システムソフトウェアと
オペレーティング・システム 61－7
（１９９３．８．１８）

分散仮想記憶に基づくオペレーティングシステム DM-1の構成

篠原 拓嗣 † 藤川 賢治 † 大久保 英嗣 †† 津田 孝夫 †

† 京都大学工学部情報工学科
†† 立命館大学理工学部情報工学科

我々は，分散仮想記憶に基づいたオペレーティングシステム DM-1を設計・開発している．分散仮想記憶は，システム上に唯一つ存在する仮想的なアドレス空間であり，その実体は各サイトの主記憶および二次記憶である．分散仮想記憶によって，メモリ資源の透過性や規模透過性，障害透過性などが実現可能となる．

DM-1では，分散仮想記憶をOSの基本機能として実現し，OS自身も分散仮想記憶を利用して構成している．現在までに，分散仮想記憶上で動作するDM-1のシステムタスクとしてスケジューラ，タスクマネージャ，メモリオブジェクトマネージャを実現した．

An Implementation of the Operating System DM-1 Based on Distributed Virtual Memory.

Takuji Shinohara†, Kenji Fujikawa†, Eiji Okubo††, Takao Tsuda†

† Department of Information Science, Faculty of Engineering, Kyoto University
Yoshida hon-machi, Sakyo-ku, Kyoto 606, Japan
†† Department of Computer Science and Systems Engineering,
Faculty of Science and Engineering, Ritsumeikan University
56-1 Tojiin Kita-machi, Kita-ku, Kyoto 603, Japan

DM-1 is an operating system based on distributed virtual memory(DVM). DVM is the unique virtual address space in the system, and consists of all main and secondary storage of all nodes in the network. The DVM provides transparency of memory resources, scalability, and fault transparency to the system.

In DM-1, the DVM is implemented as a primitive function, and is available to the operating system itself. System tasks, e.g., the scheduler, the task manager, and the memory object manager are developed on the DVM.

1 はじめに

ネットワークを介して接続された計算機資源を効率よく管理し利用するために，多くの分散オペレーティングシステム (OS: Operating System) が開発されている [1]．分散 OS では，資源が分散していることによる利点を活かし，かつ分散しているが故に生じる問題を解決しなければならない．すなわち，分散環境に適応したプロセス管理，記憶管理，プロセス間通信，ファイルシステム，高信頼化技法，セキュリティ管理などを実現する必要がある．また，ユーザにネットワークやハードウェアの構成を意識させない種々の透過性の実現も重要である．我々は，OS 機能を分散システム上に実現し，種々の透過性を実現するための手法として，分散仮想記憶に基づく OS の構成法を提案し，DM-1 と呼ぶ OS を設計・開発している [2, 3]．

分散仮想記憶とは，システム上のすべての二次記憶および主記憶からなる単一の仮想記憶である．分散仮想記憶によって，位置透過性，アクセス透過性，実行透過性などが実現される．また，分散仮想記憶によって規模透過性，障害透過性を実現することも可能である．

このような分散仮想記憶を実現することにより，ユーザは分散環境を意識することなく従来のプログラミングモデルを用いて，応用プログラムを記述することができる．また，分散仮想記憶をカーネルレベルで実現することにより，ユーザプログラムのみならず，OS 自身も分散仮想記憶を利用することができる．すなわち，OS 機能の分散実行が容易に実現できる．

2 DM-1 の概要

DM-1 が対象とするシステムは，LAN によって接続された数台から十数台程度の同一アーキテクチャのマシンである．DM-1 は，このような分散システム上にプログラミング環境を提供する汎用 OS である．DM-1 は，各サイトで動作する分散カーネルと，カーネルが提供する分散仮想記憶上で動作するシステムタスクからなっている．

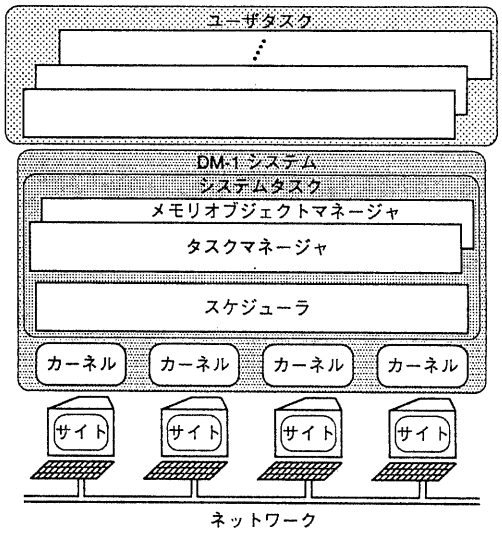

図 1：DM-1 の構成

現在，Intel 80386 マシン，および Ethernet (IEEE 802.3) を使用して実装中である．さらに今後は 486 マシン上で DM-1 システムを構築する予定である．

2.1 DM-1 カーネル

OS を構築する手法として，マイクロカーネルによる手法がある．これは，OS の基本的な機能を実現するカーネルを，プロセスを実現するのに必要な部分だけにし，他の OS の機能をサーバプロセスによって実現しようとする手法である．DM-1 は，マイクロカーネル方式を採用し，さらに分散仮想記憶をカーネルにおいて実現している．

分散カーネルは，サイト間で通信を行い，分散仮想記憶を実現する．分散仮想記憶は，分散環境における単一レベル記憶であり，また，システム上に存在する唯一のアドレス空間である．分散カーネルによってシステム上のすべての二次記憶および主記憶は，分散仮想記憶として統合され，メモリ資源の透過性が実現される．すなわち，分散仮

想記憶へのアクセスは，その実体の位置を意識することなく，任意のサイトから一様に行うことができる．

また，実行されるコードも分散仮想記憶上に配置されることになり，サイトを越えた制御の移動も容易に実現できる．したがって，各サイトの負荷やネットワークの負荷を考慮にいれたスケジューリングを行うことにより，システムの効率的な利用が可能となる [4]．

さらに，カーネルにおいて分散仮想記憶を実現することにより，OS 自身 (サーバプロセス) が分散仮想記憶を利用することが可能となる．

DM-1 では，ページングによる仮想記憶管理を行なっている [2]．主記憶に存在しないページに対するアクセスはページフォルトを発生させ，カーネルによって処理される．カーネルは，ローカルな二次記憶に当該ページが存在した場合は，そこからデータを転送する．サイト内で解決できなかった場合は，他サイトに転送要求を送る．リードアクセスに対しては，並列性を高めるためにページの読み出し専用の複製が作られる．これに対し，ライトアクセスに対しては，他の複製の無効化を行なうことによって，複製間の整合性を保証している．

現在の実装では，分散仮想記憶へのアクセスに対して，Sequential Consistency(SC)[5] を保証している．SC の下では，従来のプログラミングモデルによって記述されたプログラムが正しく動作するため，ユーザにとって望ましいといえる．しかし，整合性の強さとアクセス効率とはトレードオフの関係にあり，整合性に対する制限を緩和することにより，効率的な分散メモリを実現することができる [6, 7, 8]．したがって，効率を考慮した場合，複数の整合性モデルを実現することが必要であると考えている．

同期に関しては，カーネルは，低レベルな同期機構であるサイト間でのロック機構しか提供しない．システムタスクがロック機構を用い，ユーザタスクにさらに機能の高い同期機構を提供する．

2.2 スケジューラ

スケジューラは，最下層に位置するシステムタスクであり，プロセッサの管理を行い，プロセッサの割り当てを行うタスクである．スケジューラは，サイトローカルなタスクとスレッドのスケジューリングの機構を提供する．タスクやスレッドの生成・消滅や大域的なスレッドのスケジューリング方針の決定はより上位のシステムタスクが行なう．

2.3 メモリオブジェクトマネージャ

DM-1 では，メモリ割り当ての単位としてメモリオブジェクトを定義している．メモリオブジェクトは仮想空間上の連続した領域にとられ，互いに交差することはない．すなわち，スタートアドレスとサイズによって決定される．

カーネルは，メモリオブジェクトとして割り当てられた仮想空間に対してのみ実体を生成し，整合性の制御を行なう．したがって，分散仮想記憶を利用するためには，メモリオブジェクトを仮想空間上に生成する必要がある．メモリオブジェクトは生成されると，明示的に消去しないかぎり存在し続ける．すなわち，メモリオブジェクトは，それを生成したタスクの生存期間とは無関係に存在し永続的である．

生成されたメモリオブジェクトはすべて，メモリオブジェクトマネージャによって管理される．メモリオブジェクトには，UNIX のファイルと同様に所有者が存在し，所有者，グループ，その他のユーザに対するアクセス権が設定される．

メモリオブジェクトに対するアクセスは，それにアクセスしようとするタスクにメモリオブジェクトが割り当てられて初めて可能となる．メモリオブジェクトの割り当ての際には，正当性の検査が行なわれる．

メモリオブジェクトマネージャは，他タスクの要求に従って，メモリオブジェクトの生成，消去，属性の変更，タスクへの割り当て等を行う．

2.4 タスクマネージャ

DM-1では，タスク・スレッドモデルを採用している．タスクとはプロセッサ以外の資源の割り当ての単位であり，スレッドとはプロセッサの割り当ての単位である．例えば，ある処理を行おうとした場合，その処理に必要な資源はタスクに割り当てられ，その資源を用いて，スレッドが実際の処理を行う．タスク内には複数のスレッドが存在することができ，処理を並列に行うことができる．

DM-1では，全てのタスクは分散仮想記憶上の資源を利用するため，スレッドの実行は分散透過に行われる．したがって，1つの処理を複数のサイトで分散して実行することが容易に行え，分散システムの利点が活かされる．

タスクマネージャの機能は，タスク・スレッドの生成や消滅に関するものと，タスク間通信に関するものに分けることができる．タスク・スレッドの生成や消滅に関しては，コードの共有に伴う静的変数の扱い以外は，従来のOSで採られてきた管理が行われる．タスク間通信に関しては，分散仮想記憶上で実行される利点を活かした新しい通信手法であるタスク間手続き呼び出し (ITPC: Inter-Task Procedure Call) を採用している．

2.5 スレッド分配機構

分散システムにおいて，大域的なスケジュールを行う際に特に問題となるのが，負荷の分散および通信コストの最小化である．これらの2つの要因は，互いに相容れない面を持っており，バランスをとることが非常に困難である．

特に，分散仮想記憶を用いているDM-1では，サイト間でスラッシングを起こす可能性があり，分散仮想記憶上で効率的な処理を行うためには，この問題を解決することが必要不可欠である．

DM-1では，スレッドおよびメモリオブジェクトの配置に関する方針の決定を行うシステムサーバとして，スレッド分配機構を用意している．スレッド分配機構は，ページフォルトの発生状況やプロセッサの負荷を監視し，動的な配置の方針を決定する．

3 DM-1におけるタスク管理方式

分散仮想記憶によるメモリ資源の透過性の実現が，DM-1の最大の特徴であり，他の分散OSと最も異なる点である．分散仮想記憶機構により，タスクはメモリ資源に対してその位置を全く意識することなくアクセスすることが可能である．また，ユーザはタスクの位置を意識することなくタスク操作を行なうことが可能である．このようにDM-1上のタスクは，ネットワーク透過に存在する．メモリ資源やタスクのネットワーク透過性により，タスク間の資源の共有やタスク間通信は分散仮想記憶の上で容易に実現される．

3.1 タスクとスレッド

従来のOSでは，プログラムを実行する単位をプロセスと呼んでいる．UNIXのプロセスは，プロセスごとに独立したアドレス空間および制御情報を持っており，プロセスを生成する場合に必要とされる処理は大変重い [12]．プロセスを複数用いて並列処理を行なうことは，プロセス同士が密接なつながりを持っている場合には，効率の良いものではない．

そこでMachなどの並列OSでは，プロセスをその主要な機能である，資源管理を行う部分とプログラムの実行を行う部分とに分離し，それぞれをタスク，スレッドと呼んでいる [9]．Machでは，スレッドはプログラム実行の単位であり，1つのタスクのコンテクストの中で処理が行なわれる．また同一タスク内で，複数のスレッドの実行が可能である．スレッドの生成はプロセスの生成に比べて処理が軽く，スレッドを複数用いることによりオーバヘッドの少ない並列処理が行なえる．スレッドはプロセッサにより並行に処理が行なわれるが，さらにマルチプロセッサの場合はスレッドごとに異なるプロセッサを割り当てることが可能となる．

DM-1においても，資源割り当ての単位としてタスクを用い，プロセッサ割り当ての単位として

スレッドを用いている．ここで重要なのは，Machではタスクはサイト固有のものであるのに対して，DM-1 ではタスクはネットワーク透過に存在するものであるということである．したがって，DM-1では 1 つのタスクの処理に，ネットワーク上の複数のプロセッサを使用することが可能となる．

3.2 プログラムの実行管理方式

DM-1 におけるメモリ資源の利用の単位は，メモリオブジェクトを基本としている．メモリオブジェクトの種類としては，コード領域・データ領域・スタック領域などがある．タスクは，1つ以上のコード領域，1つ以上のデータ領域，スレッドの個数分のスタック領域を資源として割り当てられる．

コードは，基本的にタスク間で共有されることを前提に記述しなければならない．コードを共有することにより，メモリ資源の有効利用が行なえる．DM-1 では，従来の OS でライブラリと呼ばれているコードも，仮想記憶空間上のメモリオブジェクトとして実現されている．よってライブラリをタスク間で共有することが可能であり，一般にいうシェアードライブラリの機構を実現している．

3.3 タスク間通信

従来の並列/分散 OS において用いられているタスク間通信方式として，メッセージパッシング方式，共有メモリを用いた方式，さらにそれらの通信方式を利用したリモートプロシジャコール (RPC: Remote Procedure Call) 方式などがある [1]．DM-1 では共有メモリを用いた方式と，タスク間手続き呼び出しと呼ぶ RPC に近い方式とを実現している．

3.3.1 共有メモリによるタスク間通信

共有メモリに基づいた方式では，通信し合うスレッドが変数を共有する．同一タスク内のスレッド間では，メモリ資源へのアクセス権が同じであるため，特別な処理を施すことなく変数の共有が可能である．また，それぞれ異なるタスクに所属するスレッド間においても，タスク間でメモリオブジェクトを共有することにより，共有変数が実現される．

共有変数を使用する通信では，複数のスレッドが同時に 1 つの変数をアクセスするため，スレッド間の排他制御機構が必要となる．タスクマネージャはセマフォによる排他制御の機構を提供している．

3.3.2 タスク間手続き呼び出し

DM-1 では，共有メモリによる通信を利用し，RPC に近い方式である ITPC を提供している．

RPC は，異なるタスクの手続きを，タスクが異なるサイトに存在する場合も含め，同一プログラム中に定義された手続きと同様に呼び出すための機構を提供する．ITPC においても，異なるタスクの手続きを呼び出すという点は RPC と同様である．しかし通常 RPC では，手続き呼び出しのメッセージパッシングへの変換が必要であるのに対して，ITPC ではそれを必要としない．また DM-1 では，タスクはネットワーク透過に存在するので，リモートという概念は必要ない．この 2 点で ITPC は RPC と異なっている．

ITPC では，呼び出される側のタスクはあらかじめ ITPC 可能な手続きのアドレスをタスクマネージャに登録しておき，他のタスクの手続きの指定はそのアドレスを指定することによって行なう．これは，DM-1 ではネットワーク上で単一の記憶空間が形成されていることにより可能となっている．指定されたアドレスがそのタスクが実行の許可を与えられているアドレスではない場合，記憶保護例外となりタスクマネージャに処理が移行する．タスクマネージャは，指定されたアドレスから始まる手続きを実行する許可を与えられているタスクの中に新たなスレッドを走らせる．新たに起動されたスレッドが指定された手続きを実行する．その間，手続き呼び出しを行なったスレッドは，呼び出されたスレッドが終了するまで待ち状態に入る．引数の受け渡しは，2 つのスレッド間でスタックを共有することによって行なわれる．呼び出さ

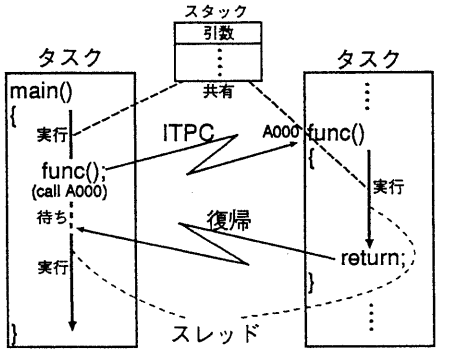

図 2 : ITPC の実行

れたスレッドは値を返すことにより処理を終了し，呼び出しを行なったスレッドはその値を受け取り，処理を再開する (図 2 参照).

ユーザが ITPC を利用する際は，通常の手続き呼び出しを行う場合と同様なプログラムを記述すればよい．ITPC は，プログラムのコード自体も通常の手続き呼び出しとなんら変わりがない．

ITPC は，タスクマネージャおよびユーザタスクに処理を依頼する場合や，ユーザタスク間で通信を行なう場合に用いられる．

3.4 タスクの構成例

DM-1 でのタスクの構成例として，2 つのスレッドを使用した場合の行列積を計算するプログラムを図 3 に示す．スレッドはそれぞれ異なるサイトのプロセッサ上で処理が行なわれる．DM-1 では複数サイトのプロセッサを使用するタスクをこのように容易に記述することが可能である．

プログラム中で行列 A, B は適当な値に初期化されており，行列積 AB の値を行列 C に代入する．行列の各要素は 32 ビットの符号付き整数で，行列のサイズは各行・各列が 32, 64, 128, 256, 512 の場合で測定を行なった．サイトごとの，データ領域として使用可能な主記憶容量はおよそ 2Mbytes であり，それよりも大きなサイズのデータを扱う計算では 2 次記憶がスワップエリアとして使用さ

```
if (tm_fork() == 0) {
  for (i = 0; i < N/2; i++)
    for (j = 0; j < N; j++) {
      c[i][j] = 0;
      for (k = 0; k < N; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
} else {
  for (i = N/2; i < N; i++)
    for (j = 0; j < N; j++) {
      c[i][j] = 0;
      for (k = 0; k < N; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
}
```

図 3 : 行列積を求めるプログラム

れる．

図 4 に，1 つのスレッドで行列積の演算を行なった場合との比較を行なったグラフを示す．x 軸は行列の大きさであり，y 軸は単位時間当たりの演算回数を示している．図 4 より，2 つのスレッドの場合の単位時間あたりの演算回数は，1 サイト 1 スレッドの場合の，最大で 1.76 倍になっていることが分かる．

4 スレッド分配機構

前章において，行列積の演算を行なう場合，複数のスレッドを使用することによって効果的なタスクが構成できることを示した．しかし行列積の演算は複数スレッドに分割しやすい問題であると言えるのでこのように良好な結果が得られたが，一般のアルゴリズムでは必ずしも効果的な複数スレッドへの分解が行なえるとは限らない．タスク中で複数スレッドを使用したため，かえって処理速度が低下してしまうということも十分にあり得る．スレッド同士がデータを共有している場合にこの問題が生じる可能性がある．この問題を解決するために，DM-1 では，通信量の抑制とサイトごとの負荷の均衡化を同時に考慮に入れた動的なスレッド

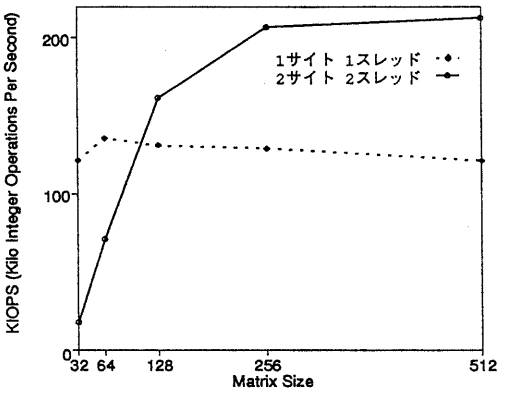

図 4：行列積の計算に要した時間

分配機構を提案している．

DM-1 では，仮想空間の整合性はページ単位で管理されている．ネットワーク間で頻繁にページを移動させることはコストの点で望ましくない．しかし，同一のメモリ領域をアクセスする 2 つのスレッドが異るサイトに割り付けられると，ページスラッシングによる性能低下を起こす可能性が十分考えられる．一方，通信量を抑制することのみを考慮してスレッド割付を行なうと，スレッドが特定のサイトに集中する傾向があり，負荷分散の点で望ましくない．従来プロセス移送などの手段による負荷均衡を図る研究はいくつかある [13, 14, 15] が，通信量と負荷均衡を同時に考慮している研究は少ない．

4.1 スレッド分配のアルゴリズム

DM-1 では，負荷均衡化と通信量の抑制という相反する 2 つの要素を考慮に入れたスレッド分配機構を考案した．スレッド分配機構のアルゴリズムの概要を以下に示す．

- キャパシタと呼ぶ変数型を定義する．この変数は，値が時間とともに指数関数的に減少する変数である．ただし，イベント (イベントは使用目的によって異なるものを指す) が発生した時に，値が増加する．
- スレッドごとに，そのスレッドが発生させるページフォールトをイベントとしたキャパシタ型変数を用意する．この変数は，スレッドが頻繁にページフォールトを発生させる場合に大きな値をとる．
- スレッドごとに，それぞれのサイトに対応するキャパシタ型変数をサイトの数だけ用意する．イベントはやはりスレッドのページフォールトであるが，最新ページを持つサイトに対応したキャパシタ型変数にのみ値を加える．すなわちこの変数は，スレッドとサイトとの関連度を表す．
- 一定期間ごとに上に述べた 2 つの変数の値を更新し，それによりスレッドを他のサイトに移送するかどうかを決定する．

現在このようなアルゴリズムをもとに，UNIX 上でシミュレーションを行ない，評価を行なっている．

5 信頼性向上と規模透過性

DM-1 では，システムの性能の向上をはかるため，分散仮想記憶の整合性制御および動的なスレッド分配を行なっている．システム性能の向上は，分散システムを構築する上での目標の 1 つであるが，その他にもいくつかの達成すべき目標がある．その中に，システムの信頼性の向上ということがあげられる．DM-1 ではシステムの信頼性向上のため，分散仮想記憶を積極的に利用することを考えている．

具体的な信頼性向上の手法として，ネットワーク上のあるサイトに障害が発生した時にユーザにはその障害を見せないという障害回復機能を実現する．分散仮想記憶はページごとに管理されている．したがって障害回復の基本であるレプリケーションもページごとに行なう．複製ページは必ず異なるサイトに置かれ，サイトに障害が発生した場合，他のサイトが複製ページを基に処理を継続して行なう．

さらに DM-1 では，障害が発生したサイトが修復されネットワークに接続された場合と，新たにサイトが追加される場合とを同様に扱うことにより規模透過性を実現する．

6 おわりに

本稿では，分散仮想記憶に基づいた分散OSの構成を提案し，DM-1における実現について述べた．本稿で述べたOS機能の他，ファイルシステムやデバイスの扱いに関する考察や，応用例に関する考察が今後の課題である．

謝辞

日ごろより御討論を頂く津田研究室の皆様に深謝いたします．

参考文献

[1] 前川守, 所真理雄, 清水謙太郎, "分散オペレーティングシステム UNIX の次に来るもの", 共立出版 (1991).

[2] 篠原拓嗣, "分散オペレーティングシステム DM-1 における分散仮想記憶の実現", 京都大学工学部情報工学科特別研究報告書 (1992).

[3] 藤川 賢治, "分散オペレーティングシステム DM-1 におけるタスク管理方式", 京都大学工学部情報工学科特別研究報告書 (1993).

[4] ン ペク ヨン, "分散オペレーティングシステム DM-1 におけるスレッド分配機構の実現", 京都大学工学部情報工学科特別研究報告書 (1993).

[5] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs", IEEE Trans. on Computers, C-28(9), pp. 690–691 (1979).

[6] D. Mosberger, "Memory Consistency Models", ACM Operating System Review, 27(1) (1993).

[7] S. Adve and M. Hill, "Weak ordering: A new definition", In Proc. of the 17th Annual ISCA, pp. 2–14 (1990).

[8] K. Gharachorloo, A. Gupta, and J. Hennessy, "Performance evaluation of memory consistency models for shared memory multiprocessors", ACM SIGPLAN Notices, 26(4), pp. 245–257 (1991).

[9] A. Tevanian Jr., R. F. Rashid, D. B. Golub, David L. Black, Eric Cooper and Michael W. Young, "Mach Threads and Unix Kernel: The Battle for Control", CMU-CS-87-149 (1987-8).

[10] M. Gien, "From Operating System to Cooperative Operating" Environments, CS/TR-92-82, Chorus systemes, 6, avenue Gustave Eiffel, F-78182 ST QUENTIN EN Y. CEDEX, France (1992-10).

[11] A. S. Tanenbaum, "The Amoeba Operating System", Vrije Universiteit, De Boelelaan 1081a, Amsetrdam, The Netherlands (1992).

[12] M. J. Bach 著 ; 坂本 文, 多田 好克, 村井 純 訳, "UNIX カーネルの設計", 共立出版 (1991).

[13] R. M. Byrant, R. A. Finkel, "A Stable Distributed Scheduling Algorithm", Proceedings of the 2nd International Conference on Distributed Computing Systems, pp. 314–323 (1981).

[14] L. M.Ni, C. W. Xu, T. B. Gendreau, "Drafting Algorithm - A Dynamic Process Migration Protocol for Distributed System", Conference on Distributed Computing Systems (Cat. No. 85CH2149-3), pp. 539–546 (1985).

[15] W. Osser, "Automatic Process Selection for Load Balancing", Technical report ucsc-crl-92-21, University of California Santa Cruz(1992-6).