シミュレータを用いたフォールトインジェクション環境

追川 修一
筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻

1 はじめに

オペレーティングシステム（OS）はユーザにサービスを提供するアプリケーションプログラムの実行環境を提供する．そのため，プログラムの実行に必要な計算資源をプロセスに割り当てる．アプリケーションプログラムにバグ（故障）があり，そのプロセスの実行に障害が起きても，他の独立に実行中のプロセスに影響は与えない．しかしながら，OS にバグがあり障害が発生すると，全てのプロセスの実行に影響を与える．従って，OS には高い信頼性が求められる．高い信頼性を実現するには，正常時の動作だけでなく，故障による例外発生時の動作も検証する必要がある．

本論文では，OS の実行が可能な機能レベルプロセッサシミュレータを用い，ハードウェアの故障やソフトウェアのバグをシミュレートすることで，OS のエラー処理が正しく行なわれるかどうかテストを行なうことを可能にするフォールトインジェクション環境について述べる．

2 フォールトインジェクション環境の設計

2.1 機能要求

本論文で述べるフォールトインジェクション環境を開発する目的は，ハードウェアの故障やソフトウェアのバグをシミュレートすることで，OS のエラー処理が正しく行なわれるかどうか検証することだけでなく，正しく行なわれなかった場合，その原因を究明し，正しく行なわれるよう OS を修正することにある．それには，まずハードウェアの故障やソフトウェアのバグをシミュレートできること，そして原因究明が容易であることが求められる．

フォールトインジェクションを行なうためには，ハードウェアの故障やソフトウェアのバグをシミュレートし，故障による例外発生時のコードを実行させる必要がある．また，アクセスされないメモリ上のデータやデバイスに故障を導入しても，それらは障害に結びつかないため，意味のないフォールトインジェクションとなる．そのため，障害に結びつきやすい故障を定義し，導入できる機能が要求される．

次に，フォールトインジェクションにより発生した障害の原因究明を容易にする機能が求められる．エラー処理が正しく行なわれない原因がたとえ単純なコーディングミスにあったとしても，それが障害を引き起こすかどうかは，さらに他の要因に依存することが多い．そのような障害は，非決定的に発生するため，同じフォールトインジェクションを行なっても，いつも同じように障害を起こすとは限らず，原因究明を困難にする．従って，単にフォールトインジェクションを行なうだけでは，エラー処理が正しく行なわれない場合があるということがわかるだけであり，必ずしも OS の信頼性を向上させることにはならない．そのため，フォールトインジェクションと発生した障害の因果関係を調査することができると，原因究明が容易になる．それには，チェックポインティングおよびリプレイ機能 [1] が要求される．

2.2 実行環境

上記の機能要求満たす実行環境としては，仮想マシンモニタ（VMM）またはプロセッサシミュレータが考えられる．VMM を用いた場合，非特権命令は実機上で直接実行可能であるため，実行環境としてはより高速になると考えられる．一方，リプレイに必要な情報を入手できるプロセッサは限定され，またフォールトインジェクションにより発生した障害の原因究明を容易にするための環境や機能をその VMM に実現するためのコストは非常に大きくなる．プロセッサシミュレータを用いた場合は，逆に実行速度は遅くなるが，リプレイに必要な情報の取得は容易になり，また Linux 等のホスト OS 上の環境が利用可能であるため，環境構築は容易になる．

そこで本研究では機能レベルプロセッサシミュレータを用いる．機能レベルプロセッサシミュレータとは，1

Fault Injection Environment using a Processor Simulator.
Shuichi Oikawa, Department of Computer Science,
University of Tsukuba

サイクルで1命令が処理されるという簡素化された実行モデルを採用するシミュレータである．プロセッサのパイプラインやキャッシュなどはシミュレートしないが，簡素化できるためより高速なシミュレーションが可能であるため，OSをシミュレーション実行し，フォールトインジェクションの影響を調査する目的には適している．

2.3 フォールトインジェクション機能

フォールトインジェクション機能は，ハードウェア故障とソフトウェアバグの両方の故障を導入する．FINE [4] は，ハードウェア故障として，メモリ故障，CPU故障，バス故障を定義している．Rio File Cache [3] は，ソフトウェアバグとして，初期化の間違い，ポインタの破壊，不正なメモリ解放，間違ったブロックコピーによるデータ破壊，間違った比較，不正な同期命令を定義している．FINEはこの他に代入の間違いも定義している．本研究では，これらのハードウェア故障とソフトウェアバグの両方の故障を導入可能にするほか，ハードウェア故障としてデバイス故障も導入する．

3 フォールトインジェクション環境の実装

フォールトインジェクション環境に用いるため，SH4機能レベルプロセッサシミュレータ sh4sim を開発している．現在，sh4sim は SH4 (SH7780) をシミュレートし，Linuxを実行可能である．外部入出力デバイスとして，コンソールとして使用するシリアルインタフェースとイーサネットデバイスをサポートしている．フォールトインジェクション機能は，現在実装中である．

4 関連研究

オペレーティングシステムに対するフォールトインジェクション機能は，実機上で動作させる場合，オペレーティングシステム自身に含める必要がある．FINE [4] および Rio File Cache [3] は，ハードウェアの故障からソフトウェアの故障まで様々な種類の故障を模擬可能にしている．Linuxでも，メモリ割り当てやI/O要求の失敗など機能的には限定されているが，一般的なミスによる故障を模擬する機能[*1]が使用可能である．

SimOS [5] は機能レベルからより詳細なレベルまで様々なレベルでのシミュレーションが可能なOSを実行可能なシミュレータである．SimOSは，フォールトインジェクションにより，Hive [2] の故障封じ込め（fault containment）が有効であることを示すために用いられた．

5 まとめと今後の課題

本論文では，OSの実行が可能な機能レベルプロセッサシミュレータを用い，ハードウェアの故障やソフトウェアのバグを模擬することで，OSのエラー処理が正しく行なわれるかどうかテストを行なうことを可能にするフォールトインジェクション環境について述べた．

今後の課題は，本論文で述べたフォールトインジェクション環境の実装をすすめ，評価を行なうことで有効性を示すことである．また，フォールトインジェクションを効果的に行なうため，フォールトインジェクションを自動化する環境の構築も必要である．

謝辞

本研究の一部は，JST-CREST「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」の助成による．

参考文献

[1] S. T. King, G. W. Dunlap, and P. M. Chen. Debugging Operating Systems with Time-Traveling Virtual Machines. In *Proceedings of the 2005 Annual USENIX Technical Conference*, pp. 1–15, April 2005.

[2] J. Chapin, M. Rosenblum, S. Devine, T. Lahiri, D. Teodosiu, and A. Gupta. Hive: Fault Containment for Shared-Memory Multiprocessors. In *Proceedings of The Fifteenth ACM Symposium on Operating Systems Principles*, pp. 12–25, December 1995.

[3] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proceedings of the Seventh International Conference on Architectural Support For Programming Languages and Operating Systems*, pp. 74–83, October 1996.

[4] W. Kao, R. K. Iyer, and D. Tang. FINE: A Fault Injection and Monitoring Environment for Tracing the Unix System Behavior under Faults. IEEE Transaction on Software Engineering, 19(11), pp. 1105–1118, November 1993.

[5] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete Computer System Simulation: The SimOS Approach. IEEE Parallel and Distributed Technology: Systems and Technology, v.3 n.4, pp. 34-43, December 1995.

[*1] CONFIG_FAULT_INJECTION