An Interactive Graphical Display Monitor in a Batch-Processing Environment with Remote Entry

ALAN H. BOND, JERRY RIGHTNOUR*
Carnegie-Mellon University,† Pittsburgh, Pennsylvania

AND

L. STEVEN COLES
Stanford Research Institute, Menlo Park, California

A graphic monitor program is described. It was developed at Carnegie-Mellon University for the CDC G21 computer, which is a general purpose, batch-processing system with remote entry. The existing G21 system and the graphics hardware are described. The graphic monitor is a resident auxiliary monitor which provides comprehensive managerial capability over the graphical system in response to commands from the human user. It also will respond to commands from a user program through a similar interface, where routine calls take the place of manual actions. Thus the human and program can interact on a symmetrical and equal basis through the medium of the graphic monitor.

The choices made in designing the graphic monitor, given the constraints of the existing hardware and computer system, are discussed.

The structure of the monitor program and the human and program interfaces are described. There is also a transient swapping version with a small resident part, and provision for swapped used submonitors.

KEY WORDS AND PHRASES: graphics, graphic monitor, man/machine interaction, graphic interface, graphics in batch environment, design of graphical system
CR CATEGORIES: 4.30, 4.31, 4.39, 4.41

1. Introduction

In this paper, we describe the design of a monitor program, developed for the CDC G-21 computer at Carnegie-Mellon University, for managing the graphic hardware system and for providing an interface between the graphic system and programs running on the G-21. The G-21 is not a dedicated machine; it handles teletypes, filing, and batch queues of programs for the user community. The scopes can be used off-line by a human to set up and manipulate display material in a buffer which is part of the main addressable core of the G-21. The scope monitor is a resident monitor which provides comprehensive managerial capability over the graphical system for the human user. It further provides an interface between any program and the graphical system, so that human and program can interact easily. The interface is designed to present as nearly as possible the same interface structure to the program as to the human, i.e. to be a "symmetric" interface. Each action of the program is a routine call and corresponds to a manual action available to the human.

The design of this monitor has provided a study of design of a graphic software system given the constraints of the existing hardware and given the existing computer.

In Section 2, we describe the graphical hardware; in Section 3, the characteristics of the G-21, both software and hardware; in Section 4, the conception and design of the scope monitor and the human interface; in Section 5, the interface to user programs and interaction of human with program; in Section 6, a transient swapping version of the monitor; and in Section 7, the provision for user submonitors.

2. Graphics Hardware

The hardware used in the system under discussion has been described in [1] by J. T. Quatse and also in various Philco documents [2], and in the *Computer Display Review* [3]. Characteristics and properties of the hardware system are briefly set forth here, mainly insofar as they are relevant to the design of the software. The scopes, Philco 512's (Figure 1), were designed primarily for on-line work. The system consists of a controller (scanner) and three scopes. They have character and straight line ("vector") generation. There are 256 characters which can be displayed in three sizes. Vectors can be drawn with any length on a raster of 1024 × 1024 on a 10″ × 10″ screen. The display material is held in an 8K buffer of 32-bit words which is addressable by the G-21 central processor. Each vector is represented by one word, and a composite figure is represented by a string of vector words in successive locations. The first word in the series, called a "header," refers to absolute coordinates on the screen and serves merely to initiate the vector string. The rest of the vector words give increments ΔX, ΔY by which the beam is

The work reported here was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense: Contract Fe-44620-67-C-0058 and is monitored by the Air Force Office of Scientific Research.
* Present address: General Electric Company, Phoenix, Arizona
† Department of Computer Science

moved, with a bit to indicate whether visible or invisible. No absolutely referenced vector strings are possible. Character strings consist of a header giving the starting screen coordinates of the string and then a series of character words, each containing three characters. Character strings are automatically formatted on the screen by the hardware. A new line is automatically started when the string reaches the edge of the screen or user set margins. Margins are represented simply as characters and are members of the string. Both vectors and characters have an extra tag bit which causes them to blink or intensify if the blink or intensify manual switches are set. The scope is manipulated manually by using several devices. First there is a set of manual switches (the "state switches"), 32 in all, constituting the *state word*. By appropriate settings the human can initiate the entry of vectors using cursor control switches or the Rand tablet, and the entry of characters from the keyboards. The modified Rand tablet controller allows continuous entry of vectors into the buffer giving curve drawing by hardware alone [13] (shown in Figure 1). There is also a lightpen, for pointing only. There are automatic off-line facilities provided by the scope hardware for inserting new material in the middle of existing strings, deleting or correcting existing material. In the buffer, the display material is arranged in blocks which are stored in sequence in the buffer with scanner controlling words directing the scanner in a regenerative path around the buffer. Each scope can have up to four independent blocks of display material associated with it, called its "pages." Which pages are visible at a given time is controlled by four of the state switches, one for each page. Each block of material has an identifying command, called a "delimit," in front of it indicating on which page(s) of which scope(s) it is to be displayed (see Figure 2). The manual entry of material into a page is controlled

FIG. 1. Photo of scope 1 and Rand tablet

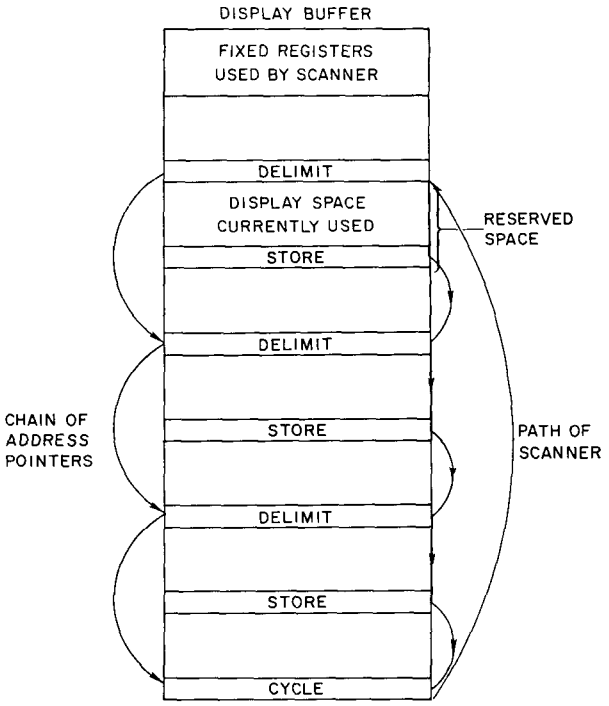

FIG. 2. Schematic diagram of delimits, cycles, blocks, and stores

by whether or not the page is visible and whether or not it is enabled, i.e. whether an enable bit is set in the delimit word of the page. The new material is entered by the scope hardware and the blocks are arranged to extend as new material is entered. until they hit against the delimit of the page next in the buffer. When this happens, an interrupt bit is set by the scanner. There is also a compare interrupt feature. To use this, a compare command is put into a given block and designates a given character. When an instance of this character is next entered into this block, an interrupt is generated. Also available are 20 manual interrupt buttons, so that 22 different types of interrupt can be generated. These are distinguished by different bits in the (32-bit) *interrupt word,* which also has room for a copy of the compare character in the case of a compare interrupt, since compares can be set for several different characters in the same block.

In addition to the state word and the interrupt word there is associated with each scope a *position word.* This contains the current X, Y screen coordinates of the cursor and also the current settings [0, 63] of two "analog knobs." The latter give the ability for the communication of quasi-continuous information from human to program.

These three words for each scope occupy fixed (addressable) locations at the beginning of the buffer. There are actually two sets of these words for each scope, each set corresponding to a *mode,* which can be "normal" or "alternate." This doubling up of all words and consequent multiplication of the different types of block designation turns out to be useful. Among other addressable registers we mention the *location register* of the regeneration circuitry, i.e. the location currently being scanned and dis-

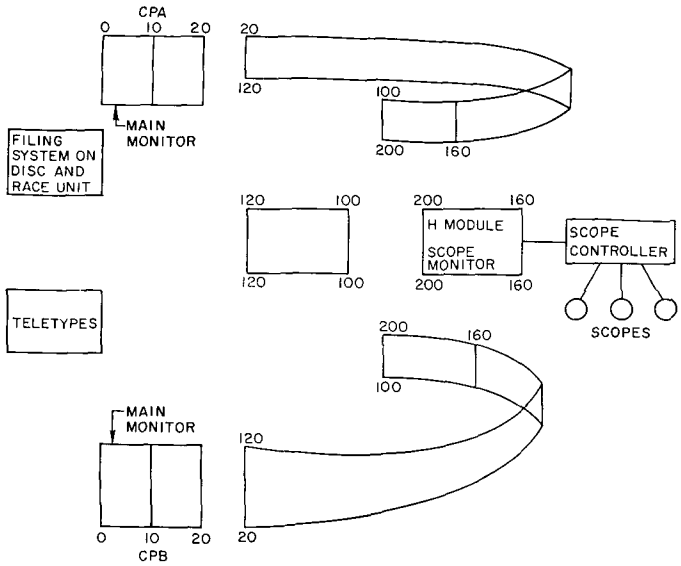

FIG. 3. G-21 and H-module—upper numbering is equivalent to addresses for CPA; lower numbering is equivalent to addresses for CPB.

played by the hardware. Using this, it is possible to tell where the scanner is and also to be able to reset it and to trap it, using software. The scanner cycles about once every 1/60 second, and when about 1000_{10} words of display material are scanned, flicker is noticeable but not intolerable.

3. G-21 Hardware and Software

As shown in Figure 3, the G-21 is a 64K, 32-bit, computer with two Central Processors (CPs). The CPs each address a separate $20K_8$ of core and also the rest of the (shared) core, usually addressing as shown, but different schemes are possible and can be switched by the operator. There is a resident monitor which services about 50 teletypes and schedules a card and teletype batch queue. A user program is loaded into one of the two 25K slots and runs to completion, with all printer output going through the single accumulator. There is a large filing system on two disks and an RCA race unit; most programs are stored on personal files and merely called in by a short card or teletype job. References to files are managed by the monitor. All I/O from programs to the disk, race file, or magnetic tape are handled by the monitor. As shown in the diagram, the monitor is in three parts, two identical parts (one for each CP) and a third shared part. This shared part handles shared facilities like I/O, e.g. disk routines, and each CP can call upon these routines in the central section. In the case of conflicts, one CP waits for the other to finish with the disk routines.

The files in the system are divided into 30 classes or "types." The physical placement of these files is handled by the monitor, and the user only refers to a given type and gives the record number within that type. Different types may have different record lengths. All the user card image files are within one type, and there is a separate directory system and file manipulation language ("AND") for these, on top of the monitor facilities. AND is not conversational; AND programs for manipulating files and running programs from files are themselves run as batch jobs.

The monitor services teletypes by obeying a small control language (each sentence of which starts with the keyword $\$=$) which enables the human to setup his program as a teletype "input file" on the disk. Thus each remote has a single input file containing a program. By another command the program is given a submission time, and the set of submission times constitutes the teletype queue. The monitor schedules by simply taking the lowest submission time. Associated with each remote is also an "output file" of length 200 cards; a user program can output material into this file, each card giving one line which can be inspected from the teletype using the $\$=$ language.

From each remote, only one program at once can be in the queue; a new program can be submitted only after the previous program has run. Thus, when using a teletype, the user has to submit his program and then wait until it runs. The maximum allowed execution time is 6 minutes (this is a 6 μsec machine) and this turns out to take about 12 minutes of real-time to run. The wait-time (i.e. turnaround-time) is usually about 60 minutes. All compilers produce nonrelocatable code to run in either a 25K slot or two 25K slots; so an ALGOL program, say, can occupy almost the entire 64K of core.

The interface between the user program and the monitor is structured into a fixed set of routines and locations which are used in the standard ways, and this is the only permitted means of interaction. These routines and locations are also given the same names (always of the form | n) in all systems and languages that can access them.

Interrupts of the CPs are handled by an interrupt word for each CP and all interrupts from the scopes set the same bit in one interrupt word. The scope monitor can be executed by either CP, chosen by manual switch settings at the G-21 operator's console. The scope buttons will then set the interrupt bit on the appropriate CP. Discrimination between scope interrupts is effected by their different settings in the scope interrupt words in the display buffer. The display buffer is a switchable 8K module which replaces a regular core module when bits in a certain G-21 hardware ("status") register are set. Thus a given G-21 machine command referring to an address in [160000_8, 177777_8] with one setting of the status register refers to one core box and on the other setting another core box. The status register can be set by a machine command. Thus a program can use the entire 64K of regular core including addresses 160000_8, etc., and can also use the display buffer (also with addresses 160000_8, etc.) by switching it in as needed, using the status register. A program can switch in the display module and use ordinary machine commands to set up or read display material. The accessing of the display buffer by the G-21 CP and by the scope scanner is interleaved if there are any conflicts.

4. Design of the Scope Monitor

Given the graphic hardware and the G-21 hardware and software, software for graphics had to be designed with several purposes in mind. It was desirable to use the scopes for graphical activities, like drawing essentially independent of the CPU. Without software support, the management of the buffer entirely off-line can be laboriously effected using hardware switches on the controller; so software aid was needed. This would presumably be supplied by resident code and requested by interrupt.

Also wanted was the capability of saving display material on files and using the scopes as teletypes for submitting programs into the batch queue. Arrangements for user programs interacting with the scopes were also desired.

It was decided to implement a monitor program auxiliary to the main monitor, and, in the initial design, resident in the actual display buffer. Thus the display buffer would contain code and display material. The code would contain entry points, and control would be passed to it from the main monitor when a scope interrupt was detected. Requests to the scope monitor would be made by the human, usually by interrupt button, and the functions available for request would include management of the display area, saving of display material on files, submission of programs, etc. The scope monitor would service the three scopes individually and simultaneously. The scope monitor would also respond to requests from a user program, and thereby provide an interface between a human and a user program.

ORGANIZATION OF REQUESTS. The scope monitor is structured as an interrupt service routine (ISR) with a fixed entry point and, corresponding to the many actions performable by it, a set of routines referenced through tables in a data area. The requests are organized into sets of 10 or so, there being 20 interrupt buttons, and the concept of *state* introduced. In a given state, the interrupts have certain meanings specific to that state. The meanings are indicated to the human user by explanatory (system) displays, like menus (Figures 5, 6, 9, 10). In every state, interrupt number 1 causes the system display to be visible or invisible. The change from one state to another is effected by interrupt, and as there are only six states implemented, this is accomplished by defining an *option state* the meaning of whose interrupts is to select a given state (see Figure 4). Interrupt number 0 in any state returns one to the option state (see Figure 5). Thus the monitor keeps a *counter indicating* which state it is in, and associated with each state, a table of routine entry points to be entered upon interrupt. The interrupts are organized into two priorities: (1) interrupt 0 and memory-full and compare interrupts, given high priority and called "immediate interrupts"; (2) all others, called "regular interrupts."

OPTION PAGE

PRESS INTERRUPT NUMBER
2. MANAGEMENT STATE
3. PROGRAM STATE
4. DEBUG STATE
5. USER MANUAL
6. USER PROGRAM INTERACTION STATE
7. TEXT EDITING STATE
8. LOG OUT

FIG. 4. Option page

ORGANIZATION OF THE DISPLAY SPACE. The core not occupied by code is organized into an available display space, with fixed size blocks of length 160_{10} words, initially as one "glob." In addition, copies of the system display messages are resident and part of the data area. Thus the regeneration cycle executed by the scanner goes through the system messages and then jumps to the regular display space. System pages can be made visible or invisible to any combination of the three scopes by setting appropriate bits in their corresponding delimit words.

The user can request display space to be reserved for a given page by asking for a certain number of blocks. He can also delete any of his pages and can enable or disenable any of them for manual entry. All user pages are in normal mode, and the system pages are disenabled and in alternate mode. The monitor keeps tables of space allocations and a table of available "globs"; no repacking is performed. In addition to space used for user pages, the scope monitor uses globs of core for working areas, not always display material, and these it obtains from and returns to the display space.

Requests involving core space and disk files are made in the management state (Figure 6). Note that the reorganization of the display space by the scope monitor can in certain cases present a malformed regeneration path to the scanner. By being able to address the register containing the working location of the scanner, the scope monitor can trap the scanner momentarily while it alters the buffer.

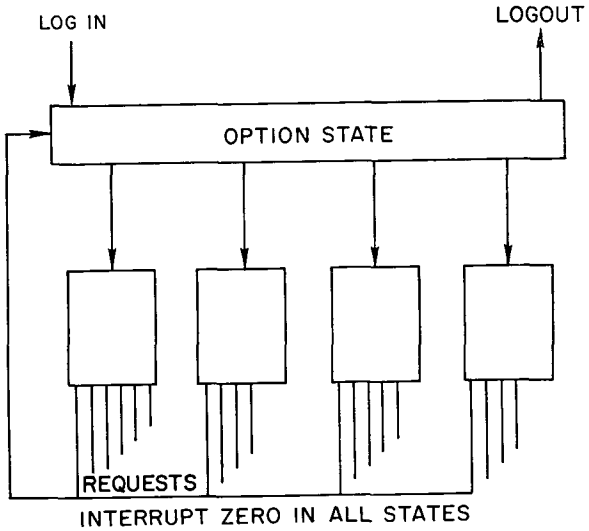

FIG. 5. State hierarchy

INPUT OF PARAMETERS FOR REQUESTS. Many of the requests have parameters which are typed in from the keyboard. This is achieved by having three small display blocks in the data area and in the regeneration cycle as input buffers for the three scopes. When input from a given scope is required, all other display blocks for that scope are temporarily disenabled by altering the delimit bits and are made invisible by altering the page switches in the state word, and the input buffer is enabled as a page. It actually is designated as Page 1, but this cannot conflict with the user's own Page 1. The cursor is also appropriately set, over the system message, by setting the cursor position part of the position word. Thus the user sees the

```
MANAGEMENT PAGE
PRESS INTERRUPT NUMBER
 2. SAVE  PAGE  AS  SCOPE  FILE
 3. READ  IN  SCOPE  FILE  AS  PAGE
 4. APPEND  PAGE  TO  PAGE
 5. DISPLAY  DIRECTORY  OF  SCOPE  FILES
 6. GET  BLOCKS  FOR  PAGE
 7. ENABLE  PAGE
 8. DISENABLE  PAGE
 9. DELETE  PAGE
10. CREATE  BLOCKS  FOR  SCOPE  FILE
11. SETUP  AND  FILE  FOR  TEXT  EDITING  USER  SHEET  TAPE
12. PUTBACK  AND  FILE  FROM  TEXT  EDITING  USER  TAPE
```

FIG. 6. Management page

system display with a gap for a parameter and the cursor, but the cursor is actually part of the input buffer, another display block in the display memory. The user then types in characters, apparently filling in the space in the menu, like filling in a form. However the form itself is unaltered, and the characters go into the input buffer page.

In the buffer, a compare instruction is set for the return character, and when the user has typed in the characters he enters a return character which triggers the scope monitor to process the input information and return display conditions to normal. The memory-full interrupt's meaning during this time is set to go to the same entry point as the compare interrupt, in case the buffer is filled accidentally.

THE DISK. To save pictures on files, each scope user is allocated 20 files of indefinite length; each file can hold one picture only. The transfer of a given page to a given file and vice versa is requested by interrupt. The scopes are allotted disk space in the G-21 system and the scope monitor manages it; however, all disk transfers are effected by having the scope monitor pass a request to the main monitor giving the relative addresser type and record number. The disk available space with the allotment is managed on a glob basis with blocks of size 160_{10} words and with an available glob list, with no repacking. The user can request a display of a directory of his files.

The request to the main monitor for a disk transfer is made on the same basis as the sharing of common routines by the main monitors on the two CPs, viz a binary switch is checked, and if set, the scope monitor "waits" until it becomes clear, before entering the communal disk routine.

WAITING AND THE STACKS. Many requests involve the execution of a set of routines in the middle of which we may have to wait, either for the disk routines to become available or for the human to type in a parameter. The wait-times may be on the order of many seconds, and we want to be able to process other interrupts during this time including those for requests from other scopes involving waiting. This is arranged as follows (see Figures 7 and 8). All variables local to routines are put in a stack,

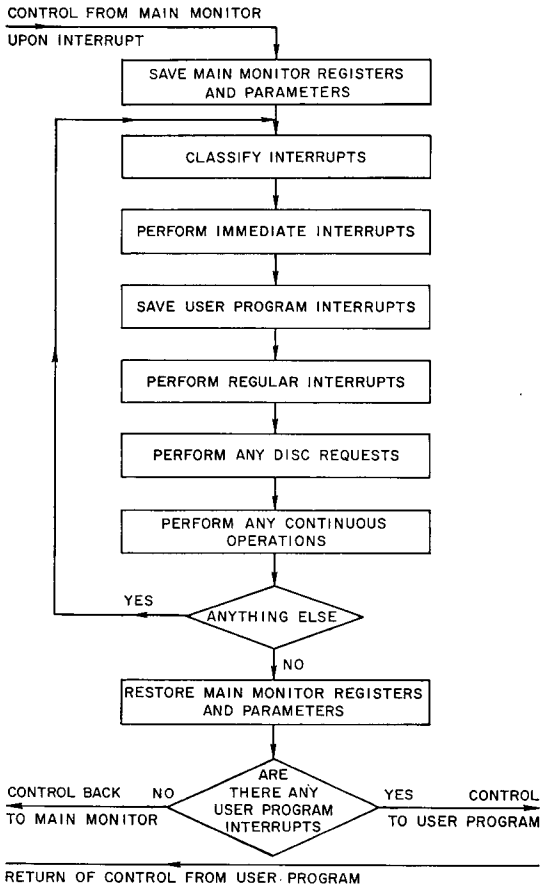

FIG. 7. ISR flowchart

and there is a separate stack for each scope. Thus data, which needs to be set up and used later after a wait during which the same routine is executed but at the request of another scope, will be safe. Next, whenever we reach a point in the code which potentially could cause a wait, we merely set up an internal request to do the operation and then return to the ISR, look for other requests to satisfy, and if none, return to the main monitor. In the ISR, we find the internal request and see if we can carry it out immediately; if not, we leave it alone, and next time through, try again. In addition to the manually generated interrupts, the scopes also receive clock interrupts, one per second. This is not frequent enough to allow us to

dispense with manual interrupts. The typical response time is negligible; whereas, when based on clock interrupts only, the response is sluggish and the whole rhythm of human interaction is different.

When the internal request is completed, we return to the point in the code execution where the request was set up and continue. The disk requests are completed by the disk routines becoming available, control being transferred to them and then returning. The manual type in requests are completed by triggering either a compare or a memory-full interrupt. In this case, control from the ISR is passed to a routine set up during the request.

The return to the ISR during an internal request is effected by setting a global return location, since control can return to and leave from the ISR at any of a number of points. The ISR is divided into a number of sections, each of which has a return location which is set as the global return location upon entry into the section. The execution location at which the internal request was initiated can be found from the stack, since all routine return locations are also stored in the stack. The stack increment is also stored in the stack, since it is different for different routines.

Finally, at the end of the total request, which may have involved several waits, control is passed to an exit routine which cleans up and returns control to the ISR.

Continuous Operations. There is also general provision for *continuous operations*, which are routines not involving waiting but are executed every time control passes through the ISR, i.e. roughly once per second. An example of this is the continuously generated core dump in the debug state, Figure 9.

The Debug State. This provides requests for patching the core and doing a routine transfer (the "TRM" in Figure 9) as well as the continuous core dump, which can show any part of the entire core, and in fact one can watch the execution of programs on the other CP!

Recovery and Errors. The scope monitor may acquire working space for use while it is in a state. In this case, it sets a "state removal routine," so that the process of changing state involves first executing the currently set state removal routine. During an interrupt request, routines may also get working space. All working space is referenced in fixed working tables, and its removal is performed by the global exit routine. In the case where the user makes one request involving a wait and then changes his mind before the completion of the request by making a further and more immediate request, the scope monitor has to find out what had been reserved and then delete it. This is always possible due to referencing the space in a standard way and knowing that any space referenced from there must be removed at the end of a request or during a change of request. In the case of errors, a similar clean up is done.

A small number of error messages is used for display to the user. Errors fall into the categories of (1) unacceptable requests, (2) undefined requests, (3) overflow of space requirements, (4) facilities not yet implemented or temporarily out of system, (5) "impossible" system errors.

Submission of Programs. The scope can be used as a teletype. Interrupt requests are provided in the program state (Figure 10). To submit programs into the teletype queue, the program is typed onto a display page and then an interrupt request takes this material and converts it to G-21 characters, formats it as card images, and places it in a *teletype input file* on the disk corresponding to the scope. Thus each scope is allotted a remote number in the G-21

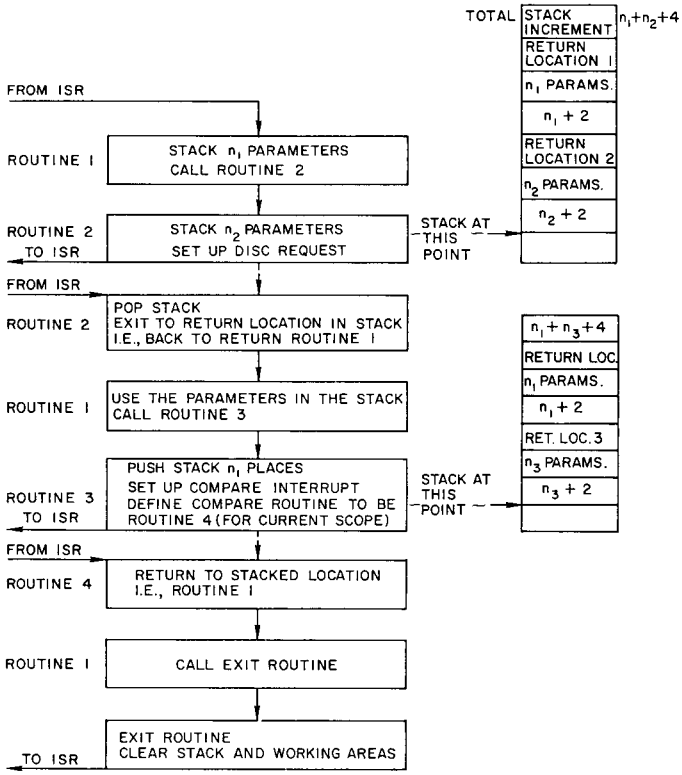

Fig. 8. The processing of a typical request—solid line, actual control; broken line, apparent control.

```
DEBUG PAGE

0. OPTION PAGE
1. CLEAR INPUT
2. STORE INPUT
3. LOAD INPUT FROM MEMORY
4. SWAP INPUT
5. TRM

00000000000

005344 00000000467 00000073626 00000001453 04050005632
005350 01550000100 01730005632 00050000100 05550006732
005354 01730006732 01770007546 00000000000 00050000004
005360 00170005353 00050000002 01770076666 01770005300
005364 00000000000 01770003106 01770003106 00170004312

THIS SPACE RESERVED FOR SYSTEM MESSAGES
```

Fig. 9. Debug page

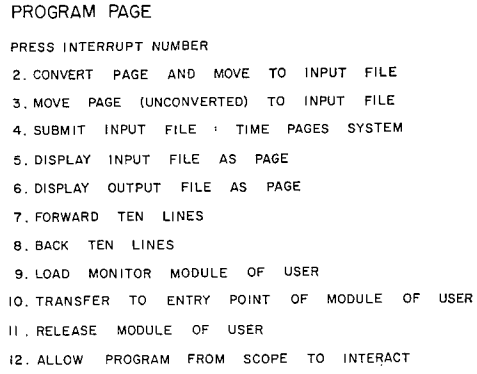

FIG. 10. Program page

system and provided with input and output files on the disk, along with all those for other remotes, within a relative addresser type reserved for that purpose. The action of placing this program in the queue is performed by a separate request, which takes a parameter and automatically sets up the correct job card as well as notifying the main monitor of the submission time.

Because the user wants to interact with his program at run-time, we did not want the usual wait-time for teletypes. We solved this problem by making the scope programs have priority—giving them an artificially low submission time, so that a scope program when submitted is at the top of the queue. The user then only has to wait for the currently running program to terminate, about 6 minutes on average. So that this priority is not unfair to teletype users, since this is not a dedicated machine but a general machine serving the entire user community, further submissions from the same scope are prevented (a request would elicit an error message) until a time had elapsed equal to that which a teletype user would have had to wait.

The perusal of input and output files, each with 200 card images, is effected by requests to select the file, placing 5 blocks (about 30 lines) from the file on a display page and then pressing a button interrupt for forward and backward motion of 1 block, which is about 6 lines. Pressing the button causes the test to be replaced by the updated sample of the file, giving the impression of "roll round." The request, involving one disk access, takes about half a second.

TRACING THE MONITOR. To find some of the harder bugs, we have switchable trace printing into the scope monitor. This is a tricky operation, since the code assumes that it will not be interrupted during some segments of code and the interface between the main monitor and the scope monitor is very complicated and again assumes noninterruptibility at times. Of course, printing involves interruption. We get around this by (1) preventing the processing of more than one interrupt at once by the scope monitor and (2) making the printing be officially performed by a special user program; thus the scope monitor passes printing material to this program and triggers it and waits in an interruptible state until printing is completed. In this way we are able to obtain comprehensive printing of the action of the scope monitor, and, during debugging, this is a great asset.

5. User Program Interaction

Having supplied a comprehensive set of requests for the human user, activated manually, we then provide a very similar set of requests for any user program running on the G-21. This is achieved by an interface of entry points to routines called, in all available languages, "B-routines." All interaction with the scopes by a user program is restricted to calls upon these routines. A list of B-routines and their meanings is given in Figure 11. Display material is handled by setting up the material in a block in the user program data area and, by means of a B-routine call, requesting that it be moved to a designated page of a designated scope and vice versa. Note that neither the program nor the human use delimits; they just work with display material—headers, vectors, etc. The delimits are handled by the scope monitor as part of its management of the display area.

Conversions between the scope character set and the G-21 character set are provided. By means of the interface (Figure 12), the program can do anything a human can do. Eight of the switches are not used by the scope hardware and are for use in human program interaction.

We have the following means for human program interaction:

(i) By mutual display.
  (a) Each displays text to be read by the other.
  (b) Each displays a general display of lines and text.

(ii) By means of hardware attachments.
  (a) Analog knobs set by human and read by program.
  (b) Cursor can be set and read by both.
  (c) The eight switches can be set and read by both.

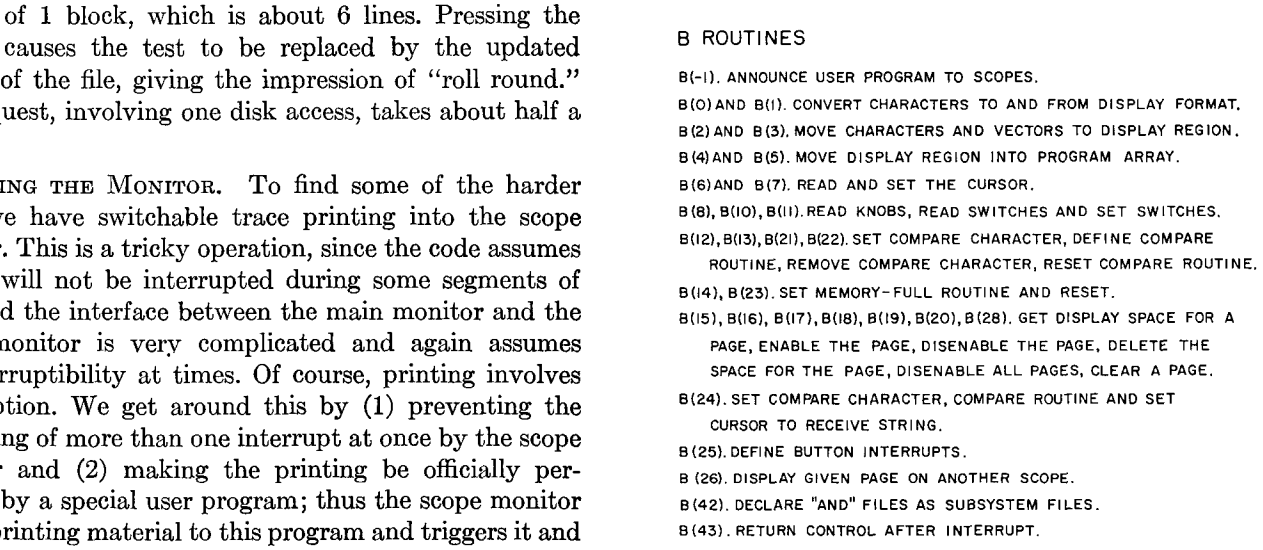

FIG. 11. List of B-routines

In addition, we have program-defined interrupt entry points:

(iii) (a) Compare interrupt on a certain (set of) character(s) on a certain page
(b) Memory-full interrupt
(c) Button interrupts 1–19.

The interrupt entry points are defined by a B-routine call. The scope monitor is put into a state, called the "user program interaction state." In this state, the meanings of the interrupts are no longer given by a "menu," but will cause control to be transferred to the defined user program entry points. Thus the program has full freedom to use all interrupt facilities and this gives fast response, interrupting being faster than checking a switch. One gets back to the option state by interrupt 0 and can change freely back and forth between interaction with the user program and interaction with the scope monitor. If a program error occurs by an incorrect B-routine call, the human is notified by an error display and the program is notified, also, by error switches and an error number. The processing of user interrupts in the scope monitor follows the normal classification procedure, but since the settings of many monitor switches, e.g. memory protect, are different for the user program than for the scope monitor and since we cannot rely on control returning to the scope monitor from the user program, we treat them differently.

The transfer of control to the user program interrupt entry point is delayed until all other processing "this time through" is completed. It is delayed until we have reset all switches, and it would normally return to the main monitor and often from there to the normal line control in the user program. At this time, we, instead, transfer control to the interrupt entry point in the user program. Control can be returned to the scope monitor by the user program directly or by a special B-routine call. This has the effect of returning control ultimately to the interrupted line of computation. Otherwise, the program continues in the new line of computation.

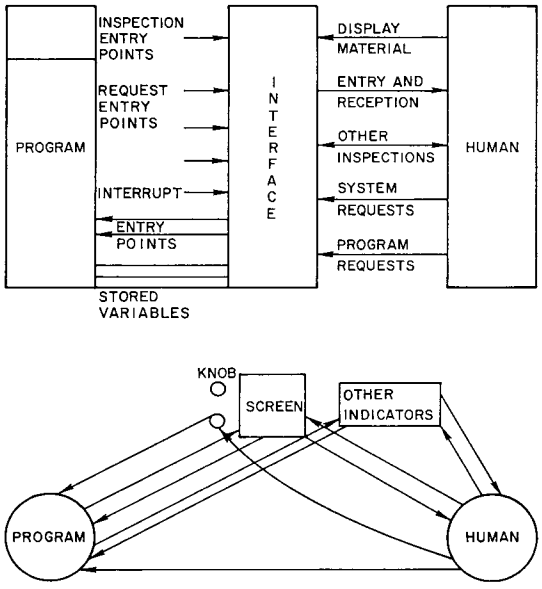

FIG. 12. Human program interface

We originally allowed different entry points for different interrupts; however, it was found more convenient to have one entry point in the user program and to pass information on the nature of the interrupts. This is done by *declaration* of communication locations to the scope monitor by the user program, by means of a B-routine.

A location in the user program data area is, for example, declared as the interrupt number. Then, when control is passed to the user interrupt entry point, the interrupt number is placed in this agreed location by the scope monitor. Other communication locations are for the character interrupting in a compare interrupt and for the number of the scope interrupting in the case of multiscope interaction. A further location is used as an interrupt control switch. When this is set the scope monitor will not pass control but will keep looking, each time through the ISR, until the switch becomes unset. User interrupts are not queued; a further interrupt, activated before the first is transferred, will cause the first to be replaced by the second.

The program can tell which scope it is interacting with, the one it has been submitted from, by examination of a register. It can interact with other scopes by setting this register to a different value and using the B-routines in the usual way. Such B-routine calls will have error exits unless permission to interact has been given by the human at the scope concerned. Permission can be given by interrupt request.

INTERACTIVE PROGRAMMING IN HIGH LEVEL LANGUAGES. The user program interface can be incorporated as a package in any language, and we have incorporated it in ALGOL and Formula ALGOL. There are a lot of problems in trying to interrupt a program in these languages. We have "B-procedures" in ALGOL and Formula ALGOL and a package of procedures in these languages giving comprehensive high level facilities. A further experimental graphical language, GRASP [6], was developed by Gene Thomas, written in ALGOL and used either as a language processing system or an outer block to any ALGOL program. These interfaces are discussed in a further paper [4].

SECURITY. Only allowed users are allowed to "log in" and nothing can be requested until a successful log in has been obtained. This is simply effected by presenting the user with a log in message so that he types in his user number. This is then checked against a file of allowed user numbers held on the disk in a master directory. Upon leaving a scope, a user logs out.

We also prevent the user in the debug state from patching anything other than his own program.

As described above, the user can interact only with a program submitted by himself unless given explicit permission by the other user. This is effected by making the program announce its presence to the scopes by a call on a certain B-routine. Only after this call can other B-routines be called or the human put the scope into the user program

interaction state for user interrupt processing. The announcement of the program causes switches to be set and these are cleared at the termination (either normal or abnormal) of the user program, by the main monitor, which passes control to the scope monitor at this time.

The main use to which the system is put is for individual users to develop *interactive programs in high level languages* for specific scientific and engineering problems. In Formula ALGOL, there is a program for graphically manipulating algebraic formulae in textbook format by F. R. A. Hopgood and a graphical inference system by L. S. Coles [9]. In ALGOL, there is an interactive pattern recognition program [10], a computer-produced movie program, an interactive job-shop scheduling program [11], and an interactive oil drilling simulation program [12]. In assembly code, the G-21 Brooker-Morris system has been given an interactive graphic input-output capability.

6. Transient Version

In a new version, close to completion, the scope monitor has been restructured as a small resident part $\sim 1000_{10}$ words and a set of relocatable modules of code (total size $\sim 4000_{10}$ words) which swap as required from the disk. This should allow a total display area of $\sim 7000_{10}$ words, which is much more than the scanner can scan without bad flicker. Resident are the data area of the scope monitor, the composite ISR, and the swapping routines. All the rest of the code for carrying out requests and also the system messages are transient. The transient code is organized into groups of routines, which are treated as swappable units. Each such "module" has a number of entry points in standard locations relative to the first word, and all references between modules must be by means of a transfer to a given entry point number. The available space is shared then by display pages, system code, system messages, and temporary working data, all of which are organized as blocks with a leading delimit. In the case of nondisplay material the second word is a zero, so that the display scanner simply jumps around the block.

When a module is swapped in, it is relocated by altering the actual code by a simple relocation algorithm, since the machine does not have base registers. Modules are never swapped out; they are overwritten when the space is needed. "Retention bits" can be set, so that the space is not released, making the module resident until the bits are unset. In this way code can be continually swapped but data retained. We minimize the number of swaps by blocking related routines together, and also we leave the module in position until the space is actually needed for something else. Modules of code are shared by all three scopes and can be retained and released by any combination of the three scopes without confusion.

Modules are assembled by a system of macros in the assembler, SPITE. This flags words which will need modification for relocation at run-time. Using an auxiliary "scope module assembly package," the modules are assembled and dumped onto disk files at assembly time, and the disk locations are automatically assembled into a table in the resident scope monitor data area. References by modules to locations in the resident part are assembled by using the symbol table of the resident part at assembly time.

7. User Submonitors

The package of macros and routines for assembling the transient version and the swapping system can be used by any user to produce a "submonitor." This can consist of an indefinite number of modules, just like the scope monitor itself. The initiation of requests to such systems is by interrupt and there are currently three interrupt specifications for this purpose provided in the program state (Figure 10). User submonitors interface with the facilities of the scope monitor by using the B-routines. They have a slightly different way of calling these routines and of address checking, and error recovery had to be generalized to deal with this case.

Reentrant modules use the scope monitor's stacks for storage of local variables; this is arranged by the macros. Modules can call each other recursively, and this is kept track of by counters in the head of the module. A module can be called recursively to a depth of 32, and it can be called by all three scopes simultaneously and independently before overflow of this counter occurs. It is also possible for a call of a routine by one scope to have an error exit and to clear the counter, etc., without disturbing simultaneous calls on the same routine by the other scopes.

Before these facilities were developed, M. Coleman wrote a text editing subsystem [5]. This is a block of nonrelocatable code which swaps into a fixed position. It provides a comprehensive set of text-editing facilities activated by interrupts or by hand drawn proofreader's symbols, which it can recognize. The text is swapped on and off a page from a special region of the disk. To move text to and from "AND" files and this editing area, auxiliary G-21 programs are automatically submitted by the scope monitor (they do not disturb the wait-time for submission of programs) upon interrupt request by the human. The main reason for this is to obtain permission to access the AND files, from the main monitor, which involves an elaborate checkout procedure. The text editor is now an established part of the graphics system; it corresponds to a special text-editing state.

8. Conclusion

After an initial design phase of six man-months, the scope monitor was started in June 1966, and two men worked on it for three months and released an initial version with limited user program interface. Then one man worked (part-time) for a further year and fully debugged it, extended the user interface, and developed the transient version. He also produced complete user documentation [7] for the system. Thus, in only two man-years, we have produced a comprehensive graphical monitor, which provides a workable and general interface for human/program

(*Continued on page 607*)

tial locks appear to be unavoidable. The major difficulty associated with such primitives is that if more than one of them operates in the data base at the same time, a deadlock is possible. One such primitive, however, may operate in parallel with other primitives using the two-step algorithm without the possibility of deadlocks as long as the other primitives do not use essential locks.

5. More Complicated Locking Procedures

Additional parallelism can be obtained in the data base if account is taken of the fact that it is often possible for several primitives to operate on a single node without interfering with each other. The most obvious example of this is primitives which do not modify information in a node. In addition, it is often the case that a primitive which changes a complete unit of information in a single instruction can be executed simultaneously with one or more primitives which only read. Finally, it may be the case that two primitives which modify the node operate on disjoint sets of information in the node, and so they can effectively be carried on at the same time.

To accommodate this type of parallelism it is possible to associate a vector lock with each node such that the ith coordinate of the vector corresponds to the ith primitive type and represents the number of primitives of that type currently operating on the node. In addition, each primitive carries along with it one or more binary vectors corresponding to the different types of nodes it expects to encounter. The ith bit of the latter vector, corresponding to a particular type of node, is 1, if the primitive carrying the vector cannot operate in a node of that type at the same time as a primitive of type i. The lock is tested by forming the dot product of the node's vector lock and the appropriate binary vector associated with the primitive. If the dot product is zero then the node is unlocked for the primitive, which then increments the node's vector lock in the appropriate position and begins operating in the node. Since the testing and setting of a vector lock cannot be done in a single instruction, it may be necessary to associate a 1-bit lock with the vector lock to prevent more than one primitive at a time from operating on the vector lock.

REFERENCES

1. Lampson, Butler W. A scheduling philosophy for multiprocessing systems. *Comm. ACM 11*, 5 (May 1968), 347–360.
2. Daley, R. C., and Newmann, P. G. A general purpose file system for secondary storage. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, Spartan Books, New York, pp. 213–229.
3. Dennis, Jack B., and Van Horn, Earl C. Programming semantics for multiprogrammed computations. *Comm. ACM 9*, 3 (Mar. 1966), 143–155.
4. Dijkstra, E. W. The structure of "THE"—multiprogramming system. *Comm. ACM 11*, 5 (May, 1968), 341–346.
5. Havender, J. W. Avoiding deadlock in multitasking systems. *IBM Syst. J. 2*, 1968, 74–84.
6. Shoshani, A., and Bernstein, A. J. Synchronization in a parallel accessed data base. Tech. Rept. No. 69-C-138, General Electric Res. and Develop. Center, Mar. 1969.

Received January, 1969; Revised May, 1969

Bond, et al.—*cont'd. from page 603*

interaction. The code occupies only 5000_{10} words. These figures are somewhat below current estimates of space and time for a graphical system. We regard the system as essentially finished and stable at the present time.

It is hoped that we have shown in this paper how we designed and implemented the system, given the graphic hardware and the existing computer environment of a general purpose batch-processing machine with remote access. We tried to describe clearly the internal design of the monitor, the use of stacks and the design of the transient version. The design of the user interface represents a definite attitude to the problems of interactive software.

Acknowledgment. The authors would like to thank R. A. Krutar for many helpful suggestions and also programming aid.

REFERENCES

1. Quatse, J. T. A visual display system suitable for time shared use. C.I.T. publication, Carnegie-Mellon U., Pittsburgh, Pa., 1966.
2. Internal documents on the Philco READ display system. Philco-Ford Corp.
3. *Computer Display Review.* Adams Assoc. Inc., Bedford, Mass., 1967.
4. Bond, A. H. A graphical interface in ALGOL. (In preparation)
5. Coleman, M. Text editing on a graphic display device as an example of simplicity in man-machine interaction. Rep., Carnegie-Mellon U., Pittsburgh, Pa., 1968.
6. Thomas, E. M. GRASP user manual. C.I.T. Int. doc., Carnegie-Mellon U., Pittsburgh, Pa., 1966.
7. ——. GRASP—A graphic service program. Proc. ACM 22nd Nat. Conf., 1967. Academic Press, New York, p. 395.
8. Bond, A. H. Scope user manual. C.I.T. doc., Carnegie-Mellon U., registered with Defense Documentation Center as AD 669 105, 1967.
9. Coles, L. S. GRANIS. Ph.D. Th., Carnegie-Mellon U., Pittsburgh, Pa., 1967.
10. Calvert, T. W. Ph.D. Th., 1968. Carnegie-Mellon U., Pittsburgh, Pa., 1968.
11. Garman, M. Ph.D. Th., Carnegie-Mellon U., Pittsburgh, Pa., 1969.
12. Cohen, G. Ph.D. Th., Carnegie-Mellon U., Pittsburgh, Pa., 1969.
13. Shoup, R. Internal document, Engineering staff, Computer Science Dept., Carnegie-Mellon U., Pittsburgh, Pa.

Received November, 1968; Revised June, 1969