オペレーティング・システム 51−13
（１９９１．７．１８）

待ち行列網によるOSモデル化の実験

柿元 満、大溝 孝

（株）東芝 総合研究所

待ち行列網によるOSのモデル化とシミュレーションによる評価を試みた。基本的には、プロセスなどのソフトウエア要素を'客'で、ハードウエア資源、セマフォアなどを'窓口'として記述する。特に問題となるのはOS内部で管理している資源の獲得をめぐる競合である。

現在我々が開発中のツールを用いてシミュレーションのプログラミングを実装し、その結果、実機では得ることが難しい性能指標を調べられることが分かった。また、OSを表現する上で、ツールに必要ないくつかの機能についても論じる。

AN EXPERIMENT OF MODELING OF SOME OPERATING SYSTEMS BY QUEUING NETWORKS

Mitsuru Kakimoto　　Takashi Ohmizo

R & D center, Toshiba

Komukai Toshiba-chou 1, Saiwai-ku, Kawasaki, Kanagawa 210, Japan

We built a couple of queuing network models of operating systems and evaluated them by simulations. Basically, hardware resources were modeled by servers, while software units, such as processes, were modeled by jobs. We were particularly interested in conflicts of processes in obtaining resources managed by the operating systems.

Simulation programs were implemented using a tool we had been developping, which enabled us to get performance indices that are hard to obtain by conventional methods. Features required of tools to represent operating system models are also discussed.

1 はじめに

我々は、計算機システムの設計、特にOSの設計を性能予測・評価の側面から支援するツール作りを目指している（図1）。特にOSに着目したのは次のような理由による。

- OSこそが計算機システムの中核である。ユーザーサイドから見た場合、計算機システムを特徴づけるのはOSである。また、OSは計算機システムを構成する様々な他の要素と比較しても寿命が永く、高い普遍性を持っている。従って、シミュレーションによる設計支援システムを作る場合、OS部分を先行して開発するのがもっとも妥当である。

- OSの設計を支援する適切なツールがない。

  OSは計算機の「裏方」であり、実行にはあまり大きくCPUパワーを使わないように設計されている。それが故に、ソフトウエア的な手法によるモニタリングは非常に大きなノイズを伴う。かと言って、ハードウエアモニタではその複雑な挙動を十分に捉え切れない。実機上でOSの挙動をモニタするのは困難である。

  シミュレーションは、OSの振舞いを調べる上で、実機では得ることができない情報を提供できる可能性がある。そもそも、設計段階のOSをモニタリングすることはできないので、OSのためのシミュレーションツールは必要である。

- 更に設計支援システムの中心にOSを持ってくることにより、様々なメリットが期待できる。

  OSの役割の一つはアプリケーション・プログラムに対して、プログラミングモデルを示すことである。従って、OSモデルを先行して開発しておけば、アプリケーションの設計者はそれとアプリケーションのモデルを組み合わせることにより、システムレベルのシミュレーションを実行できる。このようにして、自分が作ろうとするプログラムの評価を予め行なうことができる。

我々は、今回、アプローチの一つとして待ち行列網シミュレーションツールを用いたOSモデルの構築を試みた。ここでは、その経験を通して得られた知見と問題点について述べる。

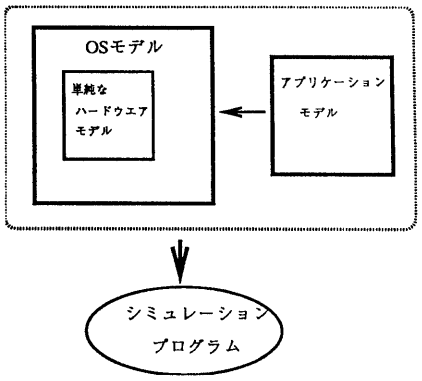

図 1: 設計支援システムの構成

2 方針

OSをモデル化すると言っても、そのすべての側面をシミュレートするのは実質的に不可能である。以下では簡単に機能と実現の二つの側面から考察を加える。

1. 機能
   OSがどういうアルゴリズムに従っていて、何を実行するのか。例えば、プロセスのスケジューリング方式や、ディスク・バッファの管理アルゴリズムなどである。

2. 実現
   OSが要求された機能を果たすためにはそれなりの資源を使う必要がある。例えば、OSはプログラムであり、その実行の間、CPUという資源を専有する。

OSを最も単純にモデル化する場合、その機能のみを問題とし、その実現は問わない、という方法がある。例えば、OSの動作は他の処理に比べて、十分短い時間で行なわれるものと仮定し、OSのコード自体の実行に要するコストは無視する。この方法は、シミュレーションプログラムの実装自体を非常に簡単にするし、アプリケーションまで含めたシステム全体の評価が可能である。

但し、アプリケーションプログラムがファイルへのI／Oやプロセス間通信を頻繁に発生すれば負荷全体の中で、OSコードの実行の占める割合を無視することはできない。特に、データベースシステムや、並列／分散計算機ではこういった状況が起きると予想される。

更に、負荷全体の中で、OSコードの実行の占める割合が大きければ、OSコードの実行中

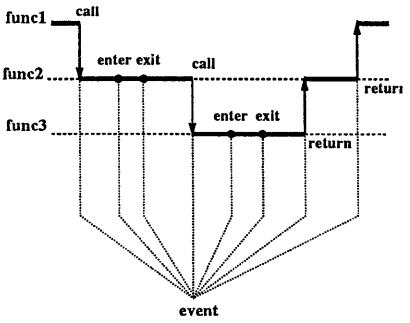

図 2: OS 内のイヴェント

に割り込みが入り、他のプロセスの中で OS コードが実行されるという可能性が大きくなる。この時、どちらの OS コードも危険領域 (critical section) に入ろうとしていたならば、タイミングの微妙な差でどちらが先に実行権を得るかが決まる。そして、どちらが先に実行されるかで、その後の経緯が大きく左右される可能性もある。

我々は機能と実現の両側面とも適切に評価し、機能と実現のトレードオフに関して、設計者の意志決定を支援したいと思っている。しかしながら、このシミュレーションを正確に実行しようとすると、OS の実行を細かな単位で分割する必要がある。

一つの方法は、CPU のインストラクションのハードウエアモデルを作っておいて、OS を CPU で実行される命令の組合せで記述する、いわゆる時間駆動 (time-driven, または clock-driven) のシミュレーションである。この方法は詳細なモデルを作ることができるが、計算機システムが、アプリケーションに対してどの様に振舞うかを調べるには、膨大な記述の労力と、計算時間を要するであろう。さらにこの場合、実現という側面に比重をかけ過ぎているので、それがシステム全体に、どのような影響を与えているのかを詳しく調べることはできない。

我々は、離散事象シミュレーションの手法で OS のモデル化を試みる。今回は特に次の 2 種類のイヴェントに注目する。

- 機能実行開始 (関数の呼びだし:call) とその終了 (復帰:return)
- 危険領域の侵入 (enter) と離脱 (exit)

OS はこれらのイヴェントの系列として捉えられる (図 2)。

また、待ち行列網シミュレーションツールを用いることにより、危険領域を一つの資源という形で捉える。

3 シミュレータ構築ツール eCceds

計算機システムをモデル化、そしてシミュレートするために我々はツール eCceds(enhanced C with constructs for event driven simulation) を開発中である ([1])。eCceds は待ち行列網をモデル化の基本とする離散事象シミュレーションツールである。

さて、待ち行列網でモデルを作るということは次のようなものを指定することに他ならない。

- サービス時間
- 経路
  複数の '窓口' の中で、'客' がどういう順序でサービスを受けるか
- '窓口' のスケジューリングアルゴリズム
  FIFO、LIFO などの指定
  優先度を用いた制御の仕方など

これらを '客' または '窓口' の固有のシナリオとして記述する。この時に 2 つのアプローチがある。

- '客' 中心の記述
- '窓口' 中心の記述

「'客' 中心の記述」とは、'客' と '窓口' の関係において、主体性 (または「意志」) は '客' の側にあると見做す。これはつまり、先に述べたような行動が、主として、'客' に内在する情報によって決定されるということである。

'窓口' 中心の記述は逆に主体性が '窓口' の側にあると見做す。

eCceds でモデル化を行なう場合、通常、'窓口' はハードウエア要素、'客' はソフトウエア要素に対応させる。具体的には、例えば CPU は一つの '窓口' になり、その上で動く一つ一つのプロセスがそれぞれ '客' となる。

eCceds は元々ソフトウエア・アーキテクチャの評価用に開発されたものである。様々なアプリケーションがどの様に動作するかを調べることを主な目的としている。このため、ハードウエアはソフトウエアの使用する資源という観点をとるので、eCceds では '客' 中心の記述をとっている。

例えば、サービス時間を決定する主要な要因は、そのサービス中に実行される処理量である。従って、サービス時間は '客' の側で指定する。

また、経路も実行されるソフトウエアによって決定されると考え、その指定も'客'の側で行なうようにした。

'窓口'で指定するのは、'窓口'内部でのスケジューリング方式である。eCcedsでは簡単なスケジューリング・アルゴリズムを持った'窓口'をあらかじめライブラリーとして用意してあるが、ユーザがスケジューリング・アルゴリズムを自由に記述することもできる。

以上述べてきたことが、通常の待ち行列網の実装に必要な機能である。さらにより複雑なモデルに対応するため、eCcedsでは次のような機能を設けている。

- '客'と'客'間の通信
  通常の待ち行列網の範疇では、'客'と'窓口'の相互作用しか考えない。しかしながら、計算機の世界では、プロセス間の通信などのように、'客'同志のダイレクトな相互作用がある。eCcedsでは、これは次の二つのプリミティヴによって実現できる。

  - send(受け取り手、メッセージ、メッセージのサイズ、遅延)
  - receive(送り手、メッセージ、メッセージのサイズ)

  send は非同期通信をシミュレートする。送信相手がまだ receive を実行していなくても、送り手が待たされることはない。転送したメッセージはメッセージのキューに入れられる。

  receive は send によって送られたメッセージを受けとる作業である。指定された送り手からのメッセージが未到着の場合には receive を実行した'客'はブロックされる。

- '客'のロック
  待ち行列網の場合、一つの'客'は一つの'窓口'でのみサービスを受けている。従って、一度には一つの'窓口'だけを資源として獲得できる。しかしながら、複数の資源を獲得したい場合もある。

  これに対応するため、eCcedsでは'窓口'にロックを掛けることを許す。ロックを掛けた'客'はロックの所有者と呼ばれる。ロックを掛けられた'窓口'はロックの所有者以外の'客'に対してはサービスを提供しない。

  ロックの所有者の'客'がロックを解除するときには、'窓口'にシグナルを送る。

4 OSのモデルの実装

先に述べたような、ソフトは'客'、ハードは'窓口'という単純な分類をした場合、OSをどう位置付けるかは難しい。OSが一種のソフトウエアであることは間違いないが、アプリケーションに対してはハードウエアを仮想化して見せるという機能を持っているため、その一部は'窓口'の機能として実装しなければならない場合がある。

OSの機能が実行されるのは次のような場合がある。

- 割り込み発生時の起動されるOSのコードとして
- CPUなどのハードウエアのスケジューリング方式として
- 危険領域での排他制御

これらはそれぞれ、性格も異なり、シミュレーション・プログラム上に実装される場合でも異なった形式をとる。以下ではそれぞれの実装形態を説明する。

4.1 割り込み

CPUに外部から割り込みが発生すると、それまで実行中であったアプリケーション・プログラムは中断され、割り込みハンドラーを経て、割り込み処理ルーチンに分岐する。この時点ではまだプロセスのスイッチは起こっていないので、論理的には割り込み処理ルーチンはそれまで実行中だったアプリケーションと同じプロセスの中で実行される。しかしながら、割り込み処理はアプリケーションとは全く別のシナリオであり、シミュレーション上は独立した'客'として実装する必要がある。

予め、各割り込み処理に対応するシナリオを持った'客'(これを割り込み'客'とよぶ)の記述が用意され、いつでも生成できるようになっている。割り込み信号がCPUに入るとこれらのシナリオを実行する割り込み'客'が生成される。

4.2 CPUスケジューリング

割り込み処理、またはシステムコールの処理が終った時点でCPUは、実行可能なプロセスのいずれかを実行する、いわゆるスケジューリングの作業がある。今の場合、プロセスは

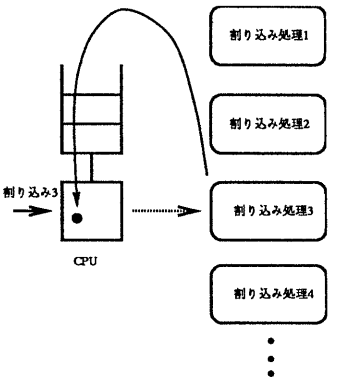

図 3: 割り込み処理のモデル

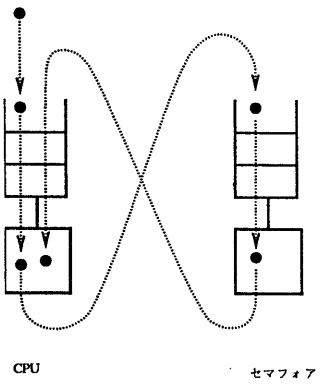

図 4: 排他制御の実現

（原則的には）一つの'客'として表現されているので、この作業は待ち行列にある'客'の中から次にサービスを受けるものを決定する作業と等しい。 従って、OS のスケジューリング機能は、待ち行列のスケジューリング方式として、シミュレーションプログラム上に実装されることになる。

4.3 排他制御のためのロック

OS は内部の共有データの一貫性を保つため、セマフォア等を用いて排他制御を行なっている。OS - OS 間の相互作用はこの危険領域の実行権の獲得の競合と言う形で発生する。 このため、この危険領域を OS がどのように管理しているかは性能に少なからず影響する。

今回も我々の実験では、排他制御のためのセマフォアも一つの資源と見なし、'窓口'として表現することにした（図 4）。プロセスを表現している'客'（プロセス'客'）は、共有変数を操作する場合に、実行権を獲得するため一旦セマフォアを表現している'窓口'（セマフォア'窓口'）の待ち行列に並ぶ。この時、CPU へロックを掛けておく。 これは、他のプロセスが CPU に入らないようにするためである。但し、実行権が獲得できなかった場合は、CPU のロックを解除し他のプロセスの実行を許さなければならない。

ロックを獲得した'客'はプロセッサ上での実行権を獲得したことになり、直ちにセマフォア'窓口'を出て、CPU'窓口'に入る。この時、プロセス'客'は他のプロセスが実行権を共有することがないように、セマフォア'窓口'にロックを掛けておかなければならない。

危険領域を終了したプロセス'客'は、他のプロセスが危険領域に入るのを許すため、セマフォア'窓口'にシグナルを送る。

5 モデル化の例

我々は eCceds を用いていくつかのモデル化とシミュレーションを試みた。 その中の二つの例について簡単に説明する。

5.1 ET-1(Debit-Credit) テスト（[2]）のモデル化

最初の実験の目的は、ファイルへのアクセスが多いアプリケーションが多数発生した場合、ファイル管理領域へのアクセスの競合を調べることである。 ファイルへアクセスするシステムコールが発生した場合、指定されたファイルの管理情報を同時に複数のプロセスが操作するのを避けるため、相互排除を行なう。 ここではアプリケーションとして、ET-1 テストを用いている。

待ち行列網で表現されたモデルの様子を図 5 に示す。 このシステムは、1 台の CPU と 2 台のディスクドライブを持っている。 また、ET-1 テストでは、口座（ACCOUNT）、窓口（TELLER）、支店（BRANCH）、履歴（HISTORY）の 4 つのファイルが配置されている。 各ファイルの管理情報のアクセスを排他的に行なうため、4 つの'窓口'がある。

'客'としては、ユーザレベルの ET-1 アプリケーションを表す ET-1'客'、OS の管理機能を表す OS'客'、ディスク装置内での処理の流れを表す Disk'客'の 3 種類がある。 アプリケーションプログラムのファイルへアクセスするためのシステムコールの発行は、eCceds

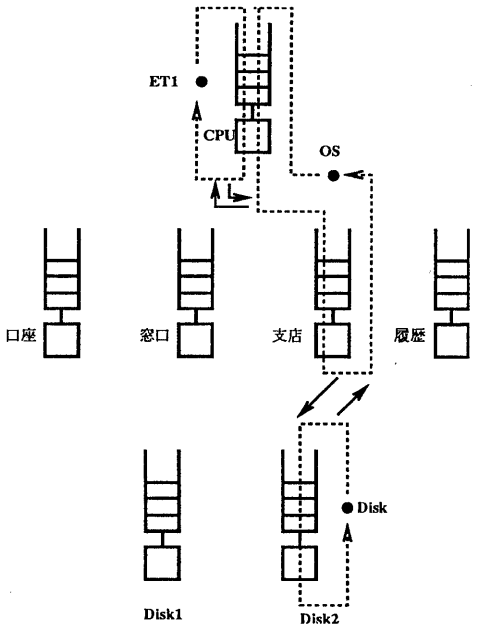

図 5: ET1 テストの待ち行列モデル

の send 機能を用いて、ET-1'客'の、OS'客'に対するメッセージの転送として表現している。 これにより、OS'客'が動きだし、セマフォア'窓口'に入り、実行権を獲得できた時、Disk'客'に対して、メッセージを送りディスク装置を起動する。

ディスクからの読み込みの終了は、Disk'客'からOS'客'に対するメッセージとして表現され、更にOS'客'がET-1'客'に対してメッセージを返すことにより、システムコールの終了が表現される。

この実験により、各'窓口'での待ち行列のでき方などから、ボトルネックの抽出が行なえる。

5.2 マルチプロセッサ OS のモデル化の試み ([3])

この実験の目的は共有メモリ環境下でのマルチプロセッサ OS の動作解析、排他制御の競合によるボトルネックの検出による OS の高速化にある。 これを実現するためのポイントは以下の 3 つである。

1. 割り込みメカニズムのモデル化

   OS は割り込みが発生することによって起動される。 割り込みはソフトウェアによる割り込み、すなわちシステムコールのみではなく、ハードウェアによる割り込み、すなわちクロック割り込みや端末からの割り込みも含まれる。 割り込みは、'窓口'である仮想 CPU によって処理される。仮想 CPU は割り込みを受け付けると、その割り込みを処理するための OS'客'を生成する。 OS'客'は処理を終えた後、割り込み前の'客'に復帰する。 また、仮想 CPU は割り込み禁止状態を作るために、割り込みマスク機能を持っている。

2. コンテクストスイッチのモデル化

   OS はコンテクストをスイッチすることで実行の流れを制御する。 コンテクストスイッチは、仮想 CPU のコンテクストをセーブ、ロードすることで実現できる。 このコンテクストには前述の仮想 CPU の割り込みマスクが含まれる。

3. 排他制御メカニズムのモデル化

   OS はその実行中に多くの排他制御を行なう。もし排他制御による競合がホットスポットを作る場合、性能上のボトルネックとなる可能性がある。 従って OS が排他制御の対象とする共有資源 (ランキュー、タスクテーブル・・・) には、それぞれの'窓口'を設け、排他制御を実現している。

このようにして実現されたマルチプロセッサ OS シミュレータは、アプリケーションプログラムからのシステムコール、タイマからの割り込み、端末からの割り込み等を入力としてシミュレーションを行なう。 その結果として得られた割り込み発生、排他制御の実行、終了、競合、コンテクストスイッチ等のログ情報を用いて、グラフィック表示ツールによる動作解析や排他制御の競合状態を調べることができる。

6 考察

我々は、OS 内部で発生する競合が、システムの性能を決定する主要な要因になるであろうという観点から OS のモデル化とシミュレーションを行なってきた。このアプローチでは、ボトルネック抽出や、動作の可視化など実機では得られない情報を与えてくれる。

但し、現状ではモデル化のためのプログラマの負担が大きい。 このため、モデルの変更も期待されるほど簡単ではなく、シミュレーショ

ンの利点が十分に生かせない。 ここでは、その問題点について考えてみる。

- 排他制御
  この実験では、危険領域の実行権も一つの資源と考え、'窓口'として表現した。しかしながら、基本的に待ち行列網では、一つの'客'は同時には一つの'窓口'しか資源として持ち得ないため、'窓口'のロック、アンロックを用いた込み入った操作を記述せざるを得なかった。資源の自動的な解放、または資源の所有権を自動的に譲渡する機能をツール側で提供する必要がある。

  また特にマルチプロセッサOSでの危険領域の実行権などの論理的な資源の場合、ハードウエア資源などに比較して、桁違いに多数生成される可能性がある。 通常、'窓口'には統計情報の収集用などに様々なデータ構造が附属しているため、多数の'窓口'を生成するとシミュレータが重くなる。論理的な資源をシミュレートする場合は、シミュレータにとって「軽い」資源もツール側で提供できなければならない。

- CPUのスケジューリング
  この実験では、プロセスを一つの'客'で表現する方針をとった。 OSの行なうプロセスのスケジューリングは、通常の待ち行列のスケジューリングよりもかなり複雑である。 このため、既存のモデルは使えず、新たに記述しなければならなかった。

  特に今回、割り込みやシステムコールが発生すると新たに'客'を生成するようにしたので、結果として、複数の'客'の集まりで一つのプロセスを表現することになった。 このため、OSのスケジューリングを待ち行列のスケジューリングに対応させるのがますます繁雑になってしまった。

- 機能を特定のオブジェクトから分離させて記述する能力
  シミュレーションを行なう場合、モデル化のためのオブジェクトを何種類か用意する。例えば、待ち行列網では'客'と'窓口'がそれに相当する。 シミュレータのプログラマはこれらのオブジェクトの属性としてシミュレートされるシステムの動作を記述する。

  すでに述べたようにOSというのはシステムを構成する様々な要素が持つ機能の総称である。このため、今回の実験でも、OSの機能のあるものは'客'の属性として、別のものは'窓口'の属性として実装される結果となった。 これは、プログラムの管理を繁雑なものにする。

最初に述べたように、この実験の目的はソフトウエア開発者を支援するツールを作ることであった。ただし現状は、OS（の一つの側面）をシミュレートしている段階で、アプリケーションプログラマ、またはOS設計者に提供できるツールとしては十分でない。

今回は主に待ち行列でモデル化したが、OSの設計者が思考のための道具として待ち行列を使うとは考えにくい。 ユーザに待ち行列を意識させないインタフェースの必要を実感した。

OSモデルの有効性については確認できたので、今後は、これまでの成果を踏まえて、よりOSを表現しやすいツールに改良していく予定である。

参考文献

1 柿元 満、 シミュレーションによる性能評価ツール eCceds、情報処理学会第41回全国大会論文集、 1990 4-105

2 喜連川 他、 データベース処理におけるベンチマーク、情報処理 Vol.31 No.3 pp328-342

3 大溝 孝、 性能評価のためのOSシミュレーションモデル、情報処理学会第42回全国大会論文集、 1991 4-67