オペレーティング・システム　45－9
（１９８９．１１．１１）

分散型リアルタイムオペレーティングシステム；ＤＩＲＯＳ

谷口　秀夫　　遠城　秀和　　井村　佳弘　　境　孝之
ＮＴＴデータ通信（株）　開発本部

ＤＩＲＯＳは、バス／ＬＡＮ／ＳＣＳＩの各通信路で結合された分散プロセッサ環境においてのトランザクション処理を可能にする分散型リアルタイムオペレーティングシステムである。ＤＩＲＯＳは、リクエスト制御と呼ばれるプログラム構築方式に基づいて作成されており、非完了システムコール機能やリモートファイルアクセス／リモートデバイスアクセス／マルチプロセス生成／リモートプロセス間通信の各機能を実現している。また、ＵＮＩＸとの協調処理を可能にするため、ＵＮＩＸシステムコール機能とＤＩＲＯＳ－ＵＮＩＸマシン間の分散ファイルシステム機能を実現している。

A Distributed Real-time Operating System ; DIROS

Hideo TANIGUCHI, Hidekazu ENJO, Yoshihiro IMURA, Takayuki SAKAI
Development Headquarters
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION
Kowa Kawasaki Nishi Bldg. 66-2 Horikawa-cho, Saiwai-ku,
Kawasaki-shi, Kanagawa 210, Japan

DIROS is a real-time operating system for transaction processings. It supports a distributed environment where one hardware internal bus connects up to 8 processors to one computer. Processors on different computers are connected by a local area network(LAN) or small computer system interface(SCSI). Because the DIROS program is based on a facility called request control, it realizes the following functions: (1) remote-file access, (2) remote-device access, (3) multi-process creation, and (4) remote inter-process communication. DIROS has two characteristics for coordinating work between DIROS machines and UNIX workstations. One is that application programs can mix unique DIROS system-calls and UNIX systen-calls. The another is that one distributed file system isconstructed between DIROS machines and UNIX workstations.

1．まえがき

ＬＳＩ技術の進歩に伴い、マルチマイクロプロセッサシステムはメインフレームに匹敵するコスト／パフォーマンスを実現している。分散型リアルタイムオペレーティングシステム（DIstributed Real-time Operating System:DIROS)は、複数のプロセッサを搭載した計算機を高速通信路で結合した分散環境におけるトランザクション処理をサポートするＯＳである。

分散環境でのトランザクション処理をサポートするＯＳを実現するには、以下の問題がある。

（１）分散処理ＯＳは、大きく３つに分類できる。１つはＵＮＩＸシステムコールインタフェースを持つ新ＯＳであり、ＬＯＣＵＳ[1]やＭａｃｈ[2]がある。もう１つは、独自のシステムコールインタフェースを持つ新ＯＳであり、ＣＨＯＲＵＳ[3]やＳｐｒｉｔｅ[4]がある。最後は、ＵＮＩＸを拡張したＯＳであり、Ｎｅｗｃａｓｔｌｅ－Ｃｏｎｎｅｃｔｉｏｎ[5]，ＣＯＣＡＮＥＴ[6]，ＮＦＳ[7]，ＤＰＥ[8]，ＲＦＳ[9]，ＭＯＳＩＸ[10]がある。これらのＯＳは、分散ファイルシステムを持ち、リモートプロシジャコールやプロセスマイグレーションの機能を持つものもある。しかし、トランザクション処理を行なうには、次の問題がある。（Ａ）非完了システムコール機能が充分でない、（Ｂ）ファイル操作関連の信頼性が低い。

（２）ＴＲＯＮ[11]のようなリアルタイムＯＳは、リアルタイム性を確保するため、複雑なプログラム構成になっていることが多い。そのため、分散環境をサポートするように改造することは難しい。単に、各ＯＳ間をメッセージ通信で結んだだけでは、分散環境でのトランザクション処理をサポートすることはできない。

本稿では、トランザクション処理をサポートする分散ＯＳへの要求条件を示すとともに、ＤＩＲＯＳにおける解決策について述べる。

2．要求条件

分散環境においてトランザクション処理を実現するＯＳには、非完了システムコール機能、ネットワーク透過性、ヘテロ環境サポート、が求められる。各々について、以下に説明する。

2．1　非完了システムコール機能

非完了システムコール機能とは、３つのタイプのシステムコール（ジョブ受付、ジョブ結果取得、ジョブ取消し）を応用プログラム（以降、ＡＰと略す）に提供する。この機能により、ＡＰは１つのプロセスで多くの処理を並列的に行なうことができ、サービス実現時のＡＰプロセス数を最適化できる。

非完了システムコール機能を実現するには、各ジョブについて、ジョブの受取り／ジョブの実行／ジョブ結果の格納の処理を行なうＯＳプログラムを分離するとともに、ジョブの取消しメカニズムを持つことが必要である。

2．2　ネットワーク透過性

ネットワーク透過性とは、スタンドアロン環境と同様なインタフェースを分散環境でも提供することである。このことは、ＯＳがＡＰに提供するシステムコールとＯＳプログラム間のインタフェース（以降、内部コールと呼ぶ）の両方に求められる。

システムコールのネットワーク透過性としては、ファイル管理とプロセス管理のシステムコールについて２つのことが求められる。１つは、装置やファイルやプロセスの資源が全プロセッサから一意の名前で管理されていることである。もう１つは、資源アクセスのセマンティクスがスタンドアロン環境と分散環境で同じであることである。これらの条件を満足することにより、ＡＰプロセスを任意のプロセッサで走行させることができる。

内部コールのネットワーク透過性としては、プロセス管理やファイル管理やディスク制御などのＯＳ基本処理プログラム間で直接通信でき、そのインタフェースは通信方式に影響されない必要がある。その結果、いろいろな分散環境をサポートするＯＳを構築できる。

２．３　ヘテロ環境サポート

トランザクション処理とＴＳＳ処理やバッチ処理の間での協調処理を可能にするため、ファイルやプログラムを共用できるヘテロ環境のサポートが必要である。

プログラムを共用できるため、ＯＳは独自のシステムコールとともにＴＳＳのシステムコールを実現する必要がある。また、そのインタフェースはマシン語レベルでＴＳＳマシン上のものと同じでなくてはならない。これにより、例えば、ＴＳＳコマンドをトランザクションマシン上でもそのまま利用できる。

ファイルを共用できるため、ヘテロ分散ファイルシステムを構築しなくてはならない。この実現時の主な問題は、システムコール処理の変換とエラー値変換である。

３．ハードウェア環境

分散環境は、いろいろな通信路でプロセッサ間が結合される。ＤＩＲＯＳがサポートするハードウェア環境について、プロセッサ間の結合状態を以下に説明し、様子を図１に示す。

（１）ＤＩＲＯＳマシン内のプロセッサは、１つのバスで結合されている。プロセッサ間の通信は、プロセッサ割込みと共有メモリを介して行なう。共有メモリは、各プロセッサからアクセスできるが、周辺装置とのデータ転送を制御するＤＭＡコントローラからはアクセスできない。

（２）各プロセッサは、他プロセッサからアクセスできないローカルメモリを持つ。

（３）ＤＩＲＯＳマシンは、ＬＡＮやＳＣＳＩで結合されている。

（４）同一のＬＡＮでＤＩＲＯＳマシンとＵＮＩＸマシンを結合している。

４．オペレーティングシステム構造

ＤＩＲＯＳは、非完了システムコール機能とネットワーク透過性を実現するため、そのプログラムは、リクエスト制御[12]の方式に基づいて作成されている。

４．１　リクエスト制御

リクエスト制御は、ＯＳの各機能を実現しているプログラム間の処理のやりとりについての制御方式である。リクエスト制御方式を以下に説明するとともに、処理フローを図２に示す。

（１）システムコールに対応する機能を実現してい

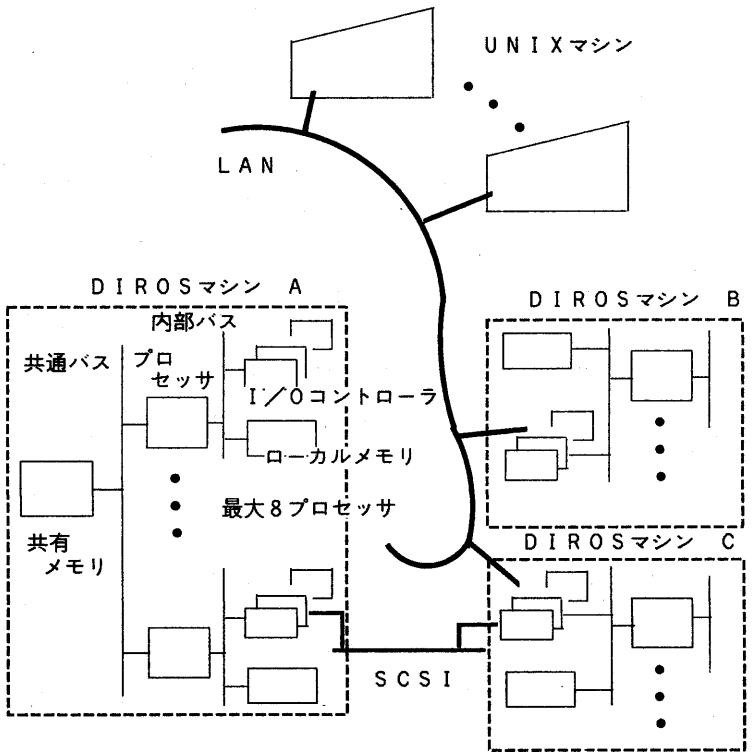

図１　ハードウェア環境

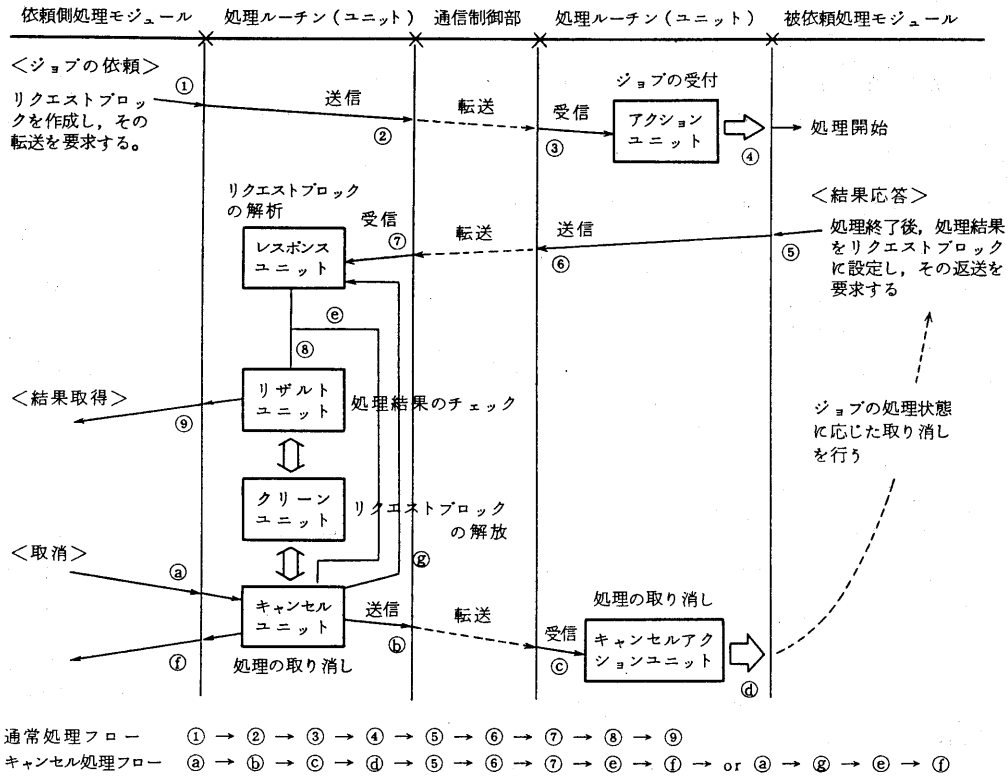

図２　リクエスト制御の処理フロー

る処理単位にプログラムを分割し、処理モジュールと呼ぶ。これには、ディスク制御、ＨＤＬＣ通信制御、ファイル管理、メッセージ制御などがある。

（２）処理モジュールは、分散環境で一意の識別子（以降、処理モジュール識別子と呼ぶ）を持つ。識別子は、処理装置番号、プロセッサ番号、モジュール番号の情報を持つ。様子を図３に示す。

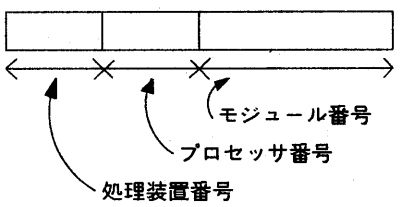

図３　処理モジュール識別子

（３）処理モジュール間は、４つのインタフェース（ジョブの依頼／結果取得／取消しおよび結果応答）を持つ。これらのインタフェースは、６つのルーチン（以降、ユニットと呼ぶ）、アクションユニット／レスポンスユニット／リザルトユニット／クリーンユニット／キャンセルユニット／キャンセルアクションユニット、で制御する。

（４）処理モジュール間は、リクエストブロックと呼ぶ制御ブロックを送受信することで情報をやりとりする。リクエストブロックは、ジョブの依頼先と依頼元の処理モジュール識別子、ジョブの内容とそのデータ、ジョブ結果、を持つ。

（５）処理モジュール間のインタフェースが、共有メモリ／ＬＡＮ／ＳＣＳＩの各通信方式の違いに影響されないように、リクエスト制御内の通信制御部が、この違いを吸収している。

上記の項目（３）が非完了システムコール機能の実現を可能にし、項目（２）と（５）がネットワーク透過性の実現を可能にしている。

４．２　非完了システムコール機能

ＤＩＲＯＳは、リクエスト制御方式に基づいて作成されているため、全てのシステムコールを非完了システムコールとしてＡＰに提供することができる。実際は、ＡＰ利用上で有効な機能を非完了システムコールとして提供している。

ＡＰに提供している非完了システムコールのインタフェースを図４に示すとともに、以下に説明する。

（１）ＡＰからのジョブを受け付けるシステムコールは、各プロセスに一意な値であるリクエスト識別子（ｒｄ：request descriptor）を返却する。

（２）ジョブ結果を受取るシステムコールは、パラメータにｒｄを持ち、次の３タイプがある。｛タイプ１：最初に処理が終了したジョブの結果を返却する｝，｛タイプ２：ｒｄに対応する非完了システムコールのジョブ結果を返却する｝，｛タイプ３：ｒｄに対応する非完了システムコールのジョブが終了していればその結果を返却し、そうでなければ、最初に終了したジョブの結果を返却する｝。いずれの処理も、パラメータｌｉｍｉｔで指定された時間だけ処理の終了を待つ。

（３）ジョブの実行を取消すシステムコールは、パラメータで指定されたｒｄに対応する非完了システムコールのジョブの実行を取消す。

ＡＰからのジョブを受け付けるシステムコールとして１４個ある。９個はファイル管理のシステムコールであり、５個はプロセス間通信やタイマ制御のシステムコールである。ＤＩＲＯＳは、入出力装置や通信回線を外部記憶装置内のファイルと同様にアクセスできるため、９個のファイル管理システムコールで大半のファイル制御処理ができる。ファイル管理機能の詳細については、４．３．１節で述べる。

```
rd = asyn_syscall_job( ... , ... , ... );
   long   rd;      /* 返却値 */
(A) ｼﾞｮﾌﾞを受け付けるｼｽﾃﾑｺｰﾙ

rt = get_first_result(rd, limt);
rt = get_named_result(rd, limit);
rt = get_named_result_first(rd, limit);
   long   rt;      /* 返却値 */
   long   rd;      /* ﾘｸｴｽﾄ識別子 */
   int    limit;   /* 制限時間 */
(B) ｼﾞｮﾌﾞ結果を返却するｼｽﾃﾑｺｰﾙ

rt = cancel(rd);
   long   rt;      /* 返却値 */
   long   rd;      /* ﾘｸｴｽﾄ識別子 */
(C) ｼﾞｮﾌﾞの実行を取消すｼｽﾃﾑｺｰﾙ
```

図４ 非完了システムコールのＡＰインタフェース

４．３　ネットワーク透過性のための機能

ネットワーク透過性のための機能として、リモートファイルアクセス機能、リモートデバイスアクセス機能、マルチプロセス生成機能、リモートプロセス間通信機能、を実現した。これらの機能は、リクエスト制御方式を利用し、システムコールや内部コールをジョブとして転送することで実現している。

４．３．１　ファイル管理機能

ＤＩＲＯＳのファイル管理[13]は、ディレクトリによりファイルを木構造で管理し、ファイルとして通常ファイルと特殊ファイルの２種類を持つ。通常ファイルとは外部記憶装置内にあるデータやプログラムのことであり、特殊ファイルとは入出力装置や通信回線を仮想化したファイルである。

これらのファイルを複数のプロセッサ間で共用できるために、リモートファイルアクセス機能とリモートデバイスアクセス機能がある。アクセスインタフェースは、ローカルファイルにアクセスする場合と同じである。リモートファイルアクセスはリモートの通常ファイル／特殊ファイルとディレクトリに

ついてアクセス可能であるが、リモートデバイスアクセスはリモートの特殊ファイルへのアクセスのみである。しかし、リモートの特殊ファイルへのアクセス速度は、リモートデバイスアクセスが速い。

〈〈リモートファイルアクセス〉〉

異なるプロセッサのファイル管理間でファイル管理システムコールを転送することにより、リモートファイルアクセスが実現できる。各プロセッサは、自プロセッサのファイル管理の処理モジュール識別子情報を持つネットワークディレクトリを有する。

ファイル管理は、システムコールで指定されたファイルパス名を解析し、ネットワークディレクトリを発見するとその内容に基づいて残ったファイルパス名を該当ファイル管理に送る。ファイルパス名の途中にネットワークディレクトリがない場合は、ローカルファイルへのアクセスとして処理を実行する。このような処理フローになっているので、ファイルパス名の中に２つ以上のネットワークディレクトリが存在すると、１つのシステムコールからリモートファイルアクセスが多重に発生する。

〈〈リモートデバイスアクセス〉〉

ファイルアクセスの内部コールをファイル管理と各入出力制御あるいはファイル管理と各通信制御の間で転送することにより、デバイスにアクセスできる。特殊ファイルは、次の情報を持つ。

・周辺装置を制御しているプロセッサを有する計算機番号

・周辺装置を制御しているプロセッサ番号

・周辺装置を制御しているプログラムのモジュール番号

・同一プロセッサ内の周辺装置番号

システムコールで指定された特殊ファイルの情報が他プロセッサの場合、そのアクセスはリモートデバイスアクセスであり、自プロセッサの場合はローカルデバイスアクセスである。このような処理フローになっているので、アクセス処理系はローカル／リモートを全く意識しない。また、１つのシステムコールからリモートデバイスアクセスが多重に発生することはない。。

４．３．２　マルチプロセス生成機能

プロセス生成の内部コールを転送することでリモートにプロセスを生成する。プロセス識別子を以下の情報を持つ形で規定し、ネットワーク透過なプロセス生成環境を実現している。

・プロセスが走行するプロセッサを有する計算機番号

・プロセスが走行するプロセッサ番号

・プロセスを制御するプログラムのモジュール番号

・同一プロセッサ内のプロセス番号

各プロセッサ上のＡＰ負荷分散をスムーズに行なうためには、各プロセッサ上へのプロセス生成を高速にできる必要がある。１プロセス生成をくり返して多数のプロセスを生成すると、過大なディスクアクセスやデータコピーやＣＰＵ負荷が発生する。このため、ＤＩＲＯＳは、１つのプログラムから複数のプロセッサ上に多数のプロセスを一度に生成できるマルチプロセス生成機能[14]を実現した。

以下の点を並列化して、処理を効率的に実現している。

（１）ディスクから共有メモリへのプロセスデータの転送、と共有メモリから各プロセス生成プロセッサのローカルメモリへのプロセスデータの転送、の並列化

（２）各プロセス生成プロセッサでのプロセス生成の並列化

さらに、マルチプロセス生成システムコールの内容通知やプロセス生成終了通知の処理が、全プロセッサを対象としているため、プロセス生成依頼処理が生成プロセスの数や場所に影響されないというメリットがある。このような特徴により、マルチプロセス生成の処理時間は、生成プロセッサ数や生成プロセス数にほとんど影響されず、プログラムサイズで決定される。

４．３．３　リモートプロセス間通信機能

プロセス間通信のシステムコールを転送すること

により、リモートプロセス間の通信を実現した。通信媒体には、メイルボックス、セマフォ、共用メモリ、があり、プロセスへの直接通信もできる。プロセス識別子は４．３．２項で述べたような情報を持ち、通信媒体は以下の情報を持つ。

・媒体を管理しているプロセッサを有する計算機番号

・媒体を管理しているプロセッサ番号

・媒体を制御するプログラムのモジュール番号

・同一プロセッサ内の媒体番号

通信媒体の位置情報に基づいてローカル／リモートを判別することにより、ネットワーク透過性を実現している。２６個のシステムコールが計算機内のリモートプロセス間通信をサポートしており、以下の６つのグループに分けられる。メッセージ／セマフォ／イベント／共用メモリ／バッファプール／プロセス制御。ＳＣＳＩで結ばれた計算機間ではメッセージ送信ができる。

５．ヘテロ環境をサポートする機能

ＵＮＩＸとの協調処理を可能にするため、ＤＩＲＯＳ上にＵＮＩＸシステムコールを実現するとともに、ＤＩＲＯＳマシンとＵＮＩＸマシン間のヘテロ分散ファイルシステムを構築した。

５．１　ＵＮＩＸシミュレート

ＯＳが別のＯＳインタフェースを実現する方式は、大きく以下の３つに分類できる。

［方式１］プロセスとしてシミュレート

［方式２］ＯＳ核内にシミュレート

［方式３］仮想マシン機能を実現し利用

［方式１］は、最も作成が易しい。［方式２］は、作成が［方式１］に比べ難しい。［方式３］は、独立な２つのＯＳ環境を生むことができる。ＤＩＲＯＳは、ＵＮＩＸ環境との協調処理を可能にするため、以下の要求を満足する必要がある。

［要求１］ＡＰプロセスは、同時にＤＩＲＯＳとＵＮＩＸの両システムコールを使える。

［要求２］ＡＰプロセスは、両ＯＳ上のファイルを共有できる。

［要求３］ＤＩＲＯＳマシンと同種のプロセッサを持つＵＮＩＸマシン上で走行するプログラムは、そのままＤＩＲＯＳのＵＮＩＸ環境上で走行する。

［方式１］は［要求３］を満足せず、［方式３］は［要求１］を満足しない。そのため、ＤＩＲＯＳは、［方式２］によりＵＮＩＸシステムコールをシミュレートし、各要求を満足する環境をＡＰに提供した。

ＡＰに提供しているＵＮＩＸ環境は、以下のようになっている。

（１）３つのシステムコール（profile, plock, ptrace）を除く全てのシステムコールを、ＵＮＩＸマシンと同じＡＰバイナリインタフェースで提供する。

（２）ＤＩＲＯＳプロセス管理は、ＤＩＲＯＳプロセスにＵＮＩＸプロセスより高い権限を与える。

（３）ＤＩＲＯＳプロセスは８つのＵＮＩＸシステムコールを利用でき、ＵＮＩＸプロセスは４２個のＤＩＲＯＳシステムコールを利用できる。但し、ＤＩＲＯＳシステムコールを利用できるＵＮＩＸプロセスは、ユーザｉｄが特定値より小さくなくてはならない。

特に、（１）の特徴は、大変有効である。例えば、ＵＮＩＸマシン上のコマンドをＤＩＲＯＳマシン上で走行させてＤＩＲＯＳファイルを操作できる。また、シェルをＤＩＲＯＳマシン上で走行させることで、ユーザフレンドリなインタフェースをＤＩＲＯＳ上に実現できる。

５．２　ヘテロ分散ファイルシステム

ファイルの共有を可能にするため、同一ＬＡＮで結合されたＤＩＲＯＳマシンとＵＮＩＸマシン間に分散ファイルシステム[15]を構築した。これは、ＵＮＩＸマシン間の分散ファイルシステム[8]をベースにしている。

以下にファイルシステムの特徴を示す。

（１）ＤＩＲＯＳは、ＵＮＩＸシミュレート機能を利用してこの機能を実現しているため、全ファイル管理システムコールについてＵＮＩＸマシンからＤＩＲＯＳマシンへのアクセスを可能にしている。こ

れにより、例えば、ＵＮＩＸマシンからＤＩＲＯＳマシン上のファイルをエディタで変更できる。
（２）８つの基本的なＤＩＲＯＳファイル管理システムコールにより、ＵＮＩＸマシン上のファイルにアクセスできる。
（３）リクエスト制御方式や非完了システムコール機能やＵＮＩＸシミュレート機能を利用することにより、非常に効率的なプログラム構造を持つ。
（４）機能の大半をＯＳ核外に実現することにより、複数タイプの分散ファイルシステム構築を容易にしている。

６．あとがき

分散したマルチプロセッサ環境においてトランザクション処理を実現するためには、非完了システムコール機能／ネットワーク透過性／ヘテロ環境サポート、がＯＳに求められる。

ＤＩＲＯＳは、上記の要求を満足し、分散環境でのトランザクション処理を実現した。具体的には、
（１）１４個の非完了システムコールを提供し、
（２）リモートファイルアクセス／リモートデバイスアクセス／マルチプロセス生成／リモートプロセス間通信の機能を実現し、
（３）ＤＩＲＯＳシステムコールとＵＮＩＸシステムコールの同時利用、とＤＩＲＯＳマシンとＵＮＩＸマシン間の分散ファイルシステムの構築、
を行なった。これらの機能は、「リクエスト制御」と呼ぶプログラム間インタフェース方式に基づいて作成された。

ＤＩＲＯＳは、既に研究開発を終え、商用システムに導入されている。

〈参考文献〉

[1] B. walker, et. al., ``The LOCUS Distributed Operating System,'' Proc. of 9th ACM Sym. on Operating Principle, 1983, pp. 49-70.

[2] M. Accetta, et. al., ``Mach:A New Kernel Foundation For UNIX Development,'' Proc. of USENIX Summer Conf., 1986, pp. 93-112.

[3] M. Rozier, et. al., ``CHORUS Distributed Operating System,'' Computing systems, Vol. 1, No. 4, 1983, pp. 305-370.

[4] J. Ousterhout, et. al., ``An Overview of the Sprite Project,'' ;login:, Vol. 12, No. 1, 1987, pp. 13-17.

[5] D. R. Brownbridge, L, F, Marshall and B. Randell, ``The Newcastle Connection or UNIXes of the World Unite!,'' Software-Practice and Experience, Vol. 12, 1982, pp. 1147-1162.

[6] L. A. Rowe and K. P. Birman, ``A Local Network Based on the UNIX Operating System,'' IEEE Trans. Soft. Eng., Vol. SE-8, No. 2, 1982, pp137-146.

[7] D. Walsh, et. al., ``Overview of the Sun Network File System,'' Proc. ofUSENIX Winter Conf., 1985, pp. 117-124.

[8]谷口, 鈴木, 瀬々, ``ﾌｧｲﾙ管理機能のﾈｯﾄﾜｰｸ化による分散処理OSの構成法'', 情処学論, Vol. 27, No. 1, 1986, pp. 56-63.

[9] A. P. Rifkin, et. al., ``RFS Architectural Overview,'' Proc. of USENIX Summer Conf., 1986, pp. 248-259.

[10] A. Barak and R. Wheeler, ``MOSIX:An Integrated Multiprocessor UNIX,'' Proc. of USENIX Winter Conf., 1989, pp. 101-112.

[11] K. Sakamura, ``The TRON Project,'' IEEE Micro, Vol. 7, No. 2, Apr. 1987, pp. 8-14.

[12] 谷口, ``OS機能の分散を可能にするOS構成法'', 信学論, Vol. J72-D-I, No. 3, 1989, pp. 168-174.

[13] 井村, 谷口, 遠城, ``ﾏﾙﾁﾌﾟﾛｾｯｻにおける分散ﾌｧｲﾙ管理方式'', 情処研報, 89-OS-43-6, 1989.

[14] 谷口, 境, ``ﾌﾟﾛｾｽ生成の高速化と並列化に関する検討'', 信学技報, CPSY89-28, 1989.

[15] 谷口, 遠城, 箱守, ``異種OSを結ぶ分散ﾌｧｲﾙｼｽﾃﾑ'', 情処研報, 89-OS-43-7, 1989.

・ＵＮＩＸはＡＴ＆Ｔのベル研究所が開発したＯＳです。