3K−9　ＤＩＲＯＳ上へのＮＦＳ実現の検討

臼渕啓明　箱守　聰　谷口秀夫
ＮＴＴデータ通信（株）　開発本部

１．はじめに

ヘテロ分散環境において、種々の計算機資源を効率的に共有し、高度な分散サービスを提供するためには、相手計算機に応じた分散制御機能を持つ必要がある。つまり、ＯＳは複数種の分散制御方式をサポートしなくてはならない[1]。この場合、分散制御機能をＯＳ核内に実現する方式と、ユーザプロセスとして実現する方式がある。ユーザプロセスとして実現する方式は、ＯＳ核内に実現する方式に比べ、以下の処理が容易であると共に、ＯＳ核を小型化できる。

(1) 複数の分散制御方式を共存させる制御。
(2) 分散制御方式の追加や削除。
(3) 実現時のデバッグ。

筆者らは上記の検討を進めるため、ＮＦＳ* (Network File System) を、ユーザプロセスとして分散型リアルタイムオペレーティングシステム[2](DIROS; Distributed Realtime Operating System)上に実現する方式について検討した。

本稿では、その検討内容について、ＮＦＳのサーバ機能の実現を中心に報告する。

２．ＮＦＳの特徴

ＮＦＳは、クライアントサーバモデルに基づく分散ファイルシステムである。サーバ機能の実現に関連する主な特徴を以下に示す。

(1) 機能の大半がＯＳ核内で実現されている。
(2) サーバの機能は、複数のプロセスにより実現されている。各プロセスは、クライアントのプロセスと１対１に対応する。
(3) クライアントは、サーバ上のファイルをアクセスする場合、ファイルと１対１に対応する識別用のデータを利用する。これをファイルハンドルと呼んでいる。
(4) クライアントとサーバの間は、ステートレスな性質を持つ。具体的には、ＮＦＳの個々の処理は、ＵＤＰ／ＩＰの一回のパケットのやりとりで完結するため、サーバはクライアントとの過去の処理経緯を保持する必要がない。

３．ユーザプロセス化の問題

ＮＦＳをユーザプロセスとして実現する場合、計算機資源に対する操作はシステムコールで行わなければならない。従って、以下の２つの問題点が生じる。

(1) 性能が低下する。
(2) ファイルハンドルは、ＯＳ核内の資源操作用情報であるため、ユーザプロセスは利用できない。

４．ＮＦＳ実現時の問題と対処策

ＤＩＲＯＳ上にＮＦＳ機能を実現する時の問題点と対処策について述べる。

４．１　非完了システムコールの利用

ＤＩＲＯＳは、非完了システムコールを持つ。非完了システムコールを利用すれば、１つのプロセスが同時に複数のＩ／Ｏ要求を発行できる。従って、非完了システムコールの利用により、複数プロセスからなるＮＦＳサーバを単一プロセスで実現できる。

ＮＦＳサーバの単一プロセス化により、プロセス切り換えのオーバヘッドがなくなり、処理の高速化が期待できる。非完了システムコールを利用して単一プロセス化する場合、次の問題点がある。

(問題1) プログラム構造の相違

ＮＦＳ機能を新規に作成するのではなく、既存のプログラムを移植する場合には、プログラム構造の違いが大きな問題になる。

完了システムコールを利用する処理では、１つのシステムコールでＩ／Ｏ要求と結果取得を行う。従って、システムコール発行前後の処理が連続したプログラム構造となる。

これに対し、非完了システムコールを利用する処理では、Ｉ／Ｏ要求と結果取得は別のシステムコールで

A Design of the NFS on DIROS
Hiroaki USUBUCHI, Satoshi HAKOMORI and Hideo TANIGUCHI
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION

* NFSはサン・マイクロシステムズ社が開発しライセンスしている製品です。

行う。このため、システムコール発行によるＩ／Ｏ要求までの処理と、結果取得後の処理が連続しないプログラム構造となる。

上記の問題に対処するための方式を以下に示し、比較を表１に示す。

方式1:完了システムコールのプログラム構造を維持して非完了システムコールに対応する。

方式2:システムコール発行までの処理と、結果取得後の処理を分割して、プログラムを再構成する。

表１　プログラム構造の比較

	方式１	方式２
関数呼び出し時のオーバヘッド	関数の入口や関数呼び出し時に、システムコール発行前の部分と、結果取得後の部分に分岐させる処理が必要であるため大きい。	小さい。
改造における観点	従来のプログラム構造を利用できるため改造量は小さい。	プログラム構造を再構成する必要があるため、改造量は大きい。

高速な処理を実現する観点からは、関数呼び出し時のオーバヘッドが少ない方式２が良い。

(問題2)処理状態の保存

非完了システムコールを利用して単一プロセス化する事により、１つのプロセスが複数の処理を並列に実行する。これにより処理の状態を保存した領域が、他の処理によって変更される可能性がある。従って、必要に応じ、並列実行される処理毎に領域を確保することが求められる。

(問題3) 複数クライアントの権限の代表

個々のサーバプロセスは、自分のアクセス権限をクライアントのプロセスと同一に設定する。これにより、サーバはクライアントのアクセス権限を正しく代表できる。

従って、単一プロセスで構成する場合には、何らかの対処が必要である。

上記の問題に対処するための方式を以下に示す。

方式A:ファイルアクセスの度に、クライアントに合わせてサーバプロセスのアクセス権限を設定する。

方式B:ＯＳの代わりにＮＦＳサーバ自身が、クライアントのアクセス権限のチェックを行う。

方式Ａは、アクセス権限の設定のためにシステムコールを発行するため、オーバヘッドが大きい。方式Ｂは、内部テーブルの参照のみで実現できるのでオーバヘッドは小さい。従って、処理の高速性の観点から方式Ｂがよい。

４．２　ステートレスなサーバの実現

ＤＩＲＯＳのシステムコールは、アクセスするファイルを識別するために、パス名またはファイル識別子を利用する。これらをファイルハンドルとして利用する場合の方式を以下に示し、比較を表２に示す。

方式a:パス名をファイルハンドルとして利用する。

方式b:ファイル識別子をファイルハンドルとして利用する。

表２　ファイルハンドルの実現方式の比較

	方式ａ	方式ｂ
サーバのステートレス性	実現できる。	実現できない。
扱えるファイルの範囲	パス名長は、３２バイトのファイハンドル長以内に制限される。	制限無し。
処理のオーバヘッド	大きい。	小さい。
実現できるＮＦＳ機能	全部。	一部。

表２に示すとおり、パス名またはファイル識別子を利用して、実用的な機能を持ち、かつステートレスなサーバを実現することはできない。

従って、パス名とファイル識別子を併用し、それぞれの長所を活かす方式が良い。以下に、併用する方式を示す。

(1) ファイルのリードやライトなど処理の高速性が要求される場合には、オーバヘッドの少ないファイル識別子を利用する。

(2) アクセス権の設定など処理の高速性が余り要求されない場合には、パス名を利用してサーバのステートレス性を実現する。

５．あとがき

本稿では、ＮＦＳをユーザプロセスとしてＤＩＲＯＳ上に実現する方式について検討した。今後、インプリメントを行い、対処策の有効性を検証する予定である。

参考文献

[1]谷口，箱守，”異なるＯＳ間での分散処理に向けて”，情処シンポジウム論文集90-5，1990

[2]谷口，遠城，井村，境，”分散型リアルタイムオペレーティングシステム ；ＤＩＲＯＳ”，情処研報，89-OS-45-9，1989