Vol.2015-OS-134 No.12
2015/8/4

マイクロカーネル向けゼロコピーファイル操作機能の評価

江原 寛人[1]　枡田 圭祐[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マイクロカーネル構造 OS は，OS 機能を OS サーバとして実現する．このため，サービス提供において，OS サーバも含めたプロセス間通信のオーバヘッドが性能低下の原因となる．ファイル操作では，OS サーバが管理するファイルキャッシュから AP プロセスへのデータ複写が発生してしまう．そこで，我々は，マイクロカーネル構造 OS 向けのファイル操作機能として，ファイル操作時のデータ複写回数とプロセス間通信回数を削減するオンメモリファイル機能を提案した．オンメモリファイル機能は，OS サーバが管理するファイルキャッシュを AP プロセスと共有する．本稿では，オンメモリファイル機能の評価として，メモリマップドファイル機能との比較結果とベンチマーク性能を報告する．

1. はじめに

Linux に代表される既存のファイル操作機能として，通常入出力機能がある．この機能は，read/write システムコール処理においてファイルキャッシュを利用し，外部記憶装置の入出力回数を削減する．しかし，この時，応用プログラム（以降，AP）のプロセスのメモリ空間とファイルキャッシュ間でデータ複写が発生する．これを防ぐ方法として，メモリマップドファイル機能 [1], [2] がある．この機能は，オンデマンドページング機能を実現するメモリ管理機能を利用している．

一方，マイクロカーネル構造 [3], [4], [5], [6], [7] は，メモリ管理，例外処理，および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といった大半の OS 機能をプロセス（以降，OS サーバ）として実現するプログラム構造である．このため，メモリマップドファイル機能をマイクロカーネル構造 OS に実現した場合，プロセス間通信が多発し，性能が低下する．

そこで，我々は，マイクロカーネル構造 OS 向けのファイル操作機能として，ファイル操作時のデータ複写回数とプロセス間通信回数を削減するオンメモリファイル（OMF: On-Memory File）機能を提案した [8]．本機能は，OS サーバが保持するファイルキャッシュをプロセスと共有する．

本稿では，オンメモリファイル機能とメモリマップドファイル機能との比較結果と，オンメモリファイル機能と通常入出力機能のベンチマーク性能を報告する．具体的には，オンメモリファイル機能を実現したマイクロカーネル構造を有する ***AnT*** [9] と，マイクロカーネル構造を有する MINIX（3.3.0）[10]，およびモノリシックカーネル構造を有する Linux（2.6.32）のファイル操作機能を比較する．

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

2. オンメモリファイル機能

2.1 基本方式

オンメモリファイル（OMF）機能 [8] は，以下の考え方に基づいている．

(1) AP プロセスとファイルキャッシュを共有する．
(2) オンデマンドページング機能を利用しない．
(3) OS サーバ間の高速な通信機構 [9] を利用する．

OMF 機能の基本方式を**図 1** に示す．OMF 機能は，AP プロセスとファイルキャッシュを共有する．つまり，AP プロセスのデータの参照や更新は，ファイルデータを格納する実メモリ空間のアドレスを AP プロセスの仮想メモリ空間へマッピングして行なう．したがって，通常入出力機能と異なり，AP プロセスのメモリ空間とファイルキャッシュ間でデータ複写は発生しない．なお，OMF 機能は，ファイルデータをブロック（4 KBytes）単位で管理する．

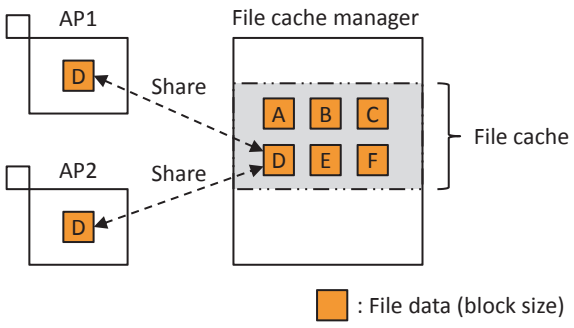


図 1　オンメモリファイル機能

2.2 処理流れと提供インタフェース

OMF 機能の処理流れを**図 2** に示す．AP プロセスがデータ参照システムコール（readblock()）を発行（A）すると，ファイル管理（File Manager）は，ファイルキャッシュを探索し，データを取得する．なお，ファイルキャッシュ内にデータが存在しない場合，ディスクドライバ（Disk Driver）へデータ読み込みを依頼（B）から（E）し，データを取得する．データ取得後，データ格納域を AP プロセスの仮想メモリ空間にマッピング（F）し，処理を終了する．

図 2 の（A）から（F）の各処理において，プロセス間通信が発生する．ただし，OMF 機能はオンデマンドページング機能を利用しないため，プロセス間通信回数はページ数に依存しない．また，同様の理由により，メモリ管理（Memory Manager）や例外処理（Exception Handler）を経由せず，直接ファイル管理へ処理を依頼する．したがって，プロセス間通信回数は，OMF 機能の方がマイクロカーネル構造におけるメモリマップドファイル（MMF）機能より少ない．さらに，OS サーバ間の高速な通信機構 [9] を利用するため，ファイル管理とディスクドライバの間のデータ複写は発生しない．

OMF 機能が提供するシステムコールのインタフェースを**表 1** に示す．readblock() は，ファイルキャッシュ内のファイルデータを AP プロセスの仮想メモリ空間にブロック単位でマッピングする．なお，ファイルキャッシュにファイルデータが存在しない場合，外部記憶装置からファイルキャッシュへファイルデータを読み込む．syncblock() は，ファイルキャッシュ内のファイルデータを外部記憶装置へ書き出す．

なお，OS サーバが保持するファイルキャッシュをプロセスと共有するため，プロセスがファイルデータを更新すると，ファイルキャッシュの内容も更新される．したがって，データをファイルキャッシュへ複写するインタフェース（通常入出力機能の write() に相当）は不要である．

システムコールの入出力単位は，ブロック単位である．このため，バイト単位でファイルデータを参照する機能をライブラリとして実現する．このインタフェースを readbyte() と名付け，**表 2** に示す．readbyte() は，ファイルのブロックデータを初めて参照する際に readblock() を発行し，ブロックデータをライブラリ内にバッファリングする．これにより，以降でデータを参照する際に，カーネルの呼び出しを削減できる．

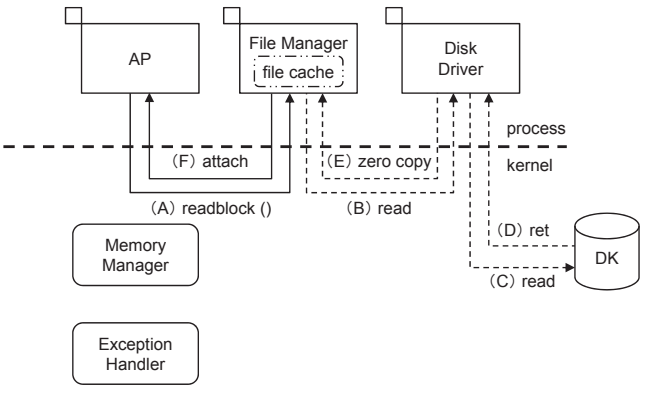


図 2 OMF 機能の処理流れ

表 1 システムコールのインタフェース

機能	形式
データの参照（ブロック単位）	readblock(fd, size, blkoff); fd: ファイル識別子 size: データのサイズ（ブロック単位） blkoff: 参照開始位置（ブロック単位）
外部記憶装置へのデータの書き出し	syncblock(fd, *buf, size, blkoff, flg); fd: ファイル識別子 *buf: データを格納するメモリ空間 size: データのサイズ（バイト単位） blkoff: 更新開始位置（ブロック単位） flg: ファイルサイズを拡縮するか否か

表 2 ライブラリコールのインタフェース

機能	形式
データの参照（バイト単位）	readbyte(fd, size, offset); fd: ファイル識別子 size: データのサイズ（バイト単位） offset: 参照開始位置（バイト単位）

3. 評価

3.1 観点と評価環境

OMF 機能と MMF 機能のファイルデータ参照性能を評価する．具体的には，単一の AP プロセスによる単一のファイルデータ参照について，マッピング処理とデータ参照処理の処理時間を比較する．

次に，OMF 機能と通常入出力機能のベンチマーク性能を評価する．具体的には，PostMark[11] と Bonnie[12] を用いて評価する．PostMark は，単一の AP プロセスが複数のファイルに対してファイルのオープン，シーケンシャルなデータ参照，およびクローズを指定した回数実行するベンチマークプログラムである．Bonnie は，複数の AP プロセスが単一のファイルデータのランダムな位置を参照し，指定した確率でデータを更新するベンチマークプログラムである．

評価では，データ複写回数とプロセス間通信回数の影響に着目するため，ファイルキャッシュのヒット率向上のための機構と実入出力処理機構による影響を排除するようにした．具体的には，各評価の測定において，すべてのデータがファイルキャッシュ上に存在する状態とした．

評価環境を**表 3** に示す．マイクロカーネル構造 OS において，OMF 機能を MMF 機能と通常入出力機能と比較するため，***AnT***（OMF 機能）と MINIX（MMF 機能，通常入出力機能）を評価対象とした．また，OS 構造の違いによ

表 3　評価環境

OS	***AnT*** MINIX 3.3.0 Linux 2.6.32-504.el6.i686（CentOS 6.6 i386）
CPU	Intel Core i7-2600 3.4 GHz（4 コア，1 コアのみ使用）
RAM	8,192 MBytes

る各機能の性能差を明確化するため，モノリシックカーネル構造を有する Linux（MMF 機能，通常入出力機能）も評価対象とした．

MINIX と Linux の MMF 機能 [1], [2] は，マッピング処理（mmap()）において，オンデマンドページング機能を用いる場合（MMF（ODP））と，事前処理によりページフォールト発生を抑制する場合（MMF（pre-get））がある．なお，MINIX は，MMF（pre-get）のみ実装されているため，新たに MMF（ODP）を実装した．

3.2　ファイルデータの参照性能

3.2.1　測定内容

単一プロセスが単一ファイルの先頭からブロック単位（4 KBytes）のマッピング処理，マッピングしたファイルのデータ参照処理，およびこれらの合計処理について，処理時間を測定し，性能を比較する．測定プログラムの処理流れを**図 3** に示し，以下で説明する．

(1) 事前に生成した 1,024 KBytes のファイルをオープンする．ファイルの内容は，作成時にすべて 0 とした．

(2) 処理（1）でオープンしたファイルを測定プロセスの仮想メモリ空間の空き領域にマッピングする．OMF 機能は readblock()，MMF 機能は，mmap() のファイルマッピング機能を用いる．

(3) ファイルマッピングした範囲をブロック単位（4 Kbytes）で，各ブロックの先頭 1 Byte のファイルデータを読み込み，ファイルデータを参照する．この際，オンデマンドページング機能を利用する MMF 機能では，ページフォールトが発生する．

(4) 処理（2）で作成したファイルのマッピングを削除する．この処理は，MMF 機能（munmap()）のみ実行する．

(5) 処理（1）でオープンしたファイルをクローズする．

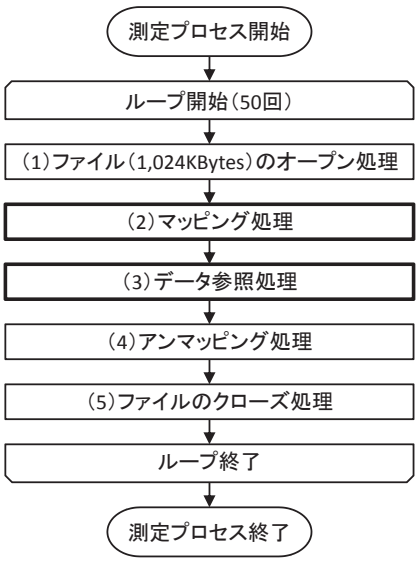


図 3　測定プログラムの処理流れと測定区間

また，図 3 の参照処理のうち，(2) マッピング処理，(3) データ参照処理，および合計処理（(2) マッピング処理 + (3) データ参照処理）の三つの区間を測定区間とした．これらの測定は，CPU のタイムスタンプカウンタ（TSC）の値を取得し，取得した TSC の差を処理時間とした．

測定は，単一の AP プロセス（測定プロセス）を実行し，測定プロセスは起動後，処理（1）から処理（5）を 50 回連続で実行し，各回で TSC を取得した．また，測定プロセスの起動時に，マッピングサイズ（4 Kbytes 単位）を引数として与え，マッピングサイズを，ファイル先頭を開始位置とし，4 Kbytes から 128 Kbytes で変化させ測定した．

なお，すべてのデータがファイルキャッシュ上に存在する場合を測定結果とするため，測定プロセスの参照処理 50 回の TSC のうち，6 回目から 45 回目の計 40 回の平均値を測定値とした．

3.2.2　結果と考察

ファイルデータ参照の処理時間を**図 4** に示す．

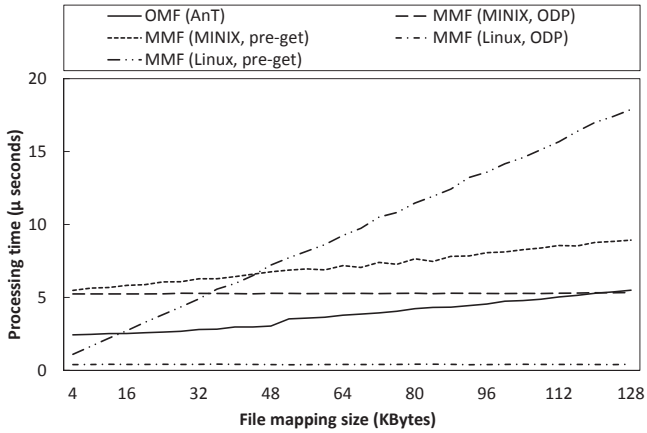
以降では，***AnT*** の OMF 機能を OMF（***AnT***），MINIX と Linux の MMF (ODP) を MMF (MINIX, ODP)，MMF (Linux, ODP)，および MINIX と Linux の MMF (pre-get) を MMF（MINIX，pre-get），MMF（Linux，pre-get）と表記する．

図 4(A) より，マッピング処理について以下のことがわかる．

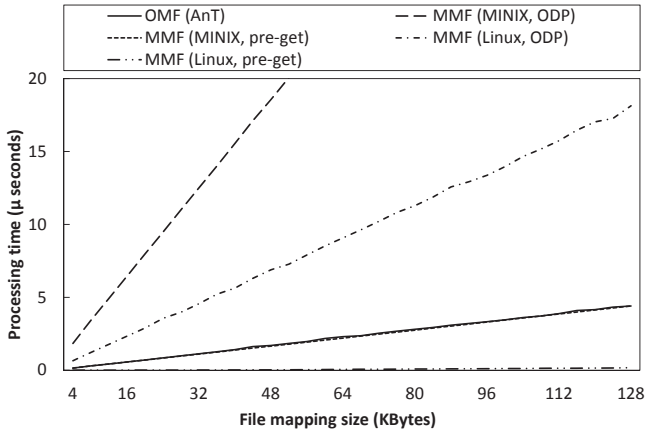
(1) OMF（***AnT***）の処理時間は，マッピングサイズに関わらず，MMF（MINIX，pre-get）より短い．これは，OMF（***AnT***）のプロセス間通信回数が MMF（MINIX，pre-get）より少ないことに起因する．具体的には，MMF（MINIX，pre-get）は，メモリ管理を経由してファイル管理へ処理を依頼する．一方，OMF（***AnT***）は，ファイル管理へ直接処理を依頼するため，プロセス間通信回数が少ない．

(2) OMF（***AnT***）の処理時間は，マッピングサイズが約 124 KBytes 未満の場合，MMF（MINIX，ODP）より短い．これは，OMF（***AnT***）のプロセス間通信回数が MMF（MINIX，ODP）より少ないことに起因する．一方，OMF（***AnT***）の処理時間は，マッピングサイズが約 124 KBytes 以上の場合，MMF（MINIX，ODP）より長い．これは，MMF（MINIX，ODP）がオンデマンドページングを利用しており，マッピングサイズに比例して増加する処理時間が僅かであるためである．

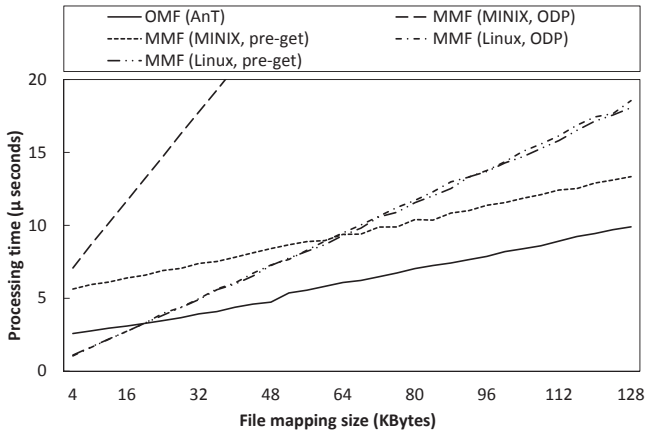
(3) OMF（***AnT***）の処理時間は，マッピングサイズが大きい（約 16 KBytes 以上）場合，MMF（Linux，pre-get）よ



(A) マッピング処理



(B) データ参照処理



(C) 合計処理（マッピング処理 + データ参照処理）

図 4 参照性能の処理時間

り短い．これは，OMF（***AnT***）のファイルデータの格納された実ページの対応付け処理の処理時間が MMF（Linux, pre-get）より短いためである．一方，OMF（***AnT***）の処理時間は，マッピングサイズが小さい（約 16 KBytes 未満）場合，MMF（Linux，pre-get）より長い．これは，***AnT*** がマイクロカーネル構造を有しており，プロセス間通信のオーバヘッドが大きいためである．

(4) OMF（***AnT***）の処理時間は，マッピングサイズに関わらず，MMF（Linux，ODP）より長い．これは，OMF（***AnT***）がオンデマンドページング機能を利用しないことと，プロセス間通信のオーバヘッドが大きいためである．

(5) MMF（MINIX，ODP）の処理時間は，マッピングサイズに関わらず，MMF（Linux, ODP）より長い．これは，MINIX がマイクロカーネル構造を有しており，プロセス間通信のオーバヘッドが大きいためである．

図 4(B) より，データ参照処理について以下のことがわかる．

(1) OMF（***AnT***）の処理時間は，マッピングサイズに関わらず，MMF（MINIX，ODP）や MMF（Linux，ODP）より短い．これは，OMF（***AnT***）がマッピング処理時において事前に AP プロセスの仮想メモリ空間とファイルデータの実メモリとの対応付けの処理を行っており，ページフォールトが発生しないためである．

(2) OMF（***AnT***）と MMF（MINIX, pre-get）の処理時間は，マッピングサイズに比例して増加する．これは，***AnT*** と MINIX，pre-get がマイクロカーネル構造 OS を有しており，プロセス間通信に伴う空間切替え時の TLB フラッシュにより，メモリアクセス時に TLB ミスが発生するためである．

(3) OMF（***AnT***）の処理時間は，マッピングサイズに関わらず，MMF（Linux，pre-get）より長い．これは，***AnT*** がマイクロカーネル構造 OS を有しており，プロセス間通信に伴う空間切替えによる TLB ミスの影響が大きいためである．

図 4(C) より，合計処理（マッピング処理 + データ参照処理）について以下のことがわかる．

(1) OMF（***AnT***）の処理時間は，参照単位が大きい（約 22 KBytes 以上）場合，最も短い．具体的には，参照単位が 32 KBytes の場合，OMF（***AnT***）の処理時間は 3.93 μ 秒であり，MMF（MINIX，ODP: 17.79 μ 秒），MMF（MINIX，pre-get: 7.39 μ 秒），MMF（Linux，ODP: 4.98 μ 秒），および MMF（Linux, pre-get: 4.92 μ 秒）より短い．これは，図 4(A)(B) より，OMF（***AnT***）のマッピング処理が MMF（MINIX，pre-get）や MMF（Linux，pre-get）より高速であり，かつデータ参照処理が MMF（MINIX，Linux，ODP）より高速であるためである．

(2) OMF（***AnT***）の処理時間は，参照単位が小さい（約 22 KBytes 未満）場合，MMF（Linux, ODP）と MMF（Linux, pre-get）より長い．具体的には，参照単位が 4 KBytes の場合，OMF（***AnT***）の処理時間は 2.58 μ 秒であり，MMF（Linux，ODP: 1.05 μ 秒）と MMF（Linux，pre-get: 1.11 μ 秒）より長い．これは，***AnT*** がマイクロカーネル構造を有しており，プロセス間通信のオーバヘッドが大きいためである．

(3) MMF（Linux，ODP）と MMF（Linux，pre-get）の処理時間は，参照単位に関わらずおよそ同じである．これ

は，MMF（Linux，ODP）と MMF（Linux，pre-get）の共通処理であるプロセスのファイルのマッピング用に確保された仮想メモリ空間とファイルデータの実メモリとの対応付けの処理が合計処理の大半を占めるためである．

以上のことから，OMF（***AnT***）の処理時間は，同じマイクロカーネル構造 OS に実現されており，オンデマンドページング機能を利用せず事前処理によりページフォールト発生を抑制する方式を採用している MMF（MINIX，pre-get）より短い．これは，OMF（***AnT***）のプロセス間通信回数が MMF（MINIX，pre-get）より少ないためである．このことから，OMF 機能は，マイクロカーネル構造 OS において有効であるといえる．

また，OMF（***AnT***）とモノリシックカーネル構造 OS に実現された MMF（Linux，ODP）や MMF（Linux，pre-get）との比較では，マッピングサイズが小さい場合（約 22 KBytes 未満），OS 構造の差により，OMF（***AnT***）の処理時間の方が長いものの，マッピングサイズが大きい場合（約 22 KBytes 以上），OMF（***AnT***）の処理時間は，MMF（Linux，ODP）や MMF（Linux，pre-get）より短く，マッピングサイズの増加に伴い，処理時間の差が開く傾向にある．このことから，OMF 機能について，性能面で有効性を確認できた．

3.3 ベンチマーク性能

3.3.1 PostMark

PostMark のパラメータとして，ファイル数を 25，処理回数を 25 回，ファイルサイズを 500 Bytes から 10,000 Bytes の間とし，測定した．また，一度に参照する参照単位をブロック単位（4 KBytes）とバイト単位について測定した．なお，すべての測定において，参照する総バイトサイズは，同じとなる．

測定結果を**図 5** に示す．図 5(A) は，プロセスがブロック単位でデータを参照する処理時間，図 5(B) は，プロセスがバイト単位でデータを参照する処理時間である．

以降では，***AnT*** の OMF 機能（readblock()，readbyte()）を OMF（***AnT***，readblock()），OMF（***AnT***，readbyte()），MINIX と Linux の通常入出力機能（read()，fread()）を Normal（MINIX，read()），Normal（MINIX，fread()），Normal（Linux，read()），および Normal（Linux，fread()）と表記する．

図 5 より，以下のことがわかる．

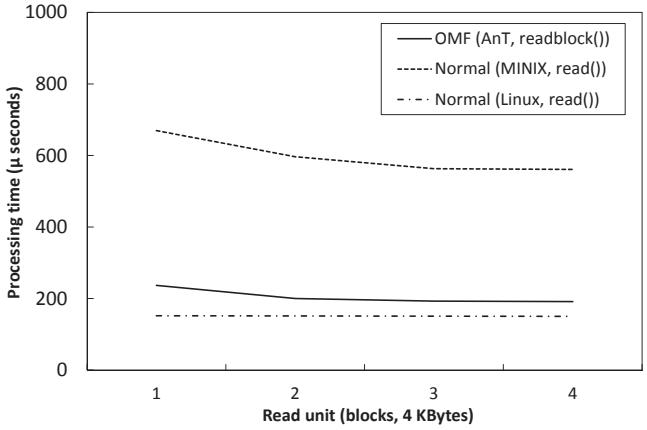
(1) 図 5(A) において OMF（***AnT***，readblock()），Normal（MINIX，read()），および Normal（Linux，read()）の処理時間は，参照単位数の増加に伴い減少する．これは，参照単位の増加に伴い，システムコール発行回数が減少するためである．

(2) 図 5(A) において，OMF（***AnT***，readblock()）の処理時間は，参照単位に関わらず Normal（MINIX，read()）より短い．これは，OMF（***AnT***）のプロセス間通信において，データ複写が発生しないためである．

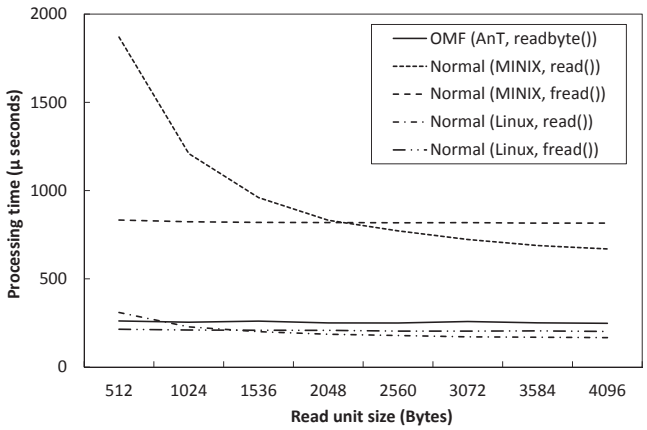
(3) 図 5(A) において，OMF（***AnT***，readblock()）の処理時間は，参照単位に関わらず Normal（Linux，read()）より長い．これは，***AnT*** がマイクロカーネル構造を有しており，データ参照時にプロセス間通信回数が増加することに起因する．

(4) 図 5(B) において，OMF（***AnT***，readbyte()），Normal（MINIX，fread()），および Normal（Linux，fread()）の処理時間は，参照単位に関わらず一定である．これは，readbyte() と fread() の処理において，ライブラリの保持するバッファにデータが存在する場合，ライブラリ内で処理が完了し，システムコールが発行されないためである．これに対し，Normal（MINIX，read()）と Normal（Linux，read()）の処理時間は，参照単位の増加に伴い減少する．これは，参照単位の増加に伴い，システムコール発行回数が減少するためである．

(5) 図 5(B) において，OMF（***AnT***，readbyte()）の処理時間は，参照単位に関わらず Normal（MINIX，read()）と Normal（MINIX，fread()）より短い．これは，OMF 機能（***AnT***，readbytes()）は，プロセス間通信においてデータ複写が発生しないためである．



(A) ブロック単位の読み込み



(B) バイト単位の読み込み

図 5 PostMark の測定結果

以上のことから，OMF（***AnT***）の処理時間は，参照単位に関わらず，マイクロカーネル構造 OS に実現された Normal（MINIX，read()）と Normal（MINIX，fread()）より短い．これは，OMF（***AnT***）はプロセス間通信においてデータ複写が発生しないためである．このことから，単一 AP プロセスによる複数ファイルのデータ参照処理において，OMF 機能は，マイクロカーネル構造 OS の通常入出力機能より有効であるといえる．

一方，モノリシックカーネル構造 OS の Linux との比較では，OMF（***AnT***）の処理時間は，Normal（Linux，read()）と Normal（Linux，fread()）より長いものの，その処理時間の差は小さい．このことから，OMF 機能について，性能面で有効であるといえる．

3.3.2 Bonnie

Bonnie のパラメータとして，ベンチマークプロセス数を 1 と 5，操作対象のファイルサイズを 128 KBytes，データの参照単位と更新単位を 4 KBytes，全ベンチマークプロセスの処理回数の合計値を 4,000 回とし，測定した．

測定結果を**図 6** に示す．図 6(A) はベンチマークプロセス数が 1 の場合，図 6(B) はベンチマークプロセス数が 5 の場合の測定値であり，図 6(C) は図 6(B) の測定値から図 6(A) の測定値を引いたものである．

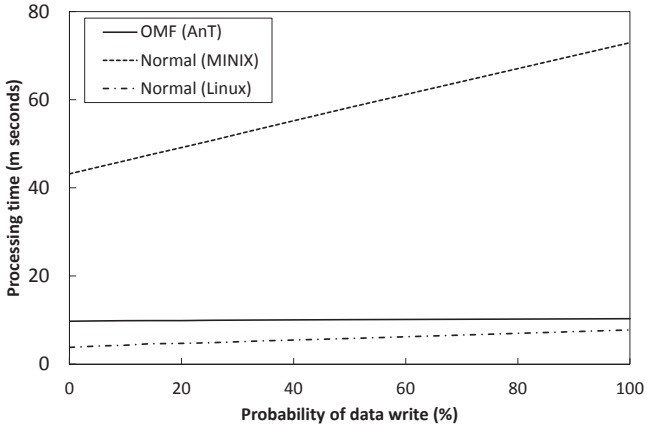
以降では，***AnT*** の OMF 機能を OMF（***AnT***），MINIX と Linux の通常入出力機能を Normal（MINIX），Normal（Linux）と表記する．

図 6 より，以下のことがわかる．

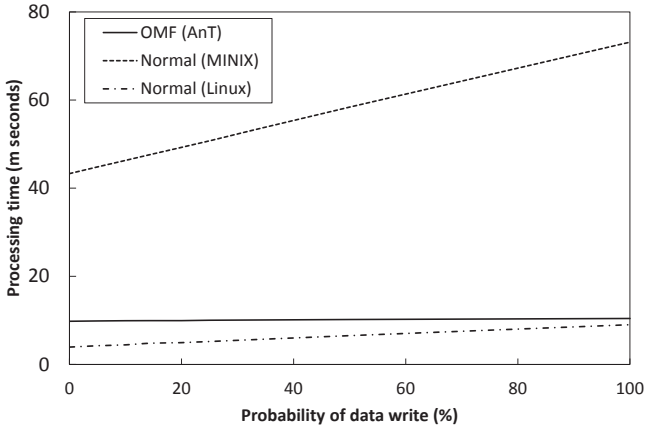
(1) 図 6(A) と図 6(B) より，OMF（***AnT***）の処理時間は，データ更新確率に関わらず，プロセス数が 1 と 5 の両方で一定である．これは，OMF（***AnT***）では，プロセスとファイルキャッシュを共有するため，ファイルデータの更新処理（write() 相当）が不要なことに起因する．これに対し，Normal（MINIX）と Normal（Linux）の処理時間は，データ更新確率に比例して増加する．これは，データ更新確率の増加に伴い，write() システムコールの発行回数が増加するためである．

(2) 図 6(C) より，OMF（***AnT***）と Normal（MINIX）のプロセス数 1 と 5 の処理時間の差は，データ更新確率に関わらず一定である．これは，***AnT*** と MINIX がマイクロカーネル構造を有しており，プロセス間通信に伴う空間切替えによる TLB ミスの影響が小さいためである．***AnT*** と MINIX では，ベンチマークプロセス数に関わらずプロセス間通信が多発するため，これに伴う TLB ミスも多発する．したがって，ベンチマークプロセス数が変化しても，全ベンチマークプロセスの処理回数は同じ（4,000 回）であるため，処理時間はほとんど変化しない．これに対し，Normal（Linux）の処理時間は大きく増加する．これは，Linux がモノリシックカーネル構造を有しており，複数のベンチマークプロセス間での空間切替えによる TLB ミスの影響が大きいためである．通常入出力機能（Linux）では，ベンチマークプロセス数が 1 の場合，空間切替えは発生せず，TLB ミスも発生しない．一方，ベンチマークプロセス数が 5 の場合，ベンチマークプロセス数が 1 の場合と比較してベンチマークプロセス間での空間切替えが多発し，これに伴う TLB ミスが多発する．したがって，空間切替えによる TLB ミスの影響が大きく，処理時間は大きく増加する．

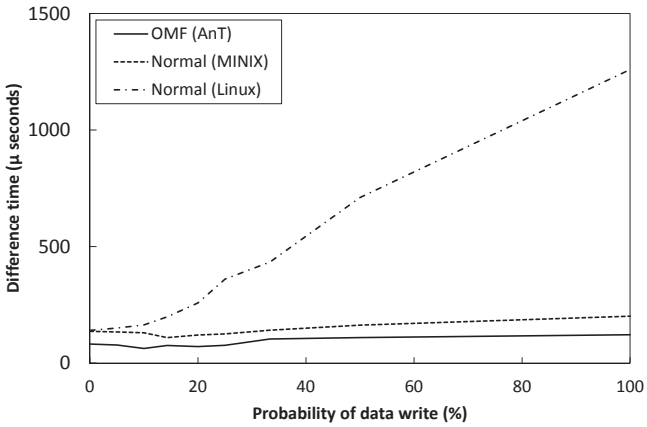
以上のことから，OMF（***AnT***）の処理時間は，データ更



(A) プロセス数 1



(B) プロセス数 5



(C) 処理時間の差分（プロセス数 5 − プロセス数 1）

図 6 Bonnie の測定結果

新確率とベンチマークプロセス数に関わらず，マイクロカーネル構造 OS に実現された Normal（MINIX，read()）と Normal（MINIX, fread()）より短い．これは，OMF（***AnT***）が AP プロセスとキャッシュを共有するため，データ参照時にデータ複写が発生しないことと，ファイルデータの更新処理（write() 相当）が不要であるためである．このことから，複数 AP プロセスによる単一ファイルのデータ参照処理と更新処理において，OMF 機能は，マイクロカーネル構造 OS の通常入出力機能より有効であるといえる．

一方，モノリシックカーネル構造 OS の Linux との比較では，OMF（***AnT***）の処理時間は，Normal（Linux，read()）と Normal（Linux，fread()）より長いものの，その処理時間の差は小さい．このことから，OMF 機能について，性能面で有効であるといえる．

4. おわりに

マイクロカーネル構造 OS 向けのファイル操作機能であるオンメモリファイル（OMF）機能の評価について述べた．

ファイルデータの参照において，マッピングサイズが大きい場合，OMF 機能（***AnT***）の処理時間は，MMF 機能（MINIX）や MMF 機能（Linux）より短いことを示した．具体的には，マッピングサイズが 32 KBytes の場合，OMF 機能（***AnT***）の処理時間は 3.93 μ 秒であり，MMF 機能（Linux，ODP : 4.98 μ 秒，pre-get : 4.92 μ 秒）や MMF 機能（MINIX，ODP : 17.79 μ 秒，pre-get : 7.39 μ 秒）より短い．このことから，OMF 機能は，マイクロカーネル構造 OS において有効であり，性能面においてもモノリシックカーネル構造 OS の MMF 機能より有効であることを示した．

ベンチマークによる評価において，PostMark では，OMF 機能（***AnT***）の処理時間は，参照単位に関わらず，通常入出力機能（MINIX）より短いことを示した．また，Bonnie では，OMF 機能（***AnT***）の処理時間は，データ更新確率に関わらず一定であり，通常入出力機能（MINIX）より短いことを示した．このことから，OMF 機能は，マイクロカーネル構造 OS の通常入出力機能より有効であることを示した．

残された課題として，マルチコア環境における評価がある．

参考文献

[1] Minix 3: *MINIX 3 Developers Guide Table of Contents - Servers VM internals* (online), available from ⟨http://wiki.minix3.org/doku.php?id=developersguide:vminternals⟩ (accessed 2015-07-09).
[2] Kerrisk, M.: *mmap(2) - Linux Programmer's Manual* (online), available from ⟨http://man7.org/linux/man-pages/man2/mmap.2.html⟩ (accessed 2015-07-09).
[3] Liedtke, J.: *Toward real microkernels*, Communications of the ACM, Vol.39, No.9, pp.70–77 (1996).
[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] 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.R. and Bohman, D.: *Microkernel operating system architecture and mach*, Journal of Information Processing, Vol.14, No.4, pp.442–453 (1992).
[6] Liedtke, J.: *Improving IPC by kernel design*, Proc. 14th ACM Symp. Operating System Principles, pp.175–188 (1994).
[7] 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).
[8] 枡田圭祐, 谷口秀夫：プロセスとファイルキャッシュを共有するオンメモリファイル機能の提案, コンピュータシステム・シンポジウム論文集, vol.2012, pp.25–32 (2012).
[9] 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] Tanenbaum, A.S., Herder, J.N., Bos, H., Gras, B. and Homburg, P.: *Modular system programming in MINIX 3*, The USENIX Magazine, Vol.31, No.2, pp.19–28 (2006).
[11] Katcher, J.: *PostMark: A New File System Benchmark*, Technical Report TR3022, Network Appliance (1997).
[12] Bray, T.: *The Bonnie home page* (online), available from ⟨http://www.textuality.com/bonnie⟩ (accessed 2015-07-09).