実時間オペレーティングシステム R²-86 核のアーキテクチャ†

大久保 英嗣†† 津田 孝夫†† 楠田 修三††
小林 正典†† 杉村 邦彦†††
白濱 和人††† 友田 和伸†††

R² は，制御用ソフトウェアの応答性と移植性の向上を目標とした実時間制御用のオペレーティングシステムである．R² 上に構築されるアプリケーションプログラムは，その大半をC言語により記述することが可能である．さらに，アセンブリ言語により記述されている機械依存部の少数のモジュールを除いて，R² 自体もC言語で記述されている．本論文では，この R² システムのうち，マイクロプロセッサ 8086 上に構築された R²-86 核の各構成要素の設計思想とアーキテクチャについて述べる．R²-86 核は R² の基本部分であり，割込み制御，タスク管理，同期と通信制御，例外制御，資源管理，メモリ管理，タイマ制御，入出力制御の各機能モジュールから構成されている．本論文では，上記の二つの目標を達成するために，これらの構成要素において実現されている種々の手法に関して述べる．

1. はじめに

16 ビットマイクロプロセッサ用オペレーティングシステム（以下 OS と記す）に関して，実時間制御向きの OS が数多く市販されている[1]．iRMX 86[2]，MTOS-86[3]，MKP 86[4]，RMS 68 K[5]，RMS[6] 等がその例であろう．しかし，これらの OS はターゲットプロセッサに依存したものが多く，プロセッサが変更された場合のシステム変更のコストが問題となっている．また，これらの OS を使用するアプリケーションプログラムは，専用のプログラム開発装置を使用して作成される場合が多く，開発装置をも変更しなければならないといった問題がある．最近，この問題を解決するために，アプリケーションインタフェースを統一するための，TRON アーキテクチャに基づく実時間 OS が各メーカで開発されている[7)~12]．これらのアプローチは，各システム間でアプリケーションプログラムの互換性を実現することを目的としており，OS 自体はプロセッサの機種ごとに開発し性能を向上させようとするものである．R² はこれらのアプローチと異なり，アプリケーションプログラムおよび OS 自体の移植性向上を目標としている．

従来，ロボットや NC 工作機械等の機器組込み型の用途で使用される OS に関しては，実時間制御を主目的とするために，システムオーバヘッドを極小化し高速応答を実現することが要求されていた．しかし，機械ごとに OS やアプリケーションプログラムを開発する場合のコストが，機能が豊富になるにつれて急激に増大している．したがって，他分野と同様に実時間制御用ソフトウェアにおいても，各アプリケーションプログラムを含めたシステム全体の移植性が注目されている．すなわち，実時間制御用のソフトウェアは高速応答性に主眼が置かれていたが，これを実現する一方でこの要求と競合する移植性の要求が重要になってきていると言えよう．

本論文では，このような背景の下で開発された R² システムの基本部分である R²-86 核 (Kernel) の設計思想とアーキテクチャについて述べる．

2. R²-86 の特徴

R²-86 は，前章で述べたように実時間制御用のソフトウェアに要求される高速応答性の達成のみならず，システム全体の移植性の向上を目標に設計されている．本章では，これら二つの目標を達成するための R²-86 の特徴について述べる．

2.1 応答性

R²-86 は，応答性向上のために以下の特徴を有している．

† The Architecture of Real-Time Operating System R²-86 Kernel by EIJI OKUBO, TAKAO TSUDA, SYUZO KUSUDA, MASANORI KOBAYASHI (Department of Information Science, Faculty of Engineering, Kyoto University), KUNIHIKO SUGIMURA, KAZUTO SHIRAHAMA and YASUNOBU TOMODA (Mechanics and Electronics Division, Daihen Corporation).

†† 京都大学工学部情報工学科
††† (株)ダイヘン メカトロ事業部

（1） 各々の割込みに関連した処理を逐次的に実行するのではなく，割込み処理に対応して一つのタスクを起動することによって割込み禁止時間を短縮している．

（2） 多様な同期および通信形態をサポートしている．ユーザは，アプリケーションに合った機能をこれらの中から選択することによって，高速応答可能なシステムを実現することができる．特に，R²-86 では，タスク間の同期用のシグナルバッファをタスク制御ブロック内に設定することによって，同期関係の SVC を高速化している．

（3） システム生成時にデバッグモードと実行モードのいずれかのモードを設定可能である．デバッグモードでは，タスク間の関係仕様（3章参照）を動的に調べることによって，SVC の矛盾性のチェックを行っている．実行モードでは，これらのチェック機構が取り外され，高速処理を行うことが可能である．

（4） メモリ管理においては，メモリプールの領域を分割し，それらの分割された単位を複数タスク間で共用可能としている．さらに，分割された単位でメモリコンパクションを行うことにより，メモリプールの確保および解放処理を高速化している．同様に，資源管理においても，資源をグループ化し，グループごとにそれを使用するタスク集合を対応付けることによって，資源の確保および解放処理を高速化している．

（5） 異常状態を引き起こす外部事象等の例外に対する迅速な対応を可能とするために，例外処理の実行をタスクの優先度を変更して行うことが可能である．さらに，SVC 実行中の例外を受け付ける機能をもサポートしている．

割込み処理においてタスクの概念を導入することにより，アセンブリ言語記述の OS である MTOS-86 (Industrial Programming, Inc.) と比較しても割込みハンドラの部分は高速化が達成できている．実測で 62 マイクロ秒の差となっている（測定環境は，5章と同様である）．また，5章で述べるが，同期用の SVC に関しては R²-86 の実行時間は MTOS-86 の実行時間の約 50% となっている．これは，割込みタスクの導入およびシグナルバッファのタスク制御ブロック内の設定のアーキテクチャのゲインによるものと考えられる．以上のほか，R² 独自のプロトコル（必要最小限の制御コードの付加，1メッセージ当たりの通信回数の低減）を実現することによって，プロセッサ間通信の高速化を達成している．現在，RS 232 C (9,600 bps) および光ファイバ(10 Mbps, CSMA/CD) の2種類の通信媒体を使用したプロセッサ間通信を実現しており，R² のプロトコルの妥当性は検証済みである．

2.2 移 植 性

R²-86 では，アプリケーションプログラムのみならず，システム全体の移植性向上のために，R²-86 核自体もC言語で記述している．R²-86 核のステップ数は，現在，約 5,000 行（注釈行，空白行等を除く）であり，このうちC言語のステップ数が約 85% を占めている．R²-86 核の記述においては，さらに，実行環境（プロセッサ，入出力機器，C 言語処理系）に依存するモジュールと依存しないモジュールを明確にファイルレベルで分離する，機械依存部の記述方式を採用している．この方式により，実行環境が変更された場合に，環境に依存するモジュールを容易に識別することが可能である．さらに，非常に少ない変更で移植を完了させることが可能である．

R²-86 核を使用したアプリケーションプログラムの移植性向上のためには，以下の機能を実現している．

（1） 割込み処理のための環境設定機能をサポートすることにより，割込みハンドラのC言語記述を可能としている．これらのインタフェースにより，アプリケーション作成者は，割込みに関する処理ロジックのみをC言語で記述することによって，割込みハンドラを実現することが可能となっている．

（2） R² 上のアプリケーションプログラムは，親子関係に基づく木構造のタスク集合を基本として構成される．これらのタスクを識別するために，各タスクにシステム全体で一意の論理的な番号を割り付けている．この方式によって，複数のプロセッサに分散したタスクを統一的に管理することが可能となる．すなわち，各機能に対応したそれぞれのタスク階層をハードウェアにマッピングすることが容易となっている．

（3） 同期のためのシグナルの領域や通信のためのメイルボックスをタスク対応に持たせることにより，シングルプロセッサやマルチプロセッサ等のプロセッサ構成を意識することなく，アプリケーションシステムを実現することが可能である．すなわち，タスクの同期と通信に関して，プロセッサ内あるいはプロセッサ間といったシステム構成を区別することなくアプリケーションプログラムから一様に見ることを可能としている．

（4） ユーザ固有の入出力ドライバを記述するためのC言語インタフェースをサポートしている．これら

のインタフェースを使用して，R² における標準的なドライバ記述の手順に従ってプログラミングすることにより，ドライバを作成することが簡単に行えるようになっている．さらに，バイトやワード単位の入出力機能や，ビット単位の入出力を行うディスクリート入出力機能を標準で提供している．特に，異なる入出力ポートを一つの入出力ポートとするための仮想化の機能も備えており，ユーザが多種類の機器に関する入出力処理の記述を容易に行えるようになっている．

3. タスク間の関係仕様

R²-86 におけるタスクは，アプリケーションの各機能単位に木構造を形成し全体として林構造をなす．各木のルート (root) は，システム生成時に SIT (System Installation Table) と呼ばれるテーブルにユーザによって定義され，システム初期化時に起動される．SIT は図 1 に示すような構成になっており，ユーザはあらかじめこのテーブルに R² のアプリケーションシステムを構成するためのルートタスクを設定しておかなければならない．システム生成時に，SIT は RTRT (Root Task Registration Table) と呼ばれるテーブルに変換され，各プロセッサ上に置かれる．このテーブルを使用して，プロセッサ間の同期と通信が行われる．

各ルートの子孫タスクは，システム稼動時に動的に生成される．これらの各木に属すタスク集合は同一のプロセッサ上に配置しなければならないが，各々の木はどのプロセッサに割り当ててもよい．R²-86 では，この木構造に基づいて，各タスク間の関係仕様を以下のように規定している（図 2 参照）．

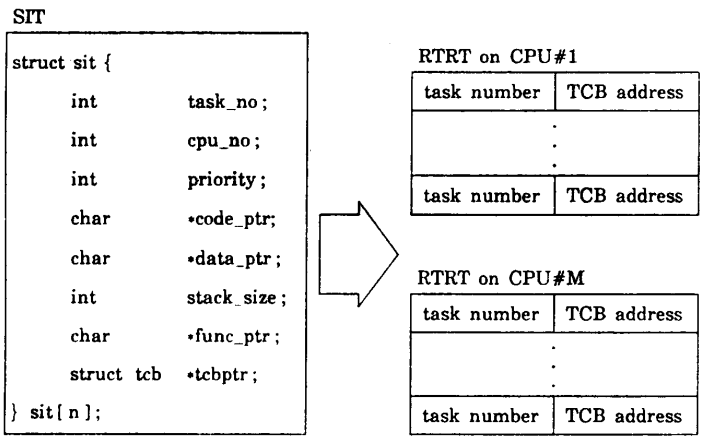
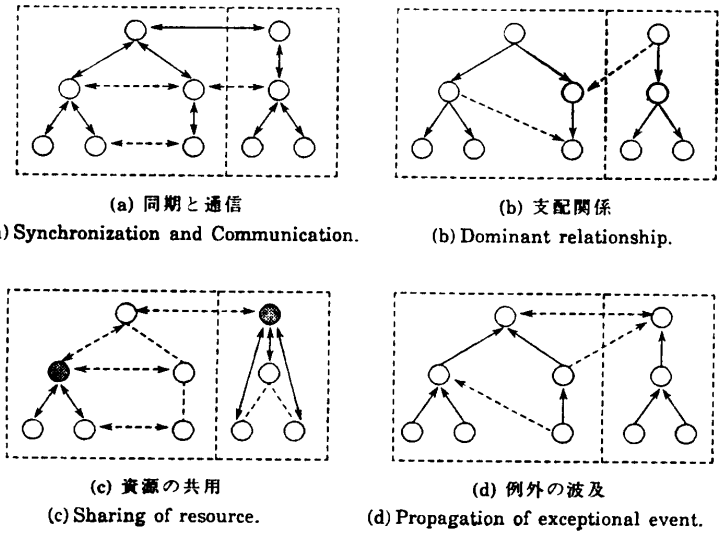


図 1　SIT と RTRT
Fig. 1　SIT and RTRT.



(a) 同期と通信
(a) Synchronization and Communication.

(b) 支配関係
(b) Dominant relationship.

(c) 資源の共用
(c) Sharing of resource.

(d) 例外の波及
(d) Propagation of exceptional event.

注）実線：可，点線：不可，黒丸：資源の所有者

図 2　タスク間の関係仕様
Fig. 2　Specification of relationship between tasks.

（1）　同期と通信

同期と通信は，親子タスク間で成立する．異なる木に属すタスク間の場合は，ルートタスク間においてのみ成立する．しかし，一度相手が認識されると（すなわち，通信相手であるタスクの識別子を得ると）同一木の子孫間，さらに異なる木に属すタスク間でも同期と通信が可能となる．

（2）　支配関係（子タスクの管理）

タスクの動的な生成，中断，再開，さらにタスクの終了待ち等は，親子タスク間でのみ可能である．これらの操作は，当該タスクの親のみが行うことができる．すなわち，親タスクが子タスクを支配 (dominate) している．

（3）　資源の共用

タスク間で共用する資源の確保および解放は，親子間あるいは兄弟間において親タスクの所有する資源に関してのみ可能である．

（4）　例外の波及

例外事象が発生すると，制御は例外処理に移行する．例外処理の中で当該タスクを終了させ，親タスクに任意の

表 1 R³-86 における SVC 一覧
Table 1 List of SVC in R³-86.

割込み制御	(Xfork)	割込みタスクを生成する.
	(Xsetenv_m)	M状態における割込み処理の環境を設定する.
	(Xexit_m)	M状態の割込み処理ルーチンの終了処理を行う.
タスク管理	fork	子タスクを生成する.
	join	子タスクの終了を待つ.
	kill	子タスクを消滅させる.
	exit	カレントタスクを終了する.
	suspend	子タスクの実行を中断する.
	resume	子タスクの実行を再開する.
	initialize	タスクの初期化を行う.
	changepriority	カレントタスクまたは子タスクの優先順位を変更する.
	getpriority	カレントタスクまたは子タスクの優先順位を得る.
	gettaskno	カレントタスクまたは親タスクのタスク番号を得る.
同期	signal	シグナルを送信する.
	reset	シグナルバッファの指定したシグナルをリセットする.
	wait	シグナルを受信する.
	test	シグナルバッファをテストする.
通信	createmailbox	メイルボックスを生成する.
	send	メッセージを送信する.
	receive	メッセージを受信する.
	forward	メッセージを中継する.
	reply	メッセージを返信する.
CPU間通信	(Ycom_init)	CPU 間通信のサービスを開始する.
	(Ycom_end)	CPU 間通信のサービスを終了する.
	(Yqin_trb)	通信要求ブロック TRB をキューにつなぐ.
	(Yqout_trb)	通信要求ブロック TRB をキューからはずす.
	(Ygetcomdev)	通信デバイス番号およびユニット番号を得る.
	(Ygettucbptr)	通信用ユニット制御ブロック TUCB のアドレスを得る.
	(Ysetctrb)	通信要求キューの先頭 TRB をサービス状態にする.
資源管理	hold	資源を確保する.
	free	資源を解放する.
メモリ管理	createpool	メモリプールを確保する.
	destroypool	メモリプールを解放する.
	alloc	メモリフレームを確保する.
	release	メモリフレームを解放する.
タイマ管理	settime	ユーザタイマに時刻を設定する.
	gettime	ユーザタイマの時刻を得る.
	setdate	ユーザカレンダに年月日を設定する.
	getdate	ユーザカレンダの年月日を得る.
	pause	タスクを一定時間停止する.
	(Zset_twb)	タイマ制御用ブロック TWB を作成する.
	(Zqin_twb)	タイマ制御用ブロック TWB をキューにつなぐ.
例外処理	setexcp	例外ハンドラを登録する.
	resetexcp	例外ハンドラの登録を解除する.
	endexcp	例外ハンドラの実行を終了する.
	raise	例外を発生させる.
	disable	例外ハンドラへの制御移行を一時的に保留する.
	enable	例外ハンドラへの制御移行を可能にする.
	setemode	SVC 処理中の異常によって例外を発生させる.
	setnmode	SVC 処理中の異常を例外としない.
	execstandard	標準の例外ハンドラを実行する.
入出力制御	sio	デバイスドライバを実行する.
	dio	ディスクリート入出力を行う.
	bi/bo	バイト入出力を行う.
	wi/wo	ワード入出力を行う.
	(Dio_init)	入出力のサービスを開始する.
	(Dio_end)	入出力のサービスを終了する.
	(Dqin-irb)	入出力要求ブロック IRB をキューにつなぐ.
	(Dqout_irb)	入出力要求ブロック IRB をキューからはずす.
	(Dgetucbptr)	入出力用ユニット制御ブロック UCB のアドレスを得る.
	(Dsetcirb)	入出力要求キューの先頭 IRB をサービス状態にする.
	(Dnext_func)	次の入出力操作を実行する.
	(Dnext_svc)	次の入出力要求のサービスを開始する.

注) 括弧はユーザ固有の処理を記述するためのインタフェースルーチンである.

例外事象を発生させることが可能である．以上の関係仕様は，R² のデバッグモードとして，関係仕様に関与する SVC が発行された場合に各 SVC 処理ルーチンで常にチェックされる．これにより，(意図的なものを除いて) 同期あるいは通信の誤用や共用資源の破壊等を回避することが可能となり，システムの信頼性を向上させている．一方，システムの開発を終了し実際に稼動させる段階では，上記の関係仕様のチェックを取り外し，高速処理を行いたい場合がある．R² では，これを実行モードと呼び，関係仕様のチェックを取り外すオプションをサポートしている．

4. R²-86 核

R²-86 の OS 核は，割込み，タイマ制御，入出力制御等の外部事象のための制御プログラムと，タスク管理およびメモリ管理等の内部事象の制御プログラムに大別される．**表 1** にこれらの構成要素に対応した R²-86 における SVC インタフェースの一覧を示す．以下，本章では，これらのインタフェースに対応して R²-86 核を構成する各機能モジュールについて述べる．

4.1 割込み制御

実時間システムにおいては，非同期に発生する外部事象に対して高速に応答し，しかもそれらの事象に対応した信号を見失わないことが要求される．信号を見失わないためには，割込み禁止時間を最小割込み到着時間間隔よりも小さくしなければならない．しかし，すべての割込みに対して同一の処理方式を用いて，この制約に対処することは一般に不可能である．したがって，実時間制御用の OS では，割込み禁止時間を可能な限り割込み到着時間間隔に近づける試みがなされている．R² では，この要求を実現するために，他 OS と同様に各割込みに優先順位を割り当てることによってネストされた割込みの処理を可能としている．さらに，このアーキテクチャ以外に，各割込みに関連した処理を逐次的に行わずに割込み処理に対応して一つのタスクを起動する方式を実現している (R² では，これを "割込みタスク (interrupt task)" と呼んでいる)．これにより，割込み禁止時間を短縮している．**図 3** に割込みタスクの状態遷移を示す．図中，Xfork ( ) はユーザに公開されているC言語インタフェースの関数であり，割込みタスクを生成する SVC である．さらに，各状態は次節で述べるアプリケーションタスクの状態と同様の意味で用いている．

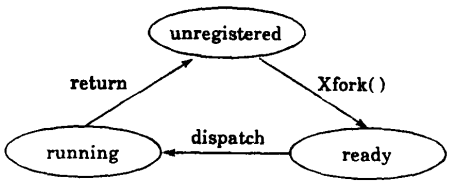


図 3 割込みタスクの状態遷移
Fig. 3 State transition of interrupt task.

割込み処理は，以下の 2 状態で行われる．

M状態：発生した割込みレベル以下の割込みが禁止された状態で処理が行われる．この間に上位のレベルの割込みが発生すると割込み処理はネストされる．

E状態：すべてのレベルの割込みが許可された状態で，割込みタスクとして処理が行われる．割込みタスクの実行中に発生する他の割込みの処理を行う場合，割込みタスクのキューが作られる．このキューは標準で二つ (優先キューと非優先キュー) あって，それぞれ発生順に並び逐次的に処理される．

M状態の処理は実測で 184 マイクロ秒，E状態の処理は 169 マイクロ秒となっており，全体で 353 マイクロ秒である．これは，MTOS-86 の 415 マイクロ秒と比較しても高速である．さらに，高速処理を必要とする場合は，M状態で行うことにより，MTOS-86 と比較して 215 マイクロ秒のオーバヘッドが削減できることになる．

R²-86 では，さらに，割込み制御自体の移植性と記述の容易性のために，割込み発生からMあるいはE状状態に遷移したり，MあるいはE状態から割込み処理を終了してスケジューラへ戻るためのインタフェース (関数 Xfork, Xsetenv_m Xexit_m) が用意されている．すなわち，レジスタの退避・回復，割込み処理のネスト，割込み処理用のデータ領域とスタック領域，割込みタスク，割込みフラグ，マスクベクトル等の割込み処理における環境を管理するモジュールを用意し，それらを用いて基本的にC言語で記述可能としている．

4.2 タスク管理

R² のタスクには，アプリケーションタスクと割込みタスクの 2 種類のタスクがある．どちらもタスクディスパッチャのスケジュールの対象となるプログラムであるが，割込みタスクの方がアプリケーションタスクに対して優先的にスケジュールされる．本節では，アプリケーションタスクの管理方式について述べる．

R² におけるタスクは 0 から 63 までの範囲の優先順位を持つことが可能であり (値の小さいほど優先度

が高い)，システム内のすべての動作可能なタスクは，この優先順位によってスケジュールされる．R² では，次に実行すべきタスクを選択するためのディスパッチングのアルゴリズムとして事象駆動 (event driven) 方式を採用している．以下に示す事象が生起すると，ディスパッチャが起動され優先度の高い順にタスクが実行される．

(1) 例外事象が発生した．
(2) 入出力装置から割込みが発生した．
(3) タスクが SVC を発行した．
(4) タスクが終了した．

R² では，各タスクはタスク識別子 (TID: Task IDentifier) と呼ばれる非負の整数によって一意に識別される．タスク識別子は，システム識別子 (プロセッサ識別子) とシステム内識別子から構成される．システムに登録可能なタスク数は，一つのプロセッサ上では最大 512 であり，複数プロセッサで構成されるシステムでは全体で 32,768 である．タスクは，以下の状態を遷移する (図 4 参照)．

(1) 実行状態 (running)

ディスパッチャにより CPU を割り当てられている状態である．この状態のタスクは，システムが停止状態である場合を除いて 1CPU 当たり一つだけ存在する．

(2) 実行可能状態 (ready)

CPU が割り当てられるのを待っている状態である．この状態のタスクは，優先度順にレディタスクキューにつながれる．

(3) 待ち状態 (waiting)

何らかの事象発生を待っている状態である．

(4) 中断状態 (suspended)

子タスクが，その親タスクにより強制的に実行を中断された状態である．

(5) 待ち中断状態 (wait suspended)

子タスクが，待ち状態中に親タスクにより中断された状態である．

(6) 未登録状態 (unregistered)

OS 核にタスクとして認識されていない状態，すなわちタスク制御ブロック TCB が作成されていない状態である．実行可能形式のプログラムは，初めはすべてこの状態である．

4.3 同期と通信制御

R² では，直接指名 (direct naming) 方式に基づいたタスクの同期と通信機能を実現している．すなわち，タスク間の同期および通信は，前述のタスク識別子を用いて相手を直接指定することにより行われる．

タスク間の同期は，タスク間で信号 (signal) を送受することにより，実現される．信号は，1ワード (16 ビット) として構成され，各ビットが一つの信号に対応している．受信側タスクは，送信された信号を保持するための信号バッファを持つ．信号が送信されると，信号バッファの該当するビットが1にセットされる．受信側がこの信号を受け付けると，そのビットはリセットされる．信号バッファは，各タスクごとに TCB 内に設けられる．さらに，TCB 内には受信タスクが待つ信号を記録するために，待ちパターンバッファが設けられている．また，その待ちパターンを AND 条件で待つか OR 条件で待つかを指定する条件フィールドも設けられている．このように，同期のための情報を，従来の OS に見られるような事象制御ブロックではなく，TCB に設定することによって，タスク間の同期の高速処理を可能としている．

待ち条件は，AND 条件の場合には，待ちパターンバッファに記録されたすべての信号が信号バッファにセットされているときに成立する．OR 条件の場合は，待ちパターンバッファのいずれかの信号が信号バッファにセットされているときに成立する．すなわち，C 言語で記述すると

```
((signal & pattern==pattern)
 && condition==AND)‖
```

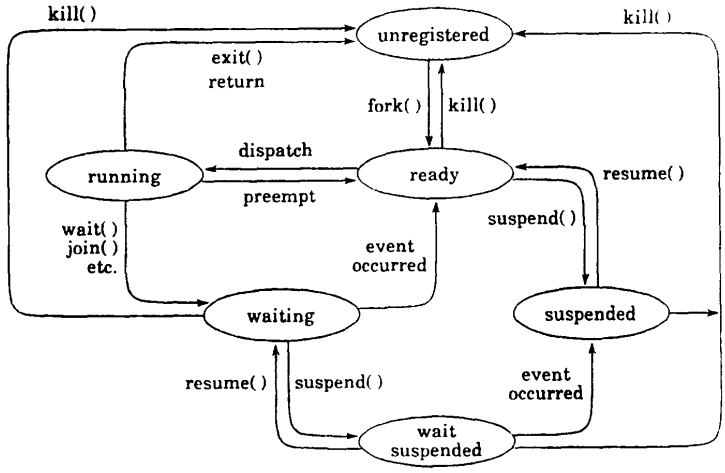


図 4 アプリケーションタスクの状態遷移
Fig. 4 State transition of application task.

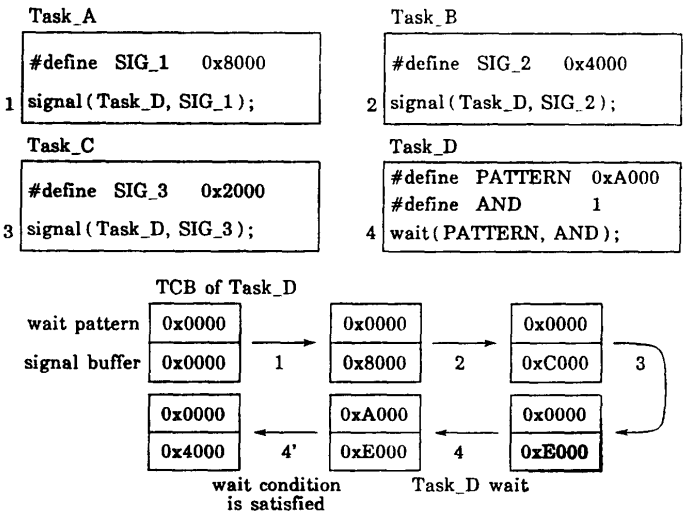


図 5 同期処理の例
Fig. 5 Example of synchronization processing.

((signal & pattern) && condition==OR)‖
pattern ==0

の場合に待ち条件が成立する．条件が成立すると，待ち条件を成立させた信号は信号バッファから除かれる．そして，待ちパターンバッファがクリアされる．以上の処理の過程を図 5 に示す．

タスク間の通信は，タスク間でメッセージを送受することにより実現される．R² における通信は，以下に示すように，多様な形態がサポートされている．ユーザはこれらの中からアプリケーションに合った方式を使用することにより，高速応答可能なシステムを構成できる．

（1）送信（同期送信，非同期送信）
（2）受信（同期受信，非同期受信）
（3）返信
（4）中継

同期送信は返信を待つことを意味し，同期受信はメッセージが送られてくるまで待つことを意味する．返信と中継は，同期送信とともに用いる機能である．すなわち，同期送信により送られてきたメッセージに対して応答を返すのが返信であり，その応答を他のタスク（子タスクに限定される）に委ねるのが中継である．

受信側に送られてきたメッセージは，受信側が所有する受信用バッファ（メイルボックス）に格納される．メッセージ長は，各メイルボックスごとに定義可能である．メイルボックスは，タスクが自身の領域をシステムに登録することによって作成される．メイルボックスは，メッセージ長ごとに区切られた連続領域である．各区画に対して CoCE (Communication Control Element) と呼ばれるシステムブロックが生成される．各 CoCE に対応する区画が使用中であるか否かによって，使用中メッセージキューあるいは未使用メッセージキューにつながれる．メイルボックスの各区画には，送られてきたメッセージの内容が書き込まれる．メイルボックスの容量(区画数)に関しては，送信側がすべて同期送信を行うのであれば 1 区画で十分である．しかし，送信側の多くが非同期送信を行うのであれば，容量を大きくすることが望ましい．

以上に述べた同期と通信機能は，シグナルバッファやメイルボックスをタスク対応に持たせることによって，プロセッサ間においても 1 プロセッサ内のそれらの処理と同様の形式で記述することが可能となっている．したがって，機能分散システムへの対応が容易である．

4.4 例外制御

R² で扱う例外には以下のものがある．

（1） タスクの実行に伴って発生する例外（内部例外）

（a） CPU例外

ゼロ除算やオーバフロー等のプログラムの実行中に内部割込みによって発生する例外．

（b） ソフトウェア例外

SVC の誤りや実行異常等によって発生する例外，およびシステムの用意する SVC (raise)[13]によってプログラム中で意図的に発生させる例外．

（2） タスクの実行と非同期に発生する例外（外部例外）

外部から割込みによって通知される事象の中で，システム生成時に定義された例外．

これらの例外が発生すると，正常なプログラムの流れは一時中断し，あらかじめ定められた手続きに制御が移行する．R² では，この手続きのことを例外ハンドラと呼んでいる．この例外ハンドラに，例外発生時の応急措置や回復措置等が記述される．例外ハンドラはタスク対応に定義可能であり，C言語における関数として記述される．さらに，例外に対する迅速な対応を可能とするために，例外ハンドラの実行をタスクの優先順位を変更して行うことも可能としている．

通常，例外発生時には直ちに例外ハンドラへ制御が移行する．しかし，当該タスクが例外ハンドラへの制御移行禁止状態にある場合は，その状態の解除後に移行する．また，外部例外が発生したタスクが待ち状態であった場合には，その待ち状態の解除後に制御が移行する．例外ハンドラは，発生した例外に対する何らかの処理を行った後で，正常なプログラムの流れに制御を復帰させる．このため2種類の方法が用意されている．一つは例外発生時の状態に戻る方法であり，もう一つはタスクを終了させる方法である．前者では例外が発生した次の命令からプログラムが続行される．後者の場合，SVC (raise) によってタスクを終了させると同時に親タスクに任意の例外を発生させることも可能となっている．

4.5 資源管理（排他制御）

R^2 が管理する共用資源には，プロセッサや入出力装置等のハードウェア資源のほか，関数やデータ等のソフトウェア資源も含まれる．R^2 では，これらの共用資源を使用目的に応じてグループに分割して管理することによって，資源の確保および解放処理を高速化している．各グループは互いに独立しており，あるグループのメンバが他のグループのメンバになることは許されない．

共用資源は，以下の資源識別子（RID: Resource IDentifier）によって一意に識別される（**図 6** 参照）．

RID=(GID, MID)

RID は，グループ識別子（GID: Group IDentifier）とメンバ識別子（MID: Member IDentifier）から構成される．GID は，システム生成時にユーザが指定する資源グループ単位に一つ割り付けられる．MID は，システム生成後，GID で示される各グループに属す資源を一意に識別するためのものである．同一グループ内の各資源は，当該グループのルートタスクと同じタスク階層に属すタスクが所有していなければならない．さらに，R^2 では資源管理のためのロック機構として排他ロック方式を採用しているが，本方式はデッドロックが生じる可能性があるので，これを回避するために各タスクは同時に唯一つの資源のみを占有するよう制限を課している．

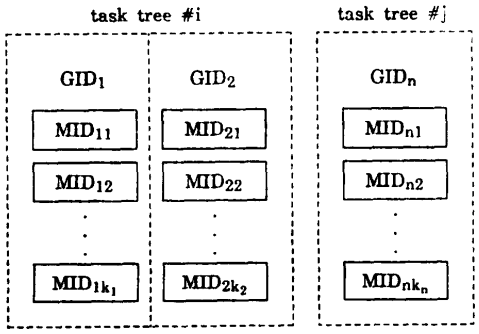


図 6 共用資源のグループ化
Fig. 6 Grouping of shared resources.

4.6 メモリ管理

タスクの実行時に，各タスクで一時的にメモリ領域が必要となる場合がある．R^2 では，このためにシステム生成時に主記憶領域の一部をメモリプールとして設定し，このグローバルなプールから各タスク対応に仮想的なプールを割り付ける機能を用意している．各タスクは，この仮想的なプールから必要となった時点で動的に要求した大きさの領域を得ることができる．この仮想プールは，連続したメモリ領域であり，可変長のフレーム(枠)の集合として定義される．フレーム長は，要求時に定義可能である（**図 7** 参照）．

仮想プール方式を採用した理由は，メモリプールを一括して管理する場合，各タスクで要求する領域の大きさが可変であるため，メモリコンパクションのための処理が複雑になり，そのオーバヘッドが無視できないことによる．

4.7 タイマ制御

R^2 のタイマ制御では，1ミリ秒を最小単位とするインターバルタイマを管理している．インターバルタイマの周期は，システム生成時に設定することが可能である．タイマ制御は，以下の処理を行う．

（1）システムタイマを更新する．

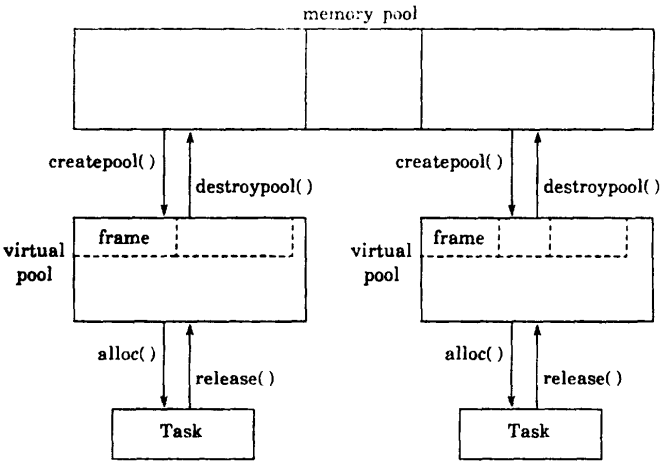


図 7 仮想メモリプールの管理
Fig. 7 Management of virtual memory pool.

（2）ユーザタイマを更新する．

（3）タスクポーズの終了を監視する．

（4）入出力およびプロセッサ間通信のタイムアウト監視を行う．

タイマ監視の SVC が発行されると，TWS (Time Wait Block) と呼ばれるシステムブロックを作成し，当該ブロックに監視終了予定のシステム時刻を設定し，時間順に TWQ (Time Wait Queue) と呼ばれるキューにつなぐ．タイマ割込みが発生すると，現在のシステム時刻と TWB の時刻を比較する．時刻が満了すると，TWB が TWQ からはずされる．この際，TWQ につながれている他の TWB の終了予定時刻の値には何ら影響を与えないため，時間監視の処理時間が短縮されている．このことは，システムタイマを十分長い周期（システムタイマの割込み間隔を 10 ミリ秒とすると 248 日）でサイクリックに更新し，TWB に設定されたシステム時刻もシステムタイマと同様の周期で更新する方式を採用していることから可能となっている．

4.8 入出力制御

ロボットシステムを初めとした各種産業用制御装置には，押しボタンスイッチやリレー，ランプ等の単純な入出力機器から，CRT，磁気バブル，磁気ディスク等の多種多様の機器が使用されている．しかも，実時間システムの特性から，これらの機器を高速にかつ非同期で制御することが要求される．R² では，これらの多様な機器の入出力制御に関して，

（1） 複数の入出力処理の同時実行を可能とする．

（2） 各機器に対応した入出力ドライバの作成を容易にする．

（3） アプリケーションプログラムから同様のインタフェースによって各機器をアクセスできるようにする．

の三つの設計目標を設定している．特に，(2)および(3)はシステムの移植性に関する要求であり，前述したとおりドライバをC言語で記述するためのインタフェースを提供している（表1参照）．

R² は，以上の目標を達成するために，次の3種類の入出力制御の形態をサポートしている．

（1） 標準入出力 (SIO: Standard I/O)

デバイスの初期化および起動，デバイスからの割込みの処理，入出力要求の完了処理，エラー処理からなる．

（2） ディスクリート入出力 (DIO: Discrete I/O)

入出力ポートの初期化，入力および出力，出力状態の確認の処理からなる．

（3） 入出力ポートの直接入出力(BIO: Basic I/O)

バイトあるいはワード単位の入力および出力の処理からなる．これは，機械命令で簡単に記述できるが，上記二つの形態と同様の記述形式とするため実行時ライブラリを呼び出す形で実現される．

SIO は，CRT，磁気ディスク等に関して R² で標準的にサポートしている入出力ドライバのための SVC である．ただし，ユーザはデバイスの初期化および起動，デバイスからの割込み等に関する処理の記述手順に従ってプログラムすることにより，ユーザ固有のドライバを構築することも可能となっている．この SIO においては，複数の入出力動作をチェーンすることが可能であり，個々の入出力動作をその都度呼び出す場合の SVC オーバヘッドを軽減している．

ディスクリートな入出力を行う場合，ビットフィールドが複数の入出力ポートに分散している場合がよくある．このことはハードウェアに対する依存性を助長していると言える．アプリケーションプログラムを作成する場合，このようなことは意識したくない．このため，R² では，最大四つの8ビット入出力ポートにまたがる DIO を，連続して1ワードの形式に変換するためのインタフェースを提供している．また，この DIO に関しては，SIO と同様に複数のタスク間で排他的に入出力動作を行う機能もサポートしている．

実時間システムでは，入出力処理の速度が非常に重要であり，入出力ごとの SVC オーバヘッドが問題となる．R² では，このため，単純なディスクリート入出力をアプリケーションプログラム中で，直接入出力ポート番号を指定して行うことを可能としている．すなわち，

```
input_data=bi (port_no)
bo (port_no, output_data)
```

のように BIO として記述する．ここで，port_no は入出力ポート番号である．

5. 性能評価

R²-86 核の性能評価のために，各種 SVC の実行時間を測定した．測定機器は，日本電気(株)製の PC 9801-VM 2 (CPU は V 30, クロックは 10 MHz) を用いた．さらに，C言語の処理系として，Lattice C V 3.00 (Lattice, Inc.) を用いた．測定は，R²-86 のタイマ割込みを禁止して，各 SVC を百万回ループさせ

表 2 R²-86 と MTOS-86 の SVC 実行時間の比較（単位はミリ秒）
Table 2 Comparison of the SVC execution time between R²-86 and MTOS-86. (Computed times are in msec.)

機能	SVC 名		実行時間				実行時間の比
			R²			MTOS	R²(3)/MTOS
	R²	MTOS	(1)	(2)	(3)		
プライオリティの変更	changepriority	cpri	0.744	0.420	0.407	0.419	0.97
タスクの起動	fork+join+exit	strt+endt	4.800	3.520	3.480	1.201	2.90
同期信号のリセット	reset	ref	0.240	0.242	0.230	0.439	0.52
同期操作	signal+wait	sef+wef+ref	1.190	0.950	0.870	1.812	0.48
同期信号のテスト	test	tef	0.212	0.212	0.212	0.423	0.50
通信（返信を待たない）注1)	send+receive	send+recv	1.697	1.370	1.300	1.119	1.16
通信（返信を待つ）注1)	send+receive+reply	send+recv注3)	2.517	1.900	1.780	2.176	0.82
資源の確保と解放	hold+free	tsf+rsf	1.230	0.850	0.800	0.846	0.95
メモリの確保と解放注2)	alloc+release	aloc+daloc	1.620	0.980	0.920	0.960	0.96

注 1) 通信のメッセージ長は2バイトである.
注 2) メモリ長は 64 バイトである.
注 3) send および recv を2回使用して実現した.
(1) デバックモード, Lattice C
(2) デバッグモード, Lattice C, Sオプション指定
(3) 実行モード, Lattice C, Sオプション指定

て実行し，PC 9801 のシステムクロック（秒単位）を使用して行った．さらに，他の実時間 OS との比較を行うために，MTOS-86 を PC 9801 上に移植し，同様の測定法によって R²-86 と類似の機能を実現する SVC の実行時間も測定した．これらの測定結果を表 2 に示す．ただし，表2の値は，百万回の実行に要した全体の時間を百万で割った値，すなわち1回の実行に要する時間である．

表2には，システム稼動中に頻繁に使用されシステムの性能を大きく左右することが予想されるタスク管理と同期および通信制御用の SVC を主に記載している．測定対象となる SVC には，一つのタスクでのみ測定可能なものと二つのタスク（親タスクと子タスク）を必要とするものがある．二つのタスクを使用した測定では，個々の SVC の実行時間ではなく共に使用されるであろう SVC の実行時間の合計を記載している．また，表2の括弧の意味は各々以下のようになっている．

(1) デバッグモード，Lattice C
(2) デバッグモード，Lattice C, S オプション指定
(3) 実行モード，Lattice C, S オプション指定

ただし，Lattice C の S オプションはポインタ操作の実行時ライブラリの呼び出しを禁止するオプションである．

表2より，(1)と(2)の差がかなりある．これは，(1)の場合において，ポインタの比較，関数の戻り値としてのポインタの返却，ポインタ同士の演算等は，そのたびに実行時ライブラリを呼び出しアドレスの正規化（セグメントアドレスとオフセットの調整）が行われることによる．これらの処理は，おのおの数十ステップのオーバヘッドとなる．

さらに，表2から分かるように，同期用の SVC に関しては R²-86 と MTOS-86 の実行時間の比は約 0.5 となっており，R²-86 の方が良いと言える．その他の SVC に関しては，(1)の場合，実行時間の比は約 1.5 となっており，逆に MTOS-86 の方が良くなっている．しかし，(2)の場合は，ほぼ同等となっている．さらに，R² においては，信頼性向上のためにタスク間の関係仕様を常に動的にチェックしており，そのためのオーバヘッドがある．このため，R² では関係仕様のチェックを取り外すためのオプションをサポートしている．これにより，R² のチェックを取り外した実行モード，すなわち表2の(3)の場合においては，タスクの生成を除けば MTOS と比較してほぼ同等かそれ以下となっており，移植性を追求しながらも，ある程度の応答性を達成していると言うことができる．

タスクの起動に関しては，実行時間は R² が MTOS の約3倍となっている．これは，R²-86 はルートタスクを除いて基本的には動的にタスクを生成するアーキテクチャを採用しており，MTOS-86 の方はシステム内に現れるタスクをあらかじめ静的に生成しておくアーキテクチャを採用していることの相違からきていると考えられる．すなわち，タスクの TCB の各項目を動的に設定するか静的に設定しておくかの相違によ

るものである．さらに，MTOS-86 では，タスクの消滅は単にタスクを登録状態に戻すだけであり，R²-86 におけるようにタスクの TCB を削除するような処理は行っていないことにも起因している．この問題に関しては，R²-86 の現在のバージョンではシステム生成時にルートタスクとその子タスクをすべて生成しておく方法しか考えられない．したがって，どうしても動的に生成しなければならないタスクに関しては，依然として問題が残る．このため，タスク生成の SVC である fork ( ) のほかに，静的に設定されたタスクを起動するための SVC である refork ( ) のサポートを検討中である．

6. おわりに

本論文では，R² の基本部分である R²-86 核の設計思想およびアーキテクチャについて述べた．R²-86 核は，機械依存部のアセンブリ言語記述を除いて，可能な限りC言語で記述されている．これは，OS 核自体の移植性を向上させることを目的としている．本論文では，特に，R²-86 の各 SVC の実行時間の測定を行った．さらに，アセンブリ言語記述の OS である MTOS-86 との性能比較も行った．R² の実行モードにおける各 SVC の実行時間は，タスクの生成を除けば MTOS-86 と比較してほぼ同等かそれ以下となっており，移植性を追求しながらも，ある程度の応答性を達成できたと言えよう．今回行った性能評価は SVC の実行時間の測定のみであり，今後は，実際の場面で複数の応用に R²-86 を適用し，システム全体の性能を評価していかなければならないと考えている．

参考文献

1) 機種対応に顔を揃えた16ビット・マイクロコンピュータ用 OS—充実するリアルタイム制御用 OS，日経エレクトロニクス，pp. 122-136 (1981. 10).
2) Intel Corp.: iRMX Nucleus Reference Manual (1981).
3) Industrial Programming Inc.: MTOS-86 User's Guide (1979).
4) 井口理博，沢田 勉，村田昌巳：応用面から見たモニタ機能と MKP 86 によるその実際，インタフェース，No. 98, pp. 206-247 (1985. 7).
5) Motorola Semiconductors Products Inc.: M 68000 Real-Time Multitasking Software User's Guide (1980).
6) 日立製作所：68000 RMS ユーザーズマニュアル (1982. 2).
7) 坂村 健：ITRON リアルタイムオペレーティングシステム—アーキテクチャと今後の展望，情報処理学会計算機アーキテクチャ研究会，61-1 (1986. 3).
8) 矢部栄一，竹山 寛，堀越健一：ITRON/68 K (ITOS 68 K)：ITRON を 68000 にインプリメントした実例，情報処理学会計算機アーキテクチャ研究会，61-2 (1986. 3).
9) 工藤健治，下原 明，安田秀徳，本田昭人：ITRON/MMU 286，情報処理学会計算機アーキテクチャ研究会，61-3 (1986. 3).
10) 坪田秀夫，中村和夫，榎本龍弥：NS 32032 上への ITRON インプリメントの検討，情報処理学会計算機アーキテクチャ研究会，61-4 (1986. 3).
11) 門田 浩，古城 隆，寺本雅則：V 60 用リアルタイム OS の検討—32 ビット I-TRON に向けて，情報処理学会計算機アーキテクチャ研究会，61-5 (1986. 3).
12) 櫛木好明：マイクロプロセッサ用オペレーティングシステムインタフェイス MOSI (IEEE 855) と ITRON の国際標準化について，情報処理学会計算機アーキテクチャ研究会，61-6 (1986. 3).
13) ADA ANSI/MIL-STD-1815 A-1983: The Programming Language Ada Reference Manual, in Goos, G. and Hartmanis, J. (eds.), *Lecture Notes in Computer Science*, Springer-Verlag, Berlin (1983).

(昭和61年10月20日受付)
(昭和62年 5 月13日採録)

大久保英嗣（正会員）

昭和26年生．昭和49年北海道大学理学部数学科卒業．昭和52年同大学工学部情報工学科大学院修士課程修了．同年(株)日立製作所ソフトウェア工場に入所．主として FORTRAN コンパイラの開発に従事．昭和54年より京都大学工学部情報工学科助手．昭和60年同講師，昭和62年同助教授，現在に至る．工学博士．オペレーティングシステム，データベースシステム等の研究に従事．日本ソフトウェア科学会，日本ロボット学会各会員．

津田 孝夫（正会員）

1932年生．1957年京都大学工学部電気工学科卒業．現職は京都大学工学部情報工学科教授．工学博士．現在の主要研究テーマは，メモリ階層間データ転送量の下界とそれによるアルゴリズムの最適化，ベクトル計算機のための自動ベクトル化の自動並列化，実時間オペレーティングシステムなど専用 OS の構成と実現法など．

楠田　修三（正会員）

昭和37年生．昭和60年京都大学工学部情報工学科卒業．昭和62年同大学院修士課程修了．同年日本アイ・ビー・エム(株)入社．オペレーティングシステム，コンピュータネットワーク，プログラミング言語等に興味を持つ．

小林　正典（学生会員）

昭和38年生．昭和61年京都大学工学部情報工学科卒業．同年同大学院修士課程入学．リアルタイムオペレーティングシステム，分散処理等に興味を持つ．

杉村　邦彦（正会員）

昭和27年生．昭和51年大阪大学基礎工学部制御工学科卒業．昭和53年同大学院修士課程修了．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．実時間システムのオペレーティングシステムおよびソフトウェア開発環境等に興味を持つ．

白濱　和人（正会員）

昭和34年生．昭和58年鹿児島大学工学部電気工学科卒業．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．プログラミング言語とその処理系，プログラミングツール等に興味を持つ．

友田　和伸（正会員）

昭和35年生．昭和59年愛媛大学工学部電気工学科卒業．同年(株)ダイヘン入社．メカトロ事業部技術部に所属．産業用ロボットの制御用ソフトウェアの開発に従事．実時間制御ソフトウェアに興味を持つ．