Vol.2018-OS-142 No.2
2018/2/27

# FreeBSDの初期化処理におけるOFF2Fの効果予測

河辺 誠弥[1]　谷口 秀夫[1]　佐藤 将也[1]

**概要**：不揮発性メモリを有効利用するソフトウェア技術が研究されている．プログラムの実行を高速化する方式として，揮発性メモリと不揮発性メモリが混載された計算機を対象に，新しい実行プログラムのファイル形式 (OFF2F) が提案されている．OFF2F は，プログラムをメモリ上で実行するときのアクセス形態に着目し，2 つのファイルからなる実行ファイル形式である．本稿では，FreeBSD 11.0-RELEASE の初期化処理における OFF2F の効果予測を示す．

## 1. はじめに

不揮発性メモリ（Non-Volatile Memory : 以降，NV メモリと略す）が登場し，NV メモリを有効利用するソフトウェア技術が研究されている．ファイルシステムで有効利用する研究として，NV メモリをメモリまたはストレージに切り替えて扱う方法 [1]，NV メモリとブロックストレージを組み合わせメモリストレージとして扱う VEMS[2]，NV メモリをメタデータの管理に利用する FSMAC[3] がある．ストレージと組み合わせて有効利用する研究として，高速な NV メモリと低速な SSD の 2 つを階層型のストレージとして扱う ATSMF[4]，NV メモリをストレージのキャッシュとして扱う研究 [5] がある．メモリとして扱う研究として，NV メモリをキャッシュ階層のひとつとして扱う研究 [6]，CPU のキャッシュと NV メモリ上のデータの一貫性を最適化する研究 [7] がある．また，NV メモリを利用した大規模なメモリネットワークの研究 [8] がある．さらに，NV メモリを利用したプログラミング言語の研究 [9] がある．

文献 [10] では，揮発性メモリと NV メモリが混載された計算機環境を対象に，新たな実行ファイルの形式（OFF2F : Object File Format consisting of 2 Files）が提案されている．この形式では，仮想記憶機構が 2 つのメモリの特徴を生かしたプログラム実行の支援を可能にするため，実行ファイルを 2 つのファイルから構成している．具体的には，テキスト部を 1 つのファイルとし，テキスト部以外を別のファイルにする．NV メモリの読み出しは速く，書き込みは遅い．そこで，OFF2F を用いて，読み出しのみ行われるテキスト部を NV メモリ上に格納し，仮想記憶を利用してそのまま仮想空間にマッピングすることで，ページ例外処理を短縮できる．

本稿では，FreeBSD 11.0-RELEASE（以降，FreeBSD と略す）の初期化処理について，プログラム実行時のページ例外処理に対する OFF2F の効果を述べる．

[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. OFF2F と ODP 機能

### 2.1 従来の実行ファイル形式の問題点

既存の実行ファイルは 1 ファイルに格納されているため，仮想記憶機構の要求時ページング（ODP）機能を利用したプログラム実行時には，次の問題がある．

(1) 外部記憶装置（例えば DK）上のファイルとして格納した場合，DK から揮発性メモリへのページ読み出し時間（ページイン処理時間）が長い．また，ページアウト処理時間も長い．外部記憶装置として DK ではなく SSD を利用することも可能になっているが，メモリ間複写に比べるとその入出力時間は長い．

(2) NV メモリ上のファイルシステムとして格納した場合，NV メモリから揮発性メモリへのメモリ間複写により，ページイン処理時間は非常に短い．しかし，全ての実行ファイルを NV メモリ上にファイルとして格納する必要があるため，大きな NV メモリが必要となり，高価になってしまう．

### 2.2 考え方

揮発性メモリは，読み書き共に高速である．一方，NV メモリは，揮発性メモリと同様にバイト単位アクセスが可能であり，読み出しは高速である．しかし，書き込みは低速である．したがって，読み出しのみ行われるデータを NV メモリに格納し，仮想記憶を利用してそのまま仮想空間にマッピングして利用できれば，ODP 処理の時間を短縮できる．

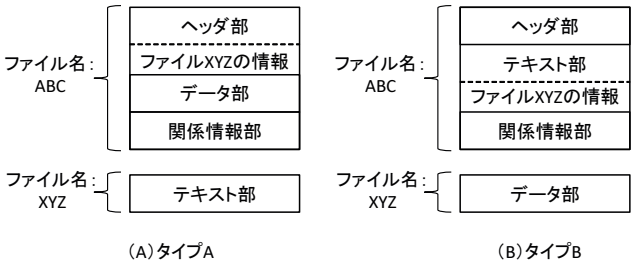


図 1　OFF2F の形式

実行ファイルは，4 つの内容（テキスト部，データ部，関係情報部，ヘッダ部）から構成されている．これらの部分は，アクセス形態から大きく 2 つに分類できる．ひとつは，読み出し（リード）のみ行われる部分であり，テキスト部，関係情報部，およびヘッダ部である．なお，ヘッダ部の読み出しは，主にプログラムをプロセスとして実行するときに発生し，このとき，関係情報部の読み出しは発生しない．一方，テキスト部の読み出しは，プログラム実行時により頻発する．もうひとつは，頻繁に読み書き（リード＆ライト）される部分であり，データ部である．

そこで，実行ファイルの内容のアクセス形態に着目し，複数のファイルからなる実行ファイル形式が提案された．実行ファイルは 4 つの内容から構成されているため，最大 4 つのファイルに分割格納できる．しかし，構造の複雑化を防ぎ，またファイルシステムでの占有領域を抑制するため，構成するファイル数を最小限とする．つまり，実行ファイルを 2 つのファイルに分割格納する実行ファイル形式（OFF2F : Object File Format consistiong of 2 Files）が提案された [10].

### 2.3　OFF2F の形式

OFF2F の形式では，実行ファイルを形成する 4 つの内容から，アクセス形態を考慮し，2 つの形式が提案された．これを図 **1** に示す．

タイプ A は，ヘッダ部，データ部，および関係情報部を 1 つのファイルとし，別ファイルであるテキスト部への情報を持つ．つまり，プログラム実行により頻繁に読み出しが発生するテキスト部のみを別のファイルとする形式である．タイプ B は，ヘッダ部，テキスト部，および関係情報部を 1 つのファイルとし，別ファイルであるデータ部への情報を持つ．つまり，プログラム実行により頻繁に読み書きが発生するデータ部のみを別のファイルとする形式である．タイプ A はファイル XYZ を NV メモリ上に置き，タイプ B はファイル ABC を NV メモリ上に置く．したがって，両者を比較すると，以下の理由によりタイプ A が好ましい．

(1) 実行ファイルの名前はヘッダ部に保持され，外部記憶装置場に存在することにより，既存の処理流れを利用できる．

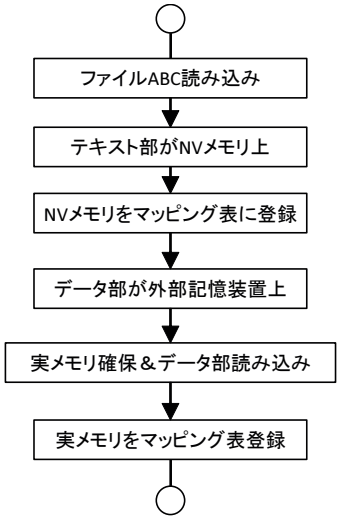


図 2　仮想メモリ空間の作成処理流れ

(2) ファイル XYZ は，必ずしも NV メモリ上に存在する必要はないため，NV メモリの有無の影響を受けない．

### 2.4　OFF2F の利用

仮想記憶機構における OFF2F の利用について図 1 タイプ A の場合を以下に述べる．ファイル ABC は外部記憶装置上に存在し，ファイル XYZ は NV メモリ上に存在する．このとき，ファイル ABC を利用してプロセスのテキスト部とデータ部を仮想メモリ空間に用意する処理流れを図 **2** に示す．

(1) 外部記憶装置上に存在するファイル ABC のヘッダ部を読み込む．
(2) ヘッダ部の情報より，テキスト部が NV メモリ上に存在することを認識する．
(3) NV メモリ上のテキスト部の各ページをマッピング表に登録する．
(4) ヘッダ部の情報より，データ部が外部記憶装置上に存在することを認識する．
(5) 実メモリを確保し，外部記憶装置上に存在するデータ部を読み込む．
(6) 実メモリの各ページをマッピング表に登録する．

処理（3）により，テキスト部について，外部記憶装置上に存在する場合と比べて入出力を削減できる．また，NV メモリ上に存在する場合に比べメモリ間複写を削減でき，大きな NV メモリを必要としない．

また，ODP 機能を有する場合，プロセス生成時には処理（3）と（6）はページ例外フラグの設定処理となり，処理（5）は行わない．

### 2.5　ODP 機能

ODP 機能を有する場合のページ例外処理について，図 **3** に示す．NV メモリ上のテキスト部に対するページ例外の

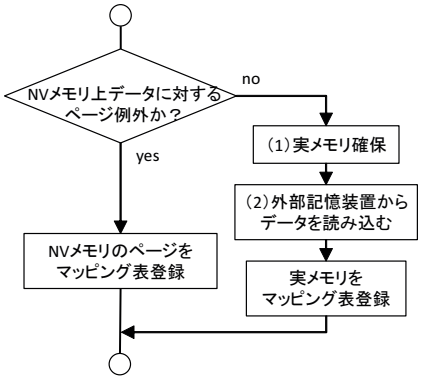


図 3　ページ例外処理の流れ

場合には，NV メモリの該当ページをマッピング表に登録するだけでよい．このため，処理（2）のページイン処理やページアウト処理で発生する入出力は不要であり，処理（1）の実メモリ確保処理も省略できる．

## 3. FreeBSD の初期化処理

### 3.1 初期化処理におけるプロセス

#### 3.1.1 生成されるプロセス

FreeBSD の初期化処理として，init プログラムの生成からログインプログラムの生成までを対象とし，その間の生成から終了までを行うプロセスを述べる．初期化処理において，プロセスの多くは，fork()，exec()，および exit() のシステムコールを用いて生成または終了を行う．これらのシステムコールを用いて生成から終了までを行うプロセスは 588 個ある．このうち，最も小さい pid を持つプロセスは pid25 であり，最も大きい pid を持つプロセスは pid613 である．

プロセスの生成から終了までの経過時間を図 4（A）に示す．図 4（A）の横軸は init 生成後からの経過時間であり，縦軸は pid である．図 4（A）では，pid25 が init プログラム生成後からログインプログラム生成前まで走行している．そして，プロセス生成の多くの場合，pid25 のプロセスが fork() により他のプロセスを生成し，その後，生成されたプロセスが exec() により新たなプログラムを生成する．この様子の典型的な例として，図 4（A）の 0.735 秒から 0.739 秒までを拡大し図 4（B）に示す．また，pid25 以外のプロセスが fork() する例として，4.662 秒から 4.6655 秒までを拡大し図 4（C）に示す．図 4（B）では，pid25 の sh が fork() により pid50 から pid53 までの sh を複製する．pid52 の sh は複製された後，exec() により swapon を生成する．図 4（C）では，pid25 の sh が fork() により pid347 と pid351 の sh を複製する．また，pid347 の sh は pid348 から pid350 までの sh を複製する．pid348 から pid350 までの sh は複製された後，それぞれ exec() により mount を生成する．したがって，図 4 より，以下のことがわかる．

(1) pid25 が init プログラム生成からログインプログラム

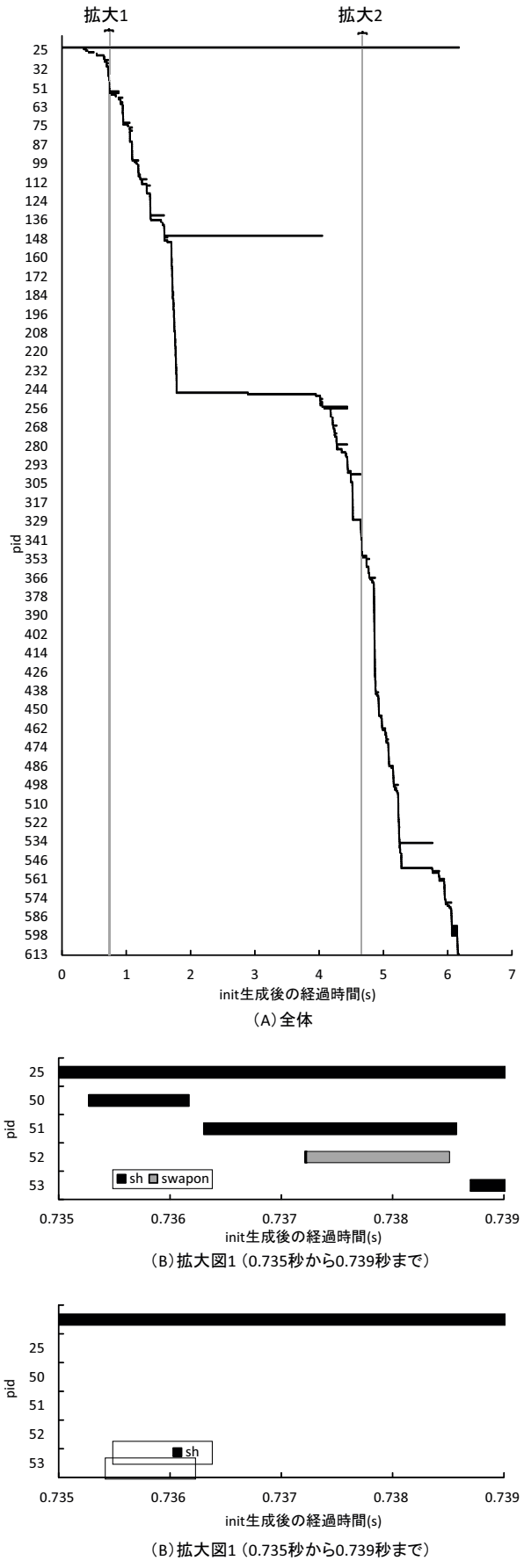


図 4　初期化処理における生成または終了するプロセス

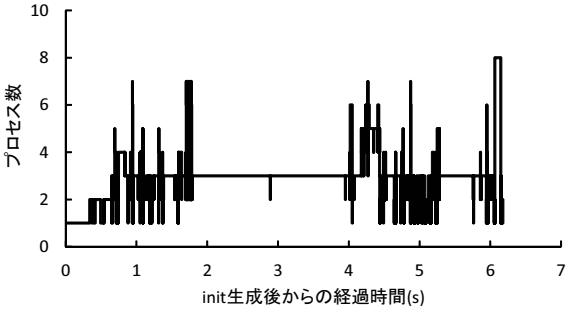


図 5　初期化処理におけるプロセス数の遷移

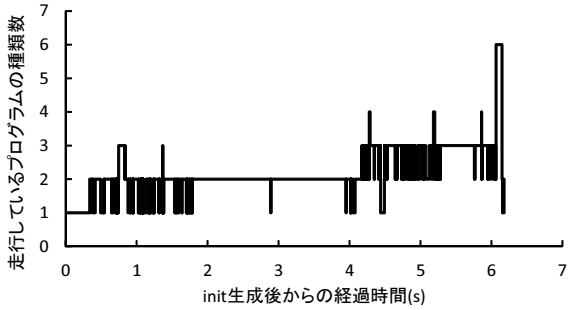


図 6　初期化処理における走行しているプログラムの種類数の遷移

生成まで走行し，fork() により sh を多数複製する．

(2) pid25 の sh から複製されたプロセスによって，sh 以外のプログラムの多くが生成される．

また，図 4 におけるプロセス数の遷移を**図 5** に示し，走行しているプログラムの種類数の遷移を**図 6** に示す．図 5 と図 6 より，以下のことがわかる．

(3) プロセス数と走行しているプログラムの種類数はどちらも，ほとんどの場合 1 から 3 の間である．

(4) プロセス数が増加した後，走行しているプログラムの種類数が増加している．

プロセス数と走行しているプログラムの種類数の最低値が 1 であるのは，図 4 で述べたように，pid25 のプロセスが長時間走行しているためと考えられる．また，プロセス数が増加した後，走行しているプログラムの種類数が増加しているのは，pid25 のプロセスが fork() によりプロセス数を増加させ，その後，生成されたプロセスが exec() により sh 以外のプログラムを生成するためであると考えられる．

### 3.1.2　分析

図 4（A）に示したプロセスの生成から終了までは，exec() によるプロセス生成とプログラムのテキスト部の共有に着目すると，以下の 4 つに分類できる．

(A) exec() の際，プログラムのテキスト部を共有できない（268 通り）．

(B) 1 回の exec() の際，1 つのプログラムとテキスト部を共有できる（4 通り）．

(C) 2 回の exec() の際，1 つのプログラムとテキスト部を共有できる（5 通り）．

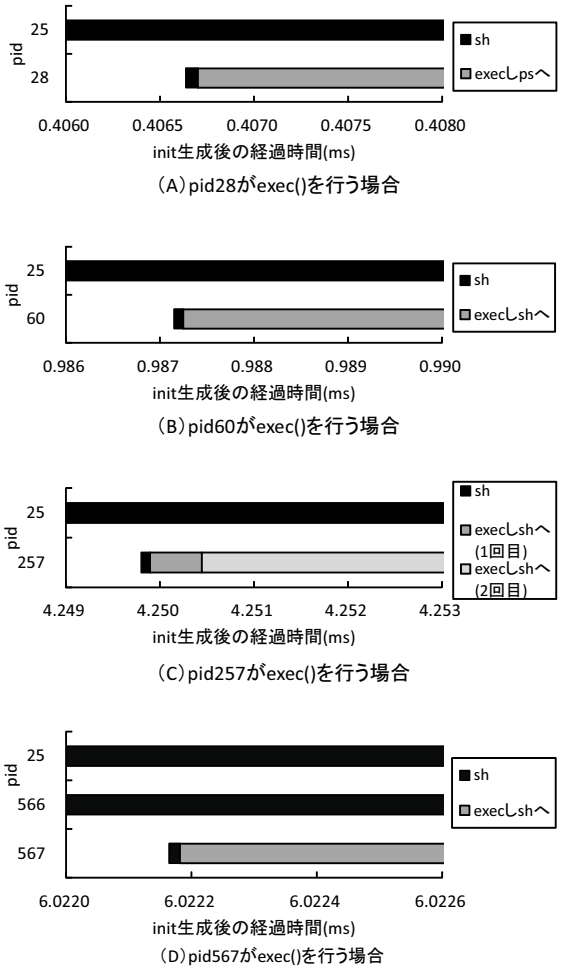


図 7　初期化処理における生成または終了するプロセスの典型例

(D) 1 回の exec() の際，2 つのプログラムとテキスト部を共有できる（1 通り）．

それぞれの場合の典型的な例を**図 7** に示す．図 7（A）は，pid25 の sh が fork() により pid28 の sh を複製し，その後，pid28 の sh が exec() により ps を生成する場合である．このとき，他に ps は走行していないため，exec() によるプロセス生成において，ps プログラムのテキスト部の読み出しが必要となる．図 7（B）は，pid25 の sh が fork() により pid60 の sh を複製し，その後，pid60 の sh が exec() により sh を生成する場合である．このとき，pid60 の sh 以外に，pid25 の sh が走行している．したがって，pid60 の sh が exec() により sh を生成するとき，sh プログラムのテキスト部を共有することができ，テキスト部の一部の読み出しを省略できる．図 7（C）と図 7（D）も同様に，pid25 の sh が走行しているため，pid257 または pid567 の sh が exec() により sh を生成するとき，テキスト部の一部の読み出しを省略できる．

したがって，初期化処理の exec() によるプロセス生成の際，テキスト部が共有される場合は 10 通りであることがわかる．これは，初期化処理の exec() の総回数 278 回に比べて少なく，テキスト部の共有によるプロセス生成の処理

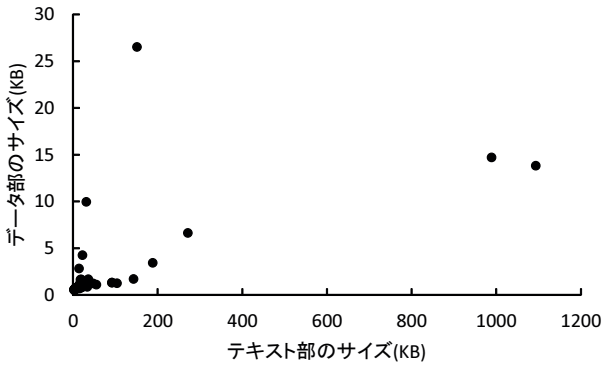


図 8　プログラムのテキスト部とデータ部のサイズ分布

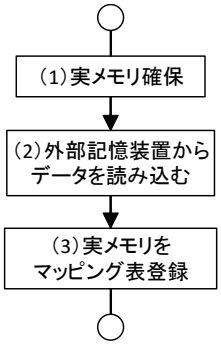


図 9　ページ例外処理の流れ

時間への影響は小さいと考えられる．

### 3.2　初期化処理における実行プログラム

FreeBSD の初期化処理におけるプロセスの生成から終了までを図 4（A）に示した．この際に実行されるプログラムを**表 1** に示す．また，プログラムのテキスト部とデータ部のサイズ分布を**図 8** に示す．表 1 と図 8 より，以下のことがわかる．

(1) プログラムの種類は 61 種ある．
(2) sh は fork() する回数が多く，sh 以外の多くのプログラムは fork() しない．
(3) 多くのプログラムは exec() により生成される．
(4) それぞれのプログラムのテキスト部のサイズは，データ部のサイズに比べ 3 倍以上大きい．

### 3.3　ページ例外処理

FreeBSD のページ例外処理の流れを**図 9** に示し，以下で説明する．ページ例外処理では，まず処理（1）として，物理メモリの空き領域を探索し，確保する．次に処理（2）として，要求されたページに対応する外部記憶装置上のデータを確保した領域に読み込む．最後に処理（3）として，読み込んだページを指定した仮想アドレス空間にマッピングする．

図 9 の処理流れは，図 3 に示した OFF2F を利用したページ例外処理の流れにおける，NV メモリ上にないデータ

表 1　初期化処理における実行プログラム

| 通番 | プログラム名 | テキスト部 (byte) | データ部 (byte) | frok() する回数 | exec() により生成される回数 |
|---|---|---|---|---|---|
| 1 | adjkerntz | 6,073 | 681 | 0 | 1 |
| 2 | atmconfig | 56,462 | 1,144 | 0 | 1 |
| 3 | awk | 192,681 | 3,528 | 0 | 3 |
| 4 | basename | 3,757 | 617 | 0 | 2 |
| 5 | cat | 8,615 | 817 | 0 | 5 |
| 6 | cmp | 8,521 | 737 | 0 | 1 |
| 7 | cron | 37,279 | 1,728 | 0 | 1 |
| 8 | date | 16,533 | 1,248 | 0 | 1 |
| 9 | dd | 20,557 | 824 | 0 | 3 |
| 10 | devd | 1,119,770 | 14,160 | 4 | 1 |
| 11 | devfs | 12,196 | 801 | 0 | 55 |
| 12 | dmesg | 5,129 | 744 | 0 | 1 |
| 13 | dumpon | 4,133 | 633 | 0 | 1 |
| 14 | egrep | 94,331 | 1,356 | 0 | 2 |
| 15 | expr | 16,114 | 713 | 0 | 8 |
| 16 | find | 48,966 | 1,256 | 0 | 1 |
| 17 | fsck | 15,380 | 897 | 2 | 1 |
| 18 | fsck_ufs | 106,206 | 1,300 | 0 | 2 |
| 19 | gpart | 22,796 | 4,352 | 0 | 1 |
| 20 | grep | 94,331 | 1,356 | 0 | 1 |
| 21 | hostname | 2,694 | 593 | 0 | 1 |
| 22 | id | 8,744 | 801 | 0 | 2 |
| 23 | ifconfig | 154,722 | 27,152 | 0 | 12 |
| 24 | init | 1,012,833 | 15,075 | 1 | 0 |
| 25 | ip6addrctl | 7,763 | 761 | 0 | 2 |
| 26 | kbdcontrol | 34,843 | 900 | 0 | 7 |
| 27 | kenv | 4,180 | 641 | 0 | 3 |
| 28 | ldconfig | 17,799 | 944 | 0 | 2 |
| 29 | limits | 15,929 | 849 | 0 | 6 |
| 30 | ln | 6,736 | 729 | 0 | 2 |
| 31 | logger | 8,886 | 761 | 0 | 2 |
| 32 | ls | 27,928 | 1,232 | 0 | 1 |
| 33 | mailwrapper | 4,789 | 713 | 0 | 2 |
| 34 | md5 | 19,723 | 1,720 | 0 | 1 |
| 35 | mixer | 8,558 | 689 | 0 | 3 |
| 36 | mkdir | 3,898 | 641 | 0 | 2 |
| 37 | mktemp | 3,882 | 649 | 0 | 2 |
| 38 | mount | 19,602 | 1,544 | 0 | 6 |
| 39 | newsyslog | 38,428 | 1,296 | 0 | 1 |
| 40 | protect | 3,395 | 609 | 0 | 1 |
| 41 | ps | 32,033 | 10,202 | 0 | 10 |
| 42 | rcorder | 8,769 | 737 | 0 | 2 |
| 43 | realpath | 2,311 | 569 | 0 | 1 |
| 44 | rm | 10,903 | 881 | 0 | 21 |
| 45 | rmdir | 2,727 | 585 | 0 | 2 |
| 46 | route | 26,438 | 1,025 | 0 | 7 |
| 47 | savecore | 15,415 | 985 | 0 | 1 |
| 48 | sed | 32,915 | 1,108 | 0 | 3 |
| 49 | sendmail | 4,789 | 713 | 2 | 2 |
| 50 | sh | 146,263 | 1,748 | 570 | 13 |
| 51 | sleep | 2,883 | 593 | 0 | 2 |
| 52 | sshd | 277,942 | 6,800 | 0 | 2 |
| 53 | stty | 18,054 | 1,680 | 0 | 1 |
| 54 | swapon | 14,569 | 984 | 0 | 2 |
| 55 | sysctl | 17,327 | 857 | 0 | 49 |
| 56 | syslogd | 37,812 | 1,556 | 1 | 1 |
| 57 | tr | 14,369 | 2,904 | 0 | 5 |
| 58 | umount | 14,390 | 929 | 0 | 1 |
| 59 | uname | 6,416 | 617 | 0 | 1 |
| 60 | utx | 3,880 | 625 | 0 | 1 |
| 61 | vidcontrol | 21,642 | 841 | 0 | 1 |
| | 合計 | 4,007,009 | 134,130 | 580 | 278 |

へのページ例外の場合と同様である．したがって，OFF2F を利用することにより，処理（1）の実メモリ確保と処理（2）の外部記憶装置上からデータを読み込むを省略できる．このため，FreeBSD ではテキスト部に対するページ例外処理の処理時間の短縮が期待できる．

## 4. OFF2F の効果

### 4.1 観点

初期化処理の実行プログラムのページ例外回数に着目し，ページ例外処理時間の定式化を行い，OFF2F の効果予測を示す．効果予測として，従来の方式と OFF2F を適用した方式における，DK または SSD からページを読み込む場合を比較する．

### 4.2 実行プログラムのページ例外数

FreeBSD のページ例外について，4KB のページ単位でページ例外処理が行われる場合に着目し，初期化処理の実行プログラムのページ例外回数について述べる．初期化処理における実行プログラムについて，プログラムのテキスト部とデータ部に対し，それぞれについてページ単位（4KB）で読み込む回数をページ例外数とし，**表 2** に示す．テキスト部に対するページ例外数を PFt とし，データ部に対するページ例外数を PFd とし，それぞれ，表 1 を用いて以下のように求めた．

PFt=テキスト部のページ数 (4KB) × exec() により生成される回数

PFd=データ部のページ数 (4KB) × (fork() する回数+exec() により生成される回数)

表 2 より，初期化処理のプロセス生成におけるページ例外回数は，テキスト部とデータ部に対するページ例外数の合計から 3,598 回であることがわかる．このうち，テキスト部に対するページ例外数は 2,627 回である．したがって，OFF2F を適用することにより，テキスト部に対するページ例外数 2,627 回において，ページ例外処理の処理時間の短縮が期待できる．

また，テキスト部の合計サイズは表 1 に示したようにテキスト部の合計サイズが約 4MB であるため，NV メモリのサイズが 4MB あれば十分に OFF2F の効果が期待できる．

### 4.3 定式化

本評価では，文献 [10] と同様の評価環境を仮定する．したがって，図 9 に示した FreeBSD のページ例外処理の処理流れを，外部記憶装置からデータを読み込む処理（DK と SSD）とその他の処理の 3 つに分け，以下を用いて定式化を行う．

Tdk：DK からページ (4KB) を読み込む処理時間 (6 ミリ秒)

Tssd：SSD からページ (4KB) を読み込む処理時間 (0.1

**表 2** 初期化処理における実行プログラムのページ例外数

| 通番 | プログラム名 | テキスト部のページ例外数 | データ部のページ例外数 |
|---|---|---|---|
| 1 | adjkerntz | 2 | 1 |
| 2 | atmconfig | 14 | 1 |
| 3 | awk | 144 | 3 |
| 4 | basename | 2 | 2 |
| 5 | cat | 15 | 5 |
| 6 | cmp | 3 | 1 |
| 7 | cron | 10 | 1 |
| 8 | date | 5 | 1 |
| 9 | dd | 18 | 3 |
| 10 | devd | 274 | 20 |
| 11 | devfs | 165 | 55 |
| 12 | dmesg | 2 | 1 |
| 13 | dumpon | 2 | 1 |
| 14 | egrep | 48 | 2 |
| 15 | expr | 32 | 8 |
| 16 | find | 12 | 1 |
| 17 | fsck | 4 | 3 |
| 18 | fsck_ufs | 52 | 2 |
| 19 | gpart | 6 | 2 |
| 20 | grep | 24 | 1 |
| 21 | hostname | 1 | 1 |
| 22 | id | 6 | 2 |
| 23 | ifconfig | 456 | 84 |
| 24 | init | 0 | 4 |
| 25 | ip6addrctl | 4 | 2 |
| 26 | kbdcontrol | 63 | 7 |
| 27 | kenv | 6 | 3 |
| 28 | ldconfig | 10 | 2 |
| 29 | limits | 24 | 6 |
| 30 | ln | 4 | 2 |
| 31 | logger | 6 | 2 |
| 32 | ls | 7 | 1 |
| 33 | mailwrapper | 4 | 2 |
| 34 | md5 | 5 | 1 |
| 35 | mixer | 9 | 3 |
| 36 | mkdir | 2 | 2 |
| 37 | mktemp | 2 | 2 |
| 38 | mount | 30 | 6 |
| 39 | newsyslog | 10 | 1 |
| 40 | protect | 1 | 1 |
| 41 | ps | 80 | 30 |
| 42 | rcorder | 6 | 2 |
| 43 | realpath | 1 | 1 |
| 44 | rm | 63 | 21 |
| 45 | rmdir | 2 | 2 |
| 46 | route | 49 | 7 |
| 47 | savecore | 4 | 1 |
| 48 | sed | 27 | 3 |
| 49 | sendmail | 4 | 4 |
| 50 | sh | 468 | 583 |
| 51 | sleep | 2 | 2 |
| 52 | sshd | 136 | 4 |
| 53 | stty | 5 | 1 |
| 54 | swapon | 8 | 2 |
| 55 | sysctl | 245 | 49 |
| 56 | syslogd | 10 | 2 |
| 57 | tr | 20 | 5 |
| 58 | umount | 4 | 1 |
| 59 | uname | 2 | 1 |
| 60 | utx | 1 | 1 |
| 61 | vidcontrol | 6 | 1 |
| | 合計 | 2,627 | 971 |

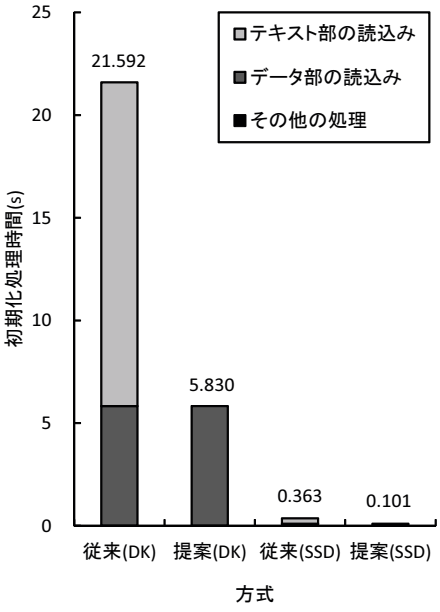


図 10　初期化処理におけるページ例外の処理時間の比較

ミリ秒)

T：読み込む処理以外の処理時間 (0.001 ミリ秒)

また，3.1.2 項で述べたように，初期化処理の exec() によるプロセス生成において，テキスト部が共有される場合は 10 回であり，exec() の総回数 278 回に比べて少なく，プロセス生成の処理時間への影響は小さいと考えられる．このため，本評価では，初期化処理のプロセス生成において，テキスト部を共有しないとする．

従来のページ例外処理（以降，従来方式と略す）では，テキスト部またはデータ部に対するページ例外のどちらも，ページを読み込む処理が必要となる．また，このとき，読み込む以外の処理時間も発生する．一方，OFF2F を適用した方式（以降，提案方式とする）は，テキスト部に対するページ例外はページを読み込む処理が省略できる．しかし，データ部に対するページ例外はページを読み込む処理が発生する．また，その他の処理時間は，テキスト部またはデータ部に対するページ例外のどちらにも発生する．

以上から，従来方式と提案方式における DK または SSD からページを読み込む場合の 4 通りについて，ページ例外処理時間を以下から求める．

従来（DK）=(PFt+PFd) × (Tdk+T)
提案（DK）=PFd × Tdk+(PFt+PFd) × T
従来（SSD）=(PFt+PFd) × (Tssd+T)
提案（SSD）=PFd × Tssd+(PFt+PFd) × T

### 4.4　比較

初期化処理における OFF2F の効果予測として，4.3 節に示した従来方式と提案方式における DK または SSD からページを読み込む場合の 4 通りを**図 10** に示す．図 10 より，従来（DK）に比べ提案（DK）は約 3.69 倍 (21.519 ÷ 5.830)，従来（SSD）に比べ提案（SSD）は約 3.59 倍 (0.363 ÷ 0.101) 高速化していることがわかる．

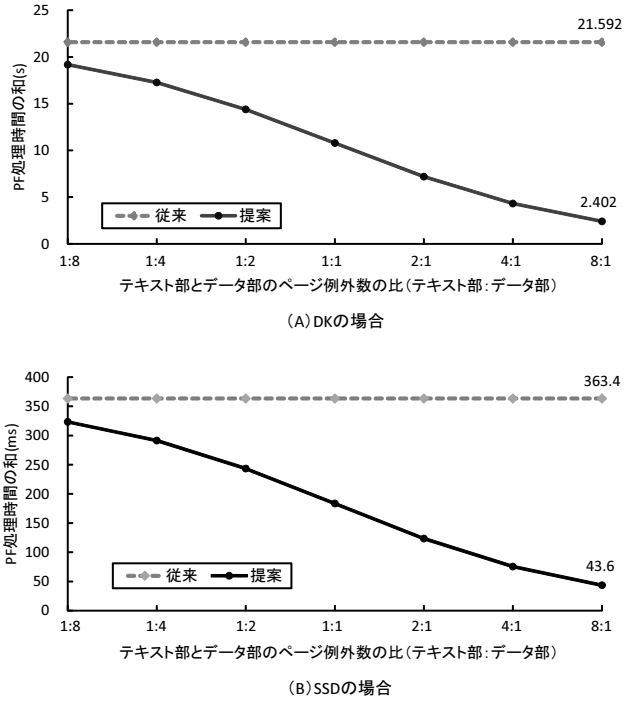


図 11　テキスト部とデータ部の比率における効果予測

初期化処理における実行プログラムのテキスト部とデータ部のページ例外数は，それぞれ 2,627 回と 971 回であり，およそ 3 : 1 の比率であるといえる．この比率を変化させた場合の各方式の初期化処理時間を**図 11** に示す．図 11 より，テキスト部とデータ部の比率が 8 : 1 のとき，DK の場合，従来方式に比べ提案方式は約 8.99 倍 (21.592 ÷ 2.402) 高速化し，SSD の場合，従来方式に比べ提案方式は約 8.33 倍 (363.4 ÷ 43.6) 高速化していることがわかる．また，テキスト部の比率が大きくなるほど，OFF2F の効果が大きくなることがわかる．したがって，プログラムの生成が多い場合，exec() 回数が多くなりやすく，テキスト部に対するページ例外の比率が大きくなる可能性が高いため，OFF2F の効果がより大きくなると予測できる．

## 5.　おわりに

揮発性メモリと NV メモリが混載された計算機環境を対象とする実行ファイルの形式（OFF2F）を FreeBSD の初期化処理に適用し，その効果予測について述べた．OFF2F は，2 つのファイルから実行ファイルを構成し，読み出しのみ行われるテキスト部を NV メモリ上に格納し，テキスト部以外を別のファイルにすることで，プログラムの実行を高速化できる．

OFF2F を FreeBSD に適用することで，テキスト部に対するページ例外処理の処理時間の短縮が期待でき，その効

果予測として初期化処理の処理時間について述べた．具体的には，DK の利用では従来から約 3.69 倍，SSD の利用では従来から約 3.59 倍高速化できることを示した．また，テキスト部とデータ部の比率による効果予測について述べ，8 : 1 のとき，DK の場合，従来方式に比べ提案方式は約 8.99 倍高速化し，SSD の場合，従来方式に比べ提案方式は約 8.33 倍高速化できることを示した．

残された課題として，OFF2F の実装による評価と CoW が有効な場合の評価がある．

**謝辞** 本研究の一部は，株式会社富士通研究所との共同研究による．

## 参考文献

[1] 追川修一 : Non-Volatile メインメモリとファイルシステムの融合, 情報処理学会論文誌, Vol.54, No.3, pp.1153–1164 (2013).
[2] 追川修一 : ブロックストレージとの組み合わせによるメモリストレージ容量拡張手法，情報処理学会論文誌コンピューティングシステム，Vol.8，No.2，pp.15–24 (2015).
[3] Qingsong Wei, Jianxi Chen, Cheng Chen : Accelerating File System Metadata Access with Byte-Addressable Nonvolatile Memory, ACM Transactions on Storage (TOS), Vol.11, Issue 3, pp.1–28 (2015).
[4] Kazuichi Oe, Mitsuru Sato, Takeshi Nanri : Automated tiered storage system consisting of memory and flash storage to improve response time with input-output (IO) concentration workloads, 2017 Fifth International Sumposium on Computing and Networking (2017).
[5] Eunji Lee, Julie Kim, Hyokyung Bahn, Sunjin Lee, Sam H. Noh : Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM, ACM Transactions on Storage (TOS), Vol.13, Issue 2, pp.1–13 (2017).
[6] Doe Hyun Yoon, Tobin Gonzalez, Parthasarathy Ranganathan, Robert S. Schreiber : Exploring latency-power tradeoffs in deep nonvolatile memory hierarchies, Proceedings of the 9th conference on Computing Frontiers, pp.95–102 (2012).
[7] Yiying Zhang, Steven Swanson : A Study of Application Performance with Non-Volatile Main Memory, Mass Storage Systems and Technologies (MSST), pp.1–10 (2015).
[8] Matthew Poremba, Itir Akgun, Jieming Yin, Onur Kayiran, Yuan Xie, Gabriel H. Loh : There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes, Proceedings of the 44th Annual International Symposium on Computer Architecture, pp.678–690 (2017).
[9] Xiaochen Guo, Aviral Shrivastava, Michael Spear, Gang Tan : Languages Must Expose Memory Heterogeneity, Proceedings of the Second International Symposium on Memory Systems, pp.251–256 (2016).
[10] 谷口秀夫：揮発/不揮発メモリ混載環境を支援する仮想記憶機構向け実行ファイル形式 : OFF2F の提案, コンピュータシステム・シンポジウム論文集，Vol.2017，pp.35–40 (2017).