記号処理 30－3
（1984. 12. 14）

SYNAPSEの機能と動作について

加藤 良信、松井 祥悟、寺村 信介、中西 正和
慶應義塾大学理工学部数理科学科

1．はじめに

我々は、数年前よりマルチプロセッサ構成のLISPマシンSYNAPSE を製作しており、昨年の第２２回の記号処理研究会［１］でもその概要を紹介した。そして今年度末には、試作システムが完成する予定になっている。本稿では、LISPマシンSYNAPSE の試作システムとその機能、製作状況を紹介する。

2．SYNAPSEの概要

SYNAPSE はマルチプロセッサ構成になっており、CPU にはモトローラ社のMC68000 を採用している。主に教育・研究用としての用途を念頭においている。図１はSYNAPSE の概念図である。L.P.U.とはLIST PROCESSING UNITの略で、CPUとローカルメモリから成っており、インタプリタ、エディタ、OSなどが組み込まれている。C.M.U.とはCOMMON MEMORY UNITの略であり、各L.P.U.はこのC.M.U.を共有のリスト領域としている。また、G.C.U.はGARBAGE COLLECTION UNITの略で、L.P.U.と並列に処理を行ないC.M.U.のごみ集めを一手に引き受けている。

L.P.U.はC.M.U.から取り外すことが可能であり、取り外して小容量メモリにつなげて使うことができる。このことにより、教育用として用いるときは多数のL.P.U をC.M.U.に接続して使い、また共有メモリのアクセスに係わるオーバ・ヘッドが気になるときは高速のパーソナル・メモリに接続する等の使い分けが可能となる。そして、研究用途で大きなセル空間が必要な場合は、もちろんC.M.U.を独占するといった使い方ができる。

将来的にはマルチプロセッサ方式の並列処理、つまり各プロセッサに一つのプロセスを割り当てる方式も考えている。このために通信機能の整備、LISPの評価関数の並列化についての研究も進めている。

3．ハードウェア

SYNAPSE の構成は図１に示した通り、大容量のC.M.U.にL.P.U.またはG.C.U.が複数個接続される。バスアービタは各ユニットから出されるアクセス要求の調停を行なう。図２はバスアービタの原理図であるが、これからもわかるようにバスアービタは多接点、多回路の切り換えスイッチであると考えることができる。各L.P.U.、G.C.U.のアクセス要求信号によりこのスイッチが切り替わり、バスの接続が行なわれる。また、他のユニットがバスを使用している時は、要求を出したユニットはバスが空くまで待たされる。MC68000 は非同期バスであるため、これらの操作が簡単に実現できる。

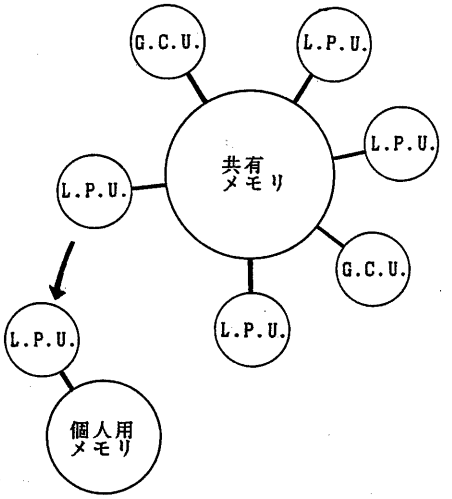

図 1 SYNAPSEの概念図

C.M.U.は512Kバイトのメモリ・ボード16枚から成る。このメモリ・ボードは64K ビットDRAMを使用し、8 ビット毎に1 ビットのパリティ・ビットを持つ。また、ボード内にリフレッシュ回路を持ち、分散リフレッシュを行う。

L.P.U.の構成を図２の示す。L.P.U.はI/O インタフェース、ローカルメモリ等から成る。I/O インターフェースには端末用としてRS-232C、プリンタ用のセントロニクス・インタフェース、ホストとの通信用として16 bitパラレル・インタフェース、そしてローカルな外部記憶装置（フロッピー・ディスク）のインタフェース等がある。また、インテリジェント・タイプのハード・ディスクなどが接続できるように汎用16 bitパラレル・ポートが用意されている。

MC68000 はスーパバイザとユーザの空間、及びプログラムとデータの空間を持っている。L.P.U.ではMC68000 の特徴を生かすため、これら

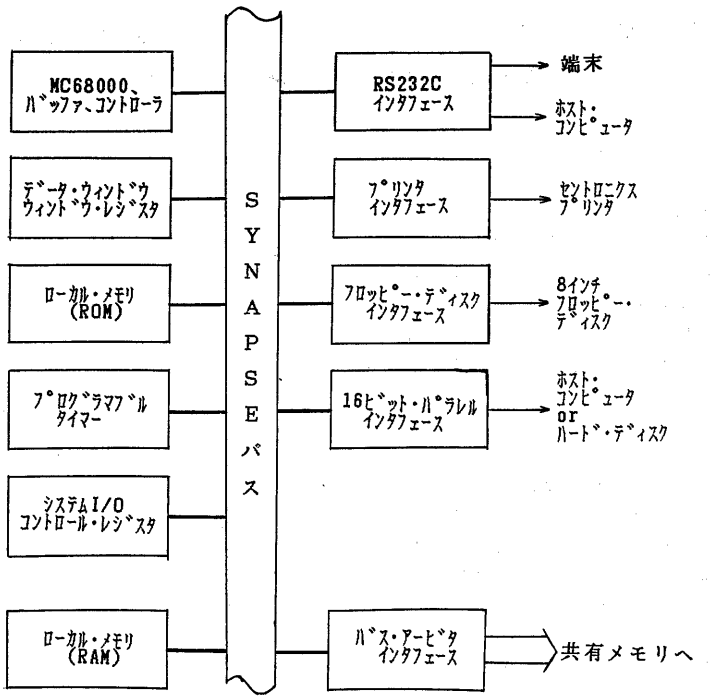

図　2　ＬＰＵの基本構成

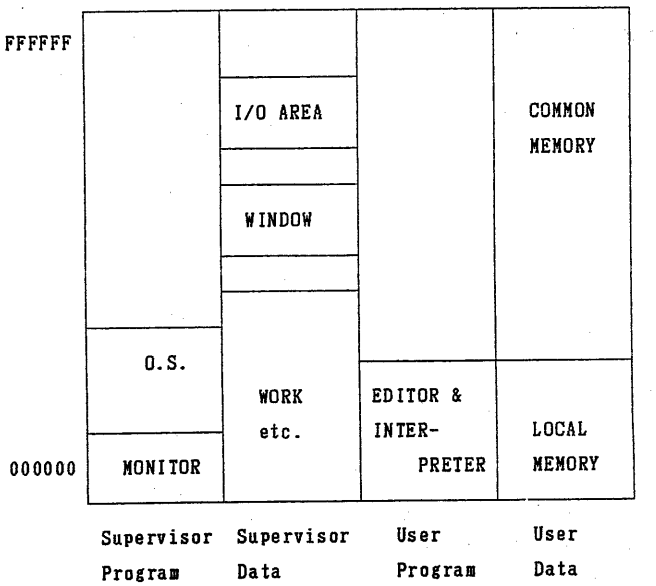

図　3－A

のメモリ空間をフルにデコードしている。このため異なるメモリ空間の間でのデータ転送が困難である。これを解決するため、スーパバイザ・モードで全空間をアクセスできるように、ハードウェアを工夫している。具体的にはスーパバイザのデータ空間に、ウィンドウを作り、このウィンドウを通して全空間がアクセス出来るようにする。この様子を図３－Ａ及び図３－Ｂに示す。

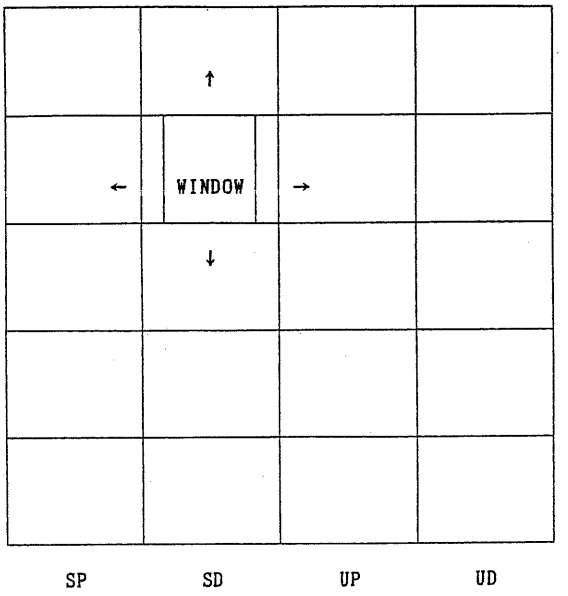

図　３－Ｂ

４．インタプリタ

プロトタイプSYNAPSE のために開発されたLispインタプリタは、慶應大学でPDP-11/60 上に作られたKLISP-11をもとにしている。ポインタは１ロング・ワードで表現され、このうちの24ビットをポインタ・フィールドとして使い、残り8 ビットを型タグとG.C.用ビットとして使用する。データ型としてリスト・セル、文字アトム、整数（fixnum）、無限長整数（bignum）があり、図４のように表現される。ガーベッジ・コレクションを行なうG.C.U.との兼ね合いのためにフルワード領域は持たず、データは一様にリスト・セルから構成されている。各データ型に対するタグの値とその判定法を図５に示す。

プロトタイプSYNAPSE -Lisp はdeep-bindingを採用し、a-list抑制の手法［２］をKLISP より受け継いでいる。この手法は、lambda-bindingのうちa-listに付け加える必要がないものはa-list中の値の部分を書き換えて済ますものである。この結果、tail-recursionは、単純なループと同じ処理になり、また、a-listの増加が押さえられるためにdeep-bindingの欠点とされる自由変数の参照の問題も軽減される。

５．ＯＳ

オペレーティング・システムは、現在のところまだ試行錯誤の段階である。ここでは、通常O.S.で行なわれる機能をSYNAPSE においてどのように実現するか、その方針について述べる。

これまでのところで明かなようにSYNAPSE のハードウェア環境は、かなり柔軟に変化する。１つのL.P.U.周辺には主記憶、二次記憶はもちろんのこと、他のL.P.U.やG.C.U.が様々な状態で存在する。こういった資源の付加や削除にSYNAPSE のO.S.はうまく対処しなくてはならない。

また、LISP処理系では対話的な操作が前提となるため、応答の時間が速く諸資源が扱い易いことが必要である。即ち、より高い機能を実現するためにユーザ・プログラムに対して負荷が大きくなるようなことは避けなければならない。

こういった要求から、O.S.の設計はオブジェクト・モデル［６］の考え方をもとに行なっている。端末やディスク装置は、データ構造とその操作のパッケージとして定義し、外部からのアクセスは特定のメッセージを通してのみ許される。O.S.は単にメッセージ・サーバとして存在する。具体的なサービス機能や保護についてはほとんどが各資源ごとにまかされている。

こういった構成法は、単純であり仕様の変更に対して柔軟であるが、メッセージの通信のコストなどいくつか問題点がある。

６．ＳＹＮＡＰＳＥのＧ．Ｃ．　ＳＹＳＴＥＭ

SYNAPSE のG.C. SYSTEM に使われているのは並列型G.C.（PARALLEL G.C.）である。並列型G.C.のアルゴリズムとしてはSteele［３］，Dijkstra［４］，Kung & Song，［５］らのものがあり、我々はKung & Song のアルゴリズムを取り上げてみた。

このアルゴリズムは双頭の待ち行列を用いたリストたどり法による印付けを行なうものであり、４種類の状態を表わす２ビットのマーキング・ビットが必要である。このアルゴリズムの基本的な考え方は、リストの構造を変えるような操作を行なった場合、印付けが必要になる可能性のあるセルをリスト・プロセッサが待ち行

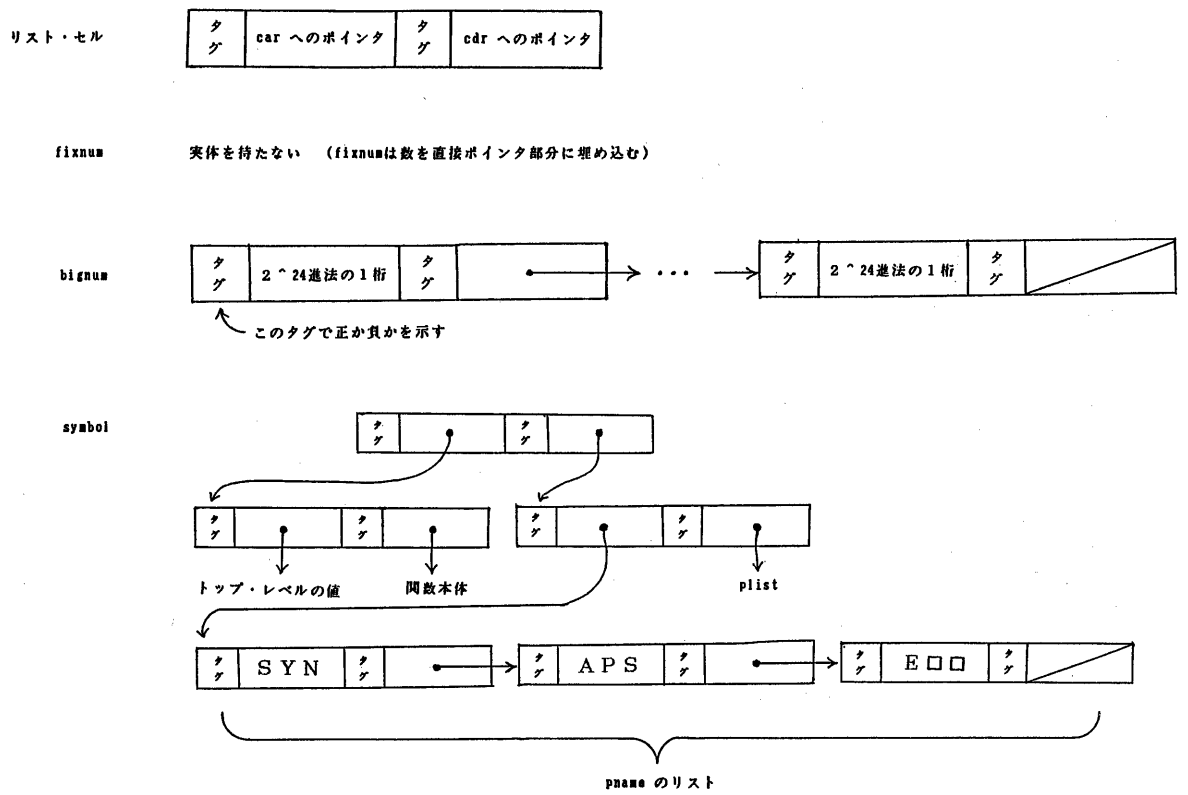
リスト・セル
タグ
car へのポインタ
タグ
cdr へのポインタ
fixnum
実体を持たない （fixnumは数を直接ポインタ部分に埋め込む）
bignum
タグ
2 ^ 24進法の1桁
タグ
このタグで正か負かを示す
symbol
トップ・レベルの値
関数本体
plist
SYN
APS
E□□
pname のリスト

図4

列に挿入するというものである。双頭の待ち行列を用いることでガーベッジ・コレクタとの競合を無くしている。また、危険部分（CRITICAL SECTION）が全く無く、印付けの効率も他のアルゴリズムよりかなり優れている。ところが、このアルゴリズムは待ち行列の実現、またその中の要素がすっかりなくなったかどうかの判定法やオーバフロー時の処理等、実現上の問題点がいくつかあってこのままではうまくいかない。

バス・アービタによる非可分操作（INDIVISIBLE OPERATION）の実現により待ち行列を使わなくても競合を避けることができる。すなわち通常のスタックで十分であり、上記の問題点の大部分を解決することが可能となった。また、この変更によりL.P.U.の複数化が容易になった。しかしこのことはセルの消費量の増大につながり、G.C.U.の負担を大きくしてしまう。複数台のL.P.U.に対してG.C.U.1台ではセルの回収が間に合わず、L.P.U.はすべて“待ち”の状態になる。これに対する解決策としては、G.C.U.の複数化とセル空間の分割が考えられる。これはメモリ・ユニットを細分化し、各G.C.U.にその部分空間のごみ集めをさせるというものである。また、L.P.U.側からはすべてのセル空間がアクセスできるようになっている。

むろん、このような変更をすればバス・アービタの負担が重くなってしまう。この負担を軽くするため、バス・アービタを増設してバスの競合を避けるようにしてやることが必要である。その他にもハードウェア・スタックの採用等、ハードウェアの支援による高速化が図られている。

ただ、むやみやたらとプロセッサの数を増やすと、やはりバスの奪い合いとなり、処理速度は遅くなる。全体の処理時間のうち、共有メモリをアクセスしている時間の割合をx、全プロセッサの台数をnとすると処理時間lnおよびメモリの使用効率enは次の式で与えられる。ただし、バスは一系統であるとする。

$$l_1 = 1 \qquad l_n = \frac{(1+x)^n+(1-x)^n}{(1+x)^n+(1-x)^n} nx$$

$$e_1 = x \qquad e_n = \frac{(1+x)^n-(1-x)^n}{(1+x)^n+(1-x)^n}$$

ポインタの上位5ビット	データ型
80	
88	リスト・セル
90	
.	
.	
.	
00	帰り番地等（データ中には現われない）
08	
10	fixnum
18	
20	
28	
30	
38	bignum
40	
48	nil, t 以外のsymbol
50	
58	t
60	
68	
70	nil

型の判定法（レジスタA0に型タグ＋ポインタが入っているとする）

```
atom:     A0  ≧  0
null:     A0  ≧  70000000
symbolp:  A0  >  40000000
numberp:  A0  ≦  40000000   （但し論理的比較として）
```

図5（数字は16進表現）

ここではバスのかちあい（conflict）が起きた場合、バスの獲得に関する優先権はなく、個別封鎖は起こらないものとして、最大（n-1）回待てば必ずバスを使えるようになっている。我々の試算ではxの値は最悪の場合で0.28、通常の使用では 0.1程度になると思われる。図6はx = 0.1, 0.2 , 0.28のそれぞれの場合についてn = 1から16までを計算したグラフである。また、上述のような仮定で簡単なシュミレーションをしてみたところ、結果はほとんど同じになった。また、システム中の自由リストがなくならないためにはL.P.U.の台数 mとG.C.U.の台数 nの間に次の関係が成り立っていなければならない。

$$nk - mr > 0$$

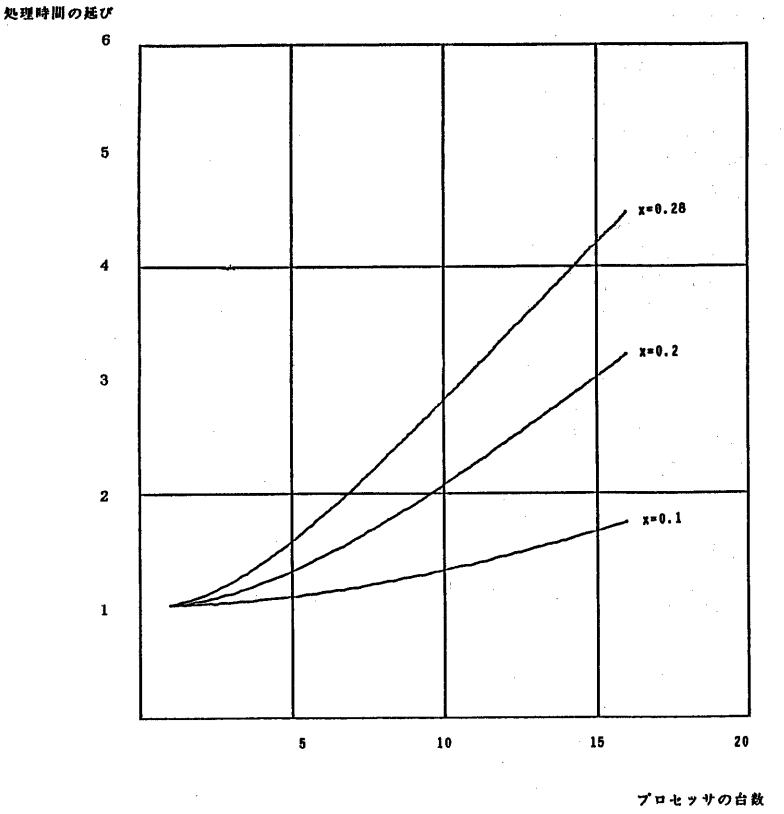

図６　処理時間の延び

ここで

k　：単位時間当たり待ち行列に挿入または削除されるセルの個数

r　：単位時間当たりのセルの生成数

である。これらの関係式から実用的なL.P.U.とG.C.U.の台数を推測することができる。

７．プログラミング環境

SYNAPSE ではテキスト・エディタを採用した。LISPがシステム記述言語である以上、LISPで他の言語処理系を実現することもある。このような場合、言語の論理構造を内部構造として持つ構造エディタでは、対象が単一の言語に限定されてしまい不便である。またSYNAPSE が開発途上であることを考慮した場合にも、対象を選ばず広く対応できるテキスト・エディタのほうが有利である。更に特定の言語（つまりLISP）に対しても専用の編集コマンドを付加することで強力なエディタとなりうると考えた。

まず、通常のテキスト・エディタとしての使い易さを重視した。コマンドはできる限り少ないキー・ストロークで入力でき、機能単位にまとめてキーに割り当てた。また、大域的なコマンド（例えば、SEARCHやSUBSTITUTE等）はインタラクティブに働き、必要事項の入力を促すメッセージに答えるかたちをとっている等、ユーザ・インタフェースの向上を図っている。

また、LISP専用の編集コマンドとしてカッコ単位のカーソル移動、カッコ単位の削除、カッコ・バランス、自動段付け等を用意した。

プログラミング環境として、将来的にはビットマップ・ディスプレイの使用を考えている。

現在SYNAPSE 本体、及び端末に内蔵出来るビットマップ・ディスプレイを開発中である。

画面のコントロールには市販のグラフィック・ディスプレイ・コントローラ(GDC NEC μPD7220) を使用する。このGDC の持つ各描写機能、スクロール機能、画面分割機能は、そのままSYNAPSE のディスプレイ装置の機能となっている。そして、ペイント、コピー等 GDCだけでは不足な機能を補うために、CPU から直接フレーム・メモリに読み書きできるようになっている。フレーム・メモリは一般のセル用及びスタック用メモリと比べて全く変わりがないので、フレーム・メモリをセル、スタック用メモリに割り当てることにより、セルやスタックの様子をリアル・タイムに観察できることになる。

ハードウェア・カーソル機能、ウィンドウ制御機能、画面重ね合わせ機能等の拡張ハードウェアを追加することにより高速な画像処理が可能となる。

８．高速化の手法について

ハードウェア構成についてはほぼ決定したので、現在の構成での高速化の可能性、その手法について考えてみる。

現在のハードウェア構成での最大の問題点はバスの使用形態である。一本のバスを複数台のL.P.U.で共有する今の構成では、当然バスがボトルネックとなる。このため高速化の手段として、第一にバスの競合の軽減を考える必要がある。具体的には、多系統のバスを導入することを考えている。

実際に多系統のバスを導入すること考えた場合、バス割り当て時のオーバヘッドが問題となり、バス・スケジューリングにある程度の制限を加える必要があろう。我々が考えた方法は、メモリを数メガ・バイト毎に分け、分割された各メモリに専用のアービタを設けるというものである。アービタを使うことにより、アクセスできるアドレス空間は今までと変わらず、かつバスの競合の軽減が期待できる。

その他の方法として、メモリ・バスの32ビット化を考えている。読み出し時は32ビットを読み出し、アービタでバッファリングを行なう。L.P.U.に対しては二回のバス・サイクルで与える。また書き込み時は、L.P.U.からの書き込み要求が二度あるまで、つまり32ビット分のバッファリングを行ない、32ビットを一度に書き込む。この方法をとると、共有メモリのアクセスは必ず32ビット単位で行なうことになるが、LISPの場合ほとんどがロングワード・アクセスなので、あまり問題は起こらない。

９．おわりに

本稿では、LISPマシンSYNAPSE の現在の状態を紹介した。現状では、動き始めてはいるが全体的な評価はまだ全くできていない状態である。今年度末には評価を終え、新バージョンの設計に入る予定である。

参考文献

［１］中西　正和　ほか、“マルチCPU･LISP処理系 SYNAPSEについて”、記号処理研究会資料 22-3、1983年 2月

［２］菱沼　千明　ほか、“LISPインタプリタにおけるスタック技法とａリストの抑制法”、情報処理、17(11)

［３］Steel,G.L.:Multiprocessing compactifyinggarbage collection,Comm. ACM,18(9),pp. 495 -508(1975)

［４］Dijkstra,E.W. et al.:On-the-fly garbage collection : An exercise in cooperation,in Lecture Note in Computer Science, 46, pp.43-56. Springer-Verlag,New York (1976)

［５］Kung, H. T. and Song, S. W. : An efficient garbage collection system and its correctness proof, Proc. 18th Annual Symposium on Foundation of Computer Science, pp. 120-131 (1977)

［６］M.J.Flynn et.al : Operating Systems. An Advanced Course. Lecxture Notes in Computer Science, 60. Springer-Velag, New York (1978).