実時間記号処理システム TAO/SILENT における軽量プロセスの実現

竹内郁雄[†] 吉田雅治[††]
山崎憲一[†††] 天海良治[†††]

実時間処理には不向きとされる記号処理システムに高い実時間性を付与するには，実時間ゴミ集めと応答遅延の小さい割込みを実現しなければならない．その前提となるのが軽量プロセスである．本論文は我々が開発中の実時間記号処理システム TAO/SILENT における軽量プロセスの特徴，実装，性能評価について述べる．それは軽量であるとともに，オブジェクト指向の意味での拡張性に富む．SILENT マシン（33 MHz）上のマイクロコードで実装されている．得られた性能は，プロセスの生成・起動・休止の最短時間が 16.91 μ秒，プロセススイッチの最短時間が 3.93 μ秒である．フィボナッチ関数の再帰をプロセス生成にしたベンチマークテストでは，数万個のプロセスを生成するもので，SuperSparc+（60 MHz）SunOS4 の軽量プロセスの 50 倍の速度を出すことができた．これらの性能と，記号処理の基本操作の実装上の工夫により，緊急度の高い割込みに対する反応遅延を最悪で 100 μ秒台にできる見通しがついた．

Lightweight Processes on the Real-time Symbolic Processing System TAO/SILENT

Ikuo Takeuchi,[†] Masaharu Yoshida,[††] Kenichi Yamazaki[†††] and Yoshiji Amagai[†††]

In order to provide symbolic processing systems with real-time programming capabilities, it is indispensable to implement a real-time garbage collection and an interrupt handling mechanism of small latency. The latter needs a preemptive lightweight process mechanism for its basis. This paper describes the lightweight process implemented on the real-time symbolic processing system TAO/SILENT which are now being developed by us. The TAO/SILENT process is not only lightweight but also extensible so that it can accommodate arbitrary object-oriented semantics. It is implemented by microcode of the SILENT machine (33 MHz). The shortest elapse time for process creation, activation, and inactivation is 16.91 μsec. The best process switching time is 3.93 μsec. A Fibonacci function that replaces the recursion by a process creation shows that the TAO/SILENT lightweight process is about 50 times faster than SunOS4 LWP on SuperSparc+ (60 MHz). The performance figures and some implementation techniques of the symbolic processing make sure that an urgent interrupt is passed to the corresponding process within 100 μsec or so.

1. まえがき

知的な処理に欠かせない記号処理は，実時間処理と適合しにくいとされている[8]．しかし，実世界で動くロボット，オブジェクト指向実時間アニメーション，知的なネットワーク制御などの新しい応用分野は，実世界に実時間で対処できる記号処理を必要としている．

記号処理システムに実時間性を付与するには，実時間ゴミ集め（以下，GC）と，計算時間の予測のつきにくい記号処理を応答遅延の小さい割込みによって実時間処理に適合させるという 2 つの課題がある．これらを解決しても，ハード実時間の実現は記号処理の性質からいって困難だが，ハード実時間とソフト実時間が混合した応用システムを作ることは可能であろう．

本論文は，我々が開発中の実時間記号処理システム TAO/SILENT[11]において，これらの課題のうち後者，すなわち割込み応答遅延を最悪 100 μ秒台にするために実現した軽量プロセスについて論ずる．ここで

† NTT ソフトウェア研究所
NTT Software Laboratories
†† NTT ヒューマンインタフェース研究所
NTT Human Interface Laboratories
††† NTT 基礎研究所
NTT Basic Research Laboratories

割込み応答遅延とは，割込みが発生してから，それを処理するプロセスがCPU資源をとって走り始めるまでの時間を指す☆．100μ秒という値は，種々のマルチメディア応用[9]やロボット応用で現在要求されている最も高い時間解像度1～5ミリ秒より1桁小さい．この値にすれば，緊急度の高い割込み処理が終了するまでのマージンを見込むことができる．

記号処理マシンSILENT[14]は我々が1980年代に開発したマシンELIS[5]を踏襲したものであり，記号処理言語TAOはELIS上に実現された同じ名前の言語TAO[12]を再設計した新しいマルチパラダイム言語である．

本論文は，SILENTの概要のあと，TAOのプロセスと割込みの概念と実装について述べる．さらに，プロセスの実行制御を行うマイクロカーネルの機能と実装について述べ，性能評価を示す．

2. SILENTの概要

本論文で述べる軽量プロセスは，近年のOSの軽量プロセスの研究[1),6)]と異なり，記号処理マシン上に実現される．ここでは，前提条件となるSILENTのアーキテクチャの必要部分について述べる．

SILENTは1語40ビット，うち8ビットをタグとするタグマシンである．システムバスは80ビット，2つのLispデータを同時に読み書きできる．すなわち，Lispのセルを1命令で読み書きできる．

SILENTは1語80ビット，128K語のマイクロプログラムで制御される．マシンサイクル（命令実行時間）は30ナノ秒である．

SILENTにはカーネルやユーザというモードが存在しない．マイクロコードで書かれた基本操作は必要な保護機構を内蔵している．

SILENTは動的な記号処理言語を構築しやすいように，主記憶と別に循環的なハードウェアスタック（以下単にスタックと呼ぶ）を持ち，スタックの連続プッシュ・ポップを1語1マシンサイクルで行える．現在，128K語のスタックを実装しており，128語ごとに2ビットのダーティフラグを備えている．スタックのアクセスはスタックベースレジスタ（spbas）相対の論理アドレスで行う．スタックオーバフローを検出するためのスタック境界レジスタ（sbr）を備えており，論理アドレスがsbrの値以下になると，以下に述べるhap条件が立つ．

割込みには2つのレベルがある．1つは外部デバイスとの交信やハードエラーの検出に使うマイクロ割込み，もう1つはhapという条件をマイクロ分岐命令でソフト的に☆☆検出するhap割込みである．hap割込みは通常の分岐命令による分岐であるが，hapを起こす条件は，スタックオーバフロー（spover），タイムアウト，特定の外部チャネルなど，割込みと呼ぶに相応しいものである．hap条件はソフト的に立てることもできる（simhap）．

実時間処理に対応するため，SILENTは640Mバイト（64Mセル）の実記憶を実装している．キャッシュはダイレクトマップ/ライトスルーで320Kバイト，1エントリが8セル（80バイト）である．ヒットしたときは読み出し命令のあと1クロックの空き，書き出し命令のあと4クロックの空きが入る．空きサイクルでは主記憶に関係しない命令を実行できる．

SILENTにはマシンサイクル，キャッシュヒット・ミスヒット，分岐命令の分岐・非分岐を数える32ビットの統計収集カウンタが付加されている．カウンタのオーバフローはマイクロ割込みを起こすので，任意多倍長精度で統計情報を得ることができる．マシンサイクル・カウンタは汎用レジスタと同等の扱いなので，空きサイクルを利用して値を別のレジスタにコピーしていくと，任意の範囲の実行時間を30ナノ秒の解像度で測定できる．1ステップの差が明確に結果に出るため，マイクロプログラムのチューニングに効果的であった．

3. プロセス

3.1 プロセスの概要

TAO/SILENTは言語と専用マシンの対であり，中間にはマイクロコードしかない．一般的な意味でのOSが独立に存在しない．システムコールに相当するものはTAOの基本関数として実現される．並行プロセスの実行制御の基本部分は，マイクロコードで書かれたマイクロカーネルが行う．言語から見ると，マイクロカーネルはGCと同様☆☆☆，言語仕様というよりも，言語の性能を保証する実行時システムである．

SILENTはSILENTどうし，あるいは市販の32ビットプロセッサと共有メモリ領域を持つマルチプロセッサにすることが可能であるが，ここではSILENTシングルプロセッサ上の並行プロセスについてのみ議

☆ 割込みに対する処理が終了するまでの時間とは異なることに注意．記号処理で割込み処理の終了までの遅延を保証することは一般に困難である．

☆☆ 本論文で「ソフト」というのは主にマイクロコードのことである．

☆☆☆ GCは使い捨てのメモリが仮想的に無限にあることを保証する実行時システムにほかならない．

論する．

実時間処理を実現するためには，役割に応じたプロセスを多数走行させ，必要なタイミングで必要な計算を行えるようにしなければならない．タイミングに関する条件を満たすためには，プロセスの制御が高速に行えることと同時に，プロセスが緊急度の高い割込みに対して小さな遅延で走り始めることができなければならない．このためには，軽量プロセスの実現が必須である．

また，マルチパラダイム記号処理言語が持つ高い機能性をプロセスに付与することも重要である．たとえば，自律的並行オブジェクトを，ユーザが望んだ構造でプログラムできなければならない．

TAO のプロセスは軽量で，拡張性に富むことを目指している．TAO のプロセスは一般的な OS におけるスレッドに相当する．SILENT のヒープメモリは TAO のすべてのプロセスから共有可能である．しかし，プロセスはそれぞれ固有の（保護可能な）環境を持つことが可能で，TAO の中ではファーストクラスのデータとして扱える．我々はこのような独立した実体性を強調するためにプロセスと呼ぶ．

プロセスには，4 段の特権レベル，64 段の優先度が備わる☆．セマフォに関する優先度の逆転に対応するために，簡易型の優先度継承を行う．低い特権レベルのプロセスが高い特権レベルのプロセスを操作することは禁止されている．

プロセスの内部構造は，マイクロカーネルが直接アクセスするので固定されている．プロセスにユーザが自由なオブジェクト意味論（メッセージインタフェースと内部構造）を与えることができるようにするために，TAO はプロセスに随伴オブジェクト（adjoint object）というハンドルを備えつけている．プロセスの 1 つのスロットに任意のオブジェクトを張りつけられるようにしたのである（**図 1**）．

プロセス P にメッセージが送られると，P の随伴オブジェクトへ転送される．転送の際に，メッセージの引数並びの先頭に P が引数として付け加えられる．プロセスごとに異なる構造のオブジェクトを随伴させることも，複数のプロセスに同一の随伴プロセスを共有させることも可能である．これらはユーザが決めるポリシーとして開放されている☆☆．オブジェクトが随伴していないプロセスにメッセージを送るとデフォルトオブジェクトにメッセージが転送される．

随伴オブジェクトによる並行オブジェクトの実現は，

☆ 数の大きいほうが高い特権と優先度を表す．

随伴オブジェクト*	プロセスステータス
CPU 時間*（単位 30 ナノ秒）	
コンテクスト変数シムタブ	原始オブジェクト表
動的変数シムタブ	プロセスシムタブ
イベント割込み受付	優先度継承情報*
チョアキュー*	スタック管理情報
sp 退避	プロセス番号等
スワップブロックリスト	実行スタックの状態

*のついたものと，シムタブという言葉は本文中で説明した．プロセスステータスはプロセスの性質と状態を表す 32 ビットのビットテーブル．コンテクスト変数とはプロセス内で高速アクセス可能な大域変数である．プロセス間で共有することも可能で，プロセスが休止しても束縛が持続する．動的変数は Lisp に特徴的な非局所変数である．プロセスが休止すると束縛がすべて消滅する．原始オブジェクト表は Lisp の基本データ型に対するメッセージインタフェースをプロセスごとに変えられるようにするための表．プロセス属性シムタブは，プロセスに付随した属性リストのようなもので，(外から見た) プロセスにいろいろな情報を付加する手段である（Lisp のシンボルの属性リストに相当する）．イベント割込み受付は，次々と起こるイベントボックスによる割込みを順序づけるためのもの．スタック管理情報には，本文で言及した固有スタック使用量，tickets, stackometer のフィールドがある．残る 4 個のスロットはプロセスの実行スタックの管理のためのものである．これらのスロットは，生成された時点ではほとんど空で，デフォルトを意味する．必要が生じたときに初めて，(新たにつくった) データ構造を指すことになる．たとえば，スワップブロックリストは退避されたスタックブロックを保持する主記憶内データブロックを指す（可変サイズの）ベクタである．

図 1 プロセスの構造
Fig. 1 The process structure.

システム定義のプロセスクラスを継承させる方法，オブジェクトのインスタンス変数の値としてプロセスを持つ方法に比べて，プロセスの内部アクセスが直接的なので，マイクロカーネルの効率が上げやすい．プロセスを生成するのは次の関数である☆☆☆．

```
(make-process
    [priority] [privilege] [adjoint-obj] )
```

作られたばかりのプロセスにはなんの環境（コンテクスト）もなく，また実行すべきタスクもない（プロセ

☆☆ 随伴オブジェクトを個別につけられるような基本データ型を TAO では擬似オブジェクトと呼ぶ．ベクタ類と，プロセス間通信のための有界緩衝機構であるバッファが擬似オブジェクトである．バッファにオブジェクトを随伴させると，Common Lisp の I/O ストリームをオブジェクトとして容易に実現できる．

☆☆☆ TAO は Lisp ベースのマルチパラダイム言語なのでプログラムは S 式で書く．説明の中で斜体文字は引数，角カッコは省略可能であることを意味する．

スに固有のコンテクストをどう作るか，ここでは説明しない）．タスクは次のように与える．

(**task** *process chore*)

ここでチョアは，関数☆とその引数並びの対である．タスクはプロセスの中でチョアキューに並ぶ．プロセスはタスクを順番に実行し，タスクがなくなると休止する．プロセスは優先度に基づいて実行制御され，横取りが可能である．同じ優先度のプロセスはラウンドロビンでCPU資源を回す☆☆．

3.2 プロセスの実装

データ型としてのプロセスは連続した8セルで表現される（図1）．この8セルは2進バディ領域から取られ，SILENTのキャッシュ境界に整合させてある．プロセスのCPU時間が64ビットになっているのは，計時解像度が30ナノ秒であるため，32ビットでは2分程度で溢れるからである．

図1に出てくるシムタブとは，SILENTの簡易ハッシュ命令（mash）を活用したハッシュ表である．シムタブはエントリ数に関係なく，シンボルに対応するデータを最良平均0.35 μ秒（キャッシュがすべてヒットした場合の平均，すべてがミスヒットした場合の最悪平均は1.1 μ秒）で検索できる．

優先度継承の実現のため，継承元（優先度の高いほうのプロセス）と継承の原因となったセマフォの対のリストが記録される．継承の原因となったセマフォを解放したときに，次に行うべき優先度継承を調べるためである．ただし，プロセス P_1 を P_2 が優先度継承したあと，P_1 が P_0 を継承しても，P_2 は P_0 を継承しない．すなわち，継承の継承は行わない．

優先度継承，あるいはユーザが行う動的な優先度変更に対処するため，プロセスステータスには5ビットで表現される優先度が3種類含まれている．それぞれ本来的優先度（タスク実行開始時に使われるプロセス固有の優先度），動的優先度（許容された範囲で動的に変更可能な優先度），一時優先度（優先度継承はここに反映されるが，通常は動的優先度と一致する）である．プロセスの実行制御は一時優先度に従って行われる．

チョアキューは（タスクを与えたプロセスの）優先度でソートされる．プロセス間割込み☆☆☆で与えられたチョアもここに並ぶが，タスクに優先され，現在実行中のスタックの上に割り込んで実行される．

プロセスは一般のデータ型と同じように管理される．休止状態でないプロセスを指すルートがないとGCの対象になってしまう可能性があるので，16384個のエントリを持つプロセス管理テーブルが用意されている．

4. 並行プリミティブ

TAOの並行プログラミングに関するデータ型は，セマフォ，ロッカー，メールボックス，バッファ，およびTAO独自のイベントボックスである．

セマフォは2値で`P`，`V`などが基本操作である．ロッカーは読み手・書き手問題に対処するカギ付きデータである．読み出しロック`R`とアンロック`UR`，書き出しロック`W`とアンロック`UW`が基本操作である．メールボックスは任意のデータの送受信の同期をとる無限容量の緩衝機構である．`send-mail`，`receive-mail`などが基本操作である．特権の異なるプロセスが安全に交信できるように，入力ポート（send側）と出力ポート（receive側）が区別されている．これらはすべてセル1個の大きさのデータ型である．

バッファは任意のデータの送受信の同期をとる有限容量の緩衝機構である．8セルのヘッダと，バッファ本体（ベクタまたはストリング）からなる．メールボックスと同様，入力と出力のポートが区別される．

イベントボックスはプロセスの代理人として特定のイベントの生起を待つものである．プロセスはそのまま走行していてもよい．イベントボックスはプロセスが待ちに入る任意のキューに並び得る．キューに並んだイベントボックスの待ちが解除されたときは，イベントボックスに代理を依頼していたプロセスに割込み信号が送られる．イベントボックスが扱うイベントは上記のセマフォ，メールボックスなどのほか，ミリ秒単位のタイムアウト，`raise-event`という関数で起こすユーザ定義のイベントがある．イベントボックスは，同期操作を非同期操作に変える手段を与える．イベントボックスによる割込みは`event-alert`という構文でしか使えないため，構文構造と割込み可能範囲が一致する．だから，整構造の割込み処理を可能にする．

これらのほか，割込みに関して次のプリミティブが存在する．hap分岐を無効にし，システム全体での非可分操作を実現するのが，次の構文である．

(**without-process-switch** *FORM ...*)

☆ 本論文ではTAOのマルチパラダイム性に言及しないので，述語を含めた作用素という概念を必要としない．

☆☆ GCは並行プロセスの一種類として走るが，優先度では実行制御されず，ゴミ集めの進行状況に応じた特殊な実行制御がなされる．

☆☆☆ Lisp Machine Lisp[13]の`process-interrupt`に該当するもの．強力であるが非整構造で，非同期性が強い．

ただし，時間制限があり，それを超えると致命的エラーになる（現在は50 μ秒に設定）．プロセス個々での内部的非可分操作を実現するのが

(without-interrupt *FORM* **...)**

である．

5. マイクロカーネル

マイクロカーネルは以下のことを行う．

- 走行可能なプロセスを優先度テーブルで管理し，次に走行させるプロセスを決める．バッファへの連続出力などの例外を除いて，同期操作は，それが起こした（待ちを解いた）プロセスの優先度が自分より高かった場合，制御をマイクロカーネルに預けて横取りを起こす．基本操作のほうで横取りの検査と準備を行うのは，動的ステップの節約になるからである．
- hap 割込みを受け付ける．これは TAO 処理系のマイクロコードの各所に仕掛けた hap 分岐命令の分岐先である．hap 分岐は，TAO の（**car** や **cons** などの）基本操作の非可分操作が完了してから行われる．これは，プロセススイッチによって，中途半端なポインタやデータ構造がシステム内に生じないようにするためである．実時間 GC の動作を保証するために欠かせない条件である．4 種類の hap 条件は以下のとおり．

  spover：プロセスの実行スタックが伸びてオーバフローしたとき，マイクロカーネルは実行スタック領域を伸ばす．しかし，スタックにはほかのプロセスの実行スタックが共存するので，衝突対策が必要である．

  timeout：プロセスのタイムアウト割込み待ちは絶対時間の早い順にソートされ，最も近いタイムアウトが減算タイマーにセットされる．受け付けられたタイムアウトはイベントボックスを経由して，当該プロセスに対して割り込む．

  comchan：通信チャネル（未使用）．

  simhap：ソフト的に起こす hap 条件．非可分操作の途中で，なんらかの割込み条件を生成したいとき，またはマイクロ割込みの処理をさらに hap 割込みのレベルで継続して処理したいときに使う．
- プロセスから要求されたタイムアウトのほか，500 μ秒に 1 回の周期で時間管理を行う．このときにラウンドロビンの制御などを行う．5〜10 μ秒が典型的なオーバヘッドである．
- プロセスが走行するとき，実行スタックがすべてスタック上にあることを保証する．すなわち，スタックと主記憶の間のスワップ管理を行う．これは一般のマシンにおけるメモリ階層間の記憶管理に相当するものである．スタックの特殊性のゆえに，スタック内での引越しも存在する．スワップも引越しも時間のかかる操作なので，全体を非可分にせず，「切り」のいいところで横取り検査を行う．
- 実行スタックのスタックへの割付け，および休止状態に入ったプロセスからの資源回収を行う．前者に関しては，スタックの割付け状況を見て，衝突が起こりにくい最善努力を行う．

6. スタック管理

マイクロカーネルの最も複雑な仕事は，SILENT の最も高価な資源であるスタックの管理である（全部で 1663 ステップ）．128 K 語のスタックは，512 語の大きさの 256 個のブロックに分割されている．実行スタックの底はこのブロック境界に揃えられる．スタックブロックの大きさを 512 語にしたのは，次の理由からである．

- TAO/ELIS（ブロックの大きさが 2 K 語）では，ほとんどのプロセスのスタック消費量が，待ちの状態で 200 語以下，走行中で 500 語以下であった．
- スタック・主記憶間の双方向スワップが，512 語のブロックの場合，ブロックあたり非可分操作で最大 70 μ秒かかる．割込み応答遅延を 100 μ秒台以下にするには，これが限度である．
- スタックブロックを小さくすると，スタック管理のオーバヘッドが増える．

実時間性を強く意識した応用では，緊急度の高いプロセスの優先度を高くするが，TAO/SILENT ではさらに，実行スタックを小さくすることも前提にしている．緊急度の高いプロセスをスタックになるべく常駐させるためである．TAO は優先度のレベルに応じて，実行スタックの適正使用量（スタックをこの範囲で使用すれば性能が劣化しないと想定される量）を**表 1** のように規定している．

制限が強すぎるように見えるが，TAO/ELIS の経

表 1 スタックの適正使用量
Table 1 Expected stack size of a process.

優先度	スタックの適正使用量
48〜63	1 ブロック
32〜47	2 ブロック
16〜31	4 ブロック
0〜15	16 ブロック〜（システムが設定）

験によれば，これで十分である．適正使用量の制限は性能保証のためであって，破ってもエラーにならず，違反回数がプロセスのフィールド（tickets）に記録されるだけである．応用プログラムのチューニングに有効であろう．

スタック管理は仮想記憶管理と異なり，ブロックの物理的な連続性を保証する必要がある．MMU を備えたハードウェアでは，これに対するソフト的な負荷は軽減されるが，1サイクルでプッシュ・ポップを行えるようなスタックの MMU の実装は SILENT の設計時，コストに見合わなかった．我々はソフト実装でこの問題を解決した．

休止状態でないプロセスには必ず実行スタックが割り当てられる．TAO の性能を保証するため，走行中のプロセスの実行スタックは全体がスタック上になければならない．プロセスの実行スタックは，完全な形でスタック上にある（whole）か，主記憶に完全にスワップアウトされている（absent）か，スタックブロックの一部が主記憶にスワップアウトされている/スタック内で引越し中である（wane）のいずれかの状態をとる．

スタックブロックの引越しは主記憶へのスワップアウトより高速で（ブロックあたり最大31 μ秒），かつ，起きたときのスワップインが不要なので，スタックブロックの衝突対策では優先的に考慮される．ただし，引越しが考慮されるのは最大3ブロックの実行スタックまでである．3ブロックの実行スタックを引越す場合，2ブロックを非可分操作（最大62 μ秒）で引越し，残る1ブロックを引越し未了として，横取り検査を行う．

スタックブロックの衝突には2つの場合がある．1つは，スタックオーバフロー（spover）による追突，もう1つは主記憶から実行スタックをスタックにスワップインするときの上書きである．なお，新しい実行スタックの割付け，absent 状態の実行スタックの再割付けには，LRU 法を使う．

Spover は sbr とスタックポインタ（sp）の論理アドレスの比較でハード的に検出され，hap 条件が立つ．横取りに備えて，マイクロコードは切りのよいところでしか hap 条件を検出しないので，spover のあとさらにスタックが深まり得る．だから，sbr はブロックの境界ではなく，64 語のマージンを持たせてある．

Spover が検出されると，すぐ上のブロックの空きが調べられる．空いていればそのまま実行スタックを延伸し，管理テーブルを更新する．プロセスにはスタックの最大使用量を記録するフィールド（stackometer）があり，必要ならそれを更新する．また，必要なら上述した tickets を増分する．

追突が起こったときの処置は，自分が引越す，相手を引越させる，相手のブロックをスワップアウトする，のいずれかである（最大62 μ秒の非可分操作）．

次に走行するプロセスのスワップインの間，ブロック1個のスワップインが終了するごとに横取りの検査を行う．スワップインは，同時にスワップアウトや引越しを起こすことがあるが，パイプライン処理的に同時に行われる（最大70 μ秒の非可分操作）．

以上により，衝突処理は非可分操作70 μ秒以下で行える保証が得られた．容易に推察できるように，小さなブロックサイズ（特に1）のプロセスはスタックが一杯でなければ，ほぼ確実にスワップアウトの対象外となる．緊急度（優先度）の高いプロセスの実行スタックは事実上スタックに常駐することになる．

スタック128語ごとの2ビットのダーティフラグのうち，1ビットは実時間 GC が実行スタックの印付け終了判定に使い，もう1ビットはスワップアウトの省略に使われる．

ほかのプロセスの実行スタックにアクセスするときは，マイクロ内部ルーティンがアクセスを仮想化する．

スタックブロックの空き状況と，LRU を判定するためのスタックブロックの使用状況はビットテーブルで表現される．SILENT には fbit（find bit）という命令があり，32 ビットレジスタの左あるいは右から見て最初に1の立ったビットの番号を1クロックで返す．これと64 ビット長のダブルシフト命令を使うと，連続空き領域の探索を高速に行うことができる．

LRU 判定のためのビットテーブルの仕組みはやや複雑である．256 ブロック分のビットテーブルを循環的に4個用い，4世代を表現する．いずれか1つが現世代を表す．現世代のビットテーブルは最初すべてのビットが1である．0に落ちるのは，プロセスが CPU 資源を得ているときに使ったブロックに対応するビットである．

プロセススイッチのあと，次のプロセスが走り始めるとき，現世代のテーブルの中の0の個数が全体の1/4を超えそうになると，LRU テーブルの世代交代を行う☆．こうして，現世代と3世代前までのスタックブロックの実際の使用状況が記録されていく．LRU テーブルは，実行スタックの（再）割付けのとき，スタックに空きがない場合にのみ参照される．最近3世

☆ スタックオーバフローのタイミングでは世代交代しない．これは1つのプロセスの実行スタックのブロック使用状況が世代間に分離しないようにするためである．

代のテーブルの論理 AND をとり，1 が立っているところがあれば，最近 3 世代使われなかったブロックである．見つからない場合は，参照世代を減らしていく．スタックオーバフローが頻発した最悪のケースでも現世代には 1/4 の空きがある．

LRU テーブルの更新処理は，スタックに空きがあるときも行われる．これはプロセススイッチに対して 0.63 μ秒以上の固定オーバヘッドをもたらす．

7. プロセススイッチ

プロセススイッチは `send-mail` のような同期動作や横取りによって起こる．hap 検出が切りのいいタイミングでしか行われないことと，スタックマシンであることから，プロセススイッチによって退避回復されるコンテクストはレジスタ 10 個だけである．このほか，CPU を解放するプロセスのスロットに中断時点の実行スタックの情報を書き込む手間が必要である☆1．

プロセス A がメールボックスの受信待ち状態に入り，一番高い優先度の走行可能プロセス B にスイッチするときの動的ステップ数を示そう．B の実行スタックは whole 状態であるとする．$[n/w]$ の n はキャッシュがすべてヒットしたときに必要なマシンサイクル数で，w はメモリ読み書きで生ずる空きステップのうち，埋められていないものの再掲である．実際のマイクロコードでは最適化のため，これらのステップがインタリーブされている☆2．

- プロセス A が CPU を離す準備 [12/0]
  自分の実行スタックに 10 個のレジスタと戻り番地を積む．
- A のスロットを更新 [20/3]
  sp を退避し，CPU 消費時間を加算する．
- A を走行可能プロセスキューから外す [12～19/2～5]
- A の使用済みスタックブロックを解放 [17～/1～]（17 は解放がなかったとき）
- 次に走るプロセスを決定 [12～17/1～2]
  64 段の優先度に対応した 64 ビットのビットテーブルを fbit 命令で探索し，見つかった優先度に対応するキューを見る．12 は優先度が 32 以上のプロセスで，キューに 1 個しかない場合，17 は優先度が 31 以下で，キューに複数個並んでいた場合．
- 走り出すプロセス B のスタック関係レジスタ 3 個を設定 [10/0]
- スタック管理用 LRU テーブルを更新 [21～/4～]（21 はスタックブロックの伸縮がなかったとき）
- カーネル消費時間と B の開始時間の記録 [10/2]
- レジスタの回復と B への制御移動 [17/0]

これらの総計をプロセススイッチの時間とすると最良で 3.93 μ秒となる．計時と LRU テーブルの更新に 30%以上の時間（1.26 μ秒）が使われている．ビットテーブルで優先度キューを表現するのは，簡単でかつ高速である．fbit 命令が効果的な一例である．

8. 評　　価

プロセスが軽量であることをいうには，少なくとも次の 2 つの性能評価が必要である．

- プロセスの生成・起動・消去の時間
- プロセススイッチの時間

このほかにプロセスの同期や通信の時間と，プロセスが持つ環境を操作する時間の評価が必要であるが，後者はここでは触れない．プロセススイッチの時間は，前章のとおり最良のケースで 3.93 μ秒である．

プロセスの生成・起動・休止の時間を我々は次のプログラムで測定した☆3．

```
(task (make-process) trivial-chore)
(task (make-process) trivial-chore)
(task (make-process) trivial-chore)
...
```

ここで `trivial-chore` とは，真を返すだけの無引数の関数のチョアである．このようにして多数のプロセスを生成・起動し，休止させる．このときの全経過時間から，プロセス 1 個の最短寿命と呼ぶべきものが計測できる．得られた最短寿命は 16.91 μ秒，そのうちマイクロカーネルが消費したオーバヘッドは 7.94 μ秒である．これは，`make-process`, `task`, `trivial-chore` の実行時間が合わせて 9 μ秒以下であることを意味する．

典型的な同期操作であるメールボックスを使った並行プロセスの性能は次のプログラムで測定した☆4．

☆1 伸びた実行スタックが縮んだことが認識されるのはここだけである．

☆2 GC プロセスによる横取りに言及していないが，SILENT には GC の状況によって分岐する命令があるので，ステップ数には影響しない．

☆3 参照されていない休止プロセスの消去は GC の仕事である．

☆4 `let` 構文の変数宣言部が TAO 独特の形をしているが，これは関数 `make-mailbox` で作られたメールボックスの出力ポートと入力ポートを多値（この場合，2 値）で受け取って変数 `s` と `r` の初期値にするものである．関数 `make-chore` の意味は自明であろう．

表2 mfibの測定結果（時間はミリ秒）
Table 2 The performance for mfib (time unit is millisecond).

引数	全時間	正味	OH(%)	プロセス数
9	2.95	1.64	44.5	108
10	4.83	2.68	44.5	176
11	7.93	4.36	45.0	286
12	12.89	7.06	45.2	464
13	23.70	11.74	50.5	752
14	45.12	19.26	57.3	1218
15	80.18	32.66	59.3	1972
16	135.92	54.24	60.1	3192
17	227.87	89.56	60.7	5166
18	377.20	147.33	60.9	8360
19	617.07	240.95	61.0	13528
20	1011.03	393.33	61.1	21890
21	1646.18	639.08	61.2	35420
22	2671.89	1036.73	61.2	57312

```
(defun mfib (n)
  (if (<= n 1) 1
      (let (((s r) ..(make-mailbox)))
        (task (make-process)
           (make-chore #'fs
              (list s (1- n)) ))
        (task (make-process)
           (make-chore #'fs
              (list s (- n 2)) ))
        (+ (receive-mail r)
           (receive-mail r) ))))

(defun fs (mb n)
  (send-mail mb (mfib n)) )
```

これは，よく知られた再帰的フィボナッチ関数

```
(defun fib (n)
  (if (<= n 1) 1
    (+ (fib (1- n)) (fib (- n 2))) ))
```

の再帰をプロセス生成に変え，求まった結果をメールボックスで受け取るようにしたプログラムである．**(mfib** n**)** は **(fib** n**)** $= v$ としたとき，$2v-2$ 個のプロセスを生成する．16384個のプロセスが走行可能であれば，**(mfib 19)** まで実行できる．我々は実験的に65536個のプロセスを走行可能にして **(mfib 22)** まで計測を行った（**表2**）．なお，通常の **(fib 22)** の実行時間は56.92ミリ秒である．

マイクロカーネルのオーバヘッド（OH，全時間から正味時間を引いたものの百分率）が **(mfib 13)** で急に増加するのは，ここからスワップが発生するからである．**(mfib 14)** からはスワップが頻発し，オーバヘッドが60%程度で安定するようになる．

個々のプロセスの正味時間は与えられたタスクや主記憶キャッシュの状態などによりバラつくが，プロセスを生成するものでは26～32 μ秒である．全体を通して空きサイクルが何もしないというウェイト率は26.7～28.9%，主記憶のキャッシュヒット率は93.7～89.0%である☆．

関数 **fs** の中で **send-mail** が実行されているが，プロセスの優先度がどれも同じなので横取りは起こらない．横取りが必ず起きるようにすると，スワップとプロセススイッチの回数が増える．これは全経過時間を10～6%増やし，オーバヘッドを5～2ポイント増やす．正味時間も0.6～0.8%増えるが，これは同期操作の中で横取りの検査と準備をしていることによるものである．

TAO/ELISでは **(mfib 9)** 相当しか走らせることができないが，この比較ではTAO/SILENTが530倍以上速い．この極端な性能差はTAO/ELISでは **(mfib 9)** でもスタックのスワップが頻繁に起こることに起因している．TAO/SILENT上の **(mfib 9)** はTAO/ELIS上で通常の **fib** をコンパイルしたもので走らせた **(fib 9)** の2倍強の実行時間しか要しない．ELISとSILENTのマシンサイクル比が6ということを考慮しても，これはTAO/SILENTのプロセスの軽量さを端的に示している．

次に **mfib** を適当にプログラム変換してほかのシステム上で走らせて得られた測定結果を示そう（**表3**）．比較に使用したのは **(mfib 22)** である．

Allegro Common Lisp[4]（ACL）にはプロセスライブラリの基礎としてスタックグループという概念がある．**mfib** をスタックグループを使ったプログラムに変換して測定した（GC時間は除く）．ただし，同期通信をしないプログラムになっているため，測定値は速めになっている．SunOS4 LWP[10]ではTAOのメールボックスに相当するものを用意し，忠実な形でプログラムをC言語に変換した（コンパイルは **gcc -O4**）．なお，メールボックスを用意せず，スレッドの終了とともに値を「親スレッド」に直接戻す，ACLのスタックグループに近いプログラムにすると，時間は約半分になる．KLIC[2]のプログラムはmergerを使って，TAOの **mfib** に近い計算に変換した．CML[7]は世代別GCを行っており，基本操作にGCのオーバヘッドがかかる．これを加えると2.10秒は4.66秒となる（どちらを経過時間と呼ぶかは難しい）．カーネルスレッドを使うSolarisのスレッド[3]はこのベンチマー

☆ どちらも引数が大きくなるにつれ，単調に変化する．以下の数値も同様．

表3 他システムとの比較
Table 3 Performance comparison with other systems.

システム	時間（秒）	マシン
TAO	2.67	SILENT 33 MHz
ACL	12.86	HP9000/755 125 MHz
LWP	138.6-	SS-20 SuperSparc+ 60 MHz
KLIC	0.65	SS-20 SuperSparc+ 60 MHz
CML	2.10	SS-20 SuperSparc+ 60 MHz

クでは非常に重かったため，比較の対象から外した．

KLIC のプロセスはクロックを揃えて2倍の速度という結果が出ているが，これは KLIC でいうプロセスが独特の軽い意味を持っているからである（たとえば，プロセスが過去の実行履歴を持たないため，実行スタックが不要）．CML は TAO に近いが，プロセスに特権や優先度がなく，実時間性も考慮されていない．この両者は「OS 系」のプロセスというより，「言語系」のプロセスを実現しているといえよう．TAO のプロセスは「言語系」にも「OS 系」にも使える適応性を持っている．

`mfib` ではスワップされる実行スタックが10個のレジスタ退避を含めて31語の大きさである．`fs` の中で1から n までの総和のような実行時間の短い再帰関数を実行させ，スタックが深まったところで `mfib` を呼び出すようにすると，スワップされる語数を任意に増やすことができる．こうするとマイクロカーネルのオーバヘッドは減少する．たとえば，実行スタックのブロック数が3（約1350語）に伸びてからスワップされるようにすると，上記の60%のオーバヘッドが34%になる．これは，スワップでスタックにアクセスする時間が，正味の計算のためにスタックにアクセスする時間よりも少ないからである．

9. 割込み応答遅延

上に述べたように，優先度の高いプロセスのスタックブロックの使用量は（少なくとも待ち状態では）1であると仮定されている．このようなプロセスはスワップアウトされていたとしても，最大70 μ秒でスワップインが完了する．

計算上，優先度の高いプロセスに対する割込みの最大応答遅延を大きくしないためには，hap 検査をスワップの最悪値と合わせた70 μ秒以内の周期で行えばよい．別のプロセスの走行のためのスワップイン中に割込みが発生し，70 μ秒後にそれが検出されて，緊急プロセスのスワップイン終了までさらに70 μ秒かかるというのが想定される最悪のケースだからである．しかし，我々は実使用状態での性能をよくするために，30 μ秒を選んだ☆．hap 検査の間隔を縮めることは軽量プロセスの実現とともに重要な課題である．

通常の基本操作は非可分である時間が短く，0.1～数 μ秒以内に hap 検査を行える．しかし，非可分操作の長い処理も存在する．たとえば，長い文字列の先頭に文字を挿入・削除すると，文字の長さに比例した時間がかかる．Lisp に特有の無限多倍長整数（bignum）の演算も同様である．しかし，これらは高い実時間性が要求される応用ではほとんど使わないものである．hap 検査が30 μ秒以内に行われることを保証するために，TAO は許容される最大の文字列を数百文字にするといったシステムパラメータの調整を可能にしている．

このほか，通常は非可分操作とされるものを我々は安全性を保って分解するようにした．たとえば，大きなベクタをとったときの要素の初期化の途中では，頻繁に hap 検査を行う．初期化中のベクタには（GC が中身をクリアしないので）意味のないデータが詰まっているが，初期化中というフラグを立てるので，並行プロセスとして走る実時間 GC が間違った印付けを行う危険はない．

もっと複雑な例は，ハッシュ表（シムタブや，パッケージの中のシンボルのハッシュ表）のエントリが増えたときの再ハッシュである．再ハッシュを非可分操作にすると，30 μ秒間隔以内の hap 検査は不可能になる．このため再ハッシュをインクリメンタルに行う．新旧のハッシュ表を用意し，アクセスは新旧両方を調べるようにする（登録は新しいほうに行う）．アクセスのたびに少しずつ新しいハッシュ表にエントリを移すのである．

実時間 GC の中にも比較的長い非可分操作が存在するが，通常の状態では最大30 μ秒間隔以内で hap 検査を行うことが可能である．ただし，ヒープメモリとスタックの消費率が高く，GC の負荷が大きくなった場合，100 μ秒台以内の割込み応答は困難になる．これはシステムの許容限度を超えた負荷がかかったということにほかならない．どの程度の負荷に耐えられるかは今後の実装と評価に課せられた課題である．

10. 結　　論

本論文では，記号処理システムが高い実時間性を得るための2つの条件（1）実時間ゴミ集め，（2）応答

☆ 30 μ秒の間隔であれば，スワップを途中でロールバックするような少し複雑なスワッププログラムを書くことによって最大遅延を110 μ秒に短縮することができる．しかし，現時点では未実装である．

遅延の小さい割込み，を満たす基礎となる軽量プロセスの実現と性能評価について述べた．我々が実装中の実時間記号処理システム TAO/SILENT は独自に開発したスタックマシン SILENT をベースにしているため，一般的なマシンではハードウェアで実装されていることをマイクロコードで実現する必要があった．しかしむしろ，ハードウェアと言語を直結した最適のチューニングが行えたと考える．

本論文で示したプロセス生成型のフィボナッチ関数は，プロセスの軽量性を測定する有効なベンチマークの1つである．ここで TAO/SILENT が示した性能は Unix 系の軽量プロセスの性能に比べて（クロックの不利をそのままにして）1桁以上高いと主張することができる．これはマイクロカーネルがマイクロコードで書かれていること，ベンチマークが多くのアドレスをランダムにアクセスするので主記憶のスループットが性能を左右すること，TAO のプロセスの仕様が本来的に軽い——より正確にいえば，軽量級としても重量級としても使える幅の広さを持っている——ことによると考えられる．

TAO/SILENT のプロセス管理とスタック管理は，待ち状態のとき 500 語以下のスタックしか使わない（高い優先度の）プロセスであれば，走行準備開始から 70 μ秒以内に走行できることを保証した．hap 割込み検出の遅れが 70 μ秒以下であれば（我々は実使用状態での性能を重視して，30 μ秒を選んだ），最悪の応答遅延を 140 μ秒，すなわち 100 μ秒台にすることができる．このためには軽量プロセスだけではなく，記号処理システムに特有の長い非可分操作を安全に分解するなどの工夫が必要である．これらについて 2〜3 の問題点と解決法を例示した．

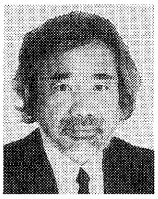
TAO/SILENT の実装は進行中であるが，本論文で述べた性能評価や実装上の工夫により，我々が目指している高性能の実時間記号処理システムが実現されるための基礎を固めることができたと考える．

謝辞 貴重な助言と情報をいただいた加藤和彦（筑波大），多田好克（電通大）の両氏，他システムの計測に協力していただいた平田圭二，佐藤孝治（NTT）の両氏，有益なコメントをいただいた査読者各位に感謝します．

参考文献

1) Accetta, M., Baron, R., Bolosky, W., Golub, D., Rashid, R., Tevanian, A. and Young, M.W.: Mach: A New Kernel Foundation for UNIX Development, *USENIX Summer 1986*, pp.93–112 (1986).
2) Chikayama, T., Fujise, T. and Sekita, D.: A Portable and Efficient Implementation of KL1, *PLILP'94*, LNCS, Vol.844, pp.25–39, Springer-Verlag (1994).
3) Eykholt, J.R., Kleiman, S.R., Barton, S., Faulkner, R., Shivalingiah, A., Smith, M., Stein, D., Voll, J., Weeks, M. and D. Williams, D.: Beyond Multiprocessing... Multithreading the SunOS Kernel, *USENIX Summer 1992*, pp.11–18 (1992).
4) Franz Inc.: Allegro Common Lisp User Guide, Version 4.2, Franz Inc. (1994).
5) Hibino, Y., Watanabe, K. and Takeuchi, I.: A 32-bit LISP Processor for the AI Workstation ELIS with a Multiple Programming Paradigm Language TAO, *J. Information Processing*, Vol.13, No.2, pp.156–164 (1990).
6) Inohara, S., Kato, K. and Masuda, T.: 'Unstable Threads' Kernel Interface for Minimizing the Overhead of Thread Switching, *7th International Parallel Processing Symposium*, pp.149–155 (1993).
7) Reppy, J.H.: CML: A Higher-order Concurrent Language, *ACM Conference on Programming Language Design and Implementation*, pp.294–305, ACM (1991).
8) Stankovic, J.: Real-time Computing Systems – The Next Generation, *Hard Real-time Systems*, pp.14–37, IEEE (1988).
9) Steinmetz, R. and Nahrstedt, K.: *Multimedia: Computing, Communications & Applications*, Prentice Hall (1995).
10) SunOS Programming Utilities & Libraries, Sun Microsystems, Inc. (1990).
11) Takeuchi, I., Amagai, Y., Yamazaki, K., Nakamura, M. and Yoshida, M.: The Multi-paradigm Symbolic Processing Kernel TAO, *J. Symbolic Computation*（投稿中）.
12) Takeuchi, I., Okuno, H.G. and Ohsato, N.: A List Processing Language TAO with Multiple Programming Paradigms, *J. New Generation Computing*, Vol.4, No.4, pp.401–444 (1986).
13) Weinreb, D., Moon, D. and Stallman, R.M.: *Lisp Machine Manual*, LMI (1983).
14) 吉田雅治，天海良治，山崎憲一，中村昌志，竹内郁雄，村上健一郎：記号処理カーネル SILENT のハードウェア構成，情報処理学会計算機アーキテクチャ研究会 114-3 (1995).

(平成8年10月9日受付)
(平成8年12月5日採録)

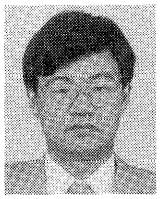


竹内 郁雄（正会員）

昭和21年生．昭和44年東京大学理学部数学科卒業．昭和46年同大学理学系研究科修士課程修了．同年，日本電信電話公社入社．現在，日本電信電話（株）ソフトウェア研究所広域コンピューティング研究部主幹研究員．主として記号処理システムの研究・開発に従事．工学博士．平成2年情報処理学会論文賞．ACM，日本ソフトウェア科学会会員．

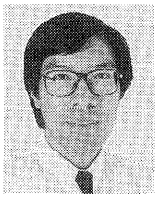
吉田 雅治（正会員）

昭和28年生．昭和51年千葉大学工学部電気工学科卒業．昭和53年同大学院工学研究科修士課程修了．同年，日本電信電話公社入社．現在，日本電信電話（株）NTT ヒューマンインタフェース研究所知能ロボット研究部主任研究員．並列処理・画像生成・記号処理等の専用計算機，知能ロボット用センサ等のハードウェアの研究に従事．ユーログラフィックス，電子情報通信学会会員．



山崎 憲一（正会員）

昭和36年生．昭和59年東北大学工学部通信工学科卒業．昭和61年同大学院情報工学科修士課程修了．同年，日本電信電話（株）入社．現在，NTT 基礎研究所情報科学研究部主任研究員．記号処理専用計算機，記号処理プログラミング言語，および大量文書検索の研究に従事．ACM 会員．情報処理学会学会誌編集委員会幹事．



天海 良治（正会員）

昭和34年生．昭和58年電気通信大学電気通信学部計算機科学科卒業．昭和60年同大学院修士課程修了．同年日本電信電話（株）入社．以来，プログラミングパラダイム，計算機アーキテクチャ，計算機ネットワークの研究に従事．現在 NTT 基礎研究所情報科学研究部主任研究員．平成6年度山下記念研究賞．日本ソフトウェア科学会会員．