

## UNIX\*システムコールの性能評価

三菱電機株式会社 情報電子研究所

中村克巳 野地 保

1991年3月15日

多くの分野で UNIX が利用されるようになってきている。UNIX では、システムコールは、プリエンブションしないため応答性に大きく影響を与える。なかでも、プロセス生成 (fork) などのシステムコールは一般的に重いといわれ、これを避けるために copy-on-write のテクニックや v f o r k のインプリメントなど幾つかのアプローチがなされている。このようなシステムコールが、定性的に重く、大きな実行時間を要するということは広く知られているが、それが実際のどの程度であるかという定量的な数字となるとほとんど知られてはいない。そこで本論文では、この定性的に重いといわれているシステムコールを、実行時間に影響を与えるパラメータを大きく変化させながら呼び出した時の CPU 時間を測定することにより、定量的な評価を試みた。

## Performance Evaluation of the UNIX System Calls

Mitsubishi Electric Corporation

Information Systems and Electronics Development Laboratory

Katsumi NAKAMURA Tamotsu NOJI

UNIX is used as Operating System in several software fields. System calls do not preempt in the UNIX, which affects dramatically to the response of the system. It is said that the system call, such as "fork" which creates a process, is heavy weight, and some implementation of UNIX employs copy-on-write technique and vfork for avoiding the heavy execution. It is well-known that these system calls are heavy weight qualitatively, but the quantitative analysis is very rare. This paper describes about the performance evaluation of system calls, measuring the CPU time, when execution parameters, which affects execution time greatly, are changed.

---

\*UNIX システムラボラトリーズ社が開発し、ライセンスしています。

## 1 はじめに

近年、数多くの分野において、計算機の OS として UNIX が利用されるケースが増えている。そもそも、研究用の OS として世に登場し、研究者を中心に世界中に広がってきた。当初は、専門家向け、玄人受けする OS などと呼ばれ、利用者はごく限られていたように思う。

その後、エンジニアリングワークステーションを中心にその利用分野を大きく広げてきたが、現在ではミニコンピュータからメインフレーム、そしてスーパーコンピュータに至るまで幅広く利用されるようになってきている。特にワークステーションを中心とする小型の計算機における勢力の拡大は目覚しく、計算機のダウンサイジングを推進する大きな力となっている。

なかでも、ワークステーションの分野では、RISC チップの登場以降、その性能の向上は目覚しく、メインフレームの性能レンジを凌駕する程のものとなっており、規模の大きなサーバタイプのワークステーションは、従来、その困難さばかりが報告されたいたデータベース管理システムなどのビジネスコンピュータの分野への応用までもカバーするようになってきている。

また、リアルタイムシステムへの応用といったものも検討されており、一部には、UNIX カーネルに手を加えたり、ユーザインタフェースを維持し、新たにカーネルを作り直したようなリアルタイム UNIX が商品化されている。

UNIX では、スケジューラが CPU を使用した時間が長くなればなるほどプロセスのプライオリティを徐々に下げていくため、高いプライオリティを維持することはできない。さらに UNIX では、システムコールは、決してプリエンブションせず、ひとたび走り出したら最後まで実行される。より高いプライオリティのプロセスが発生しても、実行を開始したシステムコールの終了まで待たされることになる。このことは、応答性という面において大きなネックである。

このように UNIX を応答性といった面から見た場合には幾つかの問題点があげられるが、その他のケースを考えた場合にも、システムコールのレベルでの性能をある程度把握していないと最適なシステム設計を行なうことはできない。そこで、UNIX のシステムコールについて、幾つかのいわゆるデファクトスタンダードといわれるマシンの上の幾つかのバージョンの UNIX について、代表的なシステムコールの実行時間を測定し、それをシステム設計のための基礎的なデータとして収集し、それぞれの評価を行なってきた。

本論文では、BSD 系 UNIX を代表とする UNIX として、かなりポピュラーとなっているサンマイクロシステムズ社の SunOS<sup>†</sup>を選び、UNIX System V としては AT&T 社の新しいバージョンである UNIX System V Release4.0 を選んで評価報告する。また、BSD 系 UNIX と UNIX System V の双方の機能を有するといわれる別のバージョンの UNIX として、ヒューレットパッカード社の独自の UNIX である HP-UX A.7.00<sup>‡</sup>を代表例として示し、比較している。特に、SunOS については、全く同一のマシンで2つの異なるバージョンの測定を行なうことができたので、SunOS Release 3.2 と SunOS Release 4.0.3 の両方について結果を示し、両者を比較している。

## 2 評価の対象

### 2.1 評価の目的

UNIX のシステムコールの中で、プロセスを生成したり、新しいプログラムでプロセスを初期化するようなプロセス管理に関するものは、非常に大きな実行時間が必要とされる。このようなシステムコールが、定性的に重いということは一般的に良く知られている。しかし、それを定量的に評価を行なった例は少なく、一部に比較対象としてデータが出されているのを見る程度である。そこで、プリエンブションしないため応答性に大きく影響を与える UNIX のシステムコールの実行時間 (CPU 時間) を測定し、そのデータに対して可能な限り定量的に分析することを目的として、今回の評価を行なった。

### 2.2 評価の項目

ここでの評価対象としたシステムコールは、プロセス管理を行なうもので、以下の2つである。

<sup>†</sup>SunOS は、Sun Microsystems, Inc. の商標です。

<sup>‡</sup>HP-UX は、Hewlett-Packard Company の登録商標です。

- fork システムコール
- exec システムコール

これらは、UNIX を代表するシステムコールでは、重要で、かつ特に処理が重いといわれるものを選択している。実際の性能評価を実施したものは、この他にも、主記憶管理、シグナル、ファイル入出力、プロセス間通信などの項目についても測定を行なっているが、本文ではスペースの関係もあり、最初に示したプロセス管理に関する fork と exec に関して説明する。

## 2.3 評価機種のシステム構成

以下に測定に使用した測定機種の諸元を示す。"AT&T Unix System V/386 Release 4.0 Version 1.0" については、これを載せることが可能なオリベッティ社の 386 マシンを使用した。

<測定機種の諸元>

機種: Olivetti M380/XP5<sup>5</sup>

CPU: 80386(20MHz)

主記憶: 4 MB

OS: AT&T Unix System V/386 Release 4.0 Version 1.0

ディスク: 5.25 inch 300MB Hard Disk Unit, ESDI interface

機種: SUN4/280S

CPU: SPARC(16.7MHz)

主記憶: 128 MB

OS: SunOS Release 4.0.3/SunOS Release 3.2

ディスク: 892MB Hitachi DK815-10 × 2

ディスクコント: Xylogics 451 Controller -SMD

機種: HP9000/835SE

CPU: HPPA

主記憶: 64 MB

OS: HP-UX A.B7.00

ディスク: 304MB 7963B HP-IB I/F ディスクコント内蔵型

304MB 97963B HP-IB I/F ディスクコント内蔵型増設ディスク

571MB 7937FL HP-FL 光ファイバリンクディスクコントローラ内蔵型

## 2.4 測定環境と測定方法

すべての測定は、上述のシステム上でシングルユーザ環境で行なった。測定プロセス以外のユーザプロセスは存在していない状態である。測定に当たっては、UNIX の標準のタイマー関数を使用して、CPU 時間、経過時間、入出力回数を測定している。測定のためにカーネルに手を加える、専用のタイマーを使うなどのことは一切していない。同時に、ページフォルトの回数、あるいは CPU 使用率、システムコールの呼び出し回数など UNIX 自身によって報告されるシステムのアクティビティを収集し、CPU が測定プロセスによって消費されたこと、別のデーモンなどが消費していないことをチェックし、データの信憑性を確認した。

プログラミングは、できる限り通常の使用に近い形で行ない、高い性能が出るような特別な措置はしていない。ただし、タイマーの精度が 16.7ms (クロックチック) であるため測定プログラムは、システムコールを最低でも 1000 回以上 (ほとんどが 10000 回) ループさせて測定している。これを越えるより高い精度での測定に関しては、ここでは考慮していない。

また、それぞれの測定に当たっては、測定データに大きく影響を与えると思われる条件 (プログラムのサイズなど) をそれぞれの OS のインプリメントから予想し抽出して、それぞれ測定プログラムのパラメータとして値を変動させ、各変動するパラメータ毎にデータを収集した。

<sup>5</sup>M380/XP5 は、Olivetti の商標です。

### 3 fork (プロセス生成)

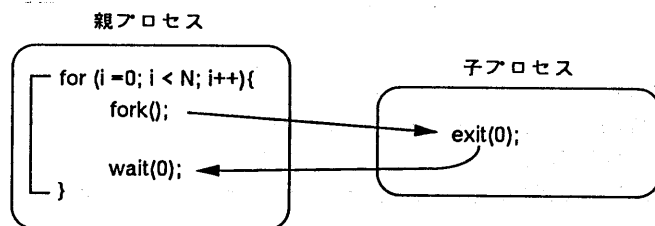
forkシステムコールは、新しいプロセスを生成するもので、呼び出した親プロセスと一部の属性を除いて全く同じものであり、親の複製を作るものである。したがって、forkでは、親プロセスのアドレス空間を子プロセスにコピーする処理が発生するため、インプリメントによっては、そのプログラムサイズによって実行時間に大きな差が出ることが予想される。実際に4.3BSD UNIXでは、複数のプロセスに単一のページを共通に参照させ、いずれかが、それを変更した時に初めて物理的なコピーを行なうcopy-on-writeのテクニックやvforkと呼ばれるforkを効率化したシステムコールなどが用意されており、不要なアドレス空間のページのコピーを行なわないような幾つかのインプリメントがなされている[1]。

#### 3.1 測定方法

forkシステムコールは、子プロセスを生成し、その時点で親プロセスと子プロセスが同時に実行されるようになる。同じCPUで複数のプロセスが同時に走っているのでは、そのCPU時間を測定したといっても、測定の対象が不鮮明となる。今、測定したいのは、forkを一回実行するのに要するCPU時間である。明らかに、測定時点でCPUが、forkだけを実行しているようにする必要がある。

生成された子プロセスを即座にpauseさせてしまえば、CPUで実行されるプロセスは親のforkの実行だけとなり、forkのCPU時間の測定が可能である。しかし、この場合には各ユーザの生成できるプロセス数の制限から測定のループ回数が30回度に制限されてしまう。測定には、少なくとも1000回以上はループさせないとミリ秒単位のタイマーでは、結果が疑わしくなってしまう。したがって、forkだけをループさせて測定することはあきらめて以下のような方法を取ることにした。

測定プログラムの親プロセスは、子プロセスをforkするとすぐにwaitし、子プロセスの終了を待つものとし、子プロセスは、走り出すと即座にexitする。従って、CPUの方から見れば、親のfork、子のexit、親のwaitの順番でシステムコールが実行される。測定結果は、この3つのシステムコールを連続して実行する時のCPU時間ということになるが、ここでは便宜的にforkの実行時間として説明している。図の3-1には、測定プログラムの構造を図示している。



● 図3-1 forkの実行時間測定プログラム

測定時のパラメータとしては、プログラムのサイズを選択した。複製するプロセスのアドレス空間内のページ数が、実行時間に大きな影響を及ぼすことが予想されるためにパラメータとして選択した。その対象としては、forkするプロセスのテキスト領域、データ領域（初期化データ、bss）、スタック領域をそれぞれパラメータとすることが考えられるが、テキスト領域を意図したサイズに、かつ測定したい数MBにまでにすることが非常に困難なため除いている。実質は、初期化データ領域に近い数値になると考えている。パラメータは、以下の3項目とした。

- 初期化データ領域サイズ
- bss領域サイズ
- スタック領域サイズ

#### 3.2 測定結果

図3-2～3-4は、それぞれ初期化データ領域、bss領域、スタック領域をパラメータとした時のforkのCPU時間を3種のUNIXについて示したグラフである。図中には、グラフの伸びの比率をそれぞれの機種毎のページサイズ単位のCPU時間として示している。

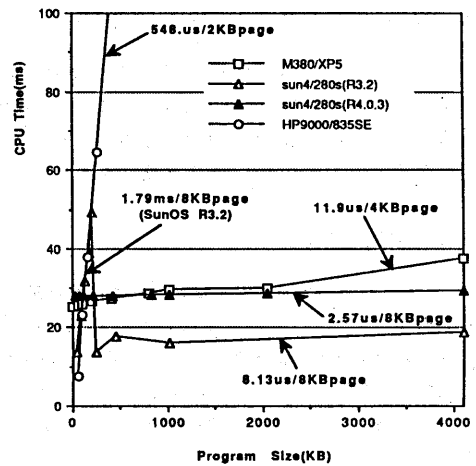
4.3BSD UNIX の例によると、`fork` の処理は、子プロセスの2次記憶、`proc structure` のエントリやページテーブル、新しい `user structure` を割り当てたり、初期化のためにカレントプロセスのそれからからコピーする。テキスト領域は、参照カウントをインクリメントして共有させるが、データやスタックでは、fill-on-demand ページは PTE を、その他のページは、データをコピーして、アドレス空間を複製する [1]。最後のページのコピーの処理が、その UNIX のインプリメントによって、大きな差を生んでいると考えられる。

初期化データの場合、グラフの傾きが明らかに2つのグループに分かれている。中でも特徴的なのは SunOS のものである。SunOS Release 3.2 では、数 100KB 付近のある値を境界に CPU 時間が大きく減少している。SunOS では、NFS のインプリメンテーションのために、マジックナンバー 413(paged in) のプログラムを、そのスレッシュホールド以下のプログラムサイズでは、410(swapped in) として取り扱わせることとしていたためである [2]。この傾向が、SunOS Release 4.0.3 では全くなっている。コピーオンライトメカニズムを使用するようになったためページが共有され、ページのコピーがこの段階では発生していないことが予想される。ただし、サイズが 0 に近い場合の CPU 時間では、SunOS Release 3.2 に比べて CPU 時間がかなり大きい。つまり、不要なコピーは削減できているが、最低限必要な処理の時間は大きくになっている。

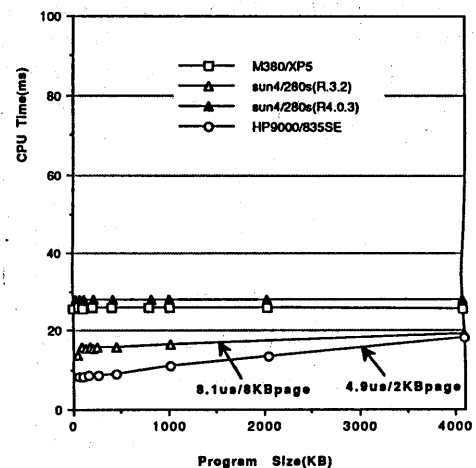
また、HP-UX の CPU 時間が非常に大きい。しかし、これは測定プログラムをデフォルトのフラグでコンパイルした時のものであるが、マジックナンバーが 410 に設定されている。オプションつきでコンパイルすると、413 のプログラムが作られるが、その時には CPU 時間は、`bss` の場合の数値程度になり、小さくすることが可能であり、特に遅いと決めてしまうことはできない。逆に実行ファイルのロードのフォーマットをユーザが、状況に応じて変更できるようになっていて、柔軟性があるともいえる。

`bss` のときには、いずれの場合にもグラフの傾きは小さい。System V R4.0 と SunOS R4.0.3 では、完全に傾きが 0 である。

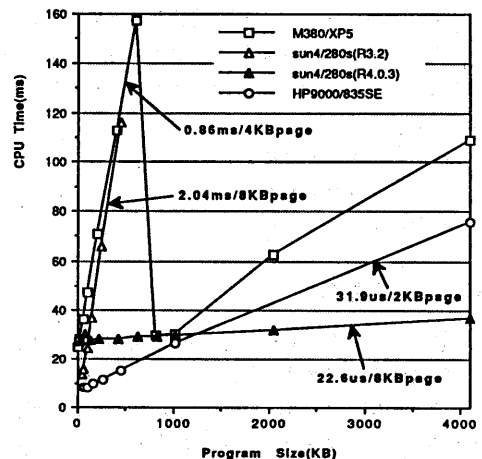
スタック領域のデータでも、初期化データと同様に2つのグループに分かれている。System V R4.0 では、スタックはページのコピーが行なわれているように考えられる。SunOS R3.2 でも、スタックのページを全てコピーしており大きな CPU 時間を要していた。このためかスタックサイズが 512KB (デフォルト値) に限定されていたが、SunOS R4.0.3 では、この傾向が消え、スタックサイズも大きくとれるようになっている。



● 図3-2 `fork` の実行時間 (パラメータ: 初期化データ領域サイズ)



● 図3-3 `fork` の実行時間 (パラメータ: `bss` 領域サイズ)



● 図3-4 `fork` の実行時間 (パラメータ: スタック領域サイズ)

## 4 execシステムコール(プロセスの初期化)

execシステムコールは、呼びだしプロセスを新しいプログラムに置き換えてしまい、古いプロセスを残して、その上にファイルシステム中の実行可能ファイルから新しいプログラムを上書きするものである。通常、プログラムを新たに実行させる時に使用されるものであり、重要なシステムコールといえる。実行可能ファイルの中には、上書きされるデータセグメントなどが格納されており、その大きさによって実行時間に差が出るのが予想される。

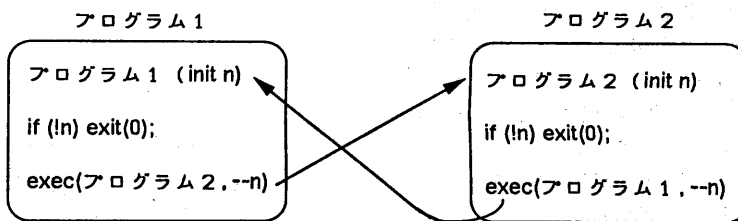
### 4.1 測定方法

execシステムコールは、プロセスを初期化してしまうので、実行時に全ての変数、ループカウンタやタイマーなどの値もすべてリセットされてしまう。従って、execを単純にループさせ前後でタイマールーチン呼び出しでも測定はできない。そこで、この測定には以下のような方法をとった。

測定用に2つのプログラムを用意する。お互いのプログラムは、それぞれ相手側をexecし、お互いに起動し合う。引数として、カウンタの値を渡して、カウンタをデクリメントし、カウンタのチェックを行ない終了条件とする。

したがって、図4-1に示すようにプログラム1は、プログラム2を自分自身の上に(子プロセスをforkすることなく)execする。また、プログラム2は、プログラム1を自分自身の上にexecする。これをカウンタの指定だけ繰り返し、その実行時を測定する。それぞれのプロセスのタイマーはexecのたび毎にリセットされているので、測定プロセスのタイマーを読み出しても1回分のCPU時間しか出力されない。

したがって、このexecの繰り返しを測定するには、測定プログラムを起動するシェルのtimeコマンドに測定プログラムを掛けてやることにより行なう。この方法は、execのCPU時間を測定するのに有効な1つの手段である。



● 図4-1 execの実行時間測定プログラム

測定時のパラメータとしては、forkと同様の理由によりプログラムのサイズを変えることとした。ただし、その対象としては、スタックを伸ばしおくことが不可能であるため、データ領域(初期化データ、bss)のみとしている。また、execは、引数と環境を新しいアドレス空間にコピーするために、カーネルの領域やスワップデバイスの一部などのテンポラリ領域に、古い空間の引数と環境を1度コピーする。この影響を見るために敢えて、大きなサイズの引数ベクトルを持たせてexecした時のCPU時間をも測定した。この結果以下の3項目をパラメータとして選択した。

- 初期化データ領域サイズ
- bss領域サイズ
- 引数サイズ

### 4.2 測定結果

図4-2~4-4は、それぞれ初期化データ領域、bss領域、引数サイズをパラメータとした時のexecのCPU時間を3種のUNIXについて示したグラフである。図中には、グラフの伸びをそれぞれの機種毎のページサイズなどに応じて示している。また、execでは、ファイルの読み出しがあるので測定に当たっては、入出力回数などのシステムアクティビティをチェックし、CPUが他のデーモンなどに使用されていないことを確認している。

UNIX System V の例では、`exec`は、`namei`を使用してファイルにアクセスし、実行可能状態を確認し、ファイルヘッダを読み込みロードモジュールであることを確認する。`exec`の引数を古いアドレス空間から新しいアドレス空間へコピーしてするために、一時的にカーネル空間のカーネルスタックや、あるいはスワップデバイスなどに格納する。次にマジックナンバーなどにしたがって、実行可能ファイルを主記憶へロードする。データ領域は、2つの部分に分けられる。初期化データ領域と `bss` 領域である。初期化データ領域が最初に確保され、次に `bss` 領域を、最後にスタック領域を確保する。カーネルは、実行可能ファイルに対して予めメモリを割り当てずに、例外を引き起こす。そこで例外処理が、"demand fill"か"demand zero"に従って、実行可能ファイルをメモリへ上書きしたり、ゼロ詰めを実行したりする[3]。

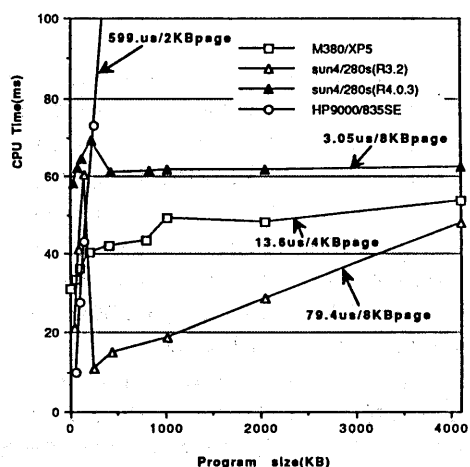
最後の実行可能ファイルからの初期化データの上書きやゼロ詰め処理の部分がインプリメントによって大きく異なっていると考えられる。

初期化データの場合、SunOS R3.2で、`fork`と同様のスレッシュホールドが見られ、CPU 時間に大きな変化がある。この傾向は、SunOS R4.0.3でも消えていない。また、SunOS R4.0.3では、サイズが0に近く小さい場合のCPU時間が、かなり大きくなっている。これは、`exec`のための必要最低限のCPU時間といえるのだが、`fork`同様に大きな時間を要するように変更されている。

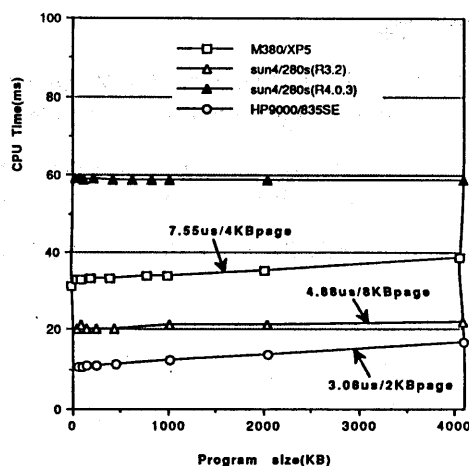
HP-UXでは、CPU 時間に大きな伸びが見えるが、これは前述の `fork`と同じコンパイル時の理由であり、デフォルトのフラグでマジックナンバー 410としてコンパイルされているためである。このため、`exec`呼び出し時に実行ファイルの初期化データが全てロードされているものである。オプションつきで、413としてコンパイルすると `bss`と同じ程度のCPU時間となり、`fork`と同様にユーザが、それを使用形態に応じてコントロールできることになる。

`bss`の場合では、いずれの場合においてもグラフの傾きは非常に小さい。SunOS R4.0.3では、完全に0である。この測定プログラムでは、`bss`は宣言されているだけなのだが、上書きやゼロ詰めは全く行なわれていないと考えられる。

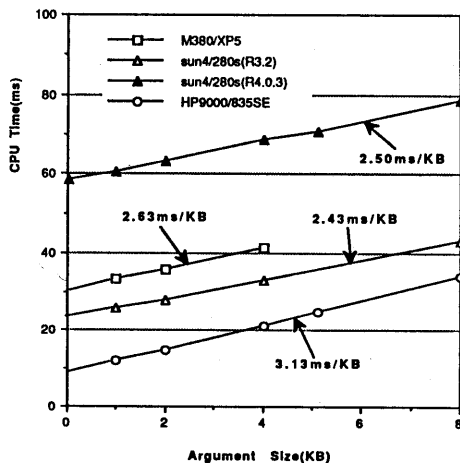
引数サイズをパラメータとした時には、グラフが全てのケースでリニアな伸びを見せている。引数を新しいアドレス空間にコピーするためにテンポラリ領域へコピーした分のCPU時間が引数サイズ0以上のところに現れている。コピーする場所の違いはあっても、処理自体には差がなく、アルゴリズム的な影響は見られない。



● 図4-2 `exec`の実行時間 (パラメータ: 初期化データ領域サイズ)



● 図4-3 `exec`の実行時間 (パラメータ: `bss`領域サイズ)



● 図4-4 `exec`の実行時間 (パラメータ: 引数サイズ)

## 5 おわりに

以上述べてきたように、"System V"系のものであるか"BSD"系であるか、あるいはその UNIX のバージョンによっては、プロセス管理の方式、主記憶管理方式の面でインプリメントが大きく異なり、全く同じプログラムを同じ条件で実行させた場合においても、それぞれの機種種の CPU 性能を考慮に入れたとしても、プロセス管理を行なうためのシステムコールの実行時間に大きな差が見られる。

もちろん、どのインプリメントが最適ということを一意に規定することはできない。それぞれの OS のインプリメントが、得意とする適応分野があるはずであり、fork システムコールが遅いということは、UNIX の応答性という観点に立って見た場合に大きな影響を持つことは確かではあるが、それはプロセス生成が遅いというだけであり、一度生成されてしまえば、それ以降のオーバーヘッドが少なく済むということであり、トータルで見た場合のスループットも低くなるとは限らない。

本論文では、プロセス管理を行なうシステムコールの実行に要する CPU 時間を、それぞれのシステムコールの実行に影響を与えるパラメータを変化させ、その影響を可能な限り定量的に評価した。これらのデータは、UNIX を用いたシステムの設計を行なうための基礎的なデータとして、数多くの場面で何らかの指針を示してくれるデータであると信じている。測定した OS には、既にバージョンがかなり古くなっているものもあり、今後のバージョンアップに応じて、継続して同じ測定をして、比較していくことも重要な課題となろう。

## 参考文献

- [1] Samuel J. Reffler, et al. *The Design and Implementation of the 4.3BSD UNIX Operation System*, Addison-Wesley Publishing Co., 1988
- [2] Russel Sandberg, et al. *Design and Implementation of the Sun Network File System*, Proceeding of Summer Usenix Conference, pp.119-130, 1985
- [3] Maurice J. Bach. *DESIGN OF THE UNIX OPERATING SYSTEM*, Prentice-Hall, 1986