オンラインコンピュテーション*

和田英一**

1. はじめに

ここでは，オンラインコンピュテーションを，主として，リモートコンソール（遠方の端末装置）から計算機をつかうことだとする．つまり最近流行のタイムシェアリングシステム（TSS）とほぼ同様なのだが，どちらかといえば，別に多勢でタイムシェアで計算機をつかわなくてはならないというのではない．ただ，とおくから計算機がつかえれば，それでよい．

リモートコンソールから計算機をつかう場合，問題となる点が多々ある．コンソールの問題，これにはテレタイプのスピード，キャラクターセット，プリントサプレスの機能，ディスプレーの使用法など，また計算機のつかい方の問題，これにはシステム全体に関する哲学，プログラム言語，ファイルシステム，アカウンティング，各種の情報サービスなど．そのような多岐にわたる問題の中から，いくつかを選んで述べてみよう．

2. 卓上計算機式使用法

これより簡単なオンラインコンピュテーションには，データを入力してそのまま出力するというのがあるが，そのつぎに位すると思われるのが，この卓上計算機式使用法か，ファイルの編集であろう．後者については別項にゆずるとして，卓上計算機式使用法について考えよう．

この中でも最低から順にならべれば，スーパーマーケットの加算器のように加算だけしかできないもの（最近はお釣まで計算されるようだ），二つの数値間の四則演算くらいを行なうもの，$\sqrt{\ }$フリーデンみたいにこれに平方根を追加したり，連乗を補足したりしてもよい．しかしこれでは電子計算機としては可愛想で（当世の卓上型電子計算機と称するものは，プッシュダウンメモリーをもっている），つぎにくるものは，データをストアするとか，（ ）や初等函数をふくんだ長い式の計算などをすることであろう．

これらの方式は，システムをつくる側からいうと，つぎのような利点をもっている．

（1） 1行の式を実行する時間は短いから，大勢の利用者の要求のスケジュールがいらない．

（2） 各利用者ごとに最大のデータ用ストーレージの場所と1行の式のはいるだけの場所をとっておけば，演算を実行するプログラムは一つでたり，しかもリエントラントにする必要もない．

（3） 利用者がプログラムを書くのではないから，システムの保護対策をする必要がない．

（4） 出力量がそう多くないから，計算機室のラインプリンタで出力をとる必要もない，等々．JOSS のマニュアルなどみると，はじめの方に述べてある使用法はこれに近い．これでもタイプをうつ速度のはやい人なら，卓上計算機をぐるぐるまわすよりずっと楽であろう．第一，結果を紙にうつしとる必要がないし，数値や計算式が正しかったかどうかの記録もちゃんと手元にのこる．

SDC だか RAND だかでは，廊下のすみにリモートコンソールがおいてあり，みていると，あちこちの部屋からかわるがわる人がでてきて，パタパタとタイプをたたいて何か計算しては，紙をちぎって帰ってゆく．こうなると TSS もかなり日常生活に結びついてきたなと感じました，という話を先達てきいたばかりだが，これなどほとんど卓上計算機としてつかっているのであろう．タイプライタの国とそろばんの国では事情はどのくらい違うだろうか．

3. サブシステム方式

プログラムとサブシステムのデータ，あるいはパラメータがどう違うのか，さほどさだかではないが，あまりプログラムの得意でない利用者にとって，卓上計算機方式のつぎにつかいよいのは，センターで用意したさまざまなサービスパッケージをコマンドでよびだし，単にパラメータかデータをあたえただけで結果をうるものであろう．もちろん上記の卓上計算機方式もこの一種だし，また，あとに述べるプログラム言語プ

* On the On-Line Computation, by Eiiti Wada (Faculty of Engineering, University of Tokyo)
** 東京大学工学部計数工学科

ロセッサーでもこの方式と考えられる場合がある．

このうち簡単なものでは，たとえばクロスフーティングをとるものがある．IEEE の特集号に紹介されていたものは，入出力装置がすこし高級で CRT をつかっている．CRT 面にクロスフートをとるべき縦横のわくがまずうつる．画面の下の方に入力キーとして数字がうつる．あるわく内に数字をいれるには，ライトペンでそのわくをさわってから，入力キーをたたくがごとくに，画面の数字を順々にライトペンでふれる．ふれると数字がわくの中にあらわれて，それと同時にか，一わく分すむとかに，下と右にある合計のわくにそれが加算され，右下にある総計にも加わってゆく．こんな方式でクロスフーティングをとるのだが，数値の間違ったわくに対しては，またコントロール用のボタンだかライトボタンだかがあって，それを利用して修正ができる．修正と同時に合計や総計も正しい値になってうつりだす．

ファイルシステムに関係のあるサービスパッケージでは，ファイルのつくりだし，修正など，またファイルのカタログうちだし，ファイルの内容のリストなどが重要である．

そもそも，ここで考えているオンラインコンピューティングは，リモートコントロールで計算機をつかおうというのだから，計算機だけは遠方にあるが，できることは近傍にある計算機，近傍にあるデータと同様でなくてはならない．したがって，計算機の制御盤と同じものが必要だし，データ(ファイル)についても，カードや紙テープをつくったり，複製したり，修正したり，いれかえたり……が，普通にできなくてはならない．

そのほか，普通の（モニターによるバッチ処理の）システムにあるシミュレータをはじめとする各種のサブシステムも逐次オンライン用として用意されてくるだろうが，どのサブシステムも当然オンライン用として設計されなおす必要がある．バッチ処理のシステムは，あらかじめきめられたパラメータで最後まで実行するよりしかたがないが，オンライン用になると，利用者は使いながら結果をうちださせてみたり，途中で適当にパラメータをかえてみたりする自由がある．それを最大限に利用できるように作りなおす．むかし計算機を自分で操作しながら好きなように使っていたとき，最大の武器はいくつかあるセンススイッチであった．

プログラムの途中にセンススイッチの状態をしらべる命令をいれておき，たとえば，途中結果を印刷する箇所をスイッチで自由にスキップできるようにしておいた．長時間の計算など，普通はその部分をスキップさせてフルスピードで計算させ，ときどき計算機室へ様子を見にいっては，スイッチをたおして途中結果をしらべ，またスイッチをもとにもどして計算を続行させたりした．また停止命令をいれて計算を適当なところで停止させ，つぎの命令からスタートさせるスタートスイッチや，つぎの命令を一つスキップしてからスタートさせるスキップスタートスイッチなどをつかって，プログラムのルートを分岐させたりしたものだが，それをオンラインコンピュテーションのシステムにもう一度とりいれて，システムを作ってゆくのがよいと思う．それにはユーザーマシンというかプロセスにいくつかのセンススイッチを用意し，特別なコマンドで，進行中のプロセスを妨げることなくスイッチの操作ができるようにする．たとえばコマンドシステムに $ SSW*n* ON, $ SSW*n* OFF というのを用意し，クイットさせてからつぎに *n* に適当な番号をいれてこのコマンドをうつと，それでそのプロセスの *n* 番のスイッチはどちらかに倒されたことになる．もちろんサブシステムのプログラム（利用者が自分で書いたプログラム）もいま自分が走っているプロセスのセンススイッチをしらべてジャンプすることができるようになっている．（センススイッチのわりこみについては後述）

4. 利用者のかくプログラム

もっと一般的にこのシステムを使うためには，お仕着せのサブシステムだけでなく，利用者が自分でつくったプログラムもかからなくてはならない．利用者に自分でプログラムをかかせて，このシステムにかけることには，もう一つ重要な意義がある．それは完成したらバックグラウンドジョブになってプロダクションランをするプログラムのデバッグとしての役割りである．

オンラインコンピュテーションにふさわしいプログラム処理は，コンパイルよりもインタプリタ方式かもしれない．まず卓上計算機方式のところにも書いたように，これだとプログラムの脱線転覆がない．エラーメッセージは実行中のものまで懇切丁寧にだすことができる．出力の印刷を例のセンススイッチで任意にきりかえることができる．また一ステートメントごとに実行しては停止することもできる．ステートメント単

位でプログラムを修正しても，全体をコンパイルしなおす必要がない．などの利点が考えられるが，一方欠点としてはなんといってもスピードが遅い．またこうしてつくったプログラムを，このシステムのサブシステムとして登録することが困難，といったことがある．

プロダクションランのデバッグ用としては，やはりインタプリタ方式がよい．オンライン方式が云々される以前，バッチ処理一本のころでも，まったく同一の言語プロセッサに，コンパイラ方式のものと，インタプリタ方式のものの二つを用意するのがのぞましいという意見があった．それはプロダクションはなんどもくりかえして使用されるから，コンパイルに多少時間がかかっても，できあがったオブジェクトプログラムの効率をできるだけ高める必要がある．しかし，デバッグの途中では，コンパイルに時間をかけても，実行開始直後にエラーのおきることがあり，そうするとまたコンパイルやりなおしだから，コンパイルにはできるだけ時間をかけず，また実行中のエラーを十分よく発見できるようにインタプリタ方式にすべきだというのである．それをオンライン方式でもとりあげて，バックグラウンドでフルスピードではしるプログラムをオンラインではインタプリタで綿密にテストするのがよい．

オンライン方式ではフォアグラウンドでコマンドを用意してバックグラウンドではしらせるというのがあるが，このとき，フォアグラウンドと同じインタプリタを使用したとすると，インタラクションをどうするかという問題のほかに，バックは同じコマンドでもコンパイラ方式にきりかえるという考慮がはらわれてもよいのではないだろうか．

九大の牛島氏たちはインタプリタ方式はおそすぎるし，コンパイルを全部やりなおすのも大変だというので中間の解決法を ALGOL で考察したのだが，これもきいてみるとなかなか面倒のようである．

5. ファイルシステムの問題点

ファイルシステムというのが，オンラインコンピュテーションではいろいろ厄介な問題をひきおこす．ファイルのつくりだしから，セグメント化，ページ化されたコアとバックアップメモリー間のやりとりをするベーシックファイルシステムにいたるまで数限りなくある．そのいくつかを列挙すれば，

利用者が端末でタイプしていれているファイルは，ただちにファイルシステムに登録されるかどうか，ダートマスの TSS などみると，ファイルシステムに登録してとっておきたいものは SAVE コマンドをだすようである．SAVE をださないと，そのファイルは適当なところで消えてしまう．これに対して，端末でうったり，計算機による処理の結果できあがるファイルは黙っていても片端から登録記録されて，いらない場合にはいちいち DELETE しなければならないシステムもある．この二つのどちらがよいか．

まず人間はうかつだから，うっかり SAVE しなかひでしまったことになるのを親切に防ぐには，どれも登録されるとよい．ところで，A というファイルをコンパイルして B ファイルをつくり，これをランさせるというようなシステムをつくったとする．

```
$ COMPILE (A, B)
$ RUN (B)
```

というようにである．その結果 A を修正する必要が生じ，A をアップデートした．もう一度，上の二つコマンドをだすと，B が 2 重になってしまう．B はすでにあるからだめだというエラーメッセージがでるのか，COMPILE (A, B) とすれば，自動的に B というファイルはデリートされるのか（これももしうっかり B という重要なファイルがあったとすれば大変危険である）その辺が，われわれで検討中のシステムでもまだ保留事項である．

どんどん登録してしまうシステムにすると，自分のファイルシステムをもっていない利用者のファイルはどうなるかという問題もある．はじめに書いた SDC だが RAND の例のような使い方で，公衆電話のように通りがかりの人がちょっと硬貨をいれて端末が使えるようにしたい．その人のファイルをどうするかというところで困ってくる．余談だが，この硬貨をいれて端末装置を使う方法では，タイプライタに公衆電話や乗車券自動販売器のような硬貨投入口をつけなければならない．貨幣の種類に応じて，適当なコードが送出される必要がある．そろそろ何にでもとりつけられる貨幣判別器を発売しないものだろうか．お釣をだすためには，反対にコードを受信すると，お釣の出口が開く仕かけもほしい．

6. バックアップメモリーのバックアップの方法

SAVE をしなければ登録されない方式にしたところで，ディスク上のファイルシステムはどんどん膨大な

ものになってゆくことは明らかである．そこで当分不用のファイルは磁気テープにでもおとしておいて，必要のときにその磁気テープからまた読みこむ．磁気テープにおとすことを奨励するには，ディスク上にファイルの本体（ディレクトリと別）をおいておくと，その面積と時間に従って利用者から料金を徴収するようにする．そしてどんどん DROP してもらう．しかしこうすると，次なる問題は磁気テープの本数がじゃんじゃんふえてゆくことであろう．磁気テープにファイルをおいてあっても，ごくわずかの料金をとるかどうかは別として，磁気テープを急激にはふやさない方法がないものか．つまりあるファイルAをテープにおとした．しばらくして使用するためディスクに読みこんだ．そうすると前のテープ上のファイルは不用になったわけである．Aを何回だしいれしても最後のものだけ入用であとはいらない．しかし方式によってはいらないファイルのつまったテープを保管することになるがまたファイルというのは何世代か前まで保存しておけという流儀もあるから，保管庫が十分あればとっておくのが望ましいかもしれない．

つぎに述べる方式は，まだテストはしないが，磁気テープの本数の膨張をできるだけおさえようというものである．テープを1番，2番……n番とn本を用意する．nは適当にきめればよい．テープはサイクリックに使用する．磁気ディスク上のファイルシステムには，ファイルの親子関係や性質を示すディレクトリが木構造にはいっていて，末端にファイル本体がついている（ディレクトリはテープにおとされないファイル本体である）．テープにおとすのは，この本体だけで，ディレクトリはディスク上に残す．いま，このシステムではテープ駆動装置を3台使用することにして，それぞれ LOAD, STANDBY, DUMP とよぶことにする．i番のテープが DUMP にかかり $i+1$ 番のテープが STANDBY にかかっている．いまかかっているテープがi番だということを，ファイルシステムは知っていて，利用者がファイルAを DROP しようとすれば，Aの本体をiのテープにかき，Aのディレクトリに，Aはi番にあってディスク上にないというマークをつける（Aの本体は適当なときにガーベージコレクタで始末される）．DROP コマンドでつぎつぎとiにダンプしているうちにi番がいっぱいになると，システムはオペレーターにiをはずして，$i+2$ をかけるよう指示する．かけかえが終わると，STANDBY と DUMP の(ロジカル)装置を交代するが，まず$i+2$をはじめからよんで，ファイルが出てくるたびにそのディレクトリをディスク上でさがし，そのファイルが$i+2$ のテープにあるかどうかをみる．$i+2$ になければ，不用としてよみとばす．$i+2$ にありとなっていれば，STANDBY にかかっている $i+1$ にうつして，ディレクトリを $i+1$ にありとかきかえる．こうして $i+2$ のすべてのファイルから入用のファイルを $i+1$ にうつしかえ，それらのディレクトリをかきかえ，$i+1$ は DUMP 用テープとして準備が完了する．$i+2$ はまきもどして，STANDBY となり，$i+3$ から入用なものがうつされてくるのをまつ．一方ファイルシステムでファイルが必要になり，その本体がディスク上にないと，ディレクトリから何番目のテープにはいっているかの情報をとって，オペレータにそのテープを LOAD にかけさせる．LOAD はそのための駆動装置である．たまたま DUMP 中のテープにあれば，それをまきもどしてそれから LOAD する．1本のテープに同じファイルが二度おとされることも生じる．上記で DUMP 中のテープから LOAD し，テープ交換時以前にまた DROP するときである．しかしこれもディレクトリにどのテープの何番目のこのファイルの本体が最終のものだという情報をつけるのは容易である．STANDBY へのうつしかえ，LOAD にはその情報を利用すればよい．ディレクトリが膨張するのはまた別の方法でにげなければならないと思う．

7. ファイルをシェアする場合の修正の権利

ファイルが一ヵ所に集まっていれば多勢で利用したくなるのがあたりまえである．各利用者ひとりひとりのファイル（これは自分でしか利用できない），システム全体のファイル（だれでも利用できる）のほか，何人かのグループで共用するファイルなどが考えられる．共同で利用する方は，許可のしかたと，許可されていない人が利用することのチェックさえできていればそれでよいが，一たんできて共同で利用しているファイルを修正しようとすると，だれでも修正できるか，いつでも修正できるか，という問題がおきる．ふつうは共用していても，所有者または担当者がひとりいて，その人だけが修正できるようにするらしいが，その権利を他人にゆずれる方式というのもどうであろうか．もちろんゆずってしまったら，もとの所有者には修正の権利はなくなる．ゆずる場合にはまずゆずられる相手に了解をえておいて（そのためのコマンドを用意して），それからゆずる．ゆずられないのにぶん

どることはできないし，いやな人にむりにゆずることもできないようにしておく．前者は知らないうちに権利を剝奪されていたことだし（つまりだれにでも修正できることになる），後者はディスクの面積を使用していてかかる料金を，他人におしつけることにもなるのでそれを防ぐのが目的である．ファイル本体をテープに DROP する権利も似たようなものであろう．

ファイルシステムを共用する場合，複雑怪奇なシステムは，木構造のディレクトリのどのレベルからも共用できるもので，これに対して本体だけ共用する方式ははるかに簡単である．木構造のディレクトリは共用の関係がなければ，ひとえに男子だけをかいた系図のごとく分岐してゆくだけだが，それに共用（リンク等）の関係をいれると，母方の系図までかきこんだ場合のように接続がややこしくなる．ディレクトリファイルが段々エスカレートしていって，一つのファイルにいれられる大きさを超越したとき，ディレクトリをいくつかのファイルに分割しなければならないが，その分割に際しては，木構造をたどってゆく場合，できるだけファイル間の移動の回数を少なくしたい．それには，ディレクトリが途中で借用されているとわけ方が困難だが，本体だけを借用する場合なら，一つのまとまった枝は一つのファイルにおさめるという分割によって，ディレクトリをたどる間は別のファイルにうつらないようにできる．もっともディレクトリを共用しないためにエスカレーションが激化するという事実も見のがせない点である．

8. ファィルの修正とコピー

ファイルシステムに修正やコピーの要求がでた場合，それをいつ実行するかという点にもいろいろ考察すべきことがある．修正もコピーもその要求をそのまま保存しておいて，いざそのファイルを使用するという段になって修正，コピーを行なう流儀と，逆に要求のでたそばから逐一かたずけてゆく主義とがある．前者では修正やコピーの要求をまた修正したりコピーしたりできるのがみそだが，だんだんわからなくなってくる心配があるので，簡単には要求の時点で直接実行してしまうことである．われわれのシステムでも，この両者の得失をいろいろ論じた結果，簡単な方を採用しようという気分になっている．

直接実行型は，たとえばコピーならば，他のファイルからうつしてくるのに，リンクをつけておくだけでなく，ファイルの内容をよんできて，いまつくりつつある，あるいは修正しつつあるファイルに実際にうつしかえてしまうのだが，ここでまた問題になったのはそのうつしつつあるファイルの情報をテレタイプで印刷するかどうかということである．ファイルの内容は古いファイル時代のライン番号がなくなって，うつされながら新しい番号がついてくる．その新番号と内容との対応を知るために，タイプする必要があるという考えと，テレタイプでいちいちうつのは時間がむだだというおもんばかりとがある．後の意見に対しては，結局新旧両ライン番号の対応表だけを，どちらかの番号に不連続のあるところだけでタイプするという方法で解決しようとしているが，それでもタイプの時間は決して馬鹿にならないから，あるいはその対応表もサプレスできるか，または新番号は整っていなくてよいから旧番に一率になにかたしたものにする，という方式かを考えてみる必要があろう．

9. インプットの問題

前にインプットとしてセンススイッチのことを書いたが，この辺にもいろいろ問題がある．センススイッチは，センス命令のところにこなければ働かない．同様なのは入力命令で，いくらテレタイプであらかじめ情報をタイプインしたくても，入力命令が実行されなければタイプするわけにゆかない．そこでわれわれのシステムではその点をもっとフレキシブルにして，センススイッチ（タイプでうつ）にしても，テレタイプで情報をうつにしても，プロセスの動作中にできるようにしたいと考えている．つまりプロセス動作中にタイプがうたれると，プロセスのプログラムのなかでわりこみがおき，その場で至急その情報処理をして，またさきほどのプログラムにもどるというのである．このわりこみ制御のプログラム（利用者にも自由につくれるようにしておく）のつくり方如何によって，入力がきわめて自由になるはずである．

われわれのシステムでは，クイットするにはプロセスの動作中（入力まち以外のとき）にテレタイプをうつことにしているが，これは上記のわりこみ処理ルーチンによってしらべられ，クイットコードなら停止するし，それ以外の文字なら適当に処理できるようにしている．しかし現在のこの方式では入力まち以外の入力に対してエコーをかえすことをしないので，正しくうけつけられたかどうかのチェックができない難点がある．

インプットがわりこみになっていると，入力回線を

経由して端末のあらゆる情報を即時にプロセスに伝える手段をのこしておくことで，極めて重要であろう．笑い話では端末の椅子にスイッチをつけ，人が腰掛けたり立ったりするたびに，硬貨判別器の出力のように特別なコードを送り，それがプロセスにわりこむ．プロセスは利用者が椅子から立って，またすわると，それは前の人と同一かどうかまたパスワードを送ってただちにチェックを試みる．それはそれとして将来いろいろな端末装置をつけた場合，そのあらゆる状態を至急プロセスに教えたいことが出てくるに違いない．その日のために入力はできるだけ一般的にして，わりこみの余地をはじめからとっておこうというのである．

10. パスワード

MIT の TSS でおもしろいのはパスワード方式である．利用者が多くなってくるとどうしても「うっかり」も多くなるので，利用者の番号のほかになにかレダンダントなものをいれて，間違わないようにする必要がある．それだけならそのレダンダントな情報もプリントされていてかまわないはずだが，MIT のパスワードはこの「うっかり」を防ぐだけではなく，「わざわざ」他人の番号で使用するのを阻止しようとしている．それはパスワードを印刷しないで，あとからみたのではなにをレダンダントの情報としたかわからないようにした．これには二つ問題がある．一つは端末のタイプライタに，入力プリントサプレスの機能をつけなければならないこと，もう一つは，レダンダント情報をタイプしているところを見られたらおしまいだということである．

そこでこれらに対する策として，われわれが考えているのは，シャノンのいう，あぶりだしのインキをもちいるのではない，正々堂々と暗号文を全部敵に見せてしまう方の暗号システムである．つまり利用者と計算機はともに共通のコードブックをもっていて（もちろん敵はもっていない．），利用者の番号をそのコードブックを利用して暗号化，平文化を行なうのである．

利用者がログインすると，計算機は0から9までの数をランダムな順にうってくる．利用者は自分の番号をキイとした暗号化で，その乱数情報を暗号にして返事する．計算機はそれが自分で期待したとおりの答えであれば，利用者を確認したことになる．暗号化の方法（コードブック）ははじめのうちは一通りでも，暗号からキー番号を解読するのは少ないデータからは不可能だが，データがたまってくるとわかってしまう．データが多く集まればどんな方式でもだめだが，乱数から暗号をつくるプログラムを利用者に自由につくらせる（あまりワード数をくわないように）のもおもしろい．とにかくこの方式は上記二つの問題点をどうにか切りぬけるものである．しかし世の中にも知恵者がいて，なんとか暗号を解こうとしているものである．そこで高橋教授の提案は，パスワードの返答の様子から，どうもコードブックを盗まれかけているらしいということがわかってきたらパスワードチェックのプログラムが利用者（本物の方）にコードブックの交換を要請するのがよい，というのである．

前述の硬貨でシステムを利用する公衆端末では，ログインの方式を変形して，パスワードは用いなくてよい．

パスワードの一種は，システムプログラマーとオペレーターのそれである．つまりふつうの利用者にはロックされていて使えない機能が，このシステムプログラマーとオペレーターのキーで解錠される方式を準備しておかなければならない．国鉄の車掌スイッチに使っているキーで電車のいろいろな部分があけられるようだが，そのキーにも似たパスワードである．この方式については，あまりここでかきたてない方がシステムの安全によいよう（身のため）だから，問題だけを提起するにとどめておく．

11. システムフリーズなど

オンラインコンピュテーションでは，なにしろ計算機が遠くにあるから，いま計算機がどんな状態にあるかなかなかわからない．いろいろな状態の変化をどうして利用者に教えるか，また休みなしに利用者が使っているシステムを修正するために一時全面停止にしたいことがあるだろう．それにはどうするかということも考えておきたい．

全面停止は，たとえば構内電話の交換が何時かで終了するときの状態に似ている．だれかが長々と話しているのを急にきるわけにはゆかないし，さりとていつまでも待つわけにはゆかない．一つの解決法は入力要求がでると，計算機は3分以内にその入力は終了するものと期待する．3分たってもメッセージ終了のコードがこないと，その入力情報は3分のところできられる．主ルーチンへはそれまではいった情報と字数，それに入力うちきりまでの時間，入力うちきりの原因などをかえす．かりにその行にタイプミスがあって，1行キャンセル符号をうち，つぎの行に新規にうちなお

すときも，その1行キャンセルのところで一たん，入力うちきりとする．入力うちきりでつぎに入力要求をだすとき，もしシステムフリーズをシステムオペレーターがだしていたら，その時点で利用者とのインタラクションは中断され，システムからしばらく使用できない旨のメッセージが出される．

システムから利用者への各種の連絡（通報）は，利用者が端末よりログインしたときに，（パスワード確認ののち）その利用者の前回のログアウトのとき通報した以後の情報を出力する．あまり細かいことを通報すると時間がかかるので，ほとんど見出しくらいしかだせないが，とにかくそれをまずうつ必要がある．そのセッション中にシステムから知らされる情報は，緊急度に応じてプロセス実行中，コマンドまちになったとき，あるいはログアウトのときにタイプされる．ある端末をたったひとりの利用者が利用するのではないから，システムから出される各通報について，どの利用者にはどこまで知らせたということを記憶しておいて，利用者単位に全部伝わる様にする．こうするといやらしいのは，新規加入した利用者がはじめてログインしたとき，天地開闢からの通報をうけてるかということと，公衆端末の利用者にどう知らせるかということである．また本当に緊急なときに，停止中のタイプライタを起動させて，通報をだすようにするかも問題である．

12. おわりに

オンラインコンピュテーションはどんなであったらよいかを，ここしばらく多勢の人々と論じてきた．まだ別にシステムができあがったわけではないが，オンラインコンピュテーションについて筆をとるにあたって，その討論の主題のいくつかを紹介してみた．もっと具体的な例を書くべきであったかもしれないが，読まれるかたに対しても，考察のきっかけを与えるのには十分であろう．

われわれのシステムというのは申し遅れたが，東大の大型計算機センター HITAC 5020 を使用して研究を開始しているシステムである．この討論には，センター長高橋秀俊教授も大いに意見を述べられ，また日立中研の TSS のグループも熱心に案を出された．それらをすべて紹介できないのは残念だが，そのおもなものを筆者が総代となって若干述べたというところである．

（昭和24年9月18日受付）