2008－OS－109 (9)
2008／8／7

# *Tender*の動作継続制御機能における入出力デバイスの扱い

山本悠太† 田端利宏† 谷口秀夫†

計算機の主記憶は揮発性メモリが用いられるため，電源切断により，主記憶上のデータは消失する．このため，処理途中に計算機が緊急停止した場合，再び計算機の電源を投入しても計算機停止前に行っていた処理を継続できない．そこで，我々は，***Tender***オペレーティングシステムにおいて，仮想記憶機構を利用した主記憶上のデータを不揮発性化するプレート機能を利用し，仮想記憶空間上のすべてのデータを不揮発性化し，計算機処理を永続化する動作継続制御機能を提案した．しかし，動作継続制御機能では，入出力デバイスの状態を考慮していない．このため，入出力処理中にプレート書き出しを行った場合，プレート復元後に入出力処理を継続できず，動作継続できない．そこで，本論文では，動作継続制御機能において入出力処理の継続を可能にするデバイス制御法について述べる．

# I/O Device Management of Persistent Mechanism on *Tender*

Yuta Yamamoto,† Toshihiro Tabata† and Hideo Taniguchi †

Data on main memory is no longer available when a computer is turned off, because volatile memory is used for main memory. Therefore, if a computer shutdowns unexpectedly, a computer cannot continue processing before the shutdown. Thus, we propose persistent mechanism for computer processing using "plate" function. Plate manages a persistent data on virtual memory space. However, this persistent mechanism ignores status of I/O devices. If plate persists data during I/O, a computer cannot continue processing after plate restore data. In this paper, we explain I/O device management method on persistent mechanism.

## 1. はじめに

計算機は，プログラムやデータを主記憶上に配置し，実行または処理する．計算機の主記憶として用いられるメモリの主流は，揮発性メモリであるため，計算機の電源が切断されると，主記憶上のデータは消失する．したがって，処理途中に計算機が緊急停止した場合，再び計算機の電源を投入しても計算機停止前に行っていた処理を継続できない．

このため，既存のオペレーティングシステム（以降，OS と略す）は，不揮発性記憶媒体である外部記憶装置上にファイルシステムを構築し，揮発性メモリ上のプログラムやデータをファイルとして保存することにより，プログラムやデータを再利用可能にしている．ただし，既存 OS は，ファイルに対するインタフェースを応用プログラム（以降，AP と略す）に提供し，ファイル操作を AP に委ねている．したがって，AP 作成者や AP 利用者が，計算機の緊急停止を意識してメモリの永続化をする必要がある．

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

計算機処理の中断と再開を可能にする機能として，Windows の休止機能や Linux の swsusp，TuxOnIce[1] のようなハイバネーション機能がある．ハイバネーション機能では，主記憶全体を外部記憶装置に保存し，計算機再起動後に保存したデータを主記憶に読み込む．外部記憶装置へのデータの書き出し処理と外部記憶装置からのデータの読み込み処理は，主記憶の大きさに比例し，処理時間は長くなる．また，ハイバネーション機能は，利用者が実行契機を与える必要があるため，予見できる停止には有効であるが，予見できない緊急停止には全く役に立たない．

KeyKOS[2] や L4Ka[3]，および Grasshopper[4] のような，計算機が緊急停止しても，電源再投入後，計算機停止前から処理を再開する機能を持つ OS の研究も行われている．しかし，これらの研究では，AP の処理のみ永続化可能であり，OS の処理を永続化できない．

我々は，***Tender***オペレーティングシステム[5](以降，***Tender*** と略す) において，仮想記憶機構を利用したメモリ上のデータを不揮発化するプレート機能[6] を実現した．プレート機能では，OS が定期的に仮想記憶空間上のデータを更新部分だけ外部記憶装置へ書き

出し，仮想記憶空間上のデータを永続化する．計算機再起動時には，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．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を不揮発性化する．プレート機能を用いてカーネル用とユーザ用の仮想記憶空間上に存在する全てのデータを不揮発化することにより，計算機処理を永続化する動作継続制御機能[7]を提案している．これにより，計算機が緊急停止した場合でも，最後に保存したプレートの状態から処理を継続でき，計算機の緊急停止による処理の消失を軽減できる．しかし，動作継続制御機能では，入出力デバイスの状態を考慮していない．このため，入出力処理中にプレート書き出しを行った場合，プレート復元後に入出力処理を継続できず，動作継続できない．

そこで，本論文では，動作継続制御機能において入出力処理の継続を可能にするデバイス制御法について述べる．具体的には，プレート書き出しの際，全てのドライバが実入出力を行っていない状態 (以降，非実入出力状態と略す) へ移行してから，プレートの書き出しを行う．これにより，ドライバが非実入出力状態で保存され，プレート復元後に入出力処理を継続できる．

## 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して，分離し，独立化している．資源には，資源名と資源識別子を付与し，資源操作のインタフェースを統一している．

### 2.2 メモリ関連資源

***Tender*** の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「仮想空間」とは，特定のアドレス領域を持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「仮想ユーザ空間」は，メモリイメージを仮想化した領域である資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付けることで作成できる．「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り，具体的には，当該の仮想アドレスに対応するアドレス変換表のエントリに，実アドレスまたは外部記憶装置のアドレスを設定する．一方，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「仮想領域」の実体は，資源「実メモリ」または外部記憶装置上に存在する．外部記憶装置上のいずれかに存在する．外部記憶装置上の領域の種類として，資源「永続ユニット」と資源「仮想ユニット」の 2 種類

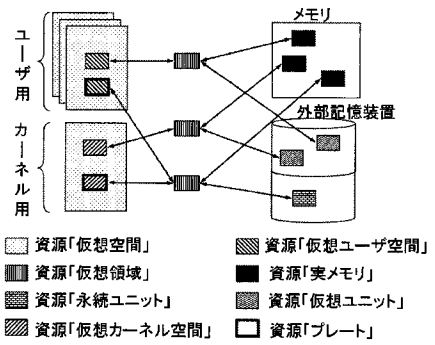


図 1　メモリ関連の資源の関係

がある．資源「永続ユニット」とは，外部記憶装置上の永続化領域の管理単位である．永続化領域は，既存 OS のファイルシステム領域に相当する．資源「仮想ユニット」とは，外部記憶装置上の仮想化領域の管理単位である．仮想化領域は，既存 OS のスワップ領域に相当する．資源「仮想カーネル空間」は，資源「仮想領域」を OS 用の資源「仮想空間」に貼り付けることにより作成される．

### 2.3 プレート機能

プレート機能とは，永続化の対象になる仮想記憶空間上の領域の永続化を OS が自動的に行うことにより，仮想記憶空間上のデータを不揮発性する機能である．この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，計算機の電源投入時にプレートを復元する．これにより，プレートは，計算機の電源再投入後でも，計算機の電源切断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，一つのプレートは仮想カーネル空間，仮想ユーザ空間，仮想領域，実メモリ，および永続ユニットからなる．各プロセスは，プレートに対応する仮想領域を，自分の仮想空間に貼り付ける（仮想ユーザ空間を生成する）ことにより，プレートにアクセスする．

### 2.4 動作継続制御機能

#### 2.4.1 目的と流れ

動作継続制御機能は，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ことを目的とする．動作継続制御機能の処理の流れを以下に示す．

(1) 仮想記憶空間上のデータ不揮発化

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り不揮発化する．

(2) プレートの書き出し

プレート化された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

プレート管理部初期化時，外部記憶装置上にプレート管理表が存在している場合，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．その後，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を行った AP に切り替わる．

#### 2.4.2 仮想記憶空間上のデータ不揮発性化

***Tender*** では，OS や AP が利用する仮想記憶空間上のデータは，仮想カーネル空間か仮想ユーザ空間によって管理される．動作継続制御機能が無効であるとき，これらのデータは揮発性である．動作継続制御機能の有効化処理により，現存するすべての仮想カーネル空間と仮想ユーザ空間をプレート化し，これらのデータに不揮発性を付与する．「プレート化」とは，仮想カーネル空間や仮想ユーザ空間を利用してプレートを生成することを意味する．動作継続制御機能の有効化処理以降に，新たに生成された仮想カーネル空間と仮想ユーザ空間は，自動でプレート化される．

#### 2.4.3 プレートの書き出し

プレートの書き出しでは，プレート上のデータを外部記憶装置上の領域に書き出す．このとき，仮想記憶空間上と外部記憶装置上の全データの整合性を保持するため，すべてのプレートにおける更新部分のみを書き出す．

#### 2.4.4 プレートの復元

プレートによって管理されるデータは，計算機処理の継続のために，計算機の再起動後も仮想記憶空間上に存在することが必要である．そこで，OS 起動処理でのプレート管理初期化時にプレートの復元機構により，すべてのプレートをプレートの書き出し時の状態で仮想記憶空間上に復元する．復元処理の流れを以降で述べる．

＜プレート復元処理の流れ＞

動作継続制御では，プレートを利用し，仮想記憶空間上のすべてのデータを不揮発性化する．プレートの中には，依存関係を持つものがあり，復元の順序を考慮しなければならないプレートが存在する．復元できないプレートも存在する．したがって，プレートの復元処理では，プレートに格納されているデータを意識した処理を要する．プレートの復元処理の流れを以下で説明する．

(1) プレート管理表の復元

プレート管理表には，各プレートの復元に必要な情報（仮想アドレス，永続ユニット名，大きさ）が格納されている．このため，プレートの復元処理では，初めにプレート管理表を復元する．以降のプレートの復元処理では，プレート管理表を参照し，復元対象のプレートが存在していた仮想アドレスに永続ユニットからデータを読み込むことでプレートを復元する．

(2) 仮想カーネル空間利用で生成したプレートの復元

仮想カーネル空間には，各資源の管理表などの OS が利用するメモリ領域以外に仮想空間のページディレクトリやページテーブルが存在する．仮想空間を復元しておかないと，当該仮想空間を使用するプレートを復元できない．このため，仮想カーネル空間を利用して生成したプレートを他の生成方法のプレートより先に復元する．

(3) 仮想カーネル空間利用以外のプレートの復元

仮想カーネル空間利用以外で生成したプレートを復元する．

#### 2.4.5 動作継続制御機能の特徴

この動作継続方式の利点を述べる．

(1) 計算機の緊急停止への対処

ファイル機能やハイバネーション機能では，主記憶上のデータを外部記憶装置へ書き出す契機を利用者が指定しなくてはならない．これに対し，動作継続制御では，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 OS が定期的に外部記憶装置へ書き出す．このため，計算機が緊急停止した場合，動作継続制御では，最後に保存したプレートの状態から処理を継続でき，計算機の緊急停止による処理の消失を軽減できる．

(2) 計算機再起動後の処理容易化

ファイル機能では，永続化されたデータを計算機再起動後に利用する場合，利用者が当該データを利用するための AP の設定，および実行をしなくてはならない．これに対し，動作継続制御では，計算機再起動時に仮想記憶空間上のすべてのデータが計算機停止前の状態に復元されるため，利用者は計算機再起動後に特定の処理をしなくてよい．

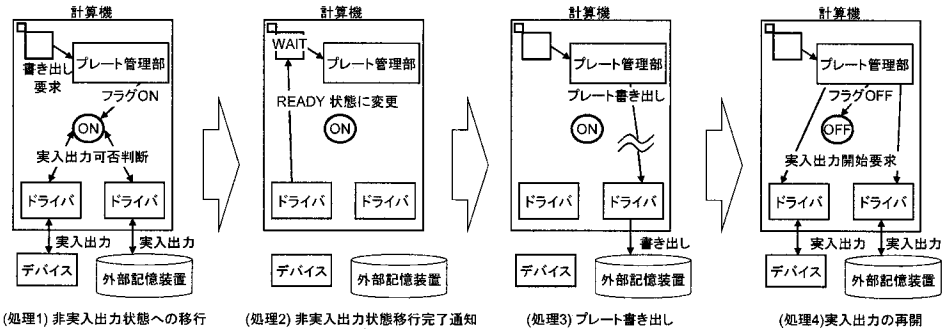


**図 2** 基本方式における処理の流れ

## 3. 動作継続制御機能における入出力デバイスの扱い

### 3.1 基本方式

動作継続制御機能では，入出力デバイスの状態を考慮していない．このため，ドライバが実入出力を行っている状態 (以降，実入出力状態と略す) でプレートの書き出しを行い，ドライバの実入出力状態を書き出しても，プレート復元後のデバイスの状態は，実入出力時の状態とは異なるため，入出力処理を継続できない．

そこで，プレート書き出しの際，ドライバが実入出力状態で書き出されるのを避けることにする．具体的には，プレート書き出しの際，全てのドライバが非実入出力状態に移行してから，プレートの書き出しを行う．

基本方式における処理の流れを図 2 に示す．図 2 に示す通り，基本方式の処理は 4 つの処理に分けられる．各処理について以下で説明する．

(処理 1)非実入出力状態への移行

プレート書き出し要求により，プレート管理部は，非実入出力状態移行フラグを ON にする．このとき，実入出力状態のドライバが存在しなければ，(処理 3) を行う．そうでない場合，プレート書き出しを要求したプロセスは，全てのドライバが非実入出力状態に移行するまで，WAIT 状態に移行する．実入出力状態のドライバは，実入出力終了後，非実入出力状態移行要求フラグを確認する．非実入出力状態移行要求フラグが ON であれば，新たな実入出力を中断する．また，非実入出力状態のドライバは，要求を処理する前に，非実入出力状態移行要求フラグを確認し，新たに実入出力状態にならないように処理を中断する．

(処理 2)非実入出力状態移行完了通知

各ドライバは，実入出力状態から非実入出力状態へ移行した後，全てのドライバの状態を確認する．全てのドライバが非実入出力状態であれば，プレート書き出しを要求したプロセスを READY 状態にする．

(処理 3)プレート書き出し

プレート管理部はプレート書き出し処理を行う．その際，プレート書き出し以外の処理は実行されない．

(処理 4)実入出力の再開

プレート管理部は，プレート書き出し完了後，非実入出力状態移行要求フラグを OFF にし，全てのドライバの実入出力処理を再開させる．

計算機再起動時，プレート復元により，計算機は (処理 3) 実行後の状態で復元される．このため，プレート復元後，(処理 4) を行う．

### 3.2 課　　題

提案方式における課題は以下の 3 つである．

(課題 1)ドライバの状態管理

(課題 2)実入出力の中断と再開に関する検討

(課題 3)プレート書き出しで利用するドライバの扱い

上記の課題について，以下で説明する．

(課題 1) ドライバの状態管理

提案方式では，全てのドライバが非実入出力状態に移行してからプレートの書き出しを行う．ここで，各ドライバは，非実入出力状態移行後，各ドライバの状態を確認し，非実入出力状態移行完了を検知する．このためには，各ドライバの状態管理方法を検討する必要がある．

(課題 2) 実入出力の中断と再開に関する検討

非実入出力状態移行時，実入出力状態のドライバの処理を中断し，非実入出力状態にする方法を検討する

必要がある．また，非実入出力状態のドライバは，入出力要求がきても，実入出力を開始しないようにする必要がある．

プレート書き出し終了後，各ドライバは，非実入出力状態移行時に中断した実入出力を開始できる必要がある．

(課題3) プレート書き出しで利用するドライバの扱い

ディスクドライバのように，プレート書き出しのために実入出力を行うドライバが存在する．これらのドライバは，プレート書き出し時，非実入出力状態への移行が完了する以前の入出力要求を保持したまま，プレート書き出しを行う必要がある．

### 3.3 対　　処

#### 3.3.1 ドライバの状態管理

非実入出力状態移行完了の検知を行うためには，全てのデバイスの状態を把握できなければならない．このためには，ドライバの状態管理を行う必要がある．このために，ドライバ状態管理表を用いる．ドライバ管理表を図3に示す．

各ドライバは，実入出力開始前に，ドライバの状態を実入出力状態に変更し，実入出力完了後に，ドライバの状態を非実入出力状態に変更する．また，非実入出力状態移行要求時，各ドライバは，非実入出力状態移行後，全てのドライバの状態を参照し，実入出力状態のドライバが存在するか否かを確認する．これにより，非実入出力状態移行完了を検知する．

#### 3.3.2 実入出力の中断と再開に関する検討

実入出力を中断するには，ドライバはプレート書き出しが終了するまで入出力要求を処理せずに保持しておき，実入出力を行わなければよい．実現方式を以下に示す．

通常時におけるドライバの処理を図4に示す．ドライバは，入出力要求を受け取ると，入出力要求を格納する構造体に入出力要求を格納し，それを要求待ちキューにつな�ぐ．ドライバは，現在実行中の実入出力が終了すると，要求待ちキューに入出力要求がつながっていれば，要求待ちキューの先頭から，入出力要求を取り出し，実入出力を開始する．

非実入出力状態移行時におけるドライバの処理を図5に示す．ドライバは，入出力要求を受け取ると，通常時と同様，入出力要求を格納する構造体に入出力要求を格納し，それを要求待ちキューにつなぐ．ドライバは，現在実行中の実入出力が終了後，要求待ちキューに入出力要求がつながれていても無視し，実入出力を開始しない．

実入出力を再開するには，ドライバが要求待ちキューの先頭から入出力要求を取り出し，実入出力を再開すればよい．実入出力再開時におけるドライバの処理を図6に示す．プレート書き出し後，およびプレート復元後，各ドライバの要求待ちキューの先頭につながれている入出力要求を取り出し，実入出力を行う．

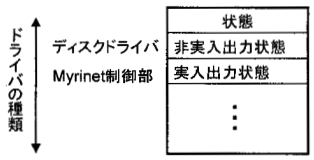


図3　ドライバ状態管理表

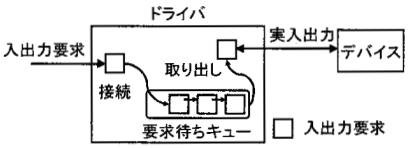


図4　通常時におけるドライバの処理

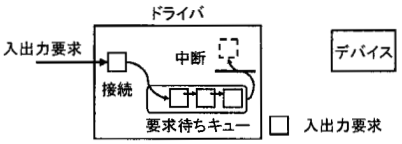


図5　非実入出力状態移行時におけるドライバの処理

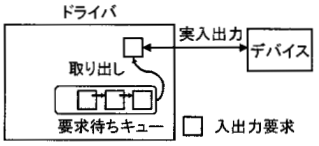


図6　実入出力再開時におけるドライバの処理

#### 3.3.3 プレート書き出しで利用するドライバの扱い

プレート書き出しで利用するドライバは，非実入出力状態への移行が完了する以前の入出力要求を保持した状態で，実入出力を行う．実現方法を以下で説明する．プレート書き出し時におけるプレート書き出しで利用するドライバの処理を図7に示す．

通常時，ドライバは，図4で示す通り，入出力要求を要求待ちキューに接続し，ドライバは要求待ちキューから入出力要求を取り出して実入出力を行う．プレート書き出し時，ドライバは，図7に示す通り，入出力要求を要求待ちキューに接続せず，直接実入出力を行う．これにより，ドライバは，入出力要求を保持したまま実入出力を行うことが可能となる．保持した入出

表 1　ドライバ状態管理の提供インタフェース

| 機能 | インタフェース | 機能内容 |
|---|---|---|
| ドライバの状態変更 | change_driver_status(type,status) | type で示すドライバの状態を status に変更する. |
| ドライバの状態取得 | check_driver_status(type) | type で示すドライバの状態を取得する. type が 0 の場合, 実入出力状態のドライバが存在するか否かを取得する. |

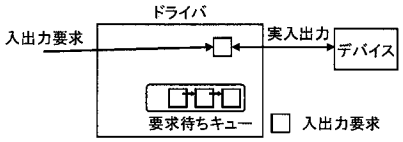


**図 7**　プレート書き出し時におけるプレート書き出しで利用するドライバの処理

力要求は，プレート書き出しにより，外部記憶装置に保存される．プレート復元後，復元した入出力要求に対して実入出力を行うことが可能となる．

## 4. 実現方式

### 4.1 ドライバ状態管理部

ドライバ状態管理部は，ドライバの状態を管理する．ドライバの状態管理部は以下の機能を各ドライバとプレート管理部に提供する．

(1)　ドライバの状態変更
(2)　ドライバの状態取得

ドライバ状態管理の提供インタフェースを表 1 に示す．各ドライバとプレート管理部は，表 1 のインタフェースを利用することにより，ドライバの状態変更と状態取得が可能になる．

### 4.2 プレート管理部の処理

実現方式におけるプレート管理部の処理を以下に示す．

＜プレート書き出し時の処理＞

プレート書き出し時における処理の流れを図 8 に示す．

(1)　非実入出力状態移行要求フラグを ON にする．
(2)　実入出力状態のドライバが存在するか確認する．
(3)　実入出力状態のドライバが存在すれば，全てのデバイスが，非実入出力状態に移行するまで WAIT 状態になる．
(4)　プレートの書き出しを行う．
(5)　非実入出力状態移行要求フラグを OFF にする．
(6)　全てのデバイスにおける実入出力を再開する．

＜プレート復元時の処理＞

プレート復元時における処理の流れを図 9 に示す．太枠の箇所が実現方式において追加した処理である．各処理について，以下で説明する．

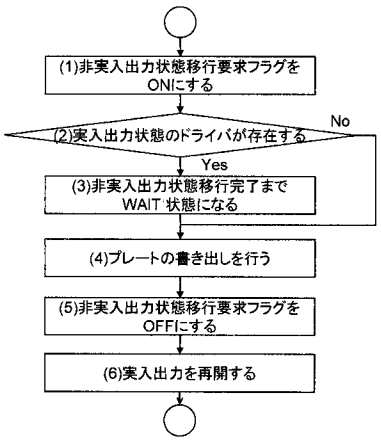


**図 8**　プレート書き出し時における処理の流れ

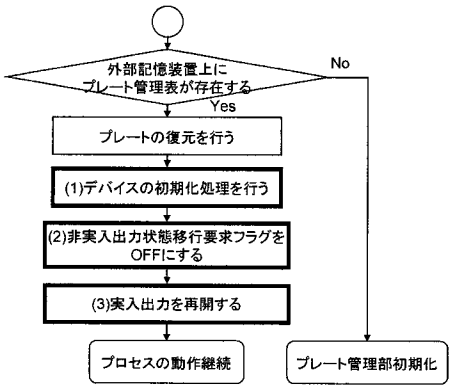


**図 9**　プレート復元時における処理の流れ

(1)　デバイスの初期化処理を行う．具体的には，各ドライバごとに用意されているデバイスの初期化プログラムを呼び出し，デバイスの初期化を行う．
(2)　非実入出力状態移行要求フラグを OFF にする．
(3)　全てのデバイスにおける実入出力を再開する．

### 4.3 ディスクドライバの制御

#### 4.3.1 制御方式

永続化領域への実入出力の流れを図 10 に示す．動作継続制御機能は，プレート機能を用いて，仮想記憶空間上の全てのデータを不揮発化する．プレートの永

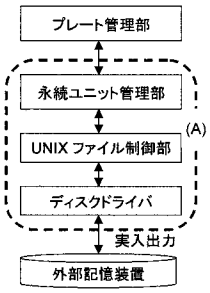


**図 10** 永続化領域への実入出力の流れ

続化には，永続化領域を用いる．永続化領域は，既存の OS のファイルシステム領域に相当し，ファイルシステムの制御部として，FFS(Fast File System) を実現している．FFS 上のファイルを制御する制御部を UNIX ファイル制御部と呼んでいる．プレートが永続化領域へデータの読み書きを行う際，永続ユニット管理部は，UNIX ファイル制御部を利用して実入出力を行う．

永続化領域への実入出力中にプレート書き出しを行う場合，実入出力中の UNIX ファイル制御部の状態を保存する必要がある．しかし，プレート書き出し時に UNIX ファイル制御部を用いるため，UNIX ファイル制御部の状態が変化する．したがって，プレート書き出し前に UNIX ファイル制御部の状態を退避し，プレート書き出し後に UNIX ファイル制御部の状態を復元する必要がある．また，UNIX ファイル制御部の状態の退避，復元を行うには，複雑な処理を要する．

そこで，今回の実装では，処理を単純化するため，図 10 の (A) をディスクドライバとみなし，実入出力の中断は永続ユニット管理部で行う．これにより，UNIX ファイル制御部の状態の退避，復元を行う必要はなくなる．

### 4.3.2 実入出力処理の流れ

実現方式におけるディスクドライバの実入出力処理の流れを図 11 に示す．ここで，図 10 の (A) 間の処理を行っているプロセスが存在すると，プレート書き出しを行うことができない．そこで，図 10 の (A) 間の処理を行っているプロセスが存在するか否かを確認するため，図 10 の (A) 間の処理を行っているプロセスの数 (以降，実入出力状態プロセス数と略す) を管理する．実入出力状態プロセス数が 1 以上になると，ディスクドライバの状態を実入出力状態にし，実入出力状態プロセス数が 0 になると，ディスクドライバの状態を非実入出力状態にする．

実入出力を行う前に，前処理と後処理を行う．プレート書き出し時，前処理と後処理はスキップし，実入出力を行う．各処理の流れを以下で説明する．

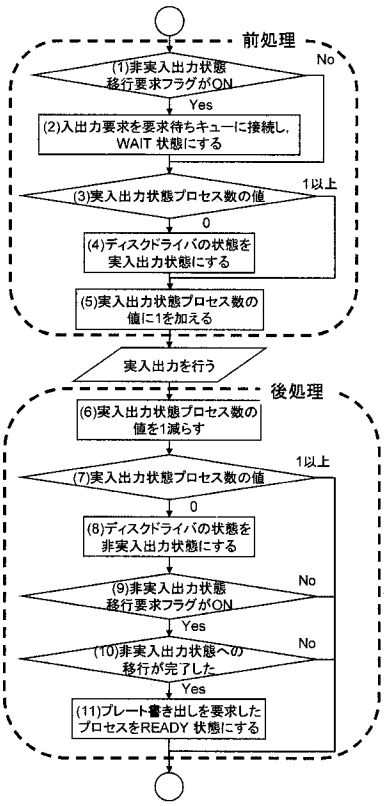


**図 11** ディスクドライバの実入出力処理の流れ

＜前処理＞

(1) 非実入出力状態移行フラグが ON であるか確認する．

(2) 非実入出力状態移行フラグが ON であれば，入出力要求にプロセス識別子を格納し，入出力要求を要求待ちキューにつなぐ．その後，入出力要求を行ったプロセスを WAIT 状態にする．

(3) 実入出力状態プロセス数を確認する．

(4) 実入出力状態プロセス数が 0 であれば，ディスクドライバの状態を実入出力状態にする．

(5) 実入出力状態プロセス数の値を 1 増やす．

＜後処理＞

(6) 実入出力状態プロセス数の値を 1 減らす．

(7) 実入出力状態プロセス数の値を確認する．実入出力状態プロセス数の値が 1 以上であれば，後処理を終了する．

(8) ディスクドライバの状態を非実入出力状態に

する．

(9) 非実入出力状態移行フラグが ON であるか確認する．非実入出力状態移行フラグが OFF であれば，後処理を終了する．

(10) 非実入出力状態への移行が完了したか確認する．

(11) 非実入出力状態への移行が完了していれば，プレート書き出しを要求したプロセスを READY 状態にする．

#### 4.3.3 実入出力の再開処理とデバイスの初期化処理

ディスクドライバにおける実入出力の再開処理では，実入出力の前処理で要求待ちキューに格納した入出力要求に対する実入出力を再開する．具体的には，入出力要求に格納されたプロセス識別子を元に，入出力要求を出したプロセスを READY 状態にする．

ディスクドライバは，プレート復元前にデバイスの初期化を行っている．したがって，プレート復元後に，ディスクドライバは，デバイスの初期化処理を行わない．

## 5. 評　　価

提案方式の実装による実入出力のオーバヘッドを明らかにするため，***Tender*** を Pentium III 550MHz，搭載メモリ 128MB の計算機で走行させ，以下の測定を行った．非入出力状態移行要求フラグは OFF とする．

(測定 1) ドライバの状態変更処理時間

(測定 2) 永続ユニットにおける読み込みと書き出しにかかる処理時間

(測定 1) ドライバの状態変更にかかる時間を測定した．

(測定 2) は，4.3 節で述べたディスクドライバの制御の実装前と実装後において，永続ユニットにおける読み込みと書き出しにかかる処理時間を測定した．永続ユニットの大きさを 4KB とした．

測定結果を以下に示す．

＜測定 1＞

ドライバの状態変更処理時間を 100 回測定した平均値は，0.204 $\mu$ sec であり，非常に短い．したがって，ドライバの状態変更処理による実入出力に対する影響は，極めて小さいと考えられる．

＜測定 2＞

永続ユニットの読み出しと書き出しにかかる処理時間を表 2 に示す．表 2 に示される通り，実装前と実装後の処理時間の差は最大でも 0.07% と非常に小さい．したがって，制御を行うことによる実入出力のオーバヘッドは極めて小さいと考えられる．

**表 2** 永続ユニットの読み出しと書き出しにかかる処理時間

| 処理 | 実装前の処理時間 ($\mu$ sec) | 実装後の処理時間 ($\mu$ sec) | 実装後の処理時間/実装前の処理時間 |
|---|---|---|---|
| read | 2773.17 | 2775.03 | 1.0007 |
| write | 9458.98 | 9459.02 | 1.0000 |

## 6. おわりに

***Tender*** の動作継続制御機能において，計算機再起動後，デバイスへの入出力処理の継続を可能にするデバイス制御法について述べた．具体的には，全てのドライバの状態を管理し，プレート書き出しの際，全てのドライバが非実入出力状態に移行してから，プレートの書き出しを行う．これにより，ドライバが非実入出力状態で保存され，プレート復元後に入出力処理を継続できる．また，評価では，ディスクドライブの制御におけるオーバヘッドが極めて小さいことを示した．

残された課題として，ディスクドライバ以外におけるドライバ制御の実装，評価がある．

**謝辞** 本研究の一部は，科学研究費補助金若手研究 (B)(課題番号 18700030) による．

## 参 考 文 献

1) TuxOnIce, “http://www.tuxonice.net”.

2) Norman Hardy. “The KeyKOS Architecture,” Operating Systems Review, vol.19, No.4, pp.8-25, October 1985.

3) Espen Skoglund, Christian Ceelen, and Jochen Liedtke, “Transparent orthogonal checkpointing through user-level pagers,” In Proceedings of the 9th International Workshop on Persistent Object Systems, pp.201–214, Sep 2000.

4) Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, and Francis Vaughan, “Grasshopper: An Orthogonally Persistent Operating System,” Computing Systems, Vol.7, No.3, pp. 289–312 (1994).

5) 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.

6) 的野　司, 田端利宏, 谷口秀夫, “***Tender*** におけるプレート機能を利用した資源の永続化機構の設計,” 情報処理学会研究報告, Vol.2003, No.42, pp.147–154, 2003.

7) 大本拓実, 田端利宏, 谷口秀夫, “仮想記憶空間上のデータ不揮発性化による計算機処理永続化方式の提案,” 情報処理学会研究報告, Vol.2007, No.10, pp.25–32, 2007.