オペレーティング・システムの論理記述方式の提案

田胡　和哉　　益田　隆司
（筑波大学）

１．はじめに

近年，半導体技術の進歩はめざましく，それにともなってハードウェア性能の向上，および，価格の低下が著しい。このため，計算機システムを，複数のプロセサにより構成し，機能，性能の向上を図ることが容易になりつつある。しかしながら，このようなシステムでは，ソフトウェアが複雑になりやすく，特に，OS (Operating Systems) は，従来の単一プロセサの場合に比べて，通信機能，スケジューリング機能等をより強化する必要があり，きわめて複雑なものになりやすい[6]。したがって，今後，OSを容易に設計，実現する方式を開発することが，より重要な課題になるものと予想される。

OSの論理構造を明らかにすることを通じて，OS設計，実現の容易化を図ることを目的とした最初の試みは，DijkstraによるＴＨＥシステム[1]である。その後，HansenのSoloシステム[3]，HoltのTunisシステム[4]等が開発されたが，このような設計法が広く採用されているとは必ずしも言いがたい。

これは，1)OSの論理構造が充分明確にされていないこと，2)性能が低下しやすいこと，等の理由であると考えられる。そこで，我々は，これらの点の改善を図ることを試みている。具体的には，1)相互排除アクセスされる資源の各々を別個のプロセスで管理し，それらを通信で結合することにより，論理的に明確な構造を持つOSを設計すること，2)既存のOS，ここではＵＮＩＸ†[5]とまったく同一の外部仕様を持つOSを提案方式により再設計，実現し，既存システムとの比較を行うことにより，利点，性能の客観的な評価を行うこと，3)評価結果をもとに改善を行い，実用性のある方式を得ること，を試みている。[7][8][9][10]

今回，UNIXと同一の外部仕様を持つシステムの実現がほぼ完了したので，その設計方式，記述言語，利点，性能の概要等について報告する。

２．設計，実現方式の概要

２．１　設計方式

提案方式によるOSは，図1.に示すような構造を持つ。すなわち，OSをプロセス（システム・プロセス）の集合と，プロセス切り換え，割り込み処理等を実現するOS核から構成する。システム・プロセスは，ユーザ・プロセスと異なり，実行時に生成されることはなく，また，より高速に切り換えられる。各ユーザ・プロセスは，それと1対1に対応するシステム・プロセスによって管理される。これを，スーパバイザ・プロセスと名づける。スーパバイザ・プロセスは，ユーザ・プロセスへの資源割り当て状況から優先度を計算し，OS核に実行を依頼する。ユーザ・プロセスの発行するシステム・コールによる割り込みは，OS核が通信に変換し，スーパバイザ・プロセスに伝達する。スーパバイザ・プロセスは，その内容を分析して，要求された資源を管理す

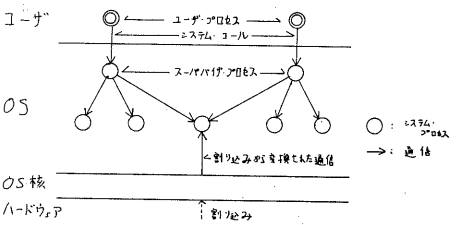

図１　提案方式によるOSの構造

†ＵＮＩＸ is a trade mark of Bell Lab. Inc.

るシステム・プロセスに対する要求に変換する。スーパバイザ・プロセス以外のシステム・プロセスは，記憶資源，ファイル，ファイル入出力のためのバッファ等，相互排除アクセスされる資源の各々を管理しており，これらの資源へのアクセスは，通信によって行われる。また，ディスク装置等の周辺機器も各々プロセスによって管理される。たとえば，ファイル入出力の際にOS内部で行われるバッファリングは，図2のようにして実現される。バッファリング・システムを利用するプロセスは，バッファ管理プロセスに求める二次記憶領域の内容を保持しているバッファ・プロセス名を問いあわせた後，そのプロセスにアクセスする。バッファの内容は，通信により利用するプロセスに伝達される。

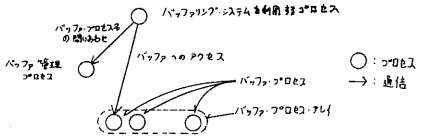

図2　プロセスによるバッファリング・システム

このような設計法は，従来の，たとえばモニタ（monitor）を用いる方式に比較して，制御構造がより明確化すると考えられる。モニタによる方式では，資源の管理が手続きの系列により実現され，動作の中断を必要とするソフトウェア割り込み等の非決定性を持つ処理の実現が複雑化しやすい。これに対して，プロセスによる方式では，資源管理単位に制御が独立しているので，非決定的動作による影響が広範囲に広がりにくく，制御構造を明確化しやすい。

また，通信によって結合されたプロセスは，変数，制御の両面において互いに独立なので，プロセスを単位とする記述を行うことにより高いモジュラリティが得られると期待される。さらに，この性質は，分散処理システムへの応用の際に有用である。

2.2　実現方式の概要

UNIXと同一の外部仕様を持つOSを実現し，従来システムと比較することにより，提案方式の利点，性能等を客観的に評価することを試みている。実現に際しては，比較を容易に行う立場から，従来と同一のハードウェアを用い，システム・プロセスはUNIXの記述言語であるC言語を用いて記述している。OS核は，アセンブリ言語で記述される。

システム・プロセス間の通信方式としては，Ada言語で提案されたランデブ（rendezvous）[2]を用いた。これは，第一に，send/receive等の方式に比較して，同期がより明確に成立すること，第二に，性能向上の余地が大きいことによる。

ランデブの実行，ユーザ・プロセスの実行等は，システム・プロセスが手続き呼び出しの形でOS核に依頼することにより実現される。この手続きを核プリミティブと名づける。表1に示すように，callプリミティブによりランデブ呼び出し（rendezvous call），accプリミティブにより受け付け（accept），runプリミティブによりユーザの実行が行われる。

通信の実行に関連する核プリミティブ	call（呼び出し）	acc（受け付け） end（終了）
	deadlock（デッドロックの判定）	
ユーザ管理に関連する核プリミティブ	run（ユーザの実行）	halt（ユーザの停止）
	time（ユーザ・プロセスのプロセッサ使用時間の計算等）	

表1　核プリミティブ

システムの記述は，プロセスを単位とする記述を行うと同時に，プロセス間の参照関係をプロセス内部の記述とは別個に記述する言語PNL（Process Network Language）を設計，実現し，用いている。

各プロセス内部は，C言語の手続き

の集合により構成され，通信のためのエントリ名，および，プログラム名のみを外部に輸出する。また，ライブラリ手続き名，および，通信相手名のみを外部から輸入する。したがって，プロセス間では変数を共有せず，また，ライブラリ以外の手続きを共有しない。このため，高いモジュラリティが期待できる。さらに，このとき，プロセス内部とプロセス間の関係を分離して記述することにより，より高いモジュラリティを得ることをめざす。プロセスが，他のプロセスに対して通信を行う場合には，通信相手のエントリ名を直接輸入するのではなく，そのプロセス内でのみ使用される仮のエントリ名を定義し，その名前に対する通信として記述する。この仮の名前は，プロセス内部の記述とは独立に，PNLを用いて通信相手のプロセスのエントリに結合する。

このような記述法の利点は，第一に，プロセスを部品として扱うことにより，システムの設計，実現が容易になることである。各プロセスは，他のプロセスで定義された名前を直接には輸入しないので，他のプロセスとは完全に独立に記述，コンパイルが可能である。このため，各プロセスの設計，実現の段階と，プロセスを組み合わせることによるシステム設計，実現の段階を分離できる。また，デバッグにおいても，各プロセスのデバッグとシステムのデバッグを分離することにより，各作業を単純，軽減化することができる。

提案方式の第二の利点は，システム構造の把握が容易になる点である。提案方式では，プロセスのみがシステムの構成要素であり，プロセスの組み合わせ構造としてシステムの構造を抽象化できると考えられる。PNLは，この組み合わせ構造を記述するものであり，ここから，システムの理解，および，変更のために有用な情報が得られる。

これに対して，モニタによる方式では，手続き呼び出しによる抽象化，階層化を行うことにより，設計，実現の容易化を試みている。そこでは，一般に，モジュール間の関係は，外部名の輸入，輸出の陽な宣言により記述される。上位のモジュール中で，より原始的な機能を実現するモジュール中の手続き名の輸入を宣言することにより，モジュール間関係の階層性を記述している。しかしながら，並行動作システムでは，非決定性動作を実現する必要があるため，手続き呼び出しの階層性は直ちに機能の階層性に結び付かない。たとえば，端末制御プログラムは，手続き呼び出しにより，機能的により上位のユーザ・プロセス管理プログラムに対して端末からの割り込みを通知しなければならない。このため，外部名の輸入，輸出の宣言による方法は，非決定的動作をともなう並行動作システムの機能の階層を記述するには不充分であり，設計，実現をむつかしくする原因になっていると考えられる。また，モニタのみの集合によってOSを実現することは不可能で，システムを構成するモジュールの性質が多様になり，かつ，モジュール間の関係が外部名の輸入，輸出の宣言，および，手続き呼び出しの形でモジュール内部にうめ込まれて記述されているので，システム全体の構造を把握することが容易でない。

提案方式は，従来の方式に比較して，以上のような利点を持つ一方，多数のプロセスが互いに関連を持って動作するためにデッドロックを生ずる危険があること，プロセス間が通信で結合されているために性能の低下をきたしやすい等の欠点がある。しかしながら，今回，それらの欠点を改善し，実用性のあるOSを実現するめどが得られたの

で，以下で報告する。

3．UNIXの再設計

2．で述べた方式にしたがい，UNIX Ver. 6 システムを再設計した結果，図3に示すようなプロセスの配置を得た。

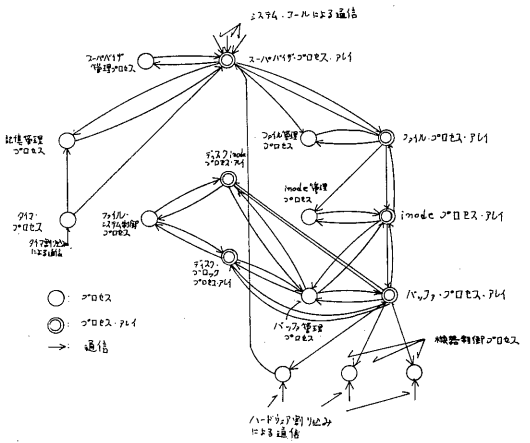

図3　再設計されたUNIXのプロセス配置

スーパバイザ管理，記憶管理，タイマの各プロセス，および，スーパバイザ・プロセス・アレイにより，ユーザ・プロセスへの資源割り当てをスケジュールする，いわゆるユーザ管理機能を実現している。残りのプロセスは，ファイル・システムを実現している。

スーパバイザ管理プロセスは，スーパバイザ・プロセスの集合（スーパバイザ・プロセス・アレイ）に属する1つのスーパバイザ・プロセスを選択し，ユーザ・プロセスの実行を依頼する。スーパバイザ・プロセスは，図4(a)に示すように記憶管理プロセスに記憶資源を要求し，その割り当てを受けた後，プログラムをロードし，runプリミティブを用いてユーザ・プロセスを生成する。その後，スーパバイザ・プロセスは，システム・コールを待ち受けると同時に，図4(b)～(d)に示す方法により，ユーザ・プロセスに対する資源割り当てをスケジュールする。すなわち，(b)に示すように，端末へのアクセス等により，ユーザ・プロセスが長時間停止する場合には，記憶管理プロセスに通知する。記憶管理プロセスは，記憶資源が不足した場合，(c)に示すように，このようなユーザ・プロセス，あるいは，主記憶上に2秒以上存在しているユーザ・プロセスを管理するスーパバイザ・プロセスに対し，スワップ・アウトの要求を行う。要求を受けたスーパバイザ・プロセスは，スワップ・アウトを実行し，領域を解放し，再び領域が割り当てられるのを待つ。

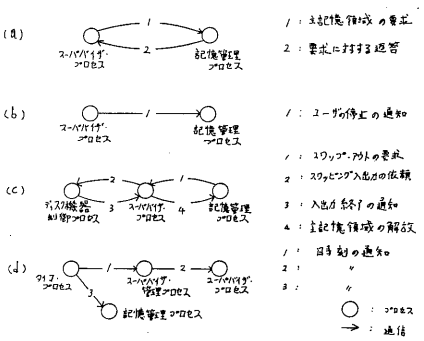

図4　ユーザ管理機能の実現

スーパバイザ・プロセスは，1秒およびシステム・コール毎に，haltプリミティブによりユーザ・プロセスを一時停止し，その間のプロセサ使用量，入出力発行回数等から，プロセサ割り当てのための優先度を計算する。

優先度計算のための時間周期は，タイマ・プロセスが通知する(d)。タイマ・プロセスは，タイマ装置からの割り込みを待ち受けた後，haltプリミティブによりユーザ・プロセスのプロセサ使用量を計測する。また，1秒毎に，記憶管理プロセス，および，スーパバイザ管理プロセスに時刻の通知を行う。スーパバイザ管理プロセスは，さらに，各スーパバイザ・プロセスに通知する。

ファイル・システムは，ファイル，inode，バッファの各プロセス・アレイおよびその管理プロセス，および，二次記憶機器制御プロセス等からなる。

オープンされたファイルの各々を，ファイル・プロセスによって実現する。ファイル・プロセスは，ファイルのオープンに際してファイル管理プロセスが割り当てる。ファイル・プロセスは，内部に現在入出力を行っているファイル部分のディスプレイスメントを保持しており，ファイル中の特定の部分の入出力をinodeプロセスに要求する。

UNIXでは，512バイトのブロックを単位として入出力およびバッファリングが行われており，また，二次記憶上でこのブロックを単位とするマッピングを行うことにより，ファイル実体を実現している。inodeプロセスは，内部にファイル実体ごとのマッピング表を保持しており，入出力すべきディスプレイスメントから二次記憶上のブロック番地を計算する。inodeプロセスは，さらに，2.で述べたバッファリング・システムに対して必要なブロックを要求し，ファイル入出力を実行する。

ディスクinodeプロセスは，二次記憶上の空きinodeを管理しており，ファイル生成（create）に際して，inodeを1つ割り当てる。ディスク・バッファプロセスは，二次記憶上の空きブロックを管理しており，ファイルの拡張に際して，ブロックを割り当てる。

4. 実現

現在，PDP11/34相当の計算機上で実現されたシステムが動作しており，単一ユーザでのプログラム編集，コンパイル等が行えるようになっている。以下において，実現の際の問題点，その解決法等について述べる。

4.1 OS核の実現

OS核は，高速に動作することを第一の目標として設計，実現された。

Callプリミティブを，通信相手プロセス名，エントリ名，通信引数を引数として呼び出すことによりランデブ呼び出しが実現される。プロセス名は，pcb（process control block）の番地，エントリ名は1語中のビット位置によって識別される。accプリミティブを，受け付けるべきエントリ名を引数として呼び出すことにより，ランデブ受け付けが実現される。このとき，callプリミティブは，直ちにランデブが成立する場合にはスケジューラを介さずに直接通信を受け付けるプロセスを実行する。また，accプリミティブは，callプリミティブの引数がつまれたスタックへのポインタを値として返すことにより，きわめて高速に通信引数の結合が行われる。このため，実測の結果，ランデブ成立，および，終了後のスケジューラ等のオーバヘッドを総計して，約450μsecでランデブが実行された。

4.2 記述言語の設計，実現

プロセス内部記述用のC言語には，entおよびcallマクロが追加された。これらのマクロ宣言から，外部名宣言が生成されるとともにcall，accプリミティブの仕様に適合する引数を生成する。

図5は，記憶管理プロセスの内部記

```
#ent    MFREE   mfree ---------- (a)
#ent    MALOC   maloc ---------- (b)
#call   SREQ    sreq ---------- (c)
#call   ANS     ans ---------- (d)

process() ---------- (e)
{
        int     **p;
        int     addr, size;

Loop:
        b = acc(MFREE : MALOC : ---); ---- (f)
        switch(*b) {
        case MALOC :
                addr = *(b + 1);
                size = *(b + 2);
                endr();                    } -- (g)
                enque(addr, size);
                schedule_allocation();
                goto Loop; ---------- (h)
        case MFREE :
-------------------------------------
        case ----------------------------
-------------------------------------
        }
}
```

図5　記憶管理プロセス内部の概要

述の概要である。記憶管理プロセスは，主記憶領域の要求（MALOC），解放（MFREE）等の通信を受け付ける。entマクロ(a)により，外部名mfreeから，プロセス内部で用いられるエントリ名MFREEが定義されている。同様に，(c)においてスワップ・アウト要求を他のプロ

セスに対して行うための呼び出しエントリSREQが，外部名sreqからcallマクロにより宣言されている。要求に対する返答のための呼び出しエントリANSも同様である(d)。

全てのシステム・プロセスは，"process"という名前を持つ手続き(e)を主手続きとしており，システムの起動時にこの手続きに制御が渡る。記憶資源の管理処理は，(f)の通信受け付けにより処理要求を受け付け，要求に対応する処理(g)を，無限ループ(h)により繰り返すことにより実行される。

PNLは，図6に示すように，プロセス定義の部分と関係記述の部分からなっている。同図では，記憶管理プロセスmemと，スーパバイザ・プロセス・アレイsupを定義している。(a)が，memの定義であり，予約語text(b)により内部記述のオブジェクト・モジュールがファイルmem.oにあることを指定している。予約語use(c)により，プロセスが使用するスタックの量を指定する。予約語ent(d)により，プロセス内部のentマクロ中で宣言されている通信受け付けのための外部名を，PNL上で宣言する。同様に，予約語call(d)によりプロセス内部で宣言されている外部名をPNL上で宣言する。(f)では，以上のような仕様を持つプロセスの実体を1個定義し，memと名づけている。(g)では，同様にスーパバイザ・プロセス・アレイの仕様を定義し，(h)において，同一の仕様を持つプロセスの実体を10個定義し，supと名づけている。プロセス・アレイに属するプロセスが参照する通信相手はすべて同一であるので，PNL上では1個のプロセスと同等に扱う。実行時には，プロセス・アレイに属するプロセスは，アレイ名にインデクスを付けて識別する。

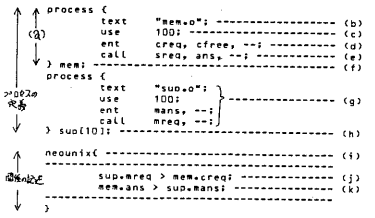

図6　PNL記述の例

(j)，(k)は，参照関係の記述であり，(j)において，スーパバイザ・プロセスの記憶管理プロセスに対する領域要求，(k)において，その返答を記述している。(i)において，これらの関係により成立するシステムを，neounixと名づけている。

PNL処理系は，以上のようなPNLプログラム，および，各プロセス内部を記述した言語のオブジェクト・モジュールを入力として，ロード・モジュールを生成する。

4.3　システムの実現

4.3.1　ユーザ管理機能の実現

ユーザ管理機能を実現するにあたっての問題点は，非決定性動作によるデッドロック発生の危険であった。非決定性の原因は，ユーザ・プロセス間での信号(signal)のやりとり，端末からの割り込み，タイマ装置等に起因するものである。このような非決定性動作を，デッドロックの危険なしに実現するために，資源アクセスによりプロセスが停止する期間をなるべくみじかくなるようにするとともに，常に複数の通信を待つようにした。たとえば，スーパバイザ・プロセスがファイル・プロセスに対してアクセスする場合には，図7に示すように，要求と終了の通知を

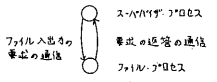

図7　ファイル入出力のための通信

別の通信により行う。ファイル入出力処理が終了する以前にも，時刻の通知，あるいは，スワップ・アウトの要求等

が行われるが，スーパバイザ・プロセスは，入出力終了の通信以外に上記の通信をも待ち合わせることにより直ちに処理できる。同様の方法により，スーパバイザ・プロセスが他のプロセスに対して通信を行う場合にも，長期間停止しないように配慮されている。

また，相互に二つのプロセスがランデブ呼び出しを行うことにより生じるデッドロックを防止するために，呼び出しを行おうとするプロセスが自分を呼び出しているか否かを判定するdeadlockプリミティブを設けた。

4．3．2　ファイル・システムの実現

ファイル・システムを実現するにあたっての問題点は性能であった。ファイル入力では，データは，バッファ→inode→ファイル→スーパバイザ→ユーザの順に伝達される。したがって，たとえば，512バイトのファイル入力に対して，2048バイトのデータ転送が必要となる。この点は，ランデブを入れ子にして用いることにより，改善を試みた。すなわち，inodeプロセスは，バッファ・プロセスとのランデブ期間中にファイル・プロセスを呼び出し，同様に，ファイル・プロセスもランデブ期間中にスーパバイザ・プロセスを呼び出すことにした。これにより，4つのシステム・プロセス間で同期が成立し，スーパバイザ・プロセスはバッファの内容を直接ユーザ・プロセスに転送することが可能になり，データ転送の回数を1回に減ずることが可能になった。

4．4　実現結果

実現の結果，各プロセスは総計6.7kステップ，PNLプログラムは511ステップであった。従来のUNIXシステムは，約10kステップであるが，より多種類の周辺機器に対応しているので，規模の点では大きな増減はない。

5．考察

再設計の結果，第一に，モジュラリティが改善された。UNIXは，手続きの集合によって記述されており，互いに関連の深い手続きが1つのファイルにまとめられ，コンパイル単位となっている。手続き名は，他のいずれの手続きからでも参照できる。また，手続き間は，引数以外に大域変数を用いて結合されている。このような記述法によるシステムは，手続き，変数の定義，アクセス位置の把握がむつかしいために，理解，変更等が行いにくい。ここで，各コンパイル単位が輸入，輸出する外部名の数を尺度として，新旧システムのモジュラリティを比較してみる。従来のシステムのコンパイル単位であるファイルと，提案方式におけるコンパイル単位であるプロセスがそれぞれ輸入，輸出する外部名の数をシステム全体で平均すると，輸入に関して，35個から5個へ，輸出に関して9個から5個へ，それぞれ大きく減ずることができた。

従来システムとの比較は困難であるが，実現に際してのデバッグもきわめて容易に行えた。デバッグは，テスト・プロセスを用いて，機能的に下位のプロセスから順に行った。たとえば，バッファリング・システムのデバッグは，図8に示すように，バッファ・プロセス

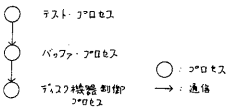

図8　バッファリング・システムのデバッグ

に通信を行い，バッファの内容を得て，それを端末に出力するテスト・プロセスを作成し，バッファ，および，ディスク機器制御プロセスのデバッグを行い，以下，同様の方法で，バッファ管理，inode，inode管理，ファイル・プロセス等の順にプロセスを追加して

スト・プロセスを作成してデバッグを行った。テスト・プロセスは，いずれも50ステップ前後で，きわめて容易に作成することができる。この結果，ファイル・システムのデバッグは，約20人日で完了した。デバッグが容易となった理由として，モジュラリティが高く各プロセスの仕様が単純なこと，各プロセスが制御的に独立していることとともに割り込み等の並行動作に関する処理をOSの核が行うので並行動作を意識する必要が少ないこと，PNLによりデバッグのための環境が容易に作成できること，等があげられる。

現段階では，性能の測定は詳細に行っていないが，表2に，新，旧システム

	UNIX上での所要時間(ms)	新システム上での所要時間(ms)	比(%)
Job1	147	179	73.7
Job2	240	344	67.8
Job3	161	213	75.6
Job4	279	373	74.8

表2　新旧システム上でのコンパイル時間の比較

上でのC言語のコンパイルの所要時間を示す。ここからシステムのオーバヘッドを直接知ることはできないが，所要時間自体は3割弱の増加に止まっており，少なくともコンパイルに関しては実用範囲内にあるものと考えられる。

デッドロックは，現在までに起きていないが，スワッピングが多発する高負荷での検査は行っていない。

実現作業の過程で見い出された問題点として，スタック領域の不足があげられる。現在50個のプロセス実体が存在するため，50個のスタックが必要で，スタックの使用量を厳密に評価することにより領域の節約を図っている。しかしながら，今後のプロセサは，論理空間が広がる傾向にあるので，問題は少ないものと考えられる。

6. おわりに

現在までに，提案方式の実用性に関する見通しを得た。今後の課題は，オーバヘッドの原因を詳細に測定，解析すること，および，その結果必要があれば性能の改善をはかることである。

また，システムの構成要素の制御が互いに独立していることを生かして，分散ハードウェア用OSへの応用をはかることが，今一つの課題である。

参考文献

1) Dijkstra, E.W. : The Structure of the "THE"-Multiprogramming System, CACM, Vol.11, No.5, pp.341-346 (1968).

2) DoD : システム言語Ada 基準文法書，共立出版 (1981).

3) Hansen, P.B. : The Solo operating system, Softw. Pract. Exper., Vol.6, pp.141-205 (1976).

4) Holt, R.C. : Concurrent Euclid The Unix System and Tunis, Addison-Wesley (1983).

5) Ritchie, D.M. and Thompson, k. : The Unix Time-Sharing System, CACM, Vol.17, No.7, pp.365-375 (1974).

6) 高平 徹 : 分散処理オペレーティング・システムの課題，情報処理，Vol.20, No.4, pp.284-288 (1979).

7) 田胡和哉，益田隆司：オペレーティング・システムの論理記述に関する考察，情処学会第23回大会予稿集，pp.281-282 (1981).

8) 田胡和哉，益田隆司：オペレーティング・システムの論理記述に関する考察（第二報），情処学会第24回大会予稿集，pp.179-180 (1981).

9) 田胡和哉，益田隆司：オペレーティング・システムの論理構造記述法，情処学会第25回大会予稿集，pp.219-220 (1982).

10) 田胡和哉，益田隆司：オペレーティング・システムの論理構造記述の設計，情報学会第26回大会予稿集，pp.341-342 (1983)