Vol.2013-SE-180 No.10
Vol.2013-EMB-29 No.10
2013/5/28

複数のポリシ・メカニズムの学習を可能にする組込み OS の実装

茂木高宏†，　早川栄一‡

本研究では，0S の機能について，複数のポリシおよびメカニズムの差異を学習できる可読性の高い組込み 0S を実装することである．本研究 0S の特徴として，タスクスケジューリングおよび優先度逆転解消プロトコルを対象として，複数のポリシを提供し、それらが動的に切り替えられる機構を用意した．システムコールの仕様は，一般的に用いられているμITR0N4.0 を採用し，複数のメカニズムを実装した．さらに，学習者が容易に検証できるように，本研究 0S の機能を網羅したサンプルタスクセットライブラリを実装した．これにより，学習者はポリシやメカニズムの変更がタスクの動作に与える影響について，単一の環境下で検証する事が可能になった．

Implementation of an embedded operating system with multiple policies and mechanisms learning

MOTEKI TAKAHIRO†　EIICHI HAYAKAWA‡

This paper describes implementation of an operating system that has highly readable operating system to learn multiple policies and mechanisms for resource management. The OS features are utilizing a dynamically switching mechanism to each policy for task scheduling algorithms and priority inversion eliminating protocols. μITRON4.0 system call specification for implementation of multiple mechanisms is also presented. Sample task sets that learners use to verify OS functions easily are prepared. The learners can verify the modification of policies and mechanisms that incurs the impact of task operations under this environment.

1. はじめに

現在，組込み 0S の設計として，組込み 0S の機能における複数のポリシ・メカニズムが考えられている．組込みアプリケーション技術者は，実現したいアプリケーションの特性に適応したポリシ・メカニズムを搭載する組込み 0S の選択が行うことが求められている．そのため，組込み 0S における複数のポリシ・メカニズムに対する知識と動作の理解が求められている．知識と動作を理解する上での問題点として,実用的な組込み 0S のソースコードが年々巨大化している点，および効率化のため記述方式が複雑化し，ソースコードレベルからの内部構造の学習が困難である点が挙げられる．また，学習向けの組込み 0S の機能におけるポリシ・メカニズムは，一般的に単一の方式が採用されている．そのため，学習者は学習向け組込み 0S から複数のポリシ・メカニズムを比較して評価することが難しい．

本研究では，コード自体の可読性が高く，複数のポリシ・メカニズムの学習を可能にする組込み 0S の実装を目的とする．実機上での学習を想定し，本研究 0S の複数のポリシ･メカニズムを学習するためのサンプルタスクセット(学習パターン)提供する．さらに，複数のポリシ・メカニズムを適用した場合の，個々の動作の差異を比較学習できる機構を 0S に用意する．これにより，学習者はポリシやメカニズムの変更がタスクの動作に与える影響について，単一の環境下で学習する事が可能となる．

†拓殖大学大学院工学研究科電子情報工学専攻
‡拓殖大学工学部情報工学科

2. 問題分析

従来の組込み 0S 上で,資源管理ポリシやメカニズムを比較評価する場合の具体的な問題は次の通りである．

2.1 組込み 0S 内部構造の複雑化による学習性低下

実用的な組込み 0S では,効率性を重視した実装やスケーラビリティの配慮がされているため，0S の内部構造は複雑化している．例えば，高速なコンテキストスイッチングを実現する機能である多重割込みや，多種多様なプロセッサアーキテクチャに対応するためのコンフィギュレーションが挙げられる．特に，0S をコードリーディングに用いる場合，効率性やスケーラビリティ重視した機構は不要と考えられる．

2.2 複数のポリシ・メカニズムに起因する問題

複数のポリシの問題点として，組込み 0S では，メカニズムからポリシの完全分離が困難である．そのため，ポリシに対する柔軟性が低下し，複数のポリシにおける動作学習が難しい．さらに，これら複数のポリシにおける動作の違いの学習も困難である．

複数のメカニズムの問題点として，メカニズムは組込み 0S の特徴に応じて変化する点が挙げられる．よって，単一環境下から,組込み 0S の特徴を無視した複数のメカニズムの学習が難しい．

2.3 学習パターンに関する問題

0S のポリシ・メカニズムの動作を学習するためには，0S の上で動くサンプルタスクセットが必要である．サンプル

タスクセットの提供はスケジューラの理解に大きく影響する.

2.4 関連研究

(1) Nachos

Nachos[1]はPC上でアプリケーションとして動作する学習向けのOSである. C++言語で記述され, MIPSやx86上のLinuxで動作する. Nachosアプリケーションは, OS内部のシュミレータによって動作するため,各種デバイス(センサやLEDなど)を視野に入れた学習はできない.学習としては,学習者に不完全な実装やインターフェースを提供し, その内部を開発させる事を想定している.また,複数のポリシ・メカニズムも実装されていない.

(2) kozos

Kozos[2]は実機上で動作し, ミドルウェアを含んだ学習向けの組込みOSである. C言語とアセンブリ言語, リンカスクリプトで記述され, H8やARMアーキテクチャ上で動作する.学習としては,タスクセットとカーネルを並行して, 開発させる事を想定している. しかし, 序盤からカーネルの開発に入るため, 学習者には敷居が高い. ソースコードリーディングでの学習も視野に入れているが, コメントが少なく, ミドルウェアを含んでいるため, OS内部構造が複雑化している. また, kozosは, OSの機能を拡大していく方向性のため, 各機能において複数のポリシ・メカニズムは実装されていない

3. 研究方針

研究方針は, 学習方針, および組込みOS設計方針, 組込みOS記述方針とし, 次の3.1～3.3に示す.

3.1 学習方針

本研究OSの動作は各種デバイス(センサやLEDなど)の学習を視野に入れるため,実機で動作させる.カリキュラムの構成としては,簡易なアーキテクチャであるH8を基本編とし, より実用的なアーキテクチャとしてARMプラットフォームを用いる. また, 学習教材として, 動作学習のための学習パターンが格納されたサンプルタスクセットライブラリ,および各種ドキュメントを提供する事とした.さらに, シリアルコンソールを使用した詳細的な学習方法と, 早川研究室で開発されたGUIベースの可視化ツール[7][8]を使用する事にした. これらを使用した学習の流れをカリキュラムとして, 提供する(図1).

図1のカリキュラムでは, H8アーキテクチャ上で(1)～(5)を学習し,次にARMアーキテクチャ上で同様に(1)～(5)を学習する.

(1) サンプルタスクセットの動作学習

操作ドキュメントに従い, サンプルタスクセットライブラリの学習パターンを動作させ, 各タスクの応答を可視化ツール, およびシリアルコンソールで学習する.

(2) ポリシ切り替えによるサンプルタスクセット動作学習

操作ドキュメントに従い, サンプルタスクセットライブラリの学習パターンに異なるポリシを適用する. これにより, ポリシの変更がサンプルタスクセットの動作に与える影響について学習可能となる.

(3) サンプルタスクセット新規開発

サンプルタスクセット新規開発支援ドキュメントに従い, 学習者はサンプルタスクセットの開発を行なう.

(4) OSソースコードリーディング

可読支援ドキュメントに従い, およびソースコードを用いて, OSのソースコードリーディングを行なう.

(5) ポリシ開発

ポリシ開発支援ドキュメントに従い, カーネルにポリシの追加開発を行なう.

Vol.2013-SE-180 No.10
Vol.2013-EMB-29 No.10
2013/5/28

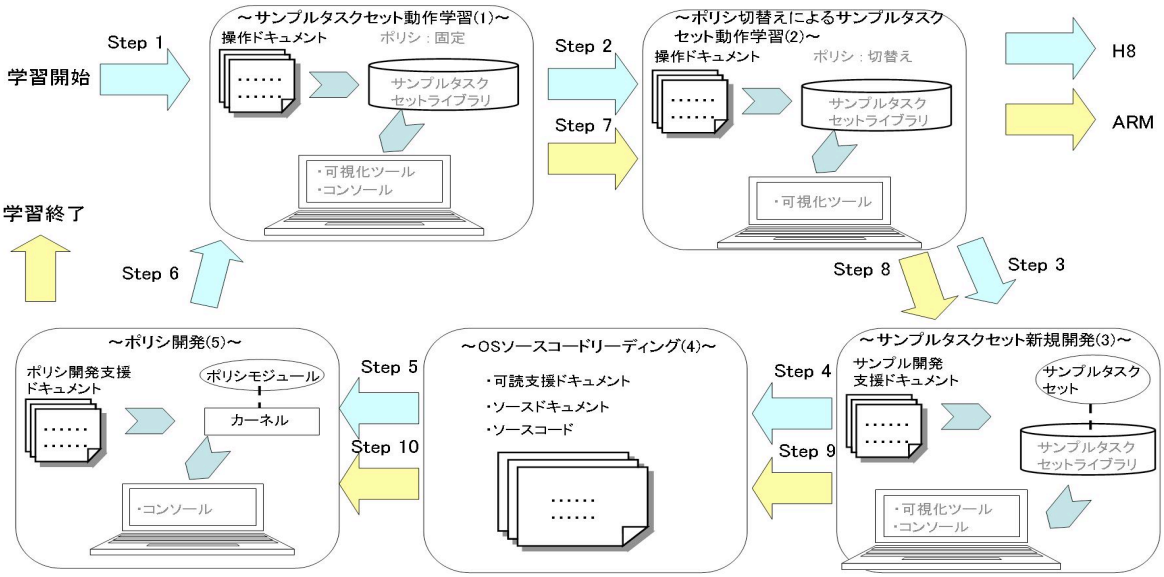

図1　学習カリキュラム

3.2　組込み OS 設計方針

本研究 OS は,実装対象とする複数のメカニズムにシステムコールの方式，複数のポリシにタスクスケジューリングと優先度逆転防止プロトコルを採用する事にした.さらに,複数のポリシ・メカニズムを適用した場合の動作を容易に切り替えて実行できる機構としてポリシ動的切り替え機構を実装する.

OS は割込みドリブン型のモノリシックカーネルとし,メモリ効率性の配慮は行わない．さらに,組込み OS としての基本的な機能を提供する．組込み OS の基本的な機能とは,同期と排他，タスク間通信と時間管理機能である．これらは，広く採用されている μITRON4.0 仕様[3]を参考にした.

また,複数のポリシ・メカニズム,および本研究 OS の機能を学習するためのサンプル(学習パターン)をタスクセットとして実装し，ライブラリとして提供する事にした.

3.3　組込み OS のコード実装方針

本研究 OS の記述は，C 言語，アセンブラ，リンカスクリプトに限定する事とした.OS ソースコード可読性を考慮し,コンフィギュレーションやそれを用いた静的 API を利用せず，すべて μITRON4.0 仕様[3]の動的 API とする．なお,H8 アーキテクチャと ARM アーキテクチャ対応は，アーキテクチャに応じて組込み OS ソースコードと実行イメージを提供する.

さらに，MISRA C2004[4]を参考にコーディングを行い,コメント比率とアセンブリ比率を減少させた．アセンブリ比率の減少に関しては,極力アセンブリでしか記述できない箇所だけに留めた.

また,組込み OS ソースコード可読性,および管理性を高めるために doxygen[5]，graphviz[6]対応コーディングを行う事にした．これにより，コードの可視化とコードドキュメントを自動的に生成することが可能となった.

4. 組込みシステム全体構成

組込みシステムの全体構成を図 2 にまとめた.図 2 にある各部を 4.1～4.5 で述べる.

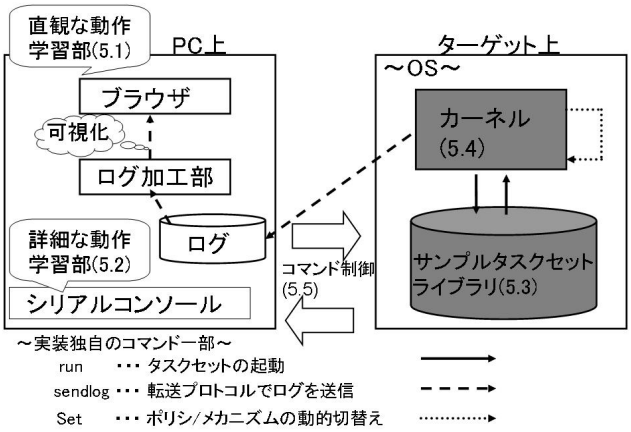

図2　組込みシステム全体構成

4.1　直観的な動作の学習

直感的な学習のサポートについて，早川研究室で開発されたブラウザ上で動く GUI ベースの組込み OS 可視化ツールを利用する．組込み OS の可視化にあたって,各種ログを取得し,ログ加工部でフォーマット変換を行う．組込み OS の可視化ツールは,タスクスケジューリング可視化ツール[7]とメモリ管理機構可視化ツール[8]を用いる.

(1)　タスクスケジューリング可視化ツール

このツールは，タスクスケジューリングに限定してタスクの動作状況の様子を可視化する．主に，複数のタスクにおいて，タスクの状態一覧(実行状態や実行可能状態,待ち状態等)と状態遷移，遷移時間の把握が可能である.タスクスケジューリング可視化ツールを図 3 に示す.

Vol.2013-SE-180 No.10
Vol.2013-EMB-29 No.10
2013/5/28

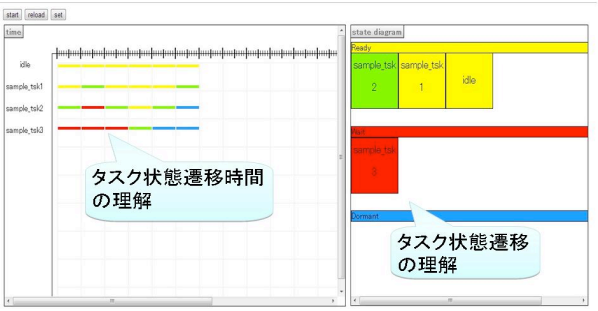

図3　タスクスケジューリング可視化ツール

これにより，組込み OS の基礎であるタスク管理，および複数あるタスクの動作状況，タスクスケジューリングがタスクに与える動作を直観的に監視できる．

(2)　メモリ管理機構可視化ツール

このツールは，カーネルレベル，およびタスクレベルに限定してメモリ利用状況の様子を可視化する．主に，複数のタスクにおいて，タスクの生成時間とタスクの終了時間，メモリの占有時間と使用メモリアドレスの把握が可能である．また，有効化しれているメモリの種類についても把握できる．メモリ管理機構可視化ツールを図 4 に示す．

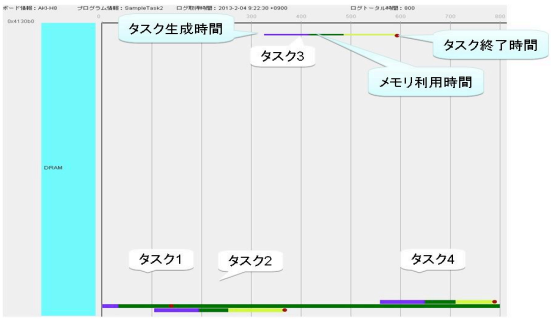

図4　メモリ管理機構可視化ツール

これにより，カーネル，およびタスクにおけるメモリ取得と解放を行なうシステムコールの呼び出しによるメモリ利用を可視化し，学習者がメモリ利用状況を直観的に監視できる．

4.2　詳細的な動作の学習

本研究 OS のサンプルタスクセットライブラリのタスクソースコードと，ターゲット上で動作させた時のタスクの出力を確認する(図 5)．

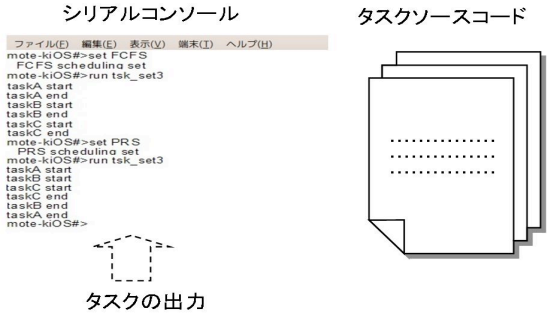

図5　詳細的な動作の学習

この学習では，複数のポリシにおけるタスク切り替え事象，μITRON4.0 仕様[3]の各種 API の学習が可能である．

4.3　サンプルタスクセットライブラリ

サンプルタスクセットライブラリは，複数のポリシ・メカニズムの学習，および本研究 OS の機能を網羅したもので，60 個のサンプルタスクセット(学習パターン)を実装した．なお，学習者がサンプルタスクセットを新規開発する場合を想定して，コンポーネントとして実装を行なった．実装の一部を表 1 に示す．表 1 のサンプルタスクセットは，時間管理，同期・排他，優先度逆転防止プロトコルでの作用から抜粋した．

表1　サンプルタスクセットライブラリー部

タスク数	環境下	シナリオ
…	…	…省略
3	–	周期ハンドラを使用した周期タスク制御
1	–	周期ハンドラを使用したポーリングチェック
…	…	…省略
2	–	セマフォを使用した同期
3	–	セマフォを使用したクレジットトラック同期
2	–	セマフォを使用した排他
3	–	セマフォを使用したゼネラル排他
…	…	…省略
3	priority ceiling protocol	デッドロックの 予防
3		優先度逆転現象
3	priority inheritance protocol	デッドロックの 誘発
3	priority inheritance protocol	ネストブロッキング現象(推移的優先度継承)
3	priority inheritance protocol	連続ブロッキング現象
3		ロックフリープロトコルを使用した優先度逆転防止
…	…	…省略

4.4　カーネル全体構成

機能からみたカーネルの全体構成を図 6 にまとめた．

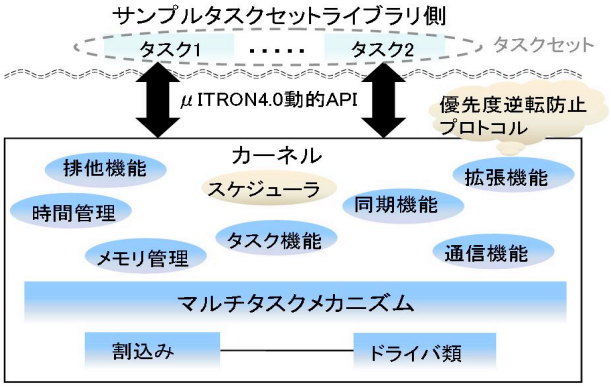

図6　カーネルの機能

カーネルには，複数のポリシ・メカニズムとマルチタスクキングのため，組込み OS の基本的な機能を提供する．組込み OS の基本的な機能とは，同期，および排他に使用するセマフォ，完全排他に使用する mutex，タスク間通信に使

Vol.2013-SE-180 No.10
Vol.2013-EMB-29 No.10
2013/5/28

用するメールボックス，時間管理に使用するアラームハンドラと周期ハンドラである．これらは，広く採用されているμITRON4.0 仕様[3]を参考にした．これらの組込み OS の基本的な機能の生成には，静的 API は用いず，すべて動的 API とした．

4.5 コマンド制御

図 2 のホスト PC とターゲットは，実装独自のコマンドで制御する．操作に用いるコマンドの概要を(1)～(3)に示す．

(1) run コマンド

シリアルコンソール上から，run コマンドでサンプルタスクセットを選択すると，カーネルが指定されたサンプルタスクセットを起動し，タスクの出力をシリアルコンソール上に返却する．

(2) sendlog コマンド

シリアルコンソール上から，sendlog コマンド選択すると，カーネルはログを転送プロトコルで PC 上に送信し，ログファーマット変換を行なう．このログは，直観な学習を行う可視化ツールであるタスクスケジューリング可視化ツール(図 3)とメモリ管理機構可視化ツール(図 4)で使用する．

(3) set コマンド

シリアルコンソール上から，set コマンドでポリシを選択すると，カーネルが指定されたポリシの動的切り替えを行い，それに補って依存するカーネルオブジェクト整合性管理を行なう．動的に切り替え可能なポリシは，タスクスケジューリングである．なお，各種タスクスケジューリングに必要となるスケジューリングパラメータは，set コマンド時に入力する方式とした．

5. 実装した複数のポリシ・メカニズム

5.1 システムコールの方式

システムコールの方式を 2 通りの方法で実装した．具体的には,トラップ命令を使用して OS の特権モードを切り替える方式とトラップを使用しないサブルーチンコールの方式である．これらの二つの方法を実装した理由は，プロセッサモード切り替えやパイプラインフラッシュ等で生じるオーバーヘッドの比較ができる点にある．

5.2 タスクスケジューリング

タスクスケジューリングを 13 種実装した.これらのタスクスケジューリングは，実用 OS，および学習 OS のコードを調査し，広く採用されているものを対象とした．また，比較対象用をして，リアルタイム性を意識した組込み OS 等で使用されるスケジューリング，また比較評価を可能にするために公平性を意識した汎用系 OS 等で使用されるスケジューリングも実装を行なった．これらの関係を図 7 に示す．

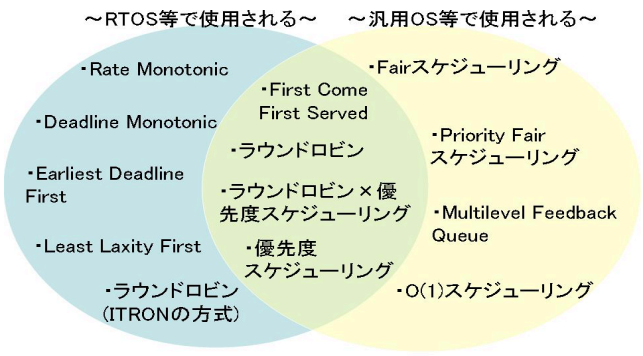

図7 実装対象としたタスクスケジューリング

タスクスケジューリングはタスクのレディー管理データ構造と密接に関わってくる．図 7 にあるタスクスケジューリングに沿って，5 種のレディー管理データ構造(単一のレディーキュー，優先度レベルのレディーキュー，優先度レベルの run キューと expired キュー，ツリー型のレディー，優先度レベルのツリー型レディー)の実装を行なった．

(1) RTOS 等で使用されるスケジューリング

RTOS 等で使用されるスケジューリングでは，デッドライン保障としてスケジューリング属性がある．具体的には，Guarantee でタスク生成時,Guarantee でスケジューリング時，Best Effect 型，Robust 型等がある．本研究 OS は学習向けの組込み OS のため，内部構造が簡素化する Guarantee でタスク生成時，および Best Effect 型を採用した．

(2) 汎用 OS 等で使用されるスケジューリング

汎用 OS 等で使用されるスケジューリングでは，CPU バウンドタスクと I/O バウンドタスクの識別に，ヒューリスティックを用いた発見アルゴリズム等が実装される事がある．スケジューラを簡素化するため,これらは本研究 OS では採用していない．

5.3 優先度逆転防止プロトコル

リアルタイム処理で重要視される優先度逆転防止プロトコルを 6 種実装した．優先度逆転防止プロトコルは静的型プロトコルと動的型プロトコルを対象とし，mutex の属性として実装する．なお，静的型と動的型の差異は，各種優先度逆転防止プロトコルに必要となるパラメータの有無である．

(1) 静的型の優先度逆転防止プロトコル

・Priority Ceiling Protocol(以下 PCP)
・Immediate Highest Locker Protocol(以下 I-HLP)
・Delay Highest Locker Protocol(以下 D-HLP)

(2) 動的型の優先度逆転防止プロトコル

・Priority Inheritance Protocol(以下 PIP)
・Stack Resource Policy(以下 SRP)
・Virtual Priority Inheritance(以下 VPI)

5.4 複数のポリシがタスクに与える影響

複数のポリシがタスクに与える影響を図 8 に示す．本研究 OS は 6 種の優先度逆転防止プロトコルを実装し,それぞ

れの優先度逆転プロトコルに対して発生する三つの主作用(デッドロックの誘発現象，推移的優先度継承現象，連続ブロッキング現象)と一つの副作用(デッドロックの予防)をサンプルタスクセットライブラリで学習パターンとして提供している．さらに，これらは本研究 OS に実装した 13 種のタスクスケジューリング環境下と，μITRON4.0 仕様[3]にある待ちタスクの ready 返却アルゴリズム(FIFO 順と優先度順の 2 通り)によって，タスクの出力が変化する．これらの組合せすべて求めると，学習パターンであるサンプルタスクセット 1 つに対して，156 通りのタスクの出力となる．これらのタスクの出力は，本研究 OS のポリシ動的切り替え機構を用いる事で実現できる．これらの関係を図 8 にまとめた．

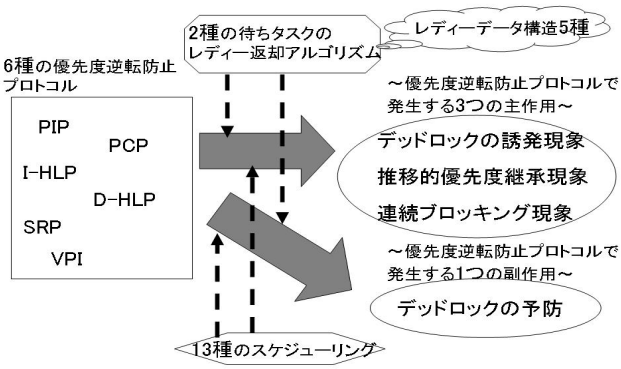

図8 複数のポリシがタスクに与える影響

6. タスクスケジューリング動的切り替え

本研究 OS は，メモリへ OS をリロードせずに，タスクスケジューリング自体を動的に切り替えていくことが可能である．切り替えは，タスクスケジューリング切り替えシステムコールで行う方式とした．タスクスケジューリング切り替えシステムコールは，コマンド応答専用の init タスク(サンプルタスクセット起動前，または終了後に処理が移るタスク)，およびユーザから任意のタイミング(ユーザタスク)で発行する事が可能である．これにより，同一のタスクセットに対して異なるタスクスケジューリング適用時のタスクの出力を体験する事ができる．

(1) init タスクからタスクスケジューリングの切り替え

本研究 OS で，set コマンド時の init タスクから異なるスケジューリングの切り替え方法の一例(FCFS スケジューリングから優先度スケジューリングへ切り替え)を図 9 に示す．

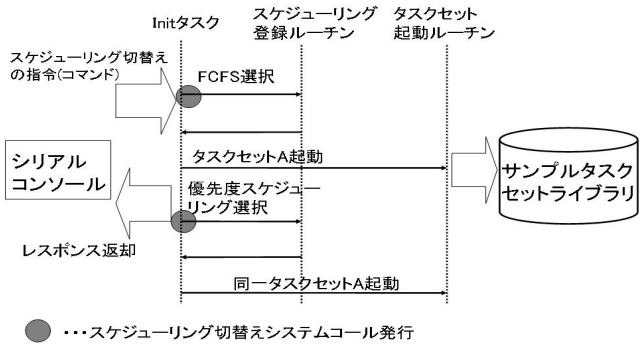

図9 異なるスケジューリングの切替え方法

上図の例は，ユーザからシリアルコンソールを通して，スケジューリング切り替えのコマンドが入力されると，init タスクからタスクスケジューリング切り替えシステムコールが発行される．そして，同一タスクセットに対して異なるタスクスケジューリングを適用している．なお，スケジューリング登録ルーチンは，要求されたスケジューリングを OS 内に登録，およびスケジューリングに必要となるパラメータチェックと管理を行なう．

図 9 の FCFS と優先度スケジューリング切り替えで，取得できるタスクの出力とサンプルタスクセットの簡易な例を図 10 に示す．

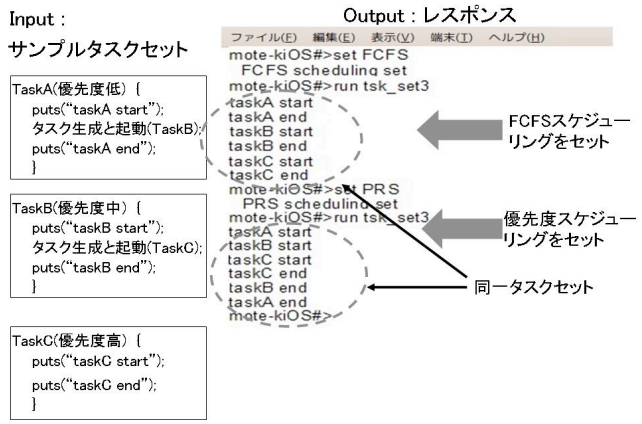

図10 異なるスケジューリング適応のレスポンス

(2) ユーザタスクからのスケジューリング切り替え

ユーザタスクからスケジューリングを切り替える場合は，(1)の図 8 に加えて，タスク実行可能状態を保持するレディー管理データ構造の切り替え，または OS 内部で各タスクの状態を保持している TCB の総移行処理を行なう．

7. 評価

本研究の目的の一つである OS 自体の可読性の向上に関して，他 OS と比較して評価を行う．.

7.1 他 OS との可読性比較調査結果

本研究 OS と他 OS との可読性比較調査の結果を下表 2 に記す．他 OS では，実機上で動く OS である kozos[2]と H8/OS[9]とした．なお，アセンブリに関しては，アセンブリでしか記述できない箇所とし，最小限に抑えた．

Vol.2013-SE-180 No.10
Vol.2013-EMB-29 No.10
2013/5/28

表2 他OSとの可読性比較調査結果

	ソースコード行（コメント比率）	アセンブラ行（アセンブラ比率）
本研究OS(H8)	28537(28%)	140(1%)
本研究OS(ARM)	30008(30%)	212(1%)
kozos[2]	6728(15%)	205(3%)
H8/OS[9]	14969(2.8%)	269(1.8%)

①goto文	②条件付きマクロ	③関数マクロ
0	0	2
0	0	6
9	25	6
0	269	31

表2の①～③はMISRA C2004[4]におけるルールの一部である．MISRA C2004 の必須ルールから①を，推奨ルールから②と③を選出した．

表2より，本研究OSはコメントが多く，MISRA C2004の対応により，可読性が向上したと考えられる．

8. まとめ

本研究では，コード自体の可読性が高く，複数のポリシ・メカニズムの学習を可能にする組込みOSの実装を行った．複数のCPUアーキテクチャで動作し，複数のタスクスケジューリングと優先度逆転防止プロトコルに対してタスクの出力情報を比較してタスクに与える影響について単一環境下から学習する事が可能となった．さらに，OSの可読性を向上させる事ができた．

今後の課題として，複数のポリシ・メカニズムに対する動作の差異学習を向上させる機能を提案し，モニタプログラムとして組込みOSから切り離す．さらに，実機上での動作を活かし，外部からサンプルタスクセットに影響を与える事が可能な専用のハードウェアを開発する．

謝辞

本システムの実装にあたってご協力頂いた拓殖大学大学院工学研究科 金森一真氏に感謝する．

参考文献

[1] Wayne A.Christopher, StevenJ.Procter, Thomas E.Anderson, ″The Nachos Instruction Operating System, ″USENIX

[2] 坂井弘亮, kozosプロジェクト, http://kozos.jp/kozos/index.html

[3] μITRON ver4.02.0仕様書, http://www.ertl.jp/ITRON/SPEC/mitron4-j.html

[4] MISRA－C 研究会: 組込み開発者におくる MISRA－C:2004 ―C 言語利用の高信頼化ガイド, 日本規格協会(2006)

[5] Dimitri van Heesch, Doxygen, http://www.stack.nl/~dimitri/doxygen/

[6] AT&T, graphviz, http://www.graphviz.org/

[7] 中川裕貴, Praween Amontamavut, 早川栄一, Androidにおける OS 可視化環境の開発,情報処理学会第74回全国大会，3K-4

[8] 金森一真, 茂木高宏, 早川栄一, メモリ管理可視化ツールの開発, 情報処理学会第75回全国大会,

[9] みついわゆきお, H8/OSver3.51, http://mes.sourceforge.jp/h8/index-j.html