Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

マルチプロセッサ対応 RTOS のテスト開発

松浦 光洋[1)] 金 ハンソル[2)] 眞弓 友宏[3)]
金 承燁[2)] 廉 正烈[2)] 金 榮柱[2)]
木村 貴寿[4)] 鴫原 一人[5)] 馬 鋭[6)]
森 孝夫[3)] 本田 晋也[3)] 山本 雅基[3)] 高田 広章[3)]

名古屋大学の附属組込みシステム研究センターでは複数の企業が研究コンソーシアムを形成し，オープンソースの RTOS(TOPPERS)に対する包括的なテストスイートの開発とテストの実施を行っている．第一段階としてシングルプロセッサに対応した RTOS を対象として，RTOS の API に着目したテストを開発した．本論文では， 開発した API テストをマルチプロセッサに対応した RTOS へ拡張するために定めたポリシーについて述べる．その上で，ポリシーに基づいて行ったテスト開発例を紹介する．

The Test Development for Multiprocessor RTOS

Mitsuhiro MATSUURA[1)] Hansol KIM[2)]
Tomohiro MAYUMI[3)] Seungyup KIM[2)]
Jungyeol YEOM[2)] Youngjoo KIM[2)]
Takatoshi KIMURA[4)] Kazuto SHIGIHARA[5)] Rui MA[6)]
Takao MORI[3)] Shinya HONDA[3)] Masaki YAMAMOTO[3)]
and Hiroaki TAKADA[3)]

The Center for Embedded Computing Systems in Nagoya University built a research consortium by some companies and is developing comprehensive test suites for "TOPPERS", an open-source RTOS. As a first step, we developed tests focused on the API for single processor RTOS. This paper describes policies aimed at expanding the tests to multiprocessor RTOS. It also describes examples of tests developed using these policies.

1. はじめに

IT 系のシステム構築におけるオープンソースは, Linux や Apache に代表されるように，広く活用されている．組込みシステムの構築においても，同様にオープンソースの活用が進んでいる．

Toyohashi OPen Platform for Embedded Real-time Systems (TOPPERS)プロジェクト[1]では，ITRON 仕様[2]のリアルタイム OS（以下，RTOS）技術を出発点として，組込みシステム構築の基盤となる各種のソフトウェアを開発し，オープンソースソフトウェアとして公開している．これまでに，ITRON 仕様をベースとしたシングルプロセッサ対応の RTOS である TOPPERS/ASP カーネル（以下，ASP）や，通信ミドルウェアなどを公開しており，実製品に利用されている．また，近年，PC だけでなく組込みシステムの分野においても，マルチプロセッサシステムの重要性が急速に増していることに対応し，TOPPERS プロジェクトでは，ASP を拡張して，マルチプロセッサに対応した TOPPERS/FMP カーネル（以下，FMP）を開発し，オープンソースとして公開している．

ASP および FMP の開発は，非営利で行われており，製品への組込み時には利用者側での改変や拡張が行われることが一般的である．したがって，カーネルのテスト開発と実施は，カーネル開発者側では行わずに，利用者側が行うことが通例である．

しかしながら，RTOS に対するテストは膨大であるので，利用者側のテスト開発は負荷が大きく，さらにマルチプロセッサに対応した RTOS に関しては，そもそもテスト経験が少ないために，テストが困難である．

そこで，名古屋大学大学院情報科学研究科附属組込みシステム研究センター(NCES)[3]は，RTOS のテストを行うコンソーシアム型の研究組織を立ち上げ，複数の企業と団体の参加を得て，ASP と FMP に対するテスト手法の検討とテストスイートの開発を実施している. 最初にシングルプロセッサ向けの RTOS である ASP のテストプロセスを開発し，テスト設計を行い，テストを実施した．その後，ASP のテストプロセスとテスト設計結果を利用して, マルチコアプロセッサ対応の RTOS である FMP のテス

[1)] 有限会社松浦商事
Matsuura Corporation
[2)] 株式会社デジタルクラフト
Digital Craft Inc.
[3)] 名古屋大学
Nagoya University
[4)] 日本電気通信システム株式会社
NEC Communication Systems, Ltd.
[5)] 富士ソフト株式会社
FUJISOFT INCORPORATED
[6)] 三洋電機株式会社
SANYO Electric Co., Ltd.

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

トケースの設計とテストの実施を進めた．

本論文では，この研究組織で実施した FMP の API テストのテストスイートの開発事例について述べる．

本論文の構成は次の通りである．まず 2 章で概要を，3 章で API テストの手順について述べ，4 章で本プロジェクトが対象とした RTOS を説明する．5 章で テストケース抽出のポリシー策定について，6 章でそれに基づいたテスト設計の実例と実施状況を紹介する．

2. テスト開発の概要

2.1 体制

本研究は，NCES を中心とした複数の企業によるコンソーシアムによって実施され，参加企業はその成果を自由に使用できる．なお，開発成果は，一定期間後にはオープンにされる．また，参加する研究者を高度な研究開発型人材として育成することも目的の一つとして，研究を推進する．コンソーシアムに参加する企業と団体の一覧を表 1 に示す．

表 1　研究コンソーシアムの参加企業一覧（五十音順）

三洋電機株式会社	富士ソフト株式会社
株式会社デジタルクラフト	有限会社松浦商事
株式会社東芝 セミコンダクター社	宮城県産業技術総合センター
日本電気通信システム株式会社	株式会社ルネサステクノロジ

2.2 テスト対象

本研究では，名古屋大学が中心となり開発し TOPPERS プロジェクトで配布している ASP と FMP をテスト対象とした．ASP と FMP は，TOPPERS 新世代カーネル（以下，新世代カーネル）と総称されており，信頼性と安全性とソフトウェアポータビリティを向上させるために，ITRON 仕様に拡張と改良が加えられている．仕様は「TOPPERS 新世代カーネル統合仕様書」[6]としてまとめられ，TOPPERS プロジェクトから配布されている．

2.3 テストの全体像

最初に，ASP および FMP のテストとして実施すべき内容を抽出し，次の 8 種類に分類した．

(A) API（静的 API 含む）に着目したテスト
(B) 処理単位に着目したテスト
(C) ターゲット依存部単独でのテスト
(D) 割込み禁止区間に注目したテスト
(E) タイマ割込み処理のテスト
(F) ロック区間に注目したテスト
(G) スピンロック中の割込みのテスト
(H) マイグレーションに関するテスト

次に ASP を対象として(A)のテストを設計し，実施した．この経験を活かして，FMP を対象とした(A)のテストを設計し，実施した．

3. シングルプロセッサ対応 RTOS の API テスト開発

本章ではシングルプロセッサ RTOS である ASP 向けに開発したテストプロセス及びテスト設計と実施の結果について述べる．

3.1 実施した API テストの概要

API テストでは，仕様に定められた振舞いが正しく実行されるかを API 毎に確認する．具体的には，API 発行前のシステム状態（前状態）を定義し，その状態でテスト対象となる API を発行（処理）し，API 発行後のシステム状態（後状態）を確認するテストプログラムを実行する．

3.2 開発した API テストの設計・実施プロセス

API テストの設計と実施プロセスを図 1 に示す．

最初に，テスト設計者が，テスト対象である API の振舞いを仕様書に基づいて理解し，テストケースを抽出する．次に，そのテストケースを実施した場合のソースコードカバレッジを手作業で確認する．これらの情報は，API 仕様及びソースコードと共に，テストシートにて管理される．次に，テストケースの文言から，前状態と処理と後状態を抽出して整理し，テストシナリオを作成する．act_tsk の API に関するテストシナリオを図 2 に示す．

次に，テストシナリオから YAML 言語による等価なテストシナリオデータを作成する．図 3 に，図 2 に示したテストシナリオに対応するテストシナリオデータの例を示す．

その後，テストシートとテストシナリオデータのレビューを行い，必要があれば修正を行う．その後，テストシナリオデータを TOPPERS Test Generator(TTG)に入力し，テストプログラムを作成する．

なお，TTG は，「OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成」[4][5]の OJL(On theJob Learning)の開発テーマとして開発された，テストシナリオデータからテストプログラムを自動生成するツールである．

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

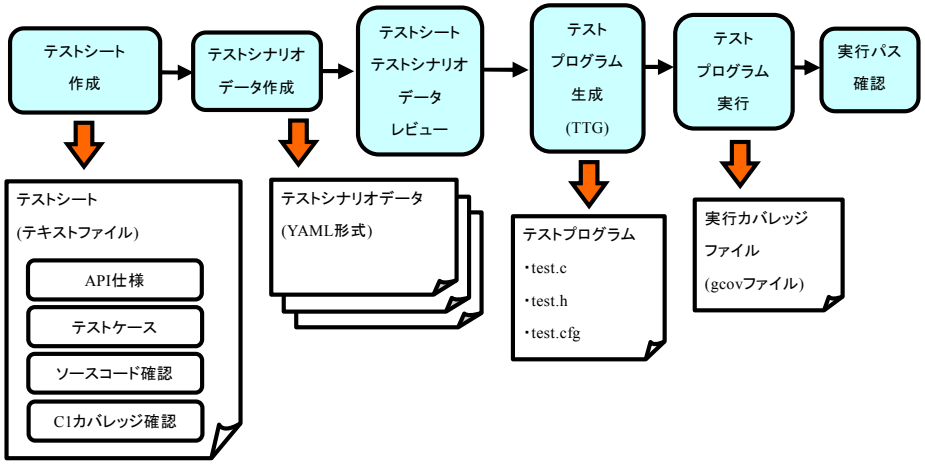

図 1　API テストの設計・実施のプロセス

```
前状態
        タイプ          : タスク
        ID              : TASK1
        優先度          : 中
        状態            : 実行

        タイプ          : タスク
        ID              : TASK2
        優先度          : 高
        状態            : 休止

処理
        TASK1 が act_tsk(TASK2) を発行する.

後状態
        タイプ          : タスク
        ID              : TASK1
        優先度          : 中
        状態            : 実行可能

        タイプ          : タスク
        ID              : TASK2
        優先度          : 高
        状態            : 実行
```

図 2　テストシナリオの例(act_tsk)

```
pre_condition:
  TASK1:
    type    : TASK
    tskpri  : MID
    tskstat: running

  TASK2:
    type    : TASK
    tskpri  : HIGH
    tskstat: dormant

do:
  - id: TASK1
    syscall: act_tsk(TASK2)

post_condition:
  TASK1:
    tskstat: ready

  TASK2:
    tskstat: running
```

図 3　テストファイルの例(act_tsk)

最後に，テストプログラムをターゲット上で実行し，テストを実施する．同時にコードカバレッジを取得するツールである gcov により，API テストプログラムを実行した際に通ったパスを取得し，手作業で求めた，通過が期待されるパスと照らし合わせ，期待したパスをテストしていることを確認する．

3.3 テストケース抽出ポリシー

テストケース抽出ポリシーは，テストケース抽出のばらつきを防ぐために定められた指針である．ここには，同値分割の指針や，境界値テストの適用の指針などが含まれており，テストの範囲やテストケース数もポリシーに依存する．ポリシーは，テストケースの抽出時だけでなく，テストシートやテストシナリオレビュー時の指針としても使用される．ポリシー作成においては，仕様面の分析だけでなく，RTOS のソフトウェア構造の分析，及びコンソーシアム型研究に参加した各社の経験が生かされる．

3.4 実施した API テストの範囲

以上のプロセスにより，ASP に関してはコードカバレッジが 100%となる API テストプログラムを開発し，実施を完了した[7]．

4. マルチプロセッサ対応 RTOS の仕様

FMP は ASP をマルチプロセッサ向けに拡張した RTOS であり，機能分散型と対称型の両タイプのマルチプロセッサシステムに適用可能である．なお，マルチプロセッサとは，複数のプロセッサで構成され，各プロセッサが同時に動き，互いに協調して

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

一つのシステムを形成するプロセッサを指す.

本章では，ASP から FMP への拡張の際に追加された仕様の概要を述べる.

(1) **ASP の API のマルチプロセッサ拡張**

ASP の API の多くは, FMP では異なるプロセッサに割り付けられたオブジェクトの操作が可能なように拡張されている．例えば，act_tsk は，マルチプロセッサ拡張により，発行元とは異なるプロセッサに割り付けられているタスクを起動可能となる.

さらに，API によっては仕様自体も拡張されているものもある．なお，オブジェクトとは, カーネルまたはシステムサービスが管理対象とするソフトウェア資源であり, 処理単位と呼ばれるカーネルまたはシステムサービスが管理対象とするプログラムやソフトウェア資源である[6]．具体的な処理単位としては，タスクや周期ハンドラやアラームハンドラなどが挙げられる.

(2) **タスクの状態の追加**

ASP/FMP には，ディスパッチ禁止状態や CPU ロック状態などの要因で，カーネルがディスパッチを行わない状態が存在する. この状態をディスパッチ保留状態という. ASP，FMP いずれの場合も，カーネルがディスパッチ保留状態にあるときには，実行状態のタスクが強制待ち状態に遷移することはない．これは API の機能として制限されているためである．しかし FMP において，他のプロセッサに割り付けられた実行状態のタスクによって強制待ち状態へ遷移させられることは制限することが出来ない．そのため，実行状態と強制待ち状態の間の過渡的な状態となる．この状態を「強制待ち状態[実行継続中]」と呼ぶ.

(3) **新規 API**

FMP には他のプロセッサに割り付けられているオブジェクトの操作や参照をする，またはプロセッサ間の排他制御を行うための 6 種類 17 個の API が新規に追加されている.

(4) **クラス**

FMP には，複数のプロセッサが管理するソフトウェア資源の集合を定義するために，クラスという概念が追加されている．システムの起動時に処理単位をどのプロセッサで実行するかを示す「初期割付けプロセッサ」や，起動後に処理単位の割付プロセッサを変更（マイグレーションと呼ぶ）する際に，割り付けることができるプロセッサを制限する「割付け可能け可能プロセッサ」などを定義する.

5. マルチプロセッサ対応 RTOS の API テスト開発

5.1 概要

本テストにおいては，２つのプロセッサを持つシステムを対象とした. 理由は, FMP が提供するほとんどの API の基本的な振る舞いを，２つのプロセッサを持つシステム上で一通り確認できること，及び RTOS の構造面の分析から，３つ以上のプロセッサであり得るケースの多くが，２つのプロセッサシステム上でのケースと「同値」とみなせることが多いためである.

また，FMP を対象とした API のテストケースを，ASP を対象とした API のテストの拡張として作成した．そのため，テストケースは次の２つに分類される.

1. ASP より流用したテストケース
2. 新規に作成したテストケース

前者は，マルチプロセッサを構成するいずれかのプロセッサの一つに，全ての処理単位を割り付けてテストをする時に使用するテストケースである．ASP 用に作成したテストケースをそのまま利用する．後者は，4 章で述べた ASP から FMP への拡張に対応するために新規に必要となるテストケースである.

テスト設計，実施プロセスとしては，ASP のテストにおいて開発したテストプロセスを適用した.

また，FMP のテストケースの後ろに，拡張元となる ASP のテストケースの番号を付与した．これは，何らかの要因で ASP のテストケースに変更があった場合に，該当する FMP のテストケースも漏れなく変更できるようトレーサビリティを確保するためである.

5.2 FMP 拡張に対応するテスト抽出ポリシー

ASP のテストケースを FMP 用に拡張する際のばらつきを抑え，かつテストケース数を適切な数に抑えることを目的として, FMP 用のテストケース抽出ポリシーを作成した．本節では，ASP のテストケースを元に新規のテストケースを作成する際の指針となるよう策定した 9 種類の FMP 用テストケース抽出ポリシーについて述べる.

(1) 2 つのプロセッサを跨ぐテストケースの拡張

API を発行することで直接的に，またはオブジェクトを介して間接的に他の処理単位に影響を与える ASP の API のテストケースは, 処理単位を異なるプロセッサに割り付けるようにして FMP の ASP のテストケースとする．ここで，API 発行元の処理単位を割り付けたプロセッサを「自プロセッサ」, それ以外の処理単位を割り付けたプロセッサを「他プロセッサ」と呼ぶ.

すなわち，ASP のテストが API を発行する処理単位以外に複数の処理単位が関係する場合には，自プロセッサには API 発行元タスクのみが割り付けられ，それ以外の処理単位はすべて他プロセッサに割り付ける.

(2) ディスパッチ保留状態は他プロセッサ

(1)で拡張したテストケースの中に，「ディスパッチ保留状態のときにタスクの切り換えが発生しないこと」を確認するテストがある．以下，このテストを「ディスパッチ保留状態のテスト」と呼ぶ．このテストを実施する時には，他プロセッサを「ディスパッチ保留状態」とする.

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

この理由は，自プロセッサの状態が他プロセッサの振る舞いに影響を与えないからであり，自プロセッサの状態は「ディスパッチ保留状態」とはしない．

(3) タスクがタスクを「実行可能状態」に遷移させるテストケースの拡張

他のタスクの状態を「実行可能状態」に遷移させる ASP の API のテストケースを FMP 向けに拡張する際は，他プロセッサにおける実行状態のタスクの有無により，２種類に分離する．

この理由は，ASP では API 発行元がタスクであればそれ自身が実行状態のタスクとして必ず存在するが，FMP ではプロセッサが異なれば実行状態のタスクが存在しないケースもあるためである．

(4) 他プロセッサで実行状態のタスクが存在する場合のテストケースの拡張

ディスパッチ保留状態のテストに，「他プロセッサが CPU ロック状態」と「他プロセッサが非タスクコンテキスト実行時」を追加する．

API は一部を除いて「自プロセッサが CPU ロック状態」または「自プロセッサが非タスクコンテキスト実行時」に発行するとエラーとなる仕様のため，ASP のテストケースではエラー条件のテストとして実施している．FMP では，他プロセッサがそれらの状態の場合のテストを追加することが必要である．

(5) 他プロセッサに実行状態のタスクが存在しない場合のテストケースの拡張

ディスパッチ保留状態のテストに「他プロセッサが非タスクコンテキスト実行時」を追加する．ただし，「ディスパッチ禁止状態」と「割込み優先度マスクが全解除でない場合」はその状態に至るシーケンスが存在しないので追加しない．

(6) 「実行状態」のテストケースの拡張

API の発行元と操作対象の両方がタスクの場合には，そのタスクの状態が「実行状態」であるテストケースを追加する．

ASP では API を発行するタスクが実行状態なので，操作対象のタスクが実行状態はない．しかし FMP では，異なるプロセッサの実行状態であるタスクを操作対象とする場合がある．

(7) 「強制待ち状態[実行継続中]」のテストケースの拡張

API の操作対象がタスクの場合には，「強制待ち状態[実行継続中]」のテストを追加する．

FMP では，API 発行元タスクと操作対象タスクのどちらも，「強制待ち状態[実行継続中]」になり得るので，この状態を追加したテストが必要である．ただし操作対象のみ追加とし，API 発行元タスクに対する拡張は機械的な組み合わせが可能であるので除外する．

(8) 他プロセッサの状態のバリエーション拡張

自プロセッサの状態を操作または参照する API のテストについては，「他プロセッサ」の関連する状態を組み合わせたテストケースを追加する．

例えば，sns_loc()で自プロセッサの CPU ロック状態の取得を確認するためには，他プロセッサのロック状態と逆に組合せた次の２通りのテストケースが必要である．

・自プロセッサが CPU ロック状態で，他プロセッサが CPU ロック解除状態
・自プロセッサが CPU ロック解除状態で，他プロセッサが CPU ロック状態

これは，誤って他プロセッサの状態を取得していない事を明確にするためである．

(9) クラスの扱い

「初期割付けプロセッサ」や「割付け可能プロセッサ」などが関わるテストケースには，個別に必要なクラス定義を用意するが，特に考慮する必要がない API に関しては，用意するのは１つだけとする．

6. 設計例

この章では，ASP の act_tsk のテスト設計を元にして， 5.2 節で定めたテストケース抽出ポリシーに従い，FMP の act_tsk に対するテストを設計した事例を示す．

6.1 ASP の act_tsk の例

act_tsk の仕様の一部を統合仕様書から抜粋し，図 4 に示す．

> 対象タスクが休止状態である場合には，対象タスクに対してタスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になる．

図 4　act_tsk の仕様の一部

この仕様を確認するために作成された ASP のテストケースを図 5 に示す．テストケースはタスク間の優先度とシステムの状態によって細分化されている．

この ASP のテストケースは，FMP の１プロセッサに閉じたテストケースとしてそのまま利用される．

6.2 FMP の act_tsk のテストケース

図 5 の５つのテストケースから 5 章のポリシーに従って図 6 に示す 9 つのテストケースを追加した．なお，図 6 の(ASP:e-1)などの記述は，ASP のテストケース(e-1)から拡張されたことを表している．

FMP の act_tsk のテストケース作成手順は，次の４ステップである．

(Step 1)　ポリシー(1)を適用して「対象タスク」を API 発行元のタスクとは別のプロセッサに割り付けられていることとするために，図 5 の(e)の文章の「対象タス

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

ク」に対して，他プロセッサに割り付けられたことを表す「他プロセッサに割り付けられている」の文言を付け加え，(h)とする．

(e) 対象タスクが休止状態である場合には，対象タスクに対してタスク起動時
に行うべき初期化処理が行われ，対象タスクは実行できる状態になること．
(e-1) 対象タスクの優先度が実行状態のタスクより高い場合．
(e-1-1) 実行状態になること．
(e-1-2) ディスパッチ禁止状態の場合，実行可能状態になること．
(e-1-3) 割込み優先度マスクが全解除でない場合，実行可能状態になること．
(e-2) 対象タスクの優先度が実行状態のタスクより低い場合は，
実行可能状態となり，同じ優先度のタスクの最後につながれること．
(e-3) 対象タスクの優先度が実行状態のタスクと同じ場合は，
実行可能状態となり，同じ優先度のタスクの最後につながれること．

図 5　act_tsk の ASP テストの一部

(Step 2)　ポリシー(2) を適用して，(h)の中に「(h-1)他プロセッサで実行状態のタスクが存在する場合」を作成し，その中に(e-1)と(e-1-1)～(e-1-3)，(e-2)，(e-3)をぶら下げる形で(h-1-1), (h-1-1-1)～(h-1-1-3), (h-1-2), (h-1-3)として追加する．その際，ポリシー(3)から，事前状態の「ディスパッチ保留状態」や事後状態の「実行可能になること」が他プロセッサであることが分かるように「他プロセッサが/で」の文言を記述する．

(Step 3)　ポリシー(4) から，(h-1-1)の中に「ディスパッチ保留状態のテスト」として「(h-1-1-4)他プロセッサが CPU ロック状態」と「(h-1-1-5)他プロセッサが非タスクコンテキスト実行時」のテストを追加する．

(Step 4) ポリシー(5) から，(h)の中に「(h-2)他プロセッサで実行状態のタスクが存在しない場合」を作成する．これは(e-1)を「対象タスクの優先度がそのプロセッサの中で最も高い」と読み替えると対象タスクの振舞いは等価である．よって(h-2)の下に(e-1-1)～(e-1-3)をぶら下げる形で追加するが，この時「ディスパッチ保留状態のテスト」として「他プロセッサが非タスクコンテキスト実行時」のテストを追加し，「(e-1-2)ディスパッチ禁止状態」と「(e-1-3)割込み優先度マスクが全解除でない場合」のテストは追加しない．

6.3 実績

表 2 に代表的な API について ASP のテストケースと，そのテストケースを元に，先に述べた手順によって新規に追加された FMP のテストケースの数を示す．

タスクの起動や待ち解除を行う API に関しては，ASP のテストケースとほぼ同数のテストケースが作成された．

act_tsk と iact_tsk, または sig_sem と isig_sem のように API 発行元の処理単位がタスクと非タスクの関係にあるものは，ASP では両者の振舞いが異なるためテストケースの内容や数が異なるが, FMP では両 API とも他のプロセッサに割り付けられている処理単位やオブジェクトを操作するため基本的な振舞いに違いは無い．そのため FMP では各々で同じ内容のテストケースが追加となる．

pol_sem はセマフォの獲得をする API であるが, 待ち状態に遷移することはなく ASP のテストケース内に発行元以外のタスクが存在しないので, FMP でテストケースは増えない．

(h) 他プロセッサに割り付けられている対象タスクの状態が休止状態である場合は，対象タスクに対して
タスク起動時に行うべき初期化処理が行われ，対象タスクは実行できる状態になること．(ASP:e)
(h-1) 他プロセッサで実行状態のタスクが存在する場合．
(h-1-1) 対象タスクの優先度が他プロセッサの実行状態のタスクより高い場合．(ASP:e-1)
(h-1-1-1) 他プロセッサで実行状態になること．(ASP:e-1-1)
(h-1-1-2) 他プロセッサがディスパッチ禁止状態の場合，
他プロセッサで実行可能状態になること．(ASP:e-1-2)
(h-1-1-3) 他プロセッサが割込み優先度マスクが全解除でない場合，
他プロセッサで実行可能状態になること．(ASP:e-1-3)
(h-1-1-4) 他プロセッサが CPU ロック状態の場合，
他プロセッサで実行可能状態になること．
(h-1-1-5) 他プロセッサが非タスクコンテキスト実行時，
他プロセッサで実行可能状態になること．
(h-1-2) 対象タスクの優先度が他プロセッサで実行状態のタスクより低い場合は，
他プロセッサで同じ優先度のタスクの最後につながれること．(ASP:e-2)
(h-1-3) 対象タスクの優先度が他プロセッサで実行状態のタスクと同じ場合は，
他プロセッサで同じ優先度のタスクの最後につながれること．(ASP:e-3)
(h-2) 他プロセッサで実行状態のタスクが存在しない場合．
(h-2-1) 他プロセッサで実行状態になること．(ASP:e-1-1)
(h-2-2) 他プロセッサが非タスクコンテキスト実行時，他プロセッサで実行可能状態になること．

図 6　act_tsk の FMP 拡張したテストの一部

FMP のテストでは，ASP のテストケースの流用と FMP で追加したテストケースの

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

両方を実施する．FMP の act_tsk の API テストプログラムとして，ASP のテストケース 20 個と FMP で追加したテストケース 22 個の合計 42 個のテストを実施した際の，act_tsk と act_tsk から呼び出される関数の全 267 行のコードカバレッジを求めた．ただし，ディスパッチ処理の部分はアセンブラで書かれているため行数に含まれていない．

表 2　テストケースの数

API	ASP のテストケース数	FMP で追加したテストケース数
act_tsk	20	22
iact_tsk	20	22
can_act	23	23
sig_sem	18	22
isig_sem	20	22
wai_sem	18	9
twai_sem	24	9
pol_sem	6	0
ini_sem	13	13

ASP のテストでは 100%をカバーしたが，FMP のテストでは 1 行だけ実行されずカバレッジは 99.6%であった．図 7 に act_tsk のソースコードを示す．実行されなかった行は act_tsk の 14 行目から呼び出される関数 t_acquire_tsk_lock の中にあり，図 8 に示すソースコードの 10 行目の x_release_lock の行である．

t_acquire_tsk_lock は複数のプロセッサに割り付けられたタスクが同一のタスクを同時に操作することによる不具合を避けるための，プロセッサ間の排他処理である．図 9 の例で解説する．タスク 1 がタスク 2 を対象に act_tsk を発行し，同時にタスク 2 が自タスクを対象にプロセッサ 2 からプロセッサ 3 にマイグレートする mig_tsk を発行する．タスク 1 とタスク 2 のどちらが先にタスク 2 を操作するかは「タスク 2 のロックを取得した処理単位が操作する」というルールに従う．また，どのロックを取得する必要があるかは，PCB に入っているが，この情報はロックを取得する前は，他のプロセッサにより書き換わる可能性があるため，ロック取得後にチェックする必要がある．ロックの取得は t_acquire_tsk_lock の 7 行目の t_acquire_lock の中で行う．タスク 2 のロックをタスク 2 が取得している期間にタスク 1 がタスク 2 のロックを取得しようとする場合，タスク 1 はタスク 2 がタスク 2 のロックを開放するまで t_acquire_lock の中で待つ．タスク 2 がマイグレーションを終えてタスク 2 のロックを解放するとタスク 1 がタスク 2 のロックを取得し，t_acquire_lock から戻る．その時には既にタスク 2 はプロセッサ 2 からプロセッサ 3 にマイグレートされている．タスクのロックは割り付けられているプロセッサ毎に PCB で管理されており，タスク 1 は t_acquire_tsk_lock の 6 行目のロック取得前に保存したタスク 2 の PCB とロック取得後のそれが異なることを 8 行目で知る．そこで対象タスクがマイグレートしたと判断して取得したロックを破棄し再度ロック取得の手続きを行う．この開放を行うのが t_acquire_tsk_lock の 10 行目の x_release_lock である．

FMP のカバレッジを 100%にするには，このようなロック取得の輻輳を考慮する必要がある．しかし，現状の API テストの手法により意図的に発生させる事は困難なため「API に着目したテスト」の網羅対象から除外し，今後計画している他のテストの中で，ストレステストなどの手法にて行う事とする．

最初に ASP のテストを行い，次に ASP から拡張した FMP のテストを実施することでシングルプロセッサの API テストの経験をもとにマルチプロセッサ拡張した．その結果，テストケースの数を抑えながら合理的にテストスイートの開発を行うことが出来た．その中でソースコードのバグ 4 件と，仕様書の記述が不十分な箇所を 2 件検出して開発にフィードバックしている．

```
00 : ER
01 : act_tsk(ID tskid)
02 : {
03 :   TCB             *p_tcb;
04 :   ER              ercd;
05 :   bool_t          dspreq = false;
06 :   PCB             *p_pcb;
07 :
08 :   LOG_ACT_TSK_ENTER(tskid);
09 :   CHECK_TSKCTX_UNL();
10 :   CHECK_TSKID_SELF(tskid);
11 :
12 :   t_lock_cpu();
13 :   p_tcb = get_tcb_self(tskid, get_my_p_pcb());
14 :   p_pcb = t_acquire_tsk_lock(p_tcb);
15 :   if (TSTAT_DORMANT(p_tcb->tstat)) {
16 :       if (make_active(p_tcb)) {
17 :           dspreq = dispatch_request(p_pcb);
18 :       }
19 :       ercd = E_OK;
20 :   }
21 :   else if (!(p_tcb->actque)) {
22 :       p_tcb->actque = true;
23 :       ercd = E_OK;
24 :   }
25 :   else {
26 :       ercd = E_QOVR;
27 :   }
28 :   release_tsk_lock_and_dispatch(p_pcb, dspreq);
29 :   t_unlock_cpu();
30 :
31 :   error_exit:
32 :   LOG_ACT_TSK_LEAVE(ercd);
33 :   return(ercd);
34 : }
```

図 7　act_tsk のソースコード

Vol.2010-SLDM-144 No.10
Vol.2010-EMB-16 No.10
Vol.2010-MBL-53 No.10
Vol.2010-UBI-25 No.10
2010/3/26

```
00 : PCB*
01 : t_acquire_tsk_lock(TCB *p_tcb)
02 : {
03 :   PCB *p_pcb;
04 :
05 :   while(true) {
06 :       p_pcb = p_tcb->p_pcb;
07 :       t_acquire_lock(&(p_pcb->tsk_lock));
08 :       if (p_pcb != p_tcb->p_pcb) {
09 :                   /* 対象タスクがマイグレートした場合 */
10 :                   x_release_lock(&(p_pcb->tsk_lock));
11 :       } else {
12 :                   return(p_pcb);
13 :       }
14 :   }
15 : }
```

図 8　カバーされなかったコード(10 行目)

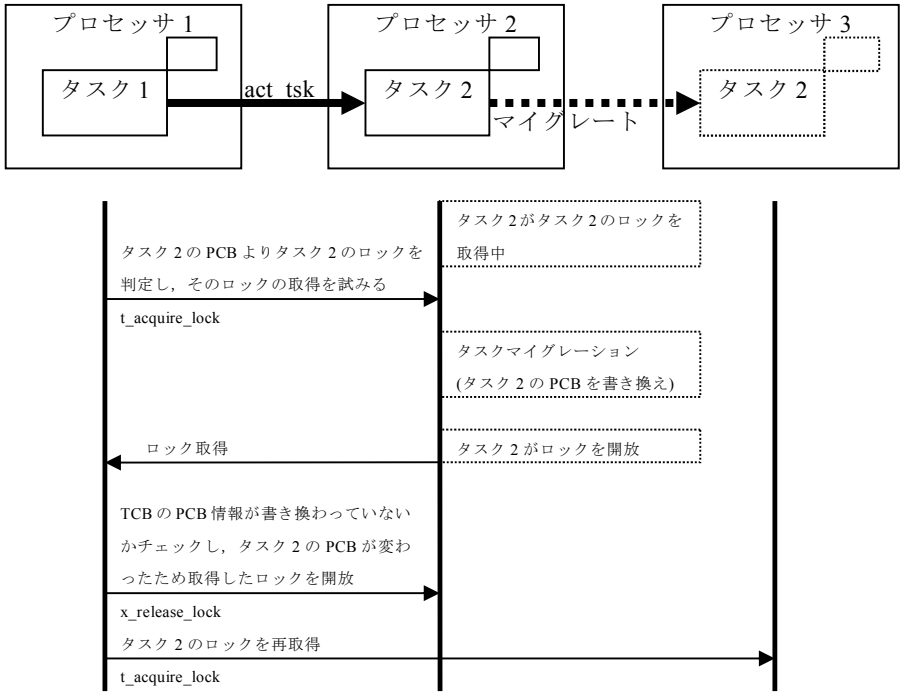

図 9　プロセッサ間の排他処理が必要なケース

7. まとめ

マルチプロセッサ対応の RTOS では，異なるプロセッサにおけるタスクの状態やリソースなどを勘案したテストが必要であるので，工夫をしなければテスト数が組合わせ爆発を起こす．これに対して我々は，対象システムの絞込みとソフトウェア構造の分析による絞込みの方針を含んだテストケース抽出ポリシーを定め，シングルプロセッサ対応の RTOS 向けのテストケースを拡張することで，一定のカバレッジを確保しつつテスト数の増大を抑制することができた．

今後，FMP に関しては，３つ以上のプロセッサを搭載するシステムでのテストも求められてくることが予想される．しかし，プロセッサ数が増大すると，それに伴い大幅にテストケース数が増えることが予想されるので，人手によるテストケースの作成だけでは，時間がかかり生産性が低下すると共に，人為的なミスが発生する危険性が高まりテスト品質が低下することが予測される．そこで今後は動的にテストケースを拡張するツール TOPPERS Test Expander(TTE)を用いたテストケース生成自動化の研究を進め，FMP のテストを通してマルチプロセッサにおける実践的なテスト手法の提案を行う．

謝辞　本研究を進めるにあたりご協力いただいた，コンソーシアム型研究の参加メンバーに深く御礼申し上げます．

参考文献

[1] TOPPERS プロジェクト http://www.toppers.jp
[2] 坂村健監修, 高田広章編, “μITRON4.0 仕様 Ver.4.02.00” トロン協会 (2004)
[3] 名古屋大学大学院情報科学研究科附属組込みシステム研究センター , http://www.nces.is.nagoya-u.ac.jp/
[4] OJL による最先端技術適応能力を持つ IT 人材育成拠点の形成, http://www.ocean.is.nagoya-u.ac.jp/
[5] 小林隆志, 沢田篤史, 山本晋一郎, 野呂昌満, 阿草清滋:On the Job Learning: 産学連携による新しいソフトウェア工学教育手法,電子情報通信学会 信学技報 SS2009-28, Vol.109, No.170, pp.95-100 (2009)
[6] TOPPERS 新世代カーネル統合仕様書, http://www.toppers.jp/docs/tech/ngki_spec-110.pdf
[7] 鴫原一人，松浦光洋，金ハンソル，金承燁，馬鋭，廉正烈，金榮柱，木村貴寿，眞弓友宏，本田晋也，山本雅基，高田広章: 組込みリアルタイム OS に対する API テストの実施，ソフトウェアテストシンポジウム 2010 予稿集，pp. 46-53 (2010)