資源の独立化機構による *Tender* オペレーティングシステム

谷口秀夫[†] 青木義則[†] 後藤真孝[†]
村上大介[†] 田端利宏[†]

オペレーティングシステムが制御し管理する対象を資源と呼び，その資源を独立化させる機構を提案し，提案方式に基づき開発している ***Tender*** オペレーティングシステムについて述べる．***Tender*** は，資源を資源識別子や資源名で管理し，資源を管理するプログラム部品とプログラム部品へのポインタを持つプログラムポインタ表からなる表プログラム構造で構成される．また，プログラムを部品化するために，共通プログラムを排除するコンパイル時複写プログラムの機構を持ち，プログラム部品の利用は，資源インタフェース制御と名づけた処理が一括管理する．さらに，資源の独立化機構を活用した機能の1つとして，プロセス生成消滅の高速化について，その機構を説明し，実測による評価結果を報告する．

Tender Operating System Based on Mechanism of Resource Independence

Hideo Taniguchi,[†] Yoshinori Aoki,[†] Masataka Goto,[†]
Daisuke Murakami[†] and Toshihiro Tabata[†]

In this paper the objects to be controlled and managed by operating system are called *resources*. We propose the mechanism of resource independence. And this paper describes ***Tender***, the operating system developing based on our proposed mechanism. ***Tender*** manages the resource by resource identifiers and resource names. ***Tender*** is composed of the program parts to control the resource and *Table Program Structure*, which has the pointers to them. And ***Tender*** has the Copy-on-Compile facility which makes to part program and avoids sharing the common program among program parts. Resource intarface controller manages call of the program part. As one of the functions of practical use of this mechanism, this paper describes the method of fast process creation and disappearance. The method is efficient by the evaluation.

1. まえがき

オペレーティングシステム（以降，OS と略す）の構造に関する研究は，数多くなされてきた．初期の UNIX [1),2)]は，そのプログラム構造が簡明であり，標準入出力と呼ぶ利便性の高い機能を提供した．これにより，数多く利用されているが，利用形態の多様化により高機能化され，プログラム構造は複雑なものになってしまっている．Mach[3)]は，OS 核の機能を削減し OS の大半の機能を OS 核外に実現するマイクロカーネルを実現した．これにより，多くの機能をサーバとして実現できるが，性能が問題になっている．日本での OS 研究のいくつかは，文献 4) にまとめられている．OS/omicron[4)]は，日本語情報処理のために16ビットの漢字コードをシステムの文字コードにした OS であり，タスクフォースと呼ぶ効率的な実行単位も提案している．文献 5) では，OS の構造記述に関する試みがなされており，OS 内の並行動作に注目して，並行動作部分を軽量なプロセスの集合で実現することを提案している．

我々は，OS が制御し管理する対象に注目し，それを資源と呼び，資源に関する処理をプログラム構造の中心とした OS 構築方式を提案し，さらに，その提案構築方式に基づく ***Tender***（The ENduring operating system for Distributed EnviRonment）の設計・開発を行い，いくつかの機能について検討している[6)~12)]．

UNIX では，入出力装置を制御するプログラムについて，表を基本にした構成で管理し，資源化している．また，マイクロカーネルの技術を利用し，物理資源を応用プログラム（以降，AP と略す）のレベルで管理することにより，性能や処理の柔軟性を向上させ

† 九州大学大学院システム情報科学研究科
Graduate School of Information Science and Electrical Engineering, Kyushu University

ている OS として Exokernel[13),14)]がある．一方，資源を分離し独立性を高める手法としてオブジェクト指向の考え方があり，これに基づく OS も研究されている[15)~20)]．いくつかの OS は，オブジェクトを支援する機能を提供している．また，OS 内部にオブジェクト指向の枠組みを適用する OS は，その適用を OS 処理の一部分に限定している．そもそも，オブジェクト指向の考え方は，資源の抽象化が主目的になるため，処理性能が低下する問題がある．このため，OS 処理の大半についてオブジェクト指向の枠組みを適用した研究は，資源の抽象化の効果を生かした分散 OS に関するもの[15)~17)]が多い．また，並列処理への適用例[18)]もある．さらに，その OS が資源を抽象化したい部分のみへ適用する OS もある．たとえば，Tornade[19)]では，メモリに関する資源のみに適用している．また，文献 20) では，ファイルシステムに適用している．

我々が提案する OS 構築方式は，オブジェクト指向の考え方を用いず，OS 処理の大半について，OS が制御し管理する資源を分離し，独立させ，資源が単独で存在できる特徴がある．このため，単独で存在する資源の利用を，プロセス管理やメモリ管理といった OS 内部モジュール相互だけではなく AP からも利用できる．また，各資源を管理するプログラムの呼び出しは，特定の処理プログラム（以降，資源インタフェース制御と名づける）を介する特徴がある．一般に，資源を細分化すると，処理のオーバヘッドが増加する．このため，資源の単独存在を可能にすることにより，資源の事前用意や保留による再利用で，資源の作成や削除の処理を高速化している．また，資源インタフェース制御が，各資源を管理するプログラムの呼び出しを管理するため，OS の動作や内部状態の理解や把握が容易になり，OS の理解を支援できる．さらに，各資源を管理するプログラムを部品化できるため，機能の追加や変更が容易になる．***Tender*** では，提案方式に基づき OS を構築し，さらに資源の永続化[7)]を可能にしようとしている．

本論文では，資源を分離し独立化する方式について説明する．また，この方式に基づいて開発している ***Tender*** オペレーティングシステムについて，開発内容を述べる．さらに，この方式を生かした例として処理の高速化を取り上げ，サービス処理実行の基本であるプロセスについて，その生成消滅の高速化の機構を説明し，実測による評価結果を報告する．

2. 資源の分離と独立化

OS は，制御する対象を資源として管理する．ここで，以降では，資源とは，資源を制御し管理する処理モジュールの外から，資源の識別と操作ができるものとする．つまり，AP からも，資源の識別と操作が可能である．資源には識別のために，文字列による名前（以降，資源名と呼ぶ）や数字による識別子（以降，資源識別子と呼ぶ）を付与する．この資源は，大きく 2 つに分類できる．1 つは，ファイルや入出力装置のように電源断でも内容が保持される資源（以降，永続資源と呼ぶ）である．もう 1 つは，プロセスやセマフォのように電源断で内容が消失する資源（以降，一時資源と呼ぶ）である．永続資源は，AP 記述の中で陽に指定し操作するため，利便性が高い資源名が付与されている．しかし，処理の高速化を行うため，OS 内部では資源名を資源識別子に変換して処理を行っている．たとえば，UNIX ファイルの場合，資源名は “/usr/bin/sh” であり，資源識別子は “i ノード番号” である．一時資源は，生成して利用する必要があるため，AP は資源生成により資源識別子を得，操作する．たとえば，プロセス識別子がこれにあたる．

既存の多くの OS では，OS が制御する対象をすべて資源として管理しているわけではない．このため，対象を制御し管理する処理モジュールの外からは，対象の識別と操作が制限される．また，AP が操作する資源としては，OS が制御する対象の一部分しか扱えない．特に，一時資源の場合，このことは顕著である．たとえば，プロセスを 1 つの資源として扱っている．プロセスを構成しているプログラムや仮想記憶空間や実メモリなどの要素は，資源として扱っていない．このため，AP が仮想記憶空間を作成できない．また，仮想記憶空間がプロセスの利用なしには存在できない．この結果，大きく 2 つの問題が生じている．1 つは，資源の単位が大きくなり，その作成や削除は多大な処理をともなってしまうことである．もう 1 つは，資源を構成する各要素間で管理表が共有され，各要素間の関係が複雑化していることである．各要素間の関係の複雑化により，資源操作の動作把握や資源管理の構造理解を困難にし，さらに，機能の追加や変更も難しくしている．

上記の問題を解決するため，資源の分離と独立化を提案する．このため，まず，資源の分離を行う．つまり，既存の OS の資源を細分割する．たとえば，既存の OS のプロセス資源を細分割し，プログラムと仮想記憶空間と実メモリなどを資源化する．資源には，単独に構成されるものと，他の資源を利用して構成されるものがある．例として，前者は実メモリ資源，後者はプロセス資源があげられる．次に，資源を独立化さ

せる．機能上，資源に資源識別子と資源名を付与し，かつ資源操作のインタフェースを統一する．また，プログラム構造上，資源の種類ごとに管理表を個別に用意し，他資源の管理表へのポインタを禁止する．さらに，資源の種類ごとに管理するプログラムを個別に用意し，共通プログラムを排除する．また，資源を管理するプログラムの呼び出しは，特定の処理プログラムである資源インタフェース制御を介するようにする．

一般に，資源の分離と独立化は処理性能の低下を招きやすい．このため，オブジェクト指向に見られるような資源の抽象化を避け，資源識別子と資源名に資源の種類に関する情報を含ませる．これにより，利用者は資源の種類の違いを意識しなくてはならないものの，利用者が資源の種類を指定することにより，処理性能の低下を抑えることができる．

このように資源の分離と独立化を行うことにより，先に述べた既存の OS の問題に対処できる．つまり，資源の事前用意や保留により，資源作成をともなう処理の高速化ができる．また，OS の動作や内部状態の理解や把握が容易になり，OS の理解を支援できる．さらに，プログラムを部品化できるため，機能の追加や変更が容易になる．以降では，資源を分離し独立化させた ***Tender*** オペレーティングシステムについて述べる．

3. *Tender* オペレーティングシステム

3.1 開発の方針と経過

Tender オペレーティングシステム[6]を開発する目的は，将来の OS に必要な機能や構造に関する研究基盤となる OS を構築することである．このため，プログラムの部品化に重点をおいた．OS 研究にとって，ソースコードを自由に変更できる OS を保有することは，非常に重要である．

ハードウェア環境として，AT 互換の計算機とした．特殊でないハードウェア環境にすることで，ハードウェア環境の構築が容易になり，開発担当者 1 人 1 人が OS を自由に変更できる環境を持てる．また，特殊なハードウェアに依存しない，多くの計算機で利用できる OS 機能を主眼に研究を行える．

大学での OS 開発であるから，開発者はほとんど学生である．このため，多人数によるソフトウェア開発技術の修得や開発担当者の移行に留意することととした．このため，年度前半は勉強と機能設計を行い，第 3 四半期にコーディングと試験を行い，年末に当年度の版を完成させる．第 4 四半期は，評価とまとめである．

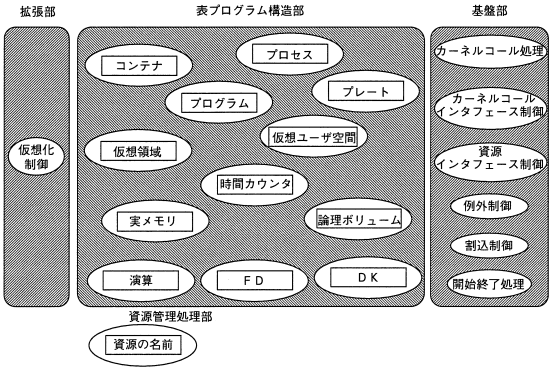


図 1 プログラム構造
Fig. 1 Program structure.

開発は，平成 7 年度に検討を開始し，毎年末に改版を重ねてきた．

設計書には，基本設計書と機能設計書および構造設計書がある．基本設計書は，***Tender*** の設計方針や特徴的な項目をまとめたもので，平成 7 年度に作成し，平成 8 年度に改版している．機能設計書は，実現する機能とインタフェースをまとめたもので，毎年改版している．構造設計書は，モジュール（C 言語の関数）の仕様をまとめたもので，ソースコードに記述されたモジュール仕様のコメントを抜粋するツールプログラムにより，自動作成される．

3.2 特　　徴

3.2.1 基 本 構 造

Tender のプログラムは大きく 3 つの部分に分けられる．基盤部，表プログラム構造部，および拡張部である．様子を**図 1** に示し，説明する．

基盤部は OS 動作の基盤となる処理を行う部分である．特に，資源インタフェース制御は ***Tender*** 特有の部分であり，表プログラム構造と名づけたプログラム管理構造に基づき，資源を管理しているプログラム部分（以降，資源管理処理部と名づける）の呼び出しを制御している．表プログラム構造とは，資源管理処理部を独立化させる機構の 1 つであり，詳細は後述する．

表プログラム構造部は，プロセス，プログラム，仮想ユーザ空間などの各資源管理処理部の集まりである．

拡張部は，OS 動作の拡張機能に位置づけられる処理を行う部分であり，仮想記憶機能を支援する仮想化制御がある．

3.2.2 資源の分離

Tender では，既存の OS の資源を細分割し，また新たな資源を導入して，多くの種類の資源を管理している．たとえば，***Tender*** では既存の OS のプロ

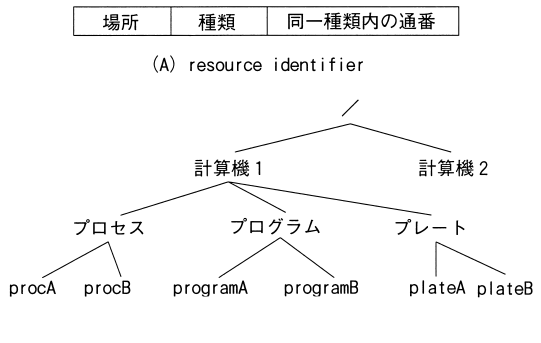


図 2 資源識別子と資源名
Fig. 2 Resource identifier and resource name.

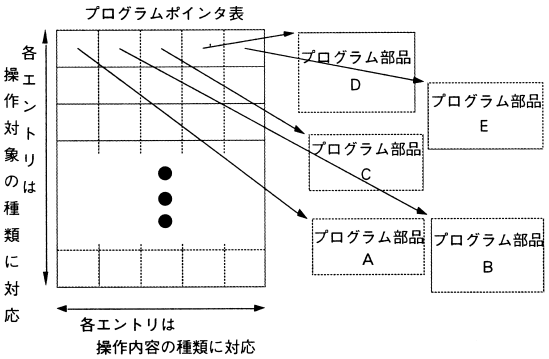


図 3 表プログラム構造
Fig. 3 Table program structure.

セスを,「プロセス」「プログラム」「仮想ユーザ空間」「仮想空間」「仮想領域」「実メモリ」の 8 つの資源に分割している．また，新たな資源として「演算」「プレート」がある．資源には，**図 2** に示す資源識別子と資源名を付与する．資源識別子は，資源の場所と種類と同一種類内の通番を情報として有する数字である．資源名は，場所名と種類名と固有名からなる文字列である．場所とは，資源が存在する計算機を指し，その数字や名前はシステム環境設定情報としている．種類については，OS で数字や名前を規定している．同一種類内通番は OS が資源生成時に決定する．固有名は AP が指定する．資源インタフェース制御は，資源名を管理する管理表を持ち，資源識別子と資源名の変換機能を提供する．

3.2.3 資源を管理しているプログラム部分の独立化

Tender では，資源を管理しているプログラム部分（資源管理処理部）を独立化させるため，大きく 2 つの機構を持つ．1 つは表プログラム構造であり，もう 1 つはコンパイル時複写プログラムの機構である．

表プログラム構造は，**図 3** に示すように，プログラム部品とプログラム部品へのポインタを持つプログラムポインタ表からなる．プログラムポインタ表の行要素と列要素は，操作する資源の種類と操作内容に対応している．資源管理処理部は，資源への操作を，資源の作成（open 系），削除（close 系），入力（read 系），出力（write 系）および制御（control 系）の 5 つに分類し，各々をプログラム部品として実現する．各プログラム部品の呼び出し形式は，操作する資源を特定する資源名または資源識別子，操作を要求するプロセス識別子，および操作内容を格納した領域へのポインタの 3 つの引数からなる．プログラムポインタ表は，資源インタフェース制御により管理される．まず，資源インタフェース制御は，プログラム部品の登録や削除および変更を行う機能を提供する．現在の版では，各資源処理管理部は，OS 立ち上げ時の初期化処理において，この機能を利用して，5 つのプログラム部品を登録している．次に，資源インタフェース制御は，プログラム部品の呼び出しを制御している．つまり，プログラム部品の呼び出しは，資源インタフェース制御へ依頼し，資源インタフェース制御がプログラムポインタ表を用いてプログラム部品を呼び出す機構としている．このため，資源インタフェース制御では，プログラム部品の呼び出し状況を把握することができ，プログラム部品の変更（更新や追加や削除）や OS の動作の可視化が可能になる．

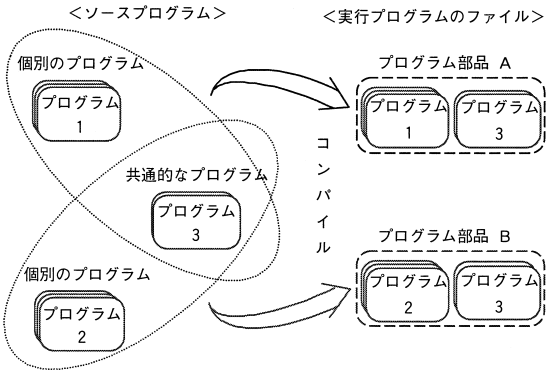


図 4 コンパイル時複写プログラムの機構
Fig. 4 Mechanism of copied program at compilation.

コンパイル時複写プログラムの機構とは，ソースプログラム上は共通的なプログラムとして管理し，コンパイル時に必要なプログラム部品に埋め込み，実行プログラムのファイル上は各プログラム部品に個別に存在させる機構である．様子を**図 4** に示す．共通的なプログラムであるプログラム 3 は，コンパイルによりプログラム部品 A とプログラム部品 B の両方に組み込まれる．これにより，実行プログラムにおける共有

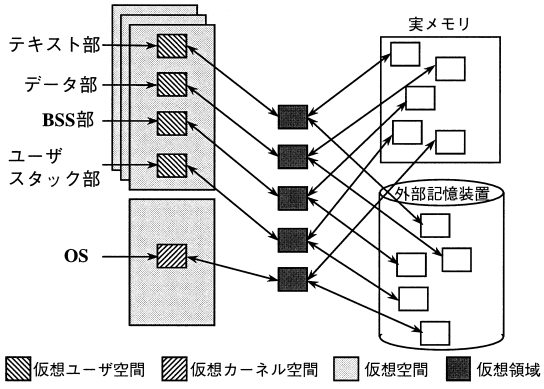


図 5 プロセスとメモリ管理関連の資源
Fig. 5 Resources of process and memory management.

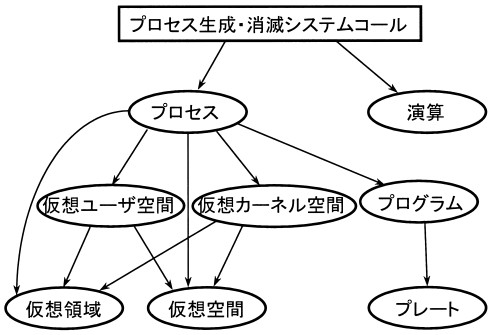


図 6 プロセスの生成と消滅時の資源管理部の呼び出し関係
Fig. 6 Relation of resource management parts on process creation and disappearance.

部分を解消することができ，プログラムの部品化ができる．共通的なプログラムの例としては，メモリ上のデータ複写処理プログラムがある．これにより，プログラム部品を変更する際に，そのプログラム部品を実行中である確率が減少し，変更が容易になる．具体的には，C コンパイラのプリプロセッサ機能を利用する．共通的なプログラム関数は，マクロで関数定義を行い，コンパイル時のプリプロセッサ機能によりインライン展開される．

3.2.4 機　　能

Tender では，資源を分離し独立化させることにより，多くの機能を容易に実現することが可能になっている．現在までに，いくつかの機能[7)~12)]を実現し，評価してきた．

資源インタフェース制御では，プログラム部品の呼び出し状況を把握することができ，「どのプログラム部品が実行されていないか」「どのプログラム部品が実行中か」「プログラム部品を実行しているプロセスはどれか」を容易に把握できる．このため，動作中の OS のプログラムの一部を変更する機能[8)]や OS の動作を可視化する機能[9)]が実現できる．また，プロセスへのプロセッサ割当てを資源化した「演算」資源の導入により，プロセスの実行速度を調整できるスケジュール法[10),11)]を実現できる．さらに，プロセスとメモリ関連の資源を独立に扱えるため，単一仮想記憶と多重仮想記憶の長所をあわせ持つヘテロ仮想記憶[12)]を実現できる．

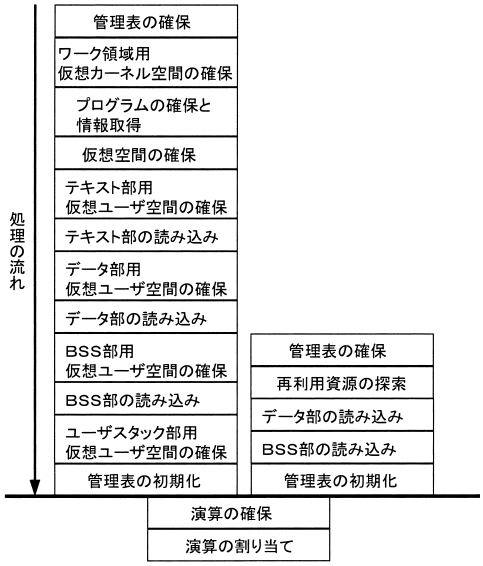
先に述べたように，資源の分離と独立化を行うことにより，処理を高速化できる．プロセスは，サービス処理実行の基本であるとともに，走行するためにメモリ関連の資源やプロセッサの資源および外部記憶装置に格納されたプログラムに関する資源と多くの資源を利用する．そこで，以降では，プロセス生成消滅の処理の高速化について述べる．

3.3 プロセス生成消滅の高速化

3.3.1 高速化の機構

プロセスとメモリ管理関連の資源の関係を**図 5** に示す．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，アドレス変換表に相当する．仮想ユーザ空間とは，メモリイメージを仮想化した領域である仮想領域をユーザ空間用の仮想空間に貼り付けることで作成できる．仮想領域の実体は，実メモリ，または，外部記憶装置上に存在する．「貼り付ける」とは，仮想アドレスを実アドレスに対応づけすることであり，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．仮想カーネル空間は，仮想領域を OS 用の仮想空間に貼り付けることにより作成される．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．

プロセスの生成と消滅時の各資源管理部の呼び出し関係を**図 6** に示し，以降に資源再利用の仕組みについて説明する．プロセスの生成時に，プロセス管理部は，仮想領域，仮想ユーザ空間，仮想空間，仮想カーネル空間，プログラムの各資源管理部に対して，処理を依頼する．ここで，プログラムとは，実行プログラムについて，テキスト部とデータ部の大きさと先頭アドレス，および開始アドレスの情報を持つ資源であり，実行プログラムの形式を隠蔽する．また，プログラムは，プレートを用いて実行プログラムを外部記憶装置上ではなくメモリ上に格納している．プレート[7)]とは，メモリ上のデータに永続性を持たせた資源である．各資



(A) 資源を再利用しない場合 (B) 資源をすべて再利用できる場合

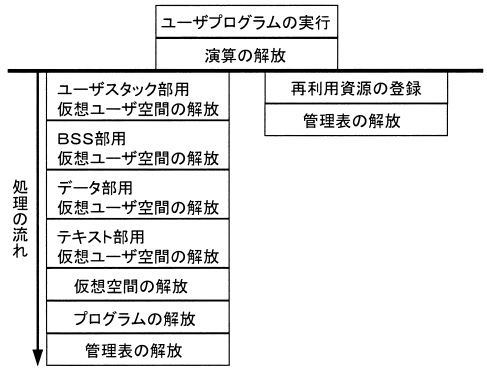
図 7 プロセスの生成処理

Fig. 7 A flow of process creation.

源管理部に処理を依頼するときには，確保したい資源の情報を与え，各資源管理部から，確保した資源に関する情報を受け取る．したがって，プロセス管理は，プロセスの消滅時に，再利用可能な資源に関する情報を保存し，プロセスの生成時に，要求する資源と同じ条件のものを保存していればその資源を再利用する．これにより，各資源管理部に資源の生成を依頼することなく，資源を利用することができる．つまり，プロセスの生成の高速化は，プロセスとは独立して存在する資源を事前生成または再利用することで実現できる．また，プロセスの消滅の高速化は，再利用可能な資源を解放せずに保存することで実現できる．

資源を再利用しない場合と再利用する場合のプロセスの生成処理の様子を**図 7** に示す．図 7 (A) に示すように，資源を再利用しない場合には，管理表の確保から管理表の初期化までの 12 段階の処理でプロセスを生成しなければならない．このため，プロセスの生成のオーバヘッドが非常に大きくなる．これに対し，資源を再利用する場合には，必要になる資源を生成せずに，再利用するのみでプロセスの生成が可能になる．これにより，プロセス生成のオーバヘッドを最小限に抑えることができる．図 7 (B) に示すように，資源をすべて再利用できる場合には，5 段階の処理のみでプロセスを高速に生成できる．

資源を再利用しない場合と再利用する場合のプロセスの消滅処理の様子を**図 8** に示す．図 8 (A) に示すよ



(A) 資源を再利用しない場合 (B) 資源をすべて再利用する場合

図 8 プロセスの消滅処理

Fig. 8 A flow of process disappearance.

うに，資源を再利用しない場合には，すべての資源を削除するため，ユーザスタック部用仮想ユーザ空間の解放から管理表の解放までの 7 段階の処理を必要とする．これに対し，資源を再利用する場合には，図 8 (B) に示すように，各資源を再利用するために情報を登録し，管理表のエントリの解放を行うのみで，高速にプロセスを消滅させることができる．

同一のプログラムからなるプロセスが複数存在するときには，テキスト部をこれらのプロセス間で共有する．共有は，テキスト部の仮想領域を共有し，それぞれの仮想空間に貼り付けることで実現できる．共有することにより，メモリ資源を節約でき，プロセスの生成や消滅の処理を高速化できる．上記のテキスト部の共有は，UNIX でも行われており，プロセスの生成や消滅の高速化への効果に差はない．一方，UNIX では，Sticky Bit によるテキスト部の再利用を行っている．これに対し，***Tender*** では，テキスト部の内容だけでなく，プロセスが使用していた仮想記憶空間も再利用できる．このため，Sticky Bit によりテキスト部を再利用した場合に比べ，再利用の効果は大きくなる．

3.3.2 再利用可能資源と利用インタフェース

プロセスを構成する資源のうち再利用可能な資源の一覧と再利用可否の観点を**表 1** に示す．ワーク領域用仮想カーネル空間とは，プロセスの生成時に利用する一時的な領域である．この大きさはつねに 4 KB と一定なので，OS を起動して最初にプロセスを生成するときに作ったワーク領域を再利用のために保存しておき，そのワーク領域の情報をプロセス管理が保持する．そして，それ以降のプロセス生成処理でその情報を利用することにより，この資源を再利用できる．この部分は，プロセスの生成時には，必ず必要な資源であり，かつその大きさは小さく固定（4 KB）なので，

表 1 再利用可能資源の一覧
Table 1 List of recyclable resources.

通番	再利用可能な資源	再利用可否の観点
1	ワーク領域用仮想カーネル空間	（つねに再利用）
2	内容を利用するテキスト部用仮想ユーザ空間	プログラムの内容とアドレス位置
3	内容を利用しない仮想ユーザ空間（テキスト部，データ部，BSS 部，ユーザスタック部用）	アドレス位置と大きさ
4	仮想空間	仮想空間の内容
5	仮想領域（テキスト部，データ部，BSS 部，ユーザスタック部用）	大きさ
6	プログラム	プログラムの内容

表 2 プロセスの作成と削除のインタフェース
Table 2 Interface of process creation and disappearance.

形式	引数の内容	機能
open_proc (plateid, arg, vmid)	plateid：プロセスとして起動するプログラムを識別する識別子 arg：プロセスへの引数 vmid：プロセスを生成する仮想空間の識別子（0 のときは新規仮想空間）	plateid で指定されたプログラムをプロセスとして生成する．vmid $\neq$ 0 であれば，vmid で指定された仮想空間上にプロセスを生成する．
close_proc (rflag)	rflag：再利用のため残存させる資源を指示するフラグ	rflag で指定された資源を保存し，プロセスを消滅させる．

つねに再利用した方がよいと考えられる．テキスト部用仮想ユーザ空間は，書き込み不可なのでその内容を利用する場合と利用しない場合に分類できる．内容を利用する場合は，プログラムの内容とアドレス位置により，再利用可否の判断ができる．内容を利用しない仮想ユーザ空間の場合は，テキスト部，データ部，BSS 部，およびユーザスタック部について再利用可能であり，アドレス位置とその大きさにより，再利用可否の判断ができる．仮想空間は，仮想空間に仮想ユーザ空間が貼り付いていなければ，仮想ユーザ空間を再利用しないときでも再利用できる．また，仮想ユーザ空間を再利用するときは，仮想ユーザ空間の貼り付いている仮想空間も再利用することになる．仮想領域は，テキスト部，データ部，BSS 部，およびユーザスタック部について再利用可能であり，特定の大きさを持つメモリ空間なので，大きさ情報のみで再利用可否の判断ができる．プログラムは，そのプログラムの内容が同じかどうかを調べることで再利用可否の判断ができる．

プロセスの作成と削除の利用インタフェースを**表 2** に示す．再利用可能な資源を個別に自由に選択できるために，各資源をビットに対応させたフラグ（rflag）で選択するようにした．プロセスの生成処理では，資源を再利用する方が必ず速いと推測できるので，そのフラグを見るのは，プロセスの消滅時の処理のみとした．プロセスを消滅させる際に，フラグで選択された資源の情報を登録し，その資源を保存する．プロセスの生成時の処理では，再利用可能な資源を検索し，再利用可能な資源が存在すれば，その資源を再利用してプロセスを生成する．

3.3.3 基本性能の評価と考察

測定は，プロセッサ i486DX2（66 MHz）メモリ 32 MB の計算機上で，***Tender*** Ver.3.0 を走行させ，プロセスの生成と消滅の処理を 1000 回繰り返し行い，処理の平均時間を算出した．測定時に，走行するプロセスは，時間を計測し，プロセスの生成と消滅の処理を 1000 回繰り返すプロセスと，繰り返し生成されるプロセスである．繰り返し生成されるプロセスは，何も実行しないで，すぐに自分自身を消滅させる処理を実行する．そのプロセスの大きさは，BSS 部 4 KB，ユーザスタック 8 KB とし，テキスト部とデータ部のサイズを変えて測定した．資源再利用の効果を把握しやすくするため，ディスク I/O は発生しない環境とした．また，時間は 1 ms 周期で割り込むタイマを用いて測定した．

測定結果を**図 9**，**図 10**，**図 11**，および**表 3** に示す．図 9 は，すべての資源を再利用した場合とそうでない場合について，データ部サイズとプロセス生成消滅時間の関係を示したものである．テキスト部サイズは 4 KB 固定である．図 10 は，すべての資源を再利用した場合と内容を利用するテキスト部用仮想ユーザ空間以外の資源を再利用した場合について，テキスト部サイズとプロセス生成消滅時間の関係を示したものである．データ部サイズは 4 KB 固定である．図 11 は，内容を利用しない仮想ユーザ空間，仮想領域と仮想空間およびプログラムを再利用した場合について，データ部サイズとプロセス生成消滅時間の関係を示したものである．テキスト部サイズは 4 KB 固定である．表 3 は，ワーク領域用仮想カーネル空間と仮想領域と仮想

表 3 ワーク領域用仮想カーネル空間と仮想領域と仮想空間の再利用効果
Table 3 Recycle effect of virtual kernel space for work area, virtual region and virtual space.

通番	再利用資源	プロセス生成消滅時間（ms）
1	資源を再利用しない場合	17.22
2	ワーク領域用仮想カーネル空間	15.77
3	仮想領域（ワーク領域用仮想カーネル空間，テキスト部，データ部，BSS 部，ユーザスタック部用）	11.45
4	ワーク領域用仮想カーネル空間 仮想領域（テキスト部，データ部，BSS 部，ユーザスタック部用）	11.20
5	ワーク領域用仮想カーネル空間 仮想空間	8.86

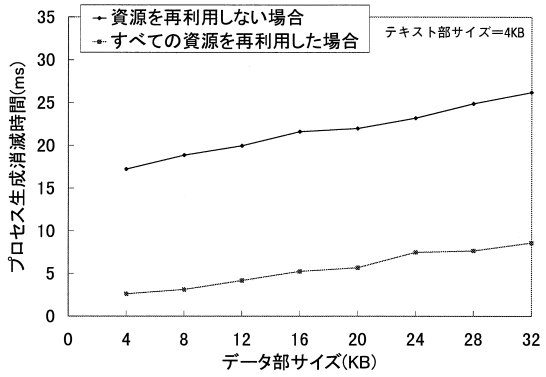


図 9 資源再利用の効果
Fig. 9 Effect of recycling resource.

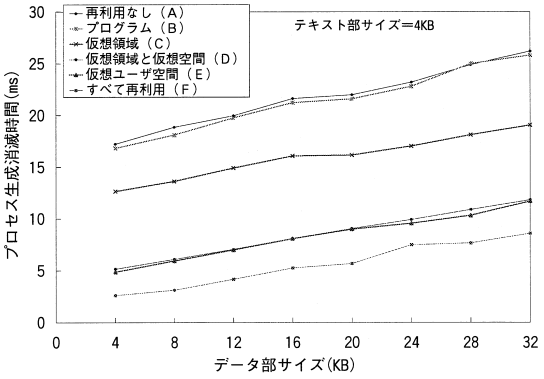


図 11 測定結果
Fig. 11 Result of measurement.

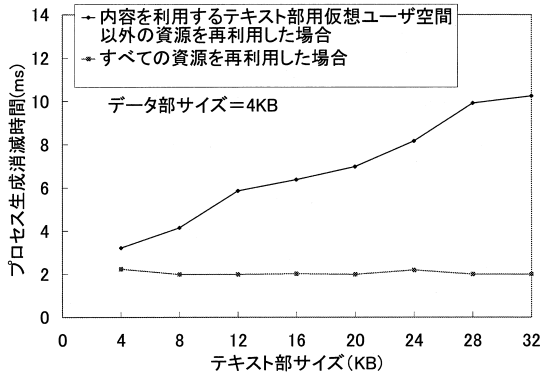


図 10 内容を利用するテキスト部用仮想ユーザ空間の再利用効果
Fig. 10 Recycle effect of virtual user space for text.

空間の再利用効果を示している．表 3 は，テキスト部とデータ部のサイズがともに 4 KB の場合である．

〈すべての資源を再利用した場合の効果〉

図 9 より，プロセス生成消滅時間は，すべての資源を再利用することにより，資源を再利用しない場合に比べて，15 ms 以上高速化できる．また，図 9 の 2 つの場合の傾きの差分より，データ部サイズに対し約 0.35 ms/4 KB の時間短縮効果があることが分かる．

〈ワーク領域用仮想カーネル空間と仮想空間の再利用効果〉

表 3 より，資源を再利用しない場合（通番 1）と，ワーク領域用仮想カーネル空間のみを再利用した場合（通番 2）のプロセス生成消滅時間の差は，1.45 ms である．つまり，ワーク領域用仮想カーネル空間を再利用することにより，プロセス生成消滅を 1.45 ms 高速化できることが分かる．

次に，仮想カーネル空間の作成時にかかる時間の内訳について考察する．仮想カーネル空間の作成は，まず仮想領域を作成し，その仮想領域をカーネル空間用の仮想空間に貼り付けることにより実現できる．そこで，仮想領域のみを再利用した場合（通番 3）と，仮想領域とワーク領域用仮想カーネル空間を再利用した場合（通番 4）を比べる．表 3 の通番 3 と通番 4 から，後者の方が 0.25 ms 速く処理が終わることが分かる．この 2 つの場合の処理の違いは，ワーク領域用仮想カーネル空間用の仮想領域を仮想空間に貼り付けるかどうかという違いしかないので，この時間の差は 4 KB の仮想領域を仮想空間に貼り付ける処理時間といえる．したがって，1.20 ms の時間が，4 KB の仮想領域の作成時間である．

仮想空間の再利用効果を，表 3 から推察する．仮想空間には，大きさや位置などの情報はなく，プロセスの生成時に必要な仮想空間は 1 つであるため，再利用効果は一定である．表 3 の通番 2 と通番 5 の差から，仮想空間の再利用により 6.91 ms の高速化ができるこ

とが分かる．

〈内容を利用するテキスト部用仮想ユーザ空間の再利用効果〉

図 10 より，内容を利用するテキスト部用仮想ユーザ空間を再利用することで，テキスト部サイズに関係なくプロセス生成消滅時間をほぼ一定にできることが分かる．これは，テキスト部用の仮想ユーザ空間の内容を利用することにより，テキスト部用の仮想ユーザ空間にテキスト部の内容を読み込むため処理が必要なくなるためである．今回の測定では，ディスク I/O は発生しない環境にしているため，2 つの場合のグラフの差は，テキスト部の内容をメモリ間で複写する時間である．

したがって，内容を利用するテキスト部用仮想ユーザ空間の再利用は，テキスト部サイズの大きなプロセスを生成する場合に効果が大きいといえる．

〈内容を利用しない仮想ユーザ空間の再利用効果〉

図 11 の (A) と (E) より，プロセス生成消滅時間は，内容を利用しない仮想ユーザ空間 (E) を再利用することにより，資源を再利用しない場合 (A) に比べて，9 ms 以上高速化できる．

仮想空間の再利用効果は一定であるので，内容を利用しない仮想ユーザ空間と仮想領域の再利用効果は，この 2 つの場合の傾きの差分より，約 0.29 ms/4 KB の時間短縮効果があることが分かる．

〈仮想領域の再利用効果〉

図 11 の (A) と (C) より，プロセス生成消滅時間は，仮想領域を再利用 (C) することにより，資源を再利用しない場合 (A) に比べて，4.5 ms 以上高速化できる．表 1 に示したように，仮想領域の再利用効果は，その大きさに比例する．したがって，仮想領域を再利用した場合の再利用効果は，この 2 つの場合の傾きの差分より，約 0.35 ms/4 KB の時間短縮効果があることが分かる．

内容を利用しない仮想ユーザ空間の再利用効果が，約 0.29 ms/4 KB であるのに対し，仮想領域のみの再利用効果の方が，約 0.35 ms/4 KB で効果が高い．内容を利用しない仮想ユーザ空間の再利用効果には，仮想領域の再利用効果も含まれるため，仮想領域のみの再利用効果よりも効果が大きいと予想できる．しかし，内容を利用しない仮想ユーザ空間を再利用する場合のプロセスの生成処理では，テキスト部，データ部，BSS 部，ユーザスタック部の条件に合う仮想ユーザ空間を 1 つ探す．次に，条件に合った仮想ユーザ空間が貼り付いている仮想空間に，貼り付いている他の仮想ユーザ空間について，再利用できるか否かを調べ，できなければ仮想空間から剥がす．このような処理をしているため，仮想領域のみを再利用する場合に比べオーバヘッドが大きく，再利用効果が小さくなったと考えられる．

〈プログラムの再利用効果〉

図 11 の (A) と (B) より，プログラムを再利用した場合 (B) と資源を再利用しない場合 (A) のプロセス生成消滅時間から，プログラムの再利用効果はわずかであることが分かる．

プロセスの生成処理で，プロセス管理は，プログラム管理に対しプログラムを確保するように要求する．次に，プログラム管理は，プレート管理に対してプログラムのデータを外部記憶装置からメモリ中に読み込むように要求し，プログラムを解析する．プログラムを再利用することにより，プレート管理に対し，プログラムのデータをメモリ中に読むように要求せず，プログラムを解析しないため，わずかに高速化できる．図の時間には，ディスク I/O の時間は含まれていない．実際の環境では，プログラムのデータを外部記憶装置からメモリ中に読み込む際に，ディスク I/O が生じる．プログラムを再利用すると，データをメモリ中に読み込む必要がなくなるため，ディスク I/O の発生する環境でもディスク I/O は発生しない．ディスク I/O の時間は，メモリの操作にかかる時間に比べて，かなり大きい．したがって，プログラムの再利用の効果は大きいといえる．

〈仮想領域と仮想空間を再利用した場合と内容を利用しない仮想ユーザ空間の再利用効果〉

図 11 の (D) と (E) より，仮想領域と仮想空間を再利用した場合 (D) と仮想ユーザ空間を再利用した場合 (E) を比較するとほとんど差がないことが分かる．この 2 つの場合の処理の違いは，仮想領域を仮想空間に貼り付けるかどうかという違いだけである．つまり，仮想領域を仮想空間に貼り付ける処理にはそれほど時間はかからないということが分かる．

ここで，資源を再利用できる頻度を考える．仮想領域は，大きさ情報のみで再利用できるのに対し，仮想ユーザ空間は，大きさとアドレス位置の情報が必要である．また，前者の場合は，仮想領域が再利用できなくても仮想空間が再利用できる場合もありうるし，この逆もありうる．したがって，仮想領域と仮想空間を再利用する場合は，わずかに効果が小さいものの，再利用できる頻度が大きいといえる．

同じプログラムファイルを使ってプロセスが何度も生成消滅する場合，仮想ユーザ空間を再利用する資源として指定すると，次にそのプロセスを生成するとき

に仮想ユーザ空間をすべて再利用できるため，効果が大きい．一方，あまり使われないプログラムファイルを利用したプロセスの仮想ユーザ空間を再利用するように指定すると，再利用される可能性が小さい．仮想領域と仮想空間を再利用する場合は，大きさ情報で再利用でき，再利用できる頻度は大きいので，あまり使われないプログラムのプロセスを消滅させる場合に，仮想領域と仮想空間を再利用するように指定すると資源が再利用されやすいと考えられる．

3.3.4 既存 OS との比較

既存 OS と比較する．測定には，PentiumII 450 MHz の計算機を利用し，ハードウェアクロックのカウンタを利用して処理時間を測定した．

図 10 で説明したように，すべての資源を再利用した場合，***Tender*** のプロセス生成消滅時間は，テキスト部の大きさに影響しない．データ部や BSS 部を変えてプロセス生成消滅時間を測定した結果，プロセス生成消滅時間（T：マイクロ秒）は，以下の式で近似できることが分かった．

$$T = 14.5 \times (\text{データ部の大きさ：KB}) + 2.0 \times (\text{BSS 部の大きさ：KB}) + 130 \quad (1)$$

また，BSD/UNIX の vfork/exec システムコールの処理時間を測定した．具体的には，vfork システムコールを発行し，wait システムコールにより子プロセスの終了を待ち，制御が戻ってくるまでの時間である．なお，子プロセスは，exec システムコールを発行する．exec システムコールで実行されるプログラムは，即座に終了するものである．実測の結果，処理時間は，約 1700 マイクロ秒であった．

最初に，プログラムの大きさとプロセス生成消滅時間について比較する．式 (1) と vfork/exec システムコールの処理時間約 1700 マイクロ秒から，

$$14.5 \times (\text{データ部の大きさ：KB}) + 2.0 \times (\text{BSS 部の大きさ：KB}) + 130 < 1700 \quad (2)$$

を満足するプログラムの場合，そのプロセス生成消滅時間は，すべての資源を再利用することにより，***Tender*** が BSD/UNIX より短いといえる．さらに，すべての資源を再利用した場合，プログラム資源の再利用により，プログラム実行時に新たな入出力は発生しない．一方，BSD/UNIX では，copy on write 機能や ODP 機能を利用しているため，プログラム実行時にページ例外が発生し新たな入出力が発生する．

次に，実 AP として，BSD/UNIX の OS カーネルの生成について，***Tender*** で行った場合と BSD/UNIX で行った場合を比較する．BSD/UNIX の OS カーネルを生成する際に実行されるプログラムについて，その大きさと生成消滅回数を実測した．その一覧を**表 4** に示す．表 4 と式 (1) より，***Tender*** において，すべての資源を再利用した際のプロセス生成消滅時間は，約 0.47 秒である．一方，BSD/UNIX におけるプロセス生成消滅時間は，約 1.96 秒（1700 マイクロ秒 × 1155 回）である．したがって，***Tender*** で BSD/UNIX の OS カーネルの生成を行った場合，BSD/UNIX で行った場合に比べ，プロセス生成消滅時間が約 4.2 倍高速になるといえる．もちろん，先にも述べたように，BSD/UNIX ではプログラム実行時にページ例外が発生し新たな入出力が発生するが，***Tender*** では発生しない．ただし，***Tender*** では，プログラム資源のために，多くのメモリを消費する．たとえば，よく利用されるプログラム（cpp，as，cc1，gcc2）をプログラム資源としてメモリ常駐させると，約 1.25 MB の実メモリが必要になる．

表 4 BSD/UNIX の OS カーネルの生成処理における実行プログラム一覧
Table 4 Program that is executed in making BSD/UNIX kernel.

name	text	data	BSS	times
cpp	36864	4096	13092	282
as	53248	16384	34964	277
cc1	970752	40960	67324	272
gcc2	40960	4096	0	272
sh	118784	8192	3348	14
[	3472	184	72	11
rm	2800	140	28	8
ln	912	140	16	4
ar	12288	4096	0	2
ld	28672	4096	620	2
cat	1956	132	36	1
chmod	1348	140	0	1
date	3204	336	12	1
expr	3876	2256	24	1
hostname	488	140	0	1
mv	1500	136	8	1
pwd	360	140	0	1
ranlib	5136	288	380	1
size	3516	180	360	1
strip	2480	136	0	1
touch	2588	128	0	1
total				1155

4. むすび

資源を分離し独立化することを提案し，その機構を実現した ***Tender*** オペレーティングシステムについて述べた．***Tender*** は，資源を資源識別子や資源名で管理し，プログラム部品とプログラム部品へのポイ

ンタを持つプログラムポインタ表からなる表プログラム構造で構成される．また，プログラム部品の利用は資源インタフェース制御と名づけた処理が一括管理する．さらに，プログラムを部品化するために，共通プログラムを排除するコンパイル時複写プログラムの機構を持つ．資源の独立化機構を活用した機能の 1 つとして，プロセス生成消滅の高速化について述べた．表プログラム構造を用いた資源の分離と独立化により，資源の単位を小さくしても処理性能の低下を抑えることができることを示した．さらに，資源の再利用により，処理性能が向上することを示した．具体的には，プロセスが利用する資源を再利用してプロセス生成消滅を高速化する方式を示し，その利用インタフェースを述べた．また，評価により，個々の資源の再利用はプロセス生成消滅時間を高速化できることを示した．すべての資源を再利用すると，プロセス生成消滅時間を 15 ms 以上短縮でき，データ部サイズに対し約 0.35 ms/4 KB の時間短縮効果がある．たとえば，多くの Web サーバで利用されている Apache ではプロセスの生成や消滅が起こりやすいため，資源再利用の効果が出ると考えられる．

今後は，資源を扱う処理間の連携処理の負荷について検討するとともに，資源の再利用の有効性について詳細に検討する．また，資源の独立化機構を活用した多くの機能を ***Tender*** に実現する予定である．

謝辞 ***Tender*** の開発を我々とともに進めきた，九州大学工学部情報工学科の田中徳穂氏〔現在,(株) 国際電気〕および，九州大学大学院システム情報科学研究科情報工学専攻の長嶋直希氏〔現在,(株) 富士通エフ・アイ・ピー〕，野口裕介氏，市川正也氏〔現在,(株) 日立製作所〕，坂口修氏〔現在，沖電気 (株)〕に感謝します．

参 考 文 献

1) Ritchie, D.M. and Thompson, K.: The UNIX Time-Sharing System, *Comm. ACM*, Vol.17, No.7, pp.365–375 (1974).
2) Ritchie, D.M. and Thompson, K.: The UNIX Time-Sharing System, *The Bell System Technical Jouranl*, Vol.57, No.6, pp.1905–1929 (1978).
3) Accetta, M., Baron, R., Bolosky, W.J., Golub, D., Rashid, R., Tevanian, A. and Young, M.: Mach : A New kernel Foundation for UNIX Development, *Proc. Summer USENIX Conf.*, pp.93–112 (1986).
4) 特集：日本におけるオペレーティングシステム研究の動向，情報処理学会学会誌，Vol.36, No.8, pp.698–768 (1995).
5) 田胡和哉，益田隆司：オペレーティング・システムの構造記述に関する一試み，情報処理学会論文誌，Vol.25, No.4, pp.524–534 (1984).
6) 谷口秀夫：分散指向永続オペレーティングシステム *Tender*，情報処理学会コンピュータシステムシンポジウム論文集，Vol.95, No.7, pp.47–54 (1995).
7) 谷口秀夫，市川正也：*Tender* オペレーティングシステムにおける資源の永続化機構，情報処理学会研究報告，Vol.99, No.32, pp.7–12 (1999).
8) 後藤真孝，谷口秀夫，牛島和夫：OS プログラムの動的入替え法，情報処理学会コンピュータシステムシンポジウム論文集，Vol.96, No.7, pp.75–80 (1996).
9) 野口裕介，谷口秀夫，牛島和夫：OS 動作の可視化機能の設計，情報処理学会コンピュータシステムシンポジウム論文集，Vol.96, No.7, pp.139–146 (1996).
10) 村上大介，谷口秀夫，牛島和夫：異なるスケジューラの共存制御法，情報処理学会研究報告，Vol.95, No.79, pp.105–112 (1995).
11) 谷口秀夫：サービス処理時間を調整するプロセスのスケジュール法，電子情報通信学会論文誌，Vol.J81-D-I, No.4, pp.386–392 (1998).
12) 谷口秀夫，長嶋直希，田端利宏：単一仮想記憶と多重仮想記憶を共存させたヘテロ仮想記憶の実現，情報処理学会研究報告，Vol.98, No.33, pp.87–94 (1998).
13) Dawson, R.E., Frans, K.M. and James Jr., O.: Exokernel: An Operating System Architecture for Application-Level Resource Management, *ACM SIGOS* (*SOSP*), Vol.29, No.5, pp.251–266 (1995).
14) Kaashork, M.F., et al.: Application Performance and Flexibility on Exokernel Systems, *ACM SIGOS* (*SOSP*), Vol.31, No.5, pp.52–65 (1997).
15) Edward, D., et al.: The Architecture of the Eden System, *Proc. 8th Symp. on Operating Systems Principles*, pp.148–159 (1981).
16) Black, A.P.: Supporting Distributed Applications: Experience with Eden, *ACM SIGOS* (*SOSP*), Vol.19, No.5, pp.181–193 (1985).
17) Rozier, M., et al.: CHORUS Distributed Operating Systems, *Computing Systems*, Vol.1, No.4, pp.305–370 (1988).
18) Andrew, S.G., Jon, B.W. and Strayer, W.T.: Portable Run-Time Support for Dynamic Object-Oriented Prallel Processing, *ACM Trans. Comp. Sys.*, Vol.14, No.2, pp.139–170 (1996).
19) Ben, G., Orran, K., Jonathan, A. and Michael, S.: Tornado: Maximizing Locality and Concurrency in a Shared Memkory Multiprocessor Op-

erating System, *OSDI'99*, pp.87–100 (1999).
20) Smolik, T.: An Objevt-Oriented File System – An Example of Using the Class Hierarchy Framework Concept, *SIGOSR*, Vol.29, No.2, pp.33–53 (1995).

(平成 11 年 11 月 18 日受付)
(平成 12 年 10 月 6 日採録)

谷口　秀夫（正会員）

昭和 53 年九州大学工学部電子工学科卒業．昭和 55 年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和 62 年同所主任研究員．昭和 63 年 NTT データ通信（株）開発本部移籍．平成 4 年同本部主幹技師．平成 5 年九州大学工学部助教授．平成 12 年九州大学大学院システム情報科学研究院助教授．博士（工学）．オペレーティングシステム，分散処理に興味を持つ．著書「オペレーティングシステム」（昭晃堂）．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員．

青木　義則（正会員）

平成 7 年九州大学工学部情報工学科卒業．平成 9 年同大学大学院システム情報科学研究科修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所に入所．分散処理，ヒューマン・コンピュータ・インタラクションの研究開発に従事．ACM 会員．

後藤　真孝（正会員）

平成 7 年九州大学工学部情報工学科卒業．平成 9 年同大学大学院システム情報科学研究科情報工学専攻修士課程修了．同年（株）東芝入社．オペレーティングシステムや IP ネットワークに興味を持つ．

村上　大介（正会員）

平成 7 年九州大学工学部情報工学科卒業．平成 9 年同大学大学院システム情報科学研究科修士課程修了．同年（株）FFC 入社．入出力制御ソフトウェア，ファームウェアの開発に従事．

田端　利宏（学生会員）

平成 10 年九州大学工学部情報工学科卒業．平成 12 年同大学大学院システム情報科学研究科修士課程修了．現在，同大学大学院システム情報科学府博士後期課程在学中．オペレーティングシステムに興味を持つ．