オペレーティング・システム　51－5
（１９９１．　７．　１８）

処理形態に適したプロセスの軽量化

横山　和俊　　箱守　聰　　谷口　秀夫
ＮＴＴデータ通信（株）　開発本部

プロセスは各種の処理を実行する実体であることから、プロセスの効率的な実現はシステム全体の実行性能に大きな影響を与える。プロセスに対する基本操作を小さいオーバヘッドで実行するためには、プロセス間で構成要素を共有し、プロセスを軽量化することが重要となる。本稿では、分散型リアルタイムオペレーティングシステム(ＤＩＲＯＳ：DIstributed Real-time Operating System)上におけるプロセスの軽量化について報告する。具体的には、プロセス構成要素の共有範囲（論理空間とデータ部と資源アクセス）が異なる３つの軽いプロセスについて、その実現方式と性能評価について述べる。また、得られた結果に基づき、軽量化の各形態に適した適用分野を考察する。

Light Weight Process Adapted to Application Processing

Kazutoshi YOKOYAMA　Satoshi HAKOMORI　and　Hideo TANIGUCHI
Development Headquaters
NTT DATA COMMUNICATIONS SYSTEMS CORPORATION
Kowa Kawasaki Nishi Bldg. 66-2 Horikawa-cho, Saiwai-ku,
Kawasaki-shi, Kanagawa 210, Japan
JUNET : yokoyama@rd.nttdata.jp

In a multi-process environment, implementation of the process effects system performance. In order to reduce the overhead of controlling processes on the operating system, it is important to support the light weight process facility. In this paper, the light weight process facility on DIROS (DIstributed Real-time Operating System) is described. Three types of light weight processes have been implemented on DIROS. They are evaluated from several view points (e.g. creation, dispatching). Furthermore, application processing for each type of the light weight processes is considered.

1．はじめに

現在、多くのシステムで実現されているマルチプロセス環境では、オペレーティングシステム（以下、ＯＳと略す）が走行するプロセスを切り換えながら、各プロセスの処理を進めている。プロセスは各種の処理を実行する実体であることから、プロセスの効率的な実現はシステム全体の性能に大きな影響を与える。従来のＯＳでは、各プロセスがプロセスの構成要素(論理空間やテキスト部など)を個別に持っており、プロセス切り換え時にＯＳがそれらも同時に切り換える必要がある。そのため、例えば、複数プロセスを並列に実行する場合、プロセス切り換えのオーバヘッドやプロセス間の同期などが大きいことが指摘されている。これらの問題点を解決するために、プロセス間で構成要素を共有し、プロセスに対する基本操作をできるだけ少ないオーバヘッドで実行する、軽いプロセスに関する研究が盛んに行なわれている[1]。

筆者らは、軽いプロセスの実現方式を検討し、構成要素の共有の範囲が異なる様々な実現方式を考察した[2]。また、その検討に基づき、分散型リアルタイムオペレーティングシステムであるＤＩＲＯＳ（Distributed Real-time Operating System）[3]上に種々のプロセスを実現し、評価している。本稿では、既に実現している３つの軽いプロセスの実現方式について報告する。また、ＤＩＲＯＳ上のの従来プロセスを含めた４種類のプロセスに対して行なった評価実験を通して、処理形態に適したプロセスの軽量化について考察を行なう。

2．プロセスの軽量化

2.1　プロセスの構成要素

プロセスが走行するために必要と考えられるプロセスの構成要素を以下に示す[2]。

（１）論理空間を実現するメモリ管理テーブル
（２）テキスト部、データ部、スタック部からなるプロセスイメージ
（３）ファイルなどの資源へのアクセスを管理する資源アクセス管理テーブル
（４）レジスタ

これらの構成要素のうち、共有可能なものとして、テキスト部、データ部、メモリ管理テーブル（論理空間）、資源アクセス管理テーブルが考えられる。他の構成要素は、プロセスが個別に所有する必要がある。

共有可能な構成要素をプロセス間で共有した場合の長所と短所を以下に示す。

（１）論理空間（メモリ管理テーブル）
同一論理空間上に複数のプロセスが存在するため、プロセス切り換え時のオーバヘッドを軽減できる。また、プロセス切り換え時にキャッシュやＴＬＢをフラッシュしないため、ヒット率が向上する。短所としては、プロセス固有の領域への不当アクセスを防ぐことが困難となる。
（２）テキスト部
メモリの効率的な利用ができる。
（３）データ部
共有データを用いた高速なプロセス間通信が可能となる。しかしながら、１つのデータを占有する処理が複雑化する。
（４）資源アクセス管理テーブル
複数のプロセスが協調して１つの資源に対する操作を行なうときなどに有効であると考えられる。しかしながら、１つの資源を占有して使用する場合、排他制御を行なうための機構を提供する必要があり、処理が複雑化する。

上で述べたように、構成要素の共有は、プロセスの実行性能に大きく影響を与えると考えられる。そのため、構成要素を共有する範囲が性能に与える影響を明確にし、軽いプロセスと処理形態の関係を明確化することは重要である。

2.2　実現した軽いプロセス

軽いプロセスの実現方式は、構成要素の共有の範囲によって様々なものが考えられる。今回の報告では、構成要素のうち、データ部と資源アクセス管理テーブルが共有された時とそうでない時の性能の違いを中心に評価を行なう。そのため、ＤＩＲＯＳ上に３つの軽いプロセスを実現した。それぞれ、アクセス共有プロセス、空間共有型アクセス非共有プロセス、空間共有型アクセス共有プロセスと呼ぶ。また、比較のため、ＤＩＲＯＳの従来プロセスについても説明する。プロセス間で共有されている構成要素を図１に示す。

＜タイプＡ＞　従来プロセス
同じプログラムファイルを利用したプロセスが存在している場合、テキスト部を共有する。
＜タイプＢ＞　アクセス共有プロセス

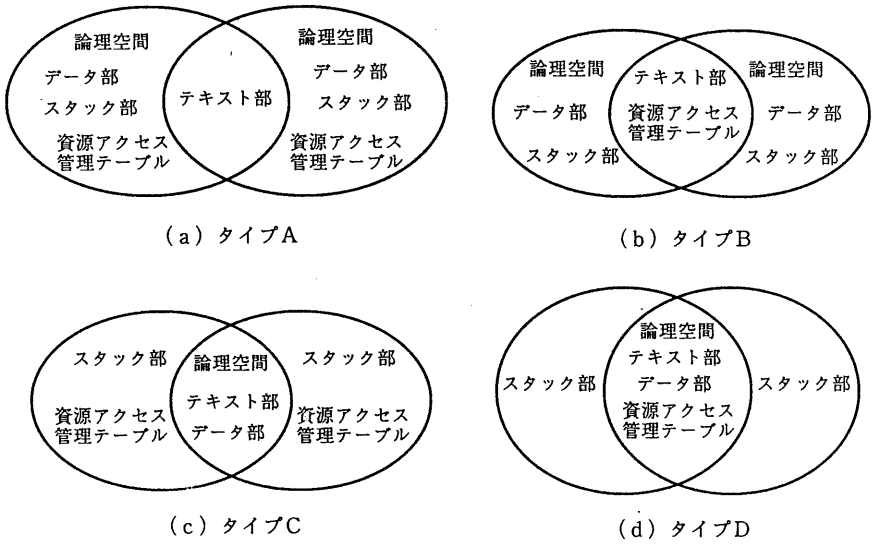


図1　プロセス間で共有された構成要素

従来プロセスと同様に同じプログラムを利用しているプロセスが存在している場合、テキスト部を共有する。さらに、資源アクセス管理テーブルも共有する。

＜タイプC＞　空間共有型アクセス非共有プロセス
テキスト部、データ部、及び論理空間を共有する。資源アクセス管理テーブルは共有していない。

＜タイプD＞　空間共有型アクセス共有プロセス
テキスト部、データ部、論理空間、及び資源アクセス管理テーブルを共有する。このタイプはＭａｃｈ[1]に見られるスレッドと呼ばれる軽量化されたプロセスと同等な構成要素共有を行なっている。

実現した軽いプロセスは、ＤＩＲＯＳのシステムコールを従来のプロセスと同様に利用できる。例えば、メッセージの送受信はプロセス単位で既存のシステムコールを用いて行なうことができる。

3．軽いプロセスの実現方式

ここでは、ＤＩＲＯＳ上での軽いプロセスの実現方式について説明する。

3.1　プロセスの管理

ＤＩＲＯＳで走行するプロセスはＰＣＢ（Process Control Block）によってその走行状態を保持する。ＰＣＢは、主に以下の情報を保持する。

（1）プロセスの走行優先度
（2）プロセスの使用するメモリ情報（メモリ管理テーブルへのポインタ）
（3）プロセスの論理空間情報（テキスト部、データ部、スタック部などの論理アドレス情報）
（4）プロセスが実行するプログラム情報
（5）プロセスの資源アクセス情報（資源アクセス管理テーブルへのポインタ）

3.2　プロセスの生成処理

各プロセスの生成処理の概要を図2に示す。タイプＡとタイプＢの生成処理は、親プロセスと同じプログラムだけでなく、別プログラムを実行するプロセス生成も考慮した処理である。このとき、既に同じプログラムを利用しているプロセスが存在すれば、そのプロセスとテキスト部の共有を行なう。一方、タイプＣとタイプＤでは、データ部を共有しているため親プロセスと必ずテキスト部を共有し、親プロセスの実行状態を子プロセスにコピーしなければならない。各構成要素の共有／非共有による処理の項目について以下に説明する。

（1）論理空間
共有しない場合は、プロセスの使用するメモリ管理テーブルの初期化を行なう。共有する場合は、ＰＣＢ内のメモリ管理テーブルへのポインタをコピーす

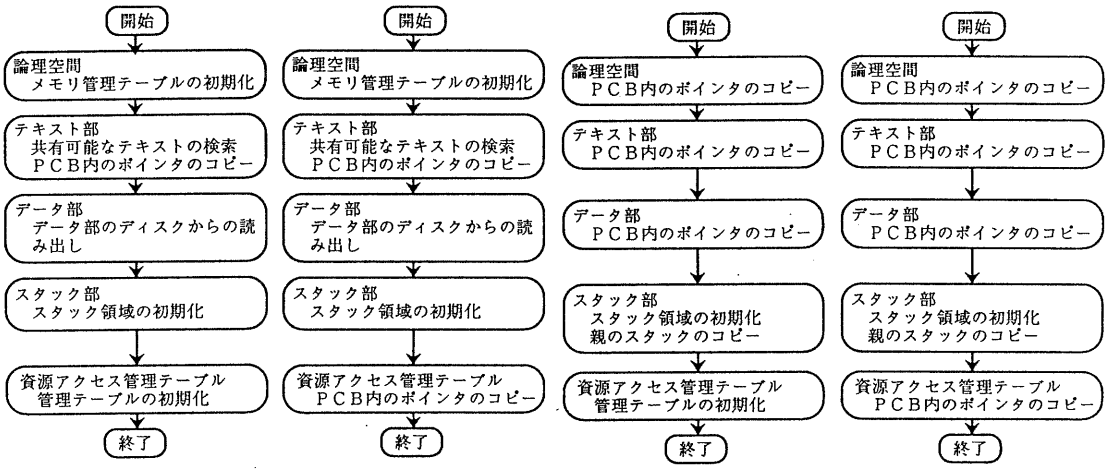


図２　プロセスの生成処理の概要

る。

（２）テキスト部

［論理空間非共有の場合；タイプAやタイプB］

同じプログラムを使用しているプロセスを検索し、もし存在すれば、そのプロセスとテキスト部の共有を行なう。これは、共有するプロセスのテキスト部の情報をコピーすることで行なう。

［論理空間共有の場合；タイプCやタイプD］

親プロセスのテキスト部の情報をコピーする。

（３）データ部

共有しない場合は、データ部領域の確保と共に、データ部の内容をディスクから読み込む。共有する場合は、ＰＣＢ内のデータ部に関する情報をコピーする。

（４）スタック部

４タイプともスタック部は共有できない。タイプAとタイプBの生成処理は、プログラムの初期化実行なのでスタック部の領域を確保し初期化を行なう。タイプCとタイプDでは、親プロセスのプログラムを途中から実行するため、親プロセスのスタックをコピーする必要がある。

（５）資源アクセス管理テーブル

共有しない場合は、資源アクセステーブルの初期化を行なう。共有する場合は、ＰＣＢ内の資源アクセス管理テーブルへのポインタをコピーする。

３．３　プロセスの終了処理

プロセスの終了処理では、使用している各構成要素の解放を順次行なう。これらの処理は、いづれのタイプも同じ流れある。ただし、他のプロセスと共有している構成要素がある場合、その構成要素の解放処理は行なわない。

３．４　プロセス切り換え処理機構

プロセス切り換え処理では、主に以下の処理を行なう必要がある。

（１）走行プロセスの切り換え

（２）論理空間の切り換え

（３）資源アクセス情報の切り換え

（４）実行するプログラムの切り換え

これらの処理は、プロセスの特長に合わせて、切り換えるものを限定する処理を行なうべきである。しかしながら、今回は４つのタイプとも同じ切り換え処理を行なっている。

ＤＩＲＯＳの切り換え処理機構は次のような特長を持つ。

（１）固定優先度のスケジューリング方式

（２）各優先度ごとのＦＩＦＯ処理

これらにより、プロセス数に関係なくほぼ一定の処理時間での切り換えを可能にしている。

4．性能評価

4.1　項目

実現した軽いプロセスに対し、以下の項目について性能評価を行う。資源アクセス管理テーブルに関しては、具体的にファイルアクセス管理テーブルを対象とする。

（1）プロセスの生成終了処理
（2）プロセスの切り換え処理
（3）ファイルを共有した協調処理

各項目の実測は、他のプロセスが走行していない状態で行う。測定に用いたタイマの精度は１０(msec)であり、最大測定誤差は２０(msec)である。このタイマの測定精度に測定結果が影響を受けないように、各処理を１０００回以上ループさせ測定する。

4.2　プロセスの生成終了処理

プロセスの生成に関して、各構成要素の共有／非共有が生成終了時間に与える影響を評価する。

4.2.1　処理内容に基づく考察

図2に示すように各プロセスの生成処理は異なる。また、3.3節では、終了処理はほぼ同じであることを述べた。これらのことから、各構成要素の共有／非共有と各プロセス生成終了時間の関係について以下のことが推測される。

（1）テキスト部

タイプCやタイプDに比べ、タイプAやタイプBは共有可能なテキストの検索処理が必要である。この処理の時間は、同時に生成されているプロセスが利用しているプログラムファイル数に比例する。これは、メモリ上の処理であるため、影響は小さい。

（2）データ部

タイプCやタイプDに比べ、タイプAとタイプBはディスクアクセスを伴うデータ部の読み出し処理が必要である。この処理の時間は、データ部の大きさに影響される。これは、メモリ上の処理ではないため処理時間に与える影響は大きい。

（3）スタック部

タイプAやタイプBに比べ、タイプCやタイプDはスタックの内容を親プロセスからコピーする必要がある。この処理時間は、その時利用しているスタックの大きさに比例する。これは、メモリ上の処理であるため影響は小さい。

（4）資源アクセス管理テーブル

タイプBやタイプDに比べ、タイプAやタイプCの資源アクセス管理テーブル初期化処理は処理量が多い。これは、メモリ上の処理であるため、影響は小さい。

4.2.2　実験結果

プロセスの生成終了時間の測定プログラムの概要を図3に示す。プロセスの生成と終了を繰り返して行ない、その開始時刻と終了時刻を取得している。

その実測結果を表1に示す。表1では、各プロセスの生成終了時間を、タイプAの生成終了時間を1としたときの相対値で表わしている。表1に示す結果より、各構成要素の共有／非共有とプロセス生成終了処理の時間との関係について、以下のことがわかる。

（1）タイプAとタイプCの差、あるいはタイプBとタイプDの差が大きい。測定プログラムのみが走行する環境であり、かつ測定プログラムが小規模なため利用スタックが小さいことから、この差はデータ部をディスクから読み込む処理に起因すると考えられる。

（2）タイプAとタイプBの差、あるいはタイプCとタイプBの差は、ほとんどない。したがって、ファ

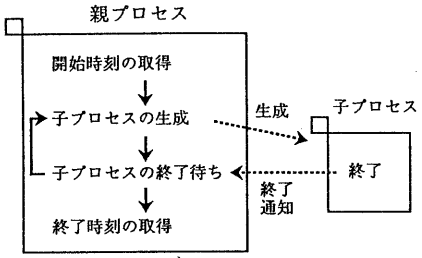


図3　プロセスの生成終了処理時間の測定プログラム

表1　プロセスの生成終了処理時間

	タイプA	タイプB	タイプC	タイプD
生成終了時間(相対値)	1	0.999	0.045	0.045

イルの資源アクセス管理テーブルの共有／非共有は、プロセス生成終了処理の時間にはほとんど影響を与えない。

4.3 プロセス切り換え処理

各プロセスについて、プロセス切り換え処理時間を測定した。

図4に測定プログラムの概要を示す。また、得られた結果を図5に示す。図5において、プロセス切り換えの処理時間は4つのプロセスともほとんど変わらない。また、DIROSの切り換え処理機構をそのまま利用しているため、プロセス数に関係なく処理時間は一定である。

今回の測定では、切り換え後に応用プログラムの走行に影響を与えるTLBのヒット率の評価はできていない。TLBのヒット率が与える影響を評価するためには、切り換え処理に、「現走行プロセスと次走行プロセスが使用する論理空間が同一であるか否かを判定し、同一である場合には論理空間を切り換えない」という機能を加える必要がある。これにより、切り換え処理機構に次のような影響がでる。

（1）切り換え処理機構自体が複雑化する。しかしながら、複雑化による処理量の増加は、切り換え処理全体と比較すると、非常に小さいと考えられる。

（2）プロセス間で論理空間を共有する場合、プロセス切り換え時にTLBのフラッシュを行なわないためヒット率が向上する。

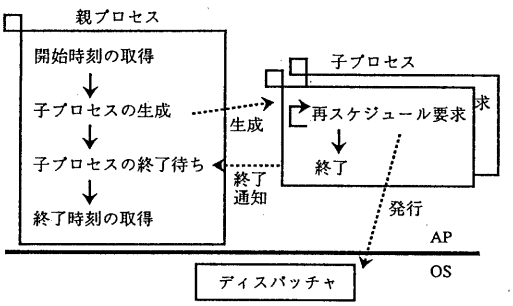


図4 プロセス切り換え処理時間の測定プログラム

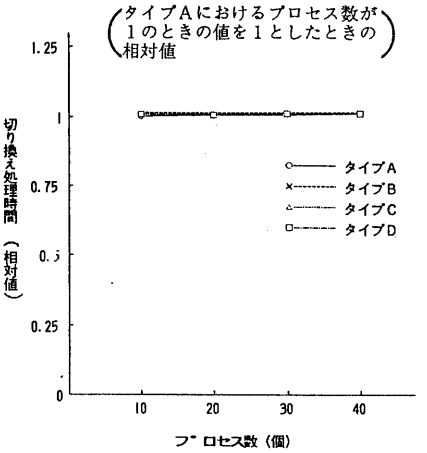


図5 プロセス切り換え処理時間

4.4 ファイルを用いた協調処理

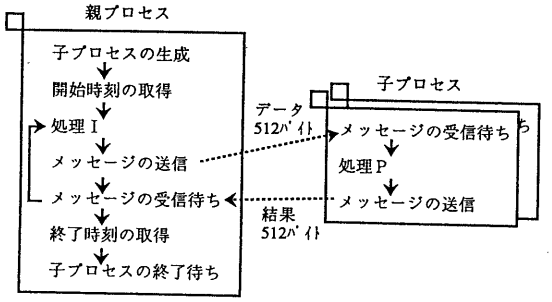
4.4.1 協調処理の形態

複数の軽いプロセスが1つのファイル資源を共有した協調処理に関して、プロセス間の同期、及びプロセス間通信のオーバヘッドを測定する。ここでは、ファイルをディスクから読み出し（処理I）、そのデータに基づいての処理（処理P）を行なう処理を考える。1つのファイル資源を用いた協調処理を行なう手段として、（1）複数プロセスによる並列処理と（2）非完了システムコール機能[3]を用いる方法がある。さらに、前者の方法では、構成要素の共有によって、協調処理の形態が大きく異なる。各プロセスに応じた処理内容を図6に示し、以下に説明する。

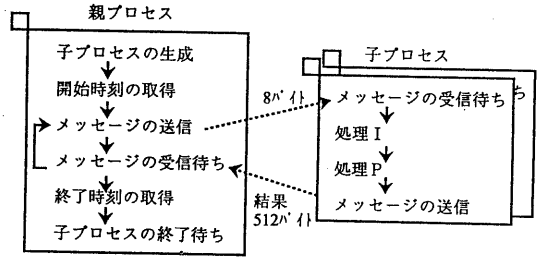
（1）複数プロセスによる並列処理

（a）タイプAでは、ファイルアクセス管理テーブルを共有していないため、ファイルアクセス情報をプロセス間で共有できない。そのため、資源へのアクセス情報を管理するプロセス（この場合、親プロセス）が必要である。親プロセスがファイルを読み出し、その内容を子プロセスにメッセージ通信で送信する。メッセージを受け取った子プロセスは、その内容を処理した後、結果を親プロセスにメッセージ通信で送り返す。

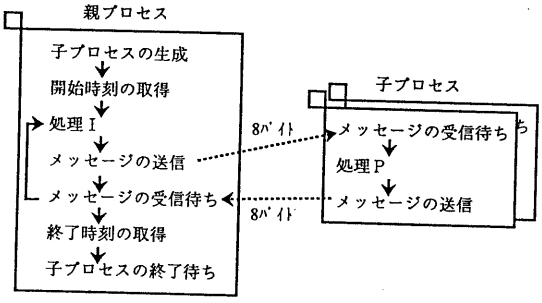
（b）タイプBでは、ファイルアクセス管理テーブルを共有しているため、複数のプロセスがアクセス情報を共有できる。そのため、各プロセスがアクセス



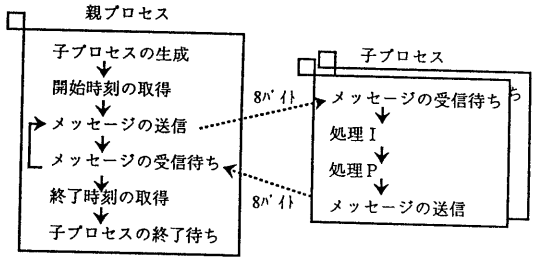
（ａ）タイプＡでの協調処理



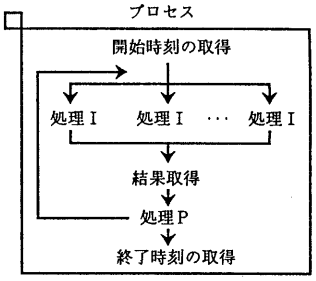
（ｂ）タイプＢでの協調処理



（ｃ）タイプＣでの協調処理



（ｄ）タイプＤでの協調処理



（ｅ）非完了システムコール機能を用いた
協調処理

図６　協調処理の処理内容

情報を参照、更新しながら、処理を行なう。

（ｃ）タイプＣでは、タイプＡと同様に、ファイルアクセス管理テーブルを共有していないため、親プロセスがファイルを読み出し、子プロセスへ転送する必要がある。タイプＡとの違いは、データ部を共有しているためプロセス間の通信がデータ部のアドレスを知らせるだけですみ、コピーが発生しないことである。

（ｄ）タイプＤでは、タイプＢと同じく、ファイルアクセス管理テーブルを共有しているため、各プロセスが読み出し処理を行なうことができる。

（２）非完了システムコール機能

同時にＯＳへ複数の要求を発行できるシステムコールを用いて並列処理を行なう（図６(ｅ)参照）。この並列処理の形態においては、１つのプロセス内でファイルアクセスを並列化できるので処理中にプロセスの切り換えが発生しない。また、プロセス間で同期や通信を行なう必要がないという利点をもつ。

4.4.2　実験結果と考察

1つのファイルを用いた協調処理の評価実験を行なった。測定プログラムは、図6の処理内容に基づいて作成した。また、プログラム中では、以下の方法で協調処理をシミュレートしている。

（1）複数プロセスによる並列処理の場合

処理Ｉ：一定時間プロセスを待ち状態にするシステムコールとする。

処理Ｐ：一定回数のｆｏｒ文とする。

（2）非完了システムコールによる並列処理の場合

処理Ｉ：一定時間後に結果を取得するシステムコールとする。このとき、システムコールを発行したプロセスは結果の返却を待つことなく処理の続行が可能である。

処理Ｐ：一定回数のｆｏｒ文を用いる。

処理Ｉの時間と処理Ｐの時間の比を約8：1としたときの実測結果を図7に示す。

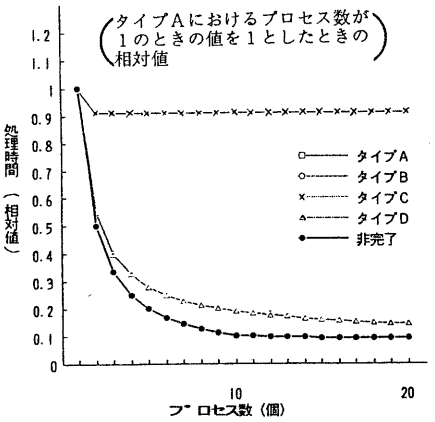


図7　協調処理の実測結果

図7からわかるように、非完了システムコールを用いた場合が最も速く、次いで、タイプD，タイプB，タイプC，タイプAの順である。ただし、タイプAとタイプCの場合は、ほぼ同じ値である。また、タイプBとタイプDの場合はほぼ同じである。これらにより以下のことがわかる。

（1）ファイルアクセス管理テーブルの共有／非共有によって並列処理の内容が変わるため、処理時間に大きく差がでる。

（2）軽いプロセスの同期に必要な時間が支配的である（非完了システムコールとの比較では、処理時間の９０％が同期のためのオーバヘッドであり、残り１０％が切り換え処理などの時間である）。

5．おわりに

本稿では、ＤＩＲＯＳ上に実現した軽いプロセスの実現方式について述べた。また、その性能を幾つかの項目について評価した。結果をまとめると次のようになる。

（1）生成終了処理に関しては、データ部の共有／非共有が最も影響を与える。

（2）切り換え処理に関しては、各プロセスともほぼ同じ処理時間である。

（3）ファイルを用いた協調処理の場合は、単一プロセスでの非完了システムコール機能を用いるのが最も有効である。

今後は、軽いプロセスに適したプロセス切り換え処理機構を実現し、その性能の定性的な評価と定量的な評価を行なう予定である。また、論理空間を共有した軽いプロセスについて、メモリアクセス保護を実現し、評価する。

参考文献

[1] M. Accetta, et. al. : "Mach : A New Kernel Foundation for UNIX Development", Proc. of USENIX 1986 Summer Conference, pp.93-112 (1986).

[2] 谷口，他："プロセスの軽量化に関する検討"，第42回情処学全大4-37 (1991).

[3] 谷口，他："分散型リアルタイムオペレーティングシステム：ＤＩＲＯＳ"，情処研報，89-OS-45-9 (1989).