マルチコア *AnT* における AP と OS の処理分散効果の評価

小林 優也[1]　佐藤 将也[1]　谷口 秀夫[1]

概要：プロセッサ内のコア数の増加に伴い，処理を各コアに分散することが重要になっている．オペレーティングシステム（以降，OS）には，マイクロカーネル構造 OS とモノリシックカーネル構造 OS がある．マイクロカーネル構造 OS は，最小限の OS 機能をカーネルとして実現し，その他の大半の OS 機能をプロセス（以降，OS サーバ）として実現する．モノリシックカーネル構造 OS は，すべての OS 機能をカーネルとして実現する．この 2 種類の OS は，実現できる処理の分散形態が異なる．本稿では，マイクロカーネル構造 OS である ***AnT*** とモノリシックカーネル構造 OS である Linux について，応用プログラム（以降，AP）処理と OS 処理の分散形態の違いに着目した性能を比較評価する．

1. はじめに

プロセッサ内のコア数の増加に伴い，マルチコアプロセッサを有効に使うためには，各コアに処理を分散することが重要である．

プロセッサを管理する OS には，マイクロカーネル構造 OS[1]–[5] とモノリシックカーネル構造 OS がある．マイクロカーネル構造 OS は，大半の OS 機能を OS サーバとして実現する．このため，AP プロセスを各コアに分散できるとともに，AP プロセスの分散形態に関係なく，OS サーバを各コアに分散することで，OS 処理も分散できる [6][7]．一方，モノリシックカーネル構造 OS は，AP プロセスのシステムコール発行によって OS 処理を同じコアで実行する．したがって，マイクロカーネル構造 OS とモノリシックカーネル構造 OS は実現できる処理の分散形態が異なる．

我々は，マイクロカーネル構造を有するマルチコア ***AnT*** オペレーティングシステム（An operating system with adaptability and tough-ness）（以降，***AnT***）[8] を提案した．***AnT*** は，プロセスを任意のコアに移譲する機能を備えている．この機能を用いることで，プロセスを各コアに分散し，負荷分散を行うことができる．

本稿では，マルチコア ***AnT*** について，処理を分散する場合の基本性能を述べる．また，***AnT*** とモノリシックカーネル構造の Linux について，AP と OS の分散形態における性能を比較評価する．

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

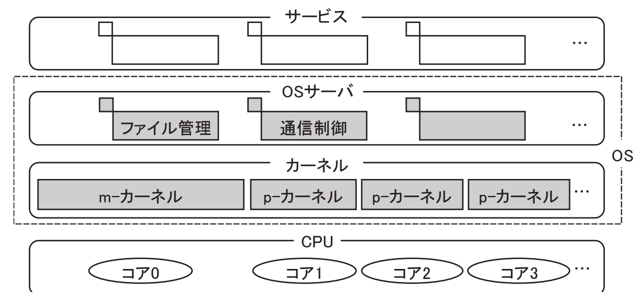

図 1　***AnT*** の基本構造

2. *AnT* オペレーティングシステム

2.1 基本構造

AnT はマルチコアに対応したマイクロカーネル構造 OS である．***AnT*** の基本構造を図 **1** に示し，以下に説明する．カーネルはシステムの動作に必要な最小限の OS 機能を実現するプログラム部分である．また，***AnT*** のカーネルは，コア毎にカーネルを配置する構造（マルチカーネル構造 [9][10]）となっており，マスタカーネル (m カーネル) とピコカーネル (p カーネル) の 2 種類のカーネルからなる．m カーネルはマイクロカーネルのすべての機能を有する．p カーネルは，プロセス実行制御機能，コア間通信制御機能，およびサーバプログラム間通信機能のみを有する．コア毎にカーネルを配置することにより，各コアで独立したスケジューラを実現し，排他制御を抑制できる．さらに，m カーネルと p カーネルの構成にすることで，カーネルのテキスト部の増加を抑制できる．OS サーバは，OS 機能をプロセス化した部分であり，ファイル管理サーバや通信制

© 2019 Information Processing Society of Japan

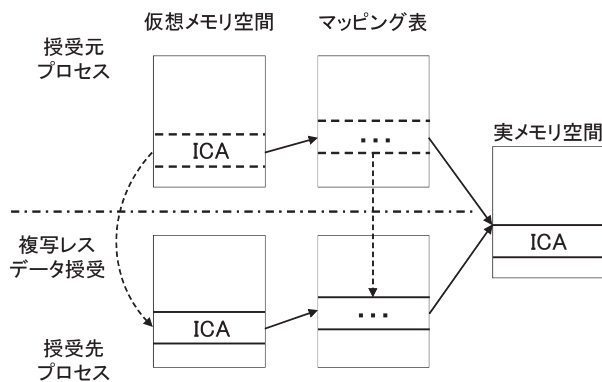

図 2　複写レスデータ授受の様子

御サーバがある．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　複写レスデータ授受機能

AnT は，AP プロセスと OS サーバが相互に通信する際に用いる，コア間通信データ域 (ICA: Intercore Communication Area) を持つ．プロセス間の複写レスデータ授受の様子を図 **2** に示し，以下に説明する．ICA は以下の 3 つの特徴を持つ．

（1）ページを単位とし，n ページ分の領域の確保と解放

（2）確保した領域（n ページ）の実メモリ連続の保証

（3）2 仮想空間での領域の張替え

ICA は，ページを最小単位として管理される領域であり，ICA へのアクセスは，プロセス毎のマッピング表を通して行われる．ここで，マッピング表への書き込みを貼り付けと呼び，マッピング表からの削除を剥がしと呼ぶ．ICA を利用したプロセス間のデータ授受は，授受するデータを格納した ICA をデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間に貼り付けることで行われる．

2.3　サーバプログラム間通信機構

2.3.1　基本機構

サーバプログラム間通信機構 [11] を図 **3** に示し，以下に説明する．この機構では，ICA を用いることにより，プロセス間のデータ授受を複写レスで実現している．具体的には，依頼先プロセスへ渡す引数や通信制御の情報（以降，依頼情報）を制御用の ICA（以降，制御用 ICA）に格納し，扱うデータをデータ用の ICA（以降，データ用 ICA）に格納する．また，カーネルは，プロセス毎に通信のための依頼用リングバッファと結果用リングバッファを持つ．サーバプログラム間通信の処理流れを以下に示す．

（1）依頼元プロセスが処理依頼を行うと，依頼元プロセスの動作するコアのカーネル（以降，依頼元カーネル）は依頼先プロセスの依頼用リングバッファに依頼情報を格納

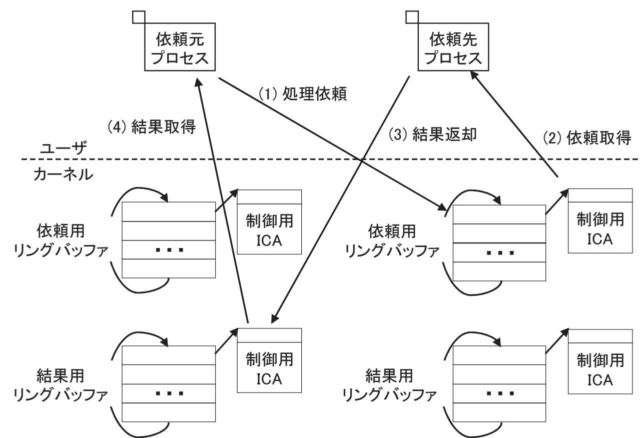

図 3　サーバプログラム間通信の基本機構

した制御用 ICA を登録する．その後，依頼元プロセスの仮想空間から制御用 ICA を剥がす．また，依頼先プロセスが WAIT 状態の場合，依頼先プロセスを起床させる．

（2）依頼先プロセスの動作するコアのカーネル（以降，依頼先カーネル）は，依頼先プロセスの仮想空間に制御用 ICA を貼り付ける．依頼先プロセスは依頼用リングバッファから依頼情報が格納された制御用 ICA を取得し，処理を実行する．

（3）依頼先プロセスが結果返却を行うと，依頼先カーネルは依頼元プロセスの結果用リングバッファに結果情報を格納した制御用 ICA を登録する．その後，依頼先プロセスの仮想空間から制御用 ICA を剥がす．また，依頼元プロセスが WAIT 状態の場合，依頼元プロセスを起床させる．

（4）依頼元カーネルは，依頼元プロセスの仮想空間に制御用 ICA を貼り付ける．依頼元プロセスは結果用リングバッファから結果情報が格納された制御用 ICA を取得し，処理を実行する．

コアを跨いだサーバプログラム間通信では，プロセスを起床させる処理において，コア間通信を伴う．各コアのカーネルは，スケジュールキューを持ち，プロセスの実行を制御している．このため，他コアのプロセスの起床は，コア間通信を用いて当該コアに依頼する．また，全てのプロセスは，コア数と同数の依頼用リングバッファと結果用リングバッファを持つ．そして，他コアのプロセスに対して依頼情報や結果情報の登録を行う際は，自身の走行するコアに対応したリングバッファに登録する．これにより，排他制御オーバヘッドを削減している．

2.4　コア間通信

コア間通信は，コア間で共有している領域（共有領域）を利用し，依頼内容の授受を行うことで実現する．

共有領域は $N \times N$（N はコア数）の配列として実現されており，依頼発行側コア（i）と依頼受理側コア（j）のコア間通信は，共有領域の（i，j）番目のエントリを利用して依頼内容を授受する．これにより，複数のコアが 1 つ

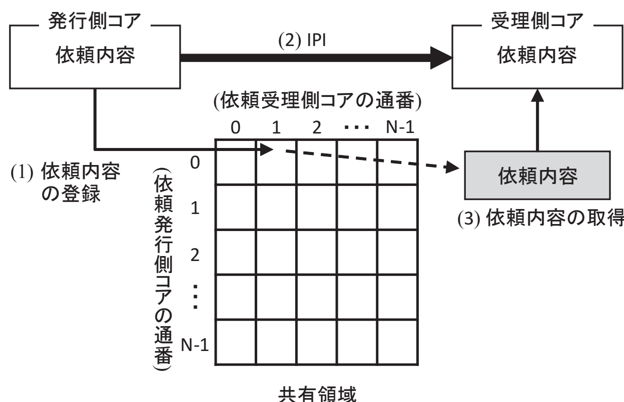

図 4　コア間通信の処理流れ

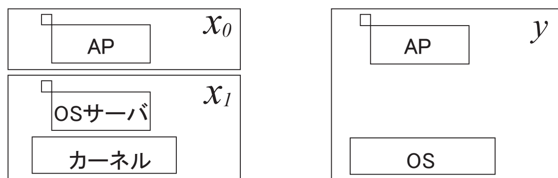

図 5　処理の分散単位

のコアにコア間通信を要求する際の排他制御処理を不要としている．

依頼内容の登録を通知するために IPI（Inter-Processor Interrupt）を利用する．コア間通信の処理流れを**図 4** に示し，以下に説明する．

（1）依頼発行側コアは依頼内容を共有領域に登録する．

（2）依頼発行側コアは依頼受理側コアに IPI を送信し，依頼受理側コアに依頼内容の登録を通知する．

（3）依頼受理側コアは，IPI の受信を契機に，共有領域から依頼内容を取得し，依頼内容の処理を実行する．

3. 処理の分散形態

3.1 処理の分散単位

AP と OS の処理の分散単位を**図 5** に示す．マイクロカーネル構造 OS（図 5（A））は，AP プロセスと OS サーバを各コアに分散することで，AP 処理と OS 処理を別のコアで実行できる．つまり，AP 処理と OS 処理を別の単位（x_0 と x_1）で分散できる．

モノリシックカーネル構造 OS（図 5（B））は，AP プロセスのシステムコール発行によって OS 処理を実行する．このため，AP プロセスを各コアに分散することで OS 処理を分散できる．いいかえれば，AP プロセスを各コアに分散しない場合，OS 処理を分散できない．つまり，AP 処理と OS 処理を 1 つの単位（y）として分散する．

3.2 比較

AP 処理と OS 処理の 4 種類の分散形態について，各 OS が実現可能か否かを**表 1** に示す．「集中」は，複数の処理を同一コア上で実行することを示し，「分散」は，複数の処理を各々別コア上で実行することを示す．マイクロカーネル構造 OS は，全形態を実現できる．また，「集中，集中」と「分散，分散」の場合において，AP 処理と対応する OS 処理を同一コア上で実行しなくてもよい．一方，モノリシックカーネル構造 OS は，AP 処理と OS 処理が同じ形態（「集中」または「分散」）でなければ実現できない．さらに，「集中，集中」と「分散，分散」の場合，AP 処理と対応する OS 処理を同一コア上で実行する．

表 1　分散形態実現の可否

AP 処理	OS 処理	マイクロ	モノリシック
集中	集中	可	可
分散	集中	可	不可
集中	分散	可	不可
分散	分散	可	可

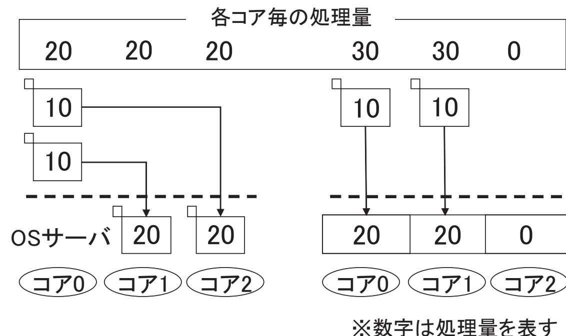

図 6　AP 処理と OS 処理の分散形態の例

AP 処理と OS 処理の分散形態の例を**図 6** に示す．プロセッサ（PU）処理の量を数字で表す．AP 処理の量を 10，対応する OS 処理の量を 20 としている．マイクロカーネル構造 OS の場合（図 6（A）），2 つの AP 処理をコア 0，対応する OS 処理（OS サーバ）をコア 1 とコア 2 に分散させることで，各コアの PU 処理量を均等化（20）できる．一方，モノリシックカーネル構造 OS の場合（図 6（B）），AP 処理をコア 0 とコア 1 に分散させると，対応する OS 処理もコア 0 とコア 1 で実行される．このため，コア 0 とコア 1 の各 PU 処理量は 30，コア 2 の PU 処理量は 0 となり，負荷分散がうまくできない．

4. 評価

4.1 サーバプログラム間通信の基本性能

4.1.1 内容

サーバプログラム間通信には，同期処理と非同期処理がある．同期処理では，依頼元プロセスは処理依頼を行った後，依頼先プロセスから結果が返却されるまで待機する．非同期処理では，依頼元プロセスは処理依頼を行った後，

© 2019 Information Processing Society of Japan

表 2 サーバプログラム間通信の評価環境

CPU	Intel (R) Core (TM) i7-2600 (3.40GHz)
メモリ	8,192 MB

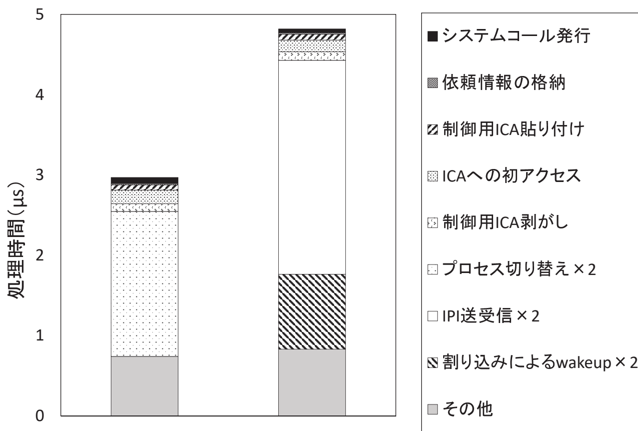

図 7 サーバプログラム間通信（同期処理）の処理時間

依頼先プロセスからの結果返却を待たずに処理を続行し，後に結果受け取りを行う．ここでは，同期処理について，依頼元プロセスと依頼先プロセスが同コアにある場合と別コアにある場合のサーバプログラム間通信の処理時間を述べる．

4.1.2 評価環境と処理流れ

評価環境を表 **2** に示す．測定区間は，依頼元プロセスが同期依頼システムコールを発行する直前から，依頼先プロセスからの結果返却後に依頼元プロセスが動作を再開した直後までの区間とした．なお，サーバプログラム間通信の処理時間のみを測定するため，依頼先プロセスはプログラム通信の処理のみを行う．

4.1.3 結果と考察

測定結果を図 **7** に示し，以下に考察を示す．

（1）別コアの場合，サーバプログラム間通信の処理時間は約 5 μs である．つまり，サーバプログラム間通信のオーバヘッドは小さい．例えば，依頼先プロセスが 1 ミリ秒の処理を行った場合でも，サーバプログラム間通信のオーバヘッドは 1%以下である．

（2）同コアと別コアでは，別コアの方が約 2 μs 長い．これは IPI の送受信と，割り込みによるプロセス起床が要因である．IPI 送受信が不要な場合として，依頼先プロセスが処理中の場合がある．この場合，依頼先プロセスの起床は不要のため，IPI 送受信が不要である．

4.2 分散効果

4.2.1 評価の環境と内容

AP と OS の分散形態による処理時間を評価する．

分散効果の評価環境は，4.1 節と同様（表 2）である．***AnT*** と Linux 3.10 (CentOS7) において AP 処理と OS 処理を分散した形態を図 **8** と図 **9** に示す．

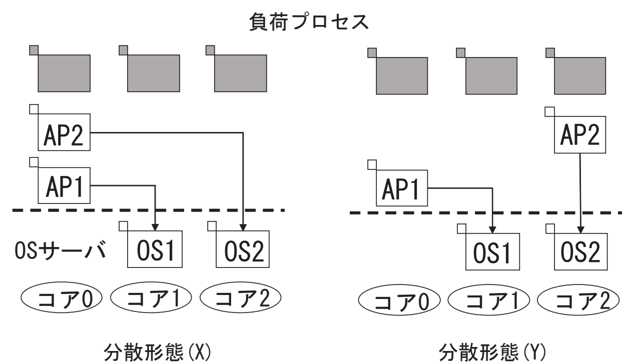

図 8 ***AnT*** の分散形態

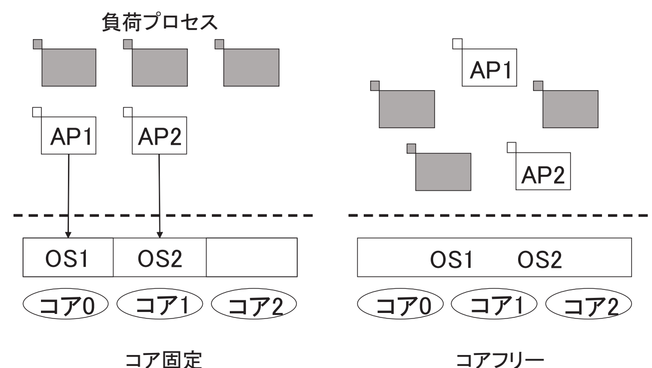

図 9 Linux の分散形態

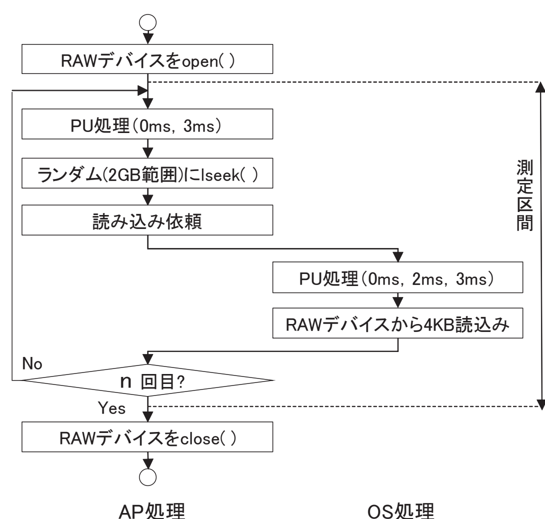

図 10 AP の処理流れ

AP の処理流れを図 **10** に示す．AP は，RAW デバイスからのランダムリード（4 KB）を n 回繰り返す．なお，ランダムの範囲は 2GB とした．***AnT*** の場合，RAW デバイスからの読み込み処理は，制御用 ICA に依頼情報を格納し，サーバプログラム間通信を用いてディスクドライバサーバ（図 8 では OS1 と OS2）に依頼する．Linux の場合，read システムコールを用いて OS に RAW デバイスからの読み込みを依頼する．

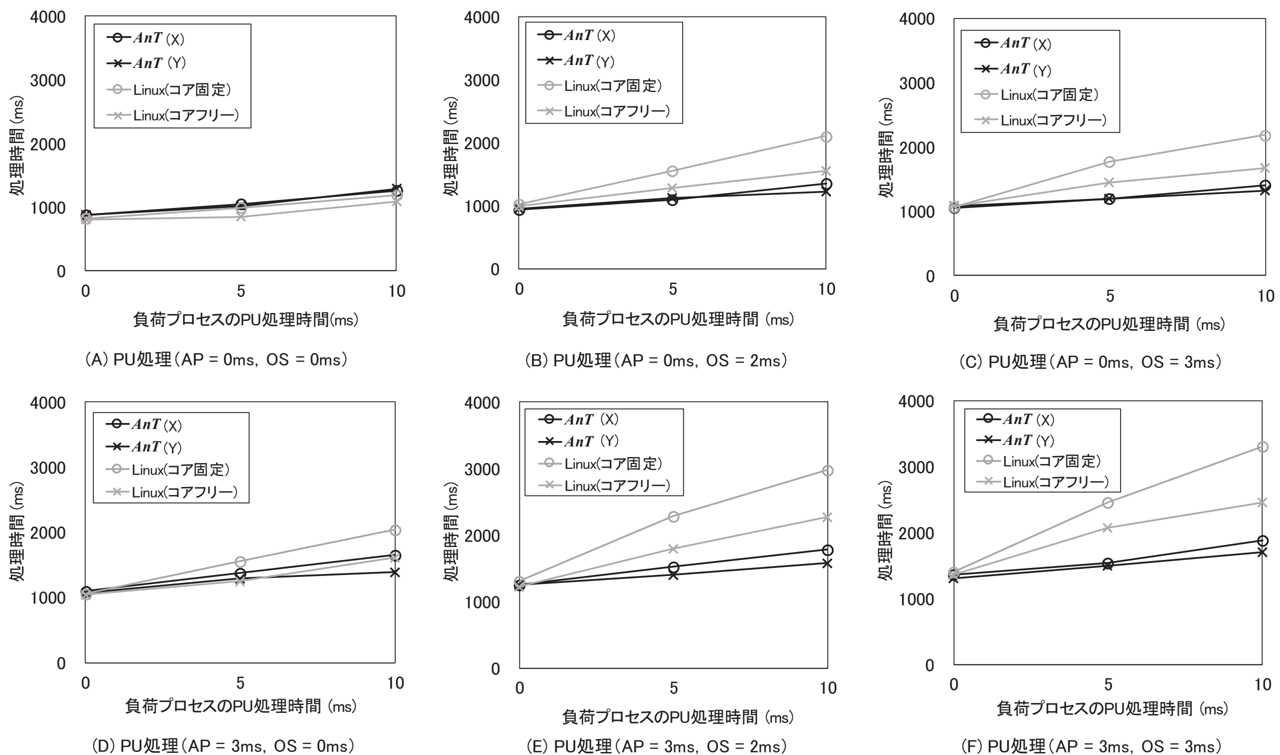

図 **11**　処理時間（負荷プロセス 1 個の場合）

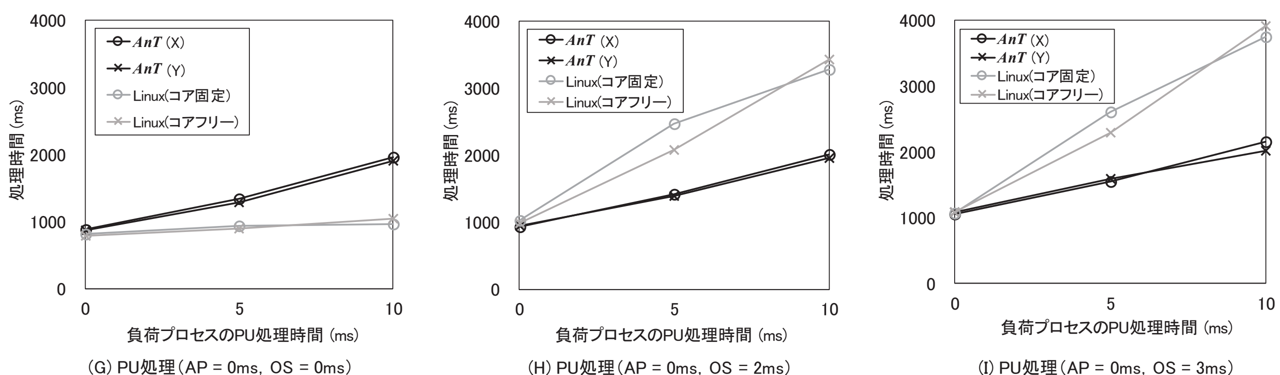

図 **12**　処理時間（負荷プロセス 2 個の場合）

AP 処理と OS 処理の PU 負荷の大小が分散効果に与える影響を明らかにするため，AP 処理と OS 処理のそれぞれに PU 処理（0，2，3ms）を追加した．また，コアの PU 負荷が大きい場合の分散効果を評価するために，負荷プロセスをコア当たり 1 個または 2 個共存走行させる．負荷プロセスは，PU 処理（0，5，10ms）と WAIT 状態（1ms）を繰り返し実行する．各処理の優先度は（OS 処理）>（負荷プロセス）>（AP プロセス）であり，AP1 と AP2 の優先度は等しい．なお，負荷プロセスと AP2（繰り返し回数 n >>100）は，AP1（繰り返し回数 n =100）が処理を終えるまで処理を継続し，AP1 の処理時間を測定した．

4.2.2　結果と考察

負荷プロセスが 1 個の場合の処理時間を図 **11** に示す．図 11 より以下がわかる．

（1）PU 処理が 0ms の場合（A），AP 処理と OS 処理のプロセッサ利用量は非常に少ない．このため，これらの処理は，負荷プロセスが待ち状態（1ms）の間に行われる．したがって，処理分散効果はなく，処理時間はコア間通信を必要としない Linux が ***AnT*** より短い．ただし，その差は非常に小さく，負荷プロセスの PU 処理時間が 0ms の時，55 ms（6%）程度である．

（2）PU 処理（OS）が増える場合（(A) から (C)），Linux の処理時間が大きく増加し，***AnT*** より長くなる．これに対し，***AnT*** の処理時間はほとんど増加しない．したがって，***AnT*** は Linux に比べ処理分散効果が大きい．例えば負荷プロセスの PU 処理時間が 10ms の時，(A) から (C)

にかけての処理時間の増加は，***AnT***（Y）では 25 ms と短い．一方，Linux（コア固定）では 1,010 ms と長い．この傾向は，PU 処理（AP）が 3ms も同様である．

（3）***AnT*** の分散形態（X，Y）による処理時間の影響は，PU 処理（AP）が大きい場合（3ms,（D）から（F））に明らかである．AP の処理分散を行っている分散形態（Y）の処理時間は，AP の処理分散を行っていない分散形態（X）より短い．例えば（D）の場合，処理時間の増加は，分散形態（X）（552 ms）よりも分散形態（Y）（320 ms）の方が 232 ms 短い．

（4）Linux の分散形態（コア固定，コアフリー）による処理時間の影響は，（A）から（F）のすべての場合で明らかである．コアを自由に使える分散形態（コアフリー）の処理時間は，走行するコアを指定する分散形態（コア固定）より短い．例えば（F）の場合，処理時間の増加は，分散形態（コア固定）（1,909 ms）よりも分散形態（コアフリー）（708 ms）の方が 1,201 ms 短い．

負荷プロセスが 2 個の場合の処理時間を図 **12** に示す．

（5）負荷プロセスが多いため，項目（1）や（2）の傾向が強く表れる．例えば項目（2）の傾向について，負荷プロセスの PU 処理時間が 10ms の時，（G）から（I）にかけての処理時間の増加は，***AnT***（Y）では 935 ms である．一方，Linux（コア固定）では 2,672 ms と非常に長い．

（6）Linux の分散形態（コア固定，コアフリー）の処理時間は，項目（4）の傾向と異なり，分散形態（コア固定）の方が早くなっている．プロセス数が多いため，コアを自由に利用できる効果がなくなった．例えば（I）の場合，処理時間の増加は，分散形態（コアフリー）（2,831 ms）よりも分散形態（コア固定）（2,672 ms）の方が 159 ms 短い．

5. おわりに

マイクロカーネル構造 OS である ***AnT*** とモノリシックカーネル構造 OS である Linux について，AP プロセスと OS 処理の分散形態の違いを述べた．また，実測によりサーバプログラム間通信，および AP と OS の処理分散効果を評価した結果を述べた．

サーバプログラム間通信の評価により，IPI の送受信が必要となる別コアへの処理依頼を行う場合でも，***AnT*** のサーバプログラム間通信のオーバヘッドは約 5 μs に抑えられることを述べた．

また，マイクロカーネル構造 OS では，AP と OS を各コアに自由に分散できることで，負荷を分散できることを示した．評価により，OS の PU 処理時間が増加した際に，Linux は処理時間が大きく増加するのに対し，***AnT*** では処理分散効果によりほとんど増加しないことを述べた．***AnT*** の分散形態の比較として，AP の処理分散を行っている分散形態（Y）の処理時間は，AP の処理分散を行っていない分散形態（X）より短いことを述べた．（G），（H），および（I）では，***AnT*** の処理分散効果が大きく表れることを述べた．

参考文献

[1] Bricker, A., Gien, M., Guillemont, M., Lipkis, J., Orr, D. and Rozier, M.,“Architectural issues in microkernel-based operating systems: The CHORRUS experience, ” *Computer Communications*, Vol. 14, No. 6, pp. 347–357 (1991).

[2] J. Liedtke,“Towards Real Microkernels, ” Communications of The ACM, vol. 39, issue 9, pp. 70–77 (1996).

[3] Hartig, H., Hohmuth, M., Liedtke, J., Wolter, J. and Schonberg, S., “The performance of microkernel-based systems, ” Proc. 16th ACM Symp. Operating System Principles, pp.66–77 (1997).

[4] 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).

[5] D.L.Black, D.B.Golub, D.P.Julin, R.F.Rashid, R.P.Draves, R.W.Dean, A.Forin, J.Bar-rera, H.Tokuda, G.Malan, and D.Bohman.,“Microkernel Operating System Architecture and March, ” Journal of Information Processing, Vol. 14, No. 4, pp. 442–453 (1992).

[6] 佐古田 健志，山内 利宏，谷口 秀夫，“高スループットを実現する OS 処理分散法の実現，”マルチメディア，分散，協調とモバイル (DICOMO2013) シンポジウム論文集，Vol. 2013, No. 2, pp. 1663–1670 (2013).

[7] Hamad, M., Schlatow, J., Prevelakis, V., Ernst, R., “A communication framework for distributed access control in microkernel-based systems, ” In 12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT16) pp. 11–16 (2016).

[8] 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の開発，”情処学研報，vol. 2012–DPS–150, no. 37, pp. 1–8, (2012).

[9] Baumann, A., Barham, P., Dagand, P. E., Harris, T., Isaacs, R., Peter, S.Singhania, A., “The multikernel: a new OS architecture for scalable multicore systems, ” In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles pp. 29–44 (2009).

[10] Matarneh, R.,“Multi Microkernel Operating System for MUlti-Core Processor, ” Journal of Computer Science, Vol. 5, No. 7, pp. 493–500 (2009).

[11] 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，Vol. J93-D, No. 10, pp. 1977–1989 (2010).

© 2019 Information Processing Society of Japan