

B-033

SoftwarePot へのチェックポイント機構の導入 Integrating a Checkpointing Mechanism into SoftwarePot

横山 陽介[†]
Yosuke Yokoyama

大山 恵弘[†]
Yoshihiro Oyama

米澤 明憲[†]
Akinori Yonezawa

1. はじめに

チェックポイントシステムとは、実行中のプロセスの状態を保存/復元する機構であり、耐故障のための実行状態のバックアップやプロセスマイグレーションに用いられてきた。また、不特定多数のホストの遊休資源を借りてタスクを処理するシステム Condor[2] では、負荷分散のためのプロセスマイグレーションでチェックポイントシステムを使用している。

一方、SoftwarePot[1] は流通するソフトウェアを安全に実行するためのシステムである。ソフトウェアをファイルシステムごと Pot ファイルと呼ばれるファイルに固めて配布し、Pot 空間と呼ばれるサンドボックス内に閉じ込めた状態で実行する。Pot 空間は仮想ファイルシステムを持ち、そのファイルシステム上には 2 種類のファイルがある。1 つは、Pot ファイル内のファイルシステム上に実体があるファイル (static ファイル)、もう 1 つは、Pot 空間外、すなわち実行するホストのファイルシステム上に実体があり、実行時に Pot 空間内にマップされるファイル (map ファイル) である。これらのファイルへのアクセスは、システムコールの引数にファイルパスがある場合、Pot 空間内の仮想ファイルシステム上のパスから実体へのパスに書き換えることで実現される。

SoftwarePot は Pot 空間内のプロセスのシステムコールを監視し、Pot 空間内からの資源への意図しないアクセスを禁止することができるので、不特定多数のホストの遊休資源を借りてタスクを処理するときにセキュリティを確保するのに有用である。

しかし、SoftwarePot にはチェックポイント機構がないため、タスクの状態をバックアップしたり、Condor のようにプロセスマイグレーションを行い負荷分散を行うことが不可能である。そこで我々は、SoftwarePot にチェックポイント機構を導入することを提案する。

2. 方針

我々は、既存のチェックポイントシステムを利用して、SoftwarePot にチェックポイント機構を導入することにした。既存のチェックポイント機構を SoftwarePot に導入する上で考慮すべき問題が 3 点ある。

1 つめは、既存の実行バイナリをそのまま実行できるようにすべきことである。SoftwarePot はこれが利点の 1 つとなっている。多くのチェックポイントシステムは、プログラムソースに変換をかけたり実行ファイルにチェックポイント用ライブラリをリンクしたりする必要があるが、これはその利点を無にしてしまう。

2 つめは、プロセスの復元時に Pot 空間を構成するファイルの位置が異なることである。プロセスを復元するためには、保存時に開いていたファイルや mmap されて

いたファイルを再び開きなおさなくてはならない。しかし、static ファイルの実体は、実行時に作られる一時ディレクトリ上に展開された Pot ファイル内のファイルである。そのため、static ファイルの実体の位置は、実行毎に変化してしまう。また、map ファイルは実行時に Pot 内のファイル空間との対応付けを与えるため、対応付けが変化する可能性がある。よって、復元時にはファイルパスを適切に書き換える必要がある。

3 つめは、Pot 空間内のプロセスが従うべきセキュリティポリシーが保存時と復元時で異なる可能性があることである。SoftwarePot は、実行時にセキュリティポリシーを与えることができ、システムコールを監視することでポリシーに反する動作を禁じている。例えば、あるファイルを書込み可能にして開くことを禁止することができる。このポリシーは、実行環境に応じて異なるものを与えることが可能であるので、復元されたプロセスは、復元時に与えられたポリシーに従っている必要がある。例えば、書込み可能状態でファイルを開いているプロセスを保存したとする。これを復元する時、与えられたポリシーでそのファイルへの書込みが禁止されていれば、その復元動作は失敗せねばならない。

1 つめの問題点を解決するためには、実行バイナリを改変する必要のないチェックポイントシステムを使用する必要がある。この条件を満たすものとして、カーネルにドライバとして組み込むもの (CHPOX[3] 等) と環境変数 LD_PRELOAD を用いて実行時にチェックポイントライブラリをリンクするもの (ckpt[4] 等) があるが、我々は、チェックポイント機能の豊富さから CHPOX を採用した。

2 つめと 3 つめの問題点を解決するには、復元を行うプログラムを Pot 空間内で実行すればよい。Pot 空間内におけるファイルパス (保存時と復元時で変化しない) を使用すれば、SoftwarePot によってそれは適切なものに変換される。また、復元を行うプログラムが使用するシステムコールは、復元時に与えられたポリシーにしたがって検査が行われるので、ポリシーに反する操作は禁止される。例えば、ファイルを復元するために開くとき、open システムコールが検査されるので、ポリシーに反するファイルを開くことは禁止される。しかし、CHPOX はカーネル内で動作するため、復元を行うプログラムを Pot 空間内で実行することができない。そこで我々は、復元を行うプログラムを実装し、保存時のみ CHPOX を使用することにした。

保存/復元の大まかな流れは以下の通りである。CHPOX に保存対象のプロセスの ID を通知し、そのプロセスに特定のシグナルを送ると CHPOX によって現在のプロセスの状態がファイルに保存される。その後 CHPOX は SoftwarePot に保存完了の通知を行い、通知された SoftwarePot は現在の Pot 空間の状態を Pot ファイ

[†]東京大学

ルに保存する。また、CHPOX がファイルの状態を記録するとき、Pot 空間ではなく実体のファイルパスを記録するので、CHPOX が保存したファイルパスと Pot 空間内のファイルパスとの対応も保存する。プロセスの復元は、保存された Pot ファイルを実行することで行われる。復元を行うプログラムが保存された Pot ファイルに含まれており、その Pot ファイルを実行すると、Pot 空間内で復元を行うプログラムが最初の実行される。

3. 実装

我々は、Linux 2.4.26 上で本機構を実装した。

まず、プロセスの保存時に CHPOX が SoftwarePot に保存完了を通知し、それを受けて復元時に実行される Pot ファイルを生成する機構を実装した。Linux 版の SoftwarePot はカーネルモジュールを用いてシステムコールの監視をしているので、このモジュールに CHPOX が保存完了時に呼び出す通知用関数を追加した。この関数が呼び出されると、システムコールが発行された時と同様に SoftwarePot の監視プロセスに制御が移る。

保存完了後に Pot ファイルを生成する機構は、既に SoftwarePot に存在する、現在の Pot 空間のファイルを Pot ファイルに保存する機能を拡張することで実現した。現在の Pot 空間のファイルに加え、プロセスの状態を保存したファイル、CHPOX が保存したファイルパスと Pot 空間内のファイルパスとの対応を記したファイル、再開を行うプログラムの実行ファイルを保存するようにした。

次に、保存されたプロセスの復元を行うプログラムを実装した。現在の実装では、開いているファイルと mmap しているファイルは復元できるが、シグナルやソケット等の他の要素の復元は行わない。また、CHPOX は複数プロセスを同時に保存/復元できるが、我々の実装した復元プログラムは単一プロセスのみしか復元できない。

復元を行うプログラムはまず、保存されたプロセスのメモリ状態を mmap システムコール等を用いて復元する。保存されたファイルパスと Pot 空間のファイルパスの対応に基づいて、適切な位置にファイルをマップし、必要ならばメモリ上のデータを書き換える。次に、開かれていたファイルの状態を復元する。ここでも保存された対応に基づいて open システムコールで適切なファイルを開き、lseek システムコールで位置を戻す。最後に、汎用レジスタを復元し実行を再開する。

4. 実験

テストプログラムとして、Pot ファイル内のファイルをマップされた Pot 空間外のファイルにコピーするものを用いた。このプログラムを Pot 空間内で実行し、コピー途中で実行状態を Pot ファイルに保存した。その後、保存した Pot ファイルを実行したところ、実行状態が復元され正しくコピーが完了した。これにより、復元時にファイルの対応付けが正しく行えていることがわかった。

また、復元時のポリシーで Pot 空間外のファイルを書込み禁止にしたところ、復元に失敗した。これにより、ポリシーが変化しても正しく対応できていることがわかった。

5. セキュリティについての考察

一般に、チェックポイントシステム (特にカーネル組込み) には、状態を保存したファイルを書き換えることで、復元したプロセスの特権ユーザの権限で実行したり、通常は開けないファイルを開けてしまうなどの脆弱性が発生しうる。しかし、適切なポリシーを設定した Pot 空間内で復元を行うことで、このような脆弱性が発生しないことを保証できる。

6. 関連研究

プロセスマイグレーションをサポートした分散環境として、Condor がある。しかし、Condor は実行するプログラムに専用ライブラリをリンクする必要がある。また、システムコールは移動前のホストで実行し、実行結果を移動後のホストへ送信するという方法で実装されているため、移動前のホストにプロセスの一部を残してしまう上、通信によるオーバーヘッドが大きいといった欠点がある。

また、これまでに多くのチェックポイントシステムが提案されてきたが、実行環境全体のチェックポイントやセキュリティの確保についてはほとんど考慮されていない。

7. 今後の課題

まず、復元時に復元できるプロセスの要素を増やす必要がある。具体的にはシグナル、パイプ、ソケット、端末等に対応する必要がある。また、複数のプロセスを同時に保存/復元できることも必要である。

また、他のホストで復元するときなど、復元時に環境が異なる場合、Pot 空間外からマップされているファイルの内容が異なることがある。そのファイルをそのまま復元に使用しても正しく復元することができないので、Pot 空間外のファイルも必要に応じて保存・取得ができる機構も必要である。

参考文献

- [1] K. Kato and Y. Oyama. SoftwarePot: An Encapsulated Transferable File System for Secure Software Circulation. In *Software Security - Theories and Systems, volume 2609 of Lecture Notes in Computer Science*, pages 112–132, February 2003.
- [2] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System. Technical Report UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, April 1997.
- [3] O. O. Sudakov and E. S. Meshcheryakov. CHPOX. <http://www.cluster.kiev.ua/tasks/chpx.html>.
- [4] V. C. Zandy. ckpt: A process checkpoint library. <http://www.cs.wisc.edu/zandy/ckpt/>.