

ABCL / c + による X I N U の 実 現

土居範久、児玉靖司

慶応義塾大学情報科学研究所

米沢らの ABCL/1 を、手続きおよび関数をオブジェクトと統一するルーチンオブジェクトを導入することにより拡張しプロセスの世界を作り出すことを可能にした C 言語版並行オブジェクト指向言語 ABCL/c+ を設計し、オペレーティングシステム核を記述することを試みた。オペレーティングシステム核の素材としては、ベル研究所の D.Comer 等によって開発されたオペレーティングシステム XINU を用い、これを ABCL/c+ で全面的に書き換えることを行なった。その結果、並行オブジェクト指向言語を用いると、極めて見通しのよいオペレーティングシステム核が作れることが分かった。本稿では、まず、XINU を概説し、ABCL/c+ についての簡単な紹介をした後で、具体例を用いて ABCL/c+ によるオペレーティングシステム核の実現方法を示し、それらの結果について検討する。

An implementation of XINU using ABCL/c+

Norihiisa Doi and Yasusi Kodama

Institute of Information Science, Keio University

4-1-1 Hiyosi, Kohoku-ku, Yokohama, 223, Japan

The result of an experiment to develop an operating system kernel using the concurrent object oriented language called ABCL/c+ is described.

ABCL/c+ is an extended version of ABCL/1 by introducing the routine object which is the concept to unify procedures, functions and objects, and is based on the language C.

By using ABCL/c+ and the object oriented approach, the kernel of the operating system XINU, which is developed by D.Comer at Bell Laboratories, is rewritten. And it is found that if concurrent object oriented languages are adopted to develop operating system kernels, comprehensible, reliable operating system kernels can be gotten.

After the description of the outline of XINU and the introduction of ABCL/c+, an approach to implement an operating system kernel using ABCL/c+ is shown and examined.

1. はじめに

オブジェクト指向は、信頼性が高くしかも再利用性が高いプログラムを作ることができるパラダイムであるといわれており、Smalltalk-80[1], Loops[2], Esp[3]等各種のオブジェクト指向言語が開発、使用されてきている。さらに、並行プログラミングが可能な機能を持たせた、いわゆる並行オブジェクト指向言語もいくつか開発されている。代表的なものとしては、ABCL/1[4][5], ConcurrentSmalltalk[6], Orient84/K[7], Beta[8]などがある。これらでは、オブジェクトの定義とその個体(instance)が実行されるとき単位であるプロセスとを1対1に対応させており、極めて素直に、並行プロセスを記述することができる。

並行プロセスの記述が容易にできるとなると、システムプログラムやオペレーティングシステム核を記述するのに使うことが考えられる。そこで、我々は、米沢らのABCL/1を拡張しプロセスの世界を作り出すことを可能にしたC言語版並行オブジェクト指向言語ABCL/c+[10]を設計し、オペレーティングシステム核を記述することを試みた。オペレーティングシステム核の素材としては、XINU[11][12]を用い、これをABCL/c+で全面的に書き換えることを行なった。その結果、並行オブジェクト指向言語を用いると、極めて見通しのよいオペレーティングシステム核が作れることが分かった。

本稿では、これ以降、2.で、まず、XINUについて概説し、3.でABCL/c+についての簡単な紹介をした後で、4.でオペレーティングシステム核を記述した試みの結果について述べ、5.でその結果について検討を行なう。

2. オペレーティングシステムXINU

XINUは、ベル研究所のD.Comerらによって開発されているオペレーティングシステムである。XINUは、最近の巨大なコードによって記述されたオペレーティングシステムに対して、コード自身を大変コンパクトに、わかりやすく記述することを目標としている。実際、XINUの全体の構造は、図1のような階層構造として機能ごとにまとめられており、わかりやすくなっている。

そのコード自身は、C言語とアセンブリ言語によって大変コンパクトに記述されている。それにもかかわらず、全体の機能としては、UNIXライクな大変強力な機能をユーザーに提供している。その主だった機能をあげると、次のとおりである。

- (1) マルチタスク、マルチユーザシステムである
- (2) 計数セマフォによる同期処理を提供している
- (3) 2種類のプロセス間通信処理を提供している
- (4) 付属のシェルがUNIXライクなコマンドを実現している
- (5) イーサネットを通して、ネットワーク通信をすることができる

2.1 プロセスの状態遷移

XINUにおけるプロセスの状態遷移は、図2の通りである。プロセスは、create関数を呼び出すことによって生成する。生成直後のプロセスは、中断(suspend)状態になる。この時の状況(context)は、create関数の引数として指定する。このプロセスを実際に実行させるためには、resume関数によって、実行待ち(ready)状態にすればよい。実行待ちと実行との間の状態の遷移はスケジューラがとりしめることになる。このとき、スケジューラはresched関数を用いてスケジューリングする。スケジューリング方式とし

ハードウェア

記憶管理

プロセス管理

プロセスの協同

プロセス間通信

実時間時計管理

デバイス管理

ネットワーク通信

ファイルシステム

ユーザプログラム

図1 XINUの階層構造

ては、基本的には、優先度スケジューリング方式をとり、同じ優先度のプロセスに対しては、ラウンドロビン方式を用いている。アクティブな(実行待ち状態か、実行状態にある)プロセスを一時中断したい場合には、suspend関数を呼ぶことによって、中断することができる。以上が、XINUでの基本的なプロセスの状態遷移である。しかし、この他にも、ユーザーに提供している以下の機能を使用した場合には、その他の状態に移ることがある。

- (1) プロセスの協同
- (2) プロセス間通信
- (3) 実時間時計処理

プロセスの協同は、計数セマフォを用いてプロセス間での同期処理を行うための機能群である。プロセスがシグナルを待つ間セマフォ列(semaphore queue)に入れられる場合には、待機(wait)状態になる。プロセス間通信では、メッセージの受け手のプロセスが、メッセージの到着を待たなければならない場合に、受信待ち(received)状態になる。実時間時計処理では、指定した時間だけ処理を中断する場合に、そのプロセスは、休眠(sleeping)状態になる。

2.2 XINUが提供するサービス

XINUが提供する主なサービスのうち、並行処理、同期処理、プロセス間通信の三つについて簡単に述べておくと次のとおりである。

- (1) 並行処理

並行処理は、プロセスを生成し、resume関数で起動をかけることによって行なうことができる。

- (2) 同期処理

同期処理には、計数セマフォを用いる。

- (3) プロセス間通信

プロセス間通信は、以下の関数を呼び出すことにより行なうことができる。

```
send( pid, msg );  
receive();
```

さらに、ポートと呼ぶ通信用バッファを用いてプロセス間で通信を行なうこともできる。このポートは、ネットワークを介して、他のマシン上のプロセスとの間でメッセージ通信を行なうときにも用いる。

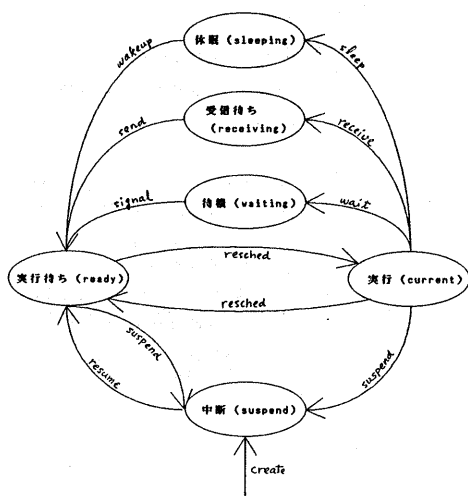


図2 XINUの状態遷移図

3. 並行オブジェクト指向言語ABCL/c+

米沢らのABCL/1がCommon Lispをベースにしたものであるのに対し、ABCL/c+はC言語をベースとしたものである。したがって、ABCL/c+は型を持つ言語である。

3.1 オブジェクトとメッセージ

各オブジェクトは、休眠、活性、待機の三つのモードのうちの一つをとる。これらのモード間の遷移を図示すると図3のようになる。また、後述のルーチンオブジェクトと各オブジェクトにはメッセージを到着順に保持する2種類の列がある。一つは通常のメッセージのための列であり、もう一つは通常のメッセージの処理中すなわち活性モード時に、その処理を中断して受け付ける緊急用のメッセージのための列である。前者を通常モードのメッセージ、後者を速達モードのメッセージと呼び、そのための列を、それぞれ、通常モードメッセージ列、速達モードメッセージ列と呼ぶ。

休眠モード時に、メッセージが到着した（到着していた）ときは、通常モードメッセージ列の先頭から調べ、最初の受理可能なメッセージを取り上げ、それ以前のメッセージはすべて除去する。待機モード時にメッセージが到着した（到着していた）ときは、通常モードメッセージ列を先頭から調べ、最初の受理可能なメッセージを取り上げ、それだけを列から除去する。

速達モードのメッセージは到着すると、速達モードメッセージ列に入れられ、受理可能であれば、直ちに受け付けられる。

ABCL/c+には、これらのオブジェクトに加えてルーチンオブジェクトと呼ぶ、メッセージ列を一切持たないオブジェクトがある。これは、手続きおよび関数をオブジェクトと統一するものであり、その個体はプロセスとしては存在しない。ルーチンオブジェクトは、後述の呼出し型のメッセージしか受け付けない。ルーチンオブジェクトは単独に定義できるだけでなく、ルーチンオブジェク

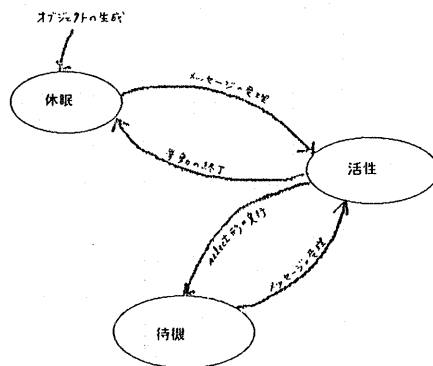


図3 ABCL/c+のモード間遷移図

トの定義は他のルーチンオブジェクトおよび通常のオブジェクトに入れ子にできる。このルーチンオブジェクトは、並行オブジェクトすなわち並行プロセスの世界を実現するためのオペレーティングシステム核をもこの言語で記述したいがために導入したものである（通常のオブジェクトはプロセスとして存在することから、現在型だけを受け付けるオブジェクトを定義することで、ルーチンオブジェクトに替えることは不可能である）。

通常のオブジェクト内にルーチンオブジェクトを定義したとき、ルーチンオブジェクトは内部手続きまたは内部関数として機能し、ABCL/1のルーチン[4]に相当する。

入れ子にしたときの名前の有効範囲は、いわゆるブロック構造の有効範囲規則に準じ、その直ぐ外側の環境で有効な名前は、それと同じ名前を内側のルーチンオブジェクトの局所名として宣言しない限り有効である。

3.2 メッセージ受渡しの型

オブジェクトが、他のオブジェクトにメッセージを受け渡すときの型として、ABCL/1と同様の過去型、現在型、未来型の三つの型[4]とルーチンオブジェクトに対する呼出し型とがある。送り手のオブジェクトをO、受けてのオブジェクトをT、メッセージをMとしたとき、これらの意味および通常モードおよび速達モードでメッセージを受け渡す時の構文は、図4の通りである。図のR、D、xは、それぞれ、返答、返答の宛先、未来変数である。

3.3 オブジェクトの定義

ABCL/c+では、通常のオブジェクトは、図5のような記法により定義する。また、ルーチンオブジェクトは、図6のような記法により定義する。

個々の詳細については次の通りである。

- (1) '状態変数'は、そのオブジェクトの内部状態を表す変数である。
- (2) オブジェクトは、'メッセージパターン'と合致し、しかも、'条件'を満たしたメッセージだけを受理する。メッセージパターンおよび条件は上から下に調べる。メッセージの'返答の宛先' (D) よび'未来変数' (x) が、'宛先変数'に束縛される。返答は、返答形を'挙動の記述'の中に書くことによって返すことができる。

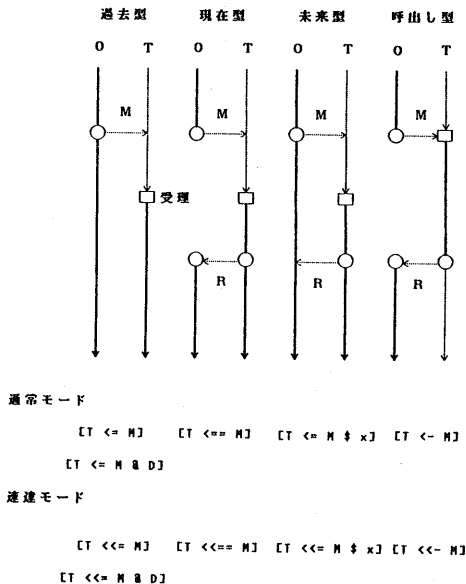


図4 メッセージの受渡しの型と構文

型/複合型の宣言

```

[object オブジェクト名
ルーチンオブジェクトの定義;
state (
状態変数の型 状態変数 := 初期値;
.
.
.
)
script (
(=>) メッセージパターン @ 宛先変数
where 条件 # 型
パターン変数の型宣言:...;
{
一時変数の宣言:...;
挙動の記述:...;
}
)
(=>) メッセージパターン @ 宛先変数
where 条件 # 型
パターン変数の型宣言:...;
{
一時変数の宣言:...;
挙動の記述:...;
}
)
)

```

図5 オブジェクトの定義

- (3) メッセージを受理すると、対応する'挙動の記述'に書かれた動作を順番に実行する。'一時変数'は、この動作の間にオブジェクトで使用する変数である。
- (4) '=>'に続くメッセージパターンが通常モードのメッセージに対するものであり、'=>>'に続くメッセージパターンが連達モードのメッセージに対するものである。呼出し型メッセージに対するメッセージパターンを指定する場合には、それぞれ、'->', '->>'を用いる。
- (5) '型'は、返答する値の型である。
- (6) 'パターン変数'は、メッセージパターンの中の要素として用いる変数である。

型/複合型の宣言

```

[object オブジェクト名
ルーチンオブジェクトの定義;
state (
状態変数の型 状態変数 := 初期値;
.
.
.
)
script (
(=>) メッセージパターン @ 宛先変数
where 条件 # 型
パターン変数の型宣言:...;
{
一時変数の宣言:...;
挙動の記述:...;
}
)
(=>) メッセージパターン @ 宛先変数
where 条件 # 型
パターン変数の型宣言:...;
{
一時変数の宣言:...;
挙動の記述:...;
}
)
)

```

図6 ルーチンオブジェクトの定義

3.4 C言語との関係

次の例外を除き、state部およびscript部では、原則として、C言語の構文を使うことができる[10]。

- (1) 代入演算子'='の代わりに':='を用いる
- (2) 配列の添字を指定する '[' および ']' の代わりに '(' および ')' を用いる
- (3) 関数を宣言することはできない

4. オペレーティングシステム核の記述の試み

我々は、ABCL/c+の有効性を確かめるために、オペレーティングシステムを実際に記述してみることを行なった。素材としてのオペレーティングシステム核としては、XINUを用いた。XINUはC言語を用いて書かれており、百数十個の関数からなる。XINUのソースファイルには、およそ5850行のCコードと650行のアセンブリコードがある。このうち、コメントを除くと、コードは、およそ4300行、アセンブリコードは550行ほどになる。システムは、図1に示したようにハードウェアおよびユーザプログラムも含めると10層に層分けされた層構造になっている。ここで、たとえば、プロセス管理層に属するモジュール(関数)およびデータの引用の関係を図示してみると図7のようになる。層構造化されているとはいえ、このように複雑な関係にあるモジュール群を理解することはそう容易なことではない。これを、基本的に、同じデータ構造を用い、機能的にも全く同じことを行なうという前提のもとで、ABCL/c+で書き換えてみた結果のオブジェクトおよびそれらの間でのメッセージ授受の関係を図示してみると図8のようになる。図8で、'記憶管理'から'実時間時計管理'までがプロセスの世界を作り上げるシステムオブジェクトで、各々はルーチンオブジェクトで実現されている。'セマフォの生成'より上がプロセスの世界で、各オブジェクト(ネットワーク通信を除く(4.2参照))は、それぞれ一つのプロセスに対応している。以下では、図1に示した'プロセス管理'および'ネットワーク通信'の二つの層を取り上げ比較検討してみる。

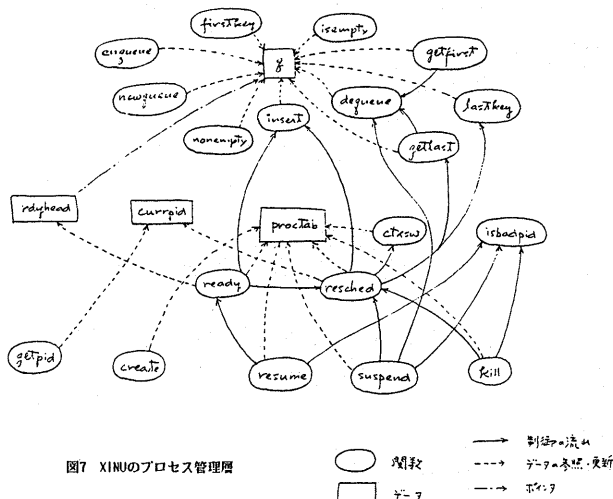


図7 XINUのプロセス管理層

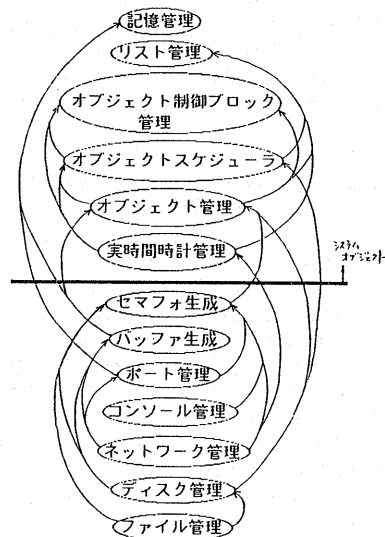


図8 オブジェクト間でのメッセージの授受関係

4. 1 プロセス管理

この階層での機能を大きく分けると次の四つになる。

- (1) リストの操作
- (2) プロセス制御ブロックの保持
- (3) プロセスのスケジューラ
- (4) プロセスの管理

(1) は、優先順位、前者、後者の三つの欄からなる配列 q を用いて、実行待ちリスト (ready list)、セマフォリスト (semaphore list) および時間を指定して眠っているプロセスを起こす時間順につなげたリスト (delta list と呼ぶ) の三つを管理するための機能群である。これらのリストは、それぞれ、前者欄と後者欄とを用いて双方向リストになっている。(2) は、プロセスを管理するための表 proctab である。(3) は、実行可能リストを用いてスケジューリングと状況の切替え (context switch) を行なう機能群と、現在実行中のプロセス一意名を保持する変数 currpid とからなる。(4) は、プロセスの出生・消滅、停止・続行などを扱う機能群である。

プロセス制御ブロック proctab、リスト用の配列 q などとは全域的なデータであり、たとえば、proctab を例にとると、この層で直接アクセスすることはもちろんのこと、この層よりも上の層である 'プロセス間通信' や、さらに上の層である '実時間時計管理' の層でも直接アクセスすることをしている。

このような、全域的なデータを用い、それを他の層で直接参照したり更新したりするようなやり方は、理解を困難にし、保守・改良を難しくするもの以外の何物でもない。

したがって、我々は、この層を、機能とデータに基づいて、次の四つのルーチンオブジェクトに分割した。

- (1) リスト管理
- (2) オブジェクト制御ブロック管理
- (3) オブジェクトスケジューラ
- (4) オブジェクト管理

ここで、(1)、(3)、(4) は、ほぼ前述の (1)、(3)、(4) に対応しているが、(2) はオブジェクト制御ブロック (プロセス制御ブロック) とそれをいじるための基本操作とからなるもので、もとの版には陽に存在しなかったものである。

たとえば、オブジェクトスケジューラを示すと図9の通りである。これはルーチンオブジェクトが入れ子になった形をしている。メッセージ: Initialize に対しては、リスト管理オブジェクト List にメッセージを送り実行待ちリストを作って、そのリストの先頭と最後尾とを ReadyHeadID と ReadyTailID に設定する。メッセージ: Ready .. では、オブジェクト一意名 ObjectID とスケジューラ仕直すかどうかを指定する論理値 ReScheduleBool とを引数として受け取る。そして、オブジェクト一意名の妥当性をチェックしてから、オブジェクト制御ブロック管理オブジェクト OCB にメッセージを送り、そのオブジェクトの状態を実行待ちにし、次に、リスト管理オブジェクト List にメッセージを送り、実行待ちリストにこのオブジェクトを挿入する。この時、OCB にメッセージを送り、このオブジェクトの優先順位をもらって List に送っている。実行待ちリストは、優先順位の低い順になっている。最後に、スケジューラ仕直すかどうかを確かめ、スケジューラ仕直す場合には、内部ルーチンオブジェクト Scheduler にスケジューラ仕直すよう要請する。メッセージ: ReSchedule に対しては、内部ルーチンオブジェクトを用いてスケジューラの仕直しをする。メッセージ: ReferCurrentObjectID に対しては、'現在のオブジェクト' の一意名 CurrentObjectID を返す。内部ルーチンオブジェクト Scheduler では、メッセージ: ReSched を受取ると、CurrentObjectID の状態が '現在 (CURRENT)' であり、しかもこのオブジェクトの優先順位が実行待ちリストの最後尾のオブジェクトの優先順位よりも高ければスケジューラの仕直しはしない。そうでなければ、状況を切り替える。まず、CurrentObjectID の状態が '現在' であれば (たとえば、実時間時計管理オブジェクトの Sleep メッセージの挙動では、CurrentObjectID の状態を '休眠' にしてから Reschedule メッセージを送ってくる)、その状態を '実行待ち (READY)' にし、そのオブジェクトの一意名と優先順位とを実行待ちリストに登録する。それから、実行待ちリストから優先順位が最も高いオブジェクトを取り出し、その状態を '現在' にする。そして、単位時間

分だけタイマを設定し、最後にアセンブリ言語を用いて実現されているctxswにより状況を切り替える。

このように入れ子になったルーチンオブジェクトを用いることにより、他のオブジェクトと統一した概念でスケジューラをコンパクトに記述することができる。

4. 2 ネットワーク通信

この層での機能を大きく分けると次の三つになる。

- (1) メッセージ通信
- (2) バッファプールの生成・管理
- (3) ポート管理

(1) は、ネットワークを経由してのプロセス間でのメッセージ通信を司る機能群で、多数のシステム関数といくつかのプロセスで構成されている。メッセージを回線の上ではブロックとして送受信するが、その間の変換の過程の概要を示すと図10のようになる。(2) は、(1) のデータリンク層およびディスク管理等で用いるバッファプールの生成およびそのプール内のバッファの割当て・解放を司る機能群である。(3) は、(1) のフレームおよびブロックを構成する基本的な論理構造である「ポート」の生成・管理を行なう機能群である。ポートは、1語を基本単位とするリストである。

そこで、我々は、(2)、(3) はそれぞれ一つのオブジェクトにし、(1) はいくつかのオブジェクトとして実現することにした。現在のところは、図10のトランスポート、インターネットおよびデータリンクはそれぞれ一つのオブジェクトとして、フレームは三つのオブジェクトとして実現している。

たとえば、バッファプール生成・管理機能のもと版におけるモジュール（関数）とデータおよびそれらの間での引用関係を図示してみると図11のようになる。バッファプール生成時に指定した大きさをもつバッファの集まりを動的に管理するのが、ここでの主な機能である。そのために、バッファプールの大きさ、空いているバッファへのポインタおよび割当て管理のためのセマフォをバッファプールごとに記録する表pbtabと、バッファプールの個数を数えておくnbpoolsとを用いている。

これに対し、ABCL/c++版では、バッファプールを一つのオブジェクトとして生成し、それを割り当てるようにしている。さらに、いちいちのバッファには、それが属するプールの一意名を持たせることで、他のプールに返却しようとする誤り等を防いでいる。バッファプール生成・管理オブジェクトBufferCreatorは、図12の通りである。

オブジェクトBufferCreatorは、InitializeメッセージとNewメッセージとを受理することができる。メッセージ:Initializeを受け取ると、現在用意されているバッファプールの個数を示す状態変数NumberOfBufferPoolsを0に初期設定する。メッセージ[:New ...]を受け取るとBufferSize（一つのバッファの大きさ）、NumberOfBuffers（バッファの個数）とからバッファプールの大きさを計算し、その大きさの記憶領域を割り当てる。実際の割当ては、記憶管理のためのシステムオブジェクトであるMemoryManagerに呼出し型メッセージを送ることによって行なう。このとき、おのののバッファにはポインタ部とプール一意名部とを追加する（2*SizeOf(int)）。MemoryManagerからは割り当てた領域の先頭のアドレスが返される。必要とするだけの記憶領域を割り当てることができる。状態変数NumberOfBufferPoolsを1増やしてから、バッファのポインタ部を用いてすべてのバッファ

```
#include <abc1.c.h>
#include <xinu.h>

[object Scheduler
[object Scheduler
script {
  (-> :ReSched # int
  {
    int OldObjectID;

    if( ( [OCB <- [:ReferState :At CurrentObjectID]]
        == CURRENT ) &&
        ( [List <- [:LastKey ReadyTailID]]
          < [OCB <- [:ReferPriority :At CurrentObjectID]] ) )
    {
      OK;

      if( [OCB <- [:ReferState :At CurrentObjectID]]
          == CURRENT ) {
        [OCB <- [:PutState READY :At CurrentObjectID]];
        [List <- [:Insert CurrentObjectID ReadyHeadID
                  [OCB <- [:ReferPriority :At CurrentObjectID]]]];
      }

      OldObjectID := CurrentObjectID;
      CurrentObjectID := [List <- [:GetLast ReadyTailID]];
      [OCB <- [:PutState CURRENT :At CurrentObjectID]];
      [Timer <- [:Quantum]];
      ctxsw( [OCB <- [:ReferObjectRegisters :At OldObjectID]],
             [OCB <- [:ReferObjectRegisters :At CurrentObjectID]] );
      OK;
    }
  }
}

state {
  int CurrentObjectID;
  int ReadyHeadID;
  int ReadyTailID;
}

script {
  (-> :Initialize
  {
    ReadyHeadID := [List <- [:NewQueue]];
    ReadyTailID := ReadyHeadID + 1;
  })

  (-> [:Ready ObjectID ReScheduleBool] # int
  int ObjectID;
  bool ReScheduleBool;

  if( isbadpid( ObjectID ) ) !SYS_ERR;
  [OCB <- [:PutState READY :At ObjectID]];
  [List <- [:Insert ObjectID ReadyHeadID
            [OCB <- [:ReferPriority :At CurrentObjectID]]]];

  if( ReScheduleBool ) [Schedule <- [:ReSched];
  OK;
}

(-> :ReSchedule
{
  [Schedule <- [:ReSched];
  OK;
})

(-> :ReferCurrentObjectID # int
{
  !CurrentObjectID;
})
}]
```

図9 オブジェクトスケジューラ

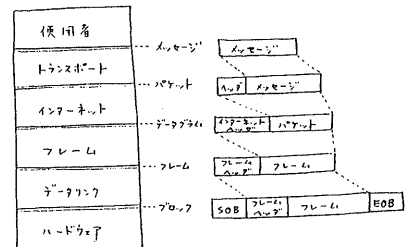


図10 ネットワークソフトウェアの階層および階層間でのデータ形式

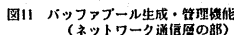


図12 バッファブール生成・管理オブジェクト

~7~

謝辞

本稿をまとめるに当たって数多くの有益なコメントを頂いた東京工業大学の米沢明憲氏に深く感謝の意を表する次第である。

参考文献

- [1]Goldberg,A. and Robinson,D., "Smalltalk-80 :The Language and Its Implementation," Addison-Wesley(1983).
- [2]Bobrow,D.G and Stefik,M., "The LOPS Manual," KB-VLSI-81-13, Palo Alto Research Center, Xerox PARC(1983).
- [3]Chikayama,T., "ESP Reference Manual," Technical Report, TR-044, ICOT(1984).
- [4]Shibayama,E. and Yonezawa,A., "ABCL/1 User's Guide," Information Science Department, Tokyo Institute of Technology(1986).
- [5]米沢明憲、柴山悦哉、J.-P.Briot、本田康晃、高田敏弘、"オブジェクト指向に基づく並列情報処理モデル ABCM/1とその記述言語ABCL/1"、コンピュータソフトウェア、Vol.3、No.3、9-23(1986).
- [6]横手靖彦、所真理雄、"並行オブジェクト指向言語 ConcurrentSmalltalk"、コンピュータソフトウェア、Vol.2、No.4、2-18(1985).
- [7]石川裕、所真理雄、"並行オブジェクト知識表現言語 Orient84/K"、コンピュータソフトウェア、Vol.3、No.3、24-42(1986).
- [8]Kristensen,B.B., Madsen,O.L., Moller Pedersen, B. and Nygaard,K., "Multisequential Execution in the BETA Programming Language," Sigplan Notice, Vol.20, No.4(1985).
- [9]土居範久、瀬川清、"Smalltalk-80による並行プログラミング"、コンピュータソフトウェア、Vol.3、No.1、18-34(1986).
- [10]土居範久、児玉靖司、"並行オブジェクト指向言語 ABCL/c+ 仕様書"、慶応義塾大学情報科学研究所(1987).
- [11]Comer,D., "Operating System Design: The XINU Approach," Prentice-Hall(1984).
- [12]Comer,D., "Operating System Design - Volume II: Internetworking with XINU," Prentice-Hall(1987).