マルチメディア通信と分散処理56－12
（１９９２．　７．　10）

分散OS ToMにおけるマルチスレッド・プログラムのデバッグ環境

堀切 和典
富士ゼロックス システム技術研究所

松田 久夫
日本ビクター マルチメディア研究所

桜川 貴司
京都大学

ToM(Threads on Modules) は、分散環境に適した新しいプログラミング・モデルを提供する OS である。ToM では、従来の OS では切り離すことのできなかったコンテキスト (スレッド) とプログラム (モジュール) が分離しており、これらを自由に結合することができる。このため、ToM におけるアプリケーション・プログラムはマルチスレッド化された複数のサーバによって構成される並行プログラムとなる場合が一般的である。本稿では、このようなプログラミング・モデルに適したデバッグ環境を実現するために用意した ToM システムコールとこれを用いたデバッガの実現方法について述べる。

A Debugging Environment for Multithread-programs on Distributed Operating System ToM

Kazunori HORIKIRI
FUJI XEROX CO., LTD.

Hisao MATSUDA
VICTOR COMPANY OF JAPAN, LTD.

Takashi SAKURAGAWA
Kyoto University

A debugging environment on distributed operating system ToM is presented in this paper. ToM supports a programming model which is suited for distributed environments. For this purpose, threads and modules are introduced. Threads represent contexts of execution and modules represent program texts. A thread executing a module can change its binding to another module through RPC. In general, an application program on ToM contains multiple modules and multiple threads interacting through this RPC mechanism. System calls of ToM which support debugging environments and an implementation of ToM debugger are also presented in this paper.

1 はじめに

マルチプロセッサ・ワークステーションや高速ネットワークの普及により、ネットワーク上に分散する資源を有効に利用する分散OSの研究が盛んになりつつある。これは、Unixなどの従来のOSが、単一の計算機に集中して存在する資源を管理するように設計されており、広がりつつある分散環境にうまく対応できなくなってきたためである。

ToM(Threads on Modules)は、分散環境に適した新しいプログラミング・モデルを提供するOSである[1]。ToMでは、従来のOSでは切り離すことのできなかったスレッド(コンテキスト)とモジュール(プログラム)が分離しており、これらを自由に結合することができる。このため、ToMにおけるアプリケーション・プログラムはマルチスレッド化された複数のサーバによって構成される並行プログラムとなる場合が一般的である。このような新しいプログラミング・モデルによるアプリケーション・プログラムでは従来のプログラミング・モデルではあまり問題にならなかったレース・コンディション等によるバグが多く発生するため、デバッガの役割が一層重要なものとなる。

また、一般に分散OSでは小さなカーネルの上にアプリケーション・プログラムとしてファイル・サーバやネットワーク透過性等を提供するサーバ群を用意する構成が用いられるが、これらのシステム・サーバを開発するためには小さなカーネル上で動作するデバッグ環境が不可欠なものとなる。

本稿では、このようなプログラミング・モデルに適したデバッグ環境を実現するために用意したToMシステムコールとこれを用いたデバッガの実現方法について述べる。

2 ToMのプログラミング・モデル

ToMでは、コンテキストとプログラムが、スレッド(Thread)とモジュール(Module)として完全に分離しており、スレッドは、RPC (Remote Procedure Call)により、あるモジュールから別のモジュールへと結合を切替えることができる(図1)。モジュールへのRPCの入口はエントリとよばれ、エントリの集合(エントリ・グループ)を指定してRPCを行なうための権限(ケイパビリティ)が作られる。ケイパビリティを保持する主体の一つにハウス(house)がある。ハウスは、Unixのプロセスに相当するオブジェクトで、モジュールとスレッドの集合からなる。スレッドは、スレッド自身か、自分が実行しているモジュールのハウスが持つケイパビリティのエントリにRPCすることができる。

[1] ToMの研究・開発は、慶應義塾大学, 京都大学, 京都高度技術研究所, オムロン, 日本鋼管, ビクター・データ・システムズ, リコー, 富士ゼロックスが共同してすすめている。

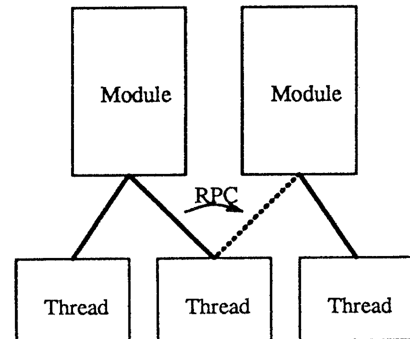

図 1: ToMのプログラミング・モデル

3 シングルスレッド・プログラムを対象とするデバッグ環境

ここでは、シングルスレッド・プログラムを対象とするデバッグ環境に必要なオペレーティング・システムの機能について述べるとともに、代表的な例としてUnixにおけるデバッグ用のシステムコールについて述べる。

3.1 オペレーティングシステムに必要とされる機能

シングルスレッド・プログラムのデバッグ環境に必要な機能として以下のものがあげられる。

- デバッグ対象となるプログラム(デバッギ)のメモリ空間やレジスタに対するアクセス
- デバッギに発生した例外等のインターセプトと例外処理
- デバッギの停止, 継続, ステップ実行, 終了

Unix では例外はシグナルとして抽象化されており、ptrace および wait システムコールを使ってデバッギの制御等を行なっている。

3.2 シグナル

シグナルがデバッグ対象となるプロセス (デバッギ) に送られるとデバッギは停止して、デバッガが ptrace を用いてデバッギを操作できるようになる。デバッギに送られたシグナルはデバッガがそれを受け取るように指定しないと無視され、デバッガが別のシグナルに置き換えることも可能である。

ブレークポイントやキー入力による割り込みはシグナルによって実現されており、ブレークポイントやトレースはSIGTRAP を、割り込み文字の入力は SIGINT を発生する。

3.3 ptrace

ptrace システムコールはデバッガがデバッギを制御したり、状態を検査, 変更するのに必要な機能を提供する。ptrace が提供している機能には、

- デバッギのメモリ空間やレジスタに対するアクセス
- デバッギにポストされたシグナルのインターセプトと置き換え
- デバッギの停止, 継続, ステップ実行, 終了
- カーネルの持つデバッギに関するデータへのアクセス

がある。

3.4 wait

wait システムコールは通常、子プロセスの exit を待って exit コードを得るのに使われるが、被トレース状態にある子プロセスがシグナルを受けとり停止した場合にも待ち状態から復帰する。

これにより、デバッガはデバッギの

- シグナルによる停止
- exit

の検出を行なうことができる。

3.5 デバッグ・セッション

ここでは、Unix においてデバッガがデバッグ・セッションを進めるさいに行なわれるデバッギとのインタラクションについて述べる。

デバッガはデバッギを子プロセスとしてフォークし、子プロセスは TRACEME という引数で ptrace を呼ぶことにより親プロセスにトレースされる状態になる[2]。そのあとデバッギは execve によりデバッグ対象のプログラムを実行するが、トレース状態にあるプロセスによる execve は SIGTRAP を発生させるので、親プロセスはこれを wait で捕捉し、ブレークポイント等を挿入しデバッギを継続させる。デバッガは再び wait を用いてデバッギに対してブレークポイントやステップ実行等によるシグナルが発生するのを待つ。その後デバッガはデバッギに対して ptrace を用いて検査, 変更を行なったのち、デバッギを継続させ wait を用いてデバッギにシグナルが発生するのを待つ。

4 ToM デバッギング・モデル

前述したように、ToM ではモジュールとスレッドの自由な結合を許している。このようなプログラミング・モデルではデバッグの対象は、一般に図 2のようなモジュールとスレッドからなる集合であると考えられる。

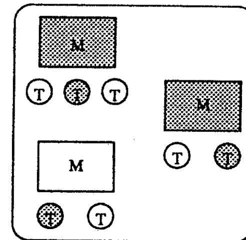

図 2: 一般的な ToM のデバッギング・モデル

すなわち、あるスレッドがあるモジュールを実行している場合、

- スレッドまたはモジュールがデバッグ対象の場合、スレッドとモジュールのペアがデバッグの

[2] OS によっては ptrace に、子プロセス以外のプロセスにアタッチする機能を追加しているものもある。

対象となる。

- スレッドがデバッグ対象であり、かつモジュールがデバッグ対象の場合、このペアがデバッグの対象となる。

というモデルが考えられる。

実際に、ToM においてデバッグ対象の指定に用いられるシステムコールはモジュールあるいはスレッドを指定できるような仕様に設計されており、どちらかがデバッグ対象の場合にスレッドとモジュールのペアがデバッグの対象になる。しかし、現在実現されているデバッガにおいては、プログラマがデバッグを行なう場合、対象は主にモジュールであるという仮定の元に、図3のように、デバッグの対象としてモジュールのみを指定し、指定されたモジュールを実行しているスレッドは全てデバッグの対象となるモデルを用いている。

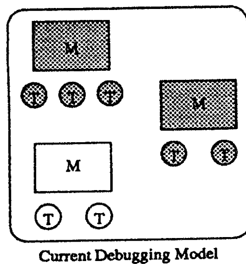

図 3: 現在用いられているデバッギング・モデル

デバッグ対象であるモジュール, スレッドに対して以下のような操作が可能になる。

- ブレークポイントの設定と停止
- トレース
- シングルステップ実行
- 状態の検査, 変更
- イベントのインターセプト

4.1 イベント

ToM では、Unix のシグナルに相当するものとしてイベントを用意している (図 4)。ToM のスレッドにハードウェア的にあるいはソフトウェア的に、ある条件が発生すると、条件の発生したスレッドが停止し、新しいスレッド (イベント・スレッド) が作られ、予め設定されたイベント・ハンドラに対して RPC が行なわれる。イベント・スレッドがハンドラからリモート・リターンすると例外を起こしたスレッドの実行が再開される。

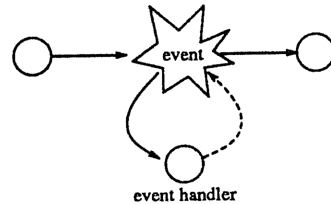

図 4: ToM におけるイベント

4.2 メモリとレジスタに対するアクセス

メモリに対するアクセスは、デバッギのモジュールまたはスレッドにマップされている仮想ページをデバッガの空間にマップすることにより行なわれる。レジスタに対するアクセスは、スレッド ID を指定してスレッドの状態をアクセスするシステムコール (thread_status, thread_set_status) によって行なわれる。

4.3 イベントのインターセプト

デバッグ対象となるスレッドにイベントが発生すると、本来のイベント・ハンドラに RPC が行なわれず、デバッガ・イベントのハンドラ (デバッガが用意するハンドラ) へ RPC が行なわれる。デバッガ側では発生したイベントに対して別のハンドラを起動したりイベントを無視したりすることができる (図 5)。

5 マルチスレッド・プログラムを対象とするデバッグ環境

ToM のシステムコールの設計において、マルチスレッド・プログラムのデバッグに必要な機能として考慮したのは以下のものである。

- 並行して発生したイベント (コンカレント・イベント) のインターセプト

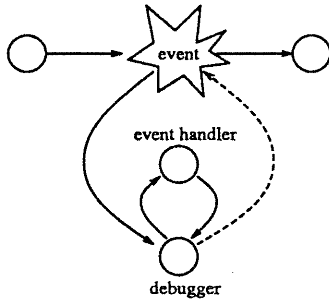

図 5: イベントのインターセプト

- モジュールおよびスレッドのリスト取得と状態検査
- モジュールおよびスレッドの生成, 消滅のインターセプト
- RPC によるモジュールとスレッドの結合の変更のインターセプト
- 複数のスレッドによるレースコンディションの記録, 再現

5.1 コンカレント・イベントのインターセプト

マルチスレッド・プログラムにおいてはシングルスレッド・プログラムではあまり問題にならなかったコンカレント・イベントの問題が発生する。これは、あるスレッドにイベントが発生してその処理を行なっている最中に別のイベントが発生するもので、複数のスレッドがブレークポイントに達した場合などがこれに相当する。Unix のデバッギング・モデルではシグナルの機構によりシリアライズが行なわれ、1 つのイベントのみが報告されるので、デバッガはデバッギの正しい状態を知ることができない。

ToM では、並行して発生した複数のイベントに対して、それぞれイベント・スレッドを生成し、さらにそのイベント・スレッドをインターセプトする機構を採用することによりコンカレント・イベントを処理することを可能としている。イベントのキューイング等が必要な場合は、イベント・スレッドの RPC 先として、モニタ等を用いたキューイング・ルーチンを指定する。

5.2 モジュールおよびスレッドのリスト取得と状態検査

ToM では house_module_list(),house_thread_list() によって任意のハウスに属するモジュールおよびスレッドのリストを得ることができる。また、スレッド ID を指定して状態を検査したり、停止, 継続を行なうことができる。

5.3 モジュールおよびスレッドの生成, 消滅のインターセプト

現在の ToM のシステムコールにはモジュールおよびスレッドの生成, 消滅を直接インターセプトする機構はない。しかし、スレッドが実行中にモジュールが消滅した場合はイベントが発生するのでこれを捕捉することができる。また、スレッドの生成に関しては、ToM のスレッドは RPC によりモジュールと結合を行ない実行を開始するため、後述の RPC のインターセプトを用いてこれを捕捉することができる。

5.4 RPC のインターセプト

RPC によって、デバッグ対象外のスレッドがデバッグ対象のモジュールと結合したり、デバッグ対象のスレッドがモジュールとの結合を解除して別のモジュールと結合する場合にこれをイベントとしてインターセプトすることができる。

5.5 複数のスレッドによるレースコンディションの記録, 再現

デバッガの設計にあたりレースコンディションとして考慮したのは以下の点である。

- モニタへの突入順序
- 共有変数のアクセス順序
- タイムアウトの発生

モニタへの突入は RPC を通じて行なわれるので、前述のように RPC をインターセプトすることにより順序の記録, 再現が可能である。また、共有変数へのアクセス順序に関しては、アクセス順序を保存したい変数を指定し、対応する仮想ページのパーミッショ

ンを制限することによりメモリ保護属性違反を発生させ、このイベントをインターセプトすることが可能である。タイムアウトに関しては以下に述べる。

5.5.1 タイムアウト再現性

ToM では Unix のインターバル・タイマに相当するものとしてタイマを用意している。これは指定された時間が経過すると新たにスレッドが作られ、予め指定されたエントリ・ポイントに対して RPC が行なわれるものである。

ところが、デバッガによりブレーク・ポイントが設定されると、通常の実行では発生しなかったタイマのエクスパイアが起こる場合がある。(レース・コンディションの一種) これにより、デバッグ中とそうでない時で計算結果が異なる状況が起こり得る。

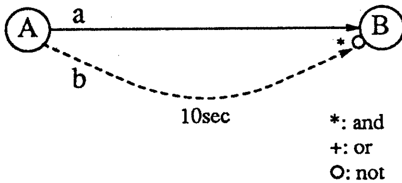

図 6: 時間制限が定められた計算

例えば、図6のように、実線で示されるスレッドaが10秒以内に状態Bに遷移しなければタイムアウトが起こり、計算が中断される場合を考える。

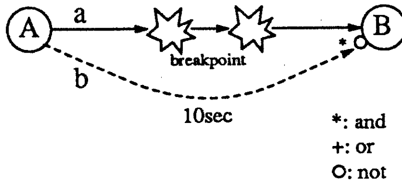

図 7: ブレークポイントによるタイムアウト

この時、図7のようにスレッドaに対してブレークポイントがセットされたとすると、計算が中断されて10秒以内に終了せず、タイムアウトが発生する状況が起こり得る。図7のような例ではタイマのエクスパイアをスレッドaがデバッグにより停止していた時間だけ遅らせることによって、適切でないタイムアウトが発生しないようにできる。

次に、図8のように、スレッドaとスレッドbの両方がが10秒以内に状態Bにならなければタイムアウトが起きるような場合を考える。

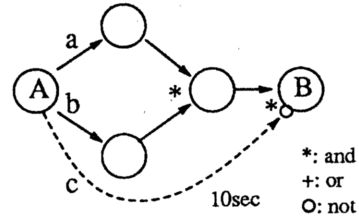

図 8: 複数の処理に対して時間制限が定められた場合

この場合スレッドaに対してデバッグによる時間遅れが発生したとしても、一般にタイマのエクスパイアを遅らせるべき時間を決定することはできない。

ToM デバッガでは、デバッギがブレーク・ポイントをヒットした場合はタイマのエクスパイアを遅らせるとともに、デバッグ対象の全てのスレッドを停止させる機構を用意する。(タイマはエクスパイアするまでに一定の時間が掛かる仮想的なスレッドであると考える) これによりタイマにより引き起こされる非再現性を避けることができる。

6 デバッガの構成

ToM はいわゆるデカーネライズド・カーネルであり、ファイル・システム等はカーネル外のサーバとして構成される。これらのシステム・サーバのデバッグを行なう必要から、ToM のデバッガはカーネルの提供するサービスのみで動作する部分とそうでない部分が切り離された構成となっている。デバッガのうち、ファイル操作などの高機能のシステム・サービスを利用する部分は Unix 等の OS 上に置きリモート・デバッグを行なうことが可能な構成となっている。

デバッガは図9に示すように、以下の4つの部分から構成される。

- デバッグ・サーバ (dbs)
  ToM 上で動作するデバッグ用のサーバで、デバッグ対象のモジュールに対して同数のデバッグ・サーバを用意する。

  このサーバはソケットを介したローカル・デバッガからのリクエストに応じて、デバッギの

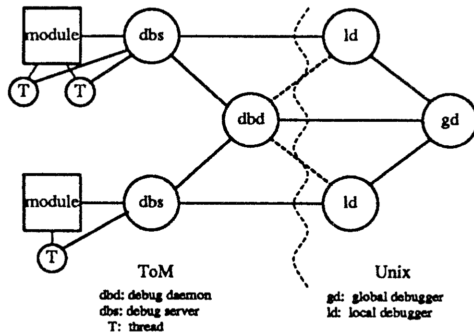

図 9: デバッガの構成

制御, イベントのインターセプトを行なう。

- デバッグ・デーモン (dbd)
  ToM 上で動作するデーモンで、ローカル・デバッガ, グローバル・デバッガからの要求に応じて、デバッグ・サーバを制御する。デバッグ・デーモンは各ホストに 1 つ用意され、新しいモジュールのデバッグを開始する場合にはデバッグ・デーモンがデバッグ・サーバを生成し、サーバが持つポートをデバッガに通知する。

- ローカル・デバッガ (ld)
  Unix 上で動作する ToM 用のデバッガで、デバッグ対象のモジュールに対して同数のデバッガを用意する。このデバッガはユーザとのインタラクションやオブジェクト・ファイルの参照を行ない、必要に応じて ToM 上のデバッグ・サーバに対してソケットを介しリクエストを発行する。GNU の C ソースレベル・デバッガ GDB を改造したものである。

- グローバル・デバッガ (gd)
  Unix 上で動作する。ローカル・デバッガの制御やモジュール間にまたがる大域的な状態を制御するため、ローカル・デバッガ, デバッグ・サーバ, デバッグ・デーモンに対して要求を出す。現在、グローバル・デバッガは GNU Emacs 上で試験的な実装のみが行なわれており、大域的な状態を制御する機能はない。

7 おわりに

本論文では現在開発中の分散 OS ToM におけるマルチスレッド・プログラムのデバッグ環境について説明した。評価・考察や詳しい各論について触れることができなかったが、本論文を拡大した別の論文で取り扱いたいと思う。最後に、本論文を書くにあたって、ToM デバッガのアイデアをまとめるために熱心に議論してくださった、分散 OS 研究・開発プロジェクトのみなさんに感謝致します。また、プロジェクトを強く押し進めていただいている慶應義塾大学環境情報学部の萩野達也助教授に深く感謝致します。

参考文献

[1] Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A., and Young, M.: Mach: A New Kernal Foundation For UNIX Development, *Proceedings of USENIX 1986 Summer Conference*, pp. 93–112 (1986).

[2] Chandy, K. M. and Lamport, L.: Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Trans. on Computer Systems*, **3** No. 1, pp. 63–75 (1985).

[3] Cooper, R.: Pilgrim: A Debugger for Distributed Systems, *Proc. of 7th Int. Conf. on Distributed Computing Systems* (1987).

[4] LeBlanc, T. J. and Mellor-Crummey, J. M.: Debugging Parallel Programs with Instant Replay, *IEEE Trans. on Compt.*, **C-36** No. 4, pp. 417–480 (1987).

[5] Leffler, S. J., McKusick, M. K., Karels, M. J., and Quarterman, J. S.: *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley (1988).

[6] 真鍋義文, 今瀬 真 (NTT ソフトウェア研究所): 分散プログラムのデバッグにおける大域的条件について, IEIC テクニカルレポート COMP 89, ソフトウェア基礎論 **33-11**, プログラミング言語 **23-11** (1989).

[7] 新井潤, 桜川貴司, 立木秀樹, 萩野達也, 服部隆志, 森島晃年: 分散OS ToM – 新しいプログラミング・モデルとセキュリティ機構について –, 情報処理学会オペレーティング・システム研究会, 45-2 (1989).

[8] 萩野達也: 分散OS ToM の仮想記憶管理機構について, 京都大学大型計算機センター研究開発部, 研究発表会報告集 第5号 (1990).

[9] 岡本利夫, 桜川貴司, 堀切和典, 山岸久夫: 分散OS ToM におけるネットワークの概要, 日本ソフトウェア科学会ソフトウェア研究会 (関西), SW-90-7-3 (1990).

[10] 堀切和典, 松田久夫, 桜川貴司: 分散OS ToM におけるデバッガの概要, 日本ソフトウェア科学会第8回大会論文集, pp. 229–232 (1991).