3F−7

forkシステムコールの機能拡張による分散実行環境の構築

山田 雄一郎* 中村 嘉志† 多田 好克‡
電気通信大学 大学院情報システム学研究科§

1 はじめに

近年、計算機の低価格化により複数の計算機を同時に使用することが可能になってきた。これらの計算機は、ネットワークに接続することで相互にプログラムを実行させることができる。しかし、現在では、アプリケーションから複数の計算機を有効に利用できていない。

UNIX環境では、プロセス単位でプログラムが実行される。各々のプロセス間には依存関係が少なく、並行に実行される。そして、プロセスは、forkシステムコールによってのみ作成される。そこで、本研究ではforkシステムコールを、セマンティクスを変更せずに、新たな子プロセスを他の計算機上へ作成するように機能の拡張を行なう。これにより、既存のプログラムは変更することなしに、また、新たなプログラムはforkシステムコールを使用することで複数の計算機を使用することができる。

本稿では、このようなforkシステムコールとそれを実行するための環境の設計について述べる。

2 実行環境の設計方針

forkシステムコールは、呼び出し元の複製を子プロセスとして作成する。子プロセスを単純にネットワーク上の他のホストに作成するだけでは、拡張前とセマンティクスに違いが生じる。このため、forkシステムコールのセマンティクスを以下のようにして保存する。

プロセスIDや親プロセスのIDは、ユーザーが直接変更できない。また、ファイル記述子は、ユーザーが変更できても状態を復元するのが難しい。これらについては、従来のforkシステムコールを使用し、子プロセスを作成することで、親プロセスが持っている状態を保存する。

次に、シグナルハンドラ、シグナルスタック、プライオリティー、インターバルタイマは、システムコールによってユーザーレベルで状態の収得、再設定が可能である。これらは、子プロセスが再設定を行うことで対応する。

子プロセスが本来のホストで行わなければならないシステムコールや、ファイルへのアクセスに関しては、ホスト毎に代理で要求を受け付けるプロセスを作成する。子プロセスは、このプロセスへの要求を行うことで対応を行う。

共有メモリや、mmapシステムコールによる仮想メモリ空間へマップされたファイルを異なったホスト間で共有するのは現実的ではない。このため、mmapシステムコールでマップされた領域が共有されない場合にのみ、その領域を複製して対応する。

3 システムの概要

本研究で設計、実装しているシステムは以下のプロセスから構成される。

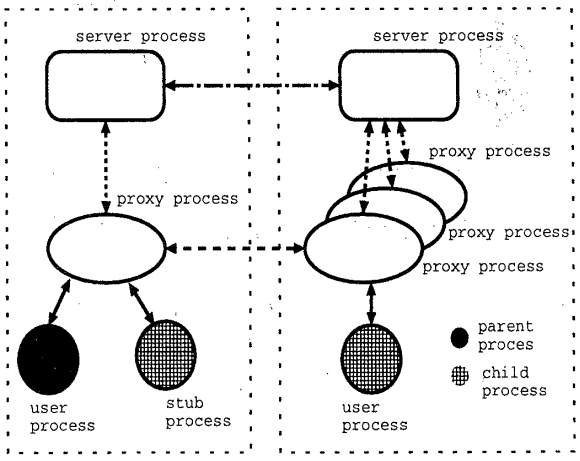

図1:本システムの概観

Constructing Distributed Execution Environment by Extending Fork System Call.
*Yuichiro Yamada
†Yoshiyuki Nakamura
‡Yoshikatsu Tada
§Graduate School of Information Systems, The University of Electro-Communications.

- server プロセス
  このプロセスは、各ホストに1つ動作するデーモンプロセスである。他ホストの server プロセスからの要求で proxy プロセスを起動する。
- proxy プロセス
  user プロセスからのシステムコール要求を受け、それが実行されるべきホストおよび stub プロセスへと中継する。
- user プロセス
  本システム内でのユーザープロセス。本システムが提供しているライブラリをリンクすることで、特定のシステムコールを上書きし、proxy プロセスへの要求を行う。動的リンクの場合には、既存のプログラムをそのまま利用できる。
- stub プロセス
  ユーザーレベルで直接変更できないプロセスのコンテクスト (プロセス ID, 親子関係、ファイル記述子) を保存するプロセス。また、proxy プロセスからのリクエストで、システムコールの発行も行う。

4 子プロセスの作成

本研究では、子プロセスのコンテクストを得るために procfs [1] を使用する。procfs はファイルシステムとして扱うことができ、目的とするプロセスの仮想メモリイメージ、レジスタセット、仮想メモリのマップ状態の収得、設定が行える。

子プロセスの作成方法を、図 2 を用いて説明する。

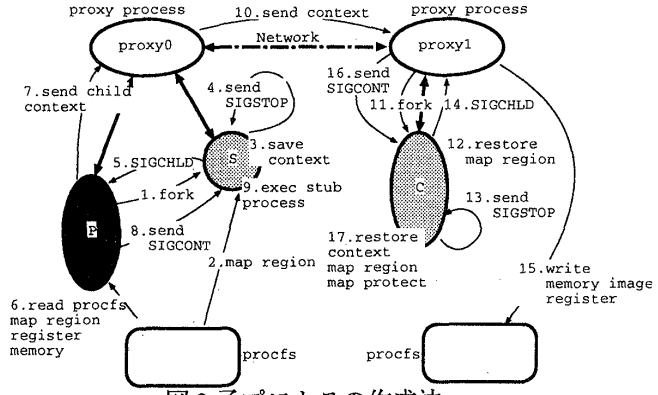

図 2:子プロセスの作成法

- 親プロセス P が fork() を行うと、まず通常の fork で子プロセス S が作成される。(図 2(1))
- 子プロセス S は、再設定すべきコンテクスト (シグナルハンドラ等) と、自分の仮想メモリのマップ状況を保存し、SIGSTOP で停止する。(図 2(2,3,4))
- 親プロセス P は、SIGCHLD を受け取ると、procfs から子プロセス S の仮想メモリのイメージ、マップ情報、レジスタ情報を得て proxy0 へ送信する。その後、子プロセス S に SIGCONT を送り再開させる。(図 2(5,6,7,8))
- 再開された子プロセス S は、プロセス ID、親子関係の保存を行う為に exec して stub プロセスになる。(図 2(9))
- proxy0 は、親プロセス P から受け取ったデータを、proxy1 へ送信する。(図 2(10))
- 送信を受けた proxy1 は、子プロセス C を作成する。子プロセス C は、仮想メモリのマップの復元をしたあと、SIGSTOP で停止する。(図 2(11,12,13))
- SIGCHLD を受け取った proxy1 は、procfs を用いてイメージの復元を行う。その後、子プロセス C に SIGCONT を送り、実行を再開させる。(図 2(14,15,16))
- 実行を再開した子プロセス C は、保存していたプロセスのコンテクストを復元し、fork システムコールから復帰する。(図 2(17))

5 システムコールの発行

本システムでは、システムコール関数をライブラリ形式で提供する。このライブラリは proxy プロセスに要求を送信する。この要求は、proxy プロセスを経由して、このシステムコールが実行されるべき stub プロセスへと転送される。stub プロセスが本来のシステムコールを実行した結果は、再度 proxy プロセスを経由してシステムコールの発行元へと返送される。

6 まとめ

fork システムコールを拡張して、セマンティクスを損なうことなく子プロセスを他のホストで実行するシステムの設計を行った。現在は、FreeBSD 上でシステムの実装を行っている。このシステムを使用することで、既存の fork システムコールを使用するプログラムは変更を加えることなく、新たなプログラムは、伝統的な fork システムコールを使用することで、ネットワーク上のホストを意識することなく利用可能となる。

参考文献

[1] MuKusick, M.K., K.Bostic, M.J. Karels and J.S. Quarterman, The Design and Implementation of the 4.4BSD Operating System, Addison-Wesly, 1996.