The Development of Systems for On-Line Mathematics at Harvard

ADRIAN RUYLE

Harvard University, Cambridge, Massachusetts

I. Project TACT

Project TACT (Technological Aids to Creative Thought), in the Aiken Computation Laboratory of Harvard University, has been designing, implementing, and using on-line systems since 1964. Our investigation has concentrated on a rather small class of interactive systems which are "dedicated," that is, tailored to provide a coherent set of tools for mathematical assistance, and which use fairly powerful terminal facilities: display scopes and natural input.

The reasons for not considering a wider class of technological aids are several. Programmed instruction, the use of predetermined heuristic sequences, does appear valuable for training in specific topics, but using programmed instruction for assisting creative thought presents vast problems. In the first place, teaching sequences may not be the most effective way of presenting much of the ideational background material used in the creative process. Moreover, even assuming that fully enlightening sequences were stored for entire batteries of background topics, the problem of selecting the proper sequence to assist the researcher at a particular point in his thinking is a full-blown information retrieval problem.

A further possibility is to use programmed instruction to produce a favorable set toward creative thinking. Covington and Crutchfield [1] report that programmed instruction can be used to enhance general creativity in school children. Their method is to tell mystery stories, breaking off at critical points to introduce programmed sequences of speculative questions.

However, it is not clear how to relate their method to advanced topics; and in any case the result would at best resemble a mental tuneup, without sufficient relevance to specific problems to be of direct use when help is needed. Thus, our main focus is on systems in which the machine does not program the man. Initiative rests, instead, entirely with the user, who employs a computer *ad hoc* as a passive tool.

To avoid the fields of artificial intelligence, linguistics, and information retrieval, which are veritable sinks for research effort, we concentrate on the fields in which data processing technology is well developed, hence the emphasis on applied mathematics.

To keep the subject matter in the foreground and avoid clerical involvement with the machine, we have restricted our attention to systems not requiring specialized knowledge of computers or programming. The Compatible Time-Sharing System (CTSS) of Project MAC, for example, is a powerful technological aid to creative thought for the programming elite; but its facilities for compiling, disk file management, text editing, etc., taken together, lack the simplicity and coherence necessary to make them readily accessible to the nonprogrammer in a problem-solving situation.

On the other hand, JOSS is a simple coherent system. But it is one in which the ratio of clerical detail required to power afforded is rather great. While JOSS performs beatifully with scalars, it does not allow the user to compute directly with functions without writing loops; and, JOSS has no facilities for graphic display.

II. The Culler–Fried System

The Culler–Fried system [2–4] is mathematics-oriented, simple, coherent, and powerful. It strongly influenced the growth of AMTRAN; and it was chosen as a point of departure for the development of TACT's in-house systems.

The Culler–Fried system functions roughly like a desk calculator, generalized to allow the user to compute directly on real and complex functions (using many of the common operations of mathematical analysis) with the same ease as with simple arithmetic operations on single numbers. As with a desk calculator, the user initiates all computing by pushing buttons. The buttons, or keys for operations, shown grouped in the upper half of the keyboard of Fig. 1, include not only the arithmetic operations but many more used in mathematical analysis, such as sine, exponential, and forward difference. The keys for introducing numbers and saving data entities are grouped in the lower half of the keyboard.

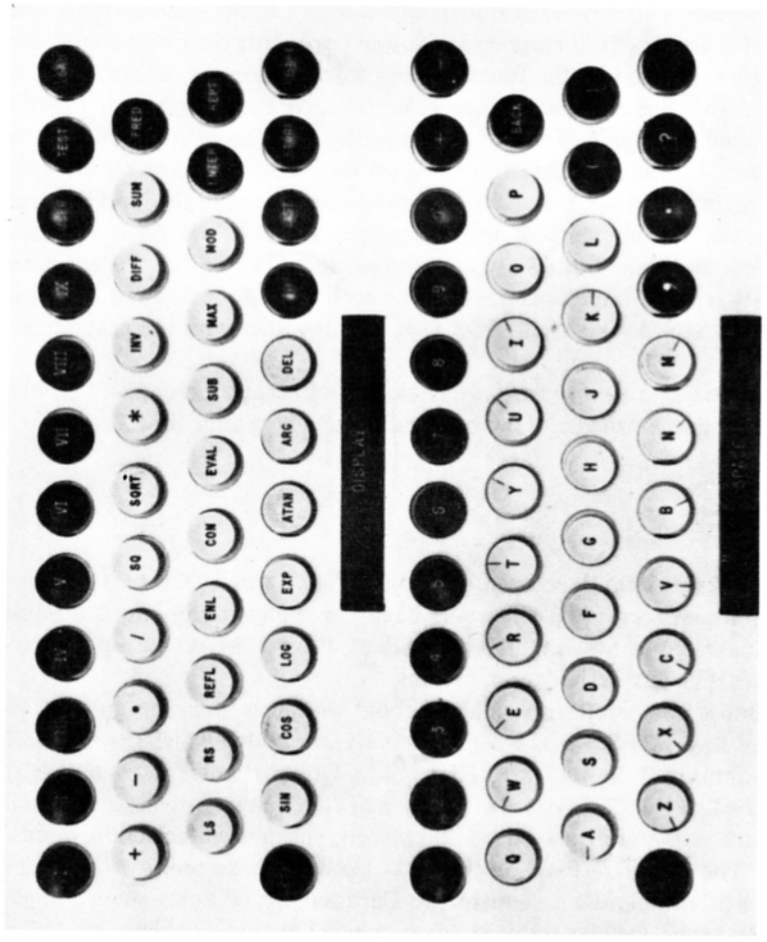

FIG. 1. Culler system console keyboard.

The DISPLAY bar, in the middle of the keyboard, causes output to appear on an associated storage oscilloscope. Scalars are displayed as numbers, real functions are displayed as graphs, and strings of complex numbers are displayed as arcs in the complex plane.

Pressing the key labeled I causes all the operator keys to refer to operations on scalars, and all lower keyboard alphabetic keys to refer to scalar data entities. This keyboard state is called level I of the system. When the key labeled II is pressed, the upper and lower keyboards deal with functions: the operators specified by the function keys work on *vectors*, or strings of scalar values; and the alphabetic keys refer to such vectors. A real function is considered as a pair of vectors, an abscissa vector and an ordinate vector; therefore, to accommodate functions on level II, two vector registers, called the X register and the Y register, are maintained. Most of the level II operators affect only the contents of the Y register. On level III all operations are complex, and data entities are complex vectors. The power of the system on level III is quite impressive, as may be seen by the presence of the complex operators SIN, EXP, ARG, SQRT, *, etc., together with the graphic display facility.

There is also facility for on-line storing of *console programs*, i.e., button-push sequences which can be executed *en bloc* at a later time.

III. TOCS

The first system developed by Project TACT was TOCS (TACT On-Line Computing System) [5], which was begun in 1965 with an effort to duplicate the Culler–Fried system under CTSS at Project MAC, using the system manuals [3,4] as design specifications.

Almost immediately, new ideas about on-line system design were incorporated into TOCS. First, a display facility was added which obviated certain disadvantages of the Culler–Fried displays. One particular disadvantage stems from the fact that this system displays each curve on its own scale. When more than one curve is displayed on the screen, spurious intersection points can result. The TACT display, on the other hand, gives an undistorted picture of a selected rectangular region in the Cartesian or complex plane. Thus, the console scope can be thought of as a window for scanning an unlimited virtual plane. Instead of always showing an entire curve when a display command for real functions or complex arcs is given, TOCS shows only those portions of the curve which pass through the specified region of interest.

Next, operators oriented toward statistical calculation were included. The ability to enter numeric data directly into vectors, an awkward process in the Culler–Fried system, was implemented.

The first steps toward providing a capability for algebraic manipulation were also taken; and, while these steps did not go far enough to allow (in themselves) algebraic manipulation of formulas, they did permit the expression of formulas within the system. A simple example will illustrate the importance of this facility. Suppose a user has calculated two quantities, a and b, stored them under buttons A and B, and reads in a textbook that a parameter of interest, x, can be calculated from the formula

$$x = \sin(a^2 + b^2)$$

In the Culler–Fried system, he must program the evaluation of this formula for himself, as he would do on a desk calculator

```
I LOAD A SQ STORE T LOAD B SQ+T SIN STORE X.
```

Nonusers will not know what the sequence means, and even adept users will have to decipher it before they know what is going on. The TOCS facility allowed this user to type

```
LET x(a,b) = sin (a**2 + b**2)
E x(A,B)
STORE X
```

or if the formula is not needed again

```
E sin (A**2 + B**2)
STORE X
```

Not only does this save the labor of programming the evaluation of formulas, but also it simplifies communication between users.

Based on a comparative use of the Culler–Fried system and TOCS, we found the idea of calling operators with pushbuttons to be excellent, but calling *all* operators in this way to be bad. Either we must provide a great number of buttons and search through a huge panel like AMTRAN's for the appropriate one, or we must double up. Culler–Fried doubles up, placing such disparate operations as square root, and the operator which sorts a string of numbers into ascending order, under the single button SQRT. To keep track of which operator is intended by a button push, levels are introduced. An operator such as exponentiation does not need level distinctions, since the data type of its operand is sufficient to identify the operation; yet under the button EXP on different levels are an exponentiation operator for scalars only, another for real vectors, and a third for complex vectors. With the button * on the other hand, there is no mathematical connection between its meaning on the complex level as conjugation and its meaning on the real level.

The worst feature of doubling up on the buttons arises with user-created operators. Any composite program of operators is itself considered an operator in the Culler–Fried system, and as such must reside under a button. This leads to storing a composite program, such as an integration program, under an available button like SUM or + on one of several "user levels." This, in turn, leads to the necessity of providing system names and comments for such buried user programs, and burying these with their programs. The name for a user program which is stored under USER I + will be stored with the five button sequence STORE USER I DISPLAY +. Since storing names and programs are distinct processes, the user must take care to keep name and program under the same button on the same level. The user who forgets where a console program is stored must hunt and peck with one eye on the console scope until the appropriate name, or program listing, or comment appears.

Therefore, in this version of TOCS, there was no doubling up on operator names. Levels were dropped altogether, while retaining all of the Culler–Fried real and complex scalar and vector arithmetic. This was made possible by allowing arbitrary names for console programs, and considering the data-type of operands when identifying operators during interpretation.

In addition, more convenient facilities for console programming were developed. These include the ability to designate which variables are local to a console program, and to designate variables as argument variables (local variables to be assigned values from a list of parameters accompanying each request for the console program's execution). The superb edit facilities of Project MAC's TYPESET were made available from within TOCS for modification of console programs.

Some further improvements include the ability to define or redefine system operators with only a limited knowledge of CTSS, the ability to use vectors of unlimited length (to within memory size), the presence of error messages, the ability to define synonyms for system operators, and the ability to give multicharacter names to data entities.

TOCS has had considerable application. Two models of biological systems were explored using an early version of TOCS [6,7]. A later version was applied to the area of ordinary differential equations. TOCS has also been used "live" in a number of lecture situations at MIT and Harvard.

To further expose TOCS to the test of use, the TOCS disk files have been made accessible on a read-only basis to any legal user of Project MAC's CTSS.[1] At least one user has made use of TOCS in this way. The Research

[1] The following commands by a MAC CTSS user will establish a working connection to the system:

```
LINK TOCS BCD T262 3491
RUNCOM TOCS
```

and thereafter the command:

```
R TOCS STDOPS
```

is sufficient to put him in touch with the system.

Laboratory for Electronics at MIT has been finding roots with TOCS using an interactive graphic method based on Cauchy's integral formula [8].

IV. Tact's Latest System: TOC[2]

During the development of TOCS, experimentation with other systems continued. A comparative study was made of AMTRAN, the Lincoln Reckoner, MAP, and the Culler–Fried system [9].

Design of a new TACT system, dubbed TOC (TACT On-Line Computer), was begun in 1966, using three basic system features borrowed from the Culler–Fried system:

1. the use of pushbuttons to invoke operators,
2. operators which work automatically on entities larger than scalars, and
3. a repertoire of operators which provides the basic tools of operator calculus. From the Culler–Fried operator set, it is easy to construct composite operators which differentiate, integrate, etc.

We also borrowed two engineering features for console output.

4. The use of storage oscilloscopes: For refresh-type display scopes, a data-image of any visible picture must be kept in memory as long as that picture is maintained on screen. This necessitates either tying up computer memory or else maintaining local memory at the console. For flicker-free display, the transfer rate from memory to scope also must be quite high. Storage oscilloscopes, which hold the picture from a single transmission indefinitely, obviate these memory and speed requirements.
5. The ability to draw freehand on the output scope from the input keyboard: Drawings are composed using 16 small directed straight line segments, each associated with a button. Any button sequence which defines a drawing can itself be stored under a button. This allows the addition of special characters, new type fonts, etc. to the system.

Many more ideas are borrowed from TOCS. Multiple character names are allowed for both operators and data. Synonymy is possible, and there are no levels in the system. There are no restrictions on the length of arrays beyond a single restriction on the user's total memory size. In TOC the user may also declare a "diagnostic level" which specifies the extent of the system messages he desires for guidance while working. In the lowest diagnostic level no messages appear, in the manner of Culler; in the highest level errors and omissions will trigger thorough explanatory messages to the user. Formula expressions are allowed, enclosed within parentheses. Facilities for console

[2] Since this paper was written this system has been renamed THE BRAIN (The Harvard Experimental Basic Reckoning and Instructional Network).

programming of composite operators within TOC include *predicate* and *local* declarations, as well as a branching ability. Access from within TOC to LISP, assembly language, and PL/1 is planned.

Direct specification of output format is allowed via the FORMAT button. It is important to furnish the user with this type of explicit control in order to discourage wasteful underground activity in beating the system. The worst thing about such activity is that it comes to be regarded as a way to optimize system use, that is, to "play the instrument properly." To illustrate this point: on command, the Culler–Fried system will display numeric values of components of vectors, one per line, on the console scope. If the vector is complex, two numbers appear per line, one for the real part and one for the imaginary part of the component. I recently mentioned to an experienced user that displaying two numbers per line, for, say, abscissa and ordinate values of a real function was difficult. His response was that the problem was transparent if one really had a "feel" for the system: we load the X vector register with an abscissa vector, the Y vector register with the ordinate vector, then go to the complex level where these will be considered by the system as a single complex vector, and displayed accordingly. Playing the instrument this way is pernicious, for by the same token, to display numbers four to a line, we would have to introduce quarternions into the system.

The TOC terminal keyboard consists of 65 operator keys, plus a full typewriter keyboard, as shown in Fig. 2. The operator keys are engraved with a number only; the name of the canonical system operator associated with each key is engraved directly under that key on a keyboard face plate overlay. Whenever the user has arrived at a comfortable keyboard definition, by replacing the system operators under function keys with programs of his own, by assigning typed names to any operators and data entities, or by creation of display values for any keys, he may store the entire keyboard configuration with a name such as SESQUIPEDALIAN. At any later time, if he does not delete this name or use it to name a different keyboard state, he may restore the keyboards to the saved state by pressing the key labeled INVOKE and then typing SESQUIPEDALIAN.

A complete description of the TOC system as it will look to the user is contained in the Users' Guide [10].

TOC is in operation on an IBM 360 model 50 computer having 256 K bytes of fast core, 1 M bytes of large capacity storage, and a high-resolution timer to allow time-sharing in the TOC system. Figure 3 shows the layout of this machine.

The underlying system design of TOC [11], is tailored to the environment of OS 360, fixed-task version. TOC, doing its own time-sharing to support its consoles, will comprise one task. A real-time experiment monitor for the Cambridge Electron Accelerator will be another task. A third task will be

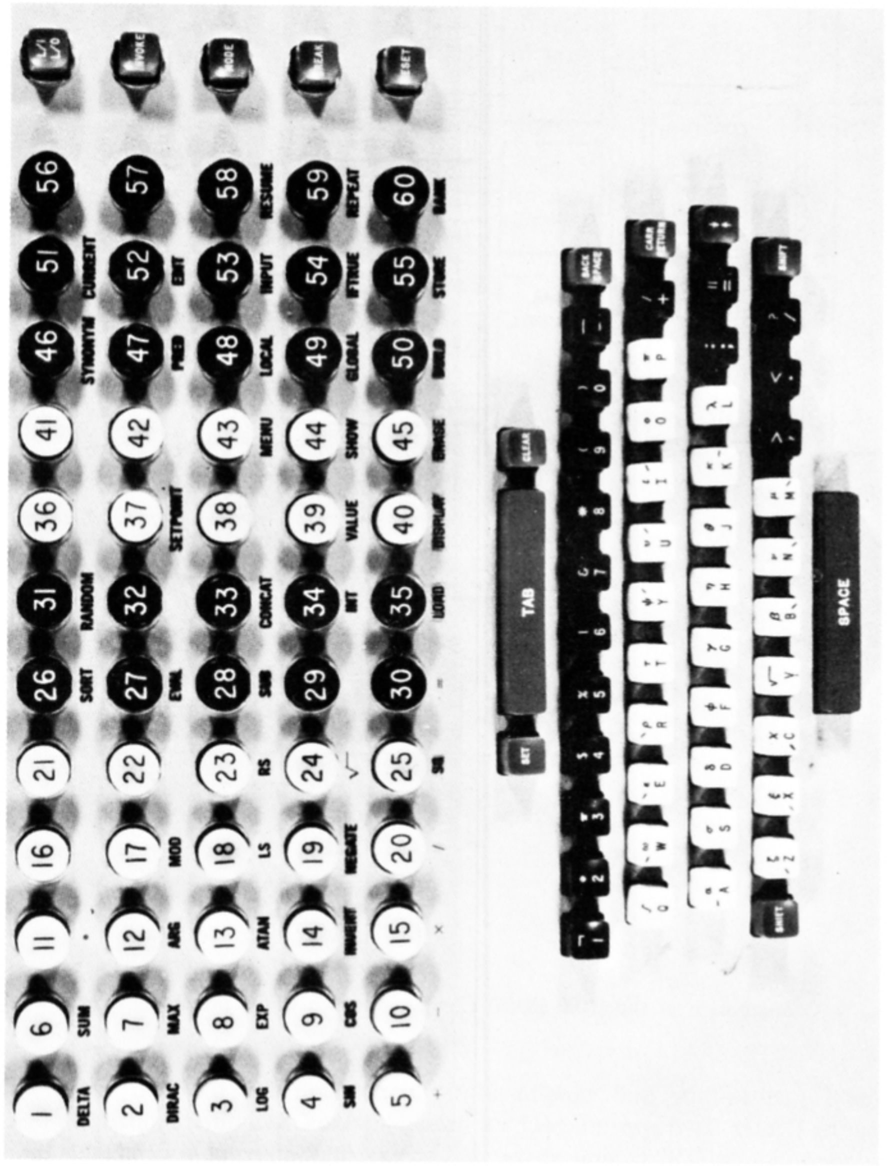

FIG. 2. TOC console keyboard.

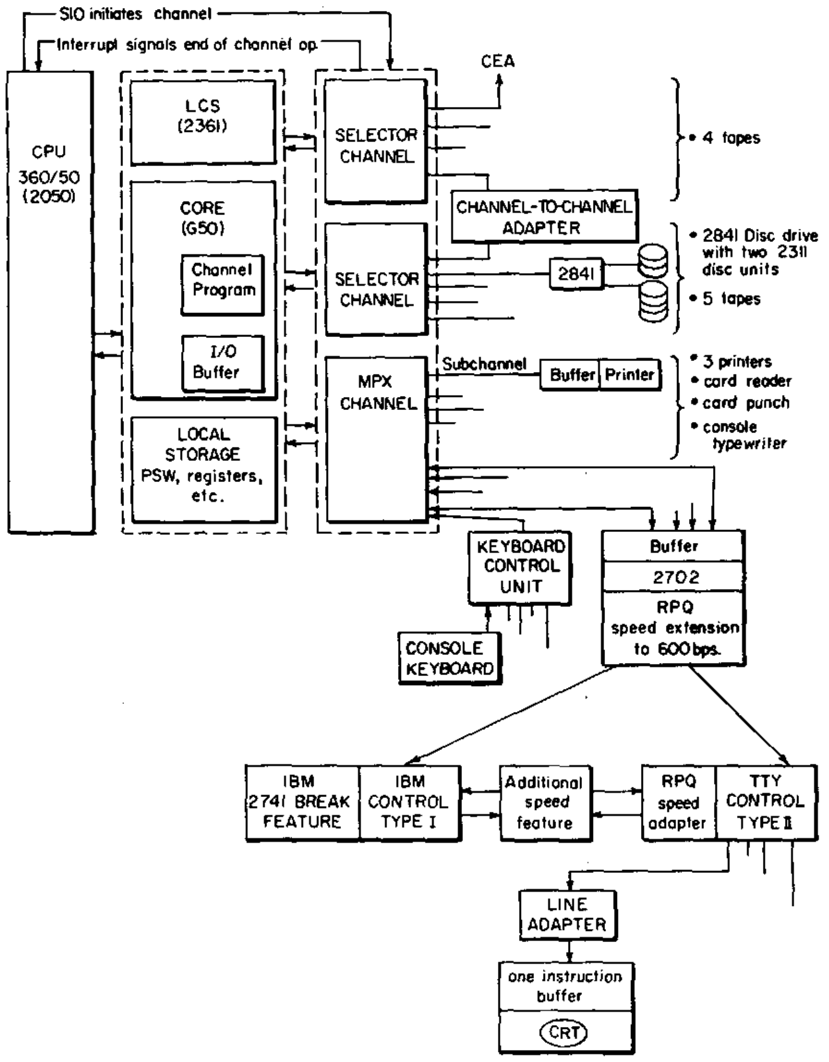

FIG. 3. Configuration of the IBM 360/50 Computer, which is the present host of TOC.

peripheral card-to-tape and tape-to-printer conversion for the Harvard Computing Center. Two appendages have been added to OS: a timer interrupt appendage to allow TOC to time-share, and an external interrupt appendage to facilitate processing of the real-time keyboard inputs.

Within the TOC task there is, for each active TOC console, an independent stream of activity called a *process*. A process is initiated when a user logs into the system and terminates when he logs out. The tables and data used by a process are saved for the user, between console sessions, on disk.

Services required by processes for keyboard input, scope output, and memory allocation are handled by the TOC executive, as shown in Fig. 4.

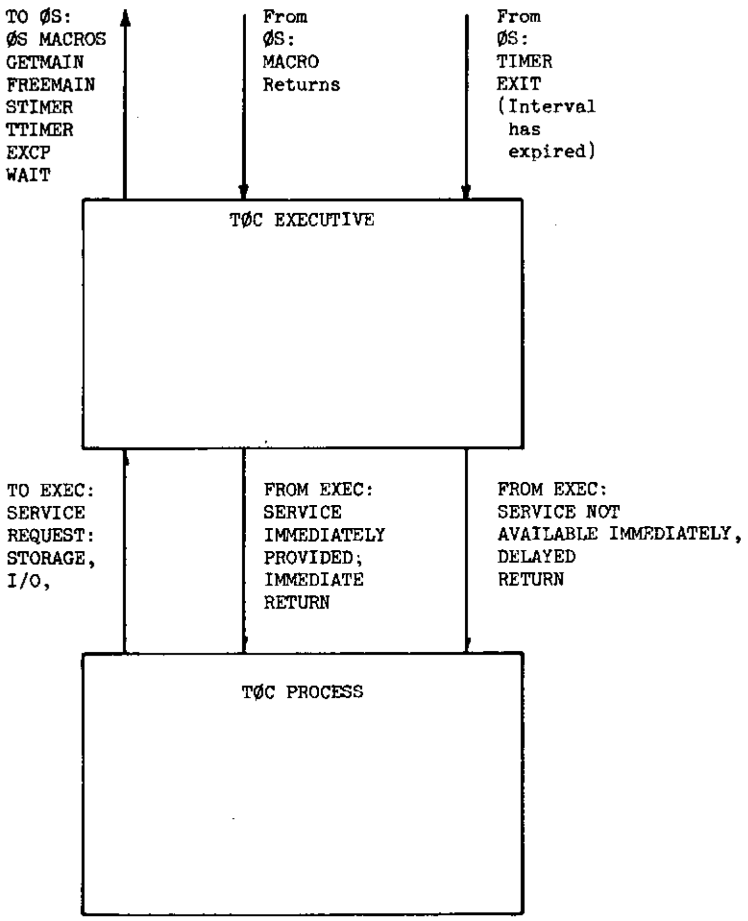

FIG. 4. Gross organization of the TOC system.

The executive also handles time-sharing between processes. Figure 5 shows roughly how the executive performs its time-sharing and service functions.

Within a process, input for a single keyboard is interpreted. The interpreter itself is a collection of several dozen reentrant routines, shared by all processes, which variously: (1) get the next operator specification from the process'

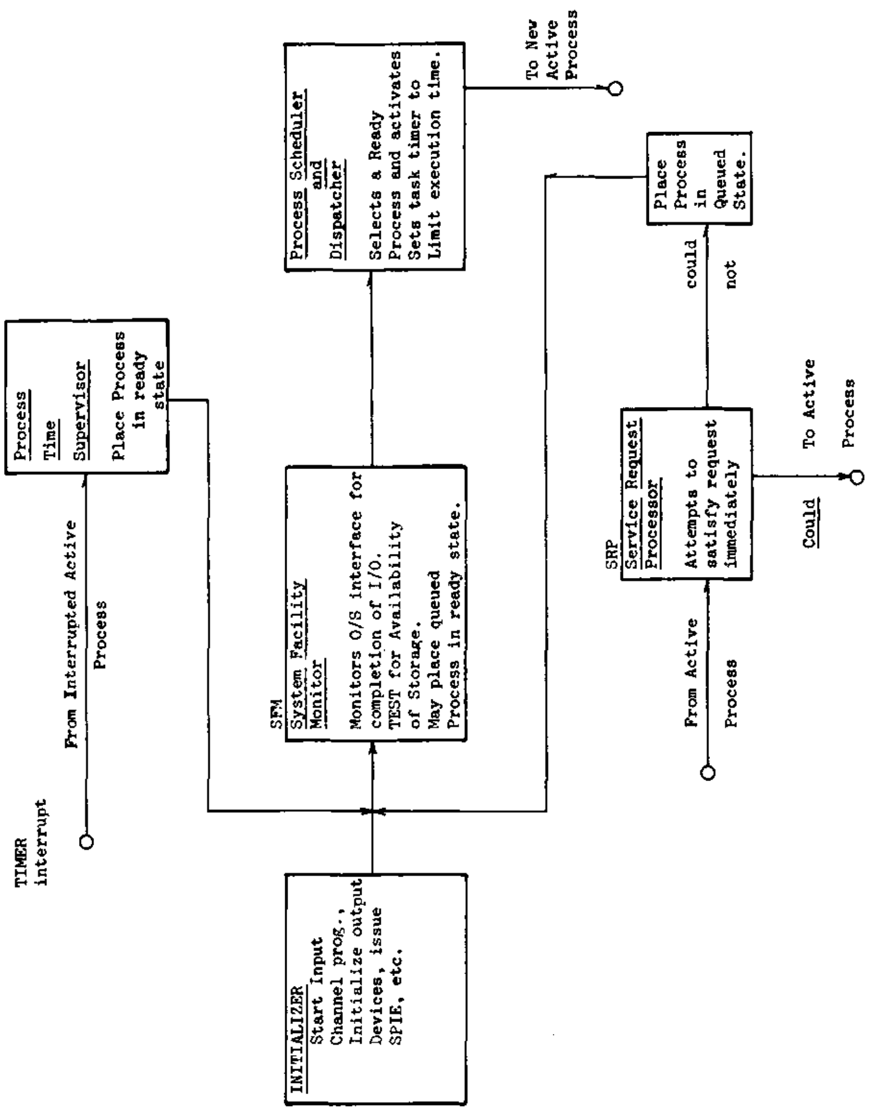
TIMER interrupt
From Interrupted Active Process
Process Time Supervisor
Place Process in ready state
INITIALIZER
Start Input Channel prog., Initialize output Devices, issue SPIE, etc.
SFM
System Facility Monitor
Monitors O/S interface for completion of I/O.
TEST for Availability of Storage.
May place queued Process in ready state.
Process Scheduler and Dispatcher
Selects a Ready Process and activates
Sets task timer to Limit execution time.
To New Active Process
Place Process in Queued State.
could not
SRP
Service Request Processor
Attempts to satisfy request immediately
From Active Process
Could
To Active Process

FIG. 5. General organization of the TOC Executive.

input string and call the appropriate operator routine, (2) perform the operation, and (3) get arguments next in the process' input string for operations which require them.

The following scenario of the events set in motion by a button push may clarify the the functioning of the system:

1. A user, Joe, who has logged onto the TOC, pushes the button SHOW, and types "ABC." His intention is to display the characters ABC on his console scope.
2. The codes for his 6 button pushes are sent by the keyboard control unit, mixed with any concurrent keypushes from other consoles, to the computer, where they are stored immediately by a continuously running channel program in the system input buffer.
3. CPU control is continually bouncing between the TOC executive, active processes, and (briefly on interrupts) OS. A process is executing. An external interrupt occurs, signalling TOC to prepare to process input. Such interrupts occur several times per second, independent of console activity.
4. OS gains control, sets a bit in a TOC executive table, and returns control to TOC. The interrupted TOC process resumes where it left off.
5. A timer interrupt occurs, signaling the expiration of a TOC time slice. OS gains control, then passes it to the TOC executive. Among other things, the executive detects the input reminder bit which was set earlier, and invokes its input director subroutine.
6. The input director distributes all new keypushes from the system input buffers. The process input buffer for Joe's process receives the keypush codes for SHOW "ABC."
7. Joe's process is given a time slice by the executive. It begins execution in the interpreter. One routine of the interpreter looks at his process input buffer, encounters SHOW, and calls the interpreter display routine. The display routine in turn calls several other interpreter routines to read "ABC," and establishes a display file. The display routine then calls the TOC executive for output service.
8. The service request processor in the executive finds the scope free at Joe's console. It sends to this scope the display file created in Joe's process. Since Joe for the moment has nothing further to do, his process relinquishes control back to the executive.

The entire scenario is played in a small fraction of a second. As more consoles are added to TOC and response becomes more sluggish, the system will have to be (to some extent) reprogrammed to maintain a lively response for each user. For instance, the scheduling algorithm used by the executive will probably be refined from its initial implementation as an equal-time cyclic succession among all processes. TOC, however, has been designed to allow this kind of reprogramming, and we anticipate considerable growth in service from the system as user demand develops.

ACKNOWLEDGMENTS

The preparation of this paper was supported by the Advanced Research Projects Agency, through Project TACT under Harvard University Research Contract SD 265, and through Project MAC under Office of Naval Research Contract No. Nonr-4102(01).

The author wishes to acknowledge the contributions of A. G. Oettinger, Robert Fenichel, Kenneth Winiecki, Robert DesMaisons, Arra Avakian, Inez Hazel, Randall Spitzer, and Don Davis.

REFERENCES

1. Covington, M.V., and Crutchfield, R.S., Facilitation of Creative Problem Solving, *Programmed Instruction* 4, 3–10 (1965). Association Press, New York, 1965.
2. Culler, G. J., and Fried, B. D., The TRW Two-Station, On-Line Scientific Computer: General Description, *in* "Computer Augmentation of Human Reasoning" (W. D. Wilkinson and M. A. Sass, eds.). Spartan Books, Washington, D.C., 1965.
3. Fried, B. D., STL On-Line Computer, General Description, Vol. I, TRW/Space Technology Laboratories, Redondo Beach, California, 1964.
4. Farrington, C. C., and Pope, D., STL On-Line Computer, User's Manual, Vol. II, TRW/Space Technology Laboratories, Redondo Beach, California, 1964.
5. Winiecki, K., (ed.), TACT On-Line Computing System, Aiken Computation Laboratory, Harvard Univ., Cambridge, Massachusetts, 1966.
6. Bossert, W., Application of the Culler Simulator to a Problem in Population Genetics. Aiken Computation Laboratory, Harvard Univ., Cambridge, Massachusetts, 1965.
7. Bossert, W., and Schwartz, W. B., Relation of Pressure and Flow to Control of Sodium Reabsorption in the Proximal Tubule, *Am. J. Physiol.* **213**, No. 3, 793–802 (1967).
8. Fried, B. D., On-Line Root Finding in the Complex Plane, Rep. No. 9990-7308-R0000, TRW Systems, Redondo Beach, California, 1966.
9. Ruyle, A., Brackett, J., and Kaplow, R., The Status of Systems for On-Line Mathematical Assistance. *Proc. Natl. Conf. ACM* (1967).
10. Winiecki, K., TACT On-Line Computer: User's Guide, Aiken Computation Laboratory, Harvard Univ., Cambridge, Massachusetts, 1967.
11. Spitzer, R., TOC System Manual, Preliminary Version, Aiken Computation Laboratory, Harvard Univ., Cambridge, Massachusetts, 1967.