Vol.2019-OS-147 No.16
2019/7/25

不揮発性メモリを利用した*Tender*における動作継続制御の実現

田中 雅大[1]　山内 利宏[1]　谷口 秀夫[1]

概要：計算機の主記憶に用いられるメモリは揮発性メモリであり，計算機の電源消失とともに，主記憶上のデータは消失する．***Tender*** では，揮発性の主記憶をあたかも不揮発性であるかのように見せるプレート機能を実現している．この機能は，揮発性の主記憶を搭載した計算機で仮想記憶空間上のデータを外部記憶装置上へ書き出し，計算機再起動後に仮想記憶空間上に復元する．また，プレート機能を使用して，仮想記憶空間上のすべてのデータを永続化することにより，計算機処理を永続化する動作継続制御を実現している．不揮発性メモリの性能向上に伴い，不揮発性メモリを利用するソフトウェアの技術が研究されている．主記憶に不揮発性メモリを搭載した計算機では，外部記憶装置上に書き出さずとも，主記憶に残ったデータを利用して動作継続制御が可能となる．計算機処理の永続化のために行っていた外部記憶装置への入出力がなくなることにより，動作継続制御の高速化が見込める．本稿では，不揮発性メモリを利用した ***Tender*** における動作継続制御の設計について述べる．また，不揮発性メモリの主記憶をエミュレーションする仮想計算機上での実現方式について述べる．

Masahiro Tanaka[1]　Toshihiro Yamauchi[1]　Hideo Taniguchi[1]

1. はじめに

計算機の主記憶に用いられるメモリは揮発性メモリである．揮発性メモリは，電源の供給がなければデータを保持することができないため，計算機の電源が切断されると主記憶のデータは消失する．したがって，オペレーティングシステム（以降，OS）や応用プログラム（以降，AP）は，計算機の終了後に利用する主記憶上のデータを外部記憶装置に保存することにより，データを永続化する．

Tender オペレーティングシステム [1]（以降，***Tender***）では，揮発性の主記憶をあたかも不揮発性であるかのように見せるプレート機能を実現している．この機能では，OS が揮発性メモリ上の仮想記憶空間のデータを外部記憶装置上へ書き出し，仮想記憶空間のデータを永続化する．計算機再起動時は，外部記憶装置に保存したデータを仮想記憶空間上に復元する．このプレートの書き出しと復元により，仮想記憶空間上のデータが不揮発性メモリ上に存在するかのように計算機再起動後に復元される．

また，***Tender*** では，プレート機能を使用して，カーネル用とユーザ用の仮想記憶空間上に存在するすべてのデータを永続化することにより，OS の処理と AP の処理を永続化する揮発性メモリを利用した動作継続制御 [3] を実現している．これにより，計算機停止前の処理を計算機再起動後に継続して実行できる．

不揮発性メモリの性能向上に伴い，不揮発性メモリを利用するソフトウェアの技術が研究されている．不揮発性メモリを主記憶として利用する場合，***Tender*** ではプレート機能を使用せずとも，不揮発性メモリ上に残ったデータを利用した動作継続制御が可能となる．主記憶を不揮発性メモリであるかのように見せるために行っていた外部記憶装置への入出力がなくなることにより，動作継続制御の高速化が見込める．しかし，不揮発性メモリを主記憶に搭載した計算機は，簡単には入手できない．このため，不揮発性メモリを利用した動作継続制御の実現には，不揮発性メモリの主記憶をエミュレーションする仮想計算機 [4] を利用した．

本稿では，不揮発性メモリの主記憶を前提とした ***Tender*** の動作継続制御の設計について述べる．また，不揮発性メモリの主記憶をエミュレーションする仮想計算機上での実現方式とソースコードの改造量の評価結果を述べる．

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

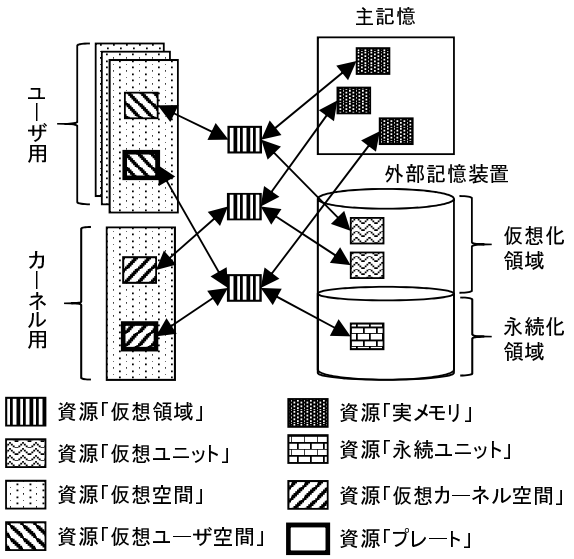

図 1　メモリ関連資源の関係

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では，OS が制御し管理する対象を資源と呼び，資源の分離と独立化を行っている．資源の種類ごとに，資源を管理する管理表と資源を操作するプログラムが存在する．資源は資源名と資源識別子によって識別される．また，***Tender*** 特有の構造に資源インタフェース制御がある．資源インタフェース制御は，表プログラム構造という構造で資源を操作するプログラムを管理する．これにより，資源操作のインタフェースを統一し，各資源を操作するプログラムを資源の種類ごとに分離する．

2.2 メモリ関連資源

Tender におけるメモリ関連資源の関係について，**図 1** に示し，以下で説明する．資源「仮想領域」は，メモリイメージを仮想化した資源であり，実体は資源「実メモリ」上，もしくは外部記憶装置上の領域に存在する．資源「実メモリ」は，主記憶上の領域を表す資源である．資源「仮想ユニット」は，仮想化領域の管理単位である．仮想化領域とは，主記憶上のデータを一時的に保存するための領域であり，既存 OS のスワップ領域を利用して実現している．資源「永続ユニット」は，永続化領域の管理単位である．永続化領域とは，仮想記憶空間上のデータを永続化するために利用する領域であり，既存 OS のファイルシステム領域を利用して実現している．資源「仮想空間」は，特定のアドレス領域をもつ仮想的な空間であり，仮想アドレスから実アドレスへのアドレス変換表に相当する．資源「仮想カーネル空間」は，カーネル用の仮想空間に存在するデータを表す．資源「仮想カーネル空間」は，資源「仮想領域」をカーネル用の資源「仮想空間」に貼り付ける際に生成される．「貼り付ける」とは，仮想アドレスを実アドレスに対応付けすることである．また，仮想アドレスと実アドレスの対応付け解除を「剥がし」と呼ぶ．資源「仮想ユーザ空間」は，ユーザ用の仮想空間に存在するデータを表す．資源「仮想ユーザ空間」は，資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付ける際に生成される．

3. 揮発性メモリを利用した動作継続制御

本章では，文献 [2] と文献 [3] で提案したプレート機能と揮発性メモリを利用した動作継続制御について述べる．

3.1 プレート機能

プレート機能とは，OS が揮発性メモリ上の仮想記憶空間のデータを外部記憶装置上へ自動的に書き出すことにより，データを永続化する機能である．永続化の対象となる領域を「プレート」と呼ぶ．

プレートは，仮想記憶空間上に存在することを基本とし，仮想記憶空間と外部記憶装置の入出力契機を OS が判断する．AP は，必要に応じてプレートを自分の仮想記憶空間にマッピングして利用する．また，AP がプレートの内容の書き出しを OS に依頼することもできる．

また，プレート機能は，計算機再起動時にプレートを復元する．これにより，プレートは，計算機終了前と同じ仮想記憶空間のアドレス領域に存在し続ける性質を持つ．

Tender では，プレート機能を資源「プレート」として実現している．図 1 に示すように，1 つのプレートは実メモリ，仮想領域，永続ユニット，および仮想カーネル空間（もしくは仮想ユーザ空間）からなる．

3.2 揮発性メモリを利用した動作継続制御

3.2.1 処理の流れ

揮発性メモリを利用した動作継続制御は，計算機停止前の処理を計算機再起動後に継続して実行する機能を持つ．揮発性メモリを利用した動作継続制御の処理の流れを**図 2** に示す．揮発性メモリを利用した動作継続制御は，以下の 3 つの処理からなる．

(1) 仮想記憶空間上のデータの永続化
　OS や AP が利用する仮想記憶空間上のデータをプレート機能によって永続化する．
(2) プレートの書き出し
　プレート機能によって管理される仮想記憶空間上のデータを外部記憶装置上の領域に書き出す．
(3) プレートの復元
　プレートの復元処理は，OS の起動処理中に行う．OS の起動処理では，はじめに OS の処理に必要な管理表の確保と初期化を行った後，デバイスを初期化する．次にプレートの復元に必要となる資源管理部の初期化処理を行う．その後，外部記憶装置上にプレート

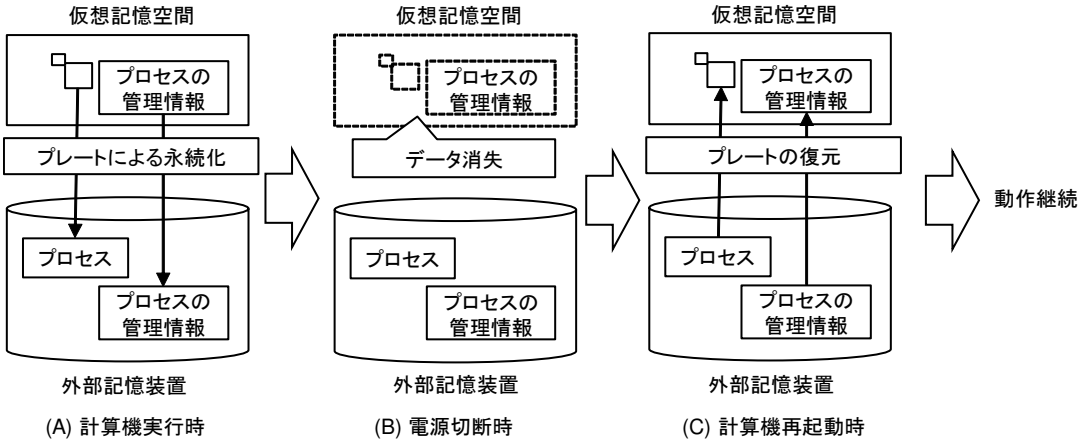

図 2 揮発性メモリを利用した動作継続制御の処理の流れ

管理表が存在する場合，プレート管理表を用いて，外部記憶装置に保存されたデータから仮想記憶空間上にプレートを復元する．最後に，復元したプロセスの動作継続を行う．具体的には，プレートの書き出し処理を依頼した AP の処理に切り替わる．

3.2.2 システムの占有

プレートの書き出し処理の実行中に，プロセスの処理実行や割り込みルーチンの実行によって，AP や OS の利用するデータが更新される可能性がある．これにより，依存関係のある複数のデータ間に不整合が生じると，そのデータを利用する AP が正常に動作できなくなる．また，プレートの書き出し処理により，実入出力終了待ちのプロセスが存在する状態を保存しても，復元後にデバイスからの割り込みは発生しない．このため，プレートの復元後は，実入出力待ちのプロセスは覚醒せず，処理を継続することができない．

永続化したデータを用いて計算機処理を正常に継続するために，資源「システム」を実現している [5]．資源「システム」は，プログラムを実行する際に利用するハードウェア資源の構成を表し，システム管理処理部によって管理される．資源「システム」を構成するハードウェア資源として，プロセッサや外部記憶装置がある．システム管理処理部は，資源「システム」を構成するハードウェア資源の占有機能を提供する．システムの占有を要求された場合，システム管理処理部は，システム占有解除まで他のプロセスやデバイスの割り込みルーチンにより，システムが管理するハードウェア資源を利用されないように制御する．揮発性メモリを利用した動作継続制御では，プレートの書き出しの前にシステムの占有を行い，正常に継続できる計算機処理を保存する．

3.2.3 カーネルスタックの扱い

揮発性メモリを利用した動作継続制御は，プレートの復元処理に利用するカーネルスタックの領域をプレート機能によって永続化しない．この理由を次に示す．プレートの復元処理に利用するカーネルスタックを永続化する場合，カーネルスタックのプレートを復元する際に，外部記憶装置上に保存されたデータで上書きされる．この上書きによって，プレートの復元処理，およびプロセスへの復帰処理ができなくなるためである．

4. 不揮発性メモリを利用した動作継続制御

4.1 目的

不揮発性メモリは電源消失後もデータを保持し続ける特性を持つメモリである．このため，プレート機能によって外部記憶装置上に主記憶のデータを書き出す必要はない．不揮発性メモリを利用した動作継続制御では，主記憶上に残った計算機終了前のデータを利用して，計算機再起動後に処理を継続することを目的とする．

4.2 要件

不揮発性メモリ上に残ったデータを利用して，計算機終了前の処理を計算機再起動後に継続するための要件を以下に示す．

（要件） 不揮発性メモリ上のデータの整合性を保証すること

不揮発性メモリ上に残ったデータを利用して処理を継続する場合，計算機終了前と計算機再起動後のデータの間に不整合があると正常に処理を継続できない．不整合が発生する要因を以下に示す．

（要因 1）OS の起動処理による主記憶の上書き

揮発性メモリを利用した動作継続制御とは異なり，不揮発性メモリの場合，計算機再起動時から主記憶上にデータが存在する．このため，揮発性を利用した場合の処理と同様の起動処理を実行すると，メモリ上のデータを上書きすることになる．上書きされたデータが，OS や AP の利用するデータである場合，正常に処理を継続できなくなる．

（要因 2）計算機終了時のキャッシュの未フラッシュ

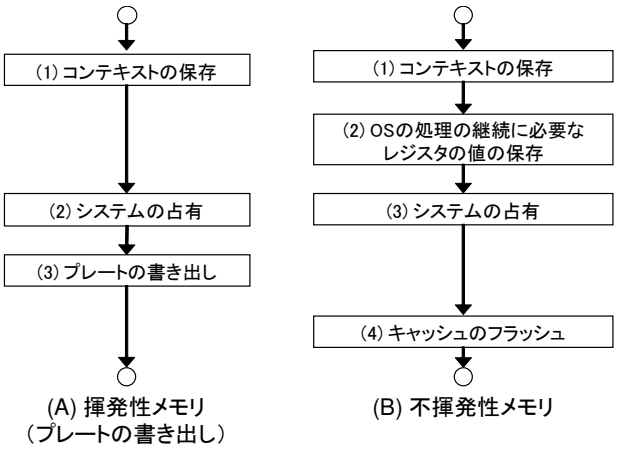

図 3 終了処理の流れ

プロセッサ内部のキャッシュの内容をフラッシュしなければ，キャッシュ上に存在する更新されたデータは，計算機の電源消失とともに消失する．

4.3 設計

4.3.1 対処

4.2 節の要件を満たすために，（要因 1）への対処方法を以下に示す．

（対処 1-A） 起動処理に使用する専用のカーネルスタックの領域の確保

起動処理の実行に計算機終了前に利用していたカーネルスタックと同じ領域を利用すると，スタックのデータが上書きされ，正常に処理を継続できない．このため，起動処理に使用するカーネルスタックは，専用の領域を確保することで対処する．これにより，起動処理を実行することによる主記憶の上書きを防ぐことができる．

（対処 1-B） 不揮発性メモリを主記憶として利用することにより不要となる処理の削除

不要となる処理としては，以下の 2 つがある．

(1) OS の処理に必要な管理表の確保と初期化
(2) プレートの復元に必要となる資源管理部の初期化

OS の処理に必要な管理表には，ページテーブルや例外ベクタ表などがある．揮発性メモリを利用した動作継続制御の場合，起動時にプレートの復元処理を行おうとしても，プレートの復元処理に必要なデータが主記憶上に存在しない．このため，これらの処理を行う必要がある．しかし，不揮発性メモリの場合，主記憶上にデータが保持されているため，これらの処理を行う必要はない．

次に，（要因 2）への対処方法を以下に示す．

（対処 2） キャッシュをフラッシュする処理の追加

終了処理時にキャッシュをフラッシュする処理を実行する．これにより，主記憶とキャッシュの間のデータの一貫性を保ち，正常に処理を継続できるようにする．

4.3.2 終了処理

揮発性メモリを利用した動作継続制御のプレートの書き出し処理の流れを**図 3**(A) に示す．設計した不揮発性メモリを利用した動作継続制御の終了処理の流れを図 3(B) に示し，揮発性メモリのものからの処理の変更部分をとともに説明する．

(B-1) AP の処理の継続に必要なコンテキストを保存する．
(B-2) OS の処理の継続に必要となるレジスタの値を保存する．OS の処理の継続に必要なレジスタとして，ページディレクトリの物理アドレスを格納する CR3 やグローバルディスクリプタテーブルのアドレスを格納する GDTR などがある．これらのレジスタの値は，OS の処理に必要となる管理表の確保と初期化処理時に設定される．（対処 1-B）より，この処理を実行しないため，終了処理でこれらのレジスタの値を保存し，再起動時にレジスタの値を復元できるようにする．
(B-3) システムを占有し，正常に継続できる計算機状態を保存する．
(B-4)（対処 2）より，キャッシュをフラッシュする．一方，揮発性メモリを利用した動作継続制御では，プレートの書き出し処理（図 3(A-3)）の際，主記憶上のデータを外部記憶装置へ PIO 方式で書き出す．これにより，主記憶とキャッシュの間のデータの一貫性が保たれるため，この処理は実行しない．

4.3.3 起動処理

揮発性メモリを利用した動作継続制御の起動処理の流れを**図 4**(A) に示す．設計した不揮発性メモリを利用した動作継続制御の起動処理の流れを図 4(B) に示し，揮発性メモリのものからの処理の変更部分をとともに処理の流れを説明する．

(B-1) 2 回目以降の起動を示すフラグを利用して，処理を継続するかどうか判定する．不揮発性メモリを利用した動作継続制御では，プレート機能を利用しない．このため，処理の継続を判定する条件をプレート管理表の有無の確認（図 4(A-5)）から変更した．2 回目以降のフラグの値を確認し，更新された値であればプロセスへの復帰処理を行う．更新されていなければ，フラグの値を更新（図 4(B-2')）後，通常の OS 起動処理を実行する．
(B-2) 終了処理で保存した OS の処理に必要となるレジスタの値を復元する．
(B-3) すべてのデバイスを初期化し，利用可能な状態にする．一方，揮発性メモリを利用した動作継続制御では，図 4(A-3) でプレートの復元処理に必要なデバイスのみ初期化し，プレートの復元処理後（図 4(A-8)）に残りのデバイスを初期化する．
(B-4) 終了処理時に占有したシステムの占有を解除する．
(B-5) 終了処理を依頼したプロセスへディスパッチする．

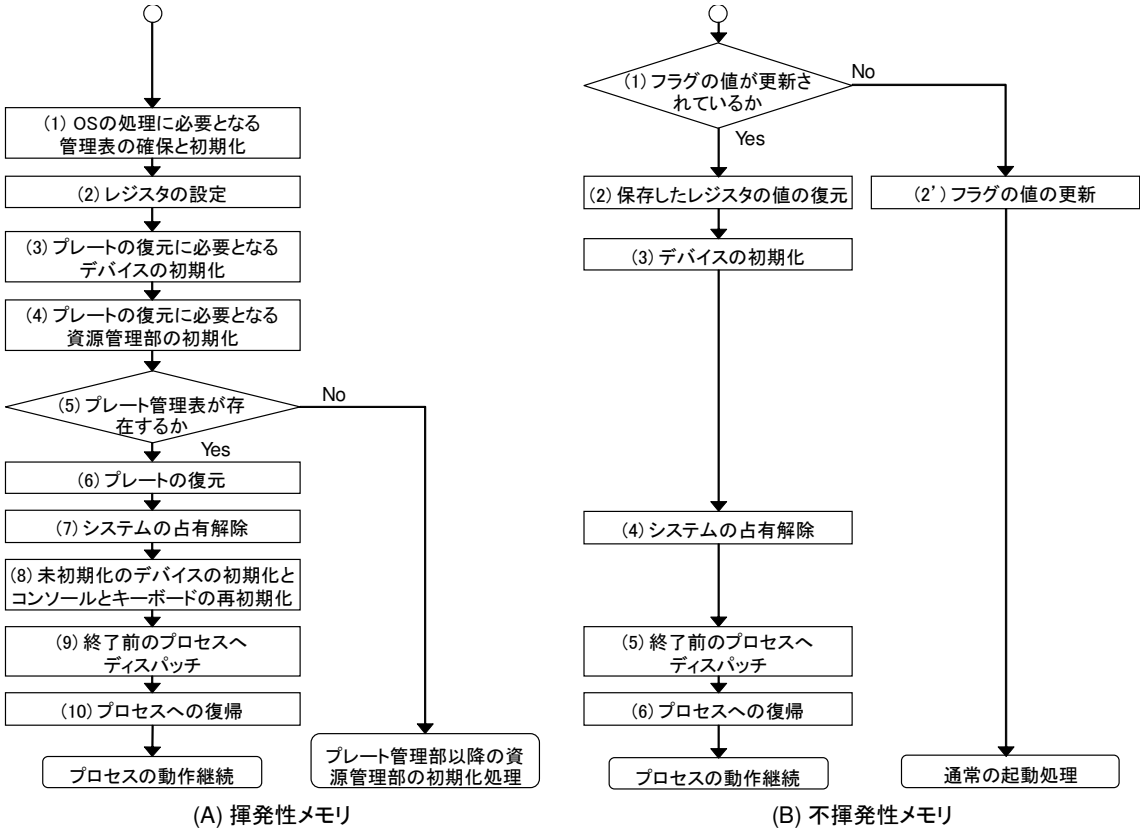

図 4　起動処理の流れ

(B-6) 終了処理を依頼したプロセスへ復帰する．

5. 実現方式

5.1 実現環境

不揮発性メモリを利用した動作継続制御を実現するためには，不揮発性メモリを主記憶として用いた計算機が必要である．しかし，この計算機は，簡単には入手できない．このため，不揮発性メモリの主記憶をエミュレーションする仮想計算機 [4] を利用する．

なお，不揮発性メモリの主記憶をエミュレーションする仮想計算機環境は，主記憶が不揮発性メモリである点を除き，既存の計算機との違いはない．このため，OS が起動するまでに BIOS とブートローダが読み込まれ，走行する．

5.2 課題

不揮発性メモリの主記憶をエミュレーションする仮想計算機環境で，不揮発性メモリを利用した動作継続制御を実現する上での課題を以下に示す．

（課題） 仮想計算機の起動処理における主記憶の上書きへの対処方法の検討

不揮発性メモリを主記憶として用いた仮想計算機環境では，OS の起動処理が開始するまでに BIOS とブートローダが読み込まれ，走行する．これらの処理によって，計算機終了前の主記憶上のデータが上書きされる．仮想計算機

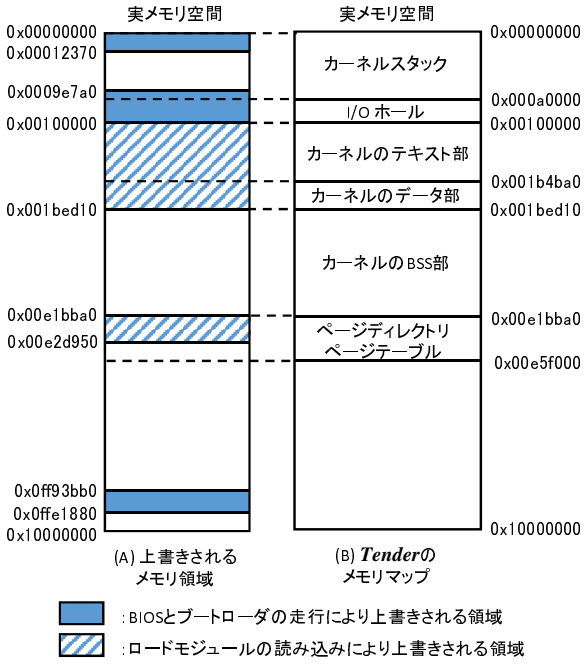

図 5　仮想計算機の起動処理による主記憶の上書きの様子

の起動処理による主記憶の上書きの様子を図 5 に示す．これらの処理により，OS が利用する以下の領域が上書きされる．

(1) カーネルスタック

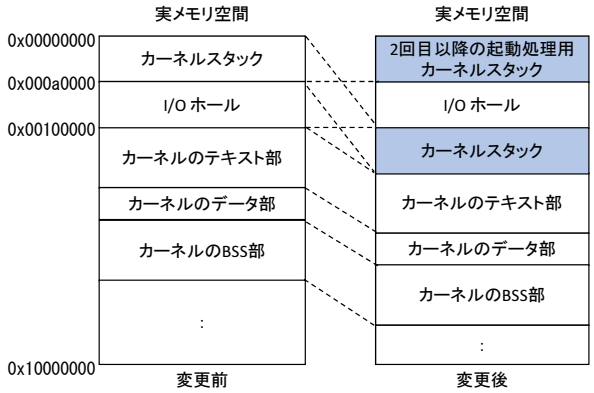

図 6　変更後の ***Tender*** のメモリマップの様子

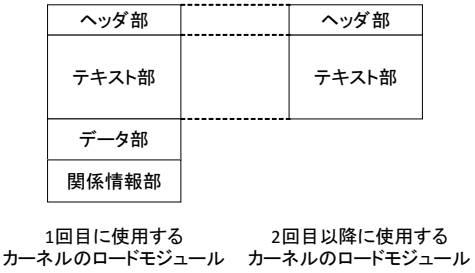

図 7　1 回目と 2 回目以降の起動に使用するカーネルのロードモジュール

(2) カーネルのデータ部
(3) ページディレクトリとページテーブル

OS が利用する領域が上書きされると，正常に計算機処理を継続できない．このため，各領域について上書きの対処方法を検討する．

5.3 対処

5.3.1 カーネルスタック

カーネルスタックの上書きへの対処を以下に示す．

（対処） ***Tender*** のメモリマップを変更し，カーネルスタックを上書きされない領域に配置

変更後の ***Tender*** のメモリマップの様子を図 **6** に示す．デバッグを容易にするため，カーネルスタックを静的に配置するように変更した．具体的には，変更前のカーネルのテキスト部のアドレスから開始できるように，カーネルのロードモジュールを読み込むアドレスをカーネルスタックの大きさの分ずらした．

また，4.3.1 項より，カーネルスタックの領域を 2 回目以降の起動処理用の領域とそうでない領域に分割した．処理の継続に必要なデータは，2 回目以降の起動処理用のカーネルスタックに積まれない．このため，上書きされる領域を 2 回目以降の起動処理用のカーネルスタックの領域として使用しても問題はない．

表 1　実現における起動処理と終了処理の改造量 [LOC]

改造内容	C 言語	アセンブリ言語
(1) 分岐処理，分岐用変数宣言	24	2
(2) 新規追加，部分修正	16	32

5.3.2 カーネルのデータ部およびページディレクトリとページテーブル

カーネルのデータ部が上書きされる原因は，ブートローダが，カーネルのロードモジュールのデータ部を読み込むためである．このため，カーネルスタックのようにメモリマップを変更することで上書きを防ぐことはできない．また，ページディレクトリとページテーブルが上書きされる原因は，ブートローダが，カーネルのロードモジュールの関係情報部を読み込むためである．これをふまえた上書きへの対処を以下に示す．

（対処）1 回目の起動では通常のカーネルのロードモジュールを使用し，2 回目以降の起動ではデータ部と関係情報部を除いたカーネルのロードモジュールを使用

1 回目と 2 回目以降の起動に使用するカーネルのロードモジュールを図 **7** に示す．2 回目以降の起動に使用するカーネルのロードモジュールから，データ部と関係情報部を取り除く．これにより，ブートローダがデータ部と関係情報部のデータをメモリに読み込まないため，カーネルのデータ部，およびページディレクトリとページテーブルが上書きされることを防ぐことができる．ただし，この対処方法を採用する場合，利用者が起動に使用するカーネルを選択できなければならない．これは，起動に使用するカーネルをブートローダに登録し，利用者が計算機の起動時に選択できるようにすることで対処する．

5.4 改造量

不揮発性メモリを利用した動作継続制御の実現に要した工数を明らかにするため，起動処理と終了処理の改造量について評価した．メモリマップ変更後のソースコードと不揮発性メモリを利用した動作継続制御実装後のソースコードの論理 LOC（Lines of Code）を評価基準として使用し，比較した．

実現における起動処理と終了処理の改造量を表 1 に示す．表には，C 言語で記述したプログラムとアセンブリ言語で記述したプログラムの改造量を示す．改造内容は，以下の 2 つに分類できる．

(1) 不要となる処理を実行しないために追加した分岐処理，および分岐のために使用する変数の宣言文（分岐処理，分岐用変数宣言）
(2) 揮発性メモリを利用した動作継続制御から新たに追加した処理，および処理の一部を修正した処理（新規追加，部分修正）

(2) について，C 言語で記述したプログラムの内，新たに

追加した処理は，終了処理の図 3(B-2) と図 3(B-4) である．また，処理の一部を修正した処理は，起動処理の図 4(B-2) と図 4(B-3) である．アセンブリ言語で記述したプログラムの内，新たに追加した処理は，起動処理の図 4(B-1) と図 4(B-2') である．

C 言語で記述したプログラムについて，揮発性メモリを利用した動作継続制御から流用できる処理が多かったため，新規に追加した処理より分岐処理の方が多い結果となった．また，図 4(B-4) 以降の処理について，揮発性メモリを利用した動作継続制御から修正した部分はない．このことから，揮発性メモリを利用した動作継続制御の処理を利用することで，実現に要する工数を抑えられたといえる．

6. 関連研究

不揮発性メモリを利用したシステムソフトウェアの研究として，pVM[6] がある．pVM は，仮想ファイルシステムではなく仮想記憶を拡張し，不揮発性メモリを NUMA ノードとして管理することで，メモリ容量を拡大する．また，永続的なメタデータを追加し，仮想記憶をさらに拡張することで，高速なオブジェクトストレージを提供する．一方，不揮発性メモリを利用した動作継続制御は，不揮発性メモリ上にＡＰの利用する一部のデータだけ配置するのではなく，処理の継続に必要となるデータをすべて配置する．

不揮発性メモリを利用したファイルシステムの研究として，不揮発性メモリとディスクにまたがる階層型ファイルシステムの Ziggurat[7] がある．Ziggurat は，ファイルアクセスの記録を分析した結果に基づいて，データの配置と移行を行い，不揮発性メモリの利点である高帯域幅と低レイテンシを活用する．たとえば，同期的に更新されるファイルへの書き込みは，同期のオーバヘッドを最小限にするために不揮発性メモリ層に配置する．一方，不揮発性メモリを利用した動作継続制御は，揮発性メモリと比較した際の利点である永続性を活用し，計算機処理を永続化する．

不揮発性メモリを利用したデータベースの研究として，HiKV[8] と MyNVM[9] がある．HiKV は，ハッシュインデックスを不揮発性メモリ上に配置し，B^+ ツリーインデックスを DRAM 上に配置することで，効率的にハイブリッドインデックスを利用する．また，システム障害が発生した場合，不揮発性メモリ上のハッシュインデックスをもとに，B^+ ツリーインデックスを再構築する．一方，不揮発性メモリを利用した動作継続制御は，混載環境ではなく，不揮発性メモリのみの環境を想定して設計している．

MyNVM は，サーバにかかる費用を減らすために，DRAM キャッシュの一部をブロックデバイスの不揮発性メモリに置き換える．また，不揮発性メモリに置き換えることによって発生する問題点を割り込みレイテンシのオーバヘッドの削減などにより解決する．一方，不揮発性メモリを利用した動作継続制御は，バイトアドレス可能な不揮発性メモリを利用する．

不揮発性メモリを利用したプログラミング言語の研究として，文献 [10] がある．文献 [10] は，不揮発性メモリだけでなく，様々な特徴を持ったメモリを活かすため，プログラマが，データ領域とコード領域の両方でメモリの特徴を活用できるようにする言語拡張を提案している．一方，不揮発性メモリを利用した動作継続制御は，プログラマは意識せずとも不揮発性メモリの特徴である永続性を利用できる．

7. おわりに

不揮発性メモリを利用した ***Tender*** の動作継続制御の設計について述べた．不揮発性メモリ上に残ったデータを利用して，計算機終了前の処理を計算機再起動後に継続するためには，この前後のデータの整合性を保証する必要がある．このために，起動処理では，管理表の確保と初期化処理を実行しない．また，起動処理に使用する専用のカーネルスタックの領域を用意し，起動処理の実行による主記憶の上書きを防ぐ．終了処理では，プロセッサ内部のキャッシュをフラッシュする処理を追加する．

また，不揮発性メモリの主記憶をエミュレーションする仮想計算機上での実現方式について述べた．仮想計算機の起動処理による主記憶の上書きへ対処するために，メモリマップを変更し，2 回目以降の起動にはデータ部と関係情報部を除いたカーネルのロードモジュールを使用する．

残された課題として，不揮発性メモリを利用した動作継続制御の性能評価と計算機の緊急停止時の処理の継続方法の検討がある．

謝辞 本研究の一部は，JSPS KAKENHI 18K11244 による．

参考文献

[1] 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏：資源の独立化機構による ***Tender*** オペレーティングシステム，情報処理学会論文誌， Vol. 41, No. 12, pp. 3363–3374 (2000).
[2] Toshihiro Yamauchi, Yuta Yamamoto, Kengo Nagai, Tsukasa Matono, Shinji Inamoto, Masaya Ichikawa, Masataka Goto, Hideo Taniguchi: Plate: Persistent Memory Management for Nonvolatile Main Memory, Proceedings of 31st ACM Symposium on Applied Computing (SAC 2016), pp.1885-1892, ACM (2016).
[3] 山本悠太，田端利宏，谷口秀夫：***Tender*** の動作継続制御機能における入出力デバイスの扱い，情報処理学会研究報告，2008-OS-109，Vol.2008，No.77，pp.61–68 (2008).
[4] 川岸昇，渡辺優，山内利宏，谷口秀夫：QEMU を利用した不揮発性メモリ搭載計算機の実現，平成 29 年度（第 68 回）電気・情報関連学会中国支部連合大会講演論文集，電子媒体 (2017).
[5] 長井 健悟，山本 悠太，山内 利宏，谷口 秀夫：***Tender*** の世代管理機能の実現，情報処理学会研究報告，Vol.2010-OS-115，No.2，電子媒体 (2010).
[6] Sudarsun Kannan, Ada Gavrilovska, Karsten Schwan:

pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage , Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16), Article No.13 (2016).

[7] Shengan Zheng, Morteza Hoseinzadeh, Steven Swanson: Ziggurat: A Tiered File System for Non-Volatile Main Memories and Disks , Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19), pp.207–219 (2019).

[8] Fei Xia, Dejun Jiang, Jin Xiong, Ninghui Sun: HiKV: A Hybrid Index Key-Value Store for DRAM-NVM Memory Systems , Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'17), pp.349–362 (2017).

[9] Assaf Eisenman, Darryl Gardner, Islam AbdelRahman, Jens Axboe, Siying Dong, Kim Hazelwood, Chris Petersen, Asaf Cidon, Sachin Katti: Reducing DRAM Footprint with NVM in Facebook , Proceedings of the Thirteenth EuroSys Conference (EuroSys '18), Article No.42 (2018).

[10] 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).