IXP425上での暗号ミドルウェアOCFのスケジューラの実装と評価

大釜 正裕† 杉浦 寛† 羅 鏡栄† 関口 聖美† 黒羽 秀一† 齋藤 孝道†

†明治大学
214-8571 神奈川県川崎市多摩区東三田 1-1-1

{ohgama, kan_s, oscar_weng, skgchi, kuroba, saito}@cs.meiji.ac.jp

あらまし 組込みシステムにおいて，暗号処理の高速化の一手法として，暗号処理専用モジュールへのオフローディングがある．しかし，OpenSSLなどの暗号ライブラリは，一般的なコンピュータアーキテクチャを前提としており，組み込みシステムでの利用には制限がある．そのため，ミドルウェアを用意し，OpenSSLと組み込み機器の中間を埋めることが求められる．さらに，複数の暗号処理を行いたいプロセスが，Intel社のIXP425のひとつの暗号処理専用モジュールを利用する際には，スケジューリングが重要である．そこで，本論文では，暗号処理専用モジュールを内蔵するIXP425上でOpenSSLを用いて，暗号ミドルウェアにおけるオフロード処理のスケジューリング方法の実装と評価を行う．

Implementation and Evaluation of Scheduler of Code Middleware OCF on IXP425

Masahiro Ohgama† Kan Sugiura† Keang-Weng Lo† Kiyomi Sekiguti †
Shuiti Kuroba† Takamichi Saito†

†Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki-shi, Kanagawa 214-8571, Japan

{ohgama, kan_s, oscar_weng, skgchi, kuroba, saito}@cs.meiji.ac.jp

Abstract In an embedded system, a hardware cryptographic module is used to offload a part of operation. However, since cryptographic library such as OpenSSL can not be applied to it, we utilize cryptographic middleware such as OCF. When we utilize OCF to execute multi processes on IXP425, we need to schedule resources for performance. Therefore, we implement OCF scheduler on IXP425, and evaluate it.

1 はじめに

ネットワーク通信における暗号処理の高速化を図るために，暗号処理専用モジュールを内蔵したネットワークプロセッサ (以下，NPと呼ぶ) が開発されている．そのため，暗号処理の負荷を暗号処理専用モジュールにオフロードすることで全体としての処理の高速化を期待できる．

一方，アプリケーションの開発で，様々な暗号処理を行いたい場合，OpenSSL [1] のように，広く利用され，実績のある暗号ライブラリを利用するのが望ましいとされている．しかしながら，一般的に，組み込み機器での利用はサポートされていないことも多い．そこで，OpenSSLと組み込み機器との間を繋ぐミドルウェアを利用し，その適用を実現することができる．

OpenSSLは，様々な暗号処理専用モジュール

(AEP, Chil, Cswift など) に対応しているが, 今回利用した, IXP425 へのオフロードには対応していない. そこで, ミドルウェアである OCF [2] を用いることで, IXP425 へのオフロードを実現した [3]. しかし, オリジナルの OCF において, 暗号処理専用モジュールがひとつしかない環境下で, 複数のプロセスが OCF に対して暗号処理のリクエストを発行した場合, 実行中のプロセスの暗号処理の途中で別のプロセスに制御が移り, 実行中のプロセスの暗号処理が遅延されてしまう.

そこで, 本論文では, OCF 内で暗号処理専用モジュールを利用するプロセスを制御することにより, 個々のプロセスの暗号処理を遅延無く実行するためのスケジューラ機能を実装し, その評価を行う.

2 IXP425 の概要

ここでは, 本論文で使用した NP である IXP425 の概要について述べ, IXP425 のハードウェアアーキテクチャを図 1 に示す. IXP425 は, 主に, XScale コアとパケット処理専用モジュールである NPE[2] A と NPE B から構成されている. XScale コアは, NPE A, NPE B と周辺装置などの制御を行い, NPE A と NPE B は, 主に IP パケットの処理を行う. Queue Manager は, XScale と NPE A または NPE B 間の通信に利用するハードウェアキューの制御を行う.

NPE B のみが内蔵する暗号処理専用モジュールは, 共通鍵暗号化方式の DES, 3DES と AES に対応しており, その暗号利用モードとして, CBC (Cipher Block Chaining) と ECB (Electronic Code Book) が利用可能である. また, ハッシュアルゴリズムとして SHA-1 と MD5 に対応している.

3 開発環境

3.1 IXP425 評価ボード

評価ボードは, 主に, IXP425, プログラムメモリである Flash ROM, メインメモリである SDRAM と 2 つの NIC から構成されている. 評価ボードの基本仕様を表 1 に示す.

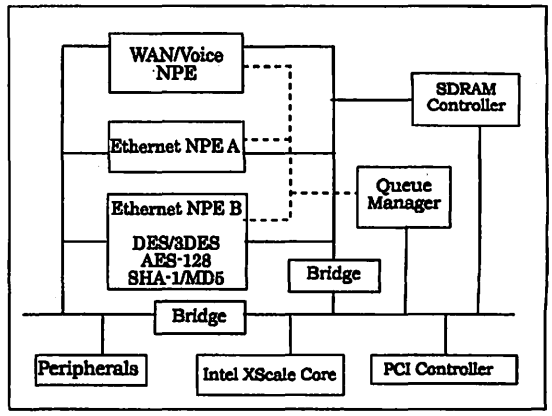

図 1: IXP425 のハードウェアアーキテクチャ

表 1: 評価ボードの基本仕様

プロセッサ	IXP425 533MHz
Flash ROM	16Mbyte
SDRAM	64Mbyte (133MHz で動作)
NIC	10BASE-T/100BASE-TX (2 基搭載)

3.2 μCLinux

評価ボード上では, μCLinux [4] を OS として利用する.

μCLinux は, embedded Linux/microcontroller project が開発した, MMU (Memory Management Unit) が無いプロセッサ上でも動作する Linux であり, 一般的な Linux と同様のシステムコールを提供し, LKM (Loadable Kernel Module) をサポートしている.

3.3 暗号処理専用モジュールの準備

3.3.1 概要

IXP425 の暗号処理専用モジュールである NPE B が提供する暗号処理機能を利用するための一手法として, IXP4xx シリーズ向けの API である IxCryptoAcc API [5] を利用することが挙げられる. しかしながら, この API はデバイスドライバの開発向けの API のみを提供する. そのため, アプリケーションから暗号処理機能を利用するためには, まず, 専用のデバイスドライバを実装し, そのデバイスドライバにアクセスする仕組みが必要となる.

そこで, 本論文では, OpenSSL と OCF (OpenBSDCryptographic Framework) を組み合わせた利用基盤を構築し, OCF 経由でユーザ空間から暗号処理専用モジュールにアクセス

[2]NPE は Network Processor Engine の略称である.

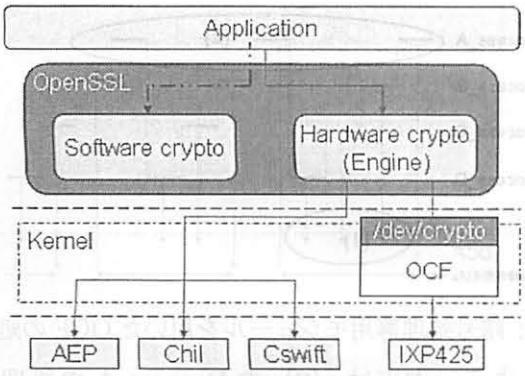

図 2: 利用の概要

する（図 2）．以下，AccessLibrary，OpenSSL と OCF の関係について示す．

3.3.2 OCF

OCF は，IXP425 の暗号処理専用モジュールを含む様々なハードウェアアクセラレータ[3]への透過的なアクセスを提供する API とデバイスドライバから構成されており，OpenSSL，OpenSSH や OpenSwan を利用したアプリケーションからハードウェアアクセラレータが提供する暗号処理機能を利用するための共通のインタフェースを提供するカーネルモジュールである．

ここでは，OCF の機能をその役割ごとに，3 つの層に分類し，上位から，インタフェース層，マネージャ層，デバイスドライバ層と呼ぶこととする．以下，各層の役割について示す．

インタフェース層

インタフェース層では，デバイスドライバとアプリケーションとのインタフェースを提供しており，アプリケーションが OCF に対して，ioctl システムコールを発行することで，OCF にアクセスし，暗号処理専用モジュールの機能を利用することが出来る機構を実現している．

さらに，インタフェース層では，暗号処理に先がけた，アプリケーションと OCF 間のセッション（以降，OCF セッションと呼ぶ）の生成，それから，暗号処理後の OCF セッションの削除を行う．ここで OCF セッションとは，アプリケーションと OCF が，セッション ID，暗号化方式，秘密鍵を共有している状態を指す．

[3]IXP422，IXP425，IXP465（以上，Intel 社），Hifn 7951，Hifn 7956（以上，Hifn 社），SafeXcel 1142，SafeXcel 1741（以上，SafeNet 社）の暗号処理専用モジュールに対応している．

マネージャ層

マネージャ層は，インタフェース層で受け取った暗号処理のリクエストに応じて，デバイスドライバが提供する API を呼び出すディスパッチャとして機能する．

さらに，下位のデバイスドライバがサポートする暗号化方式の管理も行い，上位アプリケーションのリクエストに応じて，適切なデバイスドライバの選択を行う．

デバイスドライバ層

デバイスドライバ層は，暗号処理専用モジュールを直接制御するための機能を提供する層であり，上位マネージャ層に対して，主に，暗号処理のための API を提供する．

3.3.3 暗号処理専用モジュール利用の概要

ここでは，OpenSSL と OCF を組み合わせて用いた場合の，暗号処理専用モジュールの利用方法と暗号処理の流れを示す．以下，OpenSSL と OCF 間および OCF 内の処理について説明する．

まず，OpenSSL から OCF にアクセスするための方法について述べ，次に，OCF の利用方法について述べる．

OpenSSL からは，キャラクタ型のデバイスノードである `/dev/crypto`（メジャー番号：10，マイナー番号：70）に対して，`open()` や `read()` といったシステムコールを呼び出すことで，各システムコールに対応する OCF 内で定義された関数を呼び出すことができる．この仕組みには，Linux の標準的な file_operations 構造体を利用しており，例えば，図 3 に示すように，アプリケーションは `open()`，`ioctl()` を呼び出すことにより，OCF 内の `cryptodev_open()`，`cryptodev_ioctl()` を呼び出すことができる．この仕組みにより，OpenSSL からの OCF へのアクセスを実現している．

OCF の利用法として，OCF を介して IXP425 の暗号処理専用モジュールである NPE B の暗号処理機能を利用する場合，OpenSSL は，`ioctl()` を利用し，`/dev/crypto` に対して，暗号処理のリクエストを発行する．OpenSSL は，OCF に対して暗号処理のために，主に，`CIOCGSESSION`，`CIOCCRYPT` と `CIOCFSESSION` の 3 つのリクエ

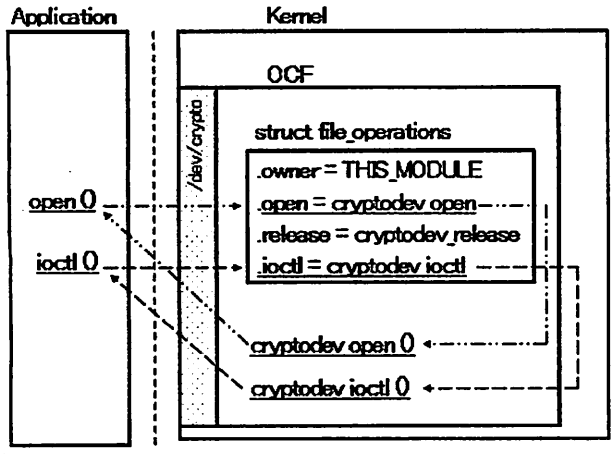

図 3: OCF における file_operations 構造体の利用

ストを発行する．ただし，これらのリクエストは，あらかじめ OpenSSL で定義されている．

OpenSSL は，CIOCGSESSION を発行することで，OCF 間での OCF セッションを生成する．この時，OpenSSL は，暗号アルゴリズム，ブロック暗号の利用モードと暗号化鍵 を指定し，OCF のマネージャ層で，利用可能な暗号処理専用モジュールを選択する．OCF セッションが確立すると，次に，CIOCCRYPT を発行し，暗号処理を行う．そして，暗号処理が終了すると，CIOCFSESSION を発行し，OpenSSL と OCF 間の OCF セッションを解放する．

4 設計と実装

4.1 概要

OCF は，複数のプロセスが暗号処理を行おうとする場合，アプリケーションから暗号処理のリクエストを受け取った順番に，処理を行う．しかしながら，暗号処理専用モジュールがひとつしかない環境下では，暗号処理専用モジュールを使用しているプロセスのタイムスライスが短い場合，暗号処理の途中で処理が中断され，カーネルによるプリエンプションされ，コンテキストスイッチが入る．そのために，暗号処理専用モジュールを用いた暗号処理の間に，複数の別プロセスからの暗号処理のリクエストが処理され，暗号処理専用モジュールを使用していたプロセスの実行が遅延される場合がある．例えば，図 4 の (1) に示す場合では，Process_A が暗号処理専用モジュールを使用している間は，Process_B, C, D は暗号処理専用モジュールを使用できないため，Process_B, C, D は割り当てられたタイムスライスを使い切るまで，無駄な処理を行っ

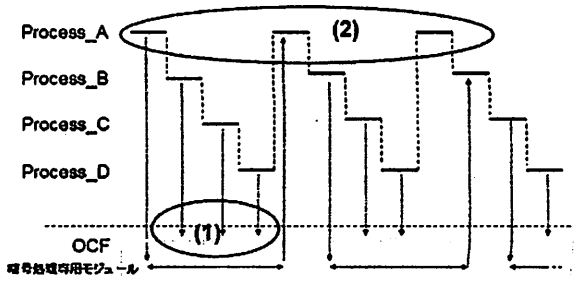

図 4: 暗号処理専用モジュールを用いた OCF の処理

てしまう．本来は，(2) の Process_A の処理はまとめて行う方が効率がよい．

そこで，本論文では，OCF 内で，OCF セッションの生成から解放までの処理の間に，OCF を通して，暗号処理専用モジュールの利用待ちをしている別のプロセスに制御を移させないために，OCF に，OCF を用いる全てのプロセスの内，ひとつのみが実行可能状態になり，他のプロセスは待ち状態になるように制御するスケジューラを追加した．

4.2 OCF スケジューラ

本論文で，OCF に導入したスケジューラ（以下，OCF スケジューラと呼ぶ）は，OCF 内でカーネルスレッドとして動作し，複数のプロセスからの暗号処理をスケジューリングする．OCF スケジューラで，暗号処理を制御するにあたって，OCF 内にリスト (以降，sched リストと呼ぶ) を用意した．sched リストは，OCF に暗号処理のリクエストを発行したプロセスのプロセス ID を保持する．これを実現するため，まず，OCF に以下の修正を行った．

- OCF セッションの確立時に，sched リストが空であるかどうかを確認する．
  空である場合：sched リストにプロセス ID を登録し，そのまま次の処理に進む．
  空でない場合：sched リストの最後にプロセス ID を登録し，現在の処理を中断し，待ち状態に入る．
- OCF セッションの解放時に，sched リストから自身のプロセス ID を削除し，OCF スケジューラを実行可能状態にする．

これらの OCF の修正により，プロセスは OCF セッションの生成時に，暗号処理専用モジュールが他のプロセスに利用されている場合，処理を

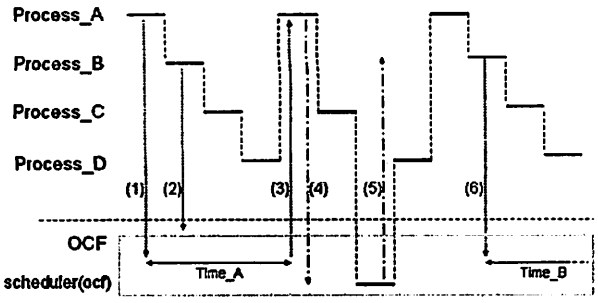

図 5: スケジューラ機能を実装した OCF の処理

中断する．その後，OCF スケジューラが sched リストに登録されたプロセス ID を基に OCF を使用するプロセスの状態「待ち状態」から「実行可能状態」へ遷移させる事で，OCF を利用するプロセスを制御することができる．

以下，OCF スケジューラの動作について示す．

スケジューラの動作：

1. sched リストの初期化を行う．
2. sched リストが空かどうかを確認する．空でないときは 3 の処理に，空のときは 4 の処理に進む．
3. sched リストの先頭からプロセス ID をひとつ取り出し，そのプロセスを実行可能状態にする．
4. 暗号処理専用モジュールを利用中のプロセスにより，起こされるまで，待ち状態に移行される．起こされたら 2 の処理に戻る．

4.3 処理手順の例

ここでは，2 つのプロセス A，B(以降，Process_A, Process_B とそれぞれ呼ぶ) が暗号処理をそれぞれ行うケースを例として示す (図 5)．ただし，2 つのプロセスは Process_A，Process_B の順で生成される．また，その時点で，既に，OCF，暗号処理専用モジュールを利用しない 2 つのプロセス C，D (以降，Process_C, Process_D と呼ぶ) が動作している．図 5 中の Time_A と Time_B はそれぞれ，Process_A と Process_B が暗号処理専用モジュールを占有している期間を表している．

以下，Process_A が実行状態となった時点からの動作を図 5 と対応させて示す．

(1) Process_A は，まず，OCF セッションを確立するために，OCF に CIOCGSESSION リクエストを発行する．Process_A は，次に，sched リストが空であることを確認し，sched リストに自身のプロセス ID を登録する．そして，Process_A は暗号処理のための CIOCCRYPT リクエストを OCF に発行する．

(2) Process_B は，Process_A と同様に，CIOCGSESSION リクエストを発行し，OCF との OCF セッションを確立する．ここで，Process_B は，sched リストに Process_A のプロセス ID が登録されていることを確認し，wait_event_interruptible を呼び出し，自ら現在の処理を中断して待ち状態に入る．

(3) 暗号処理専用モジュールでの Process_A からオフロードされた暗号処理が終了すると，Process_A は，sched リストから自身のプロセス ID を削除する．

(4) Process_A は，wake_up_process を呼び出し，待ち状態にある OCF スケジューラを実行可能状態にする．その後，CIOCFSESSION リクエストを OCF に発行し，OCF セッションを解放する．

(5) OCF スケジューラに制御が移ると，sched リストの先頭から Process_B のプロセス ID を取り出し，waku_up_process により，Process_B を実行可能状態にする．

(6) Process_B に制御が移ると，処理を再開し，暗号処理のための CIOCCRYPT リクエストを発行する．

5 評価

ここでは，オリジナルの OCF とスケジューラ機能を導入した OCF (以下，sched OCF と呼ぶ) のパフォーマンス計測とその比較を行う．

5.1 評価環境

今回提案した，OCF スケジューラの評価環境として，IXP425 評価ボード上で，μCLinux, OpenSSL-0.9.7c, OCF の実装である ocf-linux-20041204 を用いて構築した．

5.2 評価項目

オリジナルの OCF と sched OCF の性能評価として，一時に，M 個のプロセスから暗号化を行い，各プロセスで N 回の暗号化の時間計測を行う．プロセス数 M は 4, 8, 12, 16, 20 とし，暗号化回数 N は 10 から 10 刻みに 1000 ま

での値を取る．このとき，1 個めのプロセスを生成する直前から，全ての生成したプロセスの暗号化が終わるまでの時間 t を計測する．ただし，各プロセスが一回に暗号化する平文のデータサイズは 1024byte とし，暗号化方式として 3DES，暗号利用モードとして CBC モードを利用する．それぞれ処理時間を 10 回，計測し，その平均値を結果として示した．プロセス数 M が 20 の時の オリジナルの OCF と sched OCF を比較した結果を図 6 に示す．

5.3 計測結果

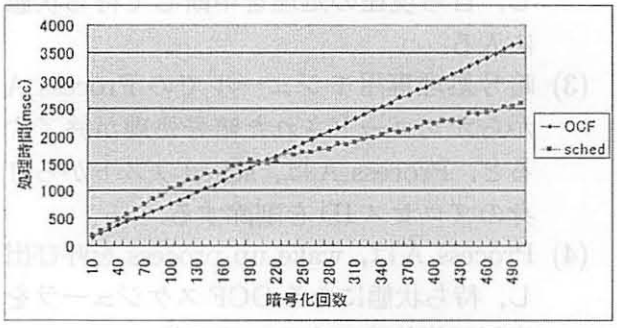

図 6: OCF と sched の比較 1

暗号化回数が約 200 回を超えるあたりから sched OCF の処理時間が オリジナルの OCF の処理時間と比べ，速くなっている．

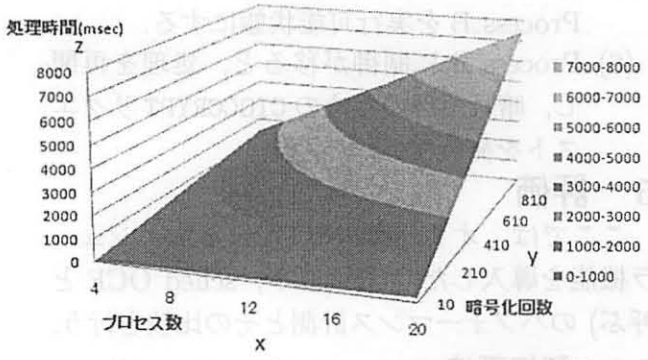

図 7: オリジナルの OCF の処理時間

さらに，プロセス数と暗号化回数の関係を調べるため，それらのパラメータを変化させながら計測を行った (図 7, 図 8)．図 7，図 8 の x 軸はプロセス数 M，y 軸は暗号化回数 N，z 軸は処理時間 t をそれぞれ表す．プロセスひとつあたりの暗号化回数の増加にともない，sched OCF の処理時間がオリジナルの OCF の処理時間に比べて速くなっていることが分かる．

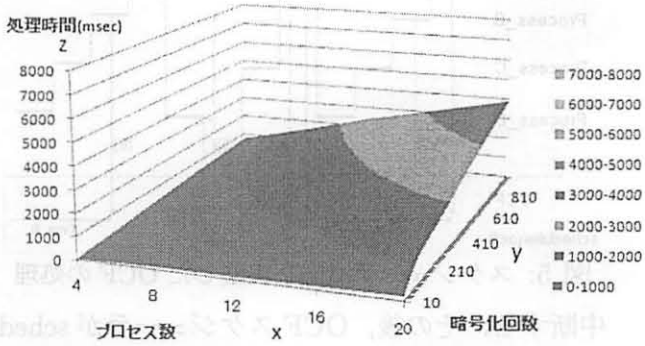

図 8: sched OCF の処理時間

6 まとめ

本論文では，Intel 社の NP である IXP425 を搭載する評価ボードを用いて，複数のプロセスが，暗号処理専用モジュールに暗号処理をオフロードする際に，効率的に暗号処理を行うスケジューラを提案し，それを OCF に適用した．さらに，その評価を行い，オリジナルの OCF より性能が良くなることを示した．

謝辞

本研究の成果の一部は，科学研究費補助金（課題番号 19700070）の助成を受けたものである．

参考文献

[1] http://www.openssl.org/

[2] A. D. Keromytis, J. L. Wright, and T. de Raadt, "The Design of the OpenBSD Cryptographic Framework", In Proceedings of the USENIX Annual Technical Conference, June 2003.

[3] 黒羽秀一，齋藤孝道: 暗号モジュールの効率的な利用のためのスケジューラの検討，暗号と情報セキュリティシンポジウム 2008 概要集，p.39，2008.

[4] http://www.uclinux.org/

[5] Peter Barry and Gerard Hartnett, "Designing Embedded Networking Applications", March 2005, Intel Press.

[6] Alan O. Freier, Philip Kocher, and Paul C. Kaltorn, "The SSL Protocol Version 3.0 draft", March 1996, http://home.netscape.com/eng/ssl3/draft302.txt

[7] Tim Dierks and Christopher Allen, "RFC2246:The TLS Protocol Version 1.0", Jan 1999.

[8] http://www.openssl.org/docs/crypto/engine.html