社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－134／2008－EMB－8（6）
2008／3／27

Linux OSのセキュリティプロセッサ向け移植

藤松 由里恵　春木 洋美　橋本 幹生

株式会社東芝 研究開発センター 〒212-8582 川崎市幸区小向東芝町1
E-mail: {yurie.fujimatsu, hiroyoshi.haruki, mikio.hashimoto }@toshiba.co.jp

あらまし 近年，アプリケーションソフトウェアの解析・改変を防止してソフトウェア保護を実現するセキュリティプロセッサの研究が進められている．我々はシステムのOSが信頼できないという前提の下，ソフトウェア保護を実現するセキュリティプロセッサL-MSP (License-controlling Multi-vendor Secure Processor) を提案し，L-MSP向けに改変したμITRONによりOSとの整合性を検証してきた．L-MSPにはレジスタ値に対する解析・改変を防止するため，保護対象プロセスのコンテキスト切り替えをハードウェアコンテキストスイッチによって行うなど，既存のプロセッサとは異なる特徴がある．本論文では，これらL-MSPのソフトウェア保護機構をLiunxに適用する際に生じる問題を解決するためのプロセス管理機構の提案と組込み向けuClinuxにおける実証結果を報告する．
キーワード セキュアプロセッサ，Linux，移植，ソフトウェア保護

Porting of Linux OS to secure processor

Yurie FUJIMATSU　Hiroyoshi HARUKI　and　Mikio HASHIMOTO

Research & Development Center, TOSHIBA Corp　1, Komukai, Toshiba-cho, Saiwai-ku, Kawasaki, Kanagawa-ken, Japan
{yurie.fujimatsu, hiroyoshi.haruki, mikio.hashimoto }@toshiba.co.jp

Abstract Recently, secure processors have been recognized as effective means of protection of software running on untrustworthy hostile operating systems. L-MSP(License-controlling Multi-vendor Secure Processor) is a secure processor which provides strict protection against dynamic attacks on running software with manipulation of memory data and register value using OS privilege. Secure hardware context switch, which is the characteristic mechanism of L-MSP, prevents register value manipulation attack. But the mechanism brings much impact on process management of existing OSs such as Linux. In this report, a proposal of revised process management mechanism of Linux adapted to L-MSP's protection mechanism is presented. The result of testbed implementation supports effectiveness of proposed mechanism.
Keyword secure processor, Linux, porting, software protection

1. はじめに

1.1. ソフトウェア保護

近年，情報家電システムの普及に伴い，ソフトウェアの解析・改変が問題となっている．ユーザ端末で使用されるソフトウェアに埋め込まれた情報が，ユーザによって解析・改ざんされる事件も発生している．たとえば，OSのオープンソース化によって，OSを利用したメモリの解析・改ざんを行う攻撃などを通してソフトウェアに埋め込まれた秘密や動作が，ユーザに漏れてしまう可能性もある．こうした攻撃からソフトウェアを守るための秘密保護機構が必要とされている．

こうした秘密保護技術として，プロセッサチップが物理的耐タンパ性を持つことを前提として，プロセッサに埋め込まれた鍵に基づいて，プログラムを保護するプロセッサが知られている[1, 2, 3]．我々は，従来のマルチタスクOS機能の秘密保護機能をプロセッサ内部に持たせることにより,たとえ信用できない敵対的なOSの管理下であっても，安全なソフトウェア保護を実現するセキュアプロセッサL-MSP (License-controlling Multiparty Secure Processor)を提案した[4, 5, 6]．L-MSPは，プロセスが保護メモリ領域を持つことによって解析・改ざんから保護して安全に実行する機構を提供する．L-MSPでは，ユーザからアクセス可能な外部メモリ上では，保護したいプロセスのプログラム・データ・コンテキストを暗号化し，

プロセッサ内部でのみ復号して平文を処理する.また, L-MSP にはレジスタ値に対する解析・改変を防止するため, 保護対象プロセスのコンテキスト切り替えをハードウェアコンテキストスイッチによって行うなど, 既存のプロセッサとは異なる特徴をもつ.

これまで, プロセッサ L-MSP, および L-MSP の保護メカニズムをサポートするプロセス管理方式について検討を続けており, 我々は, 平文・暗号プログラムを共存させ, 既存 OS が備えるマルチタスク機能との整合性も持たせている. これ以降, 提案のプロセス管理方式を実装した OS を L-MSP OS と呼ぶ.

L-MSP と L-MSP OS は, 原理的には各種既存のプロセッサや Linux などの OS を適用することが可能である.また,L-MSP ではプロセッサ内部でプロセスの状態を管理するため,状態管理情報と既存 OS でのプロセス管理との整合性を保たなければならず,これが大きな課題であった. そこで,従来,我々は MeP[7]と μITRON[8]を L-MSP 向けに改造し,既存の CPU/OS と L-MSP の枠組みとの整合性を検証してきた[9].

しかしながら,より高機能なシステムとの整合性を検証するには,Linux OS への移植を検討する必要がある.一方で,μITRON と Linux とはプロセス管理部分が異なるため,従来方式をそのまま適用するのではなく,Linux 向けに再検討する必要がある.

そこで,本論文では, L-MSP のソフトウェア保護機能を Linux に適用する際に生じる問題を解決するためのプロセス管理機構を提案する. また, 本提案方式を uClinux に適用し, その上で保護されたアプリケーションが動作することを確認する.

本論文では, 2 章で L-MSP 向け Linux 移植に向けた準備として L-MSP と Linux のプロセス管理手法について説明し, 3 章で L-MSP 向けの Linux のプロセス管理手法の検討を行い, 4 章で実装と検証, 5 章で考察を行い, 6 章でまとめを行う.

2. L-MSP 向け Linux OS 移植の準備

Linux の L-MSP 向け移植の準備として, L-MSP と Linux のプロセス管理について述べる. まず, L-MSP 概要を説明し, L-MSP と Linux のプロセス管理手法について説明する.

2.1. L-MSP 概要

L-MSP ではプロセッサのキャッシュ内部でのみプログラムを平文にし, 実行する. それ以外では常に暗号化された状態となり, ユーザからも解析・改ざんができない. これはあらかじめ, 保護対象のプログラムをプログラム鍵で暗号化して配布し, プログラム鍵をプロセッサの秘密鍵に対応した公開鍵で暗号化して得られた配布鍵を配布することで実現可能である. コンテキストスイッチ操作を使って OS がレジスタの情報を読み出すことを防ぐため, ハードウェアコンテキストスイッチ機能も備える. 図 1 に L-MSP の構成図を示す.

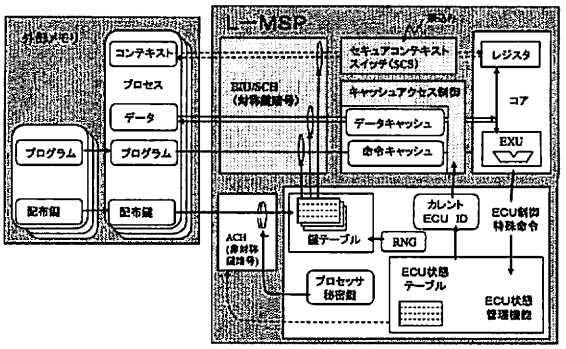


図 1 : L-MSP の構成

2.2. Linux のプロセス管理

L-MSP のプロセス管理ついて, プロセスのライフサイクル (生成・実行・切り替え・終了) 毎に説明する.

2.2.1. L-MSP のプロセス管理概要

L-MSP ではプロセッサが保護プロセスを認識する. プロセッサは, プロセス管理情報のうち, 秘密保護と実行制御に必要な部分を OS から独立して保持する. この情報の管理単位を ECU と呼ぶ. ECU は個別 ID (ECU ID) が割り当てられ, この ID に基づいてプロセッサは ECU の状態や暗号鍵管理などの ECU 保護機能を内蔵する. L-MSP では, OS は特殊命令を発行することで, ECU の生成, 開始, 再開, 削除を行うことができる. 特殊命令と ECU 状態, 状態遷移について表 1, 図 2 に示す.

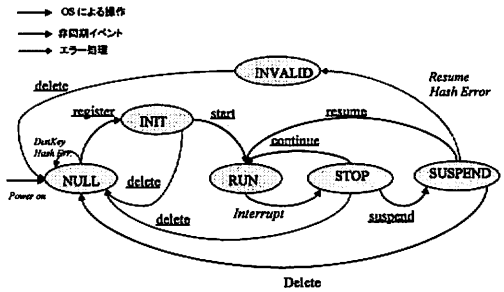
OS が ECU の秘密情報を直接操作することはプロセッサの仕様により禁止されている. つまり, OS はプロセス管理の資源管理機能を持ち, プロセッサは秘密保護機能を持つことでプロセス管理機能を分離し OS が信頼できないケースにおいても安全にソフトウェア実行させることができる.

表 1 : 特殊命令

状態名	内容
register	配布鍵の復号と命令鍵の登録, コンテキスト鍵初期化
start	初期状態から ECU 実行開始
continue	中断状態から ECU 実行開始
resume	レジスタ情報を退避
suspend	レジスタ情報の復帰
delete	ECU 関連情報の削除

2.2.2. L-MSP の ECU 生成

保護プロセスの実行を開始する前に,OS はプログラム鍵などの ECU 情報をプロセッサに通知するため



NULL:初期状態
INIT:配布鍵が復号され、プログラム鍵がテーブルに登録された状態
RUN:ECU実行状態
STOP:ECU実行停止状態
SUSPEND:レジスタ情報がメモリに保存された状態
INVALID:登録済みECUの実行中止状態

図 2：ECU 状態遷移

ECU 登録命令を発行する．ECU ID と配布鍵を指定して register 命令を発行することにより，プロセッサは配布鍵を復号し，取り出されたプログラム鍵を指定した ECU ID に関連付けて登録する．

2.2.3. L-MSP の ECU 実行

保護プロセスを実行時には，OS は ECU ID を指定して start 命令を発行し，制御が実行する保護プロセスのプログラム開始番地に移行する．それ以降命令フェッチで読み出された命令は，指定した ECU ID に対応した鍵で復号されて EXU で実行解釈される．

2.2.4. L-MSP の ECU 切り替え

割り込みが発生し，保護プロセスの切り替えが起こる場合，OS はプロセッサ内部のハードウェアコンテキストスイッチ機能を利用してコンテキストの保存，復帰を行う．保護プロセスが利用するコンテキストは，割り込み処理と同時に暗号化されて外部メモリに書き出される．そのため，コンテキストを保存するために OS が明示的に行う処理はない．

ECU の実行が中断されると，ECU 保護機能がレジスタ値をコンテキスト鍵によって暗号化して，外部メモリに退避する．実行復帰の際は，OS が ECU ID を指定して continue 命令を発行して，ECU 保護機能がレジスタ値を復号することによって復帰する．OS は ECU の実行再開を指示することはできても，レジスタの値については知ることはない．

2.2.5. L-MSP の ECU 削除

保護プロセスが終了すると，OS は delete 命令を発行してプロセッサの内部に保持されたプロセス関連情報を消去する．ある ECU ID に対する削除操作が完了すれば，OS は新たに別のプログラムを ECU ID に割り当て再利用が可能になる．

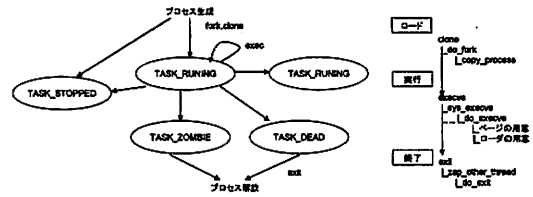


図 3：Linux のプロセスの状態遷移とライフサイクル

2.3. Linux のプロセス管理

Linux のプロセス管理手法についてプロセスのライフサイクル毎に説明する[10].

2.3.1. Linux のプロセス管理概要

Linux は,実行する基本的な単位をプロセスとして管理し，複数のプロセスを同時に実行することができる．プロセスは親子関係を持ち，個々のプロセスに関する情報を task_struct 構造体に格納し管理している．task_struct の持つメンバとしては，プロセスの実行状態を制御するための実行状態情報やメモリ管理，実行ファイルのローダなどがある．また，Linux のプロセスはユーザモードで動作するため，ハードウェアに直接アクセスすることができず，サービスコールを発行することでドライバがハードウェアにアクセスする．Linux におけるプロセスの状態遷移について図 3 に示す．図 3 のようにイベントが起こる毎に各状態へ遷移する．

2.3.2. Linux のプロセス生成

Linux のプロセス生成は，fork()または clone()システムコールで行う．プロセス生成のシステムコールの発行を通知されると，親プロセスは子プロセスを作成し各種資源を複製する．ただし，fork()の時点では新しいプロセスの生成は行うが実行しているプログラムは親プロセスと同じものである．また，プロセス生成と同時に実行可能状態に遷移する．

2.3.3. Linux のプロセス実行

プロセスの実行は，exec()システムコールの発行によって起こる．ここで，それまでの空間を破棄し，新たな空間にプログラムをロードし実行を開始する．

2.3.4. Linux のプロセス切り替え

Linux のプロセス切り替えは，割り込み発生時に起きる．schedule()関数においてのみ切り替えが実行され，新しい空間を有効にし，カーネルモードスタックとハードウェアコンテキストの切り替えを行う．

2.3.5. Linux のプロセス終了

exit()システムコールで終了する．プロセス終了の場合，メモリの解放や親プロセスへの通知などの処理を

行う.

3. L-MSP 向け Linux OS のプロセス管理手法の検討

前章で，L-MSP と Linux のプロセス管理について述べたが，L-MSP の保護機構を実現するために，Linux のカーネル部分にそれぞれのプロセスライフサイクルに対応させてマッピングをおこなっていく必要がある．ライフサイクル毎に必要な処理を検討する．

3.1. プロセス管理

L-MSP の機構を実現するためには，非保護プロセスは従来通りの処理を行い，保護プロセスは秘密保護機能を実現する各種処理を行わなければならない．そのためには，保護・非保護プロセスを区別する必要がある．

そこで，Linux のプロセス管理部分に,保護プロセスであるか否かを示すフラグを追加する．また,このフラグを利用し,プロセスの生成,実行,削除時に,L-MSP 特有の特殊命令の発行を行うか否かを判定する．

3.2. Linux における ECU 生成

L-MSP では,保護プロセスを実行するためには,ECU を生成するための register 命令と，ECU を開始するための start 命令を発行する必要がある．

Linux の場合，ECU の生成と類似するシステムコールとして，fork システムコールがある．fork システムコールではプロセスを生成するが,そのプロセスは親プロセスのままでありメモリ上では親プロセスのプログラムがマッピングされている．register 命令の発行には，保護プロセスのプログラムがマッピングされているメモリ領域を知る必要があるため，fork システムコールに register 命令を追加することはできない．そこで，保護プロセスのプログラムがメモリにマッピングされる exec システムコールで，register 命令を発行する必要がある．

3.3. Linux における ECU 実行

ECU 実行と類似する Linux のシステムコールに exec がある．上記で述べたように，ECU 生成のための register 命令も同じシステムコールにおかれるため，register 命令発行後，start 命令を発行するようにマッピングする．

3.4. Linux における ECU 切り替え

まず,ECU の切り替えとプロセスの切り替えについては，割り込み及び割り込み復帰に同期するため，これらについて特に検討する必要はない．

次に，ECU の退避・復帰については，プロセスの切り替えを行う部分に対してそれぞれの特殊命令を追加する．退避対象のプロセスが，保護プロセスである場合には ECU の退避命令を，復帰対象のプロセスが保護プロセスである場合には，ECU の復帰命令をそれぞれ追加して発行する．

表 2　L-MSP 向け対応処理

ECU　特殊命令	L-MSP 対応処理
ECU 生成　register	exec システムコール
ECU 実行　start	exec システムコール
ECU 停止	割込みに同期
ECU 切り替え suspend	schedule 関数に追加
ECU 切り替え resume	schedule 関数に追加
ECU 再開　start	Schedule 関数に追加
ECU 削除　delete	exit システムコール

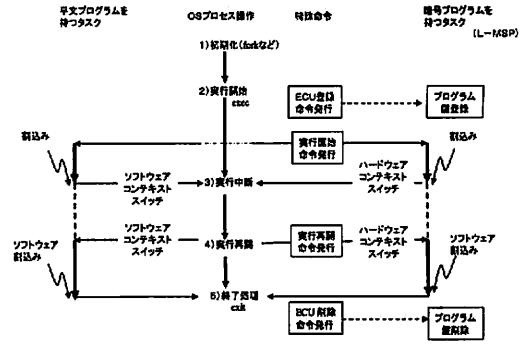


図 4：L-MSP 向けの Linux プロセスライフサイクル

3.5. Linux における ECU 削除

ECU の削除は，その情報が不要となるプロセスの終了にマッピングする．プロセスを削除するとき,該当のプロセスが保護プロセスであれば,delete 命令を発行する．

以上を踏まえて，表 2 に各プロセスライフサイクルへの L-MSP の保護機構機能のマッピングをまとめる．また，L-MSP 向けの Linux のプロセスライフサイクルの一連の流れを図 4 に示す．

4. 実装と検証

実装と検証について記述する．

4.1. 実装環境

本提案では，以下の環境を使って実装を行った．

OS：アックス社 axLinux[11]

ボード：L-MSP 向け改良済み MeP ボード

コンパイル環境：GNU for MeP

Linux OS として今回 uClinux を対象とした．これは，L-MSP が MMU なしの MeP で実装されているため，MMU をもたない Linux である uClinux が適していたためである．uClinux をベースとした組み込み向け Linux である axLinux for MeP を利用した．

4.2. 実装

3 章で各プロセスライフサイクルにマッピングした

特殊命令をカーネルに追加した．また，プログラムの起動については，非保護アプリはシェルから起動を行い，保護アプリに関しては，保護アプリ起動用のアプリを別途用意して，そのアプリの中でパスを指定して起動する．プロセスの切り替えについては切り替え元・切り替え先の組み合わせとして保護→保護，非保護→非保護，保護→非保護，非保護→保護の4パターンがあり，各ケースについて矛盾なく動くようにsuspend, resume, continueをマッピングした．

4.2.1. ECUの生成・実行の実装

まず，プロセスの保護・非保護を区別するために，プロセス管理を行うtask_struct構造体に対して保護フラグ情報をメンバとして追加した．この保護フラグプロセス情報を使って,プロセススケジュールの管理を行う．

また，3章にてECU生成命令はexec内にマッピングするのが適していると考えたが,具体的な実装を検討する．

本提案では新たなシステムコールを追加して2つのプロセス実行システムコールを保護・非保護によって使い分ける方法で実装を行う．例えば，非保護プログラム用のexecveと保護プログラム用のcncl_execveといった別々のプロセス実行システムコールを設計する．保護・非保護のプログラムの入力に合わせてどちらかを実行する．非保護プログラムに関しては従来のexecveをそのまま使用することで問題はない．保護プログラムの場合は，保護プログラムロード，ECU初期化，配布鍵登録，プロセス実行といった処理を追加して実行する．この手法は，改造部分があまり多くなくてすむという利点がある．

保護アプリの実行の際は，従来のプロセス生成のシステムコールsys_execveと同様の処理＋L-MSP向け処理を行う新規システムコールを発行する．新規execveシステムコールの流れは図4のように実現した．

```
sys_cncl_execve()  新規追加システムコール
|_mep_cncl_execve()  新規追加
   |_lock_kernel
      |_do_cncl_execve  新規追加
        |_ open_exec()
          |_ task_struct構造体にECUID設定
          |_配布鍵登録/コンテキスト保存先登録
```

図4：配布鍵登録の追加項目

4.2.2. プロセス切り替え

プロセス切り替え時においても同様にschedule関数に従来の非保護プロセス用の処理に加えて，保護プロセス用の処理を追加し，切り替え元・切り替え先のプロセスの保護・非保護に合わせて，処理を行う機構を追加した．

切り替え元が保護プロセスならば，suspend命令を発行して，コンテキストの退避を行い，切り替え先が保護プロセスならば,コンテキストの復帰を行うため，ECU IDを指定してcontinue命令を発行し，コンテキストを外部メモリから呼び出し，復号してレジスタに復帰させ，実行を再開する必要がある．それには，切り替え元，切り替え先を明示的に判断できる箇所でのマッピングを行えばよい．スケジューラ機能を実現する関数schedule()では，切り替え元と切り替え先のプロセスが一致しないことを条件としてcontext_switch関数を実行し，コンテキストの切り替えを実行する．この部分に保護プロセスフラグ情報から保護アプリかどうかを判別し，分岐する実装を追加し，非保護の場合は，通常のcontext_switchを実行し，保護タスクの場合は，新たに追加したcontext_switch機能+コンテキスト退避・復帰機能を備えたAPIを実行する．流れは以下の図5のように実現した．

```
schedule()
 |_prev = current;  //切り替え元プロセス構造体設定
 |_ next = rq->idle;//切り替え先プロセス構造体設定
 |_switch_tasks: if(likely(prev !=next))
 | //切り替え先と切り替え元が一致しない場合
 |_ if prev->ecuid = 1 || next->ecuid =1
     |_context_switch(prev,next,last)
     |_cncl_context_switch(prev,next,last)
       |_ suspend  コンテキストの暗号化
       |_cncl_ switch_to コンテキストの切り替え
       |_ resume コンテキストの復号
       |_continue 実行の再開（割込みベクタ）
```

図5：プロセス切り替え時の追加項目

4.2.3. プロセス終了

プロセス終了については，exit内で保護プロセスフラグ情報より保護・非保護の判別を行い，保護プロセスの場合はdelete命令を発行し，プロセッサ内部のプロセス関連情報を消去する．以下図6のように実現した

```
exit
 |_zap_other_thread
 |_do_exit
    |_deleteTask 新規追加 削除処理
    |_deleteECU  新規追加 ECUIDの初期化
```

図6:プロセス終了時の追加項目

4.3. 動作確認

ユーザランドに非保護アプリと保護アプリを追加し，シェル上でこれらのアプリを同時に実行する．タイマ割込みなどのソフトウェア割り込みによって，保護アプリの ECU 状態が正しく推移することを確認した．割込みに関しては，切り替え先，切り替え元の非保護・保護の組み合わせ 4 つ共全てにおいて状態遷移は正しくおこなわれることを確認した．

また，オリジナルのカーネルと変更後のコードサイズと OS の改造によるオーバヘッドを測定するために実行クロック数を調べた．クロック数に関して,ECU 生成・実行は追加したシステムコール部分を,ECU 切り替えは suspend, resume 追加分と context_switch 部分を,ECU 削除は,exit へ追加した ECU 削除処理部分を測定している．それぞれ作成したテストベッド上の値である．表 3,表 4 に結果をまとめる．

表 3：L-MSP 対応にコード量の比較

項目	コード量（従来）	コード量（変更後）
保護フラグ追加	—	1 行
ECU 生成・実行	1583 行	2098 行
ECU 切り替え	5280 行	5360 行
ECU 削除	1505 行	1535 行
合計	8368 行	8994 行

表 4：L-MSP 対応にクロック数の比較

項目	実行クロック数（従来）	実行クロック数（変更後）
ECU 生成・実行	2856075	5005510
ECU 切り替え	987	5572
ECU 削除	—	1917

5. 考察

今回の実装で，L-MSP 向けのプロセス管理モデルを Linux のプロセス管理モデルに適合させることができた．Linux OS に対してコードサイズの増加やオーバヘッドが発生する．表 4 の実行クロック数に関しては,ECU の生成では RSA 復号処理で 450 万クロックかかるためである．また，ECU 削除では,該当 ECU に対応するデータのキャッシュフラッシュのための遅延と考えられる．

また,今回実装で利用した axLinux では,プログラムの暗号化を行っていない．通常，保護アプリはアプリ開発者によって暗号化保護された状態で渡されることを前提としている．しかし,axLinux の場合,ロード時にバイナリの書換えが発生する FLAT 形式のバイナリのみをサポートする．この書換えの発生により，そのままでは復号処理が行えない．そのため,今回の実装では,axLinux のバイナリ書換えを正しく実行させるために,競合するプログラムの暗号化行っていない．今回のこの制約については,axLinux を位置独立コード PIC（Position Independent Code）対応の FLAT 形式ををサポートするように改造すればバイナリの書き換えが発生しなくなり，プログラムの暗号化が実現できる．

6. まとめ

本論文では，新たに Linux OS の移植を試み,プロセス管理を行うカーネル部分を L-MSP 向けに改造し実装を行った．結果，非保護アプリと保護アプリがプロセス生成・実行・切り替え・終了の一連のプロセスライフサイクルにおいて正しい状態遷移情報を持って動作することを確認した．

文　献

[1] Arnold, M. g. and Winkel, M. D.: Computer systems to inhibit unauthorized copying, unauthorized usage, and automates cracking of protectes software, U.S. patent, No.4,558,176(1985)

[2] Gilmont, T. and Legat, J.-D.: Hardware security for software Privacy Support, Electronic Letters, Vol.35, No. 24, pp.2096-2098(1999)

[3] Lie,D.,Thekkath,C.A.and Mitchell, M.,Lincoln, P., Boneh, D., Mitchell, J. C. and Horwitz, M.: Architectural Support for Copy and Tamper Registant software, ASPLOS 2003, pp. 168-177(2000)

[4] 橋本幹生，春木洋美: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，情報処理学会コンピュータシステムシンポジウム 2003 予稿集，pp.17-26(2003)

[5] 橋本幹生，春木洋美，川端健:オープンソース OS と共存可能なセキュリティプロセッサ技術，東芝レビュー60 巻 6 号，pp.44-47(2005)

[6] Mikio Hashimoto, Hiroyoshi Haruki, Takeshi Kawabata: Secure Processor Consistent with Both Foreign Software Protection and User Privacy Protection. Proceedings of Security Protocols Workshop 2004(LNCS3957): 276-286

[7] 東芝セミコンダクター社：Media embedded processor, http://www.semicon.toshiba.co.jp/product/micro/mep/index.html

[8] TOPPERS プロジェクト:TOPPERS/JSP カーネル，http://www.toppers.jp/

[9] 春木洋美，橋本幹生，川端健: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,暗号と情報セキュリティシンポジウム，2004.

[10] 高橋 浩和，小田 逸，山幡 為佐久(著)，Linux カーネル 2.6 解読室，ソフトバンククリエイティブ，東京，2006.

[11] 株式会社アックス：axLinux for MeP，http://www.axe-inc.co.jp/