要約版

川崎製鉄技報
KAWASAKI STEEL GIHO
Vol.24 (1992) No.1

汎用インタネットワーキング装置の開発
Development of General Purpose Internetworking Unit

小川　哲男(Tetsuo Ogawa)　服部　憲昌(Norimasa Hattori)　峯岸　克己(Katsumi Minegishi)　上田　岳(Takeshi Ueda)　相川　俊之(Toshiyuki Aikawa)

要旨：
川崎製鉄では，任意のネットワーク間の接続を実現する汎用インターネットワーキング装置のプラットフォームを開発した。本プラットフォームは，複数のネットワーク・インタフェースから構成される。各インタフェース上の OS は，プロトコル処理の特性を考慮したダイナミック・タスク・スイッチング方式を採用し，プロトコルの並行処理を効率よく実現できる。さらに，今回開発したプロトコル生成高級言語により，処理効率のよいプロトコル・プログラムを効率よく開発できる。本プラットフォームに戦略的かつ効果的なインターネットワーク機能を搭載すれば，より高度なネットワーク・インテグレーションの実現が可能となる。

Synopsis：
A general purpose internetworking platform which permits the interconnection of any networks has developed by Kawasaki Steel. The platform consists of several network interfaces with an OS. The OS is designed to realize concurrent processing of protocols by a dynamic task switching architecture based upon features of protocol processing. A high-level protocol generation language was also developed, thereby improving protocol development. It is expected that a higher level of network integration can be realized by applying effective, strategic internetworking application to this platform.

(c)JFE Steel Corporation, 2003

本文は次のページから閲覧できます。

汎用インタネットワーキング装置の開発*

川崎製鉄技報
24 (1992) 1, 20-25

Development of General Purpose Internetworking Unit

小川 哲男
Tetsuo Ogawa
新事業本部 システム・エレクトロニクス事業部開発部 主査(課長)

服部 憲昌
Norimasa Hattori
新事業本部 システム・エレクトロニクス事業部開発部 主査(掛長)

峯岸 克己
Katsumi Minegishi
新事業本部 システム・エレクトロニクス事業部開発部 主査(掛長)

上田 岳
Takeshi Ueda
新事業本部 システム・エレクトロニクス事業部開発部

相川 俊之
Toshiyuki Aikawa
川鉄システム開発㈱
FA 開発部

要旨

川崎製鉄では，任意のネットワーク間の接続を実現する汎用インタネットワーキング装置のプラットフォームを開発した。本プラットフォームは，複数のネットワーク・インタフェースから構成される。各インタフェース上の OS は，プロトコル処理の特性を考慮したダイナミック・タスク・スイッチング方式を採用し，プロトコルの並行処理を効率よく実現できる。さらに，今回開発したプロトコル生成高級言語により，処理効率のよいプロトコル・プログラムを効率よく開発できる。本プラットフォームに戦略的かつ効果的なインタネットワーク機能を搭載すれば，より高度なネットワーク・インテグレーションの実現が可能となる。

Synopsis:

A general purpose internetworking platform which permits the interconnection of any networks has been developed by Kawasaki Steel. The platform consists of several network interfaces with an OS. The OS is designed to realize concurrent processing of protocols by a dynamic task switching architecture based upon features of protocol processing. A high-level protocol generating language was also developed, thereby improving protocol development. It is expected that a higher level of network integration can be realized by applying effective, strategic internetworking applications to this platform.

1 はじめに

近年，SIS（戦略的情報システム）という言葉がもてはやされ，また，最近これが実現化されつつある。この背景には，社会変化，技術革新といったものがある。高度情報化社会の実現に向けて，各分野での情報化の動きが活発になり，個別企業内の情報システムの構築から企業間，業界内から業界間へといった水平展開のネットワーク構築の段階にはいっている。また，「OSI の標準化」，「ISDN による通信の統合化」，「LAN による資源の共有化」といった技術面での進歩が，SIS 実現のためのネットワーク側面での重要なインフラとなっている[1,2]。

こうした状況下において，当社の情報・通信事業部門では，ネットワーク・インテグレーション事業の核となる技術として，「接続性」＋「管理性」を開発テーマに掲げた。ここで，「接続性」とは水平展開によるネットワークの拡張の基礎となる機器間あるいはネットワーク間の接続性の実現であり，「管理性」とはネットワークの拡大に伴う異機種，異ベンダおよび異ネットワークを統合した管理性の確保である。この２テーマを実現する手段として，汎用インタネットワーキング装置の開発を行った。

本報では，汎用インタネットワーキング装置を支えるインタネットワーキング OS の技術について述べる。

2 汎用インタネットワーキング装置の要件

現在の情報システムは，それぞれのベンダのさまざまな機種の情報機器で構成され，また，それらをつなぐネットワークも多様である。情報システムの企業間あるいは業界間への水平展開を考えた場合，これらのシステム間のインタオペラビリティ（相互運用性）の欠如が，重大な阻害要因となり発展を妨げることになる。さらに，ネットワークの拡大により統合的な管理の必要性が高まるが，現状では統一された管理体系を適用することが非常に困難であり，これも情報システムの水平展開の発展を阻害している。

そこで，任意の（any-to-any の）機器間あるいはネットワーク間の接続（インタネットワーキング）とその管理をコンセプトとして，汎用インタネットワーキング装置の開発に着手した。

事業展開の観点から，この装置には次の２点を求めた。

（1） 事業展開に合わせたタイムリーな商品化に役立つこと。

（2） 事業発展の seeds となりうること。

これらの要求に応えるために，以下の条件を挙げた。

* 平成３年11月20日原稿受付

(1) 各種の通信インタフェースに対応できること。たとえば，IEEE 802.3, V. 24, X. 21, RS-232 C, ISDN などや，さらに今後新たに生まれるもの。
(2) (1)の通信インタフェース上で各種のプロトコルが搭載できること。さらに，新プロトコルにも対応できること。
(3) プロトコル処理プログラム開発の生産性を高められること。
(4) ネットワーク管理などのアプリケーション的なシステムを搭載できること。

以上のような条件を満足するために，以下のアプローチをとった。
(1) 機能，負荷の分散を目指し，通信インタフェースごとにプロセッサ・ボードを用いる，マルチプルプロセッサ構成によるマルチプロセッシング方式。
(2) 単一プロセッサ内でマルチ・プロトコル環境を実現するため，また，ネットワーク管理などの付加的なアプリケーションを通信と同時に実行するためのマルチ・タスキング方式。
(3) (1)，(2)の基礎となる技術である，並行処理特有のプロセス間通信/同期，排他制御などを解決する手段としての，インタネットワーキング OS の搭載。
(4) プロトコル処理プログラム開発の効率化のための，プロトコル生成言語の高級言語化。

3 汎用インタネットワーキング装置の構成

3.1 ハードウェア構成

機能分散，負荷分散を実現するための，汎用インタネットワーキング装置のハードウェアの全体構成を **Fig. 1** に示す。

装置全体の制御を行うメイン・ボードと各通信インタフェースごとの通信を制御するインタフェース・ボードのそれぞれに CPU を搭載し，バスを介して接続した。各ボード上の CPU 間では，メイン・ボード上のシステム RAM を共有し，これを利用して情報交換を行う。各ボードごとに機能を分散するとともに，同種のボードを複数用いることにより，負荷の分散も可能な構成をとっている。

3.2 インタネットワーキング OS

プロトコル・タスクの効率的なマルチ・タスキングを実現するために今回開発したインタネットワーキング OS の機能構成を **Fig. 2** に示す。

複数のタスクを処理するためのスケジューラをカーネル部分に持ち，そのまわりでバッファやタイマなど各種の資源を管理する OS サービスが動作する。アプリケーション・タスクはこの OS 上で動作するが，プロトコル・タスクの場合には，プロトコル生成言語インタプリタを介して，OS サービスの提供を受ける。

3.3 アプリケーション

インタネットワーキング OS 上で動作させるアプリケーションとしては，次に示すような処理がある。
(1) プロトコル処理
(2) 装置モニタ
(3) 自己診断
(4) 障害処理
(5) ルーティング情報交換
(6) ネットワーク管理

これらのうち，特にこの装置の主アプリケーションとなるプロトコル処理プログラムの生成を効率化するために，プロトコル生成高級言語の開発を行った。

4 プロトコル処理とインタネットワーキング OS

4.1 プロトコル処理のモデル

インタネットワーキング装置で主として実行されるプロトコル処理をモデル化すると，**Fig. 3** のようになる。

各処理ステップの概略を次に示す。
(1) 受信処理
通信用 LSI で実行されるデータ受信処理（外部処理）および通信用 LSI からの受信完了割り込みを契機に，受信データを引き取り，中継処理に移る処理(内部処理)。
(2) 中継処理
受信データを解析し，中継すべきデータであれば，あて先のプロトコル処理タスクに送信を依頼する処理(内部処理)。
(3) 送信処理

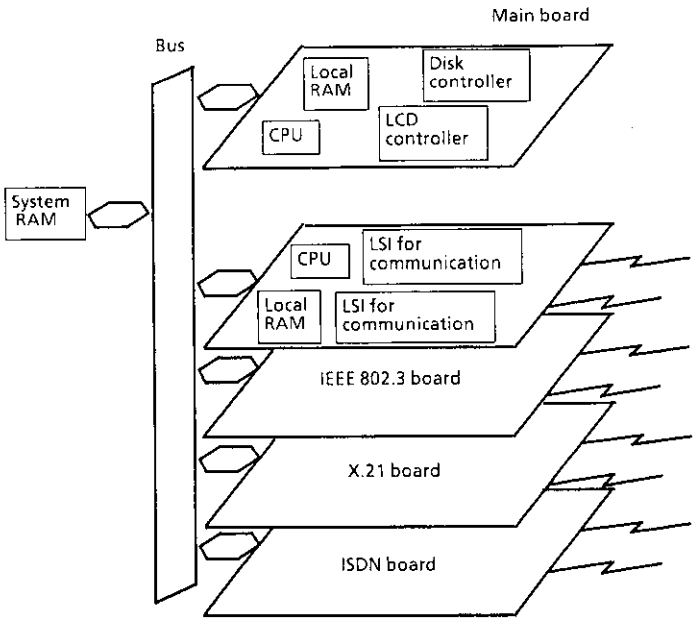

Fig. 1 Hardware block diagram of internetworking unit

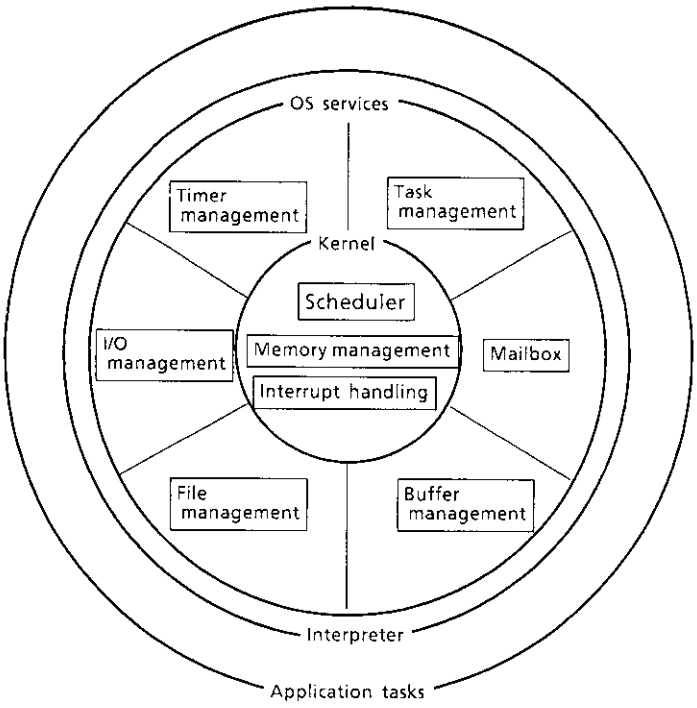

Fig. 2 Internetworking-OS functional structure

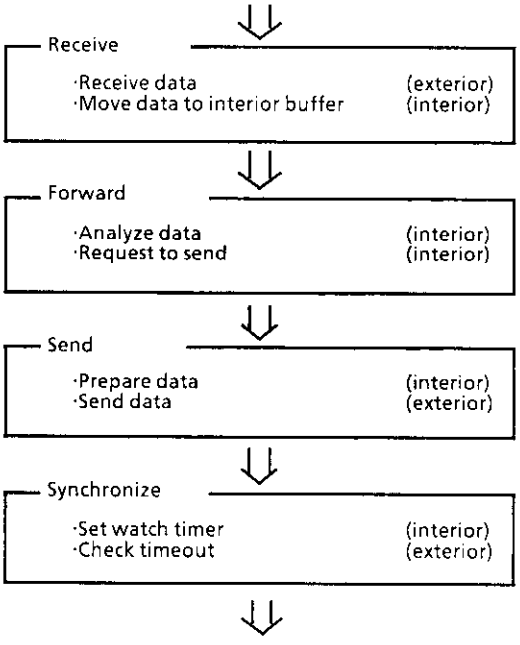

Fig. 3 Protocol process model

送信データの準備処理と通信用 LSI に対しデータ送信コマンドを発行する処理（内部処理）および通信用 LSI での送信処理(外部処理)。

（4） 同期処理

監視タイマをセットし，相手からの受信確認フレームの受信あるいは監視タイマの満了のいずれかの割込みを待つ処理（内部処理）と監視タイマの満了確認処理(外部処理)。

また，上記(1)～(4)の各処理ステップ中，常に通信回線の状況を観察し，重大な変化があった場合には，それに対処する処理が必要となる。

4.2 プロトコル処理の特性

前述のプロトコル処理モデルから，プロトコル処理の特性として以下の2点が挙げられる。

（1） 外部処理の割合が大きい。

（2） 多様なエラー処理が必要である。

通常，通信用LSIによる外部伝送処理は，伝送速度との関係から，CPU 内部処理と比較して非常に大きな時間を要する。また，同期のための監視タイマの満了時間も，内部処理に要する時間と比較して十分大きな値がとられる。すなわち，プロトコル処理は，非常に外部処理の割合が大きいという特性を持つ。この外部処理の間，プロトコル処理タスクは割込み待ち状態となるので，CPU 上では他のプロトコル処理タスクあるいは他のアプリケーション・タスクの実行が可能である。これを，**Fig. 4** に示す。

次に，プロトコル処理は常に通信路あるいは通信相手といった外部との関わりを持つ処理であるため，外部からのさまざまな例外要因に対処する必要がある。これらの例外は，外部からの割込みによって，エラーとしてプロトコル処理タスクに通知される。この割込みに際して，プロトコル処理タスクはエラー要因を解析し，それに対応する処理を実行しなければならないが，エラー要因が多岐にわ

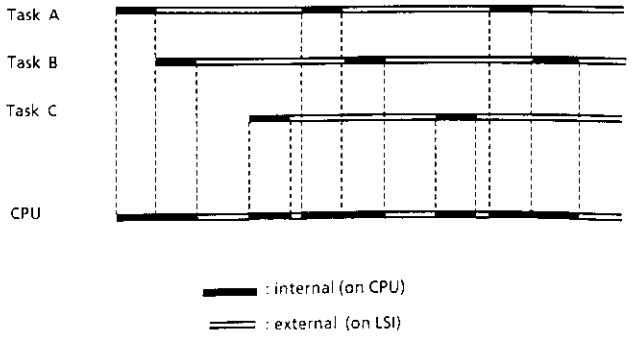

Fig. 4 Multi-tasking of protocol tasks

たるため，その解析は非常に複雑なものとなる。

以上のような特性をもつプロトコル処理タスク群を，効率的に並行処理するために，インタネットワーキングOSを開発した。このOSの特徴は，単純な制御構造による高速なタスク・スイッチング方式にある。

5 インタネットワーキングOSのしくみ

5.1 ダイナミック・タスク・スイッチング方式

今回，プロトコル処理タスクの並行処理を効率的に実現するために開発したタスク・スイッチング方式を，ここではダイナミック・タスク・スイッチング方式と呼ぶ。

一般に，タスクは **Fig. 5** に示すような状態間を遷移する。インタネットワーキングOSでは，タスクはイベント待ち（wait）状態には遷移せず，実行可能（ready）状態のままイベントの発生を待つ。

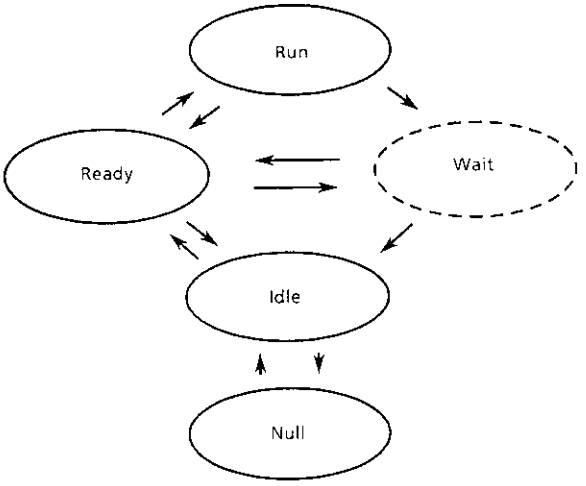

Fig. 5 Transition of task status

一般的なマルチタスクOSにおけるタスク・スイッチングは，タイマによる静的スイッチング方式（スタティック・タスク・スイッチング）が基本となっており，タスク側がタスク・スイッチングのタイミングを意識することはない[3~6]。これに対して，インタネットワーキングOSにおけるタスク・スイッチングは，タスクが能動的にタスク・スイッチングのタイミングを決定する。具体的には，実行中のタスクが自主的に実行権を放棄するコマンドを発行することにより，タスク・スイッチングが起こる。実行権を放棄したタスクは実行可能状態に遷移し，次の実行権の獲得を待つ。実行権は，基本的に実行中のタスクが放棄しないかぎり，他のタスクには渡らない。

このダイナミック・タスク・スイッチング方式のもとでのプロトコル・タスクの並行処理の効率化の実現については，6章に示す。

5.2 タスク状態キュー

ダイナミック・タスク・スイッチング方式を実現するためのデータ構造としては，巡回リスト構造[7]の実行可能状態キューのみを用意した。タスク状態キューの構成を **Fig. 6** に示す。このキューには，実行可能状態のタスク管理用データ構造（タスク・コントロール・ブロック）が並んでいる。タスク・スケジューラは，このキューに並ぶタスクのうち，実行ポインタの指すタスク（Fig. 6 中のタスクB）に実行権を与える。実行中タスクの実行権放棄コマンドにより，実行可能状態キューが巡回し，実行ポインタが指し変えられ，実行中のタスク・コントロール・ブロックはキューの最後尾にまわる。巡回の結果，実行ポインタの指したタスク・コントロール・ブロックに対応するタスク（Fig. 6 中のタスクC）が実行権を獲得する。キューの巡回は，実行ポインタの書換え処理のみで可能であり，非常に簡素な処理でタスク・スイッチングの準備ができる。

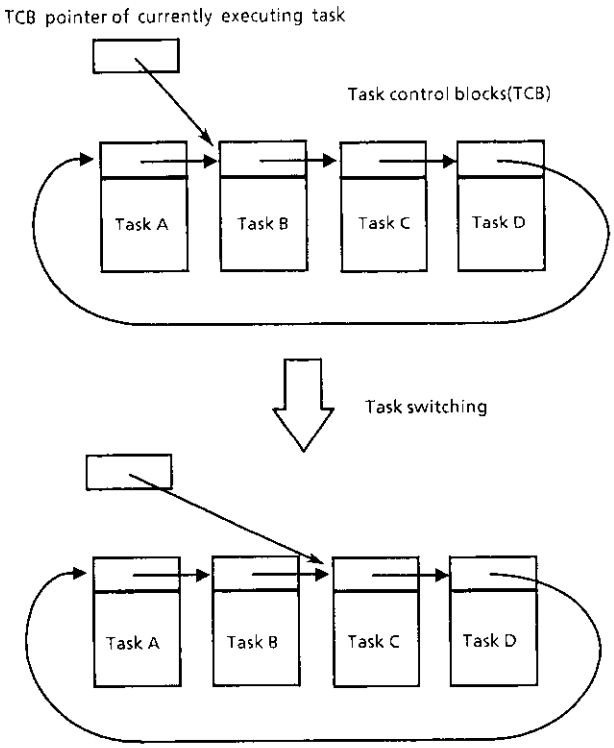

Fig. 6 Snapshot of TCB ready queue linkage in rotation fashion

6 プロトコル処理タスクの並行処理

6.1 イベント待ち動作

汎用的なOSの場合のイベント待ち動作では，タスクが待ちイベントをOSに登録し，イベント待ち状態に遷移するのが一般である[8,9]。イベントの発生時にはOSがその要因を解析し，イベント待ち状態のタスクを実行可能状態に遷移させる。タスクの状態遷移処理やイベント発生確認処理などを実行するため，OSの負荷が大きくなる。

ダイナミック・タスク・スイッチング方式では，OSは実行可能状態のタスクに対して順番に実行権を与えるのみであり，タスクがイベント待ちをしているのか否かの区別はしない。イベントの発生の確認は待ちタスク自身が行い，発生していない場合には即座に実行権を放棄する。

プロトコル処理では，以下のイベントを待つ状態が発生する。

（1） 送信完了（送信処理）

プロトコル処理タスクが通信用LSIに対してデータ送信コマンドを発行した後，LSIからの送信完了割り込みを待つ場合。

（2） データ受信（受信処理/同期処理）

プロトコル処理タスクが，受信データが届くことを期待して

LSI にデータ受信コマンドを発行する場合。特に，LSI からの送信完了割り込み受付後，通信の同期のために相手から送られてくる受信確認データの受信を待つ場合。

(3) 通信エラー

外部処理実行中に，なんらかの外部条件により通信の続行が不能となっていないかを確認する場合。

上述の(1)および(2)の場合，監視タイマをスタートさせ，LSI からの割込みとタイマからの割込みの両方を待つことになるが，いずれもプロトコル処理タスク側からのコマンド発行直後にイベント待ち状態が発生する。これに対して，(3)の場合には，通信路などの外部条件は逐次変化するため，常に状況を監視する必要があり，イベント待ち状態が続くと考えることができる。

ダイナミック・タスク・スイッチング方式では，この二つの異なる性質，すなわち，

(1) イベント待ち状態の発生が離散的でかつ予見できる。

(2) 常時イベント待ち状態である。

の両方のイベント待ち状態を持つタスクを，効率的に並行処理することができる。この実現イメージを **Fig. 7** に示す。

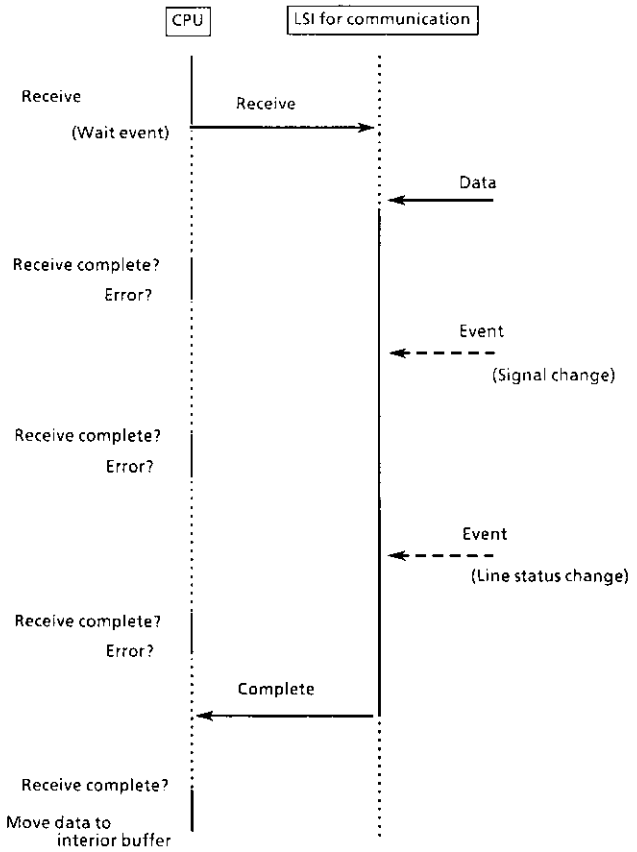

Fig. 7 Event wait mechanism between CPU and external events under "receive" semantics

6.2 スイッチングのオーバヘッド

タスク・スイッチングのオーバヘッドとして，次の二つが考えられる。

(1) 実行状態からの状態遷移のオーバヘッド

(2) イベント処理のオーバヘッド

これらのオーバヘッドについて考察する。

6.2.1 実行状態からの状態遷移のオーバヘッド

実行状態からイベント待ち状態に遷移する場合，待ちイベントの登録処理がある。プロトコル処理タスクのように，待つべきイベントの種類が多い場合には，それだけたくさんの登録処理が必要となるため，オーバヘッドも比較的大きなものとなる。

これに対し，実行状態から実行可能状態に遷移する場合，このような処理は不要で，OS は単にタスクから実行権を取り上げるのみである。したがって，ダイナミック・タスク・スイッチング方式における，イベント待ち時の状態遷移のオーバヘッドは，通常の OS 配下でのイベント待ち時よりも小さい。

6.2.2 イベント処理のオーバヘッド

イベント待ち状態に遷移する方式の場合，イベント発生確認は OS が行う。イベントが発生するまでは，イベント待ちタスクは実行可能状態にはなく，実行権を得る可能性はない。いっぽう，ダイナミック・タスク・スイッチング方式の場合は，イベント発生をタスクが確認するため，イベント待ちタスクにも実行権が渡る。イベント待ち状態に遷移する方式のほうが相対的に実行可能状態のタスク数が減るため，タスクの並行処理効率は良いと考えがちである。

しかし，イベント発生の確認およびその後の処理の起動を考えた場合，OS がこの処理をすべて受け持つのは相当の負荷となる。一つのイベントに対する待ちタスクは複数である可能性もあり，また，起動すべき処理も複数であり得る。複数のタスクが同時に複数のイベント待ちをする可能性の非常に高い環境においては，イベント処理が非常に複雑なものとなる。

これに対し，ダイナミック・タスク・スイッチング環境では，イベント発生確認をタスク自身が行う。タスク側からみれば，イベント後に起動すべき処理は決まっており，また，確認すべきイベントの種別も決まっているため，比較的効率よくイベント後の処理が実行できる。OS 側は，イベントの発生を全タスクに通知する手段をもちさえすれば，あとはただタスクに実行権を与えればよく，こちらも処理が単純になる。したがって，処理のオーバヘッドは小さくおさえることが可能である。

6.3 リアルタイム性の保証

ダイナミック・タスク・スイッチング方式では，タスク・スイッチングのタイミングをタスク側が決定するため，OS にとっては制御不能に陥る可能性を残している。すなわち，タスクが実行権を永久に保持したままであれば，システムは動作不能となる。その救済策として，本 OS では比較的長い時間間隔でのスタティック・タスク・スイッチングおよび実行権放棄コマンドのユーザ開放手段を備えている。

内部処理の割合の大きいアプリケーション・タスクの搭載に際しては，アプリケーション・プログラム生成時に，タスク・スイッチングのタイミングを考慮したプログラミングが求められる。実行権放棄コマンドを利用しながら，プロトコル処理のリアルタイム性を保証できる範囲において，タスク・スイッチングのタイミングを選ぶことが推奨される。

ただし，最悪の場合でもスタティック・タスク・スイッチングにより，プロトコル・タスクのリアルタイム性を保証しているので，システムとしての動作はプロトコルの要求を満足し得る。より効率的な並行処理のためには，実行権放棄コマンドを有効に利用していくことが必要であろう。

7 プロトコル生成高級言語

プロトコル生成高級言語は，プロトコル処理プログラムの開発に際し，プログラマがプロトコル処理の複雑さ，プロトコル処理タスクの並行処理に関して意識せずにプログラミングすることを可能とするために開発した言語である。プログラマは，プロトコル処理の流れにしたがって関数を並べるだけで，プロトコル処理プログラムの生成ができる。

先に述べたような，プロトコル・タスクの効率的な並行処理のためのダイナミック・タスク・スイッチングの実行も，高級言語インタプリタの生成するセマンティクス中に隠されている。すなわち，プロトコル生成高級言語のセマンティクスにおいては，イベント待ち状態が発生するコマンドの内部に，実行権を放棄するコマンドが存在しており，自動的にタスク・スイッチング処理が起動される。この結果，生成されたプロトコル処理プログラムは，あまり意識しなくても並行処理効率のよいものとなる。

このプロトコル生成高級言語により，汎用インタネットワーキング装置の拡張性，プロトコル生成効率の向上が実現される。

8 結　言

当社では，今後のネットワーク・インテグレーション事業の強力なツールとなる汎用インタネットワーキング装置を開発するために，そのプラットフォームとして，ハードウェア，インタネットワーキングOSおよびプロトコル生成高級言語を開発した。

本装置は，任意のネットワーク間の接続とそれらの管理を目的としたもので，各種のネットワークに対応するための通信インタフェース・ボードと装置の制御のためのメイン・ボードから構成される。各ボードに搭載したCPU上のインタネットワーキングOSの配下で，各種プロトコル・プログラムやその他のアプリケーション・プログラムがタスクとして動作する。

本開発の成果は以下のとおりである。

(1) ダイナミック・タスク・スイッチング方式を採用し，外部処理の割合が大きく，待ちイベントの種類の多いプロトコル処理タスクの並行処理の効率化を実現した。

(2) プロトコル処理プログラム開発者が，プロトコル処理の詳細および並行性を意識せずに，並行処理効率のよいプログラムを短期間で開発することを可能にした。

このプラットフォームに，今後さまざまなプロトコルおよびプロトコル変換機能を搭載し，顧客のニーズに柔軟に応えるためのツールとしていく。また，この装置にネットワーク管理機能などのアプリケーションを搭載することによって，より高いレベルのネットワーク・インテグレーションが可能となる。

参考文献

1) 石田晴久，徳田雄洋，徳田英幸：「コンピュータ・ネットワーク」，(1986)，[共立出版]
2) 松下　温：「コンピュータ・ネットワーク」，(1983)，[培風館]
3) A. S. Tanenbaum: "Operating Systems: Design and Inplementation", (1987), [Prentice-Hall]
4) M. J. Bach: 「UNIXカーネルの設計」，(1990)，[共立出版]
5) P. J. Fortier: 「分散型オペレーティング・システムの設計」，(1988)，[日経BP]
6) L. Bic and A. C. Shaw: 「オペレーティング・システムの論理設計」，(1989)，[日経BP]
7) A. V. Aho, J. E. Hopcroft, and J. D. Ullman: 「データ構造とアルゴリズム」，(1987)，[培風館]
8) 日本電気㈱：「RX 616カーネル・ユーザーズ・マニュアル」，(1987)
9) ㈱OTC：「pSOS+シリーズソフトウェア概説書」，(1988)