Inter-Office Memorandum NXS #10

To NXS Distribution Date June 18, 1975

From Doug Heying Location A1-62/Ext. 1531

Subject CP-V Description Organization Software Development DH-75-09

XEROX

This description is intended to assist us in our task of defining distributed processing for CP-V. An attempt will be made to systematically peel off the layers of 'system' and to describe what happens at each level.

There are fundamental concepts in CP-V which are all pervading and which provide the flexibility and ease of use of the system. The first of these is that CP-V is a file centered system. The ultimate convenience and flexibility is provided by keyed files. Keyed files allow an arbitrary name (key) to be given to any record within the file. Records can be added, deleted, expanded, or contracted with complete freedom. Then general structure is then used by many elements within the system for convenient filing of programs and data without the need to define specialized file structures. All file characteristics are retained on transfer to or from Xerox labelled tape, then eliminating the need for elaborate special utilities for each type of file. Another key feature of the file system is efficient use of space with full dynamic allocation allowed.

The second fundamental concept is the event driven scheduling algorithm. The scheduler maintains a set of state queues. Each user is in exactly one state queue at any point in time. Transitions between states are made by reporting events. The state queues are structured to provide priority for use of the CPU, priority for inswap, priority for outswap, etc. Some examples of events reported and the states resulting are given below:

Event	Resulting State
Terminal Read	Terminal Input Blocked
Activation Character Received	Terminal Input Complete
File Read	I/O in Progress
File Read Complete	I/O Complete
Terminal Buffer Limit Reached	Terminal Output Blocked
Unblock Limit Reached	Terminal Output Complete
Quantum End	Compute Bound
Resource Unavailable	Queued for Resource
Resource Available	Compute Bound

Note that in the above discussion of scheduling no discussion of a difference between batch and on-line jobs was made. There is no distinction made at this level of scheduling — scheduling the use of the CPU from millisecond to millisecond. The differences came at a different level of scheduling — scheduling the use of non-sharable resources.

The casual user of CP-V considers himself to be communicating directly between his terminal and a processor (Xerox supplied language, utility, or application) or a user program (a program developed by a customer using a Xerox supplied language) with some 'magic' interposed to make his kludgy terminal work nicely. Thus, if we put aside the magic for the moment, this is the outer shell of the 'system'. These programs do all of the useful work in the system. The remaining portions of the system either provide services to the above worker programs or are supportive of them in keeping the systems operating efficiently.

Memory

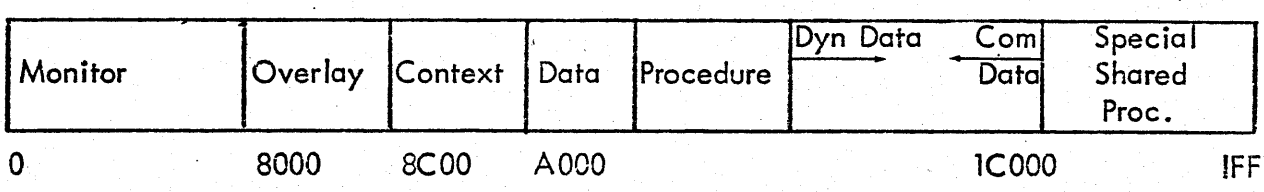
Before proceeding further down this line it will be useful to look at the memory structure of a CP-V system. This will be described from three viewpoints:

1. The virtual memory associated with a specific user process (always exactly 128KW addressable).

2. The physical main memory.

3. The system virtual memory (swap space).

The user process is best described in tangible form as the context providing continuity between steps of a batch job or on-line session. This is a collection of information describing privileges, resources used, resources allowed, file buffers, etc. A subset of user context, the JIT (Job Information Table) also contains a description of the remainder of the virtual memory associated with the user process and its location in swap store and its location in physical main memory when in. The layout of virtual memory is as follows:



The map and access controls are used to make best use of physical main memory and to limit the scope of access to memory. Each user addressing space will be uniform each time he receives the use of the CPU even though he may reside in different physical pages. The meaning of each of the areas of virtual memory is now given along with the map and access control settings.

Monitor - This is the resident portion of the CP-V operating system (to be described in more detail later). This always occupies 0 - 7FFF of all users virtual space and is always resident in the same locations of physical main memory. The map is loaded once and is not changed. The access controls are set to read only for the first page and to no access for the remainder. Thus, the user cannot access the monitor, but the monitor (master mode) can.

Overlay - This area is used for less frequently required parts of the monitor and is used only as required. The map is loaded to point to the physical pages where the current overlay currently resides in physical memory. The access controls are set to no access.

Context - This area contains the personality of the user process as described above. The map is loaded with the pages where the context is currently resident. The JIT has an access control of read only. The remainder have access controls of no access.

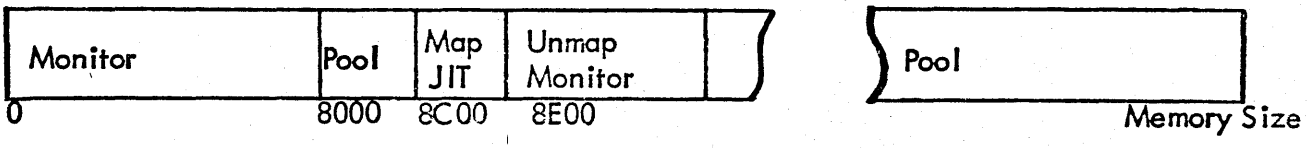
Data - This area contains the data to which the procedure is specifically bound. The map is loaded with the pages where the data is currently resident. The access controls are loaded with all access.

Procedure - This area contains the instructions which make up the logic of the program being executed. This program may be a private program or a shared processor. In either case the map is loaded with the pages where the procedure is currently resident and the access controls with read and execute. In the shared processor case there may be more than one user with identical map contents in this area.

Dynamic Data and Common Data - These provide data storage areas to the procedure that can be expanded or contracted independently. The map is loaded with the pages where the data is currently located. The access controls are set to all access. The area between these two areas is the unused portion of virtual space and is normally quite large. The access controls are set to no access.

Special Shared Processor - This area can be used for one of a variety of things to be associated with the user. It can be occupied by a command processor (TEL, EASY) or a library (FORTRAN runtime, DMS data manager) or a debugger (DELTA). The map and access controls are the same as for procedure with a shared processor. The unused portion of the area is set to no access.

The physical main memory is laid out as follows:



The monitor is divided into two parts. The first part which must exist in every user map occupies the same physical space or virtual space and is mapped 'one to one'. The second part which operates only unmapped and never accesses a user virtual memory occupies an area of physical memory which is used for other things in virtual space. The space marked Pool is the general resource of pages of memory available to put users, shared processors, etc. into. The Monitor JIT is at the same place in physical memory as each user JIT is in virtual memory in order to allow the subjective accounting clock (Counter 4) to count into the appropriate users JIT (mapped) or the Monitor JIT depending on the setting of the map (current user) and the mode (mapped or not) at the time that the clock ticks.

The system virtual memory is allocated as required to contain an image of the unique part of every users virtual memory plus an image of every shared processor. The space is allocated to keep each entity in continuous rotational position on the device to minimize latency. The memory management problem then is one of managing the physical main memory and the system virtual memory. Every change in the virtual memory for a user is reflected in a change in swap space allocation.

Worker Programs

With the above as a background, we can now proceed to discuss other worker programs in the system. All worker programs operate within the framework of a 'virtual machine' defined as follows:

1. Addressing range defined by user virtual memory.

2. Slave mode instruction set.

3. Monitor services invoked via CAL's.

The layout of user virtual memory suggests several worker programs besides processors and user programs. Some of these are:

1. Command Processor. These are the highest level of user interface which interpret control commands and request the monitor to associate the appropriate programs with the user. These are TEL, CCI, EASY.

2. Log On/Off Processor. This is the first program associated with a user when a connection with the system is established. No other program can be associated until LOGON has verified the users identity. This identity is then the basis for all security within the system. This processor is also the last program invoked before a user disappears. It updates accounting records, etc.

3. Libraries. These provide an extended set of services which are used by a large enough number of programs to a candidate for sharing. These are always subroutines which do not run on their own but must be called by a processor or user program.

4. Debugger. These co-exist with a user processor and monitor its execution to assist in debugging the program.

5. Loader. This is the system services which combines a number of Relocatable Object Modules (ROM's) as output by assemblers or compilers in standard object language. The result of this process is an executable file called a Load Module (LMN-lemon).

JOBS

Before listing other worker programs it is necessary to define the three kinds of jobs that run on CP-V. A job is the external manifestation of what was described internally as a user process. It is a set of tasks specified from a card reader or terminal with continuity provided by user context.

A batch job is one whole entire control stream and resource requirements (tapes, spindles, etc.) is known to the system before the job is put into execution. Given this knowledge it is possible to schedule batch jobs to optimize the use of non-sharable resources. Batch jobs are disconnected from any human interaction and output, in general, is not delivered until completion of the job.

An on-line job is one which is connected uniquely to a timesharing terminal. Resource requirements are not known in advance and thus must be acquired on a contention basis. This is workable since a human is in the loop and can make decisions when resources are unavailable. Other than being able to pre-allocate resources, an on-line job can do everything a batch job can, including line printer use, etc. Note that line printers, IRBT's, etc. are not considered non-sharable resources. More on this later. The terminal is always in control of a timesharing job.

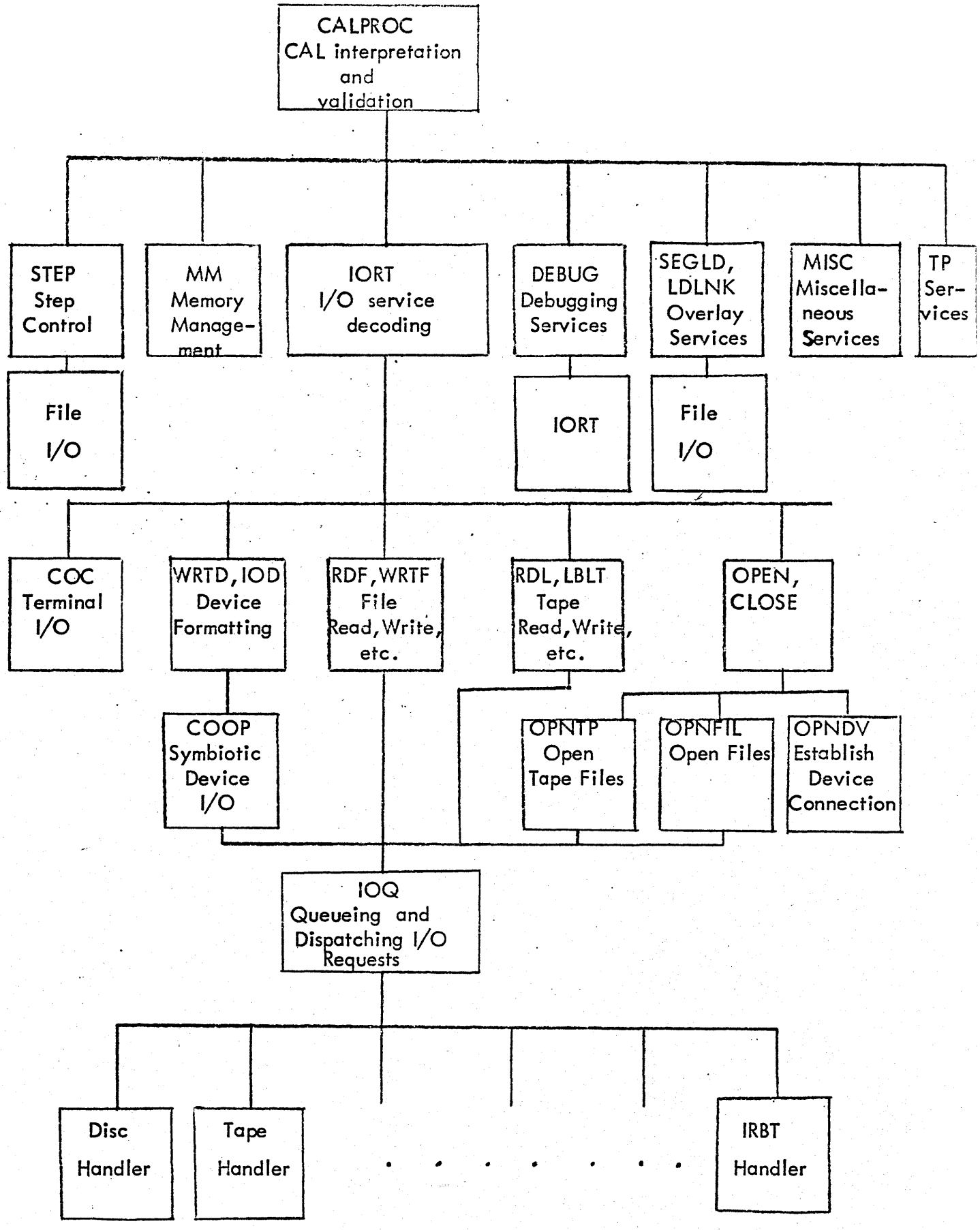
A ghost job is 'none of the above'. It has no input control stream and is not connected to any terminal. It is usually providing some service to the monitor. Its actions are controlled by communication via a file or another form of internal communication. Some examples of worker programs which operate as ghost jobs are:

1. File Space Allocater (ALLOCAT - alley cat). This job manages the entire pool of public file space available to the system.

2. Batch and Remote Batch Scheduler (RBBAT - rabbit). This job establishes connections to all remote batch stations. It also controls and schedules the use of all symbiont input and output devices (including remote). RBBAT determines and retains all information about batch job resource requirements. It uses this information plus the information about resources acquired on a contention basis to determine which batch jobs can be run concurrently to optimize resource usage.

3. Error Log File Builder (ERR:FIL). This job accepts error log entries in raw form from the monitor and inserts them into a file which can then be messaged in various ways by other programs (e.g. ELLA - error log lister and analyzer).

4. File Maintenance (FILMS). This job is responsible for all file backup - restore operations and archival processing.



Monitor Service Control Flow

5. File Inconsistency Repair (FIX). This job is responsible for repair of any damaged part of the file system.

In addition to the above classes of jobs, there is another which is really a form of batch job, but behaves significantly different from ordinary batch jobs. Transaction Processing is basically a set of two jobs cooperating to provide a TP service. The first is a job called the Terminal Interface Controller (TIC) which acquires a set of terminals as I/O devices. Note the difference from on-line jobs. Here the program is in control and the terminals are slaves. The TIC formats input into formal transactions and accepts formal reports and formats them for terminals. The actual processing of transactions is done by another job, the Transaction Processing Load Module (TPLM). The TPLM is made up of a Transaction Processing Control (TPC) module and several Transaction Processing Modules (TPM). The TPC is a collection of service routines which call the appropriate TPM and provide journal service. The TPM's are customer provided and have a simple interface for their processing.

Enough discussion of drones, now let's get into the guts of the system - the monitor itself. Basically the function of the monitor is two fold - to provide services (passive) and to keep everything perking along nicely (active).

Monitor Services

Monitor Services are those functions of the operating system which provide the user with a very rich virtual machine on which to run a program. They run the gamut of complexity from returning the time of day to elaborate file management services. Since the services are running on behalf of the user they run mapped and since they must access the monitor they run master. There are two ways to invoke monitor services: the CAL instruction and any other trap. The traps in general signify an error and appropriate action (which may be 'give control to the program') is taken. The following chart shows the flow of control after a CAL instruction is executed. Each lower level can be interpreted as a subroutine which will ultimately return.

This chart shows the gross structure of the monitor services within CP-V. The significant points shown by the structure are as follows:

1. Common CAL interpretation and validation.

2. Fanout of services into modules with related services.

3. I/O services with several distinct levels:

   a) logical I/O handling (WRTD, RDF, etc.) providing device independent I/O

   b) interposed co-operative (COOP) handling allowing direct device usage or blocking into a symbiont file

   c) centralized I/O queueing allowing bandwidth management, optimization algorithms, etc.

   d) modular reenterable handlers for each unique device type.

A brief narrative follows for each component shown on the chart:

1. STEP. This module provides the services necessary for transition from step to step within a job and provides for job initiation and termination. It makes use of file services for program retrieval and calls RBBAT for multi-batch scheduling and user MM to allocate memory.

2. MM. This module handles requests for changes in the virtual memory of a user. These requests can come from a program or from STEP. MM is responsible for coordinating space in each users virtual memory with that in the system virtual memory (swap storage).

3. DEBUG. This module provides monitor services for programmed debugging facilities. It uses monitor I/O to direct debug output as the user desires.

4. SEGLD, LDLNK. SEGLD provides overlay services between segments of a Load Module. LDLNK provides ~~services~~ for calling one Load Module from another. is a user program service where program

5. MISC. This module provides a variety of convenience services such as time of day, interval time delays trap control, break control, etc. Also provided are Enqueue/Dequeue services to allow coordination between user processes for such things as file sharing.

6. TP. This module provides the common journal and system queue facilities used by Transaction Processing.

7. IORT. This module decodes requests for I/O service and calls the appropriate module depending on the assignment of the dcb (data control block). The dcb is the logical unit addressed by the program and may be externally assigned to devices, files, etc.

8. COC. COC handles all character oriented communications. It provides all of the editing and echoing facilities evident from a timesharing terminal. It is responsible for movement of data to/from a central buffer pool for write/read and for translation between Xerox EBCDIC and device codes.

9. WRTD, IOD. These modules are responsible for device specific formatting which may be required on device I/O.

10. RDF, WRTF. These modules are responsible for all blocking, deblocking, key searches, positioning, etc. which is required for file I/O. ALLOCAT is called as necessary for allocation of file space. IOQ is used for physical transfers.

11. RDL, LBLT. These are the analog of RDF, WRTF for tape.

12. OPEN, CLOSE. OPEN is used to validate a dcb and to gain permission to use the device or file to which the dcb is assigned. CLOSE removes that validation until a subsequent OPEN.

13. OPNTP. This module does the actions which are specific to tape in the open process. It checks that a drive is allocated, verifies serial numbers and locates the proper file if labelled tape.

14. OPNFIL. This module does the actions specific to files in the open process. If a private pack is referenced, it verifies that a spindle is allocated and verifies the serial number. In any event, the file is located, security is checked, and sharing conflicts are resolved. If a new file is being opened, the file is catalogued.

15. OPNDV. The module takes specific actions relative to the use of devices. The users capabilities are checked to see if he is allowed to use the device.

16. IOQ. This module queues requests for any physical device and dispatches the requests to the appropriate handler when possible.

17. Handlers. In each of these modules the device specific action is taken. For example, the handler for IRBT lines handles all multi-leaving implications so that no using processor or other part of the monitor need know about this feature of IRBT.

Until now we have been discussing the portion of the monitor services which execute on behalf of the user, i.e. in the users context. In support of this, there are other portions of the monitor services which execute unmapped, i.e., not in any users context. An example of this is the I/O interrupt receiver which notes I/O complete by reporting event, then attempts to schedule more I/O. Other examples are the COC input and output interrupt levels. These simply put characters in buffers or remove characters from buffers until activation characters are reached or buffers reach a threshold at which time events are reported. The reporting of these events will of course permit the resumption of the appropriate monitor service in the user context.

This leaves us with the control part of the monitor; that which keeps the system running smoothly. Basically, this comes in two parts: the scheduler and swapper which cooperate to optimize the use of CPU and memory resources, and the symbionts which optimize the use of unit record peripherals. The actions of the scheduler and swapper are fairly clear from the foregoing discussion. Whenever an event is reported which raises the priority of a user to a level where he should be brought into memory to be executed, the swap scheduler is called to determine the user to be inswapped and the user(s) to be outswapped. If the swap scheduler is able to schedule a swap, the swapper is called to initiate the swap. The swap then proceeds in parallel with further action. The scheduler then determines the highest priority user in main memory and gives him control.

The symbiont system is simply a mechanism by which the use of unit record peripherals is decoupled from the program use of the devices. This is done by allowing program output to be put into a symbiont file (COOP) to be retrieved later by the symbiont. This accomplishes two things. First, it allows a buffer pool to smooth the peaks and valleys caused by program behavior. Second, it permits many programs to concurrently output to the same unit record device.

There is one more function which should be discussed; the area of accounting measurements and performance monitoring. It is difficult to point to a particular module for these functions although there is a module which records data. This is because all of the functional modules are responsible for reporting significant milestones which may be of interest for accounting or performance purposes. Accounting information is recorded in each users JIT until LOGOFF. Performance data is kept in resident tables where it may be sampled by any worker program to be displayed in any way. A reporting program (STATS) is supplied with the system.

Douglas W. Heying

D. Heying

ba

Distribution:

G. Gillette
I. Greenwald
K. Isaac
G. Justus
H. Kakita
S. Klee
L. Krasny