[illegible] 19 February [illegible]

B C C 5 0 0 S Y S T E M

BERKELEY COMPUTER CORPORATION

2131 Fourth Street

Berkeley, California 9[illegible]10

(415) [illegible]-2600

384 K bytes of main storage (1 μ sec / 6 bytes), 8-way interleaving

"[illegible] memory device" ([illegible] with four ports: CPU1
[illegible] [illegible] sec access
CPU2
auxilliary memory traffic
other [illegible])

[illegible] drums (6 megabytes each)

1 to [illegible] disks ([illegible] megabytes each) 50 m sec rotation, 40-[illegible] m sec read pos, [illegible] bytes/sec xfer

"Worst case performance": 500 users all executing at once.
Each gets 1000 BASIC statements executed / 5 seconds

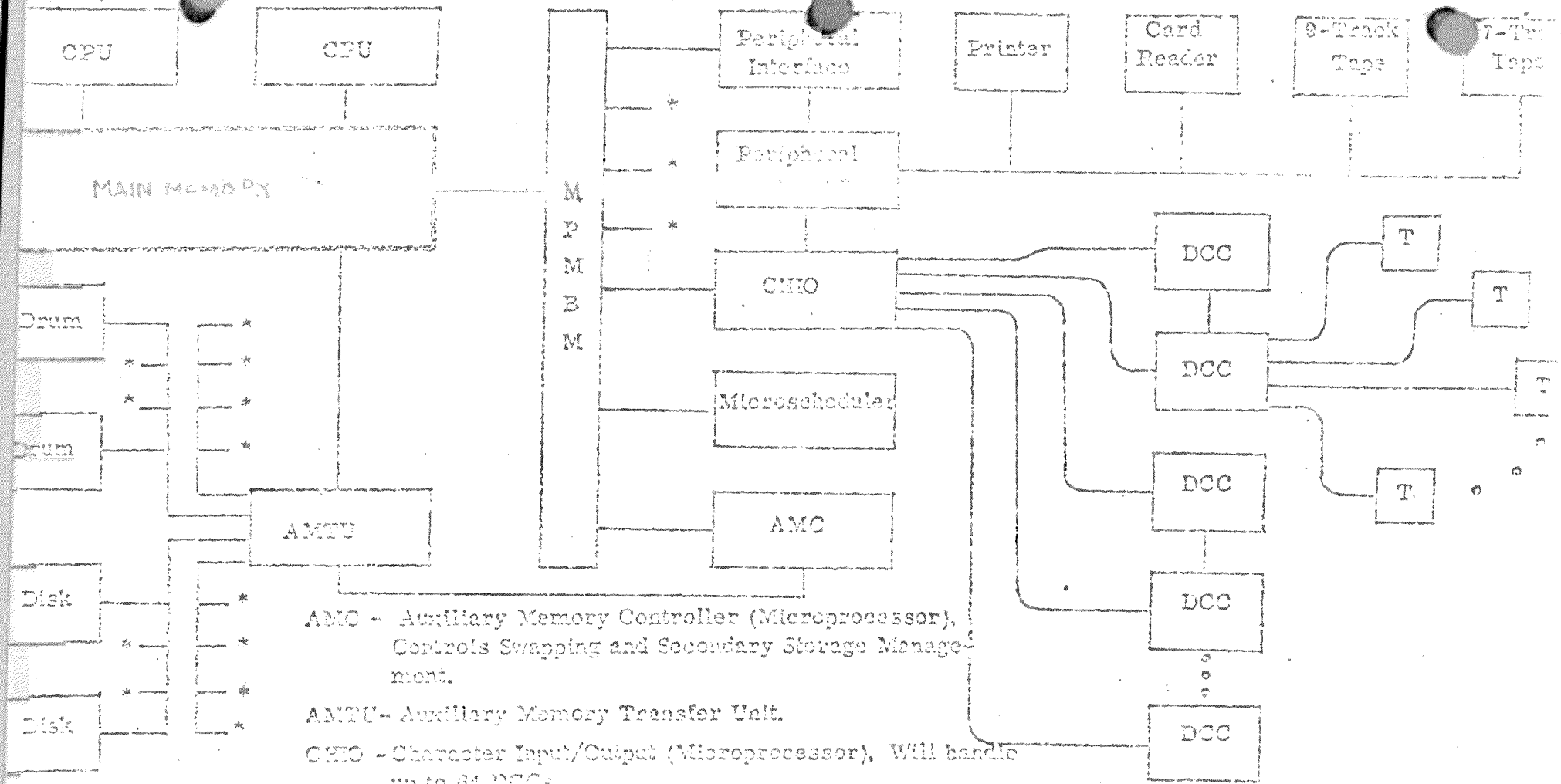
B C C 5 0 0 S Y S T E M

The system diagram shows the present BCC-500 System. This system will accommodate a very large number of terminals (over 2000); the number of users who can simultaneously make use of the system is, of course, dependent upon their computing requirements. For users of BASIC and similar conversational languages who are working on problems of complexity and size similar to those now being accommodated by remote access systems, this number is approximately 500. The system has thus been designated the BCC-500.

Features which permit this large number of simultaneously active users to compute without experiencing poor or degraded service include:

- Critical parts of the operating system are built into the hardware;
- Swapping in and out of central memory proceeds simultaneously to, and does not interfere with, the program execution by the central processing units;
- The communications system is designed as an integral part of the total system;
- The overall system design is geared to operate well near saturation level;
- The central processing unit overhead is negligible for switching processes;
- There are two central processing units;
- High transfer rate drums provide fast swapping and quick access to files.

These and other features will be explained more fully. The following discussion presents salient features of the BCC-500 System. Such a discussion cannot be detailed without also being lengthy. Hopefully, a happy medium has been reached sufficient to generate an overall understanding. This understanding can then act as the context for specific details.



AMC - Auxiliary Memory Controller (Microprocessor). Controls Swapping and Secondary Storage Management.

AMTU - Auxiliary Memory Transfer Unit.

CHIO - Character Input/Output (Microprocessor). Will handle up to 64 DCCs.

DCC - Data Communications Computer (Microprocessor). Will handle up to 224 low-speed or 56 high-speed full-duplex terminals.

MPMBM - Microprocessor Memory Bus Multiplexer.

MICRO-SCHEDULER - (Microprocessor). CPU Scheduling

T - Terminal: Teletypes 33, 35, 37; IBM 2741 (Both Models); 300 characters per second line printer; keyboard display, etc.

CENTRAL PROCESSOR UNITS

The Model 500 System uses two CPUs. (Central Processor Units) incorporating features which allow user programs to run more rapidly and more reliably. Many routine errors in programming are detected automatically without any increase in running time for correct programs.

Problem Oriented Programming

Although many computers are programmed entirely in languages such as BASIC, FORTRAN, and COBOL, few are designed to execute programs written in these languages efficiently. The BCC-500 System was designed with languages in mind, so that a simple, fast compiler can produce good object programs. Features of the CPU which contribute to this goal are:

- Array descriptors which permit access to a matrix without any program multiplications and elaborate loop optimization;
- Relocatable code to facilitate incremental compiling which minimizes user's cost for program changes;
- Table descriptors which permit any instruction to address a packed field in a table as efficiently as a full-word quantity;
- String-handling operations to speed up compiling and data processing programs;
- Use of central memory locations as indexing quantities so that registers do not have to be allocated and loaded whenever an array is referenced.

Automatic Error Checking

It is important to the user to detect erroneous use of a language (e.g. subscripts too large for an array) and to report all errors in terms of the source language (e.g. to give the source statement in error, not the memory location, which has no meaning to the user).

This is not done in many FORTRANs and is done in BASIC only by costly interpretation. Model 500 CPU features to do this kind of error checking are:

- Programs are protected from improper modifications by hardware. The user can be sure that his program will not be improperly altered, and that he will be informed of any attempt to destroy it as soon as it occurs.
- Array subscripts are checked automatically. This eliminates one of the most common sources of incomprehensible program errors: destroying data by storing into an unrelated array with a subscript which is too large.
- Subroutine calls are checked to ensure that arguments have the proper type, so that, for example, a subroutine cannot fail because it is given an integer when it expected a real.
- A special "undefined" floating point number is provided, which detects references to data which has never been defined.

Additional Features

To provide arithmetic accuracy and flexibility in handling overflow and underflow, the CPUs provide:

- Floating point with 11-bit binary exponent (number up to 10^{300}) and 36-bit fraction (11 decimal digits) for normal use.

- User-controlled traps on overflow and underflow, which are fast and which permit the correct result to be easily recovered.

- An underflow option which allows the results to drift toward zero without causing a trap.

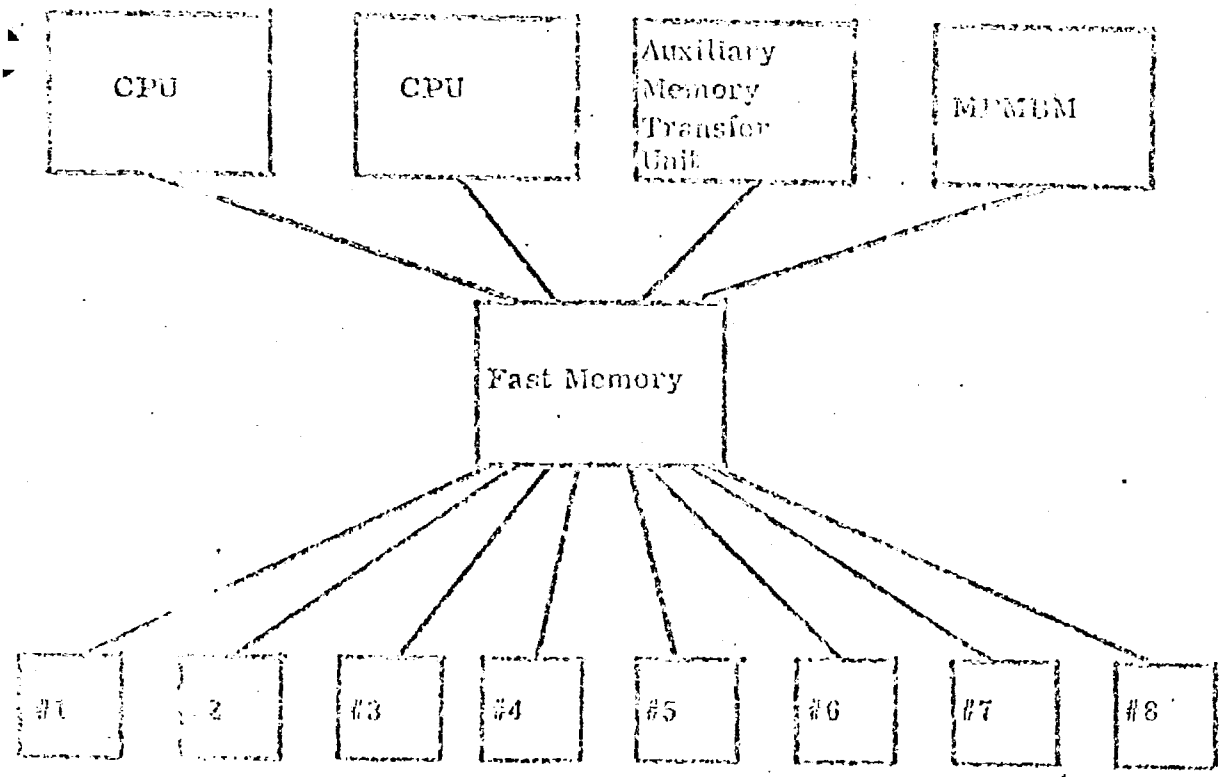
MEMORY CONFIGURATION

The B6C-500 System has a paged organization imposed on all of its memory. Basically, the system deals with 2048-word pages, 3-byte words, and 8-bit bytes. Every page has associated with it a 48-bit unique name which is generated by the hardware when the page is created and stays with it until its eventual death.

Central Memory

The central memory consists of a fast memory device and eight interleaved core modules. Each module contains 8 thousand, 52-bit words of 1-microsecond core (48 bits of information and 4 parity bits). This results in a total of 384 thousand bytes of central core.

The fast memory device interfaces the eight core modules with the rest of the system through four ports. Two of these ports are dedicated to the two CPUs; one port handles all auxiliary memory traffic; and the remaining port serves the rest of the system through the MPMUM (microprocessor memory bus multiplexer). The fast memory device contains fast, associative "stickly registers" which interface the four ports to the core modules. The basic function of the fast memory is to allow simultaneous access by the four ports to the contents of the core and to provide the access at better than core speed.



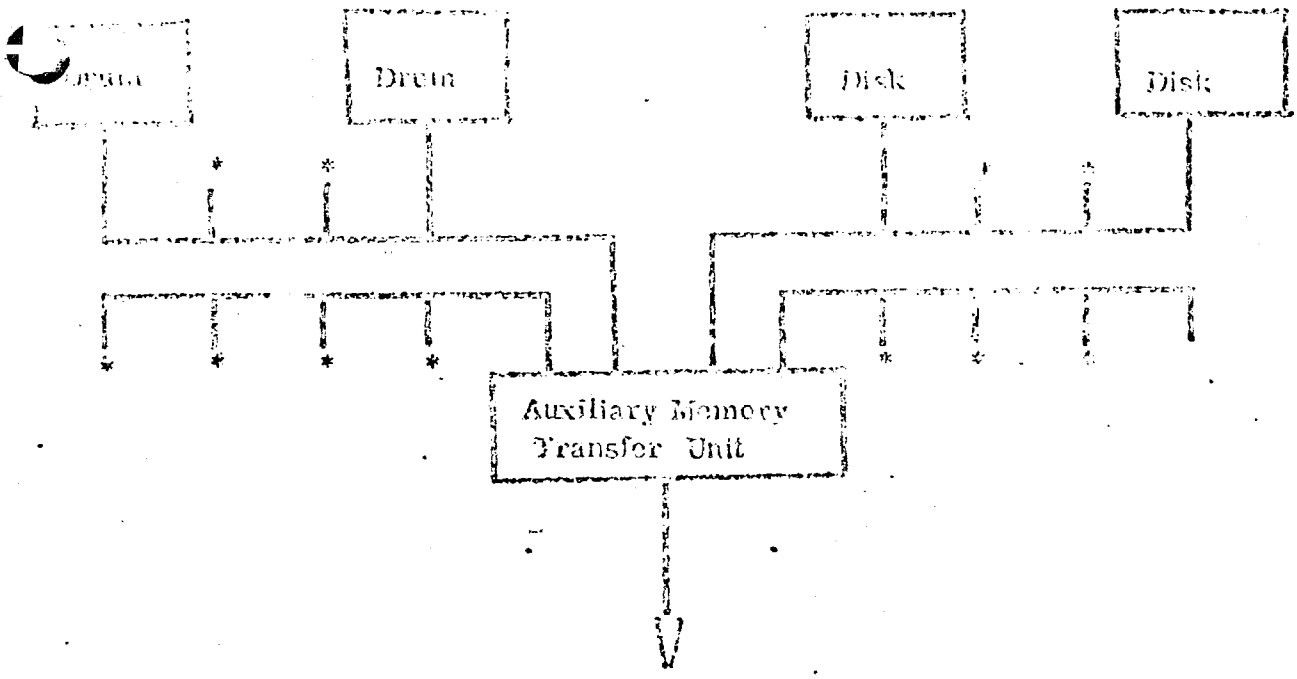
384 thousand bytes, 1- μsecond core / 6 bytes

If a CPU is making a core access and the drum wishes to transfer a word into the same module, a fast register will accept the drum word and put it in core later when the CPU access is finished. The CPU access will also pass through a register, which will retain the accessed value after passing a copy along to the CPU. If that word is accessed again in the near future, a core access will not be necessary and the register will deliver its value in 200 nanoseconds. Thus, tight program loops and frequently accessed data migrate naturally into 200-nanosecond register.

Simultaneous access is provided by a combination of the register action and core address interleaving. This allows the CPUs to execute programs while the auxiliary memory system is swapping other programs in and out of core without interference. The fast memory will accept a request on each port every 100 nanoseconds. This is a port request bandwidth of 10 Mhz. The core modules, each with an access of two words per microsecond, provide a 16-Mhz transfer bandwidth. The difference in bandwidths is tolerable since a port request may be satisfied by a register without requiring a core access. Also, if a port request is rejected, it can be made again 100 nanoseconds later, thus enhancing the probability for a reasonably early acceptance of a request. The memory is capable of functioning at half capacity. If a core module or portion of the fast memory device fails, the system will detect and locate the problem and reconfigure itself to omit the impaired portion.

Secondary Memory

The initial model 500 configuration has two 6 million byte drums and one dual positioner disk with a total of 378 million bytes. The auxiliary memory system is designed, however, to handle up to eight drums and eight disks.



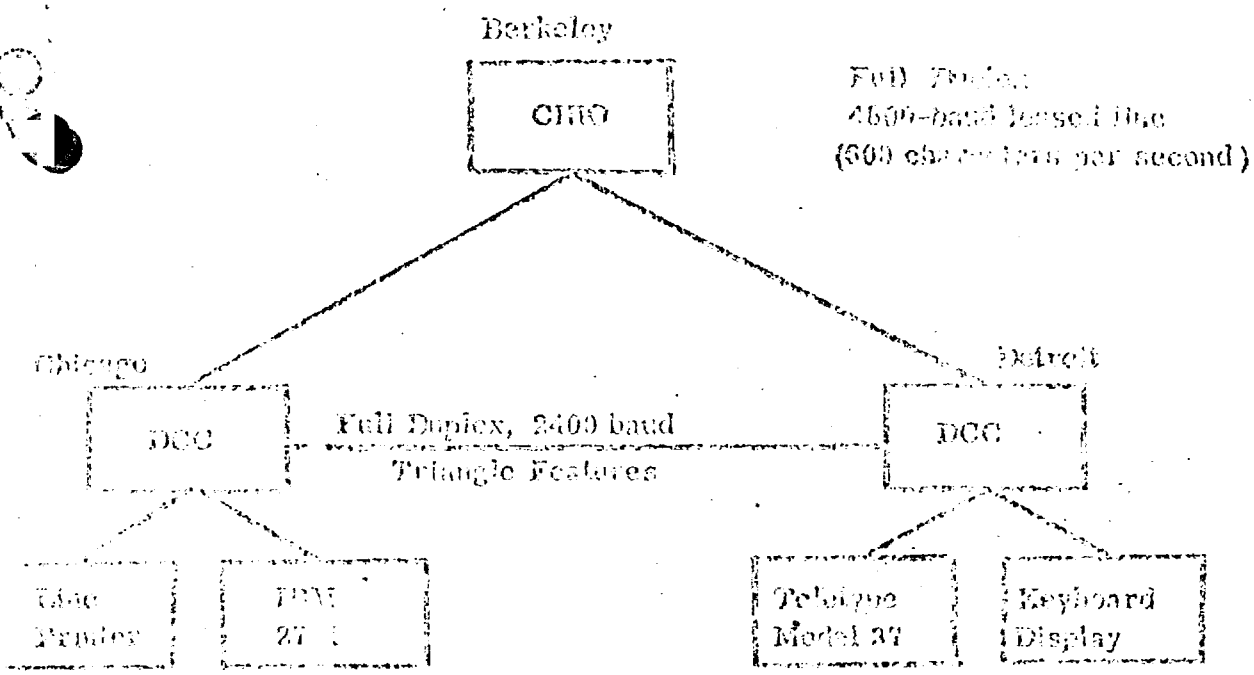
A system with eight disk files would contain three billion bytes of on-line file storage. All eight disk files can simultaneously position their heads, and any two of them can transfer simultaneously. The transfer rate for each file is six hundred thousand bytes per second. Each disk file has a 50-millisecond revolution and a 40 to 200-millisecond head positioning time.

Each of the drums initially on the system has a capacity of six million bytes. These can be replaced in the future by fifteen million byte drums currently in check-out by the manufacturer. The auxiliary memory system will accommodate up to eight drums.

This high transfer rate allows the system to efficiently overlap time-shared programs in and out of core without any CPU time being lost. Also, since drum capacity is large, directories and portions of large data bases can be maintained on the drums and accessed extremely fast. These characteristics make the BCC-500 system particularly suited to highly interactive work, like commercial time-sharing and management information systems.

COMMUNICATIONS

The heart of the communications system is the CIIO (Character Input/Output unit). On one side it is connected to the central memory via the MPMBM, and on the other to as many as 64 DCCs (data communications computers). There is also a connection to a peripheral computer used for bulk input/output. A DCC is connected to the CIIO by a full-duplex private leased line which may operate at 2400-, 4800-, or 9600-baud. Modem reliability considerations prompted the use of 4800-baud lines initially. Each DCC is also connected to another DCC by, initially, a 2400-baud private leased line. This triangle feature offers an alternate route between the DCC and CIIO should one of the lines become impaired. A DCC is a 16-bit general purpose computer with four thousand words of 16-bit memory. Its function is to multiplex up to 224 full-duplex bit scanned lines onto the private leased line and ensure error-free transmission. A bit scanned line may function at any rate up to 300 baud. A single bit scanned line is intended to service a low-speed device like a Model 35 teletypewriter. These multiplexing algorithms are interpreted as the DCC is a microprocessor. Two low-speed lines may be traded for one medium-speed line. A medium-speed line could service faster terminals like line printers or keyboard displays. For keyboard displays a full-duplex medium speed line can function at 1200 baud one way and 150 baud the other. The transmission rates are determined by the capacity of the private leased line.



A 4800-baud line transmits 600 characters per second. To make the transmission highly reliable, in addition to using high quality hardware and private leased lines, the character transmission is encoded for error detection. This encoding reduces the amount of information transferred to a figure which depends upon the type of device, direction of transmission, and bandwidth of the leased lines.

On a 2400-baud line, transmission rates are:

- DCC ⟶ CHIO between 440 and 880 characters/sec.
- CHIO ⟶ DCC about 880 characters/sec.

The CHIO exhibits perfect buffering so that it will never lose characters if its buffers become full. The DCC will refuse to echo any character which it can't accept from a terminal if its buffers become full and the terminal is in full-duplex mode. If the device is in half-duplex mode, the DCC may either lock the keyboard or increase the size of the buffer.

Higher speed communications and very specialized devices can be accommodated by using the additional MPMBM ports.

One concept which helps the ACC 360 System schedule and execute many more jobs than present day systems is swapping overlap. Overlap is a term which simply means that a CPU can be executing a program in the central memory while the next program is being swapped in and the previous program is being swapped out. This swapping in and out of central memory occurs while the CPU is making central memory demands, yet the two do not interfere with each other. The fast memory device and interleaved core modules provide this simultaneous non-interfering access.

Simulation has shown that a good scheduling algorithm in a system with overlap, like ours, should assign a time slice quantum to a job equal to the amount of time it takes to swap the job in and out of core. This is reasonable since it simply says that the amount of total swap time should equal the amount of compute time. Since both occur simultaneously and independently, each is used 100 percent of the time. Thus, while one program is computing, a second is being swapped out and a third is being swapped in.

The next question is how do we determine which portion of a user's program should be swapped in. Extensive research in this area has produced the "core working set" concept. A core working set is that portion of a user's program which is needed for the current time slice. The ACC 360 System will allow the sophisticated user to specify and change his core working set. The unsophisticated user will have his core working

set determined dynamically by the system. This default scheduling will attempt to execute a user's program until a reference to a page is made which is not in core. At this point the user's time slice will be terminated (probably prematurely) and the next user will be given control. The next time the first user is given a time slice, his core working set will include the newly required page. There is an upper bound on the number of pages any given user can have in his core working set. If his core working set is full and he requests an additional page, the new page will replace the page which has gone the longest time without a reference. Thus, the system exhibits "demand paging" characteristics only until the user's core working set is determined. Thereafter, a time slice can run to completion, uninterrupted by swapping needs. In contrast, a sophisticated user may exercise direct control over the contents of his core working set and thereby eliminate most page faults.

The following examples demonstrate the throughput possible with this design:

1 page = 1045 words = 6344 bytes

CORE WORKING SET	8 pages	3 pages
IN/OUT SWAPPING TRAFFIC	32 pages	6 pages
SWAP TIME PER PAGE	x 1.2 msec	x 1.2 msec
TOTAL SWAP TIME	34.4 msec	7.2 msec
NUMBER OF "JOBS" PER SECOND	76	140

However, an eight-page core working set will be composed of a number of read only program pages and therefore, since none of them require change, only require a partial swap out. Thus, an eight-page core working set requires eight pages swapped in and perhaps four pages swapped out.

These examples indicate the number of jobs per second which could be accommodated if each job were executed for a time slice as long as its total swap time. An indication of how much computing can be done for many users is seen in the next example.

- A 5-second guaranteed response,
- On a two-CPU system,
- Gives 20 msec to each of 500 users every 5 seconds.

A CPU sees a 500-microsecond interruption while switching from one user to another. This is negligible compared to a 20-msec slice.

- With 3 microseconds per instruction per CPU as average instruction time,
- And if there are six instructions per average BASIC statement,
- Then under worst case conditions (every user computing, none contemplating their programs), each of 500 users could execute about 3000 BASIC statements every 5 seconds.

[illegible], it is very seldom that all users are [illegible] computing, especially if there are 500 of them. The time [illegible] used by some users could be given to others. The swapping algorithms are located in the Auxiliary Memory Controller and are coded in microcode. Much of the scheduling is done by the Microscheduler and is also microcoded. Thus, scheduling and swapping are imbedded in the hardware and do not require the attention of a CPU. The 500-microsecond interruption seen by a CPU while switching from one user to another is taken up in map switching, state saving, and that portion of the scheduling algorithm in the software.

SYSTEM RESOURCE CONTROL

Each active user in the system has certain resources guaranteed to him called resource allocations. These include:

- CPU share
- Drum pages for swapping
- Number of disk accesses every 30 seconds
- Number of terminal lines
- Number of processes

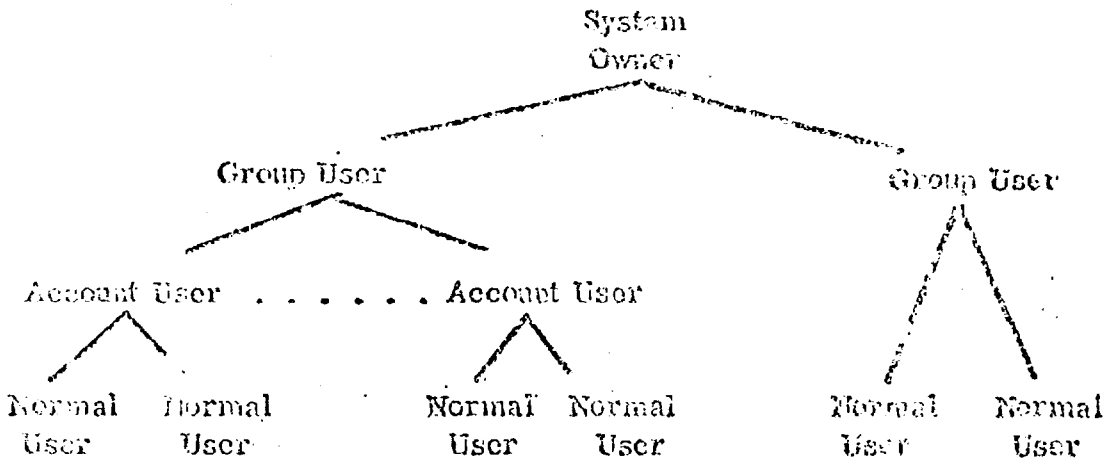
A user's resource allocations describe the bounds of the service he will get while running. In the list above, the first three items have associated with them both minimum and maximum limits. Thus, a given user may be guaranteed that he will get at least 3 percent of a CPU but no more than 5 percent, at least 10 drum pages for swapping but no more than 20, and at least 20 disk accesses every 30 seconds but no more than 40. In addition, he may have a guaranty for up to two terminal lines and three processes. His two terminal lines allow him to be logged in on two terminals, teletypewriters for instance; and his three processes allow him to have, for example, each terminal processing a job and one additional job running in the background.

Each of these resource allocations has a further qualification which describes the times in which they are valid. These validity periods may specify, for instance, that a given user has his CPU resource allocation available from July 22, 1970, to February 23, 1973, every day from 9:00 a. m. to 11:00 a. m. At least two advantages arise from this flexible and well controlled approach:

- Performance does not degrade as more and more users come on and drag it to a crawl. Guaranteed minimum service will still be met.

- Computing costs are easily controlled and relate directly to actual usage. The maximum guaranty and validity periods put an upper bound on costs and provide a reasonably direct relationship between costs and terminal connect time.

Knowing what a resource allocation is and how it relates to users, we can now discuss how resource allocations are portioned out to users. The system recognizes four kinds of users as depicted in the user hierarchy diagram.



One of these users is known as the system owner. He has possession of 100 percent of all of the resource allocations initially. From this pool he may break off pieces and pass them on to group users. A group user may, in turn, divide his resource allocations and distribute them to account users or normal users.

To illustrate this procedure, consider a large time-sharing service bureau as the system owner. The bureau has branch offices in many cities and each one of these offices is a group user. One of these offices has a few large industrial customers. Each of these industrial customers is an account user.

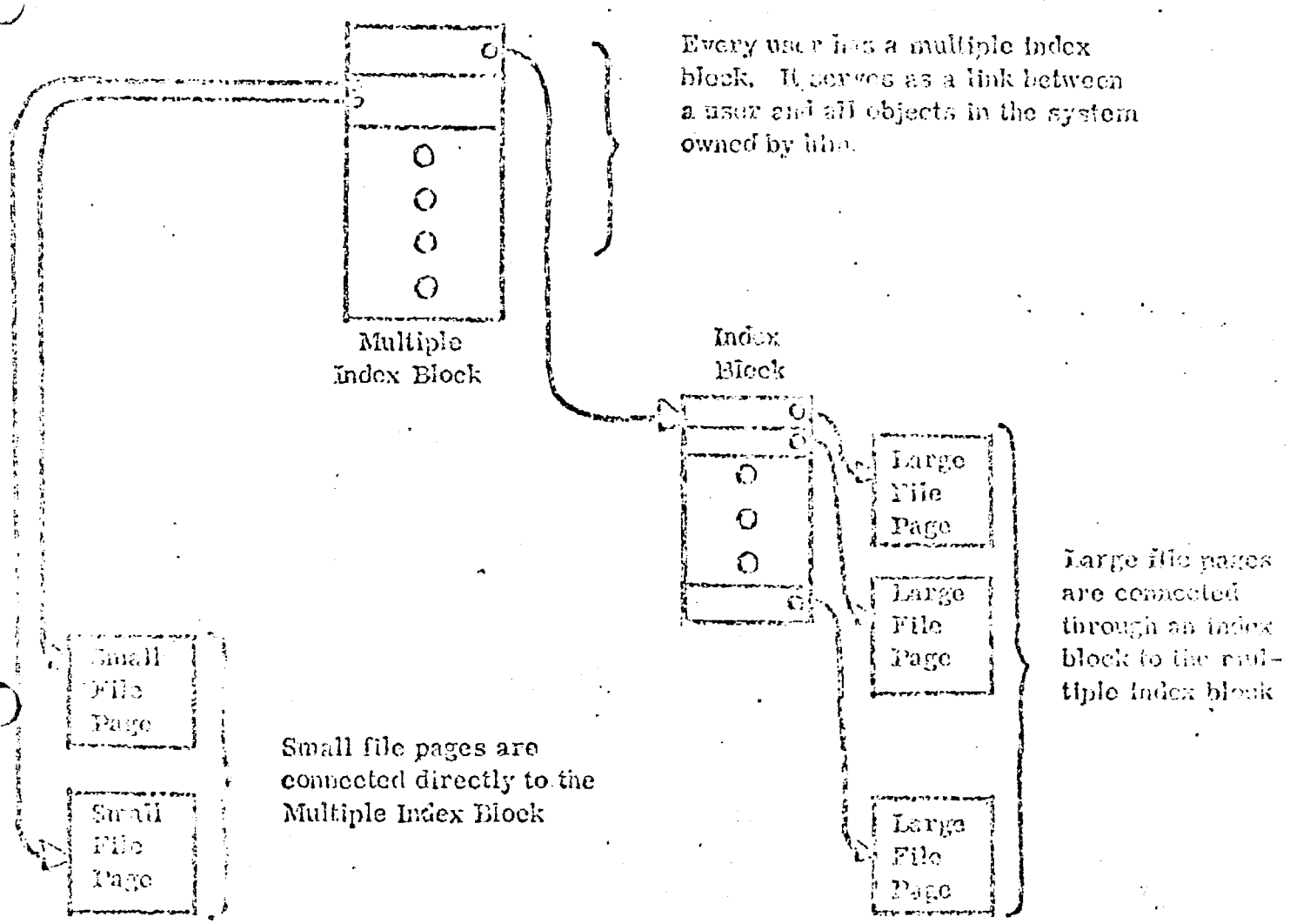
Each user in the hierarchy receives his resources when he logs on, which he enters the system from the next higher user. If an account user has experienced a particularly heavy demand during some peak time and portioned out all his resources, he is unable to accommodate any more log-ons until either one of his users logs off, and frees some resources, or he receives more from his group.

Each user known to the system has a multiple index block associated with his user number. This multiple index block connects a user to all of his "objects" in the system. In some cases the objects are contained within the multiple index block; in other cases only a pointer to the actual object is in the multiple index block. Precisely what an object can be is shown by the following list.

- Process
- Large file
- Small file
- Free object
- Resource allocation
- Friend access list
- Owner access list
- Access key

Resource allocations and processes are discussed elsewhere. All objects, with the exception of a free object, are manipulated by the system. Free objects are a means of leaving the system open ended. They are permitted by the system yet have no required properties and represent a hedge against the unknown. The remaining five objects will be discussed here.

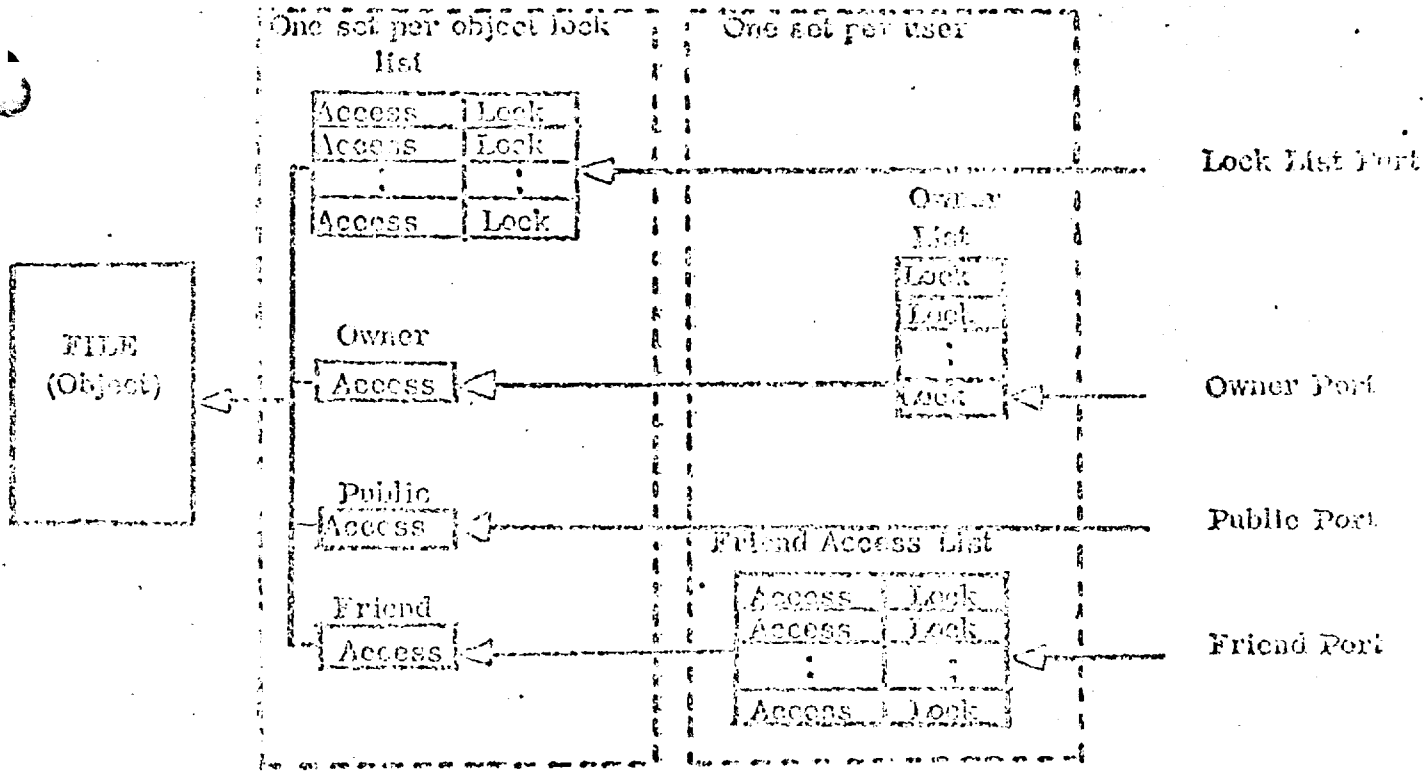
Small and large files are depicted by the following diagram:



Small files have pointers to each of their pages contained within a single item in the multiple index block. Large files, on the other hand, require another level of indirection. A large file entry in the multiple index block is a single pointer to an index block. The index block is a page (or a number of pages) which contains pointers

to the large file pages. Since the index block is usually a single 2K page, the normal maximum size of a single large file is 12 million bytes (4 million words). Using more than one page for an index block will allow an unlimited size for large files.

Objects in a user's multiple index block are protected by access keys. The following diagram, depicted a file as the protected object, will help explain the access key protection mechanism. Later we will discuss how the protection structure is built and how access keys are obtained.



The lock list attached to the file is the primary concept. It consists a paired list of accesses and locks. If someone wishing to access the file can present an access key which fits one of the locks, he will get the corresponding specified access. Access to an object may be none or any number of four possibilities: read, write, execute, and control. Read and write are self-explanatory. Execute might be the desired access to a proprietary program file. Control is the access necessary to manipulate, create, and destroy locking structures attached to an object. Control access is also necessary to destroy an object.

The PUBLIC access eliminates large lock lists containing common access and defines the access to the file regardless of what access key is presented. The FRIEND access provides a possible saving in space. Each user has a friend access list which looks exactly like a lock list. Any access key matching a lock on the friend access list will give the user who presented that key the corresponding access to all objects in that multiple index block. This access represents a maximum friend access which is further restricted by the FRIEND access to the file in question.

The owner list contains locks which can be matched by the owner of the multiple index block, and all directly "higher" users as depicted in the user hierarchy diagram. This is necessary, for example, so that an account user can take a file away from a normal user if the normal user has not returned disk pages to the account user when required. Access to those presenting keys which fit the owner list locks is restricted

only by the OWNER access to the object in questions. This access restriction is really artificial since anyone with owner privileges can set the OWNER access should they care to. However, it does provide a first line of defense against a blunder, and in that sense a user may wish to protect his files from himself.

Now that we have seen how lock lists can protect an object, we can discuss how a lock list is built and how access keys are obtained, used, and transferred.

There is a basic system function which will produce an access key upon demand. This access key is guaranteed to be unique for all time. It is simply a number in a protected box. The owner of the key may obtain the value of this number but the system will not let him or anyone else alter the value. When requesting access to an object, the system requires that an access key, in a protected box, be present.

Since the value of the key is available, he may tell another user its value. That user is free to put any number he wishes into a lock and may choose to put the value of the first user's access key in one of his locks, thereby granting access to some object. This is one way access keys may be used. Each user has an implicit access key in the form of his user number. In lieu of presenting any other access key when requesting access to an object, the system will use his user number.

Access keys themselves, like files, are objects. After obtaining one, a user may put it in his multiple index block and attach a lock list to it. This lock list

may contain the user numbers of other users. These users may then obtain this access key and store it in their multiple index block, later using it to open any objects with a matching lock.

The protection scheme ensures that no unauthorized person can gain access to another user's objects.

PROCESSES AND SUBPROCESSES

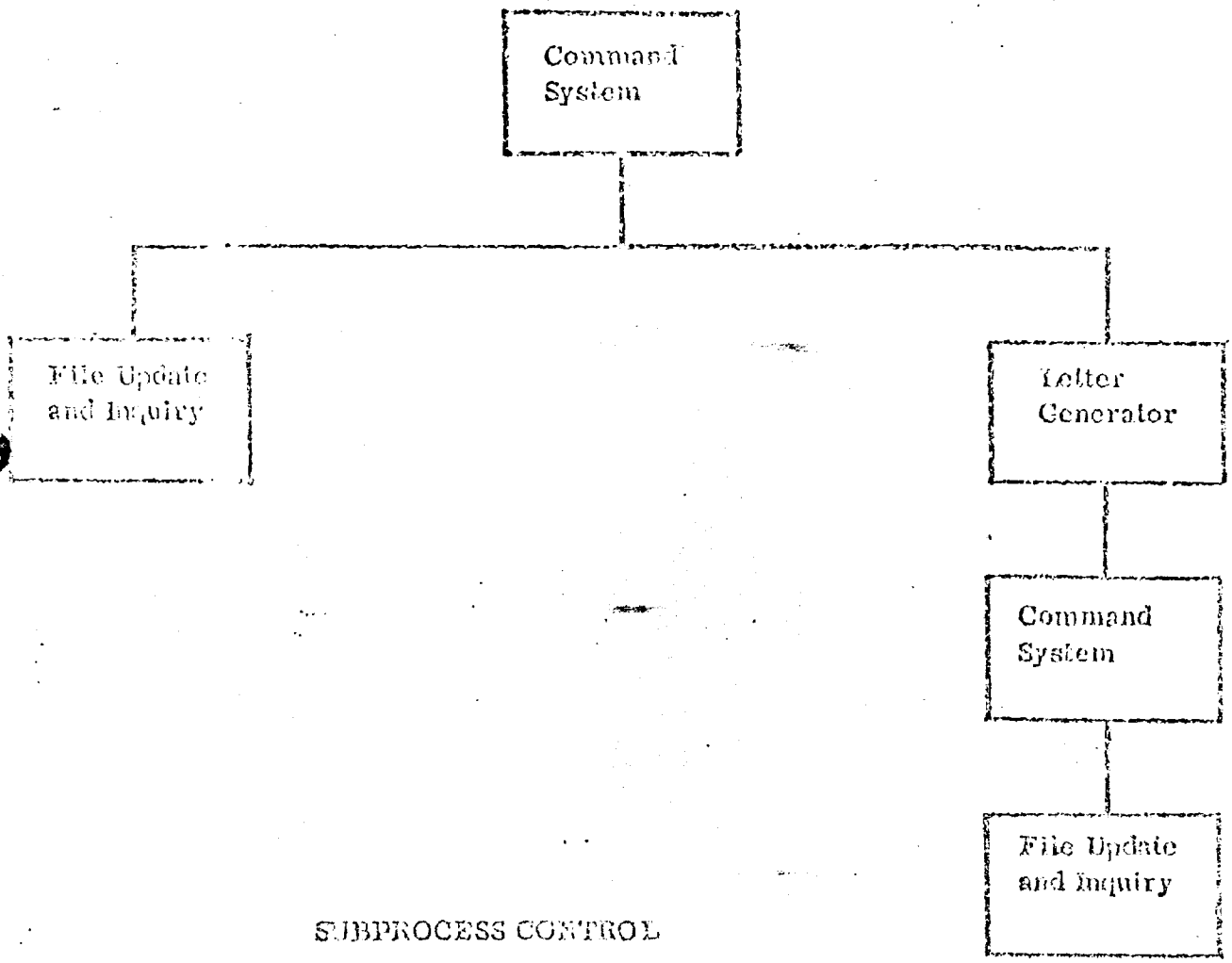
The fundamental computation unit in the DCC 500 System is called a "process." Up until this point, we have loosely been calling it a "job," but now we must be more specific. A process is a closely related sequence of computations initiated at some point in time and later terminated.

A user may be divided into as many as eight subprocesses at any one time. Every process is initialized at log-on time to contain two subprocesses: the system monitor and the system utility (includes command system). Thereafter, subprocesses will be added and deleted as the user invokes different subsystems.

For example, consider an automated banking system. The teller, whose duties will be somewhat expanded, has a terminal available. As customers wish to cash checks or withdraw savings, the teller can key in the account number and instantly see and update current balances. In addition, some customers will make inquiries requiring written response - maybe concerning the history of past and current loans. For this last purpose, we can use form letters which will have personalized sections filled in. Sometimes, as the teller is composing the letter, he will find he requires the query data base, and rather than start over, he can suspend the current letter generation subprocess, query data base, and armed with the required information then proceed to finish the letter.

Since the process is really a time progression of computations, it becomes a little difficult to illustrate it. The following diagram should be viewed as a time lapse photograph of the control for the example process described above. Each box

represents a subprocess with the entire structure being the process. The system monitor and utility subprocesses are not depicted since all other subprocesses invoke them frequently.



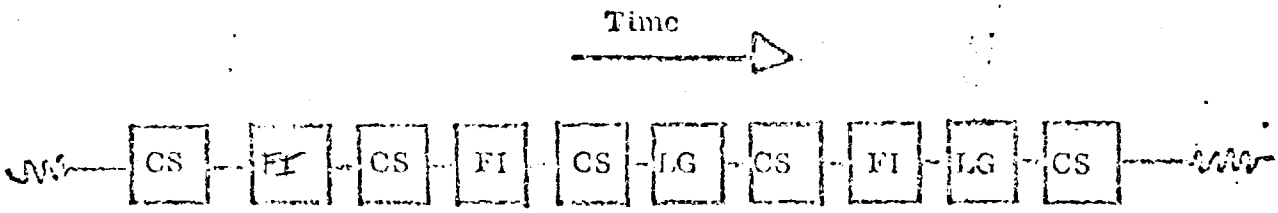
SUBPROCESS CONTROL

The diagram above shows only the control relationship between the various subprocesses. It does not show the actual time sequence of computation. The following illustration depicts how the time sequence might occur.

CS = Command System

FI = File Inquiry and Update

LG = Letter Generator



SUBPROCESS TIME SEQUENCE

The above diagram shows a hypothetical sequence where one inquiry and update is made (savings withdrawal); next a written response is requested and started. Some additional information is required to complete the letter, though, so an inquiry to the data base is made before the letter is completed.

Now that the concepts of process and subprocess have been defined, we can discuss their properties and relationships to the system. A process is known to the system by its current state. The current state of a process is a collection of various pieces of information residing in a "context block." Every process has a context block and it is the context block which defines the process. A context block is a page of memory and will contain, among other things, all state information necessary to interrupt and restart the process as time slicing occurs. It will also contain information about the core working set, open files, available access keys, and the process control structure.

The unsophisticated user need not concern himself about the context block other than knowing that it represents a page of overhead for any process he initiates.

The diagram showing a process time sequence implies that the computations all occurred in one terminal session. This may not be the case. The BCC 500 System allows a user to end the terminal session without destroying the process (context block). At some later time he may log-on again and take up where he left off. This ability to suspend processes is also used by the system in case a terminal is disconnected unexpectedly or the resource allocations assigned to the process expire. There is also provision for running a process in background mode or even deferred batch. A process in one of these two modes will run unattended by the user. Should the process run into some unexpected trouble, such as not being able to find a file it is supposed to write on, it will become suspended. The next time the user logs-on, he will immediately be informed of the suspended process. In this way he will not lose any useful work done before the problem occurred, and he may remedy the situation and let the process continue toward completion. The subprocess structure provides protection. No two subprocesses can occupy virtual memory simultaneously; this eliminates the threat of one clobbering the other. Furthermore, a subprocess may have protected entry points which define the only possible entries to it from other subprocesses. Thus, if a user's subprocess makes use of another user's proprietary package, neither can damage or gain any knowledge about the other user's program. The proprietary package protection problems are thus solved. Both the user and the package are totally protected from each other. The only remaining source of discontent is the case where the proprietary package, after solving the user's problem, retains the user's data for later examination by the proprietary package owner.

A computer manufacturer, for instance, may find that after using someone else's logic design package, the competition is producing his machine. Even this protection problem has been solved. Quite simply, the system can be instructed to prevent a proprietary package from generating any files of its own.

The process mechanisms contain a comprehensive means for intra-process communication. One method allows processes to direct interrupts at each other. A process, which may be in either an active or inactive state, can have an arbitrary number of its own interrupts armed. If one of these interrupts is invoked, control is transferred to corresponding interrupt routines which determine how to handle the interrupt. Processes may also share the same memory. File locking mechanisms as well as page locking semaphores provide a straightforward and well coordinated means for shared data.

Processes may also have a variable number of terminals connected to them. Thus, a situation like our previous bank teller example would not necessarily require a separate process for each teller. All teller terminals could be attached to the same process, thus saving the context block overhead. This savings becomes noticeable as the number of similar terminal functions grows large.

SOFTWARE

The BCC 500 System offers a wide range of popular software. Running in XDS 940 emulation mode, we can provide reliable time proven languages and systems:

- SNOBOL – a string processing language
- LISP – a list processing language
- QED – a context editor
- CAL – a conversational algebraic language
- DDT – a powerful debugger
- NARP – an assembly language with macros
- QSPL – a high level system programming language
- FORTRAN II– complete with an interactive debugger and random files.

BCC has implemented a complete set of upward compatible GE 400 series software. This software is undergoing an extensive in-house check-out to ensure high reliability. Once released, the user will be unable to detect an incompatible difference between a typical GE 400 series system he is familiary with and the service provided by BCC. Of course, he will benefit directly from the constrtent and reliable service offered by the BCC 500 system. Languages and user aids provided by the GE 400 compatible system include

- Extended FORTRAN IV
- Extended BASIC
- Editor
- File manager

- Command language
- Library

In addition to these popular languages and user aids provided by the GE 400 compatible system and XDS 940 emulation, we have an extremely powerful and complete set of in-house software. This includes a monitor and utility, file manager, command language and integrated interactive compiling system.

The compiling system, known as SPL, is geared for systems programming and includes as an integral part, a context language editor, a debugger, and a machine simulator. This in-house software has been used to implement itself as well as the GE 400 series system. We do not intend to release it for general use, however, until it has gained the polish and maturity that only come with age.

SUMMARY

The Model 500 System consists of two CPUs, several special purpose management processors (handling scheduling, swapping, storage allocation, character I/O, error detection and recovery), drum and disk transfer units, and a multiple module core memory, all attached to a high-speed, high-bandwidth integrated circuit memory system. Up to eight drums (320 million bytes) and up to eight disks (three billion bytes) may be attached to the system. Peripherals such as printers, magnetic tapes, and card equipment are treated as remote devices which can be attached by means of a special purpose processor. The system presents the following features to the prospective user:

- Accommodates large programs (maximum program segments of 384K bytes).
- Files of up to 12 million bytes (and greater if necessary).
- Multitude of languages, utility routines, and other user aids.
- Conversational and remote entry batch capability.
- Hardware protection of programs and files.
- Provisions for operating proprietary programs.
- Accessibilities assigned both to users and to programs.
- Hierarchical subprocess structure.
- User profile, specifying many characteristics of the user's virtual machine, including his resources and accessibilities.
- Fast access to large files.

The communications and multiplexing facility through which the terminals are brought into the system is centered around a special multiplexer designed and produced by BCC as an integral part of the system. These multiplexers can accommodate either full or half duplex operation of from 224 low-speed terminals to a lesser number with line printing and card reading equipment. The terminals can be of disparate types and can operate at different rates and use different codes. The remote multiplexers which communicate with the system's character input/output processor provide for the automatic detection and correction of errors arising in the communications lines.

Complex resource allocation algorithms built into the special purpose processors in the system manage and control the dynamic allocation of processor and memory resources. These algorithms permit minimum and maximum figures to be placed on many critical system resources demanded by users while operating on the system. Such a facility can guarantee the performance of the system as seen by a user or a group of users, independent of the load presented by other users or groups of users. This guaranty includes the activation of a user's program at a specified time.

The system has many features to provide for reliability and the detection and recovery from system errors. Briefly summarized, they include:

- Multiple CPUs, drums, disks, and auxiliary memory transfer electronics;
- Ability to operate using -- and automatically switch to -- one half of the memory modules;
- Provisions for backup special purpose processors and power generation equipment;

- Voltage, temperature, and other environmental monitoring to detect failures quickly and effect an orderly shut down if required;
- Parity on both memory data and addresses;
- Parity and 48-bit check sum on rotating memories;
- Routines which frequently scan system tables and initiate automatic system recovery quickly after an error occurs;
- Integrated communications system with private leased lines, alternate routines, and error detection and correction mechanisms;
- Three modes of disk dump backup occurring on a monthly, weekly, and continuous basis.
- A complete second system can access the same on-line storage.

We at BERKELEY COMPUTER CORPORATION welcome your comments and are willing to clarify any of your questions concerning the system.

REFERENCES

An in-depth understanding of some of the concepts employed in the BCC 500 System can be facilitated by the following publications:

CORE WORKING SETS

Denning, Peter J. The Working Set Model for Program Behavior. Communications of the ACM II, 5 (May 1968), 323 - 333.

FILE PROTECTION AND PROCESS/SUBPROCESS STRUCTURES

Lampson, B. W., Dynamic Protection Structures. Proceedings of the 1969 Fall Joint Computer Conference, 27 - 37.

SCHEDULING

Lampson, B. W., A Scheduling Philosophy for Multiprocessing Systems. Communications of the ACM II, 5 (May 1968), 347 - 360.

FAST MEMORY DEVICE

Lee, Francis F. Study of "Look Aside" Memory. IEEE Transactions On Computers, Vol. C-18, No. 11 (November 1969), 1062-1064.

GUIDING PHILOSOPHY

Pirtle, M. W., System Design of a Computer for Time-Sharing Applications: In Hindsight; published in: Critical Factors in Data Management, Editor: Fred Gruenberger, Prentice Hall, Inc. (1969) 57-62.