Design of a time-sharing system allowing interactive graphics*

by G. B. ANDERSON, K. R. BERTRAN,
R. W. CONN, K. O. MALMQUIST,
R. E. MILLSTEIN, and S. TOKUBO
University of California
Livermore, California

INTRODUCTION

For the past several years various investigators at Lawrence Radiation Laboratory, Livermore have been conducting studies in computer graphics and man-machine interaction. These studies have included aids to numerical analysis,[1,2] photographic input and picture processing,[3,4] and computer-aided circuit design.[5] Each of these problems required—initially at least—a rather high degree of interaction between computer and investigator. To date these and similar experiments are conducted on one of two computers (medium and small in size) dedicated from time to time to this use. The demand for graphic study time on these computers has grown to the point where it has seemed advisable to implement a time-sharing system. The purpose of any time-sharing system is to give the processing and storage resources of a computing complex the appearance of being the unique property of each of its users. This paper will outline the software specifications for a system designed to facilitate the sharing of modest computing resources among a number of users at graphic terminals—each terminal consisting of a CRT display with keyboard, function buttons, and light pen input.

There are a number of considerations in the structure of any time-sharing system which are independent of that system's ultimate use. Many of these considerations have been explored by other investigators and a few will be informally reviewed to provide a tutorial basis for the solutions described. (Those concerning memory allocation, e.g., were carefully considered by Dennis.[6]) The derived system is similar to U.C. Berkeley's project Genie[7,8] in its gross structure with a file design closer to MIT's Multics,[9] or UC LRL, Livermore's Octopus.[10] Its uniqueness derives from the manner in which it schedules processes, handles shared files, permits virtual memory overlaying, and allocates storage for terminal displays.

*Work performed under the auspices of the U.S. Atomic Energy Commission.

Attaining a comprehensive overview of a time-sharing system is often clouded by simultaneously considering its several asynchronously interacting components. Such difficulty may be minimized by considering alone and in turn the gross function of each such component before noting their interrelationships.

RESOURCE UTILIZATION

First, the heirarchy of utilization of the computing facility may be given—in descending scope—as the **user,** the **job,** the **computation** and the **process.** (Acknowledging the ambiguity in software nomenclature, these and subsequent names have been chosen because they are, (1) in common use and, (2) suggestive of their function.) The **user** is any individual authorized to avail himself of the utility. A given system may permit him to submit one or more jobs, where **job** is some unit of work—retaining the same ambiguous connotation it is afforded in batch processing. The system described does not permit a user to start a second job while continuing with a previous one. This restriction nullifies the structural distinction between user and job. We will make use of both words in their usual sense.

A **computation** may be thought of as the smallest entity using the computing resources without reference to or dependence upon any further information. Because a computation may be unloaded (for subsequent reload) by a user who may then start another, a computation is not logically equivalent to a job. Each computation consists in part of one or more processes.

A **process** has been defined[7] as the logical environment for the execution of a program, where program has its usual meaning of procedure and data. A process is completely specified by a collection of information called its **state vector.** Maintaining part of this vector allows for the process to be run or to be retained in a potentially dormant state. A running process may be interrupted by signals resulting from a variety of hardware conditions. These signals will be called **interrupts.** A running process may call upon the system to perform one of a number of tasks. Such a transfer of control will be called a **system call** or simply a **call.**

FILES

The next major component to be considered is the one responsible for the general form in which information is retained. All information will be stored in a collection of data structures called files. The file system provides a single means for naming and accessing any information—procedure or data—regardless of its source, destination, or medium of storage. The file system assures the protection of restricted information.

Files are of two types: one is made up of formatless information and is called simply a **file,** the other, called a **directory,** is the vehicle for locating any other file (including other directories).

There is a special directory, the **mailbox,** through which files may be passed from one user to another or between any user and the system. This second facility enables the user to give files to a **privileged** (system's) **user** entrusted with the handling of all input and output—exclusive of the graphic stations and secondary storage.

A file is created by a user-originated system call and is made up of a heading plus all information blocks. The heading includes a word count of the information, pointers specifying the location of each information block, the file's security level, type of access (e.g., read only), and reference count.

A directory is also created by a user-originated call and consists of a heading together with some number of **entries.** The directory heading, like the file heading, has a security level, type of access and reference count. Each **entry** has—in addition to repeated security information—allowed access, life expectancy, a time of creation, a name, and a pointer to the heading of the named file. Note that a file is named only by a directory entry; that a single file or directory may have any number of names pointing to it from other directories; and that an entry may point to the directory in which it resides. A single directory may have two or more entries with identical file names. In searching such a directory, the last entered name will be the one found. The last entered will also be first to be removed by a user-originated deletion of that name.

The reference count of any file is the number of entry names pointing to that file. This count may be decremented in one of two ways: first, if the file entry is deleted by a user-originated call; second, if the life expectancy of the file entry is exceeded. When the count drops to zero, that file's storage is released and the file is either lost or alternatively—if the life expectancy of its entries has been exceeded (and another field has been maintained in the entry), moved to lower-level storage. If the released file is a directory, the count in each file pointed to by an entry in that directory must be decremented.

Each user will be given a pointer to a directory of his own which cannot be destroyed, the **root directory,** and two pointers per process to **working directors**—which he may assign through the root directory. These working directories, together with a set of primitive file calls, permit the development of routines capable of accessing files through any desired path.

Life expectancy is normally checked while searching through a particular directory. The check is discontinued when the specified name has been found. Files will therefore exist beyond their requested life span. Directories whose count would be zero if it were not for self-references may also exist. A periodic collection will be made to re-acquire this space. Beginning with the root directory of each user—including the system itself—the entire information space may be traced.

STORAGE ALLOCATION

Another major component of a time-sharing system relates to the allocation of storage. Because high speed processor memory is expensive, present time-sharing systems normally employ a relatively fast auxiliary storage. Part of the system must be devoted to moving instructions and data between these two stores. The attempt is to affect this exchange in such a way that the auxiliary device appears as an extension of main memory.

In the present system main memory consists of two interlaced banks of sixteen thousand words each. (Word size is 32 bits.) Auxiliary storage is a fixed head disc of about three-quarter million words which we will call the **swapping disc.** The system employs a single processor so that there will be only one process—system or user—running at any time. To achieve the time-sharing goal in a system which is highly interactive (i.e., has several users making simultaneous demands on the resources), it is necessary that processes belonging to different users be run alternately and at a rapid rate of exchange. Since main memory is inadequate for the storage of all required processes, it is necessary that they be swapped to and from the auxiliary storage.

It would be a great waste of both time and memory to move each process in and out as it is needed, but if this is not done a conflict in addressing may result between two or more processes—even if storage space is sufficient. As has been pointed out, instruction addresses perform the dual function of both naming information and of locating specific places in physical storage. In the first sense addresses have been called **name space,** in the second, **memory space.** If several processes use the same name space (addresses) and are to be kept in memory at the same time, some mechanism must be developed to preserve names while changing physical locations. Both relocation registers and paging schemes have been used. The latter seems to have some advantage in that contiguous names need not be mapped into contiguous memory locations.

The graphics system employs a simple paging mechanism. Addresses are seventeen-bits long. Each page is 512 or 2^9 words so that name-space consists of 2^8 pages. Each address is the concatenation of a page number and a line (word) number. A hardware memory map of 256 eight-bit registers maps the pages of name-space into the sixty-four-page memory-space. There is associated with each page a two-bit access code permitting or inhibiting all access (reading, writing, and execution)—or allowing only reading—or only reading and execution.

The entire swapping disc has been divided logically into page-sized blocks. The location of each such page is called the disc address. File headings, directories, and the previously discussed file-information blocks are each one page in length. The software (structure) extrusted with handling the physical disc in a way that insures fastest page transfers between core and disc is called the **disc manager.**

A disc address is assigned to a particular file page in two ways. One, the assignment may be random, determined only by the rotating disc's next available page at the time such space is requested. Two, the new addresses may be acquired as a function of some given disc address in an attempt to minimize latency when the sequential transfer of these related pages is desired.

A map of disc space, the **disc map,** is maintained at all times in high-speed memory. The map indicates whether the hardware comprising a particular page is good, if that page has been used, and, if so, as a file page, heading, or directory. In addition to its obvious use in allocating space, the map will provide a redundancy check in any attempted recovery of information space.

Two other software structures involving space allocation must now be mentioned. The first of these, an **association table**—keyed to the disc addresses and with one entry for each page of high-speed memory—provides the system with relatively fast knowledge of what disc pages are in memory and what memory page each occupies.

The second, called the **real memory table** (RMT) is responsible for the actual assignment of core (real) memory. Because real-memory allocation is a function of the scheduling procedure, we will defer further discussion of these structures until process management is considered.

GROSS STRUCTURE

At this point it is necessary to name or describe in turn a few of the system structures as well as those requisite to the accomodation of an individual user.

The name given to the collective resident structures is the **executive.** It includes the interrupt and call handlers, the previously discussed **allocator** (disc manager, disc map, association table, and real memory table), and the process scheduler.

Most other systems functions are performed in answer to particular calls and are accomplished as system's processes. The procedural aspect of the process is typically a re-entrant routine. Except that they may be run in the master mode, system processes will be handled much like user processes. (Master mode execution permits the use of privileged instructions—e.g., those used to set access bits.)

The creation of new processes has been called **forking.** Any process—system or user—may **create** (fork off) a process. A special system-created process, the **proxy,** provides the interface between executive and user. The first user-created process must be forked from his proxy. A process may be **stubborn** or normal. A stubborn process may be destroyed (quit) only by a proxy or a self-originated call. The user's normal processes may quit one another.

Forking a process triggers the creation of a table, the **program active table** (PACT). The PACT either contains or points to essential elements in the state vector of that process. Included in the PACT is a map of the process name-space **(virtual memory).**

The 256-page virtual memory is mapped in a table called the **process memory table** (PMT). Table entries represent a single page of space—with each entry consisting of an access code, a disc address and a **keyword** number.

Associated with the process' PMT are fifteen keywords. As indicated, each PMT entry points to one of these words. The keyword, in turn, points to the heading of the file in which the disc page, referred to by that entry, resides. This structure establishes the link between the file system and storage allocation. The keywords additionally contain a copy of the file's access and security information.

The assignment of a file to a keyword is made by issuing a call to **open** that file. If the keyword requested had already been assigned, the previously-opened file will be **closed.** To prevent opened files from disappearing because of deletions by other processes, opening will increment and closing decrement a file's reference count. Closing a file necessitates scanning the PMT for all references to the file's keyword to guarantee that no part of a file whose disc space could be reallocated will remain in virtual memory.

Maintaining pointers to file headings enables page-by-page assignments to or exchanges within the virtual memory. Once a file has been opened, a call may be made demanding a specific page in the opened file be assigned to a specified place in name space. This technique, called **coupling,** provides each process with the facility for dynamically loading and reassigning space. Initial program loading also will be accomplished by opening a file and coupling the required pages.

The structure just described relating name-and file-space eliminates all concern with whether memory is shared or not shared. As we have seen, virtual memory is kept as a series of disc addresses. One user can share a file belonging to another if he has been granted permission by receiving through the mailbox an entry pointing to that file. With this entry in a directory he can reference, he need only open it and couple pages to the memory of one of his processes. When a page is needed in real (high speed) memory, the association table is the instrument for indicating whether it is there or must be fetched.

For purposes of inter-process communication, PACTs belonging to a user at any one time are chained together. This collection of processes is the user's computation and the chained PACTs constitute the state vector for his in-process work. The PACTs are included in the space of a special file called the **computation heading.** This file may be retained by a user-originated request to **unload.** The user may **reload** an unloaded computation provided none of his others are active.

One last structure, the **JOB table,** remains to be mentioned. The system maintains a list of eligible users, together with accounting and security information and a pointer to each user's root directory. When a user properly logs in, he is provided with a job table consisting of accounting information, the identity of his keyboard and display devices, the pointer to his root directory and a pointer to his first process, proxy.

PROCESS MANAGEMENT

It should now be clear that information belonging to any user is identified via the file system and is ordered for execution as a process. We have seen that pages constituting processes or parts of processes must be swapped between secondary storage and high-speed memory. We must now look at the structure deciding when such swapping should occur and some of the rationale underlying this structure.

Timing interrupts emanating from the physical disc and used to activate the disc manager also serve as the system's basic unit of time. An integral multiple of this unit, the **shot,** measures the minimum run time of an uninterrupted process. Two other timing units, the **short** and **long quantum,** are multiples of the shot.

When a process is forked its PACT is linked to the bottom of the short quantum queue. Suppose all other active processes happen also to be on this queue—and no calls cause them to be moved elsewhere—then, the first process on the queue will run to the end of one shot at which time it will move to the bottom of the queue—or, if its short quantum has run out, to the top of the long quantum queue.

There are several queues which operate in a similar fashion. Being on such a queue implies that a process is ready to be run. These queues are, in order of priority, the **page fault queue,** the **keyboard queue,** and the short quantum, input/output, and long quantum queues.

In all but the case cited (i.e., moving from the short to the long quantum), processes are chained at the end of a queue and must move to the top to be run. (This exception was made for the case in which a user needs some epsilon greater than one short quantum of time.) For example, a process stopped while awaiting keyboard input will be chained to the keyboard queue when—and only when—that input becomes available. The **process scheduler** sees that the process next executed is the one at the top of the queue with the highest priority. The mechanism whereby this is accomplished necessitates a look at the way a process acquires space in real memory.

When a user's proxy forks the first process, the file pages required in the virtual space of the new process are coupled to the proxy's PMT. This PMT is duplicated for the new process, and the pages relating to proxy are uncoupled. The PMT for all subsequent forks will be some sub-set of the PMT of the creating process in any order specified.

Associated with each PMT are the set of real access codes: the codes actually used by the hardware when that process is activated. At the beginning of each shot all of these codes are set to deny access. Reference to a name in virtual space, then, causes the referring instruction to be trapped. The PMT entry for the page which contains the trapped instruction is consulted for the proper access. If the PMT entry indicates that the trapped instruction is legitimate (i.e., does not require access greater than the entry specifies), the association table is consulted to see if the referenced page is in memory. If it is not, the PMT entry is linked to the **pre-load chain** and the PACT is entered at the bottom of the page fault queue. If the page is in memory, the hardware memory map is adjusted—if necessary—and the real access code is set only as high as required by the referring instruction. The real access codes serve a secondary function, then, in letting the system know whether or not a page has been written into. If a page has not been written into, it need not be swapped back to disc.

The pre-load chain links all virtual pages, up to some limit N, which were used by that process during its previous shot. A process just forked will have the page containing its starting address in its pre-load chain. Note that the chain is dynamic—those pages not used during a shot that has run to completion are eliminated, and a page newly referenced is added.

It is the scheduler's responsibility to guarantee that all pages on the pre-load chain of the **next process** are in memory, and that a real page exists for assignment to the **current process**—should it be put out on the page fault queue. The scheduler will pre-load as many processes as real memory space allows. The order of loading is that given by the queue priority, with all processes on a single queue (taken from the top down) preceding any process from a queue of lower priority. Pages are acquired for pre-loading in an order exactly the reverse of the ready queue priority.

The table relating a process' pre-load space with real memory is the previously-mentioned real memory table. All pages—not reserved for system's use only—are listed in the RMT. RMT entries representing the pre-load chain for a single process are linked together and represent the pages of that process actually present in real memory. As implied earlier, the current and next process will always have their pre-load pages linked in the RMT, and complete links will exist for subsequent processes to the extent memory allows.

It is now possible to visualize the interactive nature of the system. Processes are scheduled according to their places in the ready queues. The queues themselves indicate the reason for previously stopping a running process and were chosen in the hope of assuring maximum interaction. When a process page faults, for example, it is a trivial procedure to acquire the additional space, so that releasing its other pages for use by different processes seems obviously unwise. Each of the queues can be rationalized in a similar way. The current process will complete its shot unless prevented by an interrupt or certain systems calls. The nature of the interruption will determine whether control can be returned to the same process or if it must be stopped pending other action. In the latter case, once the interrupt procedure has been attended, the next scheduled process will be run.

Only one other factor can alter the status of a process, and that is the procedure for inter-process activation and communication. This mechanism allows a process to be made inactive (put to **sleep**) or be in one of two active states, **awake** or **hyper-awake.** An entry in the PACT indicates into which of these three states the process has been put. A process will retain its place on the ready queues but will never be selected for execution while asleep.

A process that is awake may be put to sleep by any other process in the same computation. In the same way, a process that is hyper-awake will be simply awake after having been ordered to sleep. While the user may only sleep his own processes, he may **wake** the process of any user who is then logged into the system. Again, if the wakened process were asleep, it will change one state to awake, and if already awake, to hyper-awake.

Effective use of this control necessitates the establishment of data-space accessible to two or more processes. Before going to sleep a process might scan this space for messages from some other process. In the event a message was received during such a scan, the receiving process could be informed by being put into a hyper-awake state. In the other direction, a process with a message to transmit would send its message and then awaken the recipient process.

INPUT/OUTPUT

The system has a complement of conventional input/output devices in addition to its disc and graphic display terminals. This equipment consists of two magnetic tape drives, a card reader, card punch, line printer and console typewriter. All of these devices, called collectively the **slow-I/O,** will be driven through a multiplexing input/output processor.

With minor variations in detail, the slow-I/O will be handled as suggested in the section on files. A user wishing to input or output information will create a file, **give** the mailbox an entry to that file and awaken the **slow-I/O controller.** General use of the mailbox requires that an entry be addressed to a specific user, who must issue a call to **get** that entry. The user must identify himself and ask for the file by name. In the case of slow-I/O, however, the name specifies the requested device and the I/O controller is privileged to search the mailbox. Once awakened the controller conducts this search, awakens the appropriate device handler, and puts itself back to sleep. The device handler, in turn, issues commands necessary to move the file and also goes to sleep. Interrupts from the devices are received by the controller which initiates the required action. If, for example, an interrupt signals that input has been successfully received from some device, the appropriate process on the I/O queue will be awakened. The console typewriter is used exclusively for operator control and is the vehicle for requesting manual intervention.

Two display generators with fast character and vector capabilities (under 5 and 50 microseconds, respectively) will drive the user terminals (CRT, keyboard, function buttons, light pen). One will interface with high speed memory through a special input/output processor. The other—for remote terminals—interfaces with a satellite computer, which, via communications equipment, connects with the multiplexing input/output processor. Each display generator is capable of driving up to four terminals.

Consistent with the system's general structure, the displays will be serviced by a set of primitive calls.[11] These primitives facilitate the construction of individual display words, the naming of a group of such words —together with its coordinate location—as a picture part, and the entering or removal of a named part in the display list. Other primitives are responsible for following or interrogating the light pen.

The display lists themselves will be maintained in the high-speed memory of either the main or satellite computer. Pages for display will be stolen from high-speed memory by the allocator.

CONCLUSION

Implementation of the system is under way. Its completion should supply answers to questions about time-sharing as well as those involving graphics.

Regarding time-sharing, several parameters have been designed into the system whose variation should supply significant statistics. One sets the limit on the number of pages in a pre-load chain and may be varied from one page (demand paging) to a little less than half of real memory. The others control the system's time units, the length of a shot, a short or a long quantum. Finally, we hope to establish that a system can be foolproof from a security standpoint. That is, the privacy of a user's files—whether they be in use or in storage—cannot be violated unless that user so desires, and then only if the desired recipient is himself clear for that level of information.

Since the system is being put together to answer basic questions in graphics and man-machine interaction, we will not pinpoint the areas we hope to investigate. Suffice it to say it will be used for a wide variety of stüdies—including those cited in the introduction—and ranging from drafting to pattern recognition. Initially, the system will make available a typical assembler and FORTRAN compiler. Eventually we hope to provide a macro-compiler capable of allowing the user to tailor a source language to his own liking.[12] In that way we hope to move in the direction of a long overdue graphic-processing language.

ACKNOWLEDGMENTS

The authors would like to thank George Michael for championing a graphics facility, Robert Goldstein for researching the possibilities, Peter Deutsch for suggesting the association table, Butler Lampson for his stimulating criticism and his many design recommendations, and Sidney Fernbach for his continuous support and assistance.

REFERENCES

1 R W CONN R E VON HOLDT
An online display for the study of approximating functions
J ACM 12 326 1965

2 K R BERTRAN
Paramet
Decus Proceedings Fall 1965

3 L GALES G MICHAEL
Graphic input devices
Internal Report February 1967

4 R W CONN
Digitized Photographs for Illustrated Computer Output
Proc AFIPS Conf vol 30 103 1967

5 W G MAGNUSON JR F F KUO W J WALSH
On-line graphical circuit design
UCRL-70796 Univ California Livermore Lawrence Radiation Laboratory 1967

6 J B DENNIS
Segmentation and the design of multiprogrammed computer systems
J ACM 12 589 1965

7 B W LAMPSON W W LICHTENBERGER M W PIRTLE
A user machine in a time-sharing system
Proc IEEE vol 54 1766 1966

8 B W LAMPSON
Scheduling and protection in interactive multi-processor systems
ARPA Doc 40 10 150 University of California Berkeley 1967

9 R C DALEY P G NEUMANN
A general-purpose file system for secondary storage
Proc AFIPS Conf Vol 27 213 1965

10 J G FLETCHER
PDP-6 system at LRL Livermore
Decus Proceedings Fall 1966

11 K BERTRAN A CECIL R CRALLE G MICHAEL
A new set of display subroutines for the IBM 7094 — DD80-A system
Internal Report March 1967

12 G B ANDERSON
A generalized macro compiler for an easily modifiable extension of Lisp 1.5
UCRL-70543 Univ California Livermore Lawrence Radiation Laboratory 1967