プロセス・ネットワークによるＯＳの性能について

田胡　和哉　　益田　隆司
（筑波大学）

１．はじめに

近年、ハードウエア技術の発展に伴って、小型、超小型機の普及が著しく、その応用範囲も拡大している。このため、それぞれの機種、応用向きのＯＳ、あるいは、資源を共有するための分散システム用ＯＳがあらたに設計される機会が増加している。その一方において、ＯＳは、設計、実現が容易でないシステムであり、その改善が種々試みられている。

このような観点から、筆者らは、ＯＳの設計、実現の容易化を図る一つの方法として、ＯＳを資源管理機能の集合体としてとらえ、相互排除アクセスされる資源の各々をプロセスで記述し、それらを通信を用いて結合することによりシステムを実現することを試みている[1]。このような、通信で結合されたプロセスの集合をプロセス・ネットワークと名付けた。提案方式の狙いは、各々並行に動作するプロセスの集合によってＯＳを設計することにより、その制御構造を改善するとともにモジュラリティーの改善を図ることにある。実際に、提案方式により、実用ＯＳとして広く用いられているＵＮＩＸ+システムとシステム・コール・レベルで同一の仕様を持つシステムを実現することにより、記述、デバッグ等が容易に行なえることを確認した。しかしながら、提案方式は、システムを構成する機能間の結合が通信によって実現されるために、性能が低下しやすい欠点をもつ。本稿では、実測データをもとに、従来方式との得失について解析する。

提案方式の今一つのねらいは、分散システムへの対応を図ることにある。提案方式は、その構造上、分散システムへの対応が容易である。そこで、さらに、本稿では、通信オーバヘッドと性能の間の関係を明らかにし、提案方式による分散処理ＯＳの性能を推定することを試みる。

２．提案方式の概要

提案方式では、図１に示すように、ＯＳを、通信により結合された、資源管理を行なうプロセスの集合により記述している。このとき、プロセスの切り換え、通信の実行、および、外部割込み、システム・コールの通信への変換は、ＯＳ核と呼ばれるプログラム群が実行する。ユーザ・プロセスの発行するシステム・コールは、通信に変換され、ＯＳを実現するシステム・プロセスに伝達される。このとき、システム・コールを受けつけるためにユーザ・プロセスに一対一に対応して設けるシステム・プロセスを、スーパバイザ・プロセスと名付ける。要求の内容は、順次、より原始的な資源へのアクセスに

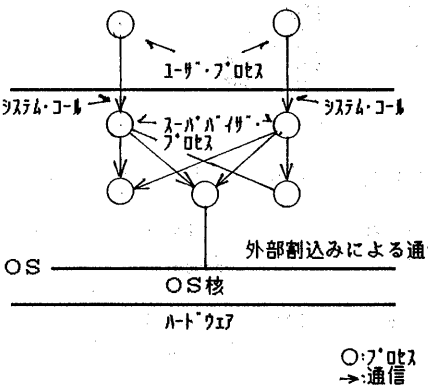


図１　提案方式によるシステムの構造

変換される。たとえば、実現したシステムでは、ファイル入出力を要求するシステム・コールは、通信により、スーパバイザ・プロセス、オープンされたファイルの各々に割付けられるファイル・プロセス、入出力すべきディスク装置の番地を計算するinode プロセス、入出力のバッファリングを行なうバッファ・プロセス、入出力を実行するディスク・プロセスの順に伝達され、最終的に物理入出力が起動されることにより処理される。

+UNIX is a trademark of Bell Laboratories.

実現したシステムは、ＵＮＩＸバージョン６のシステム核と同一仕様のシステム・コールを受けつけ、同一のユーザ・プログラムを実行することができる。通信方式としてランデブを用い、各プロセスの記述はＣ言語を用いた。それは、１５種類のプロセスの、６３個の実体の集合からなっており、ＰＤＰ１１プロセッサ上で動作する。

通信の性能はシステムの性能に与える影響が大きいので、以下で通信機構について詳しく述べる。プロセスは、手続き呼出しの形式で通信の同期に関する処理の実行をＯＳ核に依頼する。この手続きを、核プリミティブと呼ぶ。ＯＳ核内部では、各プリミティブの呼出しに対して、状態の待避、待ち合わせ状況の判定による実行可能なプロセスの選択（スケジューリング）、その状態の回復を行なう。表１に主要な核プリミティブを示す。図２は、プロセス内部の通信の記述の概要である。ここでは、プロセスＡがプロセスＢのＥエントリに対する通信を行なっている。ランデブ呼出し（rendezvous call）は、call核プリミティブを、通信相手のプロセス名(B)、エントリ名(E)、および、プロセス間で転送されるデータ(D_1～D_n)を引数として呼び出すことにより実現される。受けつけ（accept）は、acc プリミティブを受けつけるべきエントリ名(E)を引数として呼び出すことにより実現される。プロセスＡが call プリミティブを発行し、かつ、プロセスＢがacc プリミティブを発行するまでどちらかのプロセスがプリミティブの副作用により停止する。両者が発行されランデブが成立すると、acc は復帰してプロセスＢはランデブを実行し、プロセスＡはプロセスＢがendrプリミティブを発行するまで停止する。プロセス間のデータ転送は、プロセスが実行する。acc プリミティブは、復帰に際してcallプリミティブの引数列へのポインタを値として返す。そこで、プロセスＢは、自らの局所領域へ引数を転送する。

表１　核プリミティブ

名前	機能
call	ﾗﾝﾃﾞﾌﾞ呼び出し
acc	ﾗﾝﾃﾞﾌﾞの受けつけ
endr	ﾗﾝﾃﾞﾌﾞの終了

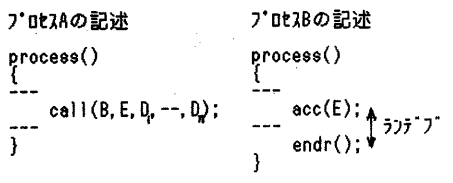


図２　通信の記述

３．性能の予測

実現されたシステムと従来のＵＮＩＸシステムの性能の差異は、制御構造の変化に由来する原理的なものと、特に通信機構の実現方式に由来する機構的なものに分けて考えることができる。

ここで、ＵＮＩＸの制御構造について考えてみる。図３は、ＵＮＩＸの制御構造の概要である。システム・コールは、システム内のユーザ・プロセスに一対一に対応するプロセス中の手続き群により処理される。手続きは再入可能であり、各プロセスは同一の手続き群により動作する。ユーザ間で共有される、ファイル、主記憶領域等の資源は、プロセス間共有変数によっ

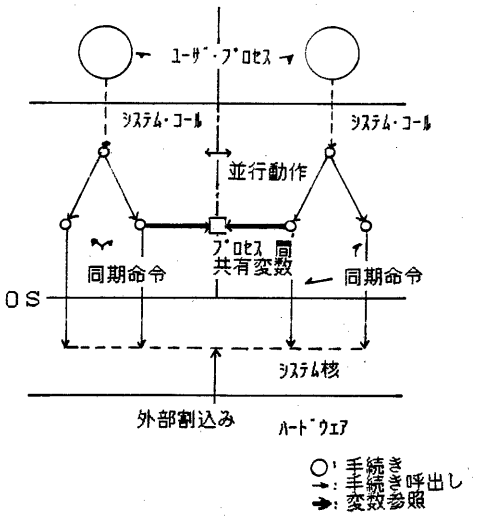
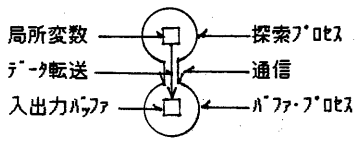


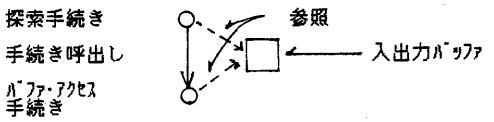
図３　ＵＮＩＸの制御構造

て管理されており、資源アクセスの相互排除は、同期命令sleep およびwakeupを用いてプロセス間共有変数へのアクセスを相互排除することにより実現している。

ここで、制御機構の相違による性能差について考察してみる。図４に示すような、ディレクトリの探索処理の実現法について考える。この



提案方式による実現法



ＵＮＩＸにおける実現法

図４　ディレクトリ探索処理の実現

とき、ディレクトリは、入出力バッファ中にあるものとする。提案方式では入出力バッファはバッファ・プロセスにより実現されるので、探索を行なう探索プロセスは、通信によりバッファ・プロセスからディレクトリの内容を受け取る。これに対して、従来方式では、バッファはプロセス間共有変数により実現され、バッファ・アクセス手続きが存在する。バッファ・アクセス手続きは、他のプロセスがバッファを使用していないことを確認した後バッファ領域へのポインタを返すので、ディレクトリ探索手続きは、バッファに直接アクセスして探索を行なう。他のプロセスがバッファを使用している場合は、sleep により待ち合わせを行なう。すなわち、通信は、バッファ・アクセス手続きの呼出し、および、sleep による待ち合わせに対応し、機構の相違により性能が異なると考えられる。また、従来同一のプロセス中にあったディレクトリの探索とバッファの管理機能が、別個のプロセスによって実行されることによって生ずる性能の差異も考えられる。たとえば、両者は別個のループにより実現されているので、ループ制御によるオーバヘッドが考えられる。一方バッファの先読み等はより単純に実現できるので、オーバヘッドが減少する可能性もある。

次に、今回実現した方法による通信機構のオーバヘッドについて考察する。２．で述べたように、通信は、a)callプリミティブの引数の作成、b)状態の待避、c)実行可能なプロセスの選択、d)状態の回復、e)引数データの転送、の順に実行される。以下では、通信処理を、大きく、同期機構と（b),c),d))、データ転送（a),e))、に分けて考えることにする。前者はＯＳ核が実行し、後者はプロセスが実行する。

ディレクトリ探索の例において従来方式と比較したオーバヘッドは、従来方式では相互排除の必要がある場合にのみプロセスの切り換えが行なわれるのに対して、提案方式では通信により常にプロセスの切り換えが行なわれるので同期機構によるオーバヘッドが生ずる。しかしながら、従来方式では、システム内のプロセスを切り換えることによりユーザ・プロセスも切り換わるので、プロセスの切り換えのたびに複雑なスケジューリングを実行する必要があり、１回の切り換えあたりのオーバヘッドは従来方式の方が大きいと予想される。

通信によれば、従来方式よりもデータ転送量が増加する。上記の例では、従来方式においてディレクトリ探索手続きが直接バッファにアクセスしていたものが、探索プロセスの局所変数へ転送した後アクセスすることになり、あらたな転送が生じる。しかしながら、大量のデータの場合には、領域へのポインタのみ引数機構により転送し実体は局所領域間で直接転送することにより、上記a)のオーバヘッドの減少を図っている。さらに、ファイル入出力では、入出力すべきデータはスーパバイザ、ファイル、inode 、バッファの各プロセスを経由して転送され転送量が大きくなるので、ランデブを入れ子にして用い中間のプロセスを停止してポインタのみを転送することにより、ファイル入出力における実際のデータ転送を１回に減らしている。

４．　性能の測定

実現されたシステムおよびＵＮＩＸシステム

上で同一のユーザ・プログラムを実行し、ストップ・ウォッチによるレスポンス・タイムの測定、および、ハードウエア・モニタによるＯＳ内部の各処理の所用時間を測定した。

測定は、単一のシステム・ディスク上に実現されたＯＳおよびＵＮＩＸシステムを格納し、交互に動作させて行なった。このため、両システムの動作環境は、ほぼ同一である。ユーザ・プログラムとして、主に、Ｃコンパイラを用いた。Ｃコンパイラは、最も頻繁に用いられるコマンドの一つであり、典型的な負荷であると考えられる。

システムの実行特性を実時間で測定するために、ハードウェア・モニタを用いた。ハードウェア・モニタは、図５に示すように、高インピ

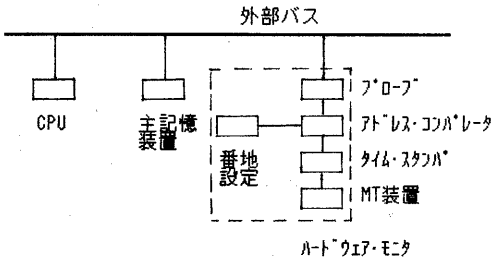


図５　性能の測定方法

ーダンスのプローブによりプロセッサの外部バスの信号をモニタし、プロセッサが主記憶への書き込みを行なうたびに書き込み番地と設定番地の比較し、トリガ信号の発生、タイム・スタンプの付加、磁気テープへの記録、等を実行する。これにより、ソフトウェアが予め定めてある設定番地への書き込みを行なうと、その内容、および、時刻が磁気テープに記録されることになる。そこで、システムの状態が変化する際にこの番地への書き込みを行なう命令を付加すれば、状態の遷移を後に解析することができる。

５．　解析

５．１　性能の比較

表２は、実現されたシステムとＵＮＩＸシステム上のユーザ・プログラムの実行時間の比較であり、実行コマンドを端末に入力した後、実行が終了して次のプロンプトが印字されるまでの時間をストップ・ウォッチで測定したものである。図中において、新システムとは実現され

表２　新旧システムの性能

ﾕｰｻﾞ・ﾌﾟﾛｸﾞﾗﾑ	旧ｼｽﾃﾑ	新ｼｽﾃﾑ
Cｺﾝﾊﾟｲﾗ	107 sec	109 sec
ｴﾃﾞｨﾀ	26 sec	29 sec

たシステム、旧システムとはＵＮＩＸをさし、Ｃコンパイラとは、Ｃコンパイラにより９４５行、１２Ｋバイト長のソース・プログラムをコンパイルするのに要した時間であり、エディタとは、同一のソース・プログラムに対してエディタのコマンドを実行するのに要した時間であ

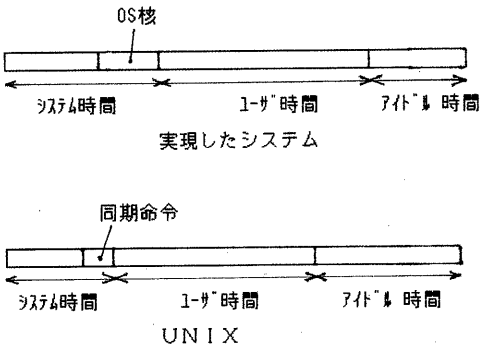


図６　処理時間の比率

る。図６は、Ｃコンパイラを実行した際のシステム各部のプロセッサ時間をハードウェア・モニタによって測定したものである。すなわち、システム・コールの入り口、割込み処理の入り口、アイドル手続きの入り口、および、ユーザ・プロセス・スケジューラの出口に設定番地への書き込み命令を埋め込み、ハードウェア・モニタによりシステム時間、ユーザ時間、および、アイドル時間を測定したものである。これによれば、システム時間が54％増加していることがわかる。

以下で、システム時間について考察する。予想されたように、通信が頻繁に実行されている。上記の負荷では、1112個のシステム・コールの処理に 29065回の通信が実行されており、平均して１回のシステム・コールの処理に26回の通信が実行されることが測定された。また、１回にスケジュールされたシステム・プロセスの平

均実行時間は、230μsecにすぎず、これは、核プリミティブの処理時間にほぼ等しい。表３は、システム・プロセスが停止する要因、および、その確率を示したものであり、外部割り込みによる1.6%以外の場合には、自ら核プリミティブを発行して停止することを示している。このため、システム時間の約４０％が、ＯＳ核の実行に費やされていることが判明した。これに対して、ＵＮＩＸシステムでは、平均して１

表３　プロセスの停止要因

ﾌﾟﾛｾｽ の停止要因	比率
callﾌﾟﾘﾐﾃｨﾌﾞの発行	24.6%
accﾌﾟﾘﾐﾃｨﾌﾞの発行	36.8%
endrﾌﾟﾘﾐﾃｨﾌﾞの発行	36.9%
外部割り込みの発生	1.6%

回のシステム・コールの処理に1.2回のプロセス切り換えが行なわれており、システム時間の約３０％がsleepおよびwakeupの処理に費やされている。すなわち、提案方式によるシステムは、ＵＮＩＸに比較して、同期機構では114%、その他の部分では29%それぞれプロセッサ時間が増加している。同期機構のオーバヘッドについては、以下で解析を行なう。他の部分におけるプロセッサ時間の増加の原因は、現状では不明である。

５．２　通信機構の性能解析

はじめに、同期機構のオーバヘッドについて解析する。表４は、ＯＳ核の各機能の、ＯＳ核

表４　ＯＳ核の各処理に要する時間

機能	実行時間	平均処理時間
call ﾌﾟﾘﾐﾃｨﾌﾞ	26.8%	181 μsec
acc ﾌﾟﾘﾐﾃｨﾌﾞ	47.3%	213 μsec
endr ﾌﾟﾘﾐﾃｨﾌﾞ	12.3%	55 μsec
ｼｽﾃﾑ･ｺｰﾙ処理	2.0%	292 μsec
外部割込み処理	18.4%	274 μsec

全体の実行時間に占める割合、および、１回の処理の平均時間である。これより、１回の通信の実行に要する平均時間は449 μsecであることがわかる。さらに各処理においては、状態の退避、回復にそれぞれ43μsecかかることが判明した。そこで、通信における同期処理では、41%が状態状態の退避、回復に費やされており、残りの59%がスケジューリングに費やされていることになる。

図７は、同期機構のオーバヘッドとシステム性能の間の関係を明らかにするために、意図的

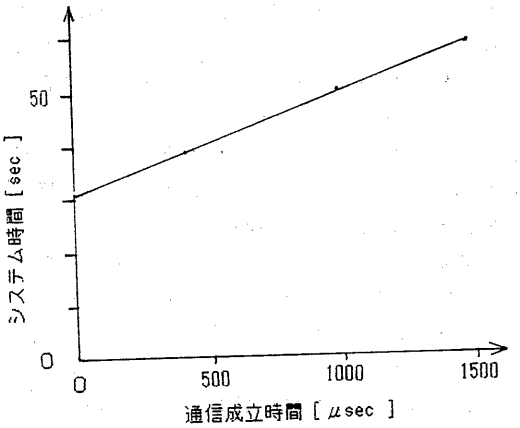


図７　同期機構の性能と全体性能

に同期機構の性能を低下させて測定を行なったものである。これは、callおよびacc 核プリミティブ内部にループ命令を埋め込んで通信が成立するのに要する平均時間を延長したものであり、オーバヘッドに比例してシステムの性能が低下しているのがわかる。

次に、通信におけるデータ転送のオーバヘッドについて解析する。図８は、Ｃコンパイラが

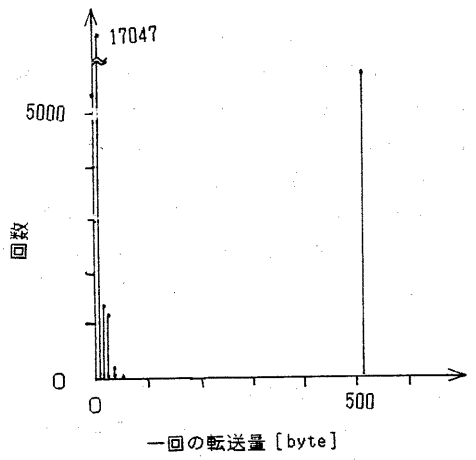


図８　データ転送量

動作している間にシステム・プロセス間で論理

的に転送されるデータ量を示したものであり、1回の通信における、callプリミティブの引数により実際の転送されたデータの転送量とポインタを用いて仮想的に転送されたデータ転送量の和を示したものである。実際の転送ののべ転送量は 118Kバイト、ポインタによる仮想的なデータ転送ののべ転送量は3Mバイト、全転送ののべ転送量は3.1 Mバイトであった。使用したプロセッサでは、1ワード（2バイト）の転送に 5.5μsec かかるので、実際のデータ転送に要する時間は、 326 msec となる。これは、システム時間の 0.1%にすぎない。すなわち、今回の実現においては、データ転送によるオーバヘッドの影響は少ないと言える。

しかしながら、他の通信実現方式、特に、分散システムにおいて他のプロセッサにあるプロセスとの通信では、すべて実際に転送する必要がある。そこで、同期機構の場合と同様に、ループ命令によりポインタによる転送量に比例するオーバヘッドを付加してシステムの性能を測定した。バッファ等の連続した領域を転送するのに図9のような機械命令を用いると、1バイトあたりの転送に4.71μsec かかる。この値に基づくオーバヘッドを付加した測定の結果、38.4sec であったシステム時間が53.3sec になることが判明した。さらに、図10は、より大きな転送時間がかかると仮定して測定したシステム時間を示したものである。これにより、通信機構の性能とシステム全体の性能の関係を明らかにできる。

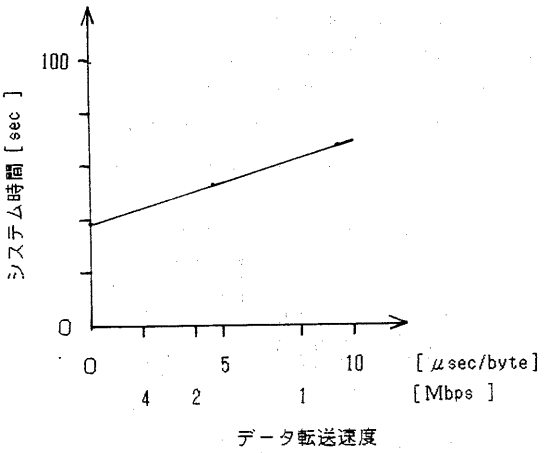
```
1:    mov   (r0)+,(r1)+    ／転送
      sob   r2,1b          ／ループ制御
```

図9　データ転送のための機械命令

6．ハードウェアによる性能改善

ここでは、2台のプロセッサを交互に用いて通信における同期処理のオーバヘッドを短縮する方法について延べる。5．で述べたように、今回の実現においては、通信機構のオーバヘッドの大半は同期機構によって生じたものである。そこで、ハードウェア機構によりこれを改善することを考える。

図10　転送オーバヘッドとシステム時間の関係



ここでは、2台のプロセッサA，Bを用いる。Aはプロセスを実行する。Bは、プロセスの状態の待避、実行可能なプロセスの選択、そのプロセスの状態の回復、すなわち、OS核に対応する処理を行なう。Aの実行するプロセスが通信待ち等により停止した場合には、直ちに、Bが他のプロセスを実行する。

このような方法をとることにより、同期機構のオーバヘッドの大半を吸収することができる。また、従来のアーキテクチャによるプロセッサ、特に、ワンチップ化されたプロセッサを用いることが可能であり、容易に実現することができる。

7．むすび

通信で結合されたプロセス集合によるOSの性能について述べた。測定によれば、通信における同期機構のオーバヘッドが大きいことが判明した。

現状では、Cコンパイラを負荷とする測定が主であるので、今後は課題は、種々の負荷に対する測定を行なって、実際の動作環境により近い測定を行なうことである。

参考文献

1）田胡和哉、益田隆司：ｵﾍﾟﾚｰﾃｨﾝｸﾞ・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).