

*Tender*におけるIPCの継続を可能にする プロセス実行途中状態の保存復元機能の設計

難波 弘樹[†] 田端 利宏[†] 谷口 秀夫[†]

[†] 岡山大学大学院自然科学研究科

運用や故障のためにプロセスの走行中で計算機が停止すると、計算機再起動後に当該プロセスを、はじめから走行させなくてはならない。このため、実行途中までの処理は無駄になってしまう。この対策に、プロセス実行途中状態の保存と復元の機能があり、いくつか研究が行われている。ここでは、計算機の停止と再開の前後で、プロセス間通信を継続して行うことのできる機能について述べる。具体的には、*Tender* (The ENduring operating system for Distributed EnviRonment) オペレーティングシステムにおける実装と評価結果を報告する。

Design of a Function to Store and Restore Process State including InterProcess Communication on *Tender*

Hiroki Nanba[†] Toshihiro Tabata[†] Hideo Taniguchi[†]

[†] Graduate School of Natural Science and Technology, Okayama University

System failure makes process restart from initial state. Function to store and restore process state are effective mechanism that prevents a process from restarting from the initial state when a system failure occurs. In this paper, we describe a function to store and restore processes state including InterProcess Communication(IPC). In addition, we report the implementation and evaluation results on *Tender* operating system.

1. はじめに

突然の故障により計算機が停止すると、走行していたプロセスの実行途中状態は消滅してしまう。また、定期的に保守を行うシステムでは、保守を行う際にプロセスを強制終了させるため、保守間隔を超えて長時間走行する応用プログラム（以下、AP と呼ぶ）のプロセスは、何らかの対処が必要である。これらの対策として、プロセス実行途中状態の保存と復元の機能がある。

システムの故障に伴う計算機停止後の再起動処理において、保存したプロセス実行途中状態の情報に基づき、プロセスの実行状態を復元することで、プロセス処理を迅速に回復できる。また、処理時間の長いプロセスについても計算機の停止と再開の前後で同様の処理を行うことにより、継続的に処理を行うことができる。

プロセス実行途中状態の保存復元機能の関連技術として、プロセスマイグレーション機能がある。プロセスマイグレーション機能とは、プロセス実行途中状態

の情報をを用いて別計算機上にプロセス実行状態を復元する機能である。プロセスマイグレーション機能を用い、AP を負荷の小さい計算機へ移動することで、動的に負荷分散することができる。また、メンテナンス前に AP を別計算機に移動し処理を継続することで、システム可用性を向上することができる。

プロセス実行途中状態の保存復元機能について様々な研究が行われている。しかし、計算機の停止と再開の前後でプロセス間通信を継続して行うことを目的とした研究は行われていない¹⁾²⁾³⁾⁴⁾⁵⁾。このため、他プロセスの実行途中状態を保存復元する場合問題になる。

本論分では、計算機の停止と再開の前後で、プロセス間通信を含むプロセス処理を継続できる機能について述べる。具体的には、この機能を実現する際の課題と対処について述べ、*Tender* (The ENduring operating system for Distributed EnviRonment) オペレーティングシステム⁶⁾における実装内容について述べる。さらに、本機能の性能評価について述べる。

表 1 関連研究での保存復元対象

研究名	保存復元契機	保存対象プロセス			差分	共有メモリ	シグナル	ファイルディスクリプタ (通常ファイル)	通信状態	
		自	他	複数					パイプ	ソケット
文献 1)	システムコール	○	○	×	○	×	×	×	×	×
文献 2)	—	—	—	×	○	×	×	×	×	×
文献 3)	システムコール	○	○	○	×	○	×	×	×	×
文献 4)	自動	—	—	×	○	×	×	×	×	×
文献 5)	割り込み	○	×	×	○	×	○	○	×	×

自：保存要求を出したプロセス自身，他：他プロセス，複：複数プロセス，差分：差分の保存，—：不明

○：保存復元可能，×：保存復元不可能

2. 関連研究

プロセス実行途中状態の保存復元機能に関する関連研究を表 1 に示す。この表は、各研究における保存復元対象について示している。

プロセス実行途中状態の保存復元機能は、大きなオーバーヘッドになる。この問題に対して、保存復元処理のオーバーヘッド最小化に関する研究、および最適化に関する研究が行われている。

オーバーヘッド最小化に関する研究では、保存データ量を減らす方法や、遅延を隠す方法について研究が行われている。保存データ量を減らす方法には、前回保存したときからの差分のみを保存する技術がある¹⁾²⁾。文献 1) では、ページ例外を利用して差分を特定している。文献 2) では、ハッシュ関数を利用して差分を特定している。一方、遅延を隠す方法には、書き出し処理の遅延³⁾や、書き込みオーバーヘッドを減らすためのデータ圧縮といった技術がある。文献 3) では、保存が完了し、プロセス処理が再開するまでディスク I/O への書き出しを遅延することでプロセス WAIT 時の膨大な I/O 時間を避けている。

最適化に関する研究⁴⁾とは、保存復元処理を効率よく行うための研究である。文献 4) では、予想されるプロセス復元時間のコスト分析により、動的に保存間隔を変更する間隔決定メカニズムについて研究している。

プロセスの実行途中状態は、プロセスのデータ構造、割り込み、シグナル、オープンファイル、通信チャンネル、共有メモリ、および協調処理を行っている他プロセスの状態と非常に広範囲である。このため、これら全ての状態を完全に保存復元することは困難である。この問題に対して、保存復元対象を特定の状態のプロセスに絞り研究が行われている³⁾⁵⁾。文献 3) では、プロセス間の関係に焦点をおき、親子関係、兄弟関係、グループ、およびセッションの保存復元を実現している。文献 5) では、シグナル、シグナルハンドラ、およびファイルディスクリプタ (通常ファイル) の保存復元を実現している。

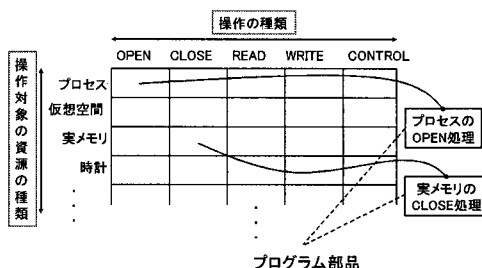


図 1 プログラムポインタ表

表 1 より、文献 1)～文献 5) では、通信状態の保存と復元が実現されていないことがわかる。

3. Tender オペレーティングシステム

3.1 資源処理呼び出し方法

Tender のプログラムは、基盤部、表プログラム構造部、入出力制御部、および拡張部の 4 つの部分で構成される。また、資源を管理するプログラムは、表プログラム構造部で管理される。

表プログラム構造部では、独立した資源処理をプログラム部品と呼ぶ単位に分割し、独立化する。これらプログラム部品の呼出は、必ず資源インタフェース制御 (RIC: Resource Interface Controller) と呼ぶモジュールを経由して行う。RIC は“操作対象の資源の種類”と“操作の種類”から構成されるプログラムポインタ表によってプログラム部品を管理している。このプログラムポインタ表を図 1 に示す。“操作対象の資源の種類”はプロセスや実メモリといった資源の種類であり、“操作の種類”は OPEN, CLOSE, READ, WRITE, CONTROL のいずれかである。ここでは、このプログラム部品により行われる処理を資源処理と呼ぶ。全ての資源処理の呼出は、RIC を経由して行われる。

3.2 資源の分離と独立化

Tender では、OS が操作する対象を資源として分離し、独立化している。資源には、資源識別子と資源

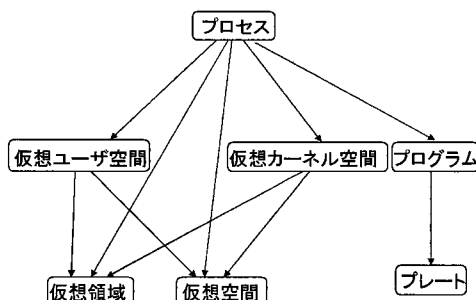


図2 プロセス構成資源

名を付与し、資源操作のインタフェースを統一している。各資源を操作するプログラム部品を資源ごとに分離し、共有プログラムを排除している。また、各資源の管理情報も資源ごとに分離し、各資源の管理表の間のポインタを禁止している。このように資源の分離と独立化を行うことで、プロセス構成資源を個別に変更することが容易になり、プロセスの実行環境を変更できる。プロセス構成資源は、各々独立しているため、プロセス構成資源を変更する際には、変更対象となる資源のみを扱うことができるので、変更処理を効率化できる。また、資源の分離と独立化を行うことで、資源の事前生成や保留が可能になり、資源の生成や削除を伴う処理を高速化することができる。

3.3 プロセスの構成資源

プロセス構成資源を図2に示す。ここで、矢印は、資源の依存関係を表している。資源「プロセス」とは、プロセス識別子とプロセス管理表からなり、OSがプログラムの動作を制御する単位になる。したがって、プロセス生成時には、資源「プロセス」が生成先計算機上に生成され、動作を制御する単位になる。資源「プログラム」とは、実行プログラムのテキスト部とデータ部の大きさと先頭アドレス、およびプログラムの開始アドレスの情報からなり、プログラムの実行形式を隠蔽している。プログラムの内容は、資源「プレート」上に存在している。ここで、資源「プレート」とは、永続的な記憶を提供するものであり、既存のOSのファイルに相当する。

ここで、プロセス構成資源間の関係を図3に示す。資源「仮想空間」とは、仮想アドレスの空間であり、仮想アドレスを実アドレスに変換する変換表に相当する。資源「仮想領域」とは、メモリーイメージを仮想化した領域であり、仮想領域の実体は、実メモリーまたは外部記憶装置上に存在する。資源「仮想カーネル空間」、ならびに資源「仮想ユーザ空間」とは、プロセス

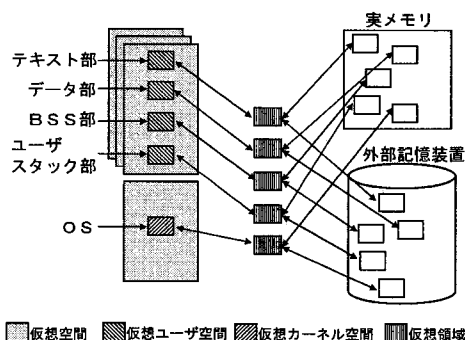


図3 プロセス構成資源の関係

サが仮想アドレスによって、前者はカーネルモードのみ、後者はユーザモードでもアクセス可能な空間である。両者は共に、仮想空間が持つアドレス変換表に、当該の仮想領域のデータ格納情報を設定する（以後、貼り付けと呼ぶ）ことで生成することができる。逆に、貼り付けた際に設定したデータ格納情報を解放する（以後、剥すと呼ぶ）ことにより、削除することができる。プロセスのテキスト部、初期値を持つ変数や文字列の集合部分であるデータ部、初期値を持たない変数や文字列の集合部分であるBSS部、およびユーザスタック部は、仮想ユーザ空間に読み込まれた状態で仮想空間上に存在する。

3.4 プロセス間通信機能

3.4.1 基本機能

*Tender*では、プロセス間通信機能として、メモリー空間上のデータをプロセス間で移動、複製、共有する機能、およびイベント発生時の処理やカウンタセマフォの機能を提供している⁷⁾。*Tender*では、資源「コンテナ」をやり取りすることでプロセス間通信を行う。*Tender*におけるプロセス間通信には、資源「コンテナボックス」を介した送受信方式、および即時同期を可能にするプロセス間通信におけるコンテナの奪い取り、押し付けの機能がある。

3.4.2 資源「コンテナ」

資源「コンテナ」は、プロセスが利用可能な大きさを持つメモリー空間で、仮想空間に存在する仮想ユーザ空間を用いて生成され、ユーザモードまたはカーネルモードでアクセス可能である。プロセスは、この資源「コンテナ」をやり取りすることでプロセス間通信を行う。プロセス間での資源「コンテナ」を用いた通信方法として、移動、共有、および複製の3つがある。移動は、コンテナを別のプロセスの空間に移動することである。共有は、プロセス間でコンテナを共有する

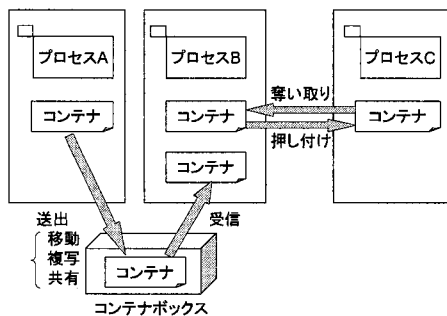


図 4 Tender におけるプロセス間通信

ことである。複写は、対象コンテナの内容を新しいコンテナに複写して、新しいコンテナを別のプロセスの仮想空間に貼り付けることである。

また、即時同期を可能にするプロセス間通信機能として、奪い取りの機能と押し付けの機能がある。

3.4.3 資源「コンテナボックス」

資源「コンテナボックス」は、プロセス間での資源「コンテナ」の受け渡しを仲介する。つまり、コンテナボックスを介したコンテナのやり取りによるプロセス間通信は、送受信方式のプロセス間通信と同等の機能である。具体的には、以下の処理を行う。コンテナボックス管理処理部は、コンテナの送出要求を受けると、送出時に指定されたコンテナボックスのキューの最後に、送出要求のあったコンテナを格納する。コンテナの受信要求があった場合には、コンテナの送出時の貼り付けアドレス要求と受信側の貼り付けアドレス要求を比較し、一致する場合にコンテナを受信プロセスに渡す。

プロセス間通信の様子を図 4 に示す。プロセス A とプロセス B は、コンテナボックスを介したプロセス間通信を行っている。プロセス A がコンテナボックスにコンテナを送出し、コンテナ B がコンテナボックスからコンテナを受信している。また、プロセス C の仮想空間のコンテナからプロセス B の仮想空間のコンテナの矢印は、コンテナの奪い取りを表している。プロセス B の仮想空間のコンテナからプロセス C の仮想空間のコンテナの矢印は、コンテナの押し付けを表している。

4. 実現における課題と対処

4.1 基本方針

計算機の停止と再開の前後で、プロセス間通信を含むプロセス処理を継続できるプロセス実行途中状態の保存復元機能の基本方針について示す。図 5 に保存機

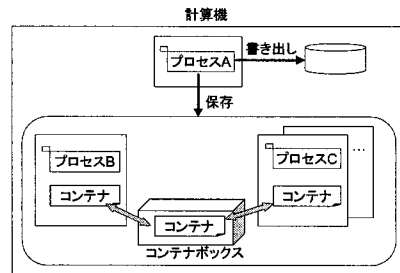


図 5 保存の様子

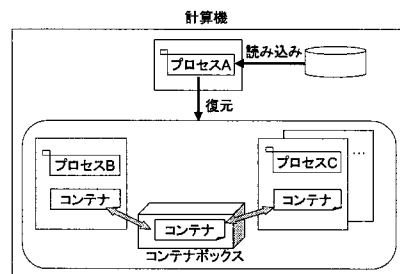


図 6 復元の様子

能の全体像を示す。図 6 に復元機能の全体像を示す。保存機能では、プロセス間通信の状態を含む、複数プロセスの実行途中状態をディスクに書き込み保存する。復元機能は、保存したプロセス実行途中状態をディスクから読み込み、この情報を基に、複数プロセスの実行状態をプロセス間通信の状態を含めて復元する。

4.2 課題

計算機の停止と再開の前後で、プロセス間通信を含むプロセス処理を継続できるプロセス実行途中状態の保存復元機能の実現においては以下の課題がある。

- (課題 1) 保存可能、不可能状態の検討
- (課題 2) 保存対象プロセス利用資源の特定
- (課題 3) 保存対象にするコンテナの検討
- (課題 4) 資源識別子衝突問題の対処

(課題 1) デバイスの I/O 待ちのプロセスを保存復元する場合、メモリの内容に加え、I/O の状態も保存復元する必要がある。しかし、デバイスの I/O 状態の保存復元は簡単ではない。このため、デバイスの I/O 待ちのプロセスは保存不可能な状態とする。また、保存対象プロセス全てが保存可能な状態にあるとは限らない。したがって、プロセスの状態ごとに保存可能であるか、保存不可能であるか確認する必要がある。

表 2 プロセスの各状態における保存可否

カーネルモード		ユーザモード	
READY	—	READY	○
WAIT	×	WAIT	—
ユーザモードに戻る直前	○		
保存要求プロセス自身	○		

○：保存できる，×：保存できない，—：存在しない。

（課題 2）プロセスの実行途中状態には，プロセスの利用している資源の状態も含まれる。このため，プロセスの利用している資源情報を管理する必要がある。

（課題 3）通信を行う過程で，保存対象プロセスは，保存対象外プロセスの所有するコンテナを利用することも考えられる。保存対象外プロセスの所有するコンテナを保存対象にする場合，復元時このコンテナを所有するプロセスは復元されないため，問題になる。このため，保存対象にするコンテナについて検討する必要がある。

（課題 4）プロセスは，資源に割り当てられている資源識別子を用いて資源を識別する。このため，復元時，復元対象資源には保存時と同じ資源識別子を割り当てる必要がある。しかし，該当する資源識別子がすでに別の資源に割り当てられている場合，復元対象資源にこの資源識別子を割り当てることができず，復元処理は失敗してしまう。（以降，この問題を資源識別子衝突問題と呼ぶ。）このため，資源識別子衝突問題を対処する必要がある。

4.3 対 処

4.3.1 保存可能，不可能状態の検討

プロセスの各状態における保存可否について表 2 に示す。ここでは，プロセスの状態を走行モード，および状態 (RUN, READY, WAIT) で分類した。また，カーネルモードの場合，ユーザモードに戻る直前，および保存要求プロセス自身について保存可能であるか否か検討した。

Tender では，カーネル内ブリエンプションを禁止している。このため，カーネルモードで READY 状態にあるプロセスは存在しない。

周辺機器へ I/O 要求を行ったプロセスは，周辺機器の処理が終了するまでカーネルモード WAIT 状態である。このため，メモリ内のプロセスの状態を保存するだけでは正確に復元できない。したがって，カーネルモードで WAIT 状態のプロセスは，保存不可能状態にあるとする。

ユーザモードに戻る直前のプロセスの状態は，仮想ユーザ空間内に閉じていると考えられるため，保存可能である。また，保存要求プロセス自身は保存可能である。

ユーザモードで READY 状態にあるプロセスは，周辺機器へのアクセスを行っていないため，保存可能である。

4.3.2 保存対象プロセス利用資源の特定

プロセスの利用資源を特定するために，プロセスの利用資源情報を管理する利用資源表を導入した。利用資源表を用いることで，保存対象プロセスの利用資源を特定できる。利用資源表で管理する資源として以下の 2 つが考えられる。

- (1) プロセスの所有する資源
- (2) プロセスの利用する資源

各資源を所有するプロセスは 1 つであり，基本的にはこの資源を生成したプロセスである。例外については以下で述べる。一方，資源を利用するプロセスは，複数存在できる。プロセスの利用する資源を利用資源表の管理対象にすると，複数プロセスで共有している資源は複数回保存され，復元時問題になる。そこで，利用資源表の管理対象は，プロセスの所有する資源にした。

利用資源表へ資源を登録する位置について説明する。プロセスがカーネルコールを呼び出し，カーネル処理に移ると，はじめにカーネルコール処理関数が呼び出される。その後，RIC を経由して該当する資源の資源処理に移る。RIC は，プログラムポインタ表を保持しているため，資源に対する操作の種類を特定できる。そこで，この RIC において，OPEN 処理により生成された資源を利用資源表に登録し，CLOSE 処理で削除された資源を利用資源表から削除する。

また，資源の状況に従い，管理情報を更新する。プロセス A の所有するコンテナを移動モードでプロセス B が受信を行う。受信後，このコンテナを利用するプロセスはプロセス B のみである。このとき，このコンテナの所有者はプロセス A からプロセス B に移動したと考えられる。このため，利用資源表のプロセス A のエントリから，プロセス B のエントリに利用資源情報を移す。このように，資源を利用するプロセスの遷移にともない，利用資源表の管理情報を更新する。

4.3.3 保存対象にするコンテナの検討

保存対象プロセスは，保存対象資源の所有するコンテナのみを利用すると制限する。このことから，利用資源表に登録されているコンテナ，および所有するコンテナボックスにつながっているコンテナを保存対象にする。コンテナボックスに保存対象外プロセスの所有するコンテナがつながっている場合は，エラーを返す。

表 3 プロセス実行途中状態の保存のインタフェース

形式	unsigned int makechp(pid, num, platename)
引数	unsigned int *pid: 保存対象プロセスを格納する領域へのポインタ unsigned int num: 保存対象プロセス数 char *platename: プロセスの状態を保存するプレート名
戻り値	成功: プレート識別子, 失敗: 負の値
機能	引数に与えられた資源名 platename をもつプレートを生成し, このプレートに pid で指定した num 個のプロセスの実行途中状態を保存する。

表 4 プロセス実行途中状態の復元のインタフェース

形式	unsigned int procrestart(platename, restore_pid)
引数	char *platename: プロセスの状態が保存されているプレートに対応する資源名 unsigned int *restore_pid: 復元したプロセスのプロセス識別子を格納する領域へのポインタ
戻り値	成功: 復元したプロセス数, 失敗: 負の値
機能	platename で指定したプレートから保存されている情報を元に, プロセス実行途中状態を復元する。

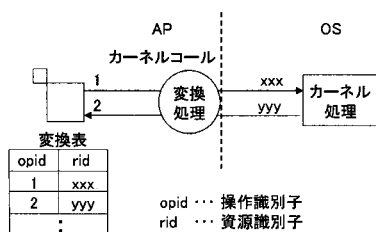


図 7 操作識別子を利用した場合の様子

4.3.4 資源識別子衝突問題の対処

資源識別子衝突問題の対処として、操作識別子を導入した。操作識別子とは、プロセス内でのみ識別可能な識別子である。操作識別子を利用する場合、プロセスは、操作識別子を用いて資源を識別する。このため、プロセスは資源識別子を意識することなく資源を扱うことができる。

操作識別子を利用した場合、カーネルコールライブラリにおいて操作識別子と資源識別子の変換を行う。操作識別子を利用した場合の処理の様子を図 7 に示す。引数に資源識別子を持つカーネルコールは、引数に操作識別子を持つように変更される。カーネルコールを呼び出すと、引数の操作識別子は資源識別子に変換され、カーネル処理に移る。また、カーネル処理が終了し、ユーザモード処理に戻る時、戻り値である資源識別子を、操作識別子に変換する。復元時、該当する資源識別子がすでに別の資源に割り当てられている場合、この資源に異なる資源識別子を割り当て、操作識別子と資源識別子の変換表を更新することで処理を継続することができる。このように、操作識別子を用いることで資源識別子衝突問題を解決できる。

4.4 保存復元インタフェース

計算機の停止と再開の前後で、プロセス間通信を含むプロセス処理を継続できるプロセス実行途中状態の

保存復元処理を実現した。保存機能について、基本インタフェースを表 3 に示す。また、復元機能について、基本インタフェースを表 4 に示す。

保存処理では、引数に与えられた資源名 platename をもつプレートを生成し、このプレートに pid で指定した num 個のプロセスの実行途中状態を保存する。具体的には、保存を要求したプロセス (以後、保存要求プロセスと呼ぶ。) は、保存対象プロセスの状態を調査し、保存可能状態にあるか確認する。全ての保存対象プロセスが保存可能状態であれば、保存処理を開始する。保存可能でないプロセスの存在する場合、保存要求プロセスは、保存可能状態にあるプロセスを WAIT し、全てのプロセスが保存可能状態になるまで自身も WAIT する。保存対象プロセスは、ユーザモードに戻る直前に WAIT し、保存可能状態になる。このとき、全ての保存対象プロセスが保存可能状態にある場合、保存要求プロセスを WAKEUP する。WAKEUP 後、保存要求プロセスは、保存に必要な大きさのプレートを生成し、このプレートに保存対象プロセスの実行途中状態を保存する。同時に、保存対象プロセスの利用しているコンテナ、コンテナボックスの状態を保存する。保存完了後、保存要求プロセスは、保存対象プロセスを WAKEUP する。

復元処理では、platename で指定したプレートから保存されている情報を元に、プロセス実行途中状態を復元する。具体的には、platename で指定したプレートから、保存されているプロセスの実行途中状態を取り出し、この情報を元にプロセスを復元する。また、コンテナ、コンテナボックスも同様に復元する。

5. 評価

5.1 測定環境と測定項目

Tender に実現したプロセス実行途中状態の保存復

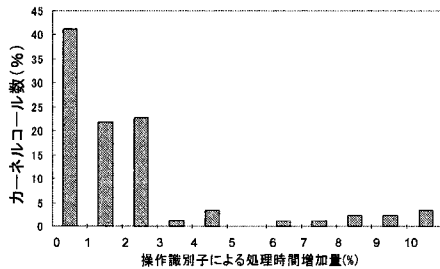


図 8 操作識別子による処理時間の増加量

元機能の評価を行うために、測定による評価を行った。測定には、プロセッサ PentiumIII 800MHz、メモリ 128MB の計算機を用いた。

ここでは、操作識別子導入に伴うカーネルコール処理時間の増加量について測定した。また、*Tender* に実現したプロセス実行途中状態の保存復元機能の基本性能を評価するために、実測を行った。具体的には、保存対象プロセス数、保存対象プロセスの大きさ、保存対象コンテナ数、保存対象コンテナの大きさを変え測定を行った。なお、保存機能の測定は、WAIT なしで保存可能な状態でを行った。測定では、各処理を呼び出してから戻ってくるまでの時間を測定した。

5.2 操作識別子による処理時間の増加量

92 個のカーネルコールの処理時間を測定し、操作識別子を使用する場合、使用しない場合で処理時間の比をもとめた。操作識別子による処理時間の増加量を図 8 に示す。85%のカーネルコールにおいて処理時間の増加量は、3%以内である。しかし、処理時間の短いカーネルコールでは、より大きな影響を受けた。また、全カーネルコールにおける処理時間の平均増加量は、2.65%である。

5.3 保存復元処理における処理時間

5.3.1 コンテナの大きさを変えた測定

保存復元処理において保存対象コンテナの大きさを変え、処理時間を測定した結果を図 9 に示す。このとき、保存対象プロセスは 2 つ、保存対象コンテナは 1 つである。また、各プロセスは、テキスト部 12KB、データ部 4KB、BSS 部 0KB、ユーザスタック部 64KB である。

図 9 より、処理時間は、コンテナの大きさが 4KB 増加するにしたがい、保存処理では 1.49 ミリ秒増加するにしたがい、保存処理では 1.49 ミリ秒増加している。したがって、保存処理は復元処理に比べ、保存対象コンテナの大きさの影響を受けやすいといえる。

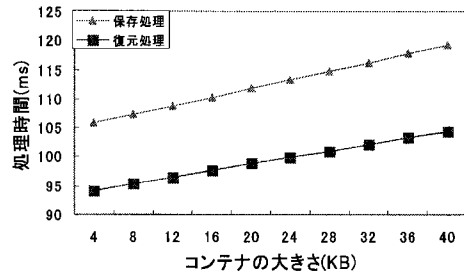


図 9 保存復元の処理時間 (パラメータ：コンテナの大きさ)

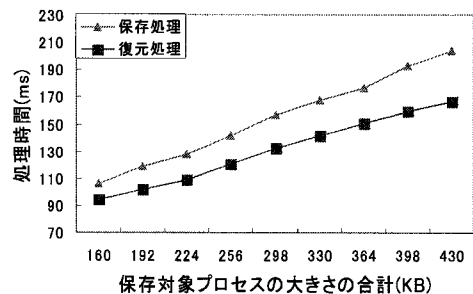


図 10 保存復元の処理時間 (パラメータ：プロセスの大きさ)

5.3.2 プロセスの大きさを変えた測定

保存対象プロセスの大きさを変え、処理時間を測定した結果を図 10 に示す。このとき、保存対象プロセスは 2 つ、保存対象コンテナは 1 つである。ここで、保存対象プロセスの大きさは、プロセスのテキスト部の大きさ、データ部の大きさ、BSS 部の大きさ、ユーザスタック部の大きさの合計である。コンテナの大きさは、4KB である。

図 10 より、処理時間は、保存対象プロセスの大きさが 32KB 増加するにしたがい、保存処理では 12.3 ミリ秒増加するにしたがい、保存処理では 12.3 ミリ秒増加している。したがって、図 9 と同様に、保存処理は復元処理に比べ、保存対象プロセスの大きさの影響を受けやすいといえる。

5.3.3 コンテナ数を変えた測定

保存対象コンテナの数を変え、処理時間を測定した結果を図 11 に示す。このとき、保存対象プロセスは 2 つであり、各プロセスはテキスト部 12KB、データ部 4KB、BSS 部 0KB、ユーザスタック部 64KB である。また、コンテナの大きさは全て 4KB である。

図 11 より、処理時間は、コンテナが 1 つ増えるにしたがい、保存処理では 1.56 ミリ秒増加するにしたがい、保存処理では 1.56 ミリ秒増加している。したがって、図 9 と同様

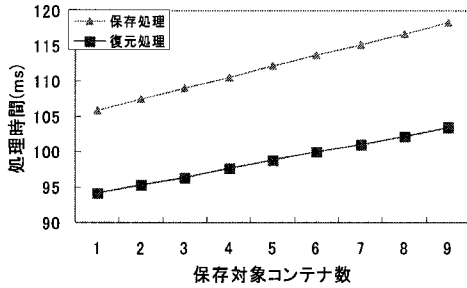


図 11 保存復元の処理時間 (パラメータ: プロセス数)

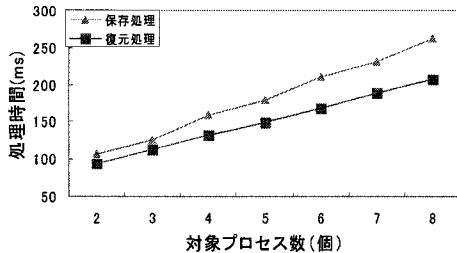


図 12 保存復元の処理時間 (パラメータ: プロセス数)

に、保存処理は復元処理に比べ、保存対象コンテナ数の影響を受けやすいといえる。

5.3.4 プロセス数を変えた測定

保存対象プロセス数を変更し、処理時間を測定した結果を図 12 に示す。このとき保存対象コンテナは 1 つであり、大きさは 4KB である。また、各プロセスは、テキスト部 12KB、データ部 4KB、BSS 部 0KB、ユーザスタック部 64KB である。

図 12 より、処理時間は、保存対象プロセス数が 1 つ増加するにしたがい、保存処理では 25.9 ミリ秒、復元処理は 18.9 ミリ秒増加している。したがって、図 9 と同様に、保存処理は復元処理に比べ、保存対象プロセス数の影響を受けやすいといえる。

6. おわりに

計算機の停止と再開の前後で、プロセス間通信を含むプロセス処理を継続できる機能の設計と実装について述べた。この機能を用い、システムの故障に伴う計算機停止後の再起動処理において、保存したプロセス実行途中状態の情報に基づき、プロセスの実行状態を復元することで、プロセス処理を迅速に回復できる。

この保存復元機能を *Tender* 上に実現し、操作識別子導入に伴うカーネルコール処理時間の増加量、およ

び保存復元機能の基本性能を測定した。操作識別子の導入に伴い、92 個のカーネルコールにおいて処理時間は平均して 2.6% 増加した。また、保存復元機能の性能測定の結果、保存対象プロセスの大きさ、および保存対象コンテナの大きさの合計が 32KB 増加するに依り、保存処理では、およそ 12 ミリ秒、復元処理ではおよそ 9 ミリ秒処理時間は増加する。このことから、復元処理に比べ、保存処理のほうが保存対象の大きさの影響を受けやすいことがわかった。

残された課題として、別計算機上のプロセスと通信を行っているプロセスの実行途中状態の保存復元機能の実現がある。

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 18300010) による。

参 考 文 献

- 1) J. Heo, S. Yi, Y. Cho, J. Hong, S. Y. Shin, "Space-efficient Page-level Incremental Checkpointing," Proceedings of the 2005 ACM symposium on Applied computing, pp.1558–1562, 2005.
- 2) S. Agarwal, R. Garg, M. S. Gupta, J. E. Moreira, "Adaptive Incremental Checkpointing for Massively Parallel Systems," Proceedings of the 18th annual international conference on Supercomputing, pp.277–286, June, 2004.
- 3) O. Laadan, J. Nieh, "Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems," Proceedings of 2007 USENIX Annual Technical Conference, pp.323–336, June 2007.
- 4) S. Yi, J. Heo, Y. Cho, J. Hong, "Adaptive Page-level Incremental Checkpointing based on Expected Recovery Time," Proceedings of the 2006 ACM symposium on Applied computing, pp.1472–1476, 2006.
- 5) R. Gioiosa, J. C. Sancho, S. Jiang, F. Petrini, "Transparent, Incremental Checkpointing at Kernel Level: a Foundation for Fault Tolerance for Parallel Computers," Proceedings of the 2005 ACM/IEEE SC-05 Conference, pp.9–22, 2005.
- 6) 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.
- 7) 福富和弘, 田端利宏, 谷口秀夫, "*Tender* における即時同期を可能にするプロセス間通信機構の実現と評価," 情報処理学会研究報告, No.2003-OS-93, pp.25–32, 2003.