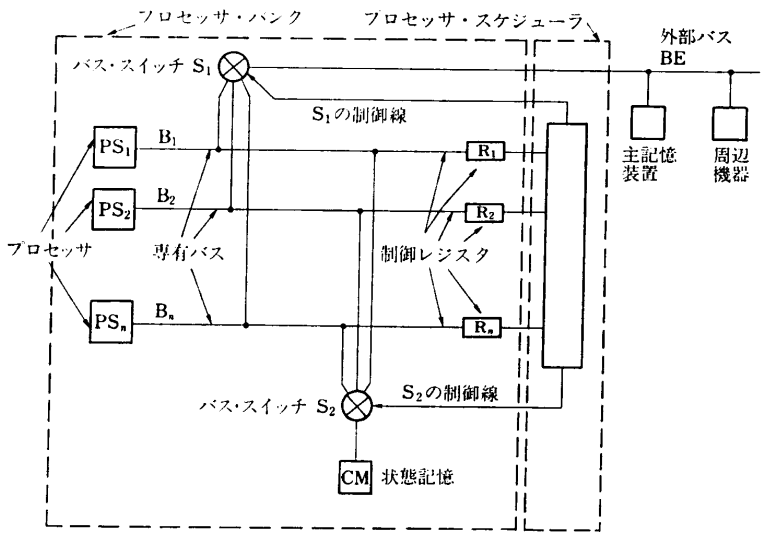

図 2 プロセッサ・バンク方式によるシステムの構造
Fig. 2 Structure of a system using processor banks.

であり他のプロセッサは待機状態にある．外部からは，プロセス切り換えが高速に行われる単一のプロセッサとして見える．マルチ・プロセッサ・モデルでは，プロセスの実行を行うプロセッサは複数台あり，プロセスの切り換えを行うプロセッサはたかだか1台である．マルチ・バンク・モデルでは，両者とも複数台存在する．両モデルとも，外部からは，プロセスの切り換えが高速化されたメモリ共有型マルチ・プロセッサ・システムとして動作するように見える．

2.2 単一プロセッサ・モデル

単一プロセッサ・モデルにおけるプロセッサ・バンク方式は，**図2**のように実現される．システムは，プロセッサ，状態記憶，バス・スイッチ等を主体とするプロセッサ・バンクと，プロセッサ・スケジューラに大別できる．n 個のプロセッサ PS_1〜PS_n は，おのおの専有バス B_1〜B_n をもつ．これらの専有バスは，バス・スイッチ S_1 を介して外部バス BE に接続される．バス・スイッチは，制御線によって制御される3状態ゲートにより構成する．プロセッサ・スケジューラは，プロセッサのうちの1台，PS_i を選択して，S_1 によりその専有バスと外部バスを結合する．これにより，PS_i は，主記憶装置，周辺機器等にアクセス可能になり，プロセスを実行する．このようなプロセッサの状態を外部サイクルと呼ぶ．単一プロセッサ・モデルでは外部サイクルに存在するプロセッサはつねにたかだか1台である．一方，各プロセッサの専有バスは，バス・スイッチ S_2 を介して状態記憶 CM に接続されている．CM は，内部に，各プロセスの pcb，スケジューリングのためのキュー，および，プロセス切り換え制御プログラム等を保持しており，主記憶装置と同程度のアクセス速度をもつ小容量の記憶装置である．プロセッサ・スケジューラは，PS_i 以外の1台のプロセッサ PS_i を選択して，S_2 によりその専有バスと状態記憶を結合する．これにより，PS_i はプロセスの切り換えを行うことが可能になる．このようなプロセッサの状態を内部サイクルと呼ぶ．内部サイクルに存在するプロセッサも，たかだか1台である．すなわち，各プロセッサは，外部サイクル，内部サイクル，および，待ち，のいずれかの状態をとり，それぞれ，プロセスの実行，プロセスの切り換え，他の状態への遷移の待ち合わせ，を行う．これらはソフトウェアで実現される．一方，複数のプロセッサのなかでどのプロセッサをどの状態とするかは，プロセッサ・スケジューラ中のハードウェアがスケジュールする．そこで，各プロセッサは，それぞれの専有バス上にもつ制御レジスタを通じてプロセスの停止および切り換えの終了時点をプロセッサ・スケジューラに通知し，状態が切り換えられるのを待つ．

プロセッサ・スケジューラは，プロセッサのスケジューリングを，**図3**に示す方法により行う．この待ち行列網では，プロセッサが客 (customers) として循環する．内部サイクルが終了したプロセッサは，優先

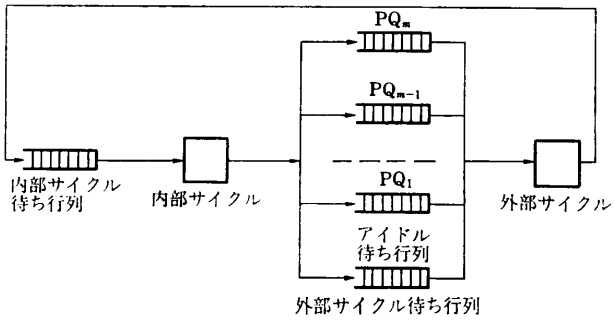

図 3 単一プロセッサ・モデルにおけるプロセッサのスケジューリング法
Fig. 3 Scheduling scheme of processor allocation on single processor model.

度付き待ち行列 PQ_1～PQ_m のいずれかに並び待ち状態となる．これらの待ち行列中の最も優先度の高いプロセッサが選択されて外部サイクルにはいるので，これらを外部サイクル待ち行列と呼ぶ．優先度は，割り込み処理を行うプロセス等，緊急度の高いプロセスを優先して実行するために設けられるものであり，その数および使用法は応用に依存する．このとき，最も優先度の低い待ち行列は，応用によらず，アイドル・プロセスを実行するための待ち行列である．これを，アイドル待ち行列と呼ぶ．

ここで，あるプロセッサ PS_i の状態の遷移を追跡してみる．PS_i が外部サイクルにあるとき，実行するプロセスが待ち状態になった場合，PS_i のプロセス切り換え制御プログラムは，制御レジスタ R_i を通じて PS_i を内部サイクルに入れることを要求し，待ち状態にはいる．プロセッサ・スケジューラは，PS_i が内部サイクル待ちであることを内部の状態レジスタに登録すると同時に，S_1 を切り換え，他のプロセスの状態を回復し待ち状態にあるプロセッサ PS_j を選択し外部サイクルに入れる．さらに，現在内部サイクルにあるプロセッサが状態の回復を終了し外部サイクル待ちになった時点で，S_2 を切り換えて PS_i を内部サイクルに入れる．

このとき，プロセス切り換えに要するオーバヘッドは，プロセッサ・スケジューラが PS_j を選択するのに要する時間とバス・スイッチ S_1 を切り換えるのに要する時間であり，これらはおのおの1クロック以内に実行できるので，全体として2クロック以内に終了する．

2.3 マルチ・プロセッサ・モデル

単一プロセッサ・モデルにハードウェアを追加することにより，複数のプロセスを並列に処理することを考える．すなわち，ここでは，外部サイクルに複数のプロセッサが存在することを許すことにする．これを，マルチ・プロセッサ・モデルと呼ぶ．単一プロセッサ・モデルでは外部バスにアクセスするプロセッサはたかだか1台であり，バス・アクセスが衝突することはなかったが，マルチ・プロセッサ・モデルでは，通常のメモリ共有型マルチ・プロセッサ・システムと同様に，外部バス・アクセスの衝突を防ぐためのアービタを設ける．一方，内部サイクルでは，pcb 等のプロセスの管理情報に対するアクセスの排他性を保証するために，つねにたかだか1台のプロセッサがプロセス切り換え処理を行う．そこで，**図4**に示すような過程によりプロセッサがスケジュールされる．

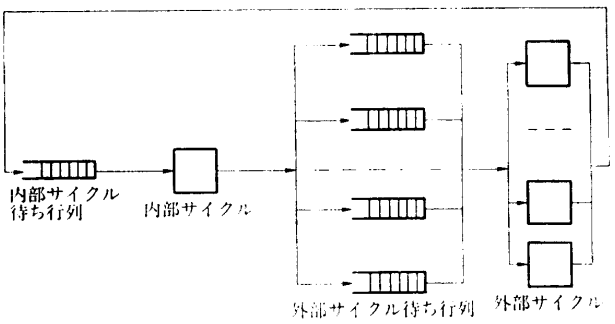

図4 マルチ・プロセッサ・モデルにおけるプロセッサのスケジュール法
Fig. 4 Scheduling scheme of processor allocation on multi processor model.

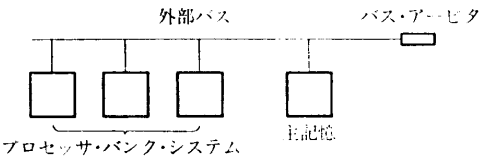

図5 マルチ・バンク・モデルによるシステム
Fig. 5 Schematic diagram of system by multi bank model.

マルチ・プロセッサ・モデルによるプロセッサ・バンク方式によれば，マルチ・プロセッサ・システムにおけるプロセス切り換えの高速化を図ることができる．さらに，そこで行われる排他的プロセス・スケジューリング処理が内部サイクルに集中され制御の簡単化が図れる．

2.4 マルチ・バンク・モデル

ここでは，単一プロセッサ・モデルによるシステムを複数用いることにより性能の向上を試みる．単一プロセッサ・モデルによるシステムは，外部からは単一のプロセッサと同一に見える．そこで，**図5**に示すように，アービトレーション機構をもつバスによりこれらを結合し主記憶装置を付加することにより，マルチ・プロセッサ・システムを構成することができる．このような構成を，マルチ・バンク・モデルと名付ける．

マルチ・バンク・モデルでは，システムのプロセスを各プロセッサ・バンクに分散して実行する．このような構成により，プロセスの切り換えおよび実行を複数並列に行うことができる．

2.5 各モデルの比較

単一プロセッサ・モデルの利点は，実現が容易であり，従来システムからの移行に際してのソフトウェア

上の変更点も少ない点にある．たとえば，外部から単一のプロセッサと同一に見えるために，従来の計算機システム，とくに，マイクロ・プロセッサを用いたシステムにおいて，プロセッサ部をおきかえて性能の向上を図ることができる．このような目的に，たとえば，従来のメモリ共有によるマルチ・プロセッサの技術を適用することを考えてみる．最近のマイクロ・プロセッサ・システムでは，プロセッサの速度と主記憶装置のアクセス速度に大きな差異はなく，したがって，現実には，メモリ共有型にしても性能の向上がさほど期待できない．このため，キャッシュ機構，もしくは，主記憶装置のバンク分け等が必要となり，構造が複雑化する．また，プロセスの並列実行の同期に関して，ハードウェア，および，ソフトウェア上の機構を必要とする．さらに，単一プロセッサのシステムとは外部バスの仕様が異なり，プロセッサ部以外の変更も必要となる．これらの理由により，このような目的にはメモリ共有型マルチ・プロセッサ構成は適切でないと考えられる．

しかしながら，単一プロセッサ・モデルでは，プロセス切り換え以外の部分の高速化は図れない．これに対して，他のモデルでは，メモリ共有型マルチ・プロセッサ・システムとして動作し，ハードウェアはより複雑になる一方，より性能の向上が図れる．さらに，通常の方式と比較して，内部サイクルと外部サイクルが別個のバスを用いているので，処理の並列度を高くできる．また，同期機構を簡単にできる．さらに，マルチ・バンク・モデルでは，並列度を最も高くできる．このとき，プロセッサ・バンク間でのプロセスの移動は容易でないので，プロセス生成時のプロセッサ割り当ての適切なスケジュールが必要である．

3. ハードウェアの構成

提案したプロセッサ・バンク方式により，モトローラ社製の68000プロセッサを使用してシステムを設計した．ここでは，簡潔な方法で性能の向上を図ることを狙って，マイクロ・プロセッサを用いた単一プロセッサ・モデルによりシステムを構成している．単一プロセッサ・モデルを用いることにより，プロセスの実行条件を単純化し，従来方式との客観的な比較を行うことを試みる．

設計を行ったシステムは，**図6**に示すような内部構造をもち，CDL[1] (Computer Design Language) によって記述されている．

3.1 制御方式

プロセスの切り換えに関する能動的制御は，プロセッサ・バンク中のプロセッサが行う．プロセッサは，制御レジスタに書き込みを行うことにより，**図7**に示す制御コマンドをプロセッサ・スケジューラに送る．プロセッサ・スケジューラは，このコマンドの内容に応じて受動的に変化する制御の状態を保存して，プロセッサ・バンク中のプロセッサを選択する．コマンドの種類は，C_COM フィールドで指定され，E_COM, I_COM, R_COM の3種類がある．E_COM は，内部サイクルから外部サイクルへの遷移を要求する．このとき，C_LEVEL フィールドにより，並ぶべき外部待ち行列の優先度を指定する．I_COM は，外部サイクルから内部サイクルへの遷移を要求する．R_COM は，新たに実行可能なプロセスが生じた場合，アイドル待ち行列に並んだ他のプロセッサを再度内部サイクルに入れて，このプロセスを実行させることを要求する．これを再スケジュールと呼ぶことにする．

制御レジスタは，制御コマンドの発生および待ち合わせ機構を実現する．制御レジスタへの書き込みは外部サイクルにあるプロセッサと内部サイクルにあるプロセッサが非同期的に行うので，排他制御がなされ，一つが選択されて制御コマンドとなる．制御コマンドを発生したプロセッサは，プロセッサ・スケジューラがデータ・トランスファ・アクノレッジ信号 (Data Transfer Acknowledge：以下，アクノレッジ信号と略記する) を返すまでアクティブの状態で待ちつづける．プロセッサの待ち要因が解除されると，そのプロセッサを外部バスまたは内部バスに接続したのち，アクノレッジ信号を返すことにより，待ち合わせ機構を実現する．

3.2 プロセッサ・バンク

プロセッサの個数が4個の場合のプロセッサ・バンクのハードウェア構成を図6に示す．プロセッサには，1～4のプロセッサ番号を付ける．各プロセッサには，それぞれ専有の記憶管理ユニットを通じて専有バスが接続されている．各専有バスは，制御レジスタ，バス・スイッチ，および，引数領域のための2kバイトの記憶装置が接続されている．引数領域は，内部サイクルと外部サイクルの間で情報をやりとりするための記憶領域で，プロセッサがどちらのサイクルにあってもアクセスできる．状態記憶は，16kバイトのROM，および，16kバイトのRAMからなる．さらに，各プロセッサには，プロセッサ・スケジューラか

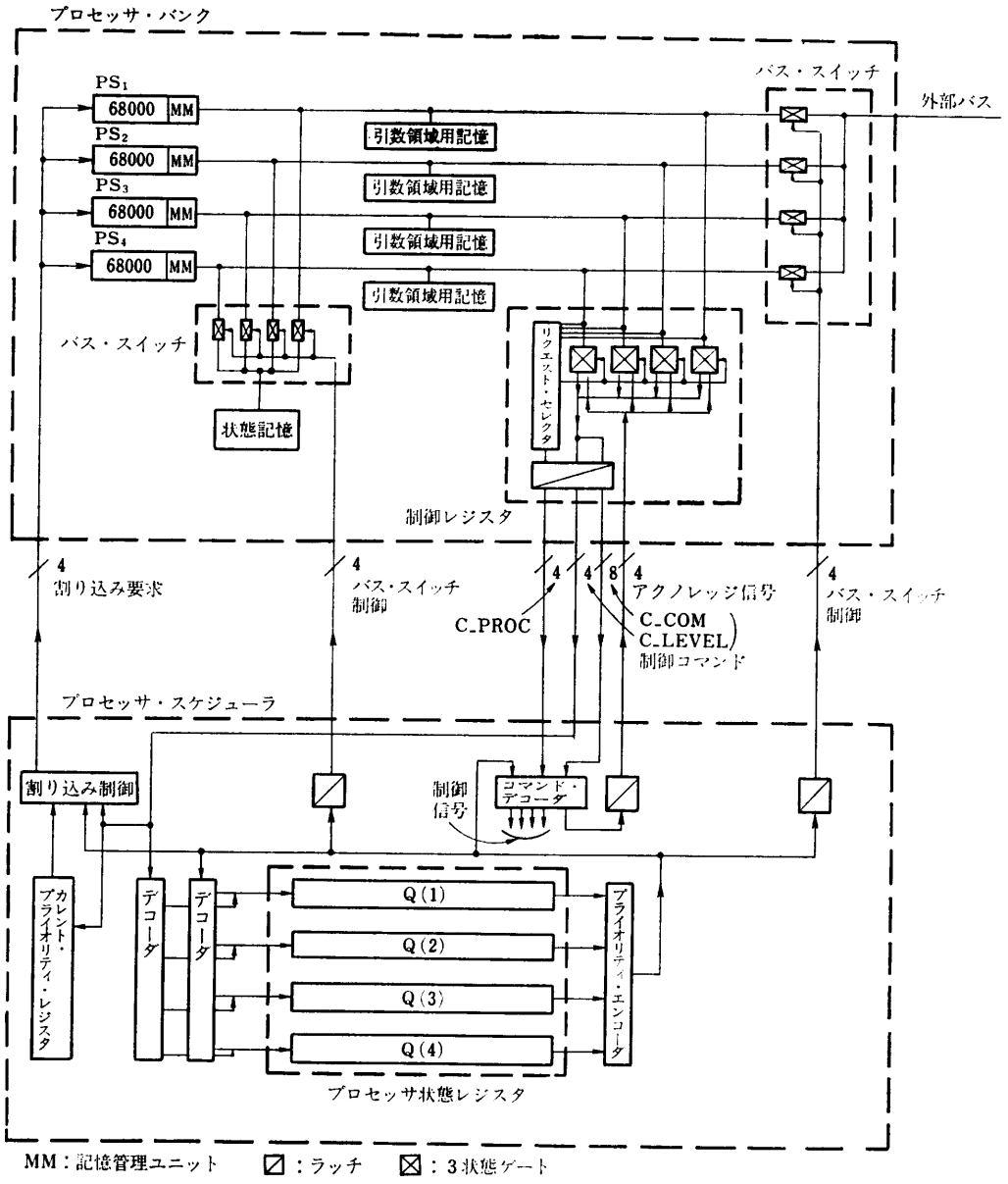

図 6 プロセッサ・バンク・システムのハードウェア構成
Fig. 6 Hardware organization of processor bank system.

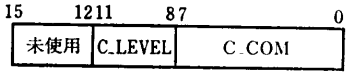

図 7 制御コマンドの形式
Fig. 7 Control command format.

らの信号により割り込みをかける機構が付加されている.

制御レジスタは，主に，排他制御を行うためのリクエスト・セレクタおよびバス・スイッチからなり，制御コマンド，および，発行したプロセッサの番号(C_PROC) を出力し，アクノレッジ信号を受け付ける.

3.3 プロセッサ・スケジューラ

プロセッサ・スケジューラのハードウェア構成を図6に示す. プロセッサ・スケジューラの内部状態は，主に，プロセッサ状態レジスタと，カレント・プライオリティ・レジスタにより決まる. 制御コマンドを受けつけることにより，この内部状態が別の状態に遷移

し，プロセッサ・バンクを制御するのに必要な信号が出力される．

カレント・プライオリティ・レジスタは，現在外部サイクルに存在するプロセッサの優先度を保持している．プロセッサ状態レジスタ Q は，各プロセッサの状態を保持している 4 組の 16 ビット・レジスタである．レジスタの i 組めの要素は，番号 i のプロセッサの状態を示しており，外部待ち行列，外部サイクル，内部サイクル待ち行列，再スケジュール，および，内部サイクルのいずれにあるかをビット位置に応じて示している．ここで，待ち行列の優先度は 0～7 の 8 順位あり，数字が大きいほど優先順位が高く，0 はアイドル待ち行列を示す．プロセッサ・バンク中の四つのプロセッサのうちどれを待ち行列のなかから選ぶかは，プロセッサ状態レジスタに接続されたプライオリティ・エンコーダが決定する．この出力により，プロセッサ・バンク中のバス・スイッチを制御する．

(1) E_COM

このコマンドは，外部サイクルへの遷移を要求する．プロセッサ状態レジスタの要求された優先順位に対応するビットを 1 にすることにより，プロセッサを待ち行列に入れる．このとき，現在外部サイクルにあるプロセッサが並んでいた待ち行列より高い優先度が要求された場合には，外部サイクルにあるプロセッサに割り込みをかけ，要求したプロセッサを優先して外部サイクルに入れることをうながす．ここで，遷移要求があったことにより，内部サイクルが空になる．そこで，内部サイクル待ち，あるいは，再スケジュールされたプロセッサのなかから 1 台をプライオリティ・エンコーダにより選択し，状態記憶へのバス・スイッチを切り換え，アクノレッジ信号を返して，それを内部サイクルに入れる．

(2) I_COM

このコマンドは，内部サイクルへの遷移を要求する．(1)と同様にして，プロセッサ状態レジスタへの登録，および，外部サイクルへ入れるプロセッサの選択を行う．

(3) R_COM

このコマンドは，再スケジュールの要求を行う．アイドル待ち行列にあるプロセッサを 1 台選択して，そのプロセッサが状態レジスタの内容を再スケジュールされた状態に変更する．

4. OS への適用法

ここでは，文献 4) で報告した OS に対して，3 章で述べたプロセッサ・バンク・システムを適用する方法について述べる．

ここで用いる OS は，UNIX システムとシステム・コール・レベルで同一の仕様をもち，図 8 に示すような論理構造をもつ．OS 内部は，通信で結合された 63 個のプロセス（システム・プロセス：以下，混乱のない限りプロセスと略記する）によって構成されている．ユーザ・プロセスからのシステム・コールおよび外部割り込みも通信によりプロセスに伝達される．各プロセスは，相互排除アクセスされる資源のおのおのを管理しており，C 言語で記述されている．通信は，ランデブ (rendezvous) 方式を採用した．通信を実現するプログラム群を OS 核と呼ぶ．通信に伴ってプロセス切り換えが行われる．そこで，OS 核の主要部を内部サイクルで実行し，ユーザおよびシステム・プロセスを外部サイクルで実行することにより，プロセッサ・バンク方式を，実現した OS に適用する．

4.1 核プリミティブ

各プロセスは，通信の実行を手続き呼び出しの形式で OS 核に依頼する．これを，核プリミティブと呼ぶ．種類および機能の概要を表 1 に示す．

ランデブ呼び出し(rendezvous call)は，call プリミティブを通信相手のプロセス名，エントリ名，およ

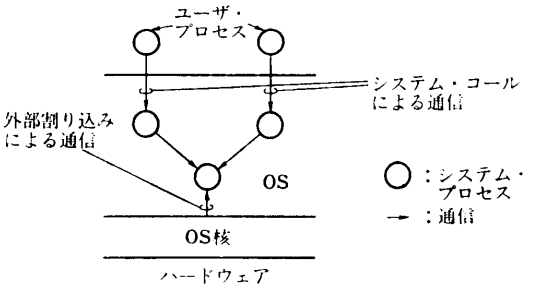

図 8 実現された OS の論理構造
Fig. 8 Logical structure of implemented operating system.

表 1 核プリミティブ
Table 1 Kernel primitives.

名 前	機 能
call	ランデブ呼び出し
acc	受け付け
endr	ランデブの終了
runu	ユーザの実行要求

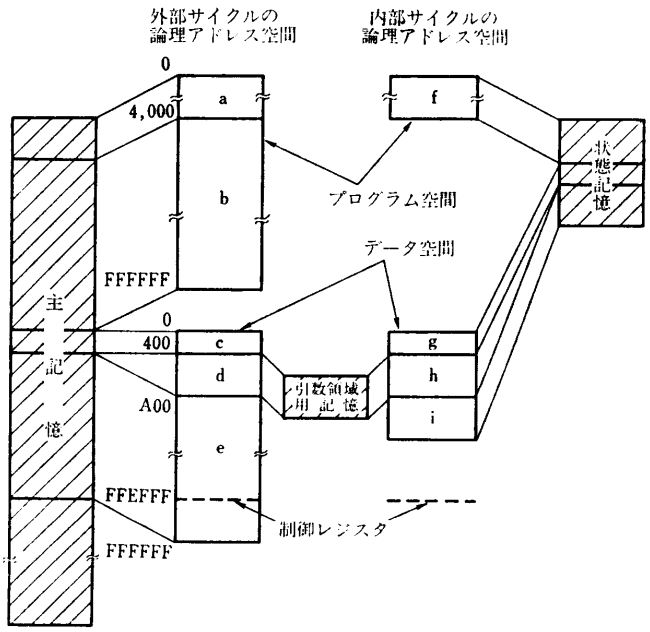

図 9 OS の論理アドレス空間の割り当て
Fig. 9 Logical space allocation of operating system.
a: OS 核のプログラム領域，b: システム・プロセスのプログラム領域，c: 外部割り込みベクタの領域，d: 引数領域，e: システム・プロセスの変数，スタック領域，f: OS 核のプログラム領域，g: 再スケジュール用割り込みベクタの領域，h: 引数領域，i: プロセス管理情報の領域

び，通信データを引数として呼び出すことにより実現される．call は，副作用として待ち合わせを行い，ランデブが終了するまで復帰しない．受け付け (accept) は，受け付けを行うべきエントリ名を引数として acc プリミティブを呼び出すことにより行う．acc は，待ち合わせを行い，ランデブが成立すると相手プロセスのスタックへのポインタをもどり値としてもどる．ランデブは，endr プリミティブにより終了し，呼び出したプロセスが実行可能になる．通信によるデータのやりとりは，主記憶中にある各プロセスのスタック間で直接行われるので，核プリミティブはプロセス切り換えのみを実行する．

ユーザ・プロセスは，システム・プロセス中から upcb (user process control block) の初期値を引数として runu プリミティブを発行することにより実行される．upcb は，ユーザの状態を保持しているデータ構造である．

4.2 論理アドレス空間の割り当て

各ユーザ・プロセスおよび OS はおのおの別個の論理アドレス空間（以下，論理空間と略記する）上で動作する．さらに，OS の論理空間は，図 9 に示すように，内，外部サイクルに応じて割り当てられる．各システム・プロセスは，同一の外部サイクルの論理空間上で作動する．OS 核は，両サイクルにまたがって動作する．

論理空間は，記憶管理ユニットおよびプロセッサ切り換え機構を介して主記憶，状態記憶，および，引数領域用記憶にマップされる．ユーザ・プロセスの論理空間はすべて主記憶にマップされる．図 9 a，b，c，e は主記憶上にマップされる．f，g，i は状態記憶上にマップされる．d，h は引数領域用記憶の同一部分にマップされる．制御レジスタは，両サイクルからアクセスできる．

図 9 b には，各システム・プロセスのプログラムが存在し，e には各プロセスの変数およびスタックが存在する．

a には，核プリミティブをはじめとする OS 核のプログラムが存在する．プロセスから核プリミティブが呼ばれると，核プリミティブは内部サイクルへの遷移を行い，論理空間が切り換わる．OS 核のプログラムは主記憶および状態記憶中に同一のものがおかれており，a と f の内容は同一である．そこで，遷移後はそのまま f 上で動作する．内部サイクルでは，管理情報領域 i がアクセス可能となる．ここには，pcb, upcb をはじめとするプロセス切り換えに必要な管理情報が存在する．

プリミティブの呼び出しに際して，その引数は，レジスタあるいは引数領域を介して内部サイクルに伝達される．たとえば，call プリミティブの通信相手名およびエントリ名の受けわたしにはレジスタを用いる．runu プリミティブの引数である upcb の初期値はデータ量が多いので引数領域を用いる．runu プリミティブは，手続き呼び出しの形でプロセスから伝達された upcb の初期値を引数領域 (d) にコピーした後，内部サイクルへの遷移を行う．内部サイクルでは，h にアクセスすることによりそれを受けとることができる．

外部割り込みは外部サイクルで受け付けるので，外部サイクルではそのための割り込みベクタが主記憶中に割り当てられる (c) のに対して，内部サイクルでは再スケジュール機能実現のためにのみ割り込みが利用され，そのための割り込みベクタが状態記憶中に設定される (g)．

各ユーザ・プロセスには独立した論理空間を割り当

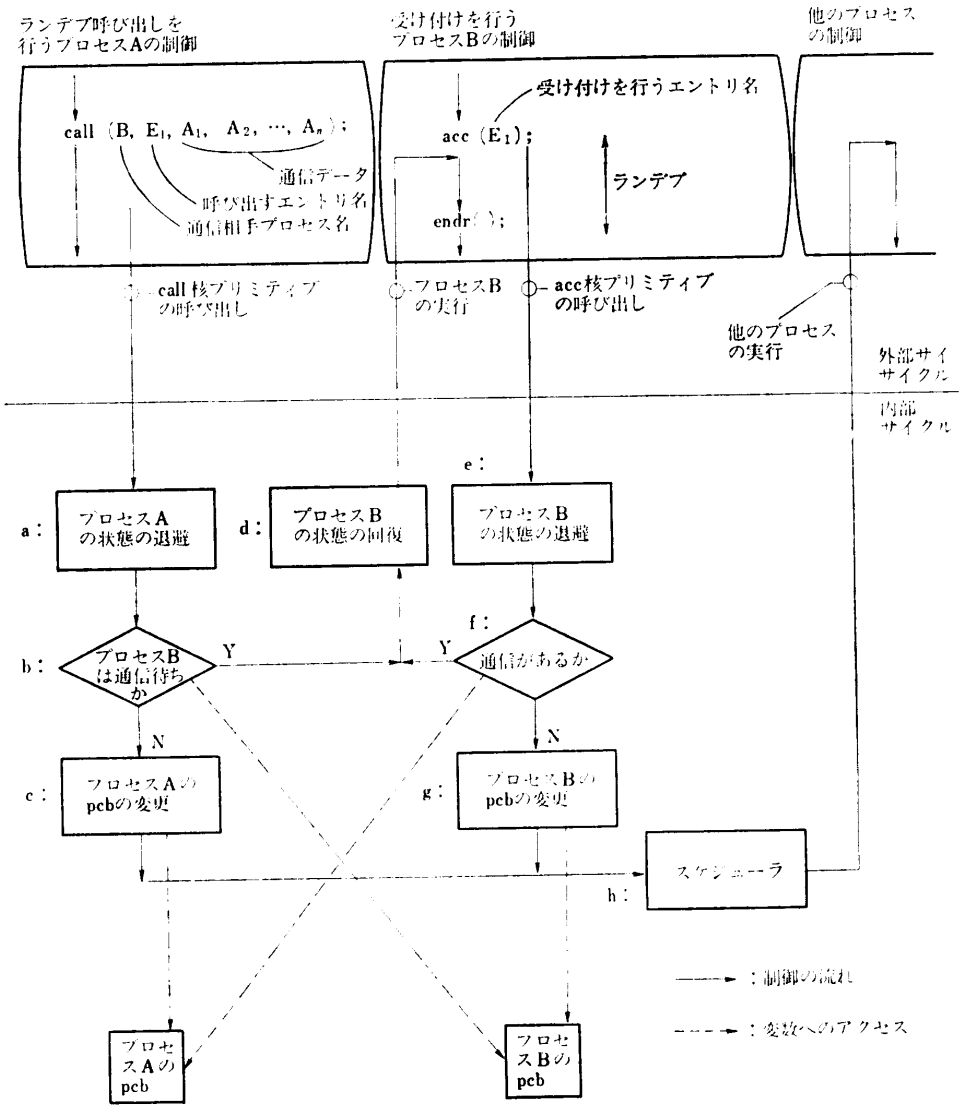

図 10 通信の実行

Fig. 10 Execution of inter-process communication.

てる．ファイル入出力等，OS とのデータのやりとりの必要がある場合には，システム・プロセスが記憶管理ユニットを操作して論理空間の割り当てを一部変更して行う．ユーザ・プロセス間の通信も同様にしてシステム・プロセスが実現する．

4.3 OS 核の動作

call，acc プリミティブの動作は**図 10** のごとくである．call プリミティブは，内部サイクルへの遷移を行った後，プロセスAの pcb に状態を退避する(a)．次に，通信相手にプロセス Bの状態を pcb から判定する(b)．プロセスBがランデブ呼び出し待ちであればその状態の回復(d)を行って，外部サイクルに遷移することによりプロセスBを実行する．そうでなければ，待ち要因をプロセスAの pcb に記入し(c)た後，スケジューラを介して他のプロセスを実行する．acc プリミティブは，内部サイクルに遷移を行った後，プロセスBの pcb に状態を退避する(e)，次に，他のプロセスからの通信があるか否かを pcb から判定し(f)，通信があれば復帰してランデブを開始する(d)．なければ，待ち要因を pcb に記入し(g)た後，スケジューラを介して他のプロセスを実行する(h)．

runu プリミティブは，内部サイクルに遷移し，管理情報領域に領域を割り当て，実行可能なユーザ・プロセスの upcb のリストを作成する．後に，実行可能

なシステム・プロセスがなくなると，スケジューラがこれを実行する．ユーザ・プロセスの実行中，システム・コールが発生すると，プロセッサの機能によりOSが起動した後，OS核が，内部サイクルへの遷移，システム・コールを処理するシステム・プロセスの起動，を行う．

外部割り込みは，外部サイクルにあるプロセッサが受け付ける．割り込み処理手続きは，割り込み要因（割り込みベクタ）を引数領域にセットし，ただちに内部サイクルに遷移して割り込み処理プロセスを起動する．

4.4 プロセッサのスケジュール

外部サイクル待ち行列は，アイドル待ち行列以外に三つ用いる．優先度の高い順に，至急，システム・プロセス，ユーザ・プロセスと名付けることにする．至急待ち行列は，割り込みによる通信を受け付けたシステム・プロセスを実行するためのものである．実行可能なシステム・プロセスがあるにもかかわらずユーザ・プロセスを実行すると，ユーザ・プロセス間でのシステム・コールの処理に不公平が生ずるので，これを防ぐために優先度を分ける．

5. 解　　析

本章では，性能モデルを作成し，それに実現されたOSの実測データを適用することにより，プロセッサ・バンク方式の有効性について考察する．

5.1 単一プロセッサの性能

図1に示したように，単一プロセッサ上の並行動作システムでは，プロセスの実行および切り換えが交互に行われる．そこで，並行動作システムによって実行される仕事の所要時間 T は，

$$T=N\cdot(E+SW) \qquad (1)$$

となる．ここで，E は1回にスケジュールされたプロセスの平均実行時間であり，SW はプロセス切り換えの平均所要時間，すなわち，

$$SW=SA+SC+RE \qquad (2)$$

であり，N はプロセス切り換えの回数である．イベント・ドリブンでプロセスのスケジューリングが行われれば，N は，仕事にのみ依存する．このとき，$N\cdot SW$ がオーバヘッドとなる．いま，

$$\rho=SW/E \qquad (3)$$

とおいて，仕事の性質の指標として用いる．すなわち，$\rho\approx 0$ では，プロセス切り換えのオーバヘッドは無視しうる程度であり，SW を一定とすれば相対的なプロセス切り換えの頻度は低い．一方，$\rho\approx 1$ では，プロセス切り換えのオーバヘッドとプロセスの実行時間は同程度であり，相対的にプロセス切り換えが頻繁に起こることを示している．

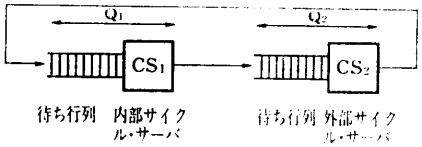

図 11 単一プロセッサ・モデルの性能モデル
Fig. 11 Performance model for single processor model.

5.2 プロセッサ・バンク方式の性能

単一プロセッサ・モデルによるシステムの動作は**図11**のような待ち行列網によりモデル化できる．ここでは，プロセッサが客となる．内部サイクル・サーバ CS_1 にあるプロセッサはプロセスの切り換えを行い，外部サイクル・サーバ CS_2 にあるプロセッサはプロセスの実行を行う．内部サイクル・サーバを出た客はすべて外部サイクル・サーバに向うので，優先度付き待ち行列は一つにまとめることができる．内部サイクル・サーバと外部サイクル・サーバは，おのおの，長さ Q_1, Q_2 の待ち行列をもっている．このとき，待ち行列の長さは，サーバ中の客の数を含めるものとする．いま，CS_1 はつねに同一処理を行うので処理時間 SW の確定過程 (Deterministic process) であり，CS_2 は平均処理時間 E のマルコフ過程 (Markov process) であると仮定する．

提案方式が効果をあげるには，外部サイクル・サーバの待ち行列長 Q_2 が1以上である必要がある．すなわち，$Q_2=0$ の期間は，プロセスの実行が行われないのでオーバヘッドとなるためである．この点に注目すると，プロセッサ・バンク方式による処理時間 T_p は $Q_2=0$ となる確率を P_0 とおけば処理時間中 $(1-P_0)$ の確率でプロセスが実行されるところから，

$$T_p\cdot(1-P_0)=N\cdot E$$

$$\therefore\quad T_p=N\cdot E\cdot\frac{1}{(1-P_0)} \qquad (4)$$

となる．そこで，単一プロセッサとの性能比 $\Delta_p=\dfrac{T}{T_p}$ は，

$$\Delta_p=(1+\rho)\cdot(1-P_0) \qquad (5)$$

となる．P_0 は，待ち行列理論により解析的に導出できる[2]．

5.3 レジスタ・バンク方式の性能

いま，m 個のプロセスからなる並行動作システム

を，n組のレジスタ・バンクをもつプロセッサ上で実行することを考える．

ここで，プロセスの切り換えに要する時間 SW を，さらに，二つの部分 SW_L と SC に分割する．すなわち，

$$SW_L=SA+RE \tag{6}$$

である．SW と同様に，

$$\left.\begin{aligned}\rho_L&=SW_L/E\\ \rho_{SC}&=SC/E\end{aligned}\right\} \tag{7}$$

とおく．SW_L は，プロセスの状態の退避，回復に要する時間であり，SC は，プロセスのスケジューリングに要する時間である．

プロセス切り換え時には，$\frac{n}{m}$の確率で状態がバンク中に存在し，そのときには SW_L が0になると仮定でき，他の場合には SW_L かかるので，レジスタ・バンク方式による処理時間 T_b は，

$$T_b=N\cdot E\cdot\left(1+\frac{m-n}{m}\rho_L+\rho_{SC}\right) \tag{8}$$

となる．よって，単一プロセッサとの性能比 $\Delta_b=\frac{T}{T_b}$ は，

$$\Delta_b=(1+\rho)\Big/\left\{1+\frac{m-n}{m}\rho_L+\rho_{SC}\right\} \tag{9}$$

となる．

5.4 性能の推定

プロセッサ・バンク・システムの性能を推定し，単一プロセッサによるシステムの性能と比較することを目的として，実現されたOS上でCコンパイラを動作させ，その期間中，プロセッサの外部バスにハードウェア・モニタ (DYNAPROBE) を接続し，実時間でシステムの特性を測定した．その結果，システム・プロセスの実行時間が 677 sec であるのに対して，OS核の実行時間が 488 sec であった．すなわち，$\rho=0.72$ となる．さらに，OS核の処理時間のうち約41%が状態の退避，回復に用いられていた．すなわち，$\rho_L=0.29$, $\rho_{SC}=0.43$ となる．

上で行った議論をもとに，当システムを68000単一プロセッサ上で動作させた場合と，3章で設計したシステム上で動作させた場合に推定される性能を比較してみると，たとえば，プロセッサ2台の場合には $\Delta_P=1.4$，3台の場合には $\Delta_P=1.6$ の性能が得られ，OS の CPU 時間がそれぞれ71%，63%に短縮できる．これに対して，63個のプロセスのおのおのにレジスタ・バンクを設けたプロセッサでは $\Delta_P=1.2$ となり，プロセッサ・バンク方式が有効である．

現状での測定例は多くないが，Cコンパイラは最も多く用いられるコマンドの一つであり，UNIX OSの負荷として典型的であると考えられる．今後は，多種類の負荷に対する測定を行い，さらに実際の使用条件に近い条件での解析を行う予定である．

6. むすび

複数のプロセッサを切り換えて使用することにより，プロセス切り換えの高速化を図るプロセッサ・バンク方式を提案した．そして，提案した方式を，プロセスの集合により記述されたOSに適用した場合の性能を推定した．その結果，典型的なプログラムの動作環境下では，OS の CPU 時間を，2台のプロセッサを用いた場合には71%，3台用いた場合には63%に短縮できることが判明した．このとき，プロセスの実行に CPU 時間の58%が用いられているので，プロセス切り換えのオーバヘッドを大きく減少できる．

提案したプロセッサ・バンク方式は，ハードウェアの構成としては汎用性があり，同種の問題に対して広く利用することができると考えられる．

参考文献

1) Chu, Y.: *Computer Organization and Microprogramming*, p. 533, Prentice-Hall, New Jersey (1972).
2) Kobayashi, H.: *Modeling and Analysis*, p. 446, Addison Wesley, Massachusetts (1978).
3) Ritchie, D. M. and Thompson, K.: The UNIX Time-Sharing System, *CACM*, Vol. 17, No. 7, pp. 365-375 (1974).
4) 田胡和哉，益田隆司：オペレーティング・システムの構造記述に関する一試み，情報処理学会論文誌，Vol. 25, No. 4, pp. 524-534 (1984).
5) 田胡和哉，益田隆司：プロセス・ネットワークによるOSの性能について，情報処理学会「オペレーティングシステム」研究会資料 24-2, pp. 9-14 (1984).
6) 和田英一：操作システムまわりの話題から，情報処理，Vol. 21, No. 5, pp. 566-573 (1980).

(昭和59年7月12日受付)
(昭和60年2月21日採録)