M-062

耐タンパプロセッサにおけるプロセス切り替え方式
Process switch for Multiparty Secure Processor

春木 洋美† 橋本 幹生† 川端 健†
Hiroyoshi Haruki Mikio Hashimoto Takeshi Kawabata

1 はじめに

近年、コンピュータシステムで扱われる著作物に対する著作権侵害が問題となっている。とりわけ、利用者端末のOSがマルチベンダ・マルチタスクのオープンシステムであるとき、この問題は深刻となる[1]。このため、利用者端末のOSが信頼できないことを前提として、マルチベンダ・マルチタスクのシステムにおいて、プログラムを解析や改ざんから守ることのできるL-MSP (License-controlling Multiparty Secure Processor)を提案した[2,3]。

L-MSPでは、タスクの秘密を保護するために、現在実行中のタスクIDを１つ保持し、これを用いてレジスタとキャッシュへのアクセスを制御する。しかし、パイプライン構成のプロセッサでは、複数の命令が同時に実行されるため、割り込み／再開直後には割り込み／再開前後のそれぞれのタスクに属する命令が混在する。この期間は、各タスクがレジスタセットやメモリにアクセスするため、適切なタスクIDやレジスタ情報を選択できない。

そこで、今回、我々は、プロセッサ上の複数のタスクを区別し、それぞれに適したタスク識別子とレジスタを選択するためのプロセス切り替え方式について提案する。これにより、パイプラインで構成されたプロセッサにおいても、他のタスクやOSからタスクの秘密を保護できる。

2 L-MSP

L-MSPは、以下の機能によりタスクの秘密を保護する。

- タスク管理機能
- タスクIDによるキャッシュ制御機能
- 暗号復号機能
- セキュアコンテキストスイッチ

まず、L-MSPでは保護対象のタスク毎に固有のIDを割り当て、保護タスクの状態を管理する。また、現在実行しているタスクのタスクIDを保持し、タスクの切り替え時には、このタスクIDを更新する。

次に、従来のキャッシュにある有効ビットとアドレスタグ情報に加えて、ラインの復号に利用したタスクIDを保存するタグを保持する。ヒット判定には、コアから指定されたアドレスとアドレスタグの比較に加えて、カレントタスクIDとIDタグの比較を行う。

そして、L-MSPでは、BIUに対称鍵暗号ハードウェア(SCH)を内蔵する。BIUでは、メモリにアクセスするとともに、タスクIDから選択した鍵で暗号・復号化する。

さらに、コンテキストを切り替えるハードウェア(SCS)を内蔵する。このハードウェアは、割込み／再開時に、割込み／再開前のタスクのレジスタ情報を退避させ、割込み／再開後のタスクのレジスタ情報を復帰させる。

このように、ハードウェアとタスクIDでタスク間の干渉を防ぐことにより、タスクの秘密を保護する。

†（株）東芝 研究開発センター、
Corporate R&D Center, TOSHIBA CORPORATION

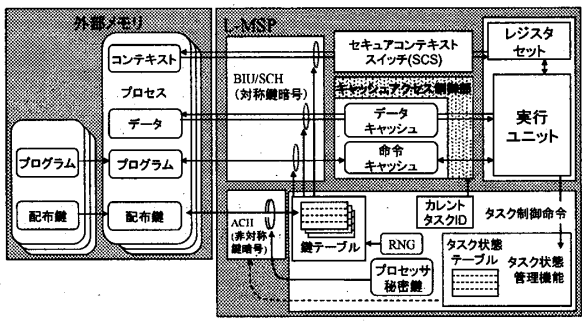

図１ L-MSPの構成

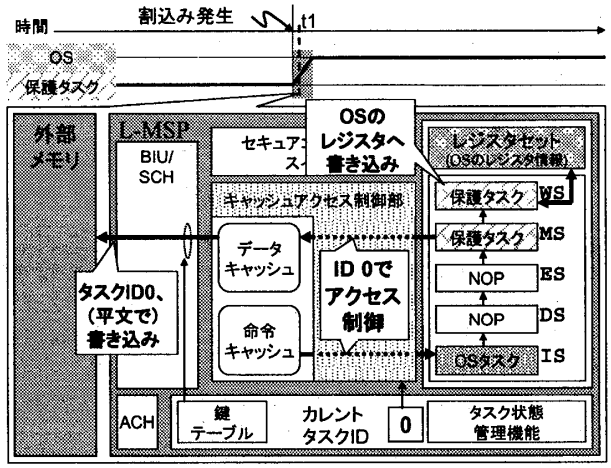

図２ 割込み直後における従来のL-MSPの状態

3 L-MSPにおけるプロセス切り替え

3.1 L-MSPのパイプライン化における問題点

パイプライン構成のプロセッサでは、複数の命令が同時に実行されるため、割込み／再開直後にはその前後のタスクに属する命令が混在する。L-MSPをパイプライン構成にした場合、割込み／再開直後において異なるタスクが同時にレジスタやメモリにアクセスするため、タスクの秘密を保護できない。

図2のように、命令フェッチ(I)、デコード(D)、実行(E)、メモリアクセス(M)とレジスタ書き込み(W)の5ステージの構成をL-MSPに適用すると仮定する。例えば、割込み時に切り替えたとする。割込み時のレジスタ情報の切り替えにより、Wステージでは保護タスクの命令がOSのレジスタ情報へ書き込む。割込み時のカレントタスクIDの更新により、Mステージでは保護タスクの命令がID 0でキャッシュに書き込み、保護タスクのデータを平文で外部メモリに保存する。

また、割込み前のタスクが完了した時、つまり、保護タスクがWステージで実行した後、切り替えたとする。保護タスク完了までレジスタ情報を切り替えないため、Dス

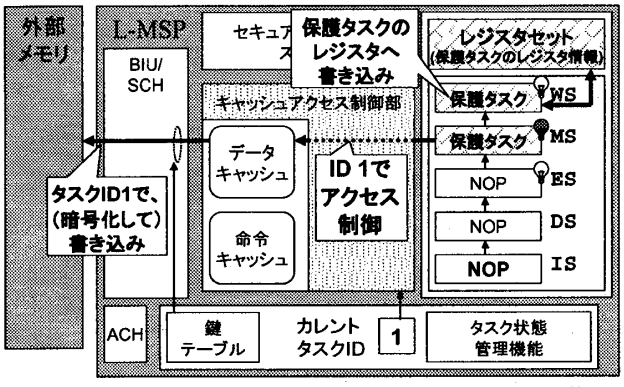

図３　割込み直後における 3.2 方式の L-MSP の状態

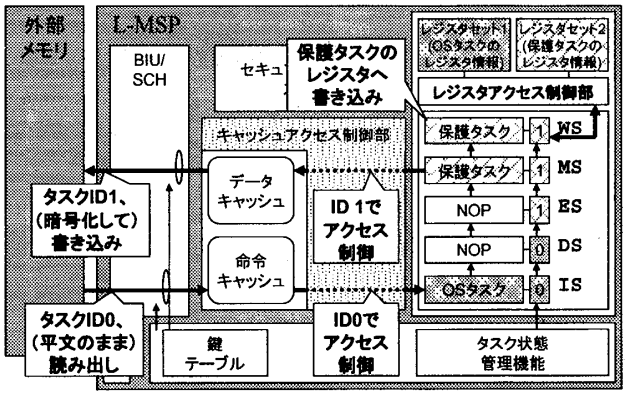

図４　割込み直後における L-MSP の状態

テージでは OS タスクの命令が保護タスクのレジスタへアクセスする。また、保護タスク完了までタスク ID を更新しないため、I ステージではタスク ID 1 で OS タスクの命令をフェッチする。

つまり、L-MSP の仕組みをそのままパイプライン化すると、他のタスク ID によるメモリアクセスや他のタスクのレジスタへのアクセスができるという問題が生じる。以下で、この問題への対策方式を提案する。

3.2 対策方式１：命令フェッチの待ち合わせ

対策方式１として、命令フェッチの待ち合わせによりプロセッサ内に複数タスクの混在を防ぐ方法を提案する。図 3 のように、E ステージから W ステージに切り替えフラグを保持する。割込み時には、E ステージのフラグを設定し、同時に命令フェッチを待ち合わせる。フラグが W ステージに達した時、レジスタ情報の退避・復帰の後、カレントタスク ID を更新し、命令フェッチを再開する。

このように、パイプライン上に ID の異なるタスクを混在させないことにより、タスクの秘密を保護する。

3.3 対策方式２：タスク ID のパイプライン化

対策方式２として、タスク ID のパイプライン化して、ステージ毎にそれぞれのタスク ID でアクセス制御を行う方法を提案する。図 4 のように、複数のレジスタセット、ステージ毎に命令に対応するタスク ID を保持し、タスク ID からレジスタセットを選択するハードウェアを内蔵する。I ステージでは、命令を読み出す時、対応するタスク ID を設定する。タスク ID は命令と共に次のステージに移る。割込み／再開時に、I ステージの ID を更新する。

例えば、図 4 において、I ステージでは、OS タスクの ID 0 でキャッシュへアクセスするが、M ステージでは、保護タスクの ID 1 でキャッシュへアクセスする。また、次の時間、D ステージでは、OS タスクの ID 0 でレジスタセットを選択するが、W ステージでは、保護タスクの ID 1 でレジスタセットを選択する。

このように、ステージ毎に保持するタスク ID でアクセス制御することにより、複数のタスクが混在してもタスクの秘密を保護できる。

3.4 考察と実装

これらの提案方式にはそれぞれ利点と欠点がある。3.2 の方式では、ハードウェアの規模も小さく既存のプロセッサへの適用が容易であるが、割込み発生からレジスタ切り替えまでのストール時間が生じる。一方、3.3 の方式では、ストール時間は生じないが、ステージ毎のタスク ID のパイプライン化によるハードウェアの規模が大きく、既存プロセッサへの適用が容易でない。

今回、これらの利点を組み合わせて、割込み前のタスク完了まで待ち合わせて、タスク ID と複数のレジスタセットを切り替える方式を、MeP(Media embedded Processor)に実装した。この方式では、タスク切り替え時に生じるレジスタ情報の退避、復帰のストール時間を削減できる。また、そのプロセッサ上で複数の保護タスクと非保護タスクが切り替わりながら実行することを確認した。これは、パイプラインで構成されたプロセッサにおいても、保護タスクのレジスタ情報やメモリ情報を、割り込み前後の他のタスクや OS から保護できることを示している。

4　おわりに

従来の L-MSP では、パイプライン構成のプロセッサにおいて他のタスク ID を用いたメモリアクセスや他のタスクのレジスタ情報にアクセスできるといった問題があった。その解決策として、パイプライン上に複数のタスクを混在させない方法と、ステージ毎に保持されたタスク ID を用いてレジスタ情報やメモリ情報にアクセスする方法を提案した。また、この方法を既存のプロセッサに実装し、その実現可能性を示した。

このように、割込み／再開前のタスクの命令と割込み／再開後のタスクの命令を区別することにより、パイプライン構成のプロセッサにおいても他のタスクや OS からタスクの秘密を保護できる。

今後の課題として、各方式に必要なハードウェアコストとパフォーマンスへの影響の調査があげられる[4]。

参考文献

[1] Ahmet M.E. et al: “Protecting Intellectual Property in Digital Multimedia Networks”, IEEE Computer, Special Issue on Piracy and Privacy, pp.39-45(2003).

[2] 橋本他: “敵対的な OS からソフトウェアを保護するプロセッサアーキテクチャ”, 情処論 ,Vol.45, No.SIG3(ACS 5), pp.1-10 (2004).

[3] 春木他: “耐タンパプロセッサ(L-MSP)システムの実装とプロセス管理”, SCIS2004, pp.549-554 (2004)

[4] Jefferey C.M. et al: “The Effect of Context Switches on Cache Performance”, ACM ASPLOS (1991).