オブジェクト指向に基づくＯＳのモジュール化について

4V-2

岩田憲和　小林吉純

NTT 電気通信研究所

1.はじめに

ｵﾍﾟﾚｰﾃｨﾝｸﾞｼｽﾃﾑ(OS)の設計に際して、次の 2つの問題が指摘されている[1]。

①構造ﾓﾃﾞﾙが明確でない。

OSでは構造ﾓﾃﾞﾙとして階層ﾓﾃﾞﾙが提唱されている[2]。例えば、ﾌｧｲﾙ管理はﾌﾟﾛｾｽ 管理の上位にあるとし、下位からは上位にある手続きを呼出すことはないとした。しかし、ﾊﾞｯﾌｧ 管理の位置がｼｽﾃﾑによってまちまちであったり、障害処理では階層関係は破られている等、実際のOSは階層ﾓﾃﾞﾙと相入れない面がある。

②客観的なﾓｼﾞｭｰﾙ設定基準がない。

障害管理のように処理の段階に対してﾓｼﾞｭｰﾙが設けられたり、ﾌｧｲﾙ管理のように制御対象に対して設けられたり、ﾓｼﾞｭｰﾙ設定基準が一貫していない。

これらの解決に向けて、ｵﾌﾞｼﾞｪｸﾄ指向という観点からﾓｼﾞｭｰﾙ設定基準として「個々の制御対象と制御対象の集合をｵﾌﾞｼﾞｪｸﾄとみなし、制御対象の種類毎にﾓｼﾞｭｰﾙを設定する」と規定し、ｵﾌﾞｼﾞｪｸﾄ指向でいうﾓｼﾞｭｰﾙ記述法（型を定義し、その型に作用できる手続きを同一のﾓｼﾞｭｰﾙ内に記述する）に従ってUNIXを再構成した。また、ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ、継承(ｲﾝﾍﾘﾀﾝｽ)の実現法を考案し、通常ｵﾌﾞｼﾞｪｸﾄ指向ｼｽﾃﾑで問題とされる性能低下を防いだ。この結果、以下の事項が明らかとなった。

・全域変数がないｲﾝﾀﾌｪｰｽ の明確な構造ができた。
・性能の低下は5%程度であると評価できた。
・構造ﾓﾃﾞﾙは階層ﾓﾃﾞﾙではなくﾓｼﾞｭｰﾙ間で互いに呼合う水平ﾓﾃﾞﾙ[3] となった。

本稿ではこれらについて報告する。

2.OS構造へのｵﾌﾞｼﾞｪｸﾄ指向概念の対応付け

ｵﾌﾞｼﾞｪｸﾄ指向の概念については誰もが認める定義はないがSMALLTALK-80が代表的ｼｽﾃﾑとされる。OSの概念と代表的ｵﾌﾞｼﾞｪｸﾄ指向ｼｽﾃﾑであるSMALLTALK-80での概念[4] とを次のように対応づけた。

・個々の制御対象（資源）をｲﾝｽﾀﾝｽ（固有ﾃﾞｰﾀと実行手順とが一体化したもの）
・制御対象の集合をｸﾗｽ （ｵﾌﾞｼﾞｪｸﾄの鋳型）

この対応は極めて自然である。例えば、ﾒﾓﾘ という制御対象を考えてみると、各ﾍﾟｰｼﾞ とその状態管理手続きに対応するのがｲﾝｽﾀﾝｽであり、ﾍﾟｰｼﾞ の払出しや回収等を行うﾍﾟｰｼﾞ の集合に対応するのがｸﾗｽ である。

3.ｵﾌﾞｼﾞｪｸﾄ指向の実現方法

ｵﾌﾞｼﾞｪｸﾄ指向は以下の性質の実現を要する。

①情報隠蔽　②ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ　③継承(ｲﾝﾍﾘﾀﾝｽ)

OSに適用する際これらの性質の効率的実現が必要である。以降では、これらの実現方法を述べる。このうち、①については手続き型言語によって従来通りの性能で実現できることが示されている[5]。ここでは、②、③の実現方法を述べる。

3.1 ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ の実現法

ｵﾌﾞｼﾞｪｸﾄ指向は、手続き名と実引数とから手続き本体を決定する方法が従来とは異なる。従来、手続き名から実引数の型を参考として手続き本体を定めた（手続き呼出し）のに対して、ｵﾌﾞｼﾞｪｸﾄ指向では実引数の型から手続き名を参考として手続き本体を定める(ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ)。しかし、実引数の型がﾌﾟﾛｸﾞﾗﾑ 記述の時点で定まっており、同じ名前の手続きが複数無いという条件の下では手続き本体をどちらの方法で定めても同じである。また、情報隠蔽を実現する上で自ﾓｼﾞｭｰﾙ内の変数を外部から操作されるようなことはあってはならないから、値渡しの手続き呼出しでなければならない。以上から、実引数の型がﾌﾟﾛｸﾞﾗﾑ 上で定まっており、同一名の手続きが複数ないという条件下では、ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ は値渡しの手続き呼出しである。従って、ﾒｯｾｰｼﾞﾊﾟｯｼﾝｸﾞ は値渡しの手続き呼出しで実現でき、このとき、従来の直接参照に対して手続き呼出しを介する手間だけしか遅くならない。

3.2 継承の実現法

継承は、別のﾓｼﾞｭｰﾙで宣言された内容（変数を除く型宣言と手続き宣言）を自ﾓｼﾞｭｰﾙに取込むことである。コンパイル時に行えば（Ｃ言語では#include文を用いて実現）、実行時の性能低下はおきない。

4.UNIX再構成の手順

「個々の制御対象(ｲﾝｽﾀﾝｽ)と制御対象の集合(ｸﾗｽ)をｵﾌﾞｼﾞｪｸﾄとみなし、制御対象の種類毎にﾓｼﾞｭｰﾙを設定する」という方針を立て、前記の実現法に従い、UNIXｶｰﾈﾙ（I/O 部を除く）の再構成を次の手順で行った。

①制御表の見直しを行う。

制御対象はﾌﾟﾛｸﾞﾗﾑ 上は制御表と呼ぶ構造体で表される。ｶｰﾈﾙを構成する構造体の各ﾌｨｰﾙﾄﾞの参照箇所を調査し、ひとまとまりとして操作されるﾌｨｰﾙﾄﾞを抽出し、構造体の再構成を行い、新たな制御表を作成する。制御表を一つの型として宣言し、その型（制御表）毎にﾓｼﾞｭｰﾙを設ける。

②手続きの見直しを行う。

制御表（制御対象）毎にそれに作用する手続きを作成する。再構成では、現状のUNIXｶｰﾈﾙの手続きを、何を対象に実行するかによって分割、統合した。作成した手続きを、①の制御表毎に宣言したﾓｼﾞｭｰﾙに記述する。

①と②から対象毎にﾓｼﾞｭｰﾙが作成できる。

Object-oriented Approach to Modular Operationg System

Norikazu IWATA , Yosizumi KOBAYASHI

NTT ECL

③ﾓｼﾞｭｰﾙ間で共用できる宣言を捜す。
幾つかのﾓｼﾞｭｰﾙで共通な部分をもつ制御表と手続きを抽出し、共用できる制御表部分と手続きとからなるﾓｼﾞｭｰﾙを作成する。このﾓｼﾞｭｰﾙは宣言の共用を目的とし、継承のために用いる。

5.再構成結果の評価

5.1 構造の比較

①ﾓｼﾞｭｰﾙ化によって全域変数がなくなり、構造（制御表とそれを参照する手続きの関係）が明確化した。図 1-1と図 1-2はUNIXのｶｰﾈﾙに登場する変数や構造体の各ﾌｨｰﾙﾄﾞがどの手続きで参照・更新しているかを点で表した図である。図 1-1は現状のUNIXの状況であり、点が縦に長く並んでおり制御表がｸﾞﾛｰﾊﾞﾙ なことを表す。図 1-2はﾓｼﾞｭｰﾙ化後のUNIXを表し、点の分布がﾌﾞﾛｯｸ 化されており、直接に参照・更新される制御表が局所化していることを表す。このﾌﾞﾛｯｸ がﾓｼﾞｭｰﾙであり、ﾓｼﾞｭｰﾙのｲﾝﾀﾌｪｰｽ はﾌﾞﾛｯｸ と交わる横軸（手続き）である。
②従来と比べて、手続き呼出しの実行時のﾈｽﾄ が深くなる。これは従来、ひとつの手続き内で直接に制御表を操作していたのに対して、ｵﾌﾞｼﾞｪｸﾄ間で処理依頼を次々と行っていくことに起因する。
③ﾓｼﾞｭｰﾙ化した場合、現状に比べて制御表のﾌｨｰﾙﾄﾞの個数の半数程度の手続きが増加した。これは各制御表のﾌｨｰﾙﾄﾞに外部からｱｸｾｽするための手続きである。
④構造は、階層ﾓﾃﾞﾙではなく水平ﾓﾃﾞﾙに合致する。ﾌﾟﾛｾｽ 、実記憶、ﾌｧｲﾙ、ﾊﾞｯﾌｧ 等の管理ﾓｼﾞｭｰﾙは互いに呼合っており上下関係はない。

5.2 性能の評価

ﾓｼﾞｭｰﾙ化によって全域変数が無くなった代わりにｵﾌﾞｼﾞｪｸﾄ間の情報授受は手続き呼出しで行う。この結果、手続き呼出しの静的な個数は、従来と比較して18% 増となる。これは68000 ｱｾﾝﾌﾞﾗ命令数では、0.6%増となる。手続き呼出しの命令実行時間はその他の命令の実行時間10倍と仮定すると、実行時間は従来と比較して5%増となる。

6.おわりに

ｵﾌﾞｼﾞｪｸﾄ指向という考え方からﾓｼﾞｭｰﾙ化したOSについて机上評価を行った。今後、以下について行う。
①ﾓｼﾞｭｰﾙ化したUNIXの性能の実測評価を行う。
②継承の有効性について、機能拡張という面から評価する。

【参考文献】

[1] 田胡和哉、益田隆司：ｵﾍﾟﾚｰﾃｨﾝｸﾞｼｽﾃﾑの記述に関する一試み、情報処理学会論文誌、Vol.25、No.4(1984)pp.524-535.
[2] ﾏﾄﾞﾆｯｸ、ﾄﾞﾉﾊﾞﾝ：ｵﾍﾟﾚｰﾃｨﾝｸﾞｼｽﾃﾑ、日本ｺﾝﾋﾟｭｰﾀ協会(1976).
[3] 情報処理学会規格委員会OSｲﾝﾀﾌｪｰｽ 専門委員会：ｼｽﾃﾑｿﾌﾄｳｪｱの標準化に関する調査研究、昭和61年 3月.
[4] Adele Goldberg and David Robson: Smalltalk-80 The language and its Implementation, Addsion-Wesley(1983).
[5] 佐渡一広、米沢明憲：抽象データ型言語、情報処理、Vol.22、No.6(1981)pp.525-530.

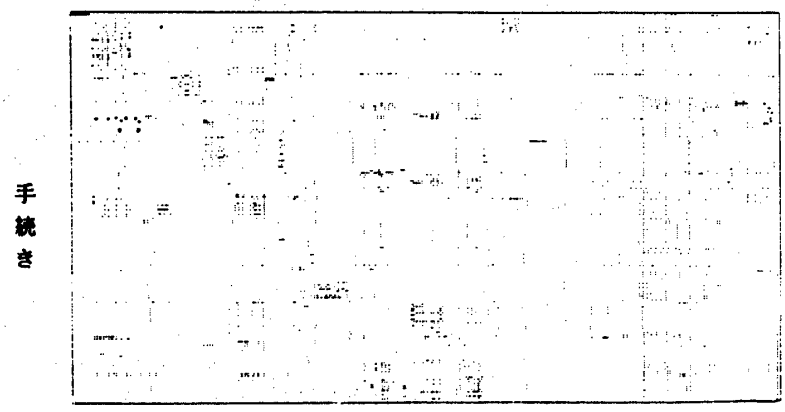

図1-1 UNIXの構成

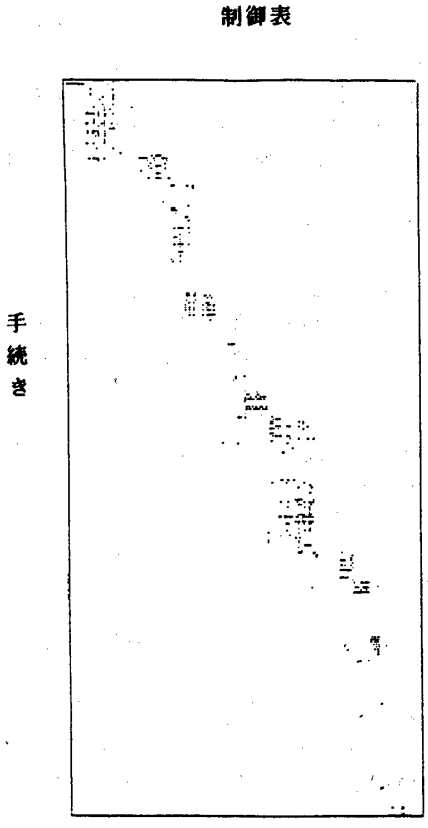

図1-2 ﾓｼﾞｭｰﾙ化した構成

ﾓｼﾞｭｰﾙ化によって制御表の参照・更新を表す点の分布がﾌﾞﾛｯｸ 化しｸﾞﾛｰﾊﾞﾙ な制御表がなく、良い構造になった。