Vol.2014-ARC-210 No.13
Vol.2014-OS-129 No.13
2014/5/15

Linuxと*AnT*オペレーティングシステムの混載システム

福島 有輝[1]　山内 利宏[1]　谷口 秀夫[1]

概要：***AnT*** オペレーティングシステムは，マイクロカーネル構造を有し，高い適応性と堅牢性の実現を目指している．***AnT*** は独自 OS であるため，使用できる入出力機器が少ない．また，多くの OS 機能を実現すると，OS サーバ数が増加し，好ましくない．そこで，既存 OS である Linux の入出力機能や OS 機能を使用できれば，***AnT*** 上のアプリケーションプログラムで多くの処理を実現できる．本稿では，マルチコアプロセッサの特徴を生かし，1 台の計算機上で Linux と ***AnT*** が独立に同時走行できる混載システムについて述べる．具体的には，混載システムの実現における ***AnT*** の課題と対処を示し，***AnT*** の修正量を示す．

1. はじめに

計算機の多様な利用を支える高い適応性と堅牢性をあわせもつオペレーティングシステム (以降，OS と呼ぶ) が必要となっている．これを実現する OS プログラム構造として，マイクロカーネル構造 [1]~[3] がある．マイクロカーネル構造は，例外・割り込み管理機能やスケジュール機能といった最小限の OS 機能をカーネルとして実現し，通信制御処理やデバイスドライバなどの OS 機能をプロセス (OS サーバ) として実現するプログラム構造である．これにより，全 OS 機能をカーネルとして実現するモノリシックカーネル構造に比べ，機能の追加や削除を容易にできる．また，OS サーバごとに機能を分担させることにより，プログラムの暴走によるシステム全体の破壊を防止できる．

そこで，我々は，高い適応性と堅牢性をあわせもつマイクロカーネル構造の ***AnT*** オペレーティングシステム (An operating system with adaptability and toughness)[4] を開発している．***AnT*** は独自に研究開発している OS であるため，使用できる入出力機器が少ない．また，多くの OS 機能を実現すると，OS サーバ数が増加し，好ましくない．そこで，既存 OS である Linux の入出力機能や OS 機能を使用できれば，***AnT*** 上のアプリケーションプログラムで多くの処理を実現できる．一方，マルチコアプロセッサの特徴を生かし，1 台の計算機上で複数 OS を動作させる OS として，Mint(Multiple Independent operating systems with New Technology)[5] がある．

本稿では，Mint を利用して，Linux と ***AnT*** の混載システムを実現する手法について述べる．具体的には，混載システムの実現における ***AnT*** の課題と対処を示し，***AnT*** の修正量を示す．

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

2. *AnT* オペレーティングシステム

2.1 基本構造

AnT はマイクロカーネル構造を有する OS であり，マルチコアに対応している (以降，マルチコア ***AnT*** と呼ぶ)[6]．マルチコア ***AnT*** の基本構造を**図 1** に示す．m-カーネルは，電源投入時に最初に起動するコアを制御し，スケジューラ機能やメモリ管理機能といったマイクロカーネルに必要な全機能を有する．一方，p-カーネルは，m-カーネルが制御するコア以外のコアを制御し，機能を例外・割り込み管理機能，サーバプログラム間通信機能，およびスケジュール機能に絞り，軽量化を図っている．OS サーバは，適応したシステムに必須なプログラム部分であり，通信制御処

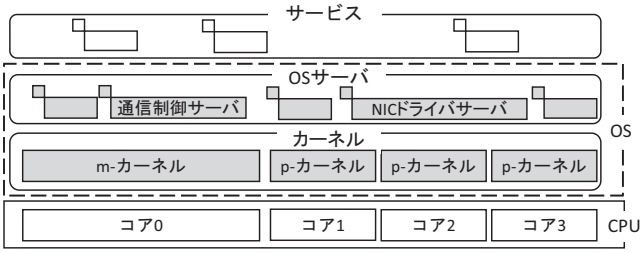

図 1　マルチコア ***AnT*** の基本構造

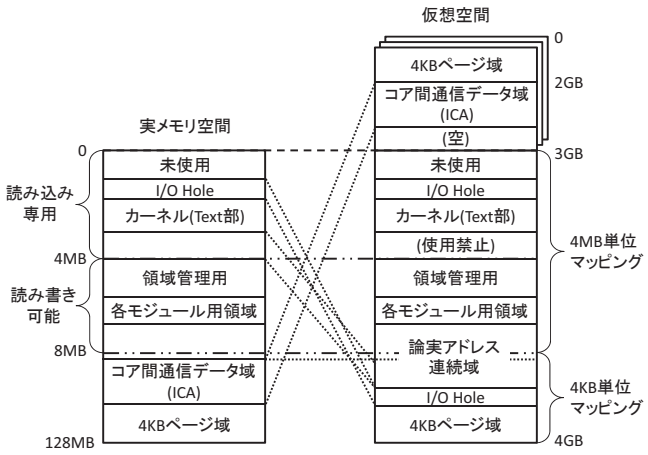

図 2 マルチコア ***AnT*** の仮想空間の構成

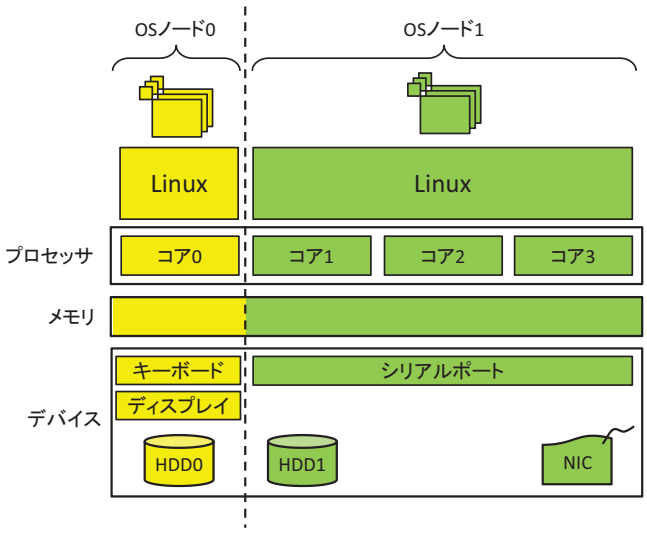

図 3 Mint の構成例

理や NIC ドライバ処理をプロセスとして提供する．サービスは，サービスを提供するプログラム部分である．

2.2 仮想空間の構成

マルチコア ***AnT*** の仮想空間の構成を**図 2** に示す．仮想空間は，0 から 3GB をプロセス空間，3GB 以降をカーネル空間とし，多重仮想記憶である．0 から 2GB のプロセス空間は，OS サーバやサービスのプログラムが使用する空間であり，4KB 単位でマッピングする．これに対し，2GB から 3GB のプロセス空間は，OS サーバとサービスのプログラムが相互のデータ通信に使用する空間である．この領域をコア間通信データ域 (以降，ICA : Inter-core Communication Area) と呼ぶ．ICA は，プロセス間で共用可能な空間に位置づけられているため，プロセス間の高速なデータ授受が可能である．3GB から 3GB+8MB のカーネル空間は，読み込み専用であるカーネルのテキスト部と読み書き可能なカーネルのデータ部をそれぞれ 4MB 単位でマッピングする．この領域は，実アドレスと論理アドレスがすべて連続しており，論実アドレスを容易に変換できる．

3. Linux と Mint

3.1 基本構造

Mint は，仮想化によらず，1 台の計算機上で複数の Linux を独立に同時走行させる OS である．以降では，Mint で同時走行させる各 Linux を OS ノードと呼ぶ．Mint は Linux を改造することで実現しており，改造内容は，3.2 節で詳しく述べる．Mint では，仮想化によらず各 OS ノードを実計算機上で直接走行させているため，実計算機に近い性能で OS ノードの走行を実現している．

Mint は，1 台の計算機上でプロセッサ，メモリ，およびデバイスといったハードウェア資源を効果的に分割し，各 OS ノードで占有する構成である．Mint の構成例を**図 3** に示し，プロセッサ，メモリ，およびデバイスの分割と占有方法について，以下で説明する．

(1) プロセッサ
コア単位で分割し，各 OS ノードで占有する．

(2) メモリ
走行させる OS ノードの数だけ実メモリを空間分割し，各 OS ノードで占有する．

(3) デバイス
デバイス単位で分割し，各 OS ノードが仮想化によらず直接占有制御する．

Mint において，OS ノードを起動させる際，最初に 1 つの OS ノードを起動し，この OS ノードから他 OS ノードを順番に起動させる．最初に起動する OS ノードを OS ノード 0 と呼び，以降に起動する OS ノードを起動順に OS ノード 1，OS ノード 2，・・・と呼ぶ．

3.2 Mint の実現方式

Mint は，Linux におけるハードウェア資源の分割方式と起動方式の 2 つを改造している．それぞれについて，以下で説明する．

(1) ハードウェア資源の分割方式

(A) コアの分割
OS ノード間で占有するコアの重複を避けるため，OS ノードが最初に使用するコアの Local APIC ID よりも小さい Local APIC ID をもつコアは先に起動した OS ノードによって占有されていると定め，このコアを起動しないようにしている．また，各 OS ノード間で割り込み通知先コアの指定に使用する論理 APIC ID の重複を防ぐため，論理 APIC ID の算出がハードウェアによって一意に定まる Local APIC ID を使用している．

(B) メモリの分割
Linux は，BIOS から取得したメモリマップをもとに，自

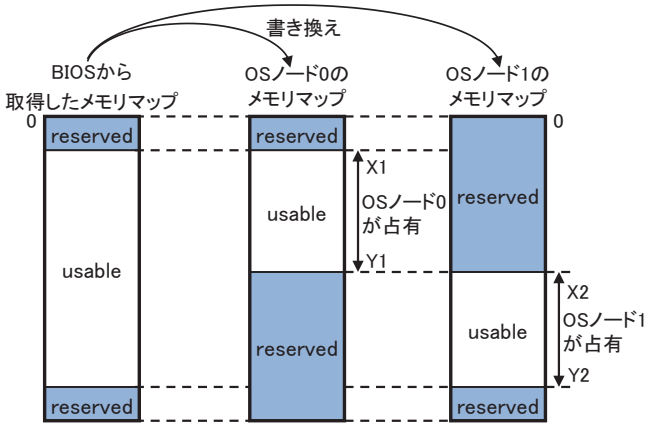

図 4 Mint におけるメモリの分割

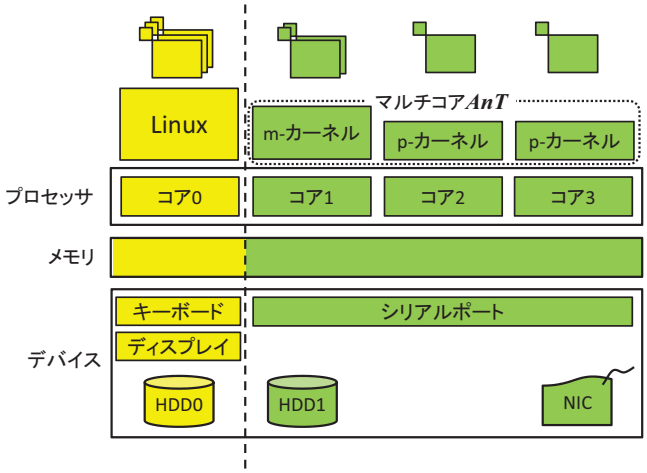

図 5 Linux と ***AnT*** の混載システムの構成

身が使用可能な実メモリ領域を認識する．実メモリを OS ノード間で排他的に分割するため，BIOS で取得したメモリマップを OS ノードごとに書き換え，各 OS ノードに重複しない実メモリ領域を認識させる．この様子を**図 4** に示し，以下で説明する．図 4 では，使用可能な実メモリ領域として，OS ノード 0 は，***X1*** から ***Y1*** まで，OS ノード 1 は，***X2*** から ***Y2*** までを認識する．各 OS ノードは，この領域以外を予約領域と認識し，使用しない．

（C）デバイスの分割

デバイスには，キーボード，ディスプレイ，およびシリアルポートといったレガシーデバイスと NIC(Network Interface Card) や HDD といった PCI デバイスの 2 つがある．OS ノード間でこれらのデバイスを分割するため，以下のような改造を行なっている．

（a）レガシーデバイス

各 OS ノードで占有しないレガシーデバイスは，初期化処理を行なわないことで無効化している．

（b）デバイスドライバ

PCI バスドライバを改造することで，占有しないデバイスのデバイスドライバを使用不可にしている．これにより，各 OS ノードで占有するデバイスのみにデバイスドライバを割り当てることを可能にしている．

（c）割り込み処理

先に起動した OS ノードが I/O APIC に設定した内容を後に起動した OS ノードが上書きしないようにしている．また，論理 APIC ID を使用することで，2 つ以上の OS ノードで共用する割り込みでは，割り込みを複数の OS ノードに通知している．

（2）起動方式

Mint では，他 OS ノードの起動に，改造した Kexec を使用している [7]．本来，Kexec は，Linux を再起動するための仕組みであるため，通常，カーネルイメージを実メモリ上に展開した後，各種初期化を省いたカーネル開始ルーチン (以降，purgatory と呼ぶ) にジャンプする．一方，Mint では，自身が purgatory にジャンプする代わりに，指定したコアへ IPI(Inter-Processor Interrupt) を送信するようにしている．

4. Linux と *AnT* の混載システム

4.1 設計方針

Linux と ***AnT*** の混載システムの実現においては，Mint を拡張し，混載の実現に要する工数を最小限にする．設計方針を以下で説明する．

（方針 1）Linux 走行コアは 1 つとし，他コアはマルチコア ***AnT*** 走行用とする

Linux 走行コアを 1 つとすることで，マルチコア ***AnT*** は，多くのコアを使用して動作できる．

（方針 2）Linux の起動後にマルチコア ***AnT*** を起動する

Mint では，他 OS ノードの起動に，改造した Kexec を使用している．一方，マルチコア ***AnT*** には，そもそも Kexec の機能は存在しない．工数を最小限にするため，マルチコア ***AnT*** を Linux の後に起動することで，マルチコア ***AnT*** に Kexec を移植しなくて済むようにする．

以上より，Linux と ***AnT*** の混載システムの構成を**図 5** に示す．なお，図 5 は，各 OS にプロセッサ，メモリ，およびデバイスを以下のように割り当てた場合の例である．

（1）プロセッサ

Linux にコア 0，マルチコア ***AnT***(m-カーネル) にコア 1，マルチコア ***AnT***(p-カーネル) にコア 2 とコア 3 を割り当てる．

（2）メモリ

Linux とマルチコア ***AnT*** に実メモリを空間分割して割り当てる．

（3）デバイス

Linux にキーボード，ディスプレイ，および HDD0，マルチコア ***AnT*** にシリアルポート，NIC，および HDD1 を割り当てる．

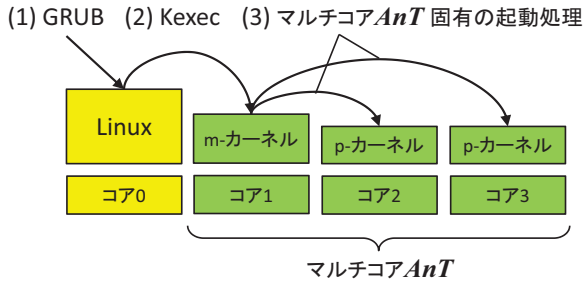

図 6 各 OS の起動順序

また，各 OS の起動順序を**図 6** に示し，以下で説明する．
（1）GRUB により，コア 0 で Linux を起動
（2）Kexec により，コア 1 でマルチコア ***AnT***(m-カーネル) を起動
（3）マルチコア ***AnT*** 固有の起動処理により，マルチコア ***AnT***(m-カーネル) がコア 2 とコア 3 にそれぞれマルチコア ***AnT***(p-カーネル) を起動

4.2 課題

Mint を利用してマルチコア ***AnT*** を動作させるため，マルチコア ***AnT*** には，ハードウェア資源の分割方式と起動方式の 2 つの課題が考えられる．それぞれについて，以下で説明する．
（課題 1）ハードウェア資源の分割方式
マルチコア ***AnT*** は，Linux によって後から起動されるため，マルチコア ***AnT*** の初期化において，Linux が使用しているコア，メモリ，およびデバイスの環境を破壊してはならない．したがって，マルチコア ***AnT*** においてコア，メモリ，およびデバイスの使用方法を改造する必要がある．
（A）コアの分割
マルチコア ***AnT*** は，起動時にすべてのコアを初期化しているため，Linux で使用しているコアも初期化してしまう．このため，Linux で使用しているコアを初期化しないように改造する必要がある．具体的には，マルチコア ***AnT***(p-カーネル) の起動において，コア起動用の IPI を Linux で使用しているコアに送信しないように改造する．
（B）メモリの分割
マルチコア ***AnT*** は，0 番地から 128MB 番地のメモリを固定で使用しているため，Linux で使用しているメモリを破壊してしまう．このため，Linux で使用しているメモリを破壊しないように改造する必要がある．具体的には，***X*** 番地から ***X***+128MB 番地 (***X*** は，Linux が使用していないメモリアドレス) のメモリを使用するように改造する．なお，***X*** に設定する値において，マルチコア ***AnT*** では次の 2 つの制限が生じる．一つ目は，マルチコア ***AnT*** は，32 ビット OS であり，実メモリを 4GB までしか認識できないため，***X*** を (4GB-128MB) 番地以降にできないことである．二つ目は，マルチコア ***AnT*** では，使用するメモリ領域の先頭は 4MB ページングを使用しているため，***X*** を 4MB 境界に配置しなければならないことである．
（C）デバイスの分割
マルチコア ***AnT*** は，起動時に，使用するデバイスを初期化するため，Linux とマルチコア ***AnT*** で同じデバイスを使用する場合，Linux で使用しているデバイスも初期化してしまう．このため，Linux で使用しているデバイスを初期化しないように改造する必要がある．また，デバイスを使用する際，割り込み通知先を設定する必要がある．この設定において，Linux の使用している割り込み通知先の設定を破壊してはならない．
（課題 2）起動方式
Mint では，他 OS ノードの起動に，改造した Kexec を使用している．このため，マルチコア ***AnT*** を Kexec で起動できるようにカーネルのビルド方法を改造する必要がある．

4.3 対処

4.2 節で示したハードウェア資源の分割方式と起動方式の 2 つの課題に対して，それぞれ以下の対処を行なった．
（対処 1-A）コアの分割
自身の Local APIC ID 以降の Local APIC ID をもつコアに対して，使用するコア数の分だけ，マルチコア ***AnT***(p-カーネル) 起動用の IPI を送信するようにマルチコア ***AnT***(m-カーネル) を改造する．
（対処 1-B）メモリの分割
Linux が使用するメモリの破壊を防ぐため，マルチコア ***AnT*** が使用する実メモリ領域を移動可能にする．なお，使用する実メモリ領域の移動に伴い，実メモリ空間と仮想空間のマッピングを改造する必要がある．ここで，既存のソースコードへの影響を抑えるため，マルチコア ***AnT*** の仮想空間の構成を改造しないこととした．改造前と改造後のマッピングの比較を**図 7** に示し，2 つの利点を以下で説明する．
（利点 1）カーネルを配置する実アドレス (***X***B) と仮想アドレス (3GB) のマッピングにおいて，アドレス変換のオフセット値が 3GB であったのを (3GB-***X***B) に改造するだけでよい
（利点 2）カーネルを配置する実アドレス (***X***B) は，4MB 境界であれば，0 番地から (4GB-128MB) 番地の任意のアドレスに配置できる
（対処 1-C）デバイスの分割
デバイスの分割方法は，デバイスがキーボードやシリアルポートといったレガシーデバイスか，NIC や HDD といった PCI デバイスのどちらかで異なる．また，デバイスを分割するために割り込みの設定が必要である．
（a）レガシーデバイス
起動時にマルチコア ***AnT*** でレガシーデバイスを初期化し

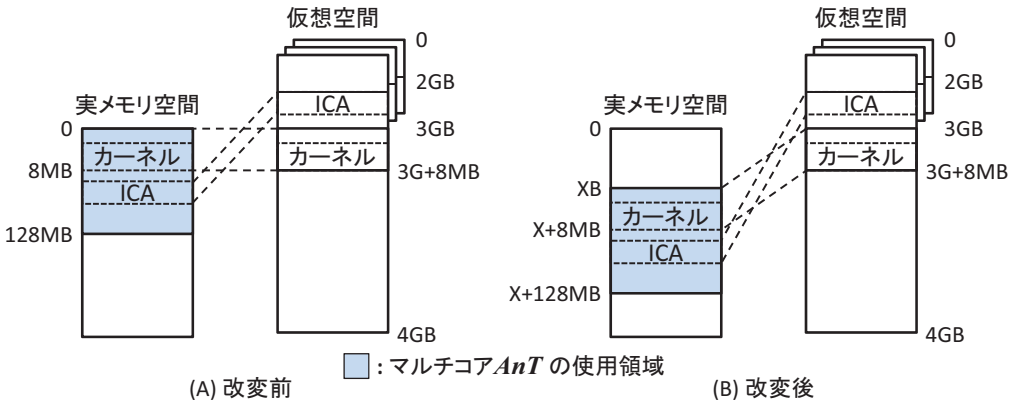

図 7 改造前と改造後のマッピングの比較

ないことで，Linux で使用しているレガシーデバイスの設定を上書きしないようにする．

（b）デバイスドライバ

マルチコア ***AnT*** は，マイクロカーネル構造 OS であるため，PCI デバイスを使用するか否かは，デバイスに対応するドライバサーバプロセスを起動するか否かで決定できる．つまり，マルチコア ***AnT*** では，カーネルに改造を加えずに PCI デバイスを分割できる．

（c）割り込み処理

Linux の割り込み通知先の設定を破壊しないようにするため，割り込み設定において，以下の 2 つの対処を行なった．

（i）先に起動している Linux の割り込み設定を初期化しない

マルチコア ***AnT***(m-カーネル) 起動時，割り込み通知先の設定を行なっている I/O APIC を初期化しないように改造する．

（ii）マルチコア ***AnT*** で使用するデバイスの割り込みだけを設定する

I/O APIC の割り込み通知先において，マルチコア ***AnT*** で使用するデバイスの割り込みをマルチコア ***AnT*** だけに通知するように設定する．一方，Linux とマルチコア ***AnT*** で同一の割り込み番号を使用する場合，割り込み通知先を全てのコアに設定する．この場合，I/O APIC では，1 つの割り込み番号に対して 1 つのベクタ番号しか設定できないため，マルチコア ***AnT*** は Linux で使用されているベクタ番号と同一のベクタ番号を設定する必要がある．

（対処 2）起動方式

Kexec で起動するために最低限必要となるファイルは，カーネルイメージ (以降，bzImage と呼ぶ) である．bzImage は，ELF 形式のカーネルをビルドした後，この ELF 形式のカーネルファイルを圧縮してセットアップルーチンを加えることで作成できる．Linux の bzImage 作成手順を**図 8** に示し，以下で説明する．

（1）Linux のソースコードから“vmlinux”(ELF 形式) をビルド

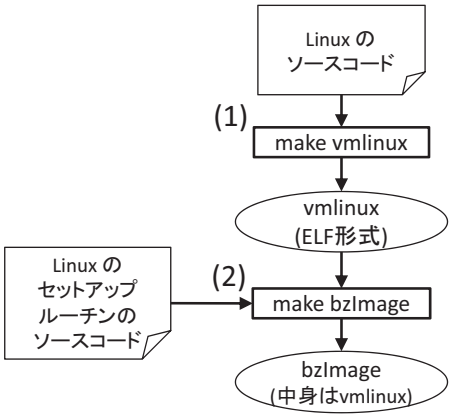

図 8 Linux の bzImage 作成手順

（2）“vmlinux”とセットアップルーチンのソースコードをもとに bzImage を作成

ここで，bzImage の圧縮カーネル部分がマルチコア ***AnT*** である bzImage を作成すれば，マルチコア ***AnT*** を Kexec で起動できると考えられる．しかし，マルチコア ***AnT*** では，ELF 形式のカーネルをビルドする手順までしか行なっていない．したがって，マルチコア ***AnT*** の ELF 形式のカーネルを圧縮してセットアップルーチンを加えるビルド手順を追加する必要がある．しかし，このビルド手順は複雑である．また，多数存在するファイルから必要なファイルを見つけ出し，マルチコア ***AnT*** 用に移植するという移植作業は工数がかかる．そこで，マルチコア ***AnT*** とセットアップルーチンのソースコードを数か所だけ改造し，bzImage の作成に使用する多数のファイルをそのまま流用するという方針にする．この方針を実現したマルチコア ***AnT*** の bzImage 作成手順を**図 9** に示し，以下で説明する．

（1）マルチコア ***AnT*** で bzImage 作成に必要なラベルを追加

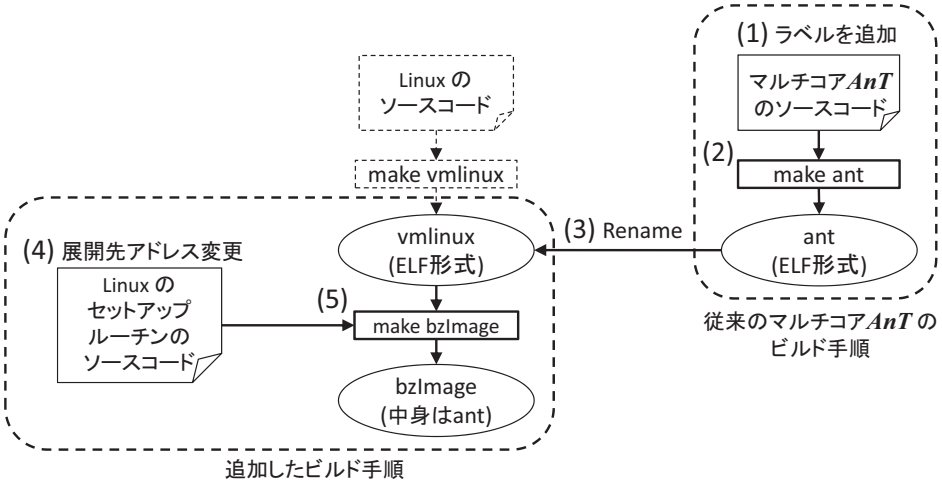

図 9 マルチコア ***AnT*** の bzImage 作成手順

bzImage 作成の過程でカーネルのテキスト部の先頭を表すラベルが必要となる．このため，マルチコア ***AnT*** のテキスト部の先頭にラベルを追加した．

（2）マルチコア ***AnT*** のソースコードから“ant”(ELF 形式) をビルド

（3）“ant”(ELF 形式) を“vmlinux”にリネーム

（4）カーネルの展開先アドレスを変更

セットアップルーチンのソースコードにおいて，カーネルの展開先アドレスをマルチコア ***AnT*** のテキスト部の配置アドレスにする．また，Linux ではカーネルの展開先アドレスを実メモリの先頭 512MB 以内に限定するためのエラーチェックを行なっている．カーネルの配置先アドレスを移動可能にするため，このエラーチェックを解除する．

（5）“vmlinux”(中身は ant) と展開先アドレスを変更したセットアップルーチンのソースコードをもとに bzImage を作成

4.4 期待される効果

4.1 節の“(方針 1) Linux 走行コアは 1 つとし，他コアはマルチコア ***AnT*** 走行用とする”により，マルチコア ***AnT*** にコアを 3 つ割り当てることができ，マルチコア ***AnT*** だけを動作させた場合に近い環境となる．

4.1 節の“(方針 2) Linux の起動後にマルチコア ***AnT*** を起動する”により，起動方式における工数の削減が期待できる．これは，マルチコア ***AnT*** に Kexec を移植しなくてよいためである．

“(対処 2) 起動方式”により，起動方式における工数の削減が期待できる．これは，bzImage の作成に使用する多数のファイルをそのまま流用することで，マルチコア ***AnT*** にほとんど改造を加えず，マルチコア ***AnT*** の bzImage を作成できるためである．

5. 評価

Linux と ***AnT*** の混載システムの実現における工数を評価する．具体的には，マルチコア ***AnT*** からの修正量を調査し，どの程度の改造が必要か，また，どの改造箇所で多くの改造が必要かを示す．

マルチコア ***AnT*** のカーネルのコード行数一覧を表 1，マルチコア ***AnT*** から改造したコードの内訳を表 2 に示す．表 2 では，各改造箇所について改造したファイル数とコード量を示している．なお，改造したコード量は，変更，追加，および削除の合計行数である．各改造箇所のファイル数の合計とファイル数の全体の合計が一致していないのは，複数の改造箇所にあてはまるファイルが存在するためである．なお，改造箇所「プログラム読み込み処理」は，FreeBSD と Linux で gcc の仕様が異なるために生じる問題である．具体的には，プロセス生成時に読み込むプログラムの ELF ヘッダにおいて，プログラムセクション情報

表 1 マルチコア ***AnT*** のカーネルのコード行数一覧

ファイルの種類	ファイル数	コード量 (行数)
C 言語	76	13051
アセンブリ言語	13	1453
ヘッダ	76	4001
合計	165	18505

表 2 マルチコア ***AnT*** から改造したコードの内訳

改造箇所	ファイル数	コード量 (行数)
コアの分割	2	3
メモリの分割	12	48
デバイスの分割	8	35
起動処理	1	2
プログラム読み込み処理	2	17
合計	22	105

が異なるため，差異に対処する必要があった．

表 1 と表 2 から以下の 2 つのことがわかる．

（1）改造したコード量は，マルチコア ***AnT*** のカーネルのコード量である **18505** 行に対して，**105** 行である．これはカーネルのコード全体の約 **0.6%**(**= (105/18505) ∗ 100**) である．つまり，修正量は，カーネルのコード全体に対して十分小さいといえる．

（2）改造したファイル数は，マルチコア ***AnT*** のファイル数である **165** に対して，**22** である．これはカーネルのファイル数全体の約 **13**% (**= (22/165) ∗ 100**) である．コード全体に対する改造したコード量の割合に比べて，ファイル数全体に対する改造したファイル数の割合が大きいことから，改造箇所は，多くの箇所に分散しているといえる．

次に，Linux と ***AnT*** の混載システムにおいて，ハードウェア構成の変化によるマルチコア ***AnT*** の修正量への影響について考察する．具体的には，コア数，メモリ量，および使用するデバイス構成が変化した場合を考え，それぞれについての考察を以下に示す．

（1）コア数の変化

コア数が変化しても，4.3 節の対処（1-A）で示した規則を変更しないため，修正量は増加しない．

（2）メモリ量の変化

メモリ量が変化しても，仮想空間の構成を改造しないため，修正量は増加しない．

（3）使用するデバイス構成の変化

（A）レガシーデバイス

使用するレガシーデバイスの構成にあわせて初期化処理を改造する必要があるため，修正量は増加する．

（B）デバイスドライバ

マイクロカーネル構造により，対応するドライバサーバプロセスを起動するか否かでデバイス構成を変更できるため，修正量は増加しない．

以上より，マルチコア ***AnT*** の修正量が増加するのは，レガシーデバイスの構成を変更した場合だけである．

6. 関連研究

仮想化を使用せず，複数の異なる OS を 1 台の計算機上で混載したシステム [8]〜[10] が提案されている．これらのシステムは，I/O 処理を含む OS 処理を行なう Linux と AP の実行に専念する軽量カーネルが動作する構成である．また，文献 [8][9] は，マルチコア・メニーコア混在型計算機を対象としている．一方，本研究では，軽量カーネルに相当する ***AnT*** において，I/O 処理を行なうことができる．また，本研究では，マルチコアプロセッサを搭載した計算機を対象としている点が異なる．

ExVisor[11] は，複数の OS をハードウェアレベルで分離して動作可能な PPC(Physical Partitioning Controller) を使用して複数の OS を動作可能にする VMM である．文献 [11] では，SH-4A 相当のコアを複数搭載したマルチコア SoC において，Linux と μ ITRON の混載システムを実現している．この混載システムは，AP 資産の再利用とリアルタイム性の確保を可能にしている．一方，本研究で使用した Mint は，VMM を使用せず，各 OS がハードウェアを直接占有制御する点が異なる．

7. まとめ

1 台の計算機上で Linux と ***AnT*** が独立に同時走行できる混載システムについて述べた．まず，複数 OS を独立に同時走行させる方式として，Mint を挙げ，Mint を利用してマルチコア ***AnT*** を動作させるためのマルチコア ***AnT*** の改造箇所について述べた．具体的には，改造箇所は，ハードウェア資源の分割方式 (コアの分割，メモリの分割，およびデバイスの分割) と起動方式であった．コアの分割では，マルチコア ***AnT***(m-カーネル) を自身の Local APIC ID 以降の Local APIC ID をもつコアに対して，使用するコア数の分だけ，マルチコア ***AnT***(p-カーネル) 起動用の IPI を送信するように改造した．メモリの分割では，Linux が使用していない領域でマルチコア ***AnT*** を動作させるため，実メモリ空間と仮想空間のマッピングを改造した．デバイスの分割では，レガシーデバイスと PCI デバイスで分割方法が異なることを示した．マルチコア ***AnT*** では，レガシーデバイスは，初期化しないことで分割する．一方，PCI デバイスは，デバイスに対応するドライバサーバプロセスを起動するか否かで決定できる．また，割り込み処理では，I/O APIC の初期化処理とマルチコア ***AnT*** で使用するデバイスの割り込みの設定を改造した．起動方式では，マルチコア ***AnT*** を Kexec で起動するため，マルチコア ***AnT*** の bzImage を作成するように改造した．

評価では，マルチコア ***AnT*** の修正量を示した．具体的には，改造したコード量 (行数) は，マルチコア ***AnT*** のカーネルの全体の約 0.6%，改造したファイル数は，約 13%となった．

残された課題として，***AnT*** 上のアプリケーションプログラムから Linux に処理依頼する機能の実現がある．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（課題番号：24300008）による．

参考文献

[1] Liedtke, J.: Toward real microkernels, Communications of the ACM, Vol.39, No.9, pp.70-77 (1996).
[2] Tanenbaum, A.S., Herder, J.N., and Bos, H.: Can we make operating systems reliable and secure?, IEEE Computer Magazine, Vol.39, No.5, pp.44-51 (2006).
[3] Black, D.L., Golub, D.B., Julin, D.P., Rashid, R.F., Draves, R.P., Dean, R.W., Forin, A., Barrera, J., Tokuda, H., Malan, G., and Bohman, D.: Microkernel operating system architecture and mach, Journal of Information Processing, Vol.14, No.4, pp.442-453 (1992).

[4] 岡本幸大，谷口秀夫：***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価，電子情報通信学会論文誌 (D)，Vol.J93-D，No.10，pp.1977-1987 (2010).
[5] 千崎良太，中原大貴，牛尾　裕，片岡哲也，栗田祐一，乃村能成，谷口秀夫：マルチコアにおいて複数の Linux カーネルを走行させる Mint オペレーティングシステムの設計と評価，電子情報通信学会技術研究報告，Vol.110，No.278，pp.29-34 (2010).
[6] 井上喜弘，佐古田健志，谷口秀夫：マルチコアプロセッサ上での負荷分散を可能にする ***AnT*** オペレーティングシステム，情報処理学会研究報告，Vol.2012-DPS-150，No.37，pp.1-8 (2012).
[7] 中原大貴，千崎良太，牛尾　裕，片岡哲也，乃村能成，谷口秀夫：Kexec を利用した Mint オペレーティングシステムの起動方式，電子情報通信学会技術研究報告，Vol.110，No.278，pp.35-40 (2010).
[8] 佐伯裕治，清水正明，白沢智輝，中村　豪，高木将通，Balazs Gerofi，思　敏，石川　裕，堀　敦史：ヘテロジニアス計算機上の OS 機能委譲機構，情報処理学会研究報告，Vol.2013-OS-125，No.15，pp.1-8 (2013).
[9] 深沢　豪，佐藤未来子，長嶺精彦，坂本龍一，吉永一美，辻田祐一，堀　敦史，石川　裕，並木美太郎：マルチコア・メニーコア混在型計算機における演算コア側資源管理の代行方式，情報処理学会研究報告，Vol.2012-HPC-134，No.7，pp.1-8 (2012).
[10] Yoonho Park, Eric Van Hensbergen, Marius Hillenbrand, Todd Inglett, Bryan Rosenburg, and Kyung Dong Ryu, “A Hybrid Approach to Exascale Operating Systems,” ExaOSR-2012 Workshop (2012).
[11] 杉本　健，野尻　徹，平松義崇，寺田光一：組み込み向けマルチコアプロセッサにおける複数 OS 実行環境の構築技術，情報処理学会研究報告，Vol.2010-OS-114，No.3，pp.1-8 (2010).