情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2011-OS-118 No.20
2011/7/28

ブロック単位入出力を API とするファイル管理機能の提案

枡 田 圭 祐[†1] 谷 口 秀 夫[†1]

既存の多くのオペレーティングシステムのファイル管理機能は，ファイルのデータを入出力する際のデータ複写がオーバヘッドになっている．また，ファイルの同一ブロック内において別領域を更新する際は，プロセス単位で排他制御を行う必要があり，オーバヘッドが大きい．そこで，これらの問題への対処法として，ブロックのデータを複数のプロセス間で共有するファイル管理機能を提案する．また，提案機能を ***AnT*** オペレーティングシステムに実現し，FreeBSD（4.3-RELEASE）のファイル管理機能と比較する．さらに，メモリマップドファイルとの比較も行なう．

Proposal of the File Management Mechanism to Use Block as I/O API

Keisuke Masuda[†1] and Hideo Taniguchi[†1]

In the file management mechanism of many existing operating systems, the processing of file I/O causes large overhead by the data copy. In addition, in the updating of another area in same blocks of the file, exclusion control causes large overhead, because it is needed in each process. To solve these problems, we propose the file management mechanism, which shares the block data between many processes. This paper presents the implementation of our proposal mechanism which is applied to the ***AnT*** operating system, and the evaluation which compares the overhead of the file management between the ***AnT*** operating system and FreeBSD(4.3-RELEASE). In addition, this paper also reports the evaluation of the memory mapped file.

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

1. はじめに

多くのサービス処理はデータの操作を行うため，ファイル管理機能は重要な機能である．しかし，既存の多くのオペレーティングシステム（以降，OS）のファイル管理機能は，ファイルのデータを入出力する際にデータ複写が多数発生し，オーバヘッドが大きい．また，2 個以上のプロセスが同一ファイルの同一ブロック内の別領域を更新する際は，データの読み込み，更新，および書き出しの間，プロセス単位で排他制御を行う必要があるため，オーバヘッドが大きい．

そこで，これらの問題への対処法として，データ入出力の単位をブロック単位とし，複数のプロセス間でブロックデータを共有するファイル管理機能を提案する．提案機能は，データ入出力時におけるデータ複写回数を削減し，データ複写によるオーバヘッドを削減できる．また，2 個以上のプロセスが同一ファイルの同一ブロック内の別領域を更新する際に，データの読み込み，更新，および書き出しの間，プロセス単位で排他制御を行う必要がなくなり，オーバヘッドを削減できる．また，提案機能を ***AnT*** オペレーティングシステム[1]（以降，***AnT***）に実現し，FreeBSD（4.3-RELEASE）のファイル管理機能と比較する．さらに，メモリマップドファイル（以降，MMF）との比較も行なう．

2. 既存の問題

2.1 データ複写

既存の多くの OS のファイル管理機能では，データの入出力時にデータ複写が多数発生する問題がある．データ複写を**図 1** に示す．プロセス A がファイルのブロックデータの読み込みを要求した際，ファイル管理機能がブロックデータをキャッシュする領域（以降，ブロックキャッシュ）に要求したブロックデータが存在しない場合は，外部記憶装置（以降，ディスク）からブロックキャッシュへのブロックデータの複写（読み込み）(1) が発生する．次に，ブロックキャッシュからプロセス A の使用する実メモリ空間へのブロックデータの複写 (2) が発生する．プロセス A がブロックデータのブロックキャッシュへの書き出しを要求した際，プロセス A の使用する実メモリ空間からブロックキャッシュへのブロックデータの複写 (3) が発生する．さらに，プロセス A がブロックデータのディスクへの書き出しを要求した際，ブロックキャッシュからディスクへのブロックデータの複写（書き出し）(4) が発生する．したがって，ディスク上のデータをプロセスが更新するには，最大 4 回のデータ複写が発生してしまう．

© 2011 Information Processing Society of Japan

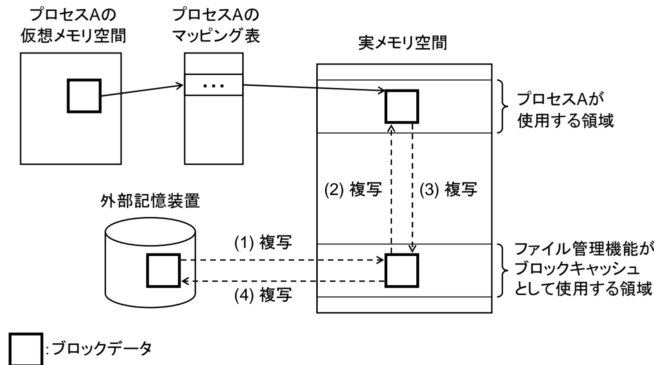

図 1 データ複写の発生

2.2 同一ブロック内の別領域の更新

複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合の問題について述べる．2 個のプロセスが同一ファイルの同一ブロックを読み込む様子を**図 2** に示し，同一ブロック内の別領域を更新する様子を**図 3** に示す．ここでは，ディスクとのデータ複写については議論しない．図 2 に示すように，まず最初にプロセス A によるブロックデータの読み込み (1) とプロセス B によるブロックデータの読み込み (2) が行なわれる．その後，図 3 に示すように，プロセス A によるブロックデータの更新 (3) とプロセス B によるブロックデータの更新 (4) が行なわれ，プロセス A によるブロックデータのブロックキャッシュへの書き出し (5) とプロセス B によるブロックデータのブロックキャッシュへの書き出し (6) が行なわれる．処理 (5) と処理 (6) は順序関係により，先に実行された処理による更新は無効になってしまう問題がある．

この問題を解決するには，読み込みと更新と書き出しの処理をプロセス単位で排他制御する必要がある．したがって，複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合のオーバヘッドは大きい．

3. ブロック単位入出力を API とするファイル管理機能

3.1 基 本 方 式

ブロック単位入出力を API とするファイル管理機能は，以下の考え方に基づく．

(1) データ入出力の単位をブロック単位とする．

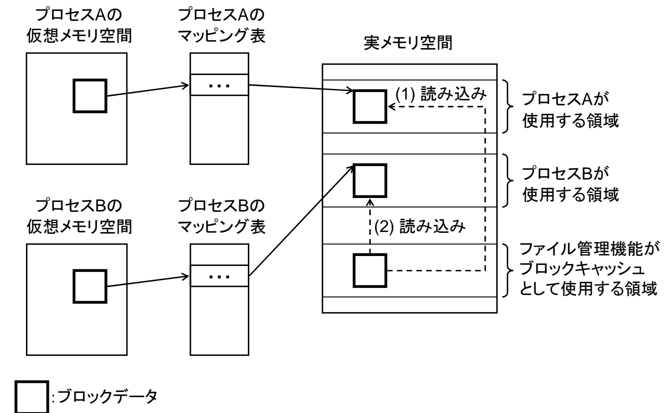

図 2 同一ブロックの読み込み

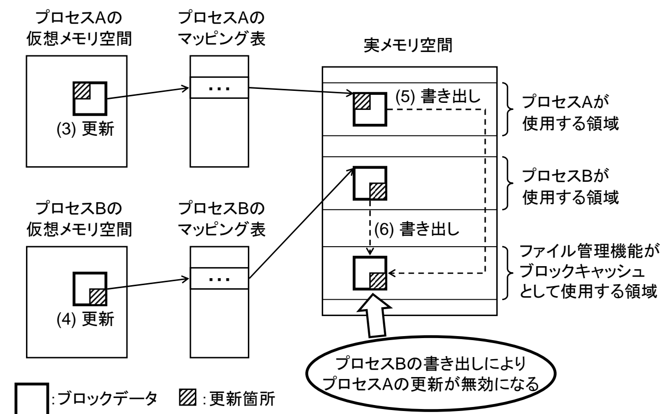

図 3 同一ブロック内の別領域の更新

(2) 複数のプロセスとブロックキャッシュ間でブロックデータを共有する．

上記の考え方に基づくファイル管理機能について，基本方式を**図 4** に示す．プロセス A がブロックデータの読み込みを要求した際，ブロックキャッシュに要求したブロックデータが存在しない場合は，ディスクからブロックキャッシュにブロックデータを複写 (1) する．次に，プロセス A のマッピング表にブロックデータの実アドレスを書き込む (2)．これにより，プロセス A がブロックデータを更新 (3) した際，プロセス A はブロックキャッシュ

© 2011 Information Processing Society of Japan

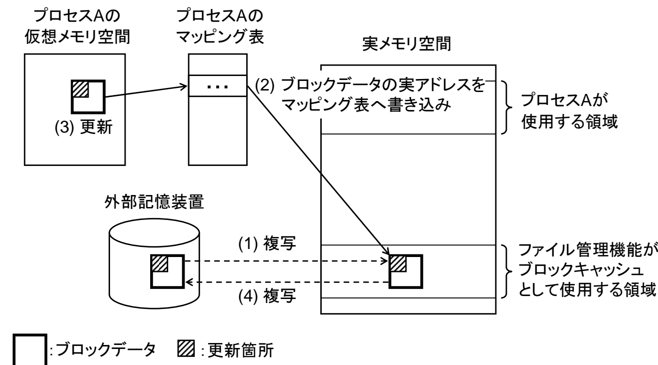

図 4 基本方式

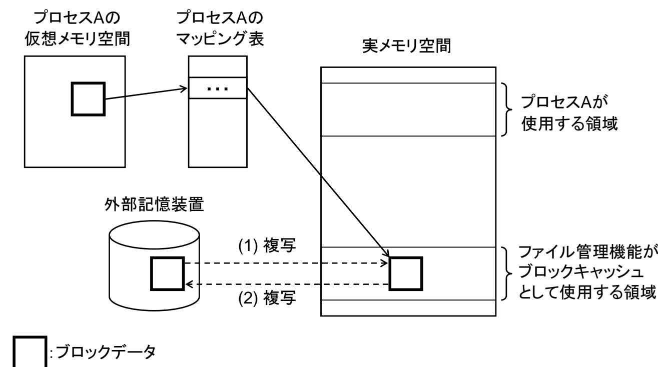

図 5 提案機能によるデータ複写の発生

内のブロックデータを直接更新することになる．したがって，ブロックキャッシュへの書き出し API は不要である．プロセス A がブロックデータのディスクへの書き出しを要求した際，ブロックキャッシュからディスクへブロックデータを複写 (4) する．

3.2 既存問題への対処

3.2.1 データ複写

提案機能は，データ入出力時にデータ複写が発生する問題に対処できる．提案機能において発生するデータ複写を**図 5** に示す．プロセス A がファイルのブロックデータの読み込みを要求した際，ブロックキャッシュに要求したブロックデータが存在しない場合は，ディスクからブロックキャッシュへのブロックデータの複写（読み込み）(1) が発生する．次に，プロセス A のマッピング表にブロックデータの実アドレスを書き込むことで，プロセス A はブロックデータを参照できる．したがって，図 1 の (2) に示したブロックキャッシュからプロセス A の使用する実メモリ空間へのブロックデータの複写は発生しない．さらに，図 1 の (3) に示したプロセス A の使用する実メモリ空間からブロックキャッシュへのブロックデータの複写も発生しない．また，プロセス A がブロックデータのディスクへの書き出しを要求した際，ブロックキャッシュからディスクへのブロックデータの複写（書き出し）(2) が発生する．したがって，ディスク上のデータをプロセスが更新するには，最大 2 回のデータ複写が発生する．つまり，提案機能によるデータ複写回数は，既存 OS のファイル管理機能のデータ複写回数（最大 4 回）よりも少なく，オーバヘッドは小さいといえる．

3.2.2 同一ブロック内の別領域の更新

提案機能は，複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合の問題に対処できる．図 2 や図 3 と同様に，提案機能において，2 個のプロセスが同一ファイルの同一ブロックを読み込む様子を**図 6** に示し，同一ブロック内の別領域を更新する様子を**図 7** に示す．図 6 に示すように，まず最初にプロセス A によるブロックデータの読み込み (1) とプロセス B によるブロックデータの読み込み (2) が行なわれる．なお，ブロックデータの読み込みは，各プロセスのマッピング表にブロックデータの実アドレスを書き込むことで行なわれる．その後，図 7 に示すように，プロセス A によるブロックデータの更新 (3) とプロセス B によるブロックデータの更新 (4) が行なわれる．提案機能では，各プロセスとブロックキャッシュ間でブロックデータを共有するため，処理 (3) でプロセス A がブロックデータを更新すると，プロセス B とブロックキャッシュでも同時に更新が反映される．同様に，処理 (4) でプロセス B がブロックデータを更新すると，プロセス A とブロックキャッシュでも同時に更新が反映される．つまり，プロセス A とプロセス B の更新は両方とも反映される．したがって，図 3 の (5) と (6) に示した各プロセスによるブロックデータのブロックキャッシュへの書き出しは発生しない．

したがって，提案機能は，読み込みと更新の処理をプロセス単位で排他制御する必要がなく，複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合のオーバヘッドは小さいといえる．

3.3 特　　徴

既存 OS のファイル管理機能と提案機能の比較を**表 1** に示す．提案機能は，ファイルの

ⓒ 2011 Information Processing Society of Japan

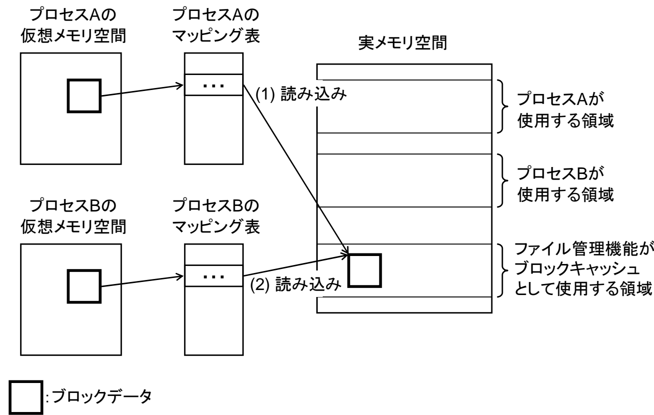

図 6　提案機能による同一ブロックの読み込み

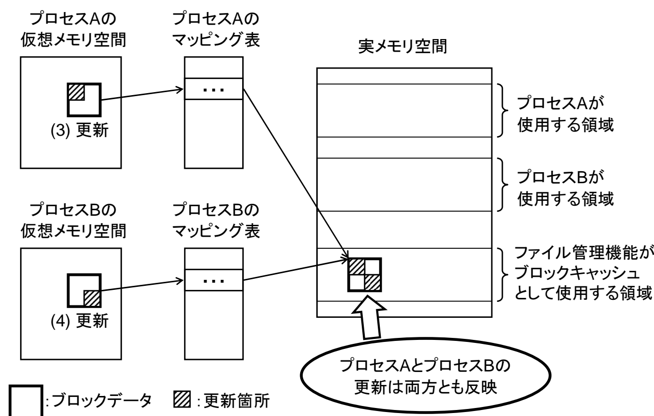

図 7　提案機能による同一ブロック内の別領域の更新

データを入出力する際のデータ複写オーバヘッドが小さく，かつ複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する際のオーバヘッドが小さいという利点を持つ．しかし，入出力するデータのサイズがブロック単位に制限されるという欠点を持つ．

表 1　既存 OS のファイル管理機能と提案機能の比較

	入出力データ単位	データ複写	同一ブロック内の別領域更新
既存 OS のファイル管理機能	1Byte	オーバヘッド大	オーバヘッド大
ブロック単位入出力を API とするファイル管理機能	4KByte	オーバヘッド小	オーバヘッド小

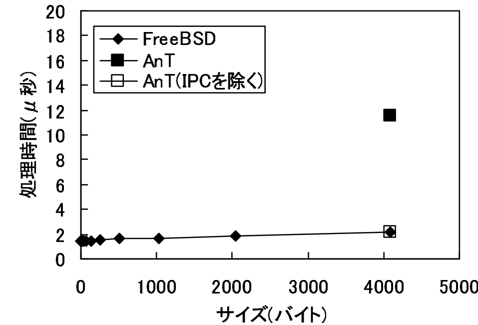

図 8　1 個のプロセスによるデータ読み込み時間

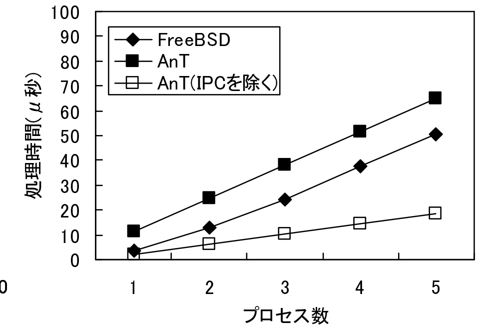

図 9　複数のプロセスによるデータ読み込み時間

4. 評　　価

4.1 環　　境

提案機能を ***AnT*** オペレーティングシステムに実現し，FreeBSD（4.3-RELEASE）のファイル管理機能と比較する．両 OS を Pentium4（2.8GHz）の計算機で走行させ，評価した．***AnT*** はマイクロカーネル構造 OS[2)3)4)] であり，ファイル管理機能やディスクドライバといった多くの OS 機能をプロセスとして実現[5)] している．

4.2 基 本 性 能

4.2.1 読み込み性能

1 個のプロセスが 4kByte 以下のデータを読み込む処理時間を**図 8** に示す．ただし，ブロックキャッシュからの読み込み時間である．図 8 より，以下のことがわかる．

(1)　4kByte のデータを読み込む処理時間は，***AnT*** は 11.50μ 秒，FreeBSD は 2.21μ 秒である．***AnT*** の処理時間が長い原因は，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．具体的には，1 回のデータ読み込みに要するプロセス間通信の処理時間は 10.43μ 秒[6)] であり，4kByte のデータ読み込みに要

© 2011 Information Processing Society of Japan

する処理時間の 90(= 10.43/11.50 × 100)%を占めている．

(2)　OS 構造（マイクロカーネル構造）の影響を排除し，提案機能そのものの性能を考察するため，プロセス間通信に要する処理時間を 0 として議論する．この場合，新たにマッピング表の書き替えに要する処理時間（0.64μ 秒）とシステムコール発行 1 回分の処理時間（0.48μ 秒）が発生する．このため，4kByte のデータ読み込みに要する処理時間は 9.31（= 10.43 − 0.64 − 0.48）μ 秒短くなり，2.19（= 11.50 − 9.31）μ 秒になる．したがって，FreeBSD の処理時間（2.21μ 秒）より短くなる．

次に，複数のプロセスが同一の 4kByte のデータを読み込む処理時間を**図 9** に示す．ただし，ブロックキャッシュからの読み込み時間である．図 9 より，以下のことがわかる．

(1)　FreeBSD と ***AnT*** において，処理時間の増加率はほぼ等しい．しかし，***AnT*** の処理時間は FreeBSD よりも約 14μ 秒長い．これは，***AnT*** がマイクロカーネル構造 OS であり，プロセス間通信が発生するためである．

(2)　先の記述と同様に，提案機能そのものの性能を考察するため，プロセス間通信に要する処理時間を 0 として議論する．この場合，4kByte のデータ読み込みに要する処理時間は 9.31μ 秒短くなる．これにより，提案機能の処理時間は短くなり，増加率も FreeBSD より低くなる．したがって，提案機能の読み込み性能は高いといえる．

4.2.2　書き出し性能

1 個以上のプロセスが 4kByte 以下のデータを書き出す処理時間（ただし，ブロックキャッシュへの書き出し時間）について議論する．FreeBSD の場合，2μ 秒から数十 μ 秒である．これに対し，***AnT*** の場合，各プロセスとブロックキャッシュ間でブロックデータを共有するため，プロセスがデータを更新すると同時に，更新した内容がブロックキャッシュに反映される．つまり，***AnT*** ではブロックキャッシュへの書き出しを行う処理は存在しない．したがって，書き出す処理時間は 0 といえ，書き出し性能は高いといえる．

4.3　同一ブロック内の別領域更新の性能

4.3.1　処 理 流 れ

複数のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合について，処理の流れを**図 10** に示す．

図 10 において，(A) は，既存 OS のファイル管理機能において，各プロセスが独立に読み込んだメモリ上のデータ更新とディスク I/O を行なう処理の流れである．これに対し，(B) は，各プロセスが独立に読み込んだメモリ上のデータ更新を行なうものの，同期を取り，最終処理として 1 個のプロセス（以降，最終プロセス）がディスク I/O を行なう処理の流れ

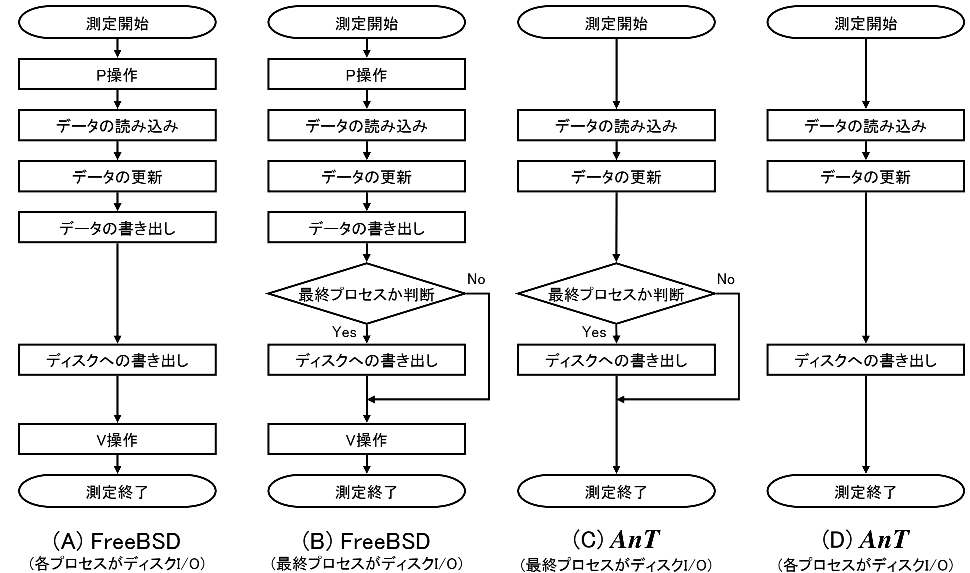

図 10　複数のプロセスによる別領域更新の処理流れ

である．(C) と (D) は，提案機能において，上記 (B) と (A) に対応する処理を行なう流れである．(A)(B) と (C)(D) の大きな違いを以下に示す．

(1)　既存 OS のファイル管理機能（FreeBSD）では，データの読み込み，更新，および書き出しの間，排他制御（P 操作と V 操作）を行なう必要がある．一方，提案機能（***AnT***）では，排他制御を行なう必要がない．

(2)　既存 OS のファイル管理機能（FreeBSD）では，データの書き出しを行なう必要がある．一方，提案機能（***AnT***）では，データの書き出しを行なう必要がない．

また，MMF 機能を利用した場合，MMF の処理流れは図 10 の ***AnT*** の場合（(C) と (D)）と同様になる．

4.3.2　性　　能

図 10 の処理において，各処理の時間を分析し考察する．1 個のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合の処理時間を**図 11** に示す．なお，プロセスはブロックキャッシュからブロックデータを読み込み，1Byte を更新する．図 11 の (A)〜(D)

© 2011 Information Processing Society of Japan

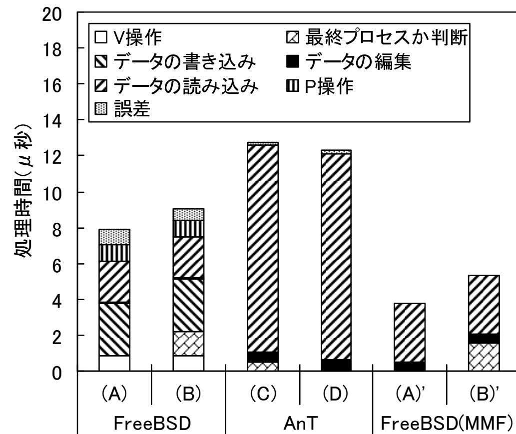

図 11 1 個のプロセスによる別領域更新の処理時間 (実 I/O 時間を除く)

は，図 10 の (A)〜(D) に対応している．また，FreeBSD と ***AnT*** における CPU の処理量を明確にするため，測定結果からディスク I/O 時間を除いている．図 11 より，以下のことがわかる．

(1) (A) と (D) より，各プロセスがディスク I/O を行なう場合，***AnT*** における処理時間は，FreeBSD よりも 4.37μ 秒長い．これは，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．具体的には，1 回のデータ読み込みに要するプロセス間通信の処理時間は 10.43μ 秒であり，(D) の処理時間（12.28μ 秒）の 84($= 10.43/12.28 \times 100$)%を占めている．

(2) (B) と (C) より，最終プロセスがディスク I/O を行なう場合，***AnT*** における処理時間は，FreeBSD よりも 3.73μ 秒長い．これは，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．具体的には，1 回のデータ読み込みに要するプロセス間通信の処理時間は 10.43μ 秒であり，(C) の処理時間（12.75μ 秒）の 81($= 10.43/12.75 \times 100$)%を占めている．

(3) 先の記述と同様に，提案機能そのものの性能を考察するため，プロセス間通信に要する処理時間を 0 として議論する．この場合，(D) の処理時間は 9.31μ 秒短くなり，2.97（$= 12.28 - 9.31$）μ 秒になる．したがって，(A) の処理時間（7.91μ 秒）より短くなる．

(4) 先の記述と同様に，提案機能そのものの性能を考察すると，(C) の処理時間も 9.31μ 秒短くなり，3.44（$= 12.75 - 9.31$）μ 秒になる．これは，(B) の処理時間（9.02μ 秒）より短くなる．したがって，1 個のプロセスが同一ファイルの同一ブロック内の別領域を更新する場合，提案機能の性能は高いといえる．

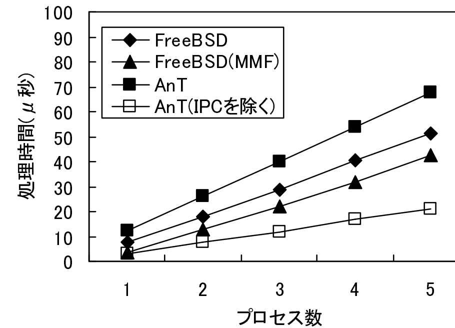

図 12 複数のプロセスによる別領域更新の処理時間 (各プロセスがディスク I/O)(実 I/O 時間を除く)

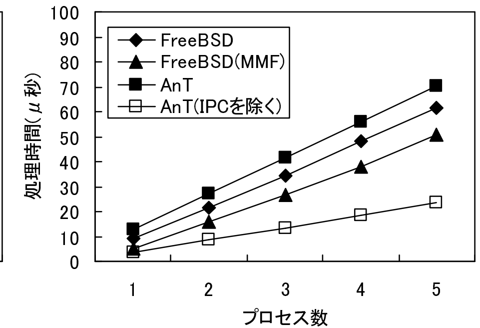

図 13 複数のプロセスによる別領域更新の処理時間 (最終プロセスがディスク I/O)(実 I/O 時間を除く)

次に，図 10 の (A) と (D) の場合の処理時間を**図 12** に示し，(B) と (C) の場合の処理時間を**図 13** に示す．なお，各プロセスはブロックキャッシュからブロックデータを読み込み，それぞれ 1Byte を更新する．また，FreeBSD と ***AnT*** における CPU の処理量を明確にするため，測定結果からディスク I/O 時間を除いている．図 12 より，以下のことがわかる．

(1) FreeBSD と ***AnT*** において，処理時間の増加率は ***AnT*** の方が高く，プロセス数の増加に伴い，FreeBSD と ***AnT*** の処理時間の差は約 3μ 秒ずつ広がっている．これは，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．

(2) 先の記述と同様に，提案機能そのものの性能を考察するため，プロセス間通信に要する処理時間を 0 として議論する．この場合，1 回のデータ読み込みに要する処理時間は 9.31μ 秒短くなる．これにより，提案機能の処理時間の増加率は FreeBSD よりも低くなる．したがって，各プロセスがディスク I/O を行なう場合の同一ブロック内の別領域更新処理において，提案機能の性能は高いといえる．

また，図 13 より，以下のことがわかる．

(1) FreeBSD と ***AnT*** において，処理時間の増加率はほぼ等しい．しかし，***AnT*** における処理時間は，FreeBSD よりも約 7μ 秒長い．これは，***AnT*** がマイクロカーネル構造

© 2011 Information Processing Society of Japan

OS であり，プロセス間通信が発生するためである．なお，既存のマイクロカーネル構造 OS では，データの書き出しを行なう必要があり，***AnT*** の場合に加え，データの読み込み，更新，および書き出しの処理をプロセス単位で排他制御する必要がある．このため，既存のマイクロカーネル構造 OS では，別領域更新に要する処理時間が非常に長くなる．しかし，提案機能により，マイクロカーネル構造 OS でもモノリシックカーネル構造 OS と同等の性能が得られる．

(2)　先の記述と同様に，提案機能そのものの性能を考察すると，1 回のデータ読み込みに要する処理時間は 9.31μ 秒短くなる．これにより，提案機能の処理時間の増加率は FreeBSD よりも低くなる．したがって，最終プロセスがディスク I/O を行なう場合の同一ブロック内の別領域更新処理において，提案機能の性能は高いといえる．

最後に，MMF との比較を行なう．図 11 において，(A)' は MMF を利用した (A) に相当する処理の場合であり，(B)' は MMF を利用した (B) に相当する処理の場合である．図 11 より，***AnT*** はマイクロカーネル構造であるために，処理時間が長い．しかし，提案機能そのものの性能に着目すると，いずれの場合も 0.36μ 秒または 2.39μ 秒 ***AnT*** が短く，提案機能の性能は良いといえる．

図 12 より，以下のことがわかる．

(1)　MMF を利用した FreeBSD と ***AnT*** において，処理時間の増加率は ***AnT*** の方が高く，プロセス数の増加に伴い，FreeBSD と ***AnT*** の処理時間の差は約 4μ 秒ずつ広がっている．これは，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．

(2)　提案機能と MMF は，どちらもプロセス間でブロックデータを共有しており，メモリアクセスの形でブロックデータを読み込み，更新する．このため，先の記述と同様に，提案機能そのものの性能を考察した場合，MMF を利用した FreeBSD と同等の処理時間となるはずである．しかし，実際には MMF を利用した FreeBSD の処理時間の増加率の方が高く，プロセス数の増加に伴い，処理時間の差は約 4μ 秒ずつ広がっている．これは，MMF を利用した FreeBSD と提案機能において，プロセス数が 1 個増加するのに伴い発生する，データ読み込み，データ更新，およびプロセス切り替えの性能差によるものであると考えられる．それぞれの性能差は $1.11(=3.3-2.19)\mu$ 秒，$-0.14(=0.49-0.63)\mu$ 秒，および $3.06(=4.73-1.67)\mu$ 秒であり，合計は $4.03(=1.11-0.14+3.06)\mu$ 秒となる．

また，図 13 より，以下のことがわかる．

(1)　FreeBSD と ***AnT*** において，処理時間の増加率は ***AnT*** の方が高く，プロセス数の増加に伴い，FreeBSD と ***AnT*** の処理時間の差は約 3μ 秒ずつ広がっている．これは，***AnT*** がマイクロカーネル構造 OS であり，データ読み込み時にプロセス間通信が発生するためである．

(2)　提案機能と MMF は，どちらもプロセス間でブロックデータを共有しており，メモリアクセスの形でブロックデータを読み込み，更新する．このため，先の記述と同様に，提案機能そのものの性能を考察した場合，MMF を利用した FreeBSD と同等の処理時間となるはずである．しかし，実際には MMF を利用した FreeBSD の処理時間の増加率の方が高く，プロセス数の増加に伴い，処理時間の差は約 5μ 秒ずつ広がっている．これは，MMF を利用した FreeBSD と提案機能において，プロセス数が 1 増加するのに伴い発生する，データ読み込み，データ更新，最終プロセスか否かの判断，およびプロセス切り替えの性能差によるものであると考えられる．それぞれの性能差は 1.11μ 秒，-0.14μ 秒，$1.09(=1.56-0.47)\mu$ 秒，および 3.06μ 秒であり，合計は $5.12(=1.11-0.14+1.09+3.06)\mu$ 秒となる．

提案機能と MMF は，どちらもプロセス間でブロックデータを共有しており，メモリアクセスの形でブロックデータを読み込み，更新する．このため，先の記述と同様に，提案機能そのものの性能を考察した場合，処理時間は，MMF を利用した既存 OS と同等となる．しかし，提案機能と MMF では，以下 2 つの大きな違いがある．

(1)　提案機能では，プロセス毎にブロックデータの保護属性（読み込み専用，読み書き可能）を指定することができる．しかし，MMF では，プロセス毎にブロックデータの保護属性を指定することができない．

(2)　提案機能と MMF は，どちらもブロックキャッシュ内のブロックデータを取得できる．しかし，提案機能はブロックデータをブロックキャッシュに登録できるが，MMF はブロックデータをブロックキャッシュに登録できない．

5. おわりに

データ入出力の単位をブロック単位とし，複数のプロセス間でブロックデータを共有するファイル管理機能を提案した．提案機能は，データ入出力時におけるデータ複写回数を削減し，データ複写によるオーバヘッドを削減できる．また，同一ファイルの同一ブロック内において別領域を更新する際に，データの読み込み，更新，および書き出しの間，プロセス単位で排他制御を行う必要がなく，オーバヘッドを削減できる．

提案機能を実現した ***AnT*** と既存 OS である FreeBSD のファイル管理機能との比較では，***AnT*** の方が処理時間が長く，性能は低かった．しかし，これは ***AnT*** がマイクロカーネル

© 2011 Information Processing Society of Japan

構造 OS であり，プロセス間通信が発生しているためであった．OS 構造（マイクロカーネル構造）の影響を排除し，提案機能そのものの性能を考察した場合，FreeBSD のファイル管理機能よりも性能は高いといえた．また，提案機能そのものの性能は MMF と同等である．しかし，提案機能は MMF と異なり，プロセス毎にブロックデータの保護属性を指定でき，さらに，ブロックデータをブロックキャッシュに登録できる．

参 考 文 献

1) 谷口秀夫, 乃村能成, 田端利宏, 安達俊光, 野村裕佑, 梅本昌典, 仁科匡人, “適応性と堅牢性をあわせもつ ***AnT*** オペレーティングシステム,” 情報処理学会研究報告, vol.2006-OS-103, pp.71-78 (2006.07).
2) J. Liedtke, “Toward real microkernels,” Commun. ACM, vol.39, no.9, pp.70-77, 1996.
3) A.S. Tanenbaum, J.N. Herder, and H. Bos, “Can we make operating systems reliable and secure?,” IEEE Computer Magazine, vol.39, no.5, pp.44-51, 2006.
4) D.L. Black, D.B. Golub, D.P. Julin, R.F. Rashid, R.P. Draves, R.W. Dean, A. Forin, J. Barrera, T. Hideyuki, G.R. Malan, and D. Bohman, “Microkernel operating system architecture and mach,” J. Inf. Process., vol.14, no.4 (19920315), pp.442-453, 1992.
5) 野村祐佑, 谷口秀夫, “***AnT*** におけるファイル管理サーバの設計,” 情報処理学会研究報告, vol.2008-OS-109, pp.53-60 (2008.08).
6) 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.1977-1989 (2010.10).

© 2011 Information Processing Society of Japan