Vol.2012-OS-120 No.3
2012/2/28

ヘテロジニアスマルチコアを統合管理するオペレーティングシステム構成法の検討

中川　岳[†1]　追川修一[†1]

ヘテロジニアスマルチコアアーキテクチャにはプロセッサ性能向上をクロック周波数を抑えながら実現することができるという特徴があり，処理性能の高さと低電力消費を同時に要求される組み込みシステムにおいても研究開発が進んでいる．また，組み込みシステムはその利用領域によって高い信頼性が求められることがあり，従来のC 言語中心ではなく，現代的なオブジェクト指向言語を取り入れた開発手法の検討が必要である．

以上の背景を踏まえ，本研究では，ヘテロジニアスマルチコアアーキテクチャの効率的な管理を目的として，Java で実装した OS と，それが動作する新たなシステムモデルを提案する．そして，その実現に向けた課題の発見や知見を得るために，Java 仮想マシンで動作する OS である Nachos 5.0j のヘテロジニアス化を行った．結果として，ARM アーキテクチャと MIPS アーキテクチャのヘテロジニアスマルチコアアーキテクチャ環境を管理するオペレーティングシステムをシミュレータ上で動作させることができた．本稿では，そのモデルの提案と実装について述べる．

Consideration on Operating System for Heterogeneous Multi-Core Architecuture

Gaku NAKAGAWA[†1] and Shuichi OIKAWA[†1]

Heterogeneous Multi-Core Architecture have a feature that step up processor performance with low increase in the clock frequency. That feature is useful for today's embedded systems, because they are sometimes required both high performance and low power consumption at the same time. So, the use of Heterogeneous Multi-Core Architecture for embedded system is increasing. In this study, we propose a efficient operating system model for heterogeneous multi-core architecture. And we implement a simulator of operating system for Heterogeneous Environment to develop our model. This paper describe ourmake model and implementation.

1. はじめに

近年，組み込み機器への性能要求が高まっている．家電やスマートフォンなどの機器は高性能化が進んでおり，その中心となる組み込み向けアーキテクチャへの要求はとどまるところを知らない．また，組み込み機器の中にはバッテリーで動作するものや，長時間運用されるものも多く，高い処理能力と同時に低消費電力を求められることも多い．さらに，組み込みシステムには高い信頼性が要求されることも多い．組み込みシステムの応用範囲は多岐にわたっており，システムの対象によっては，利用者の生命や財産の安全に関わることもある．そのため，システムを制御するオペレーティングシステム (OS) には高い信頼性が要求される．

近年，プロセッサの性能向上の新たな手法として，異種のプロセッサコアを複数搭載したヘテロジニアスマルチコアアーキテクチャの研究が盛んになってきている．これは，それぞれ目的特化したプロセッサコアを組み合わせ，処理に応じて選択的にタスクを分散し，プロセッサ全体の処理性能向上を図るものである．これまでのプロセッサ性能向上手法は，クロック周波数の向上，命令パイプラインの工夫，複数の同種コアによる並列処理を中心としていたが，CPU のヘテロジニアス化はこれらの手法と違う新たな手法として，研究開発が進められている．また，ヘテロジニアスマルチコアアーキテクチャでは，それぞれのプロセッサコアが得意な仕事を分担することで，コアごとのクロック周波数を抑えることが可能であり，これはプロセッサの省電力につながる．つまり，プロセッサの処理能力向上を，低消費電力で行うことができる．この点から，組み込みシステムにおいても，ヘテロジニアスマルチコアアーキテクチャが採られる事例が増えている．具体例としては，ARM アーキテクチャと DSP コアのヘテロジニアスマルチコアアーキテクチャである，Texas Instruments 社の OMAP シリーズが挙げられる．以上の背景より今後，組み込みシステムでのヘテロジニアスマルチコアアーキテクチャの採用の増加とそれを管理する OS を含めたシステムへの信頼性要求の高まりが予想される．

ヘテロジニアスマルチコアアーキテクチャのシステム管理には，コア間で使用できる命令セットやメモリ管理機構が異なるため，同種のコアを複数搭載したホモジニアスマルチコアアーキテクチャの管理とは異なった手法が必要である．ヘテロジニアス環境を管理する手法

†1 筑波大学
University of Tsukuba

© 2012 Information Processing Society of Japan

としては，それぞれのコアで OS を動作させ，共有メモリを利用した OS 間通信を利用するものがある．しかしながら，この手法では，それぞれのコアで OS が独立して動作するため，本質的に同じ処理がコア間で重複して行われ, リソースを無駄にしてしまうという欠点がある．

また，先に挙げた高信頼性への要求を満たすためには，OS の実装法として従来の C 言語中心の OS 開発ではなく，他の言語を積極的に組み合わせた開発を検討する必要がある．C 言語はハードウェアやメモリに対する操作が容易である一方，型に起因するエラーやメモリリークが起こりやすいなど，プログラムの信頼性を確保する上での問題点を抱えている．これに対し，現代的なオブジェクト指向言語の中には，ハードウェアや低レベルのメモリ操作は貧弱であるが，強い型検査や，強力なメモリ管理機構を備えているものがあり，C 言語による開発と組み合わせることにより，より信頼性を高めた OS を開発できる可能性がある．

C 言語以外での OS の実装法としては，過去の研究で様々な方式が提案されてきた．その中でも，カーネル, システムプログラム, デバイスドライバなどの OS の中核を成すコンポーネントを，C 言語, アセンブリ言語ではなく，Java で開発するという手法は様々な試みが行われており, 先行事例としては oJNode[1), Jx[4) などが存在する．

OS を Java で実装する利点としては，C 言語で実装する場合に比べて，次の点が挙げられる．

- Java は C 言語より強い型付けを持ち，より型安全である．これにより，データ型に起因する瑕疵の減少が期待できる．
- Java 仮想マシンにはガーベッジコレクタが搭載されており，使用されないメモリ領域を自動的に解放することができる．これによりメモリリークが起こる可能性を減少させることができる．
- オブジェクト指向言語である Java には継承の概念があり，実装完了後の機能追加や仕様変更に柔軟に対応できる．これはシステムの保守性向上につながる．

以上の背景を踏まえ本研究では，ヘテロジニアスマルチコアプロセッサの管理を目的として，Java で実装した OS とそれが動作するシステムモデルを提案し，その具体化と検討課題の発見のため．Java で実装された OS である Nachos 5.0j (Nachos-j) に ARM アーキテクチャシミュレータを実装し，既に実装されている MIPS アーキテクチャシミュレータとの並行動作を実現する．以下，本稿ではこれをヘテロジニアス化と表す.

本稿の構成は以下の通りである．まず第 2 章にて，提案するシステムについて述べる，第 3 では，ヘテロジニアス化の対象である Nachos-j について述べる．第 4 章では，ヘテロジニアス化するターゲットである ARM アーキテクチャについて述べる．第 5 章では，本研究にて Nachos-j に加えた変更点について述べる．第 6 章では動作確認について述べ， 第 7 で関連研究に触れ，第 8 章で結論を述べる．

2. 提案するシステムモデル

本研究の狙いは，主に次に挙げる 3 つを実現したヘテロジニアスマルチコアアーキテクチャを実現することである．

(1) システム管理タスクを特定のコアに一元化すること
(2) コア間で主記憶を共有すること
(3) 主に管理を担当する OS の実装には Java を用いること

(1)〜(3) を目的とする理由は以下の通りである．第 1 章で取り上げたように，ヘテロジニアスマルチコアアーキテクチャの管理手法として，コアごとにそれぞれ OS を動作させ，プロセッサ間通信によりシステム管理を行うものがある．これはそれぞれのアーキテクチャ向けに実装されている既存の OS にプロセッサ間通信を実装すれば実現することができるが，複数のプロセッサ間で OS の機能が重複してしまう．これを解消するために，特定のコアへのシステム管理タスクの一元化を実現し，それぞれのコアがシステム管理タスクではない，本質的なタスク (アプリケーションタスク) に集中できるようにする．また，コアごとに独立した OS を動かすには，それぞれのコアにある程度の主記憶を接続しなければならず，実装面積の増大という点で，無駄が生じてしまう．(1) の管理タスクの一元化と併せて (2) の主記憶の共有により，この問題を解消する．(3) の Java による実装については，第 1 章でも触れたように，より信頼性の高いシステムにするためである．特に，ガーベッジコレクタによるメモリリークの防止は，長時間運用されるケースが多い組み込みシステムにおいては，特に有用であると考えられる．以上をふまえ，本研究が提案するシステムモデルの概要を図 1 に示す.

このモデルは，Master コア，Slave コア，Communication Memory, Shared Main Memory から成っている．それぞれのコアは，Communication Memory と Shared Main Memory を介して接続されている．Master コアでは，タスク管理，I/O 管理，メモリ管理などコア間で一元化できるタスクを行う OS (Master OS) が実行される．Master OS は Java で実装され，Master コアではアプリケーションタスクは実行しない．図 1 では，x86 アーキテクチャコアで必要最小限の機能を持った OS (Micro OS) や Java Virtual Machine (JVM) が動作，その上で Master OS が動作することを想定しているが，コアのアーキテクチャは

© 2012 Information Processing Society of Japan

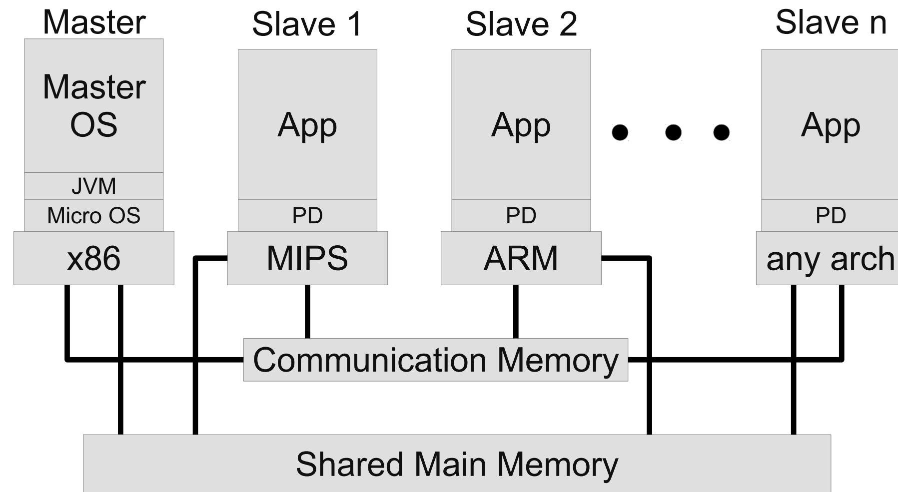

図 1 提案するシステム

x86 に限定するのではなく，他の汎用プロセッサコアや，Java プログラムの実行に特化したコア[11] を使うことも想定される．Slave コアでは最小限のシステム管理タスクを持った Processor Driver (PD) が動作し，その制御の下，アプリケーションタスクが動作する．PD は Communication Memory を経由して，Master OS とメッセージを送受信し，その制御を受ける．主記憶については，Shared Main Memory を共有する．Shared Main Memory は相互排除により主記憶へのアクセス競合時にその整合性を保つ．

3. Nachos Operating System

Nachos-j は，University of California, Berkeley にて OS およびシステムプログラムの授業教材として開発されたものであり，C++言語で記述された教育用 OS である Nachos[3] を Java Virtual Machine (JVM) で動作するように移植したものである．現在では同校だけに留まらず，広く OS やシステムプログラムの学習題材として採用されている．以下，本稿では Nachos と表記した場合，Nachos-j を指すものとする．Nachos はホスト OS 上で，JVM により 1 つのプロセスとして実行され，コンソール，タイマなどを持った簡単な計算機をシミュレートし，その環境下にてカーネルが動作する．また，MIPS シミュレータを搭載しており，COFF 形式で記録された MIPS アーキテクチャのプログラムを読み込み，実行することができる．以下では Nachos の概要について述べる．

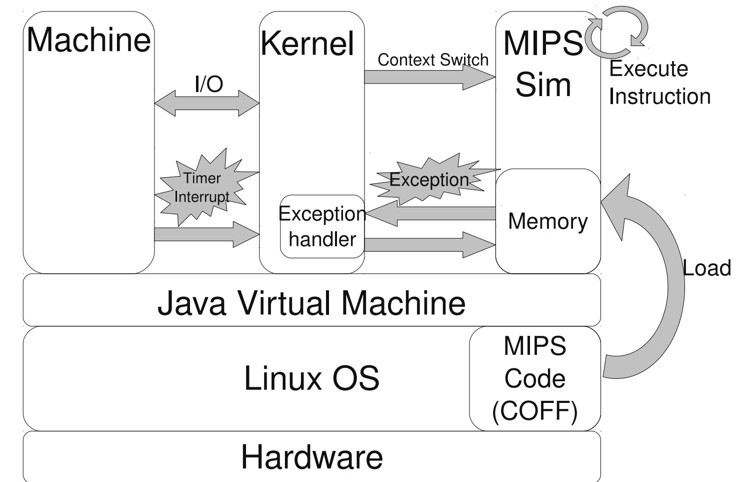

図 2 動作の概要

Nachos は機能の面から，Machine, Kernel, MIPS Sim に分けることができる．Machine は入出力やハードウェアシミュレーションを担当し，Kernel が OS の本体である．MIPS Sim は Machine, Kernel 経由でホスト OS 上の MIPS アーキテクチャのプログラムを読み込み実行する．シミュレータ上でシステムコールなどの例外が発生した場合，MIPS Sim は動作を一時停止し，制御は Kernel に遷移する．Kernel は例外に応じた処理を行い，結果を返して MIPS Sim の実行が再開される．Machine は定期的に Kernel に対してタイマ割り込みをかけており，Kernel はその割り込みを利用して，MIPS Sim に対してコンテキストスイッチさせ，マルチタスクを実現している (図 2 を参照) これらの機能は，それぞれ Java のスレッド機能を応用した独自の機構によって実現されている．

MIPS Sim は MIPS R3000 の動作をシミュレートし，32bit MIPS アーキテクチャの命令を実行することができる．MIPS Sim は，MIPS の完全なシミュレートを実現しておらず，実際の R3000 プロセッサとの間に以下のような相違点がある．

- 浮動小数点演算装置 (FPU) がサポートされていない
- プロセッサモードの区別がない（特権モードが無い）

© 2012 Information Processing Society of Japan

表 1 Nachos のシステムコール一覧

番号	システムコール名	説明
0	Halt	Nachos を停止する.
1	Exit	現在実行中のプロセスを終了させる.
2	Exec	指定したファイルに記録されているプログラムを実行する.
3	Join	現在のプロセスを指定のプロセスが終了するまで待機する.
4	Creat	指定された名前のファイルを作成し，ファイル識別子を返す.
5	Open	指定の名前のファイルを開き，ファイル識別子を返す識別子を返す.
6	Read	ファイル識別子が示すストリームから，バッファに読み込む
7	Write	ファイル識別子が示すストリームに，バッファから書き込む
8	Close	ファイル識別子が示すストリームを閉じる
9	Unlink	ファイルの削除行う
10	Mmap	ファイルをメモリ空間に割り当てる
11	Connect	ネットワーク接続を試みる
12	Accept	ネットワーク接続要求を受け入れる

物理メモリのシミュレーションは Java の byte 型配列として，MIPS Sim 内部に実装されており，実際の R3000 プロセッサと同様に，メモリ上のデータはリトルエンディアンで配置され，4 バイト単位でアクセスすることができる．メモリ管理方式としてページング方式を採用している．ユーザプロセスには 0 番地から始まる仮想メモリ空間が割り当てられ，仮想ページ番号と物理ページ番号の変換は，シングルリニアページテーブルを用いて行う．先に挙げた制約の通り，動作モードの区別が無いため，本来 R3000 が持っているメモリ保護機能は持っていないが，MIPS Sim がアクセスできるのは MIPS Sim が内部に保持しているバイト配列のみであり，プロセスに割り当てられた仮想空間にのみ操作を実行できる．従ってカーネルが使用するメモリ空間とは独立しており，ユーザプロセスのメモリアクセスがカーネルの動作に悪影響を与えることは無い．

Nachos はディスク装置のハードウェアシミュレーションを実装していないが，StubFileSystem と呼ばれる仮想的なファイルシステムを搭載している．Kernel やユーザプロセスは，ホスト OS のファイルを Machine 経由でアクセスすることができる．StubFileSystem からはホスト OS の特定のディレクトリが 1 つの空間として参照できるが，ディレクトリ構造による管理は実装されておらず，ホスト OS 側の子ディレクトリは参照することはできない．

Nachos は表 1 に示す．システムコールを提供する．行頭の数字はシステムコール例外発行時にカーネルに渡されるシステムコール番号である システムコール発行時には，システムコールごとに規定された規則に基づきレジスタに引数を格納する．MIPS Sim がシステムコールを発行すると，システムコールハンドラはレジスタから引数を読み出し，適切な処理を行う．

4. ARM アーキテクチャ

ARM アーキテクチャは，ARM 社により開発された，32bit RISC(Reduced Instruction Set Computer) アーキテクチャである．性能に対する消費電力が小さいこと，IP コアでの販売を行いペリフェラルのカスタマイズが容易であることから，組み込み機器において利用されることが多い．また，近年では ARM プロセッサコアを FPGA (Field-Programmable Gate Array) に実装することが可能になっており，任意の構成で ARM コアを利用することができる．以上に挙げた，組み込み機器での採用事例が多い点，FPGA に自由に実装することができる点より，今回の研究の題材に適合すると判断し，Nachos に追加するシミュレータの実装ターゲットとした．この章では，Nachos にシミュレータとして追加実装する ARMv4T アーキテクチャについて述べる．

ARM アーキテクチャにはいくつかの命令セットがあり，ターゲットとするシステムの目的に合わせて選択することができる．ここでは使用されることの多い 32 ビット ARM 命令セットと 16 ビット Thumb 命令セットについて取り上げる．32 ビット ARM 命令セットは ARM の命令セットの中でも最も基本的な命令セットである．命令の先頭に実行条件フラグを持ち，分岐命令の数を減らすことができるなど，他のアーキテクチャにないユニークな特徴がある．16 ビット Thumb 命令は 32 ビット ARM 命令セットの命令長を半分に圧縮したもので，32 ビット ARM 命令セットに対して処理性能は劣るものの，コード効率を上げることができる．

ARM アーキテクチャはユーザ，FIO，IRQ，スーパバイザ，アボート，未定義，システムの 7 個の動作モードを持つ．モードは非特権モードのユーザとそれ以外の特権モードに分類でき，特権モードが全ての命令を実行できるのに対し，非特権モードでは，システムに影響を与える一部の命令の実行が制限される．この機構により，ユーザプログラムの意図しない，あるいは意図的なシステム破壊を防止している．プロセッサモードの確認は CPSR レジスタの値を参照することで行い，変更はそのレジスタに特権モードで値を書き込むことで行われる．また，FIQ, IRQ, スーパバイザ, アボート, 未定義については，それぞれ対応する例外が発生した際に自動的に移行される．レジスタは 37 個用意されており，このうちの 20 個のレジスタはプロセッサの実行モードにより切り替えが行われる．このうち，ユーザプログラムからアクセスすることができるのは，RO から R15 の汎用レジスタとプロセッサの実行状態などを保持する CPSR レジスタである．

© 2012 Information Processing Society of Japan

ARM アーキテクチャはリセット，未定義命令，ソフトウェア例外，プリフェッチアボート，データアボート，IRQ 割り込み，FIQ 割り込みの 7 種類の例外を持つ．リセット，IRQ 割り込み，FIQ 割り込みについては外部信号の入力により発生し，その他の割り込みはプロセッサの命令により発生する．それぞれの例外にはベクタアドレスが定義されており，そのアドレスに例外ハンドラのアドレスを書き込んでおくことで例外処理を行うことができる．例外発生時には，例外に応じた動作モードの変更や環境の保存などの処理を行い，例外処理が完了した後には，復帰処理が行われる．

5. 実　　装

この章では Nachos のヘテロジニアス化のために変更を加えた点について述べる．以下の記述では「オリジナル」と表記したものは University of California, Berkeley で開講されている CS162: Operating System and System (2011 Spring) で授業教材として使用されているものを指す．

5.1 Nachos の追加実装

3 章で述べたように，Nachos は大学教育の教材として開発された．そのため，公式に配布されているソースコードは，OS の動作に必要な機能が一部省略されていたり，機能が制限されている．そこで本研究を進めるにあたり，不足している機能の実装を行った．

5.1.1 システムコールハンドラ

MIPS Sim にて syscall 命令によりシステムコール例外が発生すると，Kernel の例外ハンドラに処理が移り，対応するシステムコールハンドラ (ハンドラ) が呼び出される．オリジナルでは，ハンドラの呼び出し部までが実装されており，ハンドラを実装する必要があった．ハンドラのうち，ユーザプロセスのメモリ空間の操作やレジスタ操作が伴うものについては，MIPS Sim に用意されているインタフェースを利用した．なお，システムコールのうち，Join, Unlink, Mmap, Connect, Accept については，本研究との関連が他のシステムコールに比べて比較的少ないため，本稿の段階では未実装である．

5.1.2 メモリ割り当て機構

オリジナルは，プロセスごとにページテーブルを持ち，タスク切り替え時のコンテキストスイッチで，プロセッサが参照するページテーブルを切り替える．したがってプロセス毎に独立した仮想メモリ空間を使用することができる．しかしながら，初期状態ではユーザプロセス作成時に物理アドレス空間全体を，仮想アドレス空間に割り当てるため，複数プロセスでメモリを共有することができない．本実装では，MIPS アーキテクチャと ARM アーキテクチャのプログラムを並行して動作させることを目的としており，マルチプロセス対応にすることは不可欠である．そこでオリジナルに修正を加え，ユーザプロセスのロード時に必要なメモリサイズを計算し，物理ページを割り当てるように変更を加えた．また，プロセスの終了時には，割り当てられていた領域を解放するように変更を加えた．

5.2 プロセッサ間共有メモリ

オリジナルでは，プロセッサからアクセスする主記憶のシミュレーションは MIPS Sim 内にバイト配列として実現されている．この構造のまま ARM Sim を実装すると，プロセッサ毎に主記憶のシミュレータを持つこととなる．これを提案するシステムのモデルに近づけるために，共有メモリをシミュレートするクラスを実装し，Machine が管理するように変更を加えた．このクラスは，オリジナルの MIPS Sim が行っていた操作と同様にアクセス可能である．また，この共有メモリクラスは，2 つのプロセッサから同時アクセスされることがあるので，セマフォによる相互排除を行って，メモリ内容の不整合を防いでいる．なお，物理メモリアクセスに伴う遅延については再現されていない．

5.3 MIPS Sim の構成変更

MIPS Sim は Nachos のクラス群のうち，nachos.machine.Processor として実装されている．Nachos 起動時にこの Processor クラスのインスタンスが作成され，ユーザプログラムのロード時やコンテキストスイッチ時に，Kernel から参照されている．MIPS Sim と ARM Sim のヘテロジニアス構成にするためには，Nachos 起動時に MIPS Sim と ARM Sim それぞれのインスタンスを作成して管理することが必要である．しかしながら，ARM Sim と MIPS Sim がそれぞれ独立して実装されていると，Kernel から Processor を参照する際に，プロセス種別に応じて参照先を切り替える必要が生じる．一部の例外を除いて，Kernel からシミュレータを参照する際は，それが MIPS Sim であるか ARM Sim であるかの区別は必要ないので，どちらのシミュレータも同じ参照型で扱うことができると，プログラムの冗長化を回避でき，保守性が保たれる．

そこで本研究では，新たに Processor インタフェースを作成し，Processor クラスとして実装されていた MIPS Sim を MipsProcessor という名前で Processor クラスの実装クラスに再実装した．Processor インタフェースには，MIPS Sim と ARM Sim で共通するメソッドが定義してあり，Kernel からは Processor 参照型でアクセスすることができる．この構成変更により，カーネルはプロセスの種別を意識することなく，シミュレータのメソッドを呼び出すことができる．

© 2012 Information Processing Society of Japan

5.4 ARM Sim の実装

Nachos を MIPS アーキテクチャとのヘテロジニアス構成にするために，既に実装されている MIPS Sim を元に，ARM Sim の実装を行った．なお，ARM Sim は前述した Processor インタフェースを ArmProcessor として実装している．この節ではまず，基盤となった MIPS Sim の命令実行について，特に ARM シミュレータの実装に必要な箇所について述べ，次に MIPS Sim からの変更点について述べる．

MIPS Sim の実体は，オリジナルの nachos.machine.Processor である．このクラスには，命令のフェッチ，解釈，実行，結果の書き戻しを担当するメソッドが準備されており，この 4 つのメソッドをスレッドが繰り返すことでシミュレーションを実施する．シミュレータの初期化時には，Kernel の特定のメソッドが例外ハンドラとして登録され，例外発生時には，MIPS Sim は動作を中断し，例外処理に移る．また MIPS Sim は，レジスタの内容を管理する int 型配列を持つ．この配列はシミュレータ内からは通常の配列アクセスが可能であるが，クラスの外から直接操作することはできず，実装されているアクセサメソッドを経由して操作を行う．

次に，ARM Sim の具体的な実装について述べる．まず，レジスタのシミュレーションを ARM のものに変更した．具体的にはレジスタの数や参照名に変更を加えた．次に，命令解釈部に変更を加え，ARM アーキテクチャの命令が解釈可能なように変更を加えた．また，実際のレジスタ操作などを行う実行部にも変更を加えた．現段階では，第 6 章で動作確認に用いるプログラムが使用している命令のみ実装が完了している．MIPS Sim では実行と分離されていた，結果のレジスタやメモリへ書き戻しについては簡略化のために命令実行部でまとめて行っている．

今回実装した ARM Sim には，MIPS Sim と同様に実際の ARM アーキテクチャに対していくつか制限がある．代表的なものを以下に述べる．

- プロセッサの動作モードがない
- 浮動小数点演算を行うことができない
- ARM アーキテクチャの例外のうち，発行できるのはシステムコール時に利用する SWI のみである

5.5 MIPS Sim と ARM Sim の並列動作

Nachos は MIPS Sim 単体が動作することを前提としているため，シミュレータ以外の部分にも変更が必要である．以下では，並列動作のために Kernel に変更を加えた点を述べる．

5.5.1 プロセス種別の管理

Nachos の内部では，プロセスの情報を管理する UserProcess クラスが利用されている．これはプロセス生成時にプログラムのロードを担当したり，コンテキストスイッチ時のレジスタ内容の保存，復元などを担当するクラスである．今回のヘテロジニアス化では，プロセスのアーキテクチャ種別を管理することが必要であるので，このクラスのインスタンスフィールドとして，プロセスの種別を取り入れた．プロセスの種類の判定はプログラムのロード時に COFF ヘッダを参照することで行っている．

5.5.2 レジスタ参照の変更

Kernel の中には，シミュレータのレジスタを MIPS アーキテクチャ固有のレジスタ名で参照しているものがあった．この参照については，実行しているプロセスの種別に応じてレジスタ参照名を変更する処理を追加した．

6. 動 作 確 認

この章では第 5 章で述べた変更，実装を行った Nachos に対して行った動作確認について述べる．

6.1 実 行 環 境

以下の環境で動作確認を行った．

- ホスト OS: Gentoo Linux(Linux2.6.38)
- Java コンパイラ: javac 1.6.0_26
- Java 実行環境: Java SE Runtime Environment (build 1.6.0_26-b03)

テストのためのユーザランドプログラムは C 言語で記述し，MIPS, ARM それぞれのアーキテクチャに合わせたツールチェインを利用して実行ファイルを作成した．MIPS アーキテクチャのプログラムについては，コンパイラとして gcc 3.2.2, リンカとして GNU ld 2.13.2 を利用した．*1ARM アーキテクチャのプログラムについては，コンパイラとして gcc 4.5.3, リンカとして GNU ld 2.22 を利用した．なお，利用した gcc では COFF 形式の ARM アーキテクチャプログラムが出力できなかったので，gcc が出力したオブジェクトファイルを GNU objcopy 2.21.1 で COFF 形式に変換した．

今回の実装の目的である，ARM シミュレータと MIPS シミュレータのヘテロジニアス化を確認するために，write システムコールを呼び標準出力に文字列を出力するプログラム

⋆1 [6] で配布されているものを利用した

© 2012 Information Processing Society of Japan

をC言語で作成し，それぞれのアーキテクチャ向けの実行ファイルを作成し，nachosで実行したところ，どちらのシミュレータも正しく動作していることを確認した．

7. 関連研究

ヘテロジニアスマルチコアアーキテクチャ向けのOSの研究としては，The multikernel[2]がある．このmultikernelは本研究が提案したテーマと対照的に，OSの管理機能を分散するものである．また，別の先行研究としてはHelios[9]がある．

オブジェクト指向言語による信頼性の高いOSの実装に関しては，先に触れたように，Javaでの実装を検討したJNode[1]やJx[4]，Ocamlによる実装を検討したOCOS[14]などが先行研究として挙げられる．

8. 結　論

本研究では，ヘテロジニアス環境における効率的かつ信頼性の高いシステム管理を実現するために新たな管理システムモデルを提案した．また，そのモデルの更なる具体化と実現に向けての知見を得るために，Java仮想マシンで動作するOSであるNachosのヘテロジニアス化を行った，結果として，MIPSアーキテクチャとARMアーキテクチャが並列動作するヘテロジニアスマルチコア環境をシミュレータ上に構築することに成功した．しかしながら，本稿の段階では，第2章で提案したモデルを検証することができる段階まで実装できているとは言い難い．今後は，命令セットのさらなる実装とハードウェアシミュレータの詳細化を進め，提案したモデルの検証を進める予定である．

参考文献

1) : JNode.org, http://www.jnode.org/.
2) Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A. and Singhania, A.: The multikernel: a new OS architecture for scalable multicore systems, *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, New York, NY, USA, ACM, pp. 29–44 (2009).
3) Christopher, W.A., Procter, S.J. and Anderson, T.E.: The Nachos Instructional Operating System, Technical Report UCB/CSD-93-739, EECS Department, University of California, Berkeley (1992).
4) Golm, M., Felser, M., Wawersich, C. and Kleinöder, J.: The JX Operating System, *Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, pp.45–58 (online), available from ⟨http://dl.acm.org/citation.cfm?id=647057.713870⟩ (2002).
5) Hettena, D.: A Guide to Nachos 5.0j, http://www.eecs.berkeley.edu/~kubitron/courses/cs162-F10/Nachos/walk/walk.pdf.
6) Joseph, A.D. and Stoica, I.: Spring 2012 CS162: Operating Systems Programming. http://inst.eecs.berkeley.edu/~cs162/sp12/.
7) Kane, G.，前川守監訳：mips RISCアーキテクチャー R2000/R3000 ー，共立出版株式会社，初版1刷 edition (1992年).
8) Nellans, D., Balasubramonian, R. and Brunvand, E.: OS execution on multi-cores: is out-sourcing worthwhile?, *SIGOPS Oper. Syst. Rev.*, Vol.43, pp.104–105 (online), DOI:http://doi.acm.org/10.1145/1531793.1531812 (2009).
9) Nightingale, E. B., Hodson, O., McIlroy, R., Hawblitzel, C. and Hunt, G.: Helios: heterogeneous multiprocessing with satellite kernels, *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, New York, NY, USA, ACM, pp. 221–234 (online), DOI:http://doi.acm.org/10.1145/1629575.1629597 (2009).
10) N.Sloss, A.，Symes, D.，Wright, C.，アーム(株)監訳：ARM組み込みソフトウェア入門，CQ出版社，第3版 edition (2011年).
11) Schoeberl, M.: A Java Processor Architecture for Embedded Real-Time Systems, *Journal of Systems Architecture*, Vol. 54/1–2, pp. 265–286 (online), DOI:http://dx.doi.org/10.1016/j.sysarc.2007.06.001 (2008).
12) the interface(ed.): ARM0/11/XScaleハンドブック，CQ出版社 (2010).
13) W., R.M.: Cell Broadband Engine processor : Design and implementation, *IBM*, Vol. 51, No. 5, pp. 545–557 (online), available from ⟨http://ci.nii.ac.jp/naid/80018240880/⟩ (2007).
14) 井上翔大，大山恵弘：OCamlによるOSの実装，情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム]， Vol.2010, No.4, pp.1–8 (2010-01-20).

© 2012 Information Processing Society of Japan