システムソフトウェアと
オペレーティング・システム 75－6
（１９９７．６．５）

Tender における永続制御機能

市川 正也　谷口 秀夫　牛島 和夫
九州大学大学院システム情報科学研究科
Email: ichikawa,tani,ushijima@csce.kyushu-u.ac.jp

継続的な処理を行なう応用プログラムでは、計算機の緊急停止や定期的な停止に備える必要があり、プログラム記述の負担となる。この負担を軽減するために、プロセスの走行環境を保存し永続化する機能をオペレーティングシステムに備える。具体的には、***Tender*** の永続制御機能について述べる。***Tender*** では、プロセスが扱うメッセージや時計だけではなくプロセスのメモリ空間などの全てを資源として管理しており、永続制御機能はこの資源を保存することで、永続化を実現する。永続制御機能を実現する際の課題と対処を述べ、簡単な評価を行なった結果を示す。

キーワード
オペレーティングシステム、永続、資源

Persistent Control on *Tender*

Masaya ICHIKAWA, Hideo TANIGUCHI and Kazuo USHIJIMA
Graduate School of Information Science and Electrical Engineering,
Kyushu University
E-Mail: ichikawa,tani,ushijima@csce.kyushu-u.ac.jp

Application programs need transaction management to cope with urgent and regular suspensions of operation. To meet the problem, an operating system shuld have some functions that preserve and maintain the process execution environment permanently. To put it concretely, we investigate Persistent Control on our operating system, ***Tender***. On ***Tender***, all items such as messages and clocks and memory space are managed as resources. They are permanently maintained by the Persistent Control function. First, we discuss the problems and solutions of implemenation of the Persistente Control function. Then we evaluate a brief evaluation of the results of the Persistent Control function obtained on our system.

key word
operating system, persistent, resource

1　はじめに

近年、計算機の低価格化と共に、プロセッサ性能は向上し、メモリ量も増加している。これらを背景として、膨大な計算量やメモリ量を必要とする処理や、複雑な処理を行う応用プログラム (以降 AP と略す) が登場している。これらの AP では、処理の連続走行時間が長く、また AP 起動時の処理時間も長くなっている。一方、計算機は、不具合による緊急停止を完全に防ぐことはできない。また、運用からくる定期的な停止もある。

計算機を停止させると、プロセスが利用するメモリ空間上の情報は消滅する。また、オペレーティングシステム (以降、OS と略す) が管理しているメモリ空間に関する情報やプロセスに関する情報も消滅してしまう。このため、計算機の停止は走行プロセスの消滅を意味する。そこで、計算機の停止に対処するため、連続走行時間が長い AP は、処理途中の情報を退避し、再開時は退避した情報に基づいて処理を継続実行できるようにしなければならない。この対処は、AP に対し、プログラム記述の負担を大きくし、

かつ通常の処理効率を低下させてしまう。このため、銀行などのオンライントランザクション処理を行うAPは、その処理の重要性から、トランザクションを管理するミドルソフトウェアを利用して、これに対処している。しかし、多くのAPは、この対処を行わない。連続走行時間が長いと事前に予想され、かつ処理の途中で計算機の停止が予見できるAPでは、停止の通知を受けて処理を中断できるように対処し、定期的な停止に備えることもある。しかし、この対処では、停止の通知がない緊急停止には有効でない。したがって、計算機の停止により、APは処理を停止させられ、それまでの処理を全て無駄にしてしまうことも少なくない。また、再開時の処理は、最初の起動時以上に複雑になるため、再開時間が長くなってしまう。

AP処理の中断と再開を可能にする機能として、ハードウェア対処によるレジューム機能がある。この機能は、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。

本稿では、これらの問題への対処法として、OSの永続制御機能について述べる。具体的には、我々の研究室で開発している***Tender***[1]の永続制御機能について報告する。***Tender***は、プロセスが扱う時計やメッセージなどに加え、プロセスを構成する要素(プロセスが利用するメモリ空間やコンテキストなど)を「資源」として管理している[2]。永続制御機能は、この資源を外部記憶装置へ保存する機能である。

2 *Tender*

我々は、プログラム構造に重点を置いて設計した分散指向永続オペレーティングシステム***Tender***(*The ENduring operating system for Distributed Environment*)を開発している。

Tenderの特徴的な主な項目として、以下の9つがある。

(1) 可変プロセッサ性能を提供するスケジュール
(2) 資源管理の統一[3]
(3) 資源の永続化
(4) 資源生成の高速化
(5) 遠隔資源の操作
(6) プログラム部品の入替え[4]
(7) 動作内容に合わせたプロセスの走行制御
(8) プロセスの最適分散[5]
(9) OSやプロセスの動作状況の可視化[6]

永続制御機能は、上記(3)に関連する。

Tenderでは、資源を「一つの操作を行なう場合、その操作で処理が完結し、他の処理を必要としないもの」と定義している。全ての資源は固有の資源識別子と資源名を持ち、資源名は資源名管理木によって管理される。個々の資源の操作は、資源識別子や資源名を用いて行なわれる。

Tenderにおける仮想メモリ空間[7]のモデルを図1に示す。仮想メモリ空間は、仮想領域、仮想空間という二つの概念により構成されている。仮想領域は、メモリイメージを仮想化した領域で、その実体は実メモリ、または、外部記憶装置上に存在する。仮想空間は、特定のアドレス範囲を持つ仮想的な空間である。具体的には、仮想アドレスを実アドレスに対応付けるための管理表のみからなる。仮想空間の任意の仮想アドレスに仮想領域を「貼り付ける」ことにより、仮想メモリ空間が作成される。ここで「貼り付ける」とは、仮想アドレスを実アドレスに対応付ける対応表に、対応付けのための情報を格納することである。仮想領域と仮想空間はそれぞれプロセッサが単独ではアクセスすることはできない。仮想領域を仮想空間上の任意アドレスに「貼り付ける」ことにより、仮想ユーザ空間、または、仮想カーネル空間が作成され、その上のデータへのアクセスが可能となる。

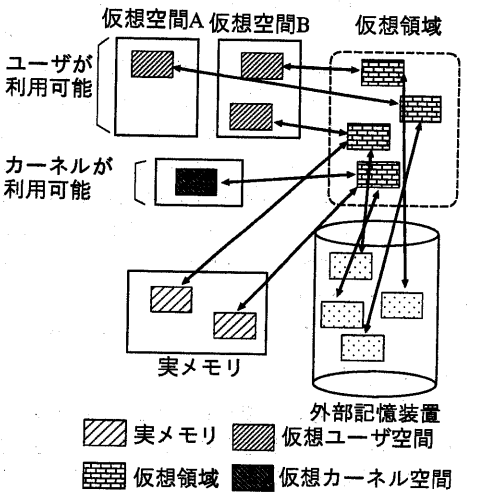

図1 ***Tender***における仮想メモリ空間のモデル

3 課題と対処

永続制御機能の実現においては、永続化する情報を効率的に処理することが必要である。この処理の

大半は入出力であり、入出力処理を効率化することで、永続制御の負荷を小さくすることができる。このための課題として、以下のものがある。

(1) 永続化の対象
(2) 永続化と仮想化の共存
(3) 入出力の契機
(4) 外部記憶装置上での記録形式

各課題について、以降に説明し、対処法を示す。

3.1 永続化の対象

永続化を行う対象は、メモリ上にあり、プロセッサが扱える資源である。この資源は、その扱いから大きく二つに分類できる。一つは、メモリを管理する観点で操作する仮想ユーザ空間と仮想カーネル空間である。これらを利用形態を意識しない資源と呼ぶ。もう一つは、時計や周期タイマやデータなど、APプロセスが操作する資源である。これらを利用形態を意識した資源と呼ぶ。利用形態を意識した資源と利用形態を意識しない資源の関係を図2に示す。利用形態を意識した資源は、利用形態を意識しない資源の上に確保されている。たとえば、時計は、仮想カーネル空間の一部分の領域に、時計を管理する管理表を確保し制御することにより、時計資源を実現する。

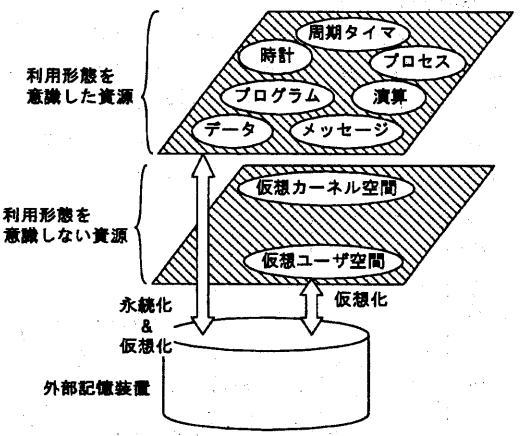

図2 利用形態を意識した資源と利用形態を意識しない資源の関係

したがって、永続化の対象とする資源として、大きく二つある。

＜方式1＞利用形態を意識しない資源を永続化の対象とする

＜方式2＞利用形態を意識した資源を永続化の対象とする

＜方式1＞は、永続化のための入出力処理の効率が悪い。これは、入出力の契機が、メモリを管理する観点から発生するからである。一方、永続化では、各資源ごとに「同期」が必要になる。このため、この方式では、例えば、永続化に際する情報の保存においては、すべての資源を保存することになるかもしれない。これは、膨大な量になる。一方、＜方式2＞は、永続化のための入出力処理の効率を向上できる。これは、利用形態を意識した入出力処理ができるからである。例えば、永続化の際の情報の保存においては、資源の利用停止を契機として、当該資源のみを保存すればよい。したがって、＜方式2＞を採用した。

ここで、***Tender***には、資源「データ」がある。資源「データ」は、既存OSのファイルに相当する。ファイルは外部記憶装置上にあるが、資源「データ」はメモリ上にある。このため、資源「データ」の永続化により、その内容を保証する。このようになっているため、資源「データ」上に時計や周期タイマなどの資源を構築することもできる。この様子を図3に示す。利用形態を意識しない資源である、仮想ユーザ空間や仮想カーネル空間を利用して、資源「データ」を実現する。さらに、この資源「データ」を利用して、時計や周期タイマなどの資源を実現する。このようにすることで、資源「データ」の永続化により、すべての資源を永続化することが可能である。もちろん、仮想ユーザ空間や仮想カーネル空間の管理表なども資源「データ」の上に実現するのである。

3.2 仮想化との共存

メモリ上の情報を外部記憶装置上に格納する処理には、永続化のほかに、仮想化がある。仮想化は、仮想メモリ空間を実現するためのものであり、***Tender***では要求時ページング機能を実現する。

永続化非対象資源のための仮想化処理は、仮想化領域を利用する。これに伴い、永続化対象資源のための仮想化処理も仮想化領域を利用することが考えられる。しかし、同じメモリ領域の内容が外部記憶装置上に2ヵ所存在してしまうことになり、領域の無駄と共に出力が無駄である。このため、永続化非対象資源のための仮想化処理は、永続化領域を利用する工夫を行う。つまり、永続化非対象資源については、外部記憶装置上の一つの領域を用いて、永続化と仮想化の処理を共存させる。

この際、仮想化のための出力であっても、その出力データには、次の対処が必要である。様子を図4

に示し、説明する。仮想化のためのページアウトの場合、永続化非対象資源では、当該ページを書き出す。これに対し、永続化対象資源では、当該ページを含む一つの資源に関する情報について、関連するページ全部を書き出す必要がある。さもないと、一つの資源に関する情報に矛盾が発生し、再利用不能になる。

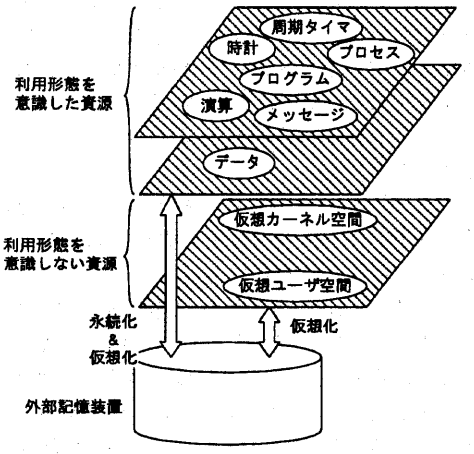

図3 資源「データ」を利用した永続化

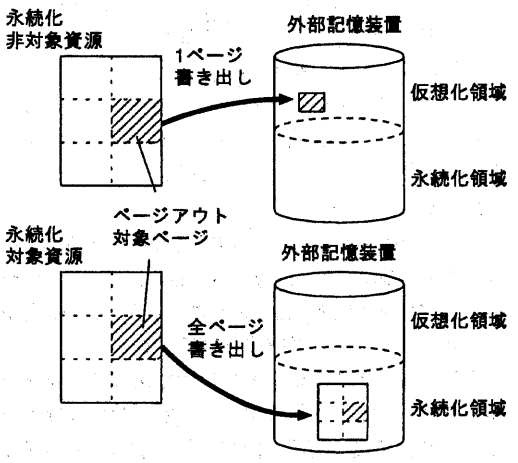

図4 永続化対象資源の書き出し

3.3 入出力契機

永続化を行なう際、外部記憶装置への入出力をいつ行なうかが重要である。外部記憶装置への書き出し処理を頻繁に行なえば、計算機が停止する直前のプロセスの状態を保存できる。しかし、入出力処理が頻繁に発生し、他のサービス処理の低下を招いてしまう。このため、適当な時間間隔で入出力を行うことが必要である。

仮想化の契機としてページ例外があり、これを、永続化の契機としてもとらえることができる。ただし、この契機だけでは不十分であるため、外部からの強制永続化の契機を提供する。この契機の利用として、定期的な永続化を各資源の重要度に合わせて行うことが考えられる。

3.4 記録形式

永続化した資源の復元を行なう際、資源の外部記憶装置上での格納場所に関する情報が必要となる。このため、永続化したい資源の資源名と外部記憶装置上での格納場所とを対応付け、この対応付けに関する情報を外部記憶装置上に記録する。これにより、資源名から資源の格納場所を知ることができる。

したがって、外部記憶装置上での記録形式として「ファイルシステム」の形式が必要になる。永続化対象資源の資源名をファイル名、資源そのものをファイル実体部とすることで資源単位での管理が可能になる。

一方、***Tender***が扱う情報を、既存の他のOSでも扱えることが好ましい。このため、この「ファイルシステム」の形式を既存のファイルシステムと互換とする。具体的には、UNIX ファイルシステムと互換の形式にする。

UNIX ファイルシステムとの入出力の様子を図5に示す。***Tender***の各資源に割り当てられた資源識別子と資源名を管理する資源名管理木を UNIX ファイルシステムのディレクトリに対応付け、資源のメモリイメージを通常ファイルに対応付ける。

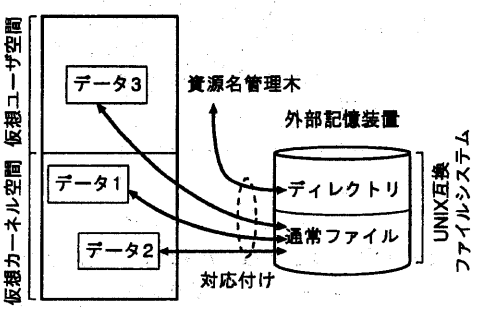

図5 資源の外部記憶装置上における記録形式

4 実装と評価

以降で、永続制御機能の実現内容について述べる。外部記憶装置として、磁気ディスクを使用した。さらに、簡単な評価を行なう。

4.1 実現内容

永続制御では、永続化と仮想化を行う対象の単位を「永続ユニット」と名付け、識別子を付与する。なお、永続ユニットの識別子は各々の資源の資源識別子に相当する。

永続制御機能は、大きく次の2つがある。

(1) 立ち上げ時の初期化に関する機能
(2) 動作中の機能

立ち上げ時の初期化に関する機能の大半は、永続化された資源の復元処理である。永続化された資源の復元処理とは、磁気ディスクに保存した資源をメモリに読み込み、APの再処理開始を可能にする処理である。具体的には、ディスクに格納された初期化テーブルの内容に従って、UNIXファイルの内容を実メモリ領域に読み込む機能である。

動作中の機能には、基本機能と拡張機能があり、基本機能には、

(1) ディスク領域の予約と解放
(2) ディスク領域の空き領域管理
(3) 永続ユニットの生成と削除
(4) 永続ユニットのイン・アウト処理
(5) 永続ユニットの大きさ取得

の機能がある。ディスク領域の予約と解放の機能は、仮想領域に対応する大きさの領域を磁気ディスク上に予約または解放する機能である。ディスク領域の空き領域管理の機能は、磁気ディスク領域の空き領域を管理する機能である。これにより、永続ユニットの仮想化の時に磁気ディスク領域が不足することを防ぐ。永続ユニットの生成と削除の機能は、仮想領域に対応する永続ユニットを磁気ディスク上に生成または削除する機能である。永続ユニットのイン・アウト処理の機能は、永続化対象資源が使用する仮想領域と、対応する磁気ディスク領域との間で、データの授受を行う機能である。永続ユニットのイン処理は1ページ単位で処理を行ない、アウト処理は、書き出し対象ページを使用する資源に対し、資源全体の書き出しを行なう。永続ユニットの大きさ取得の機能は、当該の永続ユニットの大きさを取得する機能である。

拡張機能には、

(6) UNIXファイルシステム変換

の機能がある。UNIXファイルシステム変換の機能は、資源名管理木と資源「データ」の磁気ディスク上での記録形式を、UNIXファイルシステム互換の形式に変換する機能である。これにより、UNIXシステムとのデータ授受が可能になる。UNIXファイルシステムとして、BSD/OS Ver 2.0のFFS(Fast File System)の形式を用いた。

4.2 評価

永続制御の処理の中心となる永続ユニットのイン処理と永続ユニットのアウト処理の処理時間を実測した。

4.2.1 測定環境

実測は、AT互換機(Pemtium 100MHz)を用い、入出力は磁気ディスク装置に対して行なった。

処理時間の測定は、同じ処理を100回繰り返し測定し、1回当たりの処理時間を求めた。時間の計測は、時間計測機能(単位:ミリ秒)を利用した。

4.2.2 結果と考察

1ページの大きさを4KBとして、永続ユニットのイン処理における、1ページ読み込みにかかる時間を測定したところ、26.62ミリ秒であった。これは実I/O時間を含んだ時間である。そのため、実I/O時間を計測し、永続制御の処理部分を算出したところ、10マイクロ秒以下であった。従って、永続制御の永続ユニットのイン処理時間は非常に小さいと言える。

永続ユニットのアウト処理時間と、実I/O時間を計測し、永続制御の処理部分を算出した。その結果を図6に示す。図6から、実I/Oを含まない永続制御のアウト処理時間については、以下のことがわかる。

(1) 数マイクロ秒であり、非常に小さい。
(2) アウト処理時間が書き出すページ数に比例して増加する理由は、ページ数に比例した実I/Oを行なう処理があるからである。

5 おわりに

Tenderの永続制御機能について述べた。具体的には、永続化の目的を述べ、さらに、永続化の対象となる資源、仮想化処理との共存、永続化の入出力の契機、外部記憶装置上での記録形式について述べた。永続化の対象となる資源としては、利用形態を意識した資源とすることが処理効率上好ましい。仮想化処理との共存は、外部記憶装置の領域を分離することにより可能となる。永続化の入出力の契機として、ページ例外だけではなく、ユーザ指定による契機も入出力契機とした。外部記憶装置上での記録形式は、既存OSのファイルとの互換を考え、UNIXファイルシステム形式とした。また、永続制御機能

を***Tender***上に実現し、評価した。その結果、永続制御処理のオーバヘッドは非常に小さいことがわかった。

今後は、応用プログラム処理の動作も含めた評価を行なう予定である。

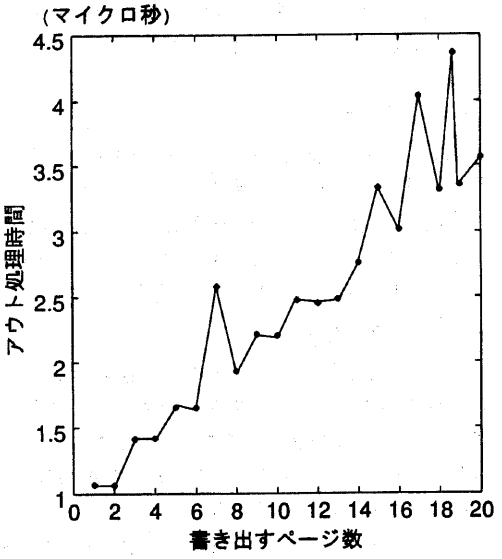

図6 アウト処理時間 (実I/O 部分を除く)

参考文献

[1] 谷口秀夫: "分散指向オペレーティングシステム***Tender***", 情報処理学会シンポジウム論文集 Vol.95, No.7, pp.47-54(1995).

[2] 谷口秀夫, 田中徳穂: "***Tender***のプロセス管理構造", システムソフトウェアとオペレーティング・システム, 73-19 (1996).

[3] 後藤真孝, 谷口秀夫, 牛島和夫: "***Tender***における資源管理方式", 情報処理学会第51回全国大会予稿, 5L-7 (1995)

[4] 後藤真孝, 谷口秀夫, 牛島和夫: "走行中のプロセス間で共有されたプログラムの部分入れ替え法", 情報処理学会第50回全国大会予稿, 7H-2 (1995).

[5] 青木義則, 谷口秀夫, 牛島和夫: "応答時間に着目した静的な処理分散法の提案と実現" 情報処理学会シンポジウム論文集 (1995)

[6] 野口裕介, 谷口秀夫, 牛島和夫: 'OS動作の可視化機能の設計", コンピュータシステム・シンポジウム, Vol. 96, No.7, pp. 139-146 (1996).

[7] 長嶋直希, 谷口秀夫, 牛島和夫: "***Tender***のメモリ管理機能", 情報処理学会第53回全国大会, (1-5)-(1-6)(1996).