時分割共同利用システム——ETSS について*

淵　　一　博**

1. はじめに

ETSS† は，通産省において行なわれている超高性能電子計算機の開発プロジェクトの一部として，すなわち，その基礎的研究実験の一テーマとして，電気試験所で開発された TSS である．

ETSS は，昭和41年夏に計画され，43年春に一応の完成をみた．その後，現在にいたるまで，連日の運用実験に供されている．その間いくつかの改良が加えられた．

ここでは，ETSS の基本的構成を紹介するとともに，ETSS での問題意識，構想や，その再検討などについても，基本設計担当者として，その主観的評価も含めて，報告することにしたい．

2. ETSS 発足時の技術的状況と問題意識

2.1 技術的状況

ETSS が企画された昭和41年夏における，TSS に関する技術的状況を振り返ってみよう。

アメリカにおいては，CTSS (Compatibe Time Sharing System-MIT) をはじめとする TSS 開発の蓄積があり，TSS ビジネスもはじまろうとしていた．当時進行中であった MIT の Multics プロジェクトは，その技術的ピークと目されていた．

わが国でも，TSS の重要性についての論議が高まりつつあり，メーカにおける TSS 開発計画の発表や，いくつかの研究グループの結成があった．

TSS ブームの一方では，TSS 批判も紹介され，それが一面において正当であることが納得されていた．

しかし，国内における TSS 開発の経験が皆無であったため，わが国の論議は観念的であらざるをえず，そのかぎりでは水準は高かったにしても，したがって，また偏位も大きく，TSS に対する過大要求とともに，TSS 開発の困難性（わが国における不可能性）についての神話的状況が存在した．

この年の8月に来日した MIT の J. H. Saltzer 助教授の Multics についての連続講義[1] は，TSS の内部構造を紹介したことで，わが国の計算機技術界に大きな刺激を与えた。これをきっかけに，わが国における TSS 論議も，外面的議論から技術の内面に焦点がむけられるようになった．

2.2 問題意識

ETSS は以上のような状況のなかで出発したものである．そこでの問題意識と基本構想は，おおよそ次のようなものであった．

まず（少なくともわが国において）TSS は，まだ商業生産のレベルに達していない．そのレベルまで持ち上げるためには，TSS 批判の提起する論点を含めて，多くの技術的問題が明らかにされなければならない．それには実験が必要である．

そして，そのための実験としては，専用的†† TSS への“後退”ではなく，あえて TSS の正統的イメージに固執してみるべきである．とくに，会話，汎用処理，拡張性は，TSS 批判の集中するところであるゆえに．

そこで，ETSS のねらいとしては，

（1） 正統的 TSS イメージを実現すること　すなわち，会話中心であること，オンライン・ファイル中心であること，汎用処理を中心とすること．

（2） TSS の実験として　最小構成，最小機能でよいこと，拡張性を重んじること，モジュール化を徹底すること（ある程度能率を無視してよいこと）．

一方，TSS を含めてオンライン・システムの形態の多様化の傾向を見とおして，システム開発の“生産性”の問題の認識を根底におくべきである．したがって，ETSS の開発方針として，

（1） システム・イメージの透明な構成法

（2） システム制御の記述方法のくふう

（3） デバッグ・システムのくふう

をたてる．

このような実験として，システム（プログラム）は

---

* On a Time Sharing System——ETSS by Kazuhiro Fuchi (Electrotechnical Laboratory, Computer Division)

** 電気試験所・電子計算機部

† ETL Time Sharing System, E はまた Exprimental, Expansible などのつもりでもある．

†† 特定の（高水準）プログラミング言語を固定するという意味において．

すべて自前でつくる．メーカ提供の“バッチ”OS は開発手段としてのみ用いる．

内面的（技術的な）問題意識としては，以上のようなことをもとにして ETSS は出発した，ということができる．

後述するページ方式，シミュレーション方式の着想は，ETSS の出発点においてあり，それが ETSS プロジェクトの推進力の一つでもあった．

2.3 Multics の影響

前述した J. H. Saltzer 氏の連続講義によって，Multics はわが国の TSS（わが国にかぎるわけではないが）に大きな影響を及ぼした．ETSS 自体，その出発と Saltzer 氏招待は密接な関連があったわけであるが，ここではもっとも直接的な関連についてだけ触れよう．

ETSS の開発方針の一つである，システム・イメージの構成法については，シミュレーション言語のシステム記述能力に着目し，SIMSCRIPT 流のイベントによるシステム把握と，GPSS 流のトランザクション的とらえ方の2方向を検討していた．Multics で示された“プロセス”は，後者の流れに属するものであるが，すでに明快に整理されていたことから，ETSS にもそのとらえ方を採用することにした．

第2点の制御の記述について，Multics は PL/I（サブセット）を用いている．ETSS でも，基本的には，そのアプローチを採用したが，コンパイラ作製の見通しの暗い状況を逆用して，PL/I の一種のスーパーセットである APL[2] を利用することにした．

Multics の眼目の一つは，ページ・セグメント方式（二次元アドレス方式）であるが，ETSS では使われたハードウェアの制限により，それと直接関連をもたせることがなかった．

TSS の基本的ユティリティとしての“エディタ”は，CTSS のコンテクスト・エディタを参考にした．

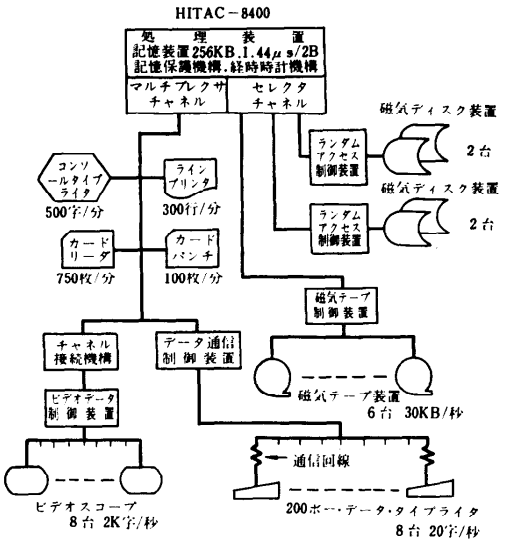
3. ETSS の基本構成[3]

3.1 ハードウェア構成

ハードウェアは，以下の点を除けば，商用の標準的な装置〔HITAC 8400 システム（**第1図**参照）〕によって構成されている．

（1） 経時時計機構：分解能 100 μs のものを追加

（2） 記憶保護機構：標準のものは，2,048 バイトのコア・ブロックに対して4ビットのキイを付随させ，write 操作についての保護を行なっている．ETSS 用



第1図 ETSS ハードウェア・システム

として，さらに read（+execute）操作に対する保護機能を追加し，これを利用して後述の“ページング”を行なう．

（3） 端末装置：一群はビデオスコープ（ビデオ・データ問合せ表示装置）でチャネルに直結している．他の群は新規開発（沖電気製）の 200 ボーのデータ・タイプライタで，通信回線を介してチャネルに接続される．

3.2 プロセスの切換えとモジュール・ルーチンのコール

ETSS では，“複数個”のイメージ・プロセッサを単一の与えられたハードウェア上でシミュレートする，ということをソフトウェア建設の指導方針にした．その中心となるのは“プロセス”というイメージである．いわゆる割込制御ルーチンなどは，このプロセス制御（モジュール）の中に，その中核部として吸収される．

プロセスの基本状態として，

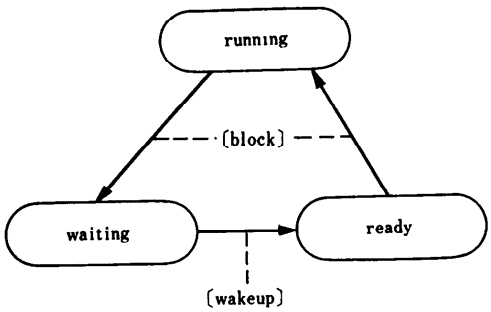
（i） running（走行中）

（ii） ready（走行可能）

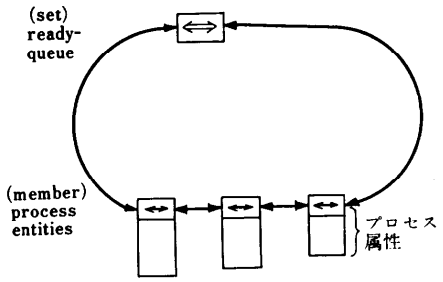
（iii） waiting（待ち）

の3状態があり，その状態遷移の制御が，プロセス制御の基本となる（**第2図**）．

プロセスの制御のために各プロセスには，その性質を収納するための記憶領域が付随しており，process entity と呼ばれる．状態の制御は，process entity をリストでつなぐことによって形成される set（集合）を利用して行なう．ready 状態にあるプロセス（群）



第 2 図　プロセスの状態と切換え基本命令

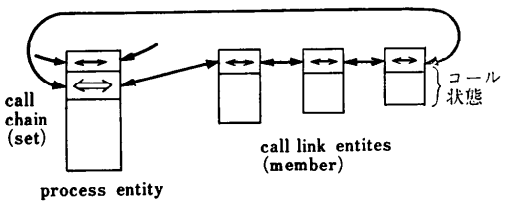


第 3 図　プロセスの待ち行列

は ready-queue と呼ばれる set を形成している（**第 3 図**）．プロセスの複合されたユーザ向けのイメージが後述の GPP である．

プロセス制御に関連して，re-entrant なサブルーチン・コール（SVC 命令による割込みを介する）の制御が行なわれる．これは“縮退した”プロセスの制御と見たてることもできる．コールに際しては，process ertity の代わりに calllink entity と呼ばれる領域がとられ，ここに制御に必要な情報が収納されるとともに，LIFO* 的にリストでつながれる．これがコールのためのプッシュ・ダウン・スタックを形成している．このスタックは callchain set と呼ばれ，プロセスにその属性として付随する（**第 4 図**）．

ETSS では，同じ capability（権能）をもった“サブ”ルーチンのクラスをモジュールと呼ぶが，このコールの制御はまたモジュール間の capability の制御であるといってもよい．



第 4 図　コールのチェーン

* last-in-first-out（後入先出）

プロセス制御を使いやすくするために，プロセス制御の基本命令（wakeup と block）を組み合わせたイベント・システムがつくられている．イベントの発生 (cause) 命令とイベント（の集合）に対する wait 命令がある．

これらの部分のイメージ設定には，SIMULA, SOL などのシミュレーション言語に体現されている諸概念が，Multics の構成とともに参考になった．

3.3 GPP のイメージとページングの新方式

ETSS ではユーザに対して仮想的計算機が 1 台与えられる．これを GPP (General Purpose Processor—汎用プロセッサ）と呼ぶ．これはイメージとしては，ベースのハードウェアである H 8400 によく似た機械で，ユーザは任意のプログラムをここで実行することができる．

GPP は，“非特権”機械語命令の直接実行に加えて，つぎの諸機能をそれぞれがもっている．

（i）　モードの切換え（割込み）機能
（ii）　ファイルに対する入出力チャネル機能
（iii）　コンソール機能（入出力および割込み）
（vi）　経時時計機能
（v）　記憶保護機能（2048 バイト・ブロックに対するロック）
（iv）　例外処理機能

この機械は，

（i）　仮想的なレジスタ類
（ii）　仮想的なコア・メモリ

によって構成され，GPP スーパバイザ（プログラム）が，この仮想的計算機をシミュレートしている．

とくに，仮想コアのシミュレーションは新しい方式のページングによって行なわれる．ETSS のページング方式の特長を述べるには，通常ページング方式と呼ばれているもののもつ機能を分析してみる必要がある．

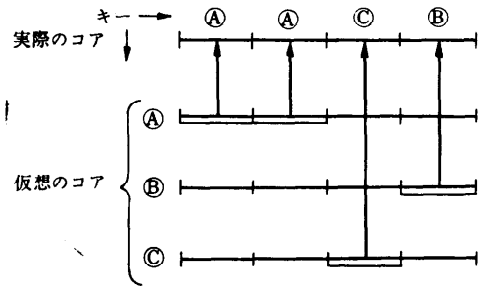
通常の方式には，実はいくつかの独立した機能が“複合”されているのであって，それらは，

（i）　本来の意味（固定長ブロックに分割してスワップ・インアウトする）でのページング
（ii）　記憶内容の（ページングを介しての）動的再配置
（iii）　セグメンテーション

である．これらは相互に関連しあっており，いずれもTSS 用の記憶装置としての有用な機能であるが，しかし，分離しえないものではない．

これらの諸機能を全体として実現するためには，ぜいたくなハードウェア付加装置が必要になる．ETSSにおいては，上記の第1項の機能だけを取り上げ，以下に述べる，ハードウェア追加のほとんどない方法でページングを実現している．

その原理は，既述の ETSS オプションの記憶保護機構を制御機構に転用する点にある．すなわち，保護キーをプログラム番号(より正確には仮想コアの指定)と読み換えて制御する（**第5図**）．



第 5 図　ETSS ページングの原理

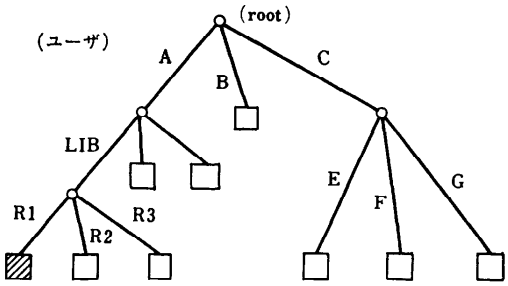
仮想コアの内容（磁気ディスク上にある）はページ(2048 バイト）単位で実際のコアに動的に移される．プログラム側の番号（キー）とコア側の番号（キー）が一致している間は，命令の直接実行が進行する．プログラムの発生するアドレスが，キーの異なる領域を指しているということは，そのプログラムの実行に必要なページがそこにまだ移されていないということを意味する．この状況が生ずれば既述の保護機構が働いて "missing page fault" 割込みが起こる．これを受けて GPP スーパバイザがページの入れ換え (swap) を行なう．

3.4　ファイル・システムの構造と制御

ETSS のファイル（の集合）は，ディレクトリによる木構造によって管理されている．これはユーザ・ファィル集合を（システムとして）管理するだけでなく，ユーザ自身による自己のファイルの階層的管理をも許す．

ファイルは qualified name によって指定される．たとえば**第6図**で，ユーザ A は，LIB. R1 によって黒印のファイルを指定する。

この構造の上に ETSS 独自の link-joint 機構により，ファイル共用のパスをつくることができる．共用



第 6 図　ファイルの木構造

のパスにはファイルに対するアクセス権が付随し，これによって指定されたファイルへの操作（の種類）を限定する．このことによって共用に伴うファイルの保護を行なっているのである．

joint はファイルの差出し人がつくり，それに対して受けとり人が link をつなぐ．joint は Multics におけるユーザ・アクセス・リストを分解したものと見立てることもできる。この joint-link 機構は共用パスの病的なループ現象を禁ずる働きももっている．

各ファイルは，磁気ディスク上にトラック単位で動的に割りつけられる．ファイルに割りあてられたトラック（番号）はマップ（ファイル）によって管理される．

未使用のトラックは，自由トラック・ファイルに登録されている．

3.5　デバイスの管理と制御

デバイスは，コンソール（端末機器），磁気ディスク・センタ内 I/O 装置に大別される．

コンソール・デバイスには，ビデオ端末，データ・タイプ端末，システム・コンソール・タイプの3種類があるが，それらに対して共通のイメージ（ロジカル・ターミナル）が設定されている．各デバイスの個性を吸収して，この共通イメージにあわせる働きをするのが，device interface module（群）である．

コンソール（端末）については入出力機能のほかにコンソール割込み機能がシミュレートされている。またコンソール機能とデータ入出力機能の重ね合わせもロジカル・ターミナルのレベルで共通に制御される．

磁気ディスクについては，チャネルとデバイスに関して，並列制御ならびにアーム・シーク時間を吸収するためのエレベータ方式の制御が行なわれる．エレベータ方式は，磁気ディスクに関する要求待ち行列をシリンダ番号順に並べ換え，全体としてシーク時間最短（に近い）の順でサービスを行なわせようとするもの

である.

通信制御は，device interface の“大がらな”一module としてつくられている．最初，制御の簡単なチェック・オフ方式がつくられたが，ハードウェア機能の整備をまってチェック・オン方式が具体化された.

3.6 ユーザの管理とコマンド

ユーザとシステムの会話は，ユーザが端末コンソールから割込みをかけたときにはじまる．その最初の login シークエンスでユーザは名前と合言葉 (pass word) を与え，その結果，ユーザ，コンソール，ファイルのユーザ枝の間の対応がつけられ，またコマンド・サービスのためのプロセスが生成される.

ETSS のコマンドには基本コマンドと複合コマンドがある．後者は前者の組み合わせとしてユーザが任意に定義できる.

基本コマンドには，

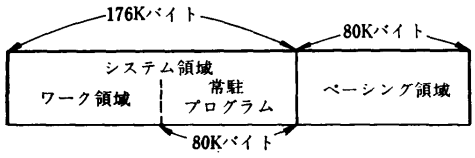
(1) ファイルに関するもの
　listup, create, delete, joint, link, change
(2) ファイル内容の編集に関するもの
　edit
(3) GPP の制御に関するもの
　execute (=loadgo), restart, halt, interrupt
(4) 複合コンドに関するもの
　define, delete command

などがある.

複合コマンド（一種のマクロ）の定義は，entry, exit コマンドでくくられ，前者ではパラメータの指定ができる．定義の中味は，基本コマンドのほかに制御コマンドとしての if コマンド，goto コマンドなどによって構成される.

3.7 システムの具体化

前述した諸機能は，その他の機能とともにシステム・プログラムとして主記憶装置に常駐させている．また基本的なユティリティとしての“エディタ”も常駐にしてある（これは SPP-Special Purpose Processor と呼ばれる)．これはエディタの使用頻度をみこしたものである.



第 7 図　主記憶領域の分割

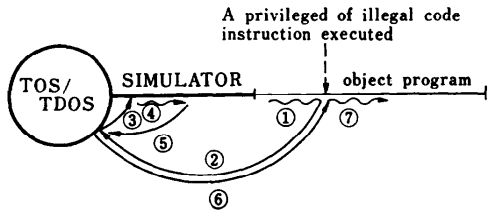
主記憶領域は常駐プログラム領域，システム・ワーク領域，ページング領域に分割されている（**第 7 図**).

3.8 システムの開発の補助手段

システム・プログラムのデバッグ用に，部分的解釈法[4] に基づくプログラム・シミュレータを作製した.

システム制御プログラムは，システム制御のための諸命令を含んでいる．ところで，通常バッチ OS の下でのユーザ・プログラムは，それらの命令（特権命令と呼ばれる）の実行を禁じられている．この保護機構を逆用すれば，特権命令を含んだままの（システム）プログラムのためのシミュレータが作製できる.

すなわち，システム・プログラムをそのままユーザ・プログラムとして直接実行してしまう．特権命令にさしかかると OS の set exit 機能により，（誤りの一種として）ユーザの（誤り）処理ルーチンに割り出されてくる．ここで，その特権命令を，シミュレートされた処理装置（ならびに入出力機器を含むシステム）のイメージの上で，シミュレートすればよいわけである（**第 8 図**).



第 8 図　TOS/TDOS SIMULATOR object program

このシミュレータは，テストすべきプログラムの形態を（ほとんど）変えず，かつ大部分の命令は直接実行されるから，通常の（フル）インタープリータ方式に比べて，非常に高速である．したがって，（システム）プログラム全体に対して経済的に適用することができる.

このシミュレータの存在により，実物上でのプログラム・テスト（ならびにデバッグ）の期間を大幅に減らすことができた.

4. ETSS の開発経過

(1) **前期**（昭和 41 年 9 月）
基本構想の形成．ページング，シミュレータの着想．ハードウェア構成の決定．(J. H. Salzer 氏講演)

(2) **第 1 期**（昭和 41 年 10 月～昭和 42 年 3 月）
プロジェクト・チームの結成．研究員は 16 名，そ

のうち10名は，昭和41年度採用の若い研究者であった．H 8400 ならびにプログラミング講習．

昭和42年2月に H 8400 納入．ビデオ端末による会話型ルーチン“エディタ”の実験．

詳細設計の一部．

（3） **第2期**（昭和42年4月～11月）

研究員22名に増加．詳細設計とプログラム製作．11月に，ETSSの中核モジュールによって，複数ビデオ端末で，“エディタ”，ゲーム“DUEL”の SPP が稼動．

（4） **第3期**（昭和42年12月～昭和43年4月）

GPP スーパバイザ稼動．データタイプライタ（回線制御）用ルーチン完成．昭和43年2月に．“ロジカルに”ETSS が稼動しはじめる．アセンブラ完成．

性能面での改良をはかり，速度で約10倍改善される．4月に，東大，通研，電試（永田町）に専用データ伝送回線が設置され*，遠方使用が可能になった．

（5） **第4期**（昭和43年5月～）

ETSS の運用実験開始．毎週月曜から金曜まで，毎日4時間運転（現在まで）．

ファイル安定化のための手当を行なう．ETSS の動作の測定と解析がはじまる．その結果に基づいて改良を加え，総合的に約3倍の性能改善．ユティリティ・プログラムの蓄積．

一方，マルチ・プロセッサ（2台）への拡張（昭44年1月納入）のためのプログラム製作が行なわれ，昭和44年10月，実験的に成功．

5. ETSS の基本設計開発上の諸問題の反省

昭和44年5月以降の，ETSS の改良状況，測定解析のデータ，動作状態の観測から，いくつかの反省を行ないたい．

ETSS の基本構成は，それが TSS 商品を目指すものでなく，実験システムであって，TSS およびそれに関連する計算機技術の問題点を浮きぼりにすることをねらいとして，設計されたものである．

したがって（そのねらいに十分合致した設計か否かは別にして），商用システムの設計者の目からは奇異に見えるかもしれないことを，あえてしている面が多い．

はじめに，ETSS の独自な点であるページ方式とシミュレータについて検討し，あとシステムの問題や改良点を検討する．

* 電電公社技術局のご協力による．

5.1 ページング方式について

ETSS ページング方式の着想は，M. V. Wilkes [4]が提案した slave memory の使用方式[4]にヒントを得たものである*．

このページング方式は（もまた），ページ・スワップのスケジューリングのアルゴリズムが効率に大きな影響を及ぼす．ETSS で現在使われているアルゴリズムは，単純にデマンドごとにスワップ・インするものであるが，これではスワップの振動現象が比較的早く生ずる．

ページの滞在時間を長くするアルゴリズムが提案されている（まだ具体化されていない）．これを勘案して，理論的，シミュレーション実験的に，その後検討されたところによれば，それは，sophisticated なページング方式に劣らない特性を有しうることが明らかにされ[5]，筆者の着想時の直観的予想が実証されつつある．

ETSS のページ方式は，ハードウェア改造の微小なことから，妥協的方式とも見られがちであるが，上の結果によれば，むしろ大型機にも適した，実用的な方式ということができると思う．

なお，現在使用されている記憶保護機構は，キーの異なるブロックへのアクセスを禁ずる方式であるが，それに加えて，書き込み禁止（場合によっては実行禁止）を独立したモードとしてもつよう拡張すれば，ロジカルに，よりスマートなイメージが構成できる．

5.2 部分的解釈によるシミュレータについて[6]

ETSS の完成が，もし予想より早かったとするならば，一つにはこのシミュレータの効果があろう**．

（特権命令）保護機構の，制御機能としての利用は，ページ方式における（記憶）保護機構の利用とともに，よい保護機構はよい制御機能でもありうる，ということを示している．

現実には，このシミュレータの実現の際，与えられたバッチ OS に若干の難点があった．それについての改良は，OS をむしろスマートにする方向にあった．このことは，この種のシミュレータの発想が自然なことを示していると思う．

この種のシミュレータは，オンライン・システム用 OS の多様化の傾向からすると，OS 開発の基礎的手

* この slave memory の方式はまた，ETL Mk 6 で筆者自身の提案した“プログラム・スタック”方式を参照している．

** 実際には“ティーチング・マシン”としての役割が大きかったと評する人もいる．とすれば，現在，（制御）システム・プログラマの養成は by chance であるが，その種のプログラマ養成のための実習システムとしても役だつだろう．

段となるものであろう*.

5.3 システム（プログラム）記述について

ETSS の問題意識の一つであった，システム（プログラム）の記述の方針は，APL の採用によって，実験的にはある程度の成果を上げた．しかし APL がシステム記述言語として最適というわけではなく，効果的なシステム記述言語の設定は今後の課題に残されている．

APL の使用は（コンパイラはなかったが），システム開発手段として効果的であった．

システム・プログラム用言語として，Multics 以後 PL/I の“サブ”セットを設定することがわが国で流行しているが，システム用言語のあるべき姿について，コメントを加えれば（コンパイラ問題はひとまずきりはなして），それは“サブ”セットであるより，むしろ，“スーパー”セットに向うべきだと思う．

ETSS では，APL だけでなく，（複合）コマンド処理について SNOBOL（これもコンパイラはないが）をプログラム設計に利用して有効であった．

現段階でまずいえることは種々の（高水準）プログラミング言語（に体現されている概念）を，つなぎ合わせて使ってみる実験が，現実のシステム開発の際にも実際有効であり，また将来のシステム記述言語への素材を提供するだろう，ということである．

5.4 ハードウェア構成上の問題

ハードウェア構成で問題になるのは，まず，スワップ用として磁気ドラムを用意しなかったことであろう（マルチ・プロセッサへの拡張もその後行なわれたがそれについてはいまは触れない）．

磁気ドラムのないことには最初から批判があったが，一つには財政的問題があり，また，ETSS の問題意識が，直接そこになかったこともある．

すなわち，事後的にいえば，（磁気ドラムの導入はシステムの性能の観点からいわれるのであるが），システムの性能のネックは，ETSS の場合には，そこにはなかった．

ETSS では，ファイル主媒体である磁気ディスク上のファイルに対して，直接スワップ・インアウトしている．しかし，既述したシステムのスピード・アップは，主として他のモジュールの改良の結果であり（これについてはあとで触れる），またページ・スケジューリング・アルゴリズムの改訂を行なえば，あと数倍の性能向上が予想される．

すなわち，ETSS は，本質的なディスク・リミットにまだ達していないことを示している*.

5.5 磁気ディスクの使用法

現在，磁気ディスクは可変長レコードの様式で使われている．すなわち，ディスク制御装置のもつ機能をほとんどすべてユーザに開放している．

基本設計の段階で，固定長（ページ・サイズ）制御との選択に迷った．結局，ディスク制御装置のもつ機能（可変長レコードサイズ，キー捜索機能）に色気を出して，上記の方式にした．しかし，これは結果的にどうであったか．

ディスクにおいても，ブロッキングの効果は大きい．しかし自由を許したために，ユティリティ・プログラムにブロッキングを強制することにならなかった．

機能は殺しても固定長方式にし，制御を単純化し，かつブロッキングを強制した方が，効率面でかえってよいようである．

5.6 システムの改良について

システムの一応の動作開始後，いくつかの改良が行なわれて，性能が大幅に向上した**（まだ十分ではないであろうが）．そのいくつかを検討しよう．

一つはファイルの安定性についてである．システム・プログラムに虫がいる状況で運転し，システム・ダウンを繰返すと***，ファイル状態に“みだれ”が少しずつ蓄積され，それがしばらくたって，あとのダウンの原因になる，という状況が発生した．

これには，ダウン時点での，ファイル状態復帰の手順の設定と，ダンプ・リロード機能の具体化がなされた．この後，ファイルの保管はほぼ確実になった．これは，もとはといえば，基本設計に不備があったからだということができる．

他の改良点はスピードの向上である．システム（プログラム）の性能については，基本設計時の予想に対して，昭和 43 年 5 月の段階ではプログラム・サイズで

---

* ユーザ（あるいはソフトウェア・メーカ）メイドの OS は今後の一つの方向でなければならないが，そのためには，ハードウェア自身とハードウェアメーカに，あと若干のエレガンスが要求されよう．

* ということは，しかし，磁気ドラムを使用しても結果的な効率があがらなかったであろう，ということではない．その場合は，見掛け上の効率から，問題がシールドされたかもしれない，ということである．もし，ETSS をベースにして，実用システムを再作製するとすれば，磁気ドラム使用が必要であろうし，また本質的にも有効になるであろう．

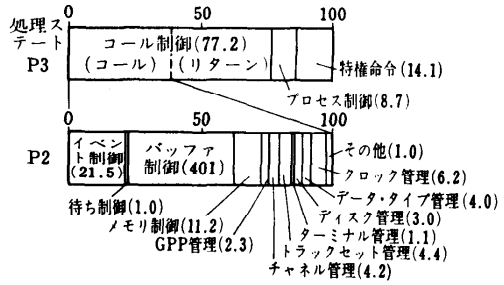
** これは自慢になることではないが，しかし，“チェーン・アップ”の効果が大きいことについての事例が多くなり，チェーン・アップをせざるを得ないこと，そうすべきことの認識が確立してきたようである．そこにもシステム“実験”の意味がある．

*** プログラムの虫だけでなく，ハードウェア自体の故障による回数もかなりあった．

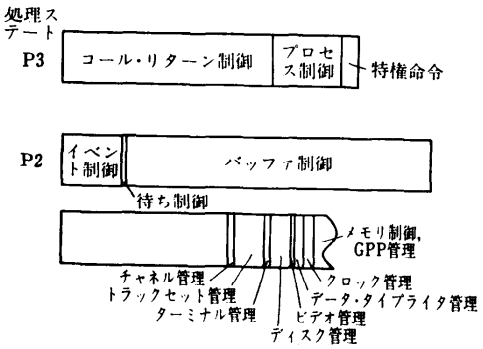
2倍，スピードで10倍の差があった*.

その後の改良の状況を現時点での測定データ[7]をもとに検討しよう（**第9図，第10図**）**.

まず，第9図からわかるように，コール・リターンの頻度が高い．ETSS のコールは（SVC 命令割出しを介する），システム・プログラム間の連絡用で，したがって多用されているわけである．第10図のコール・リターン制御時間はすでに改良が行なわれて約 1/4 に短縮された結果である．改良前は従って，それがシステム速度を規定する最大の要因であった．この改良はコーディング技術的に行なわれた．プログラムも短くなった．



第9図　ETSS 基本機能の使用頻度のバランス



第10図　使用頻度×処理時間を測度とした ETSS のバランス

コールについて行なわれるべき他の点は，頻度自体を減らすことであろう．入出力動作の解剖学的解析から，イベントに経由するコールを相当減少させることができることがわかり，その改良で命令によっては CPU 時間について約2倍に性能が上がった（第9図，第10図はそのあとの状況である）．

なお抜本的には，システム・プログラムの安定を待って，コールを（割出し機能を通さず）直結することである*.

第10図のプロセス制御の部分の時間は，まだ数倍の改良の余地を残している．

現段階の最大の問題は，バッファ制御の部分にあることが第10図からわかる．頻度もさることながら，単位時間が大きい．これについては，数分の一に改良する作業が現在進行中である．

このバッファ制御は，可変長バッファ制御であり，それ自体としてある程度時間のかかるものではある．頻度の点からは，現在ほとんどのモジュールが，いちいち可変長バッファ制御を呼んでいることにも問題があるといえよう．

なお，第9図からは，イベント制御部分の比重が読みとれ，これは“プロセス間”通信の比重そのものを示している．

上に述べた改良点は，多くは“コーティング技術”的改良である．というのは，マクロなプログラム構造を保存したままでの改良である．

非能率の原因は，たとえばプログラムを APL を用いて設計したことにあるのではなく，APL の機能をコードに具体化する点にあった，ということである**．コーディング的改良が，プログラムの基本ロジックをこわして（ロジック自体にも改良される余地があるであろう，というのは別問題である）まで行なわれるとすれば邪道であろう．またコード化自体，曲芸ではなく正当化されるものでなくてはならない。ここで行なわれたのはその範囲内である．

上のような改良は，結果的には，当初の予想線に近づくものであったとはいえ，しかし，実際の改良の手がかりは，システム動作の測定と解析にあったことを強調しなければならない．それは各時点での最大のネックを指摘し，改良（チューン・アップ）の正しい順序を示すからである．

5.7　ユティリティの蓄積について

ETSS では，基本的ユティリティとして，エディタを主記憶に常駐させている．これは結果的にもその使

* その予想はまだ達成されていない．その後の改良結果や検討によれば，予想線は達成不可能とは思われない．しかし，それは基本設計の枠内での理想線に近いものであり，そのような予想をたてること自体が，“プロジェクト”参画者としてはどうであったかと反省している．

** このような結果の公表は，気恥しいものである．貧弱な裸体を公衆にさらすようなものだから，普通には行なわれないことのようである[8]．しかし，そこを我慢しなければ，“実験”システムとしての意義は薄れよう．

* ETSS のコールは，システム・モジュール間の capability の制御（保護）であるが，そうすることによって，虫とり，保護に大きく役だつ．性能は落ちる．

** ETSS 開発時に，コーディングより，プログラム・ロジックの重要なことを強調した．その副作用でもあったかもしれない．

用頻度からみて良い選択であったと思う.

その他のユティリティは，ファイルの中に蓄積する方針で，出発点では，アセンブラだけがあった.

その後十数個のユティリティが作られ，またユーザプログラムとしては種々のものが作製されたが*，しかし，基本的ユティリティとしての言語プロセッサとしては，現在でも，初期のアセンブラを拡張した，アセンブリ・リンケージ・システムしか，実用にたえるものがない. したがって，ETSS のユティリティの蓄積は十分でないのである.

実用に耐えるコンパイラがないのが問題であると指摘されている. 現在ではそれがユティリティ生長の必要条件なのであろう.

これに関連しては，バッチ（商用）OS との関連が一つの問題点である. ETSS の設計は，商用 OS と全く別に行なわれた. このこと自体は，ETSS が実験システムであるということで，商用 OS に束縛されずに進んだのは，よいことであったと思う**. しかし，そのために，商用 OS 上のユティリティとの連続性がなくなった.

基本設計レベルでいえば（簡単化された），商用 OS のシミュレータが，ETSS で容易に作製されるように，したがって商用 OS 上のユティリティがそれを介して移行できるように，GPP の設計に考慮を払ってあった. しかし，そのシミュレータはまだ具体化されていない.

技術外的には，ETSS 開発グループが（次のテーマのために），昭和 43 年 4 月段階で解消されたことがきいている. その後のシステムの改良やユティリティの拡張は，有志によって非組織的に行なわれているからである.

1. あ と が き

以上に，ETSS の構成の紹介とともに，（主観にかたよっているかもしれないが）率直な技術的な反省を試みてみた.

開発の実際は，グループ・メンバの協働の意欲によってのみ支えられるのであり，ETSS の場合は，若手研究者諸君の自発的なチームワークが，その成功の決定的要因であったといえるであろう. それはまたグループ解散後も，自発的な改良への参加・努力として現われている.

最後に，グループのマネジャーとしての重責を果した相磯（現）計算機方式研究室長と，また，この政治的季節に ETSS 開発を推進させた野田計算機部長の労を特筆すべきであるし，大型プロジェクト研究開発連絡会議の諸先生方の精神的・知識的ご援助に感謝しなければならない.

* その中には，研究室内図書の管理・検索システム，研究室内消耗品管理システム，種々のゲームのプログラムなども含まれている.

** 商用 OS は，現段階ではとくに洗練されたものではなく，それをベースにするとすれば，長所よりむしろ研所にひきずられる.

参 考 文 献

1) J. H. Saltzer: MIT のタイムシェアリング・システム—Multics System（昭 42-02）日本電子工業振興協会.
2) G. G. Dodd: APL—a language for associative data handling in PL/I, Proc. FJCC (Nov. 1966).
3) 電気試験所彙報，ETSS 特集号，（昭 43-08）.
4) M. V. Wilkes: Slave Memories and Dynamic Storage Allocation, Proc. IEEE, p. 270 (1965-04)
5) 金田，ほか: ETSS における擬似ページ・アドレス方式とその効率，信学誌，**52-C**, 5, p. 267（昭 44-05），ほか.
6) K. Fuchi, et al.: A Program Simulator by Partial Interpretation, Proc ACM 2nd Symp on Operating Systems Principles, p. 97 (1969-10).
7) 弓場，他: タイムシェアリング・システムの解析と評価について—ETSS を素材にして—第11回プログラミング・シンポジウム報告集（情報処理学会プログラミング・シンポジウム委員会）p. C 65-C 79.
8) P. G. Neuman: The Role of Motherhood in the Pop Art of System Programming, Proc. ACM 2nd Symp. on Operating Systems Principles, p. 13 (1969-10).