Vol.2012-ARC-200 No.4
Vol.2012-OS-121 No.4
2012/5/7

ファイルサイズの拡縮が可能なメモリ上ファイル操作機能の提案

枡田 圭祐[1]　谷口 秀夫[1]

概要：既存の多くのオペレーティングシステムのファイル操作機能は，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．この問題への対処として，複数のプロセス間でファイルのデータを共有する方法がある．この代表的な機能として，メモリマップドファイル機能がある．しかし，メモリマップドファイル機能は，ファイルサイズを拡縮できない．そこで，ファイルサイズの拡縮も可能なメモリ上のファイル操作機能（オンメモリファイル機能）を提案する．また，オンメモリファイル機能を ***AnT*** オペレーティングシステムに実現し，FreeBSD（6.3-RELEASE）のファイル操作機能と比較する．

Proposal of Advanced Memory Mapped File that Can Change Size of File

Keisuke MASUDA[1]　Hideo TANIGUCHI[1]

Abstract: In the file operation mechanism of many existing operating systems, the processing of file I/O causes large overhead by the data copy. To solve this problem, there is the file operation mechanism, which shares the data of file between many processes. This representative mechanism is memory mapped file. However, memory mapped file can not change size of file. To solve this problem, we propose advanced memory mapped file that can change size of file. This paper presents the implementation of advanced memory mapped file which is applied to the ***AnT*** operating system, and the evaluation which compares the overhead of the file operation between the ***AnT*** operating system and FreeBSD(6.3-RELEASE).

1. はじめに

高い適応性と堅牢性を実現するオペレーティングシステム（以降，OS）の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，例外処理や割込処理といった最小限の OS 機能をカーネルとして実現し，ファイル管理やディスクドライバなどの処理をプロセス（以降，OS サーバ）として実現するプログラム構造である．

しかし，マイクロカーネル構造では，OS サーバ間の通信が頻発するため，FreeBSD のようなモノリシックカーネル構造の OS に比べ，データ複写が多発し，性能が低下する．そこで，この性能低下を抑制する機構が必要であり，OS サーバ間の高速な通信機構 [4] が提案されている．一方，ファイル操作においても高速化が必要である．モノリシックカーネル構造の OS において，複数のプロセス間でファイルのデータを共有し，データ複写を抑制する方法がある．この代表的な機能として，メモリマップドファイル機能 [5] がある．しかし，メモリマップドファイル機能は，ファイルのデータを参照することに重点がおかれており，メモリ上でファイルのデータ操作を自由に行なうことができない．例えば，ファイルサイズを拡縮できない．そこで，ファイルサイズの拡縮も可能なメモリ上のファイル操作機

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

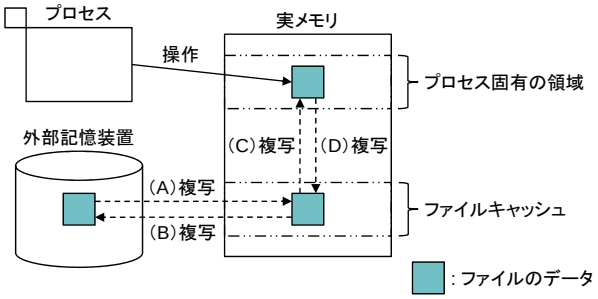

図 1 通常入出力機能におけるデータ操作処理

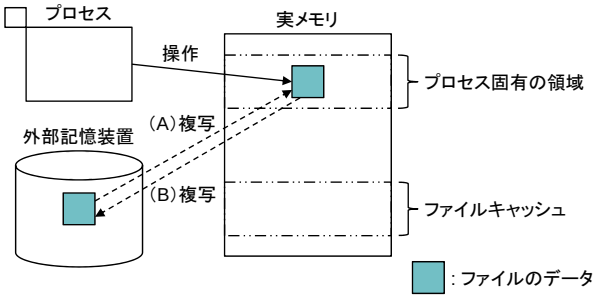

図 2 直接入出力機能におけるデータ操作処理

能（以降，オンメモリファイル機能）を提案する．また，オンメモリファイル機能を ***AnT*** オペレーティングシステム [4],[6]（以降，***AnT***）に実現し，FreeBSD（6.3-RELEASE）のファイル操作機能と比較する．

2. 既存のファイル操作機能

2.1 通常入出力機能

通常入出力機能は，ファイルのデータを入出力する際にデータ複写が多発する．この様子を**図 1** に示す．

図 1 に示すように，プロセスが外部記憶装置 [7]~[9] 上のデータを更新する際は，4 種類のデータ複写が発生する．このうち，データ複写（A）とデータ複写（B）は，ファイルキャッシュを利用することで，データ複写回数を削減することができる．しかし，データ複写（C）とデータ複写（D）は，プロセスがファイルのデータを更新する度に毎回発生する．したがって，プロセスがファイルのデータを更新する際，少なくとも 2 回の実メモリ間データ複写が発生してしまう．

2.2 直接入出力機能

直接入出力機能は，ファイルのデータを入出力する際にファイルキャッシュを利用しない．この様子を**図 2** に示す．

図 2 に示すように，プロセスが外部記憶装置上のデータを更新する際に，2 種類のデータ複写が発生する．これらは，ファイルのデータを更新する度に発生する．このため，直接入出力機能は，サービス処理そのものに独自のキャッシュ機構を持つ場合（例えば，データベース処理）に多く

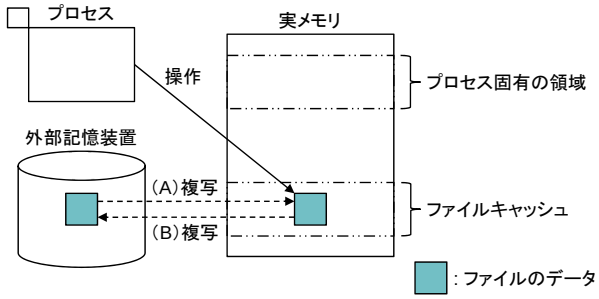

図 3 メモリマップドファイル機能におけるデータ操作処理

利用されている．したがって，独自のキャッシュ機構を持たない場合は利用されていない．

2.3 メモリマップドファイル機能

メモリマップドファイル（MMF）機能は，ファイルのデータをメモリ上に読み出し操作できる機能であり，複数のプロセス間でデータを共有することもできる．なお，この機能は，ファイルのデータを参照することに重点がおかれている．MMF 機能の様子を**図 3** に示す．

図 3 に示すように，プロセスが外部記憶装置上のデータを更新（上書き）する際に，2 種類のデータ複写が発生する．これらのデータ複写は，ファイルキャッシュを利用することで，データ複写回数を削減することができる．このため，MMF 機能は，通常入出力機能の場合に比べ，データ複写回数を削減できる．しかし，MMF 機能は，i ノードの更新を行なわないため，以下の問題がある．

(1) ファイルを参照しても，参照日付が更新されない．
(2) データを更新しても，更新日付が更新されない．
(3) データの追加や削除ができない．つまり，ファイルサイズの拡縮ができない．

したがって，MMF 機能は，ファイルの参照のみに利用されており，ファイルの更新には利用されていない．例えば，複数のプロセス間で共有されるプログラムテキストを読み込む際に利用されている．

2.4 比較

通常入出力機能，直接入出力機能，および MMF 機能の比較を**表 1** に示す．

通常入出力機能は，1Byte 単位でファイルのデータを入出力できる利点を持つ．しかし，ファイルのデータを入出力する際に実メモリ間データ複写が必ず発生する欠点を持つ．

直接入出力機能は，ファイルのデータを入出力する際に実メモリ間データ複写が発生しない．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欠点を持つ．

MMF 機能は，ファイルのデータを入出力する際に実メモリ間データ複写が発生しない．しかし，i ノードの更新を行なわないため，ファイルサイズの拡縮ができない欠点

表 1 既存のファイル操作機能の比較

	実メモリ間データ複写	i ノードの更新	ファイルキャッシュ	入出力データ単位
通常入出力機能	2 回	可能	有効	1Byte
直接入出力機能	0 回	可能	無効	ブロック
MMF 機能	0 回	不可能	有効	ページ

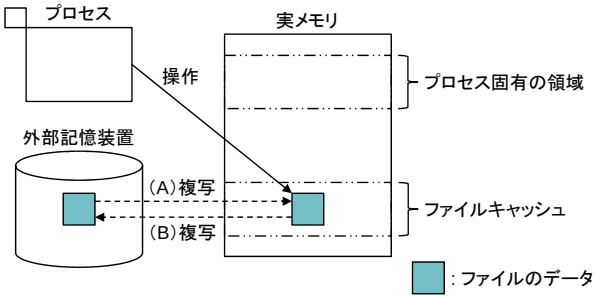

図 4 オンメモリファイル機能におけるデータ操作処理

を持つ．また，入出力するデータのサイズがページ単位に制限される欠点もある．

したがって，MMF 機能の問題を克服した機能を実現できれば，独自のキャッシュ機構を持たない多くのサービス処理において，通常入出力機能を用いたファイル操作よりも高速にファイルの参照や更新を行なうことができる．

3. オンメモリファイル機能

3.1 基本機構

文献 [10] の機能を拡充し，MMF 機能の問題を解決するオンメモリファイル（OMF）機能を提案する．

OMF 機能は，MMF 機能と同様に，ファイルのデータをメモリ上のデータのように扱うことができ，複数のプロセス間でデータを共有する．OMF 機能の様子を**図 4** に示す．プロセスが OMF 機能を用いてファイルのデータを操作する際，以下のデータ複写が発生する．

(1) プロセスがファイルのデータを読み込む場合
　外部記憶装置からファイルキャッシュへのデータ複写（A）が発生する．
(2) プロセスがファイルのデータを書き出す場合
　ファイルキャッシュから外部記憶装置へのデータ複写（B）が発生する．

このように，プロセスが外部記憶装置上のデータを更新する際に，2 種類のデータ複写が発生する．これらのデータ複写は，ファイルキャッシュを利用することで，データ複写回数を削減することができる．このため，OMF 機能は，通常入出力機能を用いたファイル操作に比べ，データ複写回数を削減できる．

一方，OMF 機能は，MMF 機能と異なり，i ノードの更新を行なう．i ノードの更新は，メモリ上のデータを外部記憶装置に書き出す際に行なう．このため，以下のことができる．

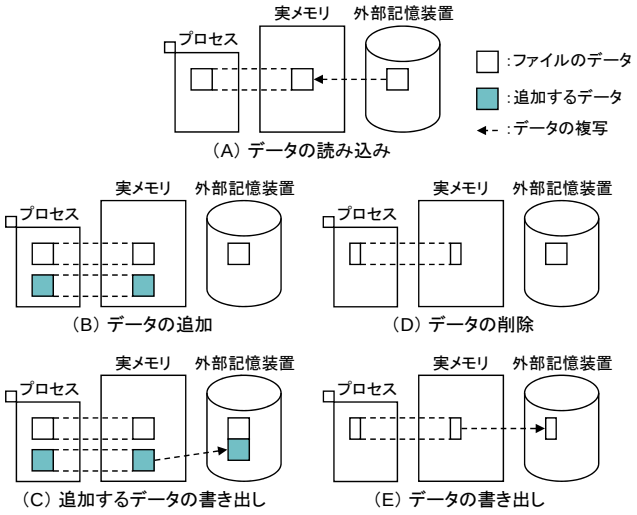

図 5 ファイルサイズの拡縮が発生する際の処理

(1) ファイルを参照した際，参照日付を更新する．
(2) ファイルを更新した際，更新日付を更新する．
(3) データの追加や削除ができる．つまり，ファイルサイズを拡縮できる．

さらに，OMF 機能は，ファイルサイズの拡縮を可能にする．データの追加や削除によりファイルサイズの拡縮が発生する様子を**図 5** に示す．ファイルサイズ拡大の場合とファイルサイズ縮小の場合について，以下に説明する．

(1) ファイルサイズ拡大の場合
　(a) ファイルのデータを読み込む．（図 5 の（A））．
　(b) 追加するデータのサイズ分の実メモリ領域を確保する．この領域に，追加するデータを格納する．（図 5 の（B））．
　(c) 追加するデータを外部記憶装置に書き出す．（図 5 の（C））．この際，書き出しサイズ，書き出し開始位置，およびファイルサイズを拡縮するか否かのフラグをもとに，拡大後のファイルサイズを算出し，i ノードの更新を行なう．また，この際，追加するデータを格納するための外部記憶装置上の領域を確保する．
(2) ファイルサイズ縮小の場合
　(a) ファイルのデータを読み込む．（図 5 の（A））．
　(b) 読み込んだファイルのデータの一部を削除する．（図 5 の（D））．
　(c) ファイルのデータを外部記憶装置に書き出す．（図 5 の（E））．この際，書き出しサイズ，書き出し開始位置，およびファイルサイズを拡縮するか否かのフ

表 2 提供インタフェース

機能	形式
外部記憶装置からのデータの読み込み	readblock(fd, size, offset); fd: ファイル記述子 size: 読み込みサイズ offset: 読み込み開始位置
外部記憶装置へのデータの書き出し	writeblock(fd, *buff, size, offset, flag); fd: ファイル記述子 *buff: 書き出すデータを格納する領域 size: 書き出しサイズ offset: 書き出し開始位置 flag: ファイルサイズを拡縮するか否か

ラグをもとに，縮小後のファイルサイズを算出し，i ノードの更新を行なう．また，この際，不要になった外部記憶装置上の領域を解放する．

3.2 提供インタフェース

OMF 機能が提供するインタフェースを**表 2** に示す．readblock() は，外部記憶装置からのデータの読み込み，writeblock() は，外部記憶装置へのデータの書き出しである．

なお，readblock() と writeblock() を用いて，1Byte 単位で入出力できる機能をライブラリとして実現できる．このインタフェースを readbyte() と writebyte() と名付ける．これらの処理は，通常入出力機能の処理に比べ，データ複写回数やカーネルの呼び出し回数が異なる．特に，ライブラリとして実現するため，多くの場合において，カーネルの呼び出しが発生しない．

データ読み込みの場合，通常入出力機能では，データ格納域を確保し，read() を発行する．これに対し，OMF 機能では，readbyte() を発行する．したがって，データ複写回数は，通常入出力機能では 1 回，OMF 機能では 0 回である．また，カーネルの呼び出し回数は，通常入出力機能では 2 回，OMF 機能では最大 1 回である．OMF 機能では，ブロックへの最初のデータ読み込みの場合にのみカーネルの呼び出しが発生する．

ファイルのデータ更新の場合，通常入出力機能では，データ格納域を確保し，read() を発行し，データを更新し，write() を発行する．これに対し，OMF 機能では，readbyte() を発行し，writebyte() でデータを更新する．この際，writebyte() の処理は，ライブラリ内に閉じている．したがって，データ複写回数は，通常入出力機能では 3 回，OMF 機能では 1 回である．また，カーネルの呼び出し回数は，通常入出力機能では 3 回，OMF 機能では最大 1 回である．

次に，fread() と fwrite() を用いたデータ読み込みとデータ更新について議論する．

通常入出力機能では，一度の read() でデータを大量に読み込み，以降の複数回の fread() に答えることができる．この際，fread() の処理は，ライブラリ内に閉じている．つまり，データ読み込みの場合，データ格納域を確保し，fread() を発行することでデータを読み込める．この場合，データ複写回数は，最低 1 回，最大 2 回である．また，カーネルの呼び出し回数は，最低 1 回，最大 2 回である．一方，OMF 機能では，データ複写回数は 0 回であり，カーネルの呼び出し回数は最大 1 回である．

また，通常入出力機能では，複数回の fwrite() をメモリ上に保持し，一度の write() でデータを書き出すことができる．この際，fwrite() の処理は，ライブラリ内に閉じている．つまり，ファイルのデータ更新の場合，データ格納域を確保し，fread() を発行し，データを更新し，fwrite() を発行することで，データを更新することができる．この場合，データ複写回数は，最低 3 回，最大 5 回である．また，カーネルの呼び出し回数は，最低 1 回，最大 3 回である．一方，OMF 機能では，データ複写回数は 1 回であり，カーネルの呼び出し回数は最大 1 回である．

上記により，いずれの場合もデータ複写回数やカーネルの呼び出し回数は，OMF 機能を用いた方式が通常入出力機能を用いた方式より少ないことがわかる．

3.3 比較

既存のファイル操作機能と OMF 機能の比較を**表 3** に示す．OMF 機能は，以下の特徴を持つ．

(1) 通常入出力機能に比べ，ファイルのデータを入出力する際に実メモリ間データ複写が発生しない．
(2) 直接入出力機能に比べ，ファイルキャッシュを利用するため，独自のキャッシュ機構を持たないプロセスにとって有効である．
(3) MMF 機能に比べ，i ノードの更新を行なうため，ファイルサイズの拡縮ができ，ファイルの参照だけでなく更新にも有効である．

3.4 有効な処理例

OMF 機能が有効な処理の例を以下に示す．

(例 1) ファイルのデータの一部だけをランダムアクセスする処理の場合，ファイルのデータをメモリの読み書きと同様に操作できる．
(例 2) 複数のプロセスが同一ファイルの別々の領域を操作する処理の場合，各プロセス間でファイルのデータを共有しており，プロセスが更新した内容は他のプロセスでも即座に反映されるため，排他制御を考慮する必要がない．
(例 3) 複数のプロセスがサイズの大きな同一ファイルを操作する処理の場合，各プロセス間でファイルのデータを共有するため，メモリの利用効率が良い．

表 3 ファイル操作機能の比較

	実メモリ間データ複写	i ノードの更新	ファイルキャッシュ	入出力データ単位
通常入出力機能	2 回	可能	有効	1Byte
直接入出力機能	0 回	可能	無効	ブロック
MMF 機能	0 回	不可能	有効	ページ
OMF 機能	0 回	可能	有効	ブロック（1Byte）

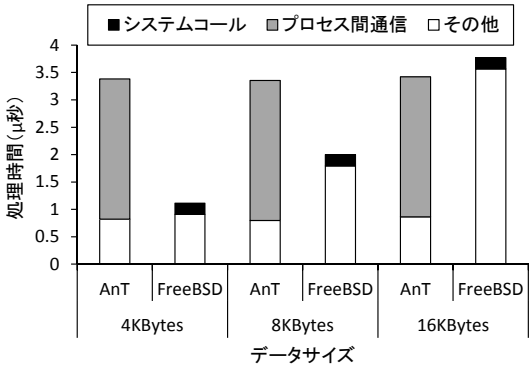

図 6 読み込み処理時間

4. 評価

4.1 観点

3.3 節で述べたように，直接入出力機能は，ファイルキャッシュを利用していない．また，MMF 機能は，i ノードを更新できない．したがって，i ノードを更新可能でファイルキャッシュが有効である通常入出力機能と OMF 機能について性能を比較する．

4.2 測定環境

OMF 機能を ***AnT*** に実現し，FreeBSD（6.3-RELEASE）の通常入出力機能と比較する．両 OS を Core i7-2600（3.4GHz）の計算機で走行させた．なお，***AnT*** は，マイクロカーネル構造 OS 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している．

4.3 基本性能

4.3.1 読み込み性能

プロセスが 4Kbytes, 8KBytes, および 16KBytes のデータを読み込むのに要する処理時間を**図 6** に示す．ただし，実 I/O 時間は同等と判断し，この時間は，ファイルキャッシュからの読み込み処理時間である．図 6 より，以下のことがわかる．

(1) 4KBytes のデータの読み込み処理時間は，***AnT*** で 3.38μ 秒であり，FreeBSD で 1.11μ 秒である．***AnT*** の読み込み処理時間が FreeBSD よりも長い原因は，***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．なお，1 回のデータ読み込みに対するプロセス間通信の処理時間は，2.56μ 秒である．これは，4KBytes のデータ読み込みに要する処理時間の 75(= 2.56/3.38 × 100)% を占めている．

(2) ***AnT*** の読み込み処理時間は，読み込むデータのサイズによらず，一定である．これは，***AnT*** の OMF 機能では，データ読み込み時に実メモリ間データ複写が発生しないためである．

(3) 16KBytes のデータの読み込み処理時間は，***AnT*** で 3.42μ 秒であり，FreeBSD で 3.77μ 秒である．この場合，***AnT*** の読み込み処理時間は，FreeBSD よりも短くなり，***AnT*** の読み込み性能は高いといえる．したがって，16KBytes 以上のファイルを読み込む場合，マイクロカーネル構造 OS であっても，モノリシックカーネル構造 OS よりも高い性能を実現できるといえる．

なお，OMF 機能をモノリシックカーネル構造 OS に実現した場合，図 6 のプロセス間通信の時間が 0 になる．したがって，OMF 機能は直接入出力機能より高速である．

4.3.2 書き出し性能

プロセスが 4KBytes, 8KBytes, および 16KBytes のデータを書き出すのに要する処理時間について議論する．ただし，ファイルキャッシュへの書き出し処理時間である．FreeBSD の場合，それぞれ 0.50μ 秒，1.48μ 秒，および 3.64μ 秒である．これに対し，***AnT*** の場合，各プロセスとファイルキャッシュ間でファイルの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，更新した内容がファイルキャッシュに反映される．つまり，***AnT*** では，ファイルキャッシュへの書き出しを行なう処理は存在しない．したがって，書き出し処理時間は 0 といえ，書き出し性能は高いといえる．

4.3.3 読み書きの性能

データの読み書きに要する時間について議論する．具体的には，文字置換の処理を利用する．プロセスがファイルのデータ（小文字のアルファベット）を大文字のアルファベットに置換する処理の流れを**図 7** に示す．図 7 において，（A）は，通常入出力機能（FreeBSD）において，プロセスが読み込んだデータ内の文字を小文字から大文字へ置換する処理の流れである．（B）は，OMF 機能（***AnT***）において，（A）に相当する処理を行なう処理の流れである．（A）と（B）の大きな違いを以下に示す．

(1) 通常入出力機能（FreeBSD）では，データの書き出し

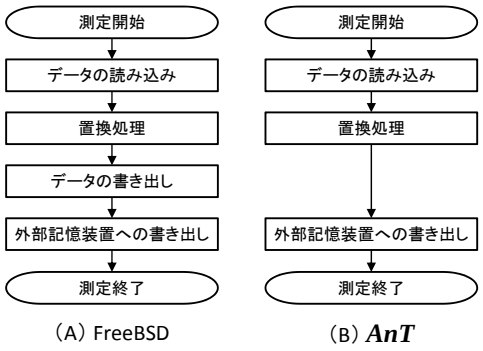

図 7 文字置換の処理流れ

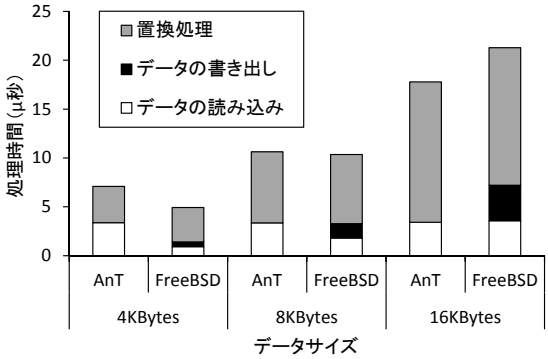

図 8 文字置換の処理時間

を行なう必要がある．一方，OMF 機能（***AnT***）では，データの書き出しを行なう必要はない．

図 7 の処理において，各処理の時間を分析し考察する．プロセスが 4KBytes，8KBytes，および 16KBytes のデータ（小文字のアルファベット）を大文字のアルファベットに置換する際の処理時間を**図 8** に示す．なお，FreeBSD と ***AnT*** における CPU の処理量を明確にするため，実 I/O 時間を除外している．図 8 より，以下のことがわかる．

(1) 4KBytes のデータを置換する場合の処理時間は，***AnT*** で 7.08μ 秒であり，FreeBSD で 4.93μ 秒である．***AnT*** の処理時間が FreeBSD よりも長い原因は，***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 2.56μ 秒のプロセス間通信が発生するためである．

(2) 8KBytes のデータを置換する場合の処理時間は，***AnT*** で 10.63μ 秒であり，FreeBSD で 10.30μ 秒である．***AnT*** と FreeBSD の処理時間が同程度になっている原因は，***AnT*** のデータ読み込み処理時間（3.36μ 秒）が，FreeBSD のデータ読み込み処理時間（1.79μ 秒）とデータ書き出し処理時間（1.48μ 秒）の合計（3.27μ 秒）と同程度になっているためである．

(3) 16KBytes のデータを置換する場合の処理時間は，***AnT*** で 17.78μ 秒であり，FreeBSD で 21.28μ 秒である．この場合，***AnT*** における文字置換の処理時間は，FreeBSD よりも短くなり，***AnT*** における文字置換の性能は高いといえる．したがって，データサイズが 16Kbytes 以上の文字を置換する場合，マイクロカーネル構造 OS であっても，モノリシックカーネル構造 OS よりも高い性能を実現できるといえる．

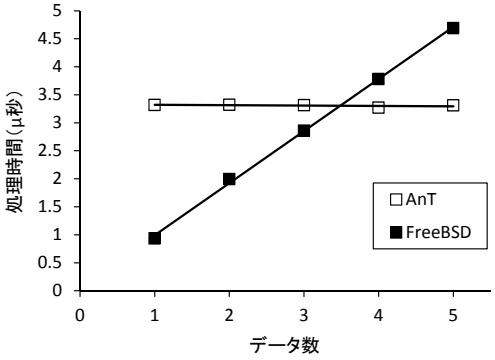

図 9 ファイルの一部をランダムアクセスする処理時間

なお，OMF 機能をモノリシックカーネル構造 OS に実現した場合，図 8 のプロセス間通信の時間（2.56μ 秒）が 0 になる．したがって，4KBytes のデータを置換する場合であっても，OMF 機能は直接入出力機能より高速である．

4.4 ランダムアクセス性能

ファイルのデータの一部だけをランダムアクセスするのに要する処理時間について議論する．プロセスがファイルのデータにランダムアクセスしてデータを 1Byte 読み込むのに要する処理時間を**図 9** に示す．なお，FreeBSD と ***AnT*** にける CPU の処理量を明確にするため，実 I/O 時間を除外している．図 9 より，以下のことがわかる．

(1) 通常入出力機能（FreeBSD）では，アクセスするデータ数が増えるにつれて，処理時間が約 1μ 秒ずつ単調増加している．この原因は，通常入出力機能では，連続でないデータを読み込む場合，1 個のデータにつき，読み込み開始位置の設定とデータの読み込みを行なう必要があるためである．

(2) OMF 機能（***AnT***）では，アクセスするデータ数によらず，データ読み込み時間は一定である．この原因は，OMF 機能では，ファイルのデータをメモリ上のデータのように扱うことができ，メモリへのアクセスと同様にデータを読み込めるためである．したがって，最初にファイルのデータを全てマッピングしておけば，データ数によらず，一定の処理時間でデータを読み込むことができる．

(3) 1 個のデータをランダムアクセスして読み込む場合の処理時間は，***AnT*** で 3.32μ 秒であり，FreeBSD で 0.94μ 秒である．***AnT*** の処理時間が FreeBSD よりも長い原因は，***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 2.56μ 秒のプロセス間通信が発生するためである．

(4) 4 個のデータをランダムアクセスして読み込む場合

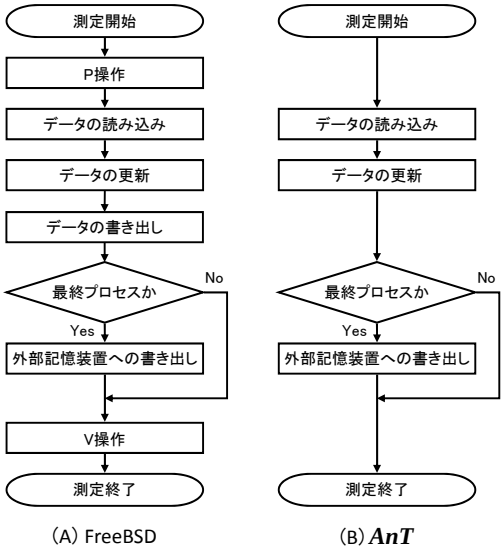

図 10 同一ファイル内の別領域更新の処理流れ

の処理時間は，***AnT*** で 3.27μ 秒であり，FreeBSD で 3.78μ 秒である．この場合，***AnT*** におけるランダムアクセスでのデータ読み込み処理時間は，FreeBSD よりも短くなり，***AnT*** におけるランダムアクセス性能は高いといえる．したがって，4 個以上のデータに対してランダムアクセスする場合，マイクロカーネル構造 OS であっても，モノリシックカーネル構造 OS よりも高い性能を実現できるといえる．

なお，OMF 機能をモノリシックカーネル構造 OS に実現した場合，図 9 のプロセス間通信の時間（2.56μ 秒）が 0 になる．したがって，1 個以上のデータに対してランダムアクセスする場合，OMF 機能は直接入出力機能より高速である．

4.5 同一ブロック内の別領域更新の性能

4.5.1 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，通常入出力機能において，複数のプロセスが独立に読み込んだメモリ上のデータを更新する処理の流れである．この際，各プロセス間で同期を取り，最終処理として 1 個のプロセス（最終プロセス）が外部記憶装置への書き出しを行なっている．また，（B）は，OMF 機能において，（A）に相当する処理を行なう処理の流れである．（A）と（B）の大きな違いを以下に示す．

(1) 通常入出力機能（FreeBSD）では，データの書き出しを行なう必要がある．一方，OMF 機能（***AnT***）では，データの書き出しを行なう必要はない．

(2) 通常入出力機能（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．しかし，OMF 機能（***AnT***）では，排他制御を行なう必要はない．

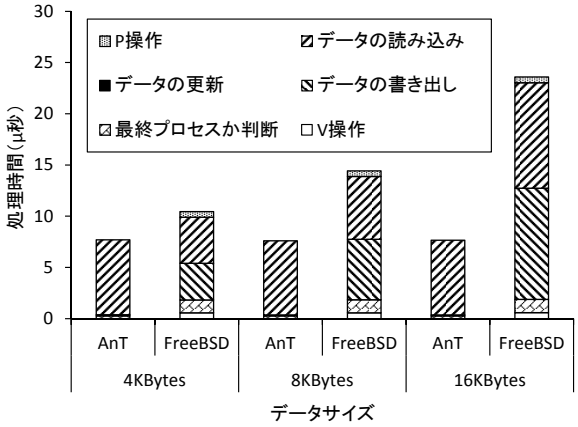

図 11 1 個のプロセスによる別領域更新の処理時間

4.5.2 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，ファイルキャッシュからファイルのデータをそれぞれ 4KBytes，8KBytes，16KBytes 読み込み，1Byte を更新し，それぞれ 4Kbytes，8KBytes，16KBytes 書き出す．また，***AnT*** と FreeBSD における CPU の処理量を明確にするため，実 I/O 時間を除いている．図 11 より，以下のことがわかる．

(1) データのサイズによらず，OMF 機能（***AnT***）の処理時間は一定である．この原因は，OMF 機能では，データ読み込み時にデータ複写が発生しないためである．

(2) 通常入出力機能（FreeBSD）の場合，読み込むデータサイズに比例し，処理時間が増加している．この原因は，通常入出力機能では，データの読み込み時とデータの書き出し時にデータ複写が発生するためである．

(3) データのサイズによらず，***AnT*** の処理時間は，FreeBSD よりも短く，***AnT*** における同一ファイル内の別領域更新の性能は高いといえる．したがって，1 個のプロセスが同一ファイル内の別々の領域を更新する場合，マイクロカーネル構造 OS は，モノリシックカーネル構造 OS よりも高い性能を実現できるといえる．

なお，OMF 機能をモノリシックカーネル構造 OS に実現した場合，図 11 のプロセス間通信の時間（2.56μ 秒）が 0 になる．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，OMF 機能は直接入出力機能より高速である．

次に，図 10 の処理において，読み込むデータサイズとプロセス数を変更した場合の処理時間を**図 12** に示す．なお，各プロセスは，ファイルキャッシュからファイルのデータをそれぞれ 4Kbytes，8KBytes，16KBytes 読み込み，1Byte を更新し，それぞれ 4Kbytes，8KBytes，16KBytes 書き出す．また，***AnT*** と FreeBSD における CPU の処理量を明

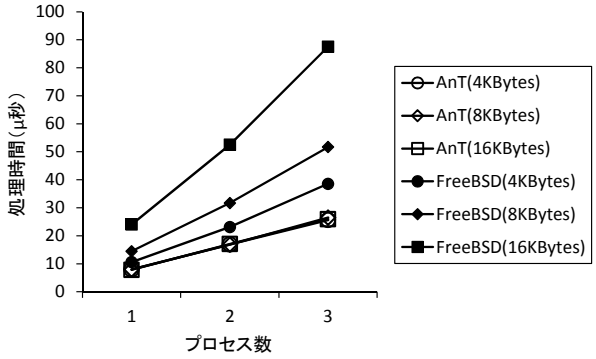

図 12 複数のプロセスによる別領域更新の処理時間

確にするため，実 I/O 時間を除いている．図 12 より，以下のことがわかる．

(1) データのサイズによらず，OMF 機能（***AnT***）の処理時間は一定である．この原因は，通常入出力機能では，データの読み込み時とデータの書き出し時にデータ複写が発生するためである．

(2) 全ての場合において，処理時間は，「図 11 に示した処理時間 × プロセス数 + プロセス切替え処理時間 ×(プロセス数 −1)」になっている．なお，プロセス切替え処理時間は，***AnT*** で 0.84μ 秒，FreeBSD で 2.70μ 秒である．

(3) データのサイズによらず, ***AnT*** の処理時間は, FreeBSD よりも短く，***AnT*** における同一ファイル内の別領域更新の性能は高いといえる．したがって，同一ファイル内の別々の領域を更新する場合，マイクロカーネル構造 OS であっても，モノリシックカーネル構造 OS よりも高い性能を実現できるといえる．

なお，OMF 機能をモノリシックカーネル構造 OS に実現した場合，図 12 のプロセス間通信の時間（2.56μ 秒 × プロセス数）が 0 になる．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，OMF 機能は直接入出力機能より高速である．

4.6 メモリの利用効率

通常入出力機能と OMF 機能のメモリの利用効率について議論する．

通常入出力機能の場合，実メモリ領域は，ファイルキャッシュとして利用する領域に加え，プロセスがデータを読み込むための領域がプロセスと同じ数だけ必要になる．

一方，OMF 機能の場合，複数のプロセス間でファイルのデータを共有するため，実メモリ領域は，ファイルキャッシュとして利用する領域のみで良い．

したがって，メモリの利用効率は，OMF 機能を用いた方式が通常入出力機能を用いた方式より良いことがわかる．

5. おわりに

ファイルサイズの拡縮が可能なメモリ上のファイル操作機能（オンメモリファイル機能：OMF 機能）を提案した．OMF 機能は，データ入出力時におけるデータ複写回数を削減できる．

マイクロカーネル構造を持つ ***AnT*** に OMF 機能を実現し，FreeBSD の通常入出力機能と比較評価した．読み込みと書き出しの基本性能は，データ長が短い場合（4KBytes と 8KBytes）は ***AnT*** の OMF 機能の性能が低い．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しているためである．なお，データ長が 16KBytes の場合は ***AnT*** の OMF 機能の性能が高い．また，ランダムアクセス性能ではデータ数が 4 個以上の場合，同一ファイル内の別領域更新の性能ではデータ長に関係なく，OMF 機能の性能が高い．さらに，メモリの利用効率では，OMF 機能を用いた方式が通常入出力機能を用いた方式より効率が良いことを明らかにした．

参考文献

[1] J. Liedtke, “Toward real microkernels,” Commun. ACM, vol.39, no.9, pp.70-77 (1996.09).

[2] 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 (2006.05).

[3] 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, pp.442-453 (1992.03).

[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.1977-1989 (2010.10).

[5] mmap(2) - Linux man page, http://linux.die.net/man/2/mmap

[6] 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).

[7] Alexander Thomasian, “Survey and Analysis of Disk Scheduling Methods,” ACM SIGARCH Computer Architecture News, vol.39, no.2, pp.8-25 (2011.05).

[8] Xiangyong Ouyang, David Nellans, Robert Wipfel, David Flynn, Dhabaleswar K. Panda, “Beyond Block I/O: Rethinking Traditional Storage Primitives,” 2011 IEEE 17th International Symposium on High Performance Computer Architecture, pp.301-311 (2011.02).

[9] Yoon Jae Seong, Eyee Hyun Nam, Jin Hyuk Yoon, Hongseok Kim, Jin-Yong Choi, Sookwan Lee, Young Hyun Bae, Jaejin Lee, Yookun Cho, Sang Lyul Min, “Hydra: A Block-Mapped Parallel Flash Memory Solid-State Disk Architecture,” IEEE Transactions on Computers, Volume 59, No.7, pp.905-921 (2010.07).

[10] 枡田圭祐, 谷口秀夫, “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,” 情報処理学会研究報告, vol.2011-OS-118, pp.1-8 (2011.07).