ESD-TR-71-368

MTR-2150

THE DESIGN OF THE VENUS OPERATING SYSTEM

B. H. Liskov

OCTOBER 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 6710

Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract F19(628)-71-C-0002

AD733585

ESD-TR-71-368

MTR-2150

THE DESIGN OF THE VENUS OPERATING SYSTEM

B. H. Liskov

OCTOBER 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 6710

Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract F19(628)-71-C-0002

FOREWORD

This report presents the results of an effort conducted by The MITRE Corporation, Bedford, Massachusetts, in support of Project 6710 under Contract F19(628)-71-C-0002. Dr. J. B. Goodenough (ESD/MCDT-1) was the ESD Project Monitor.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

EDMUND P. GAINES, JR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command & Management Systems

ABSTRACT

The Venus Operating System is an experimental multiprogramming system which supports five or six concurrent users on a small computer. The system is defined by a combination of microprograms and software. This paper describes the development of the system, with particular emphasis on the principles which guided the design.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

The Venus Operating System is an experimental multiprogramming system for a small computer. It supports five or six concurrent users, who operate on-line and interactively through teletypes. It may be distinguished from other multi-user systems in that it primarily caters to users who are cooperating with each other; for example, a group of users sharing a data base or building a system composed of cooperating processes.

The operating system was produced to test the following hypothesis: The difficulties encountered in building a system can be greatly reduced if the system is built on a machine with the "correct" architecture. We had in mind a complex system whose data and processing requirements vary dynamically; for example, an operating system or an on-line data management system. We felt that machines available today do not support programming of such systems very well and that considerable software complexity is introduced to cope with the inadequacies of the hardware.

First, it was necessary to build the machine with the correct architecture. This was done through microprogramming on an Interdata 3 computer; the result is called the Venus machine. The microprogram contained solutions to some of the time-consuming and complex tasks performed by such systems as well as useful mechanisms for building these systems.

Next, it was necessary to use the Venus machine for a software application. An operating system was selected as the initial effort because it was the type of system the machine was intended to support and because it would provide a facility which could support later applications.

SYSTEM DESIGN PRINCIPLES

Two main principles were followed in the design of the operating system:

1. The system was built as a hierarchy of levels of abstraction, defined by Dijkstra.[1] We expected this would lead to a better design with greater clarity and fewer errors. A level is defined not only by the abstraction which it supports (for example, virtual memories) but also by the resources which it

1

uses to realize that abstraction. Lower levels (those closer
to the machine) are not aware of the resources of higher
levels; higher levels may apply the resources of lower levels
only by appealing to the functions of the lower level. This
reduces the number of interactions among parts of a system
and makes them more explicit. Examples of levels of abstrac-
tion, which occur in both the microprogram and software, will
by given throughout the paper.

2. The features of the Venus machine were allowed to directly
    influence the operating system design in order to evaluate
    the effect of the architecture on the development of the
    software.

Several other principles also guided the design of the system:

3. Efficiency of performance was always considered when making
    design choices but was not the major criterion.

4. Independent users were protected, insofar as possible, from
    each other's mistakes by limiting the effects of errors to
    the user or group of cooperating users involved.

5. Users were given as much access as possible to the features
    of the machine and the software mechanisms developed for the
    operating system.

This paper is primarily concerned with the development of the
operating system according to the design principles with special
emphasis on the two main principles. In the next section the Venus
machine features of most influence on the operating system design are
described. The following section explains how these features were
extended to support the design of the operating system. Then the
design of the resource management portion of the system is described.
Finally, the design experience is evaluated. The Appendix gives a
brief summary of the services available to a user of the system, for
the benefit of those who wish to know the external characteristics of
the system.

SECTION II

THE VENUS MACHINE

HARDWARE

The Venus machine was built by microprogramming an Interdata 3,
a small, slow, and inexpensive computer. The microprogram is stored
in a read—only memory; it is limited to 2,000 microinstructions which
imposes fairly severe restrictions on its content.

The Interdata is connected to several teletypes, a card reader,
a printer, and two magnetic tapes. In addition, there is a small
paging disk with a capacity of half a million bytes of storage. The
disk and tapes have direct memory access through hardware selector
channels; the other devices transfer data a byte at a time.

THE MICROPROGRAM

In addition to an ordinary instruction set, the Venus micropro-
gram supports a number of non-standard architectural features. Those
features most important to the design of the operating system will be
briefly described here;[1] they are:

segments,
multiprogramming of 16 concurrent processes,
a microprogrammed multiplexed input/output channel, and
procedures.

Segments

Segments are named virtual memories, as defined for MULTICS.[3]
Each segment contains a maximum of 64 thousand bytes of data and has
a 15-bit name; segments are the primary storage structure on the Venus
machine. Segments and core memory are both divided into 256-byte
pages. Information about the contents of each core page is kept in a
single, centralized core-resident table, the core page table, which
is used by the microprogram to map virtual addresses into real addresses.
All references to a given virtual address will be mapped by the micro-
program into the same real address, which implies that segments are
physically shared among processes. In addition, there is no way for
one process to protect a segment from access by other processes. This
restriction makes sense only because the Venus machine was designed to
support a system composed of cooperating processes.

---

[1] For a complete description of the Venus machine see Reference 2.

3

Paging on the Venus machine is performed on demand. If the microprogram cannot locate the desired segment page in the core page table, it starts a software routine, the page fault routine, to fetch the page from the disk. The page fault routine acts like a subroutine of the microprogram, called by the microprogram when needed and returning, via a special instruction, to the microprogram at the point where the page fault was detected. There are two main advantages to handling page faults in software: It permits experimentation with methods of selecting the pages to be swapped out (at present, the page not referenced for the longest time is removed), and it makes the microprogram independent of the particular device(s) being used for paging. The Venus machine is not completely defined without this piece of software; therefore, we view it as a part of the machine, rather than as a part of the system to be defined on that machine.

Multiprogramming

A process is defined to be a procedure in execution.[2] Sixteen processes are supported by the Venus machine; each runs on a virtual machine consisting of an address space and a work area. The address space encompasses all segments and is the same for all processes, although only a few segments are of interest to a particular process. The work area is permanently located in core and contains the state word[4] of the process; this comprises about 150 bytes of process-related information including the general registers, program counter, and other information about the state of the process.

Scheduling of the CPU to the processes is performed by the microprogram, which enabled us to define a single uniform mechanism for passing control of the processor among the processes. This mechanism, used even for indicating the end of input/output, is provided by semaphores, first defined by Dijkstra.[1] Semaphores are used to control the sharing of resources and to synchronize processes. Two operations may be performed on semaphores: P and V. P is performed when a process wishes to wait for an event to occur or a resource to become available. A process performs a V to free a resource; either a process or the input/output channel performs a V to signal the occurrence of an event.

Dijkstra defined a semaphore to be an integer variable and an associated waiting list. In Venus the waiting list is represented by a queue, and the association is made explicit by defining a semaphore to be an ordered pair (count, link). If count is negative, its absolute value equals the number of processes on the queue. In this case,

---

[2] Dennis and Van Horn[4] define a process as follows: "... a process is that abstract entity which moves through the instructions of a procedure as the procedure is executed by a processor."

4

link points to the work area of the process which most recently per-
formed a P on the semaphore; in the work area of that process is a
pointer to the work area of the next newest process on the queue, and
so on. Also in the work area of each process is a priority which can
be set by software. When the time comes to remove a process from the
queue (because a V was performed on the semaphore), the process with
the highest priority is removed; if the highest priority is associated
with more than one process, the one which has been on the queue the
longest is selected.

Process-swapping in Venus only occurs as the result of P's and
V's being performed by processes or by the I/O channel. All pro-
cesses in Venus are in one of three states:

a single running process,
processes which are ready to run, or
processes which are blocked.

Ready processes are listed on a special queue, the J queue. Blocked
processes are on queues associated with semaphores. If the running
process performs a P, requesting a resource which is not available or
waiting for an event which has not occurred, it becomes blocked; at
this point, the highest priority, oldest process on the J queue is
selected to be the new running process. If a V is performed, some
blocked process may become ready. If this process does not have higher
priority than the running process, it is added to the J queue; otherwise,
it becomes the running process, and the old running process is added
to the J queue. The running process always has a priority at least
as high as all ready processes.

Thus, when implementing software on the Venus machine, P's and
V's are used to synchronize processes and control shared resources,
and priorities can be assigned to tune the system or accommodate some
favored process.

Input/Output Channel

The microprogrammed input/output channel relieves the software
from all real-time constraints associated with devices by permitting
the specification of I/O transfers in increments which suit the require-
ment of the devices. The microprogram runs the channel between the
execution of instructions; from the software point of view, the chan-
nel is running simultaneously with the execution of instructions. The
channel signals the completion of I/O by performing a V on a special
semaphore located in the work area of the process which requested the
transfer. When the process wishes to synchronize with the I/O, it per-
forms a P on this semaphore. Figure 1 shows the relationship of the
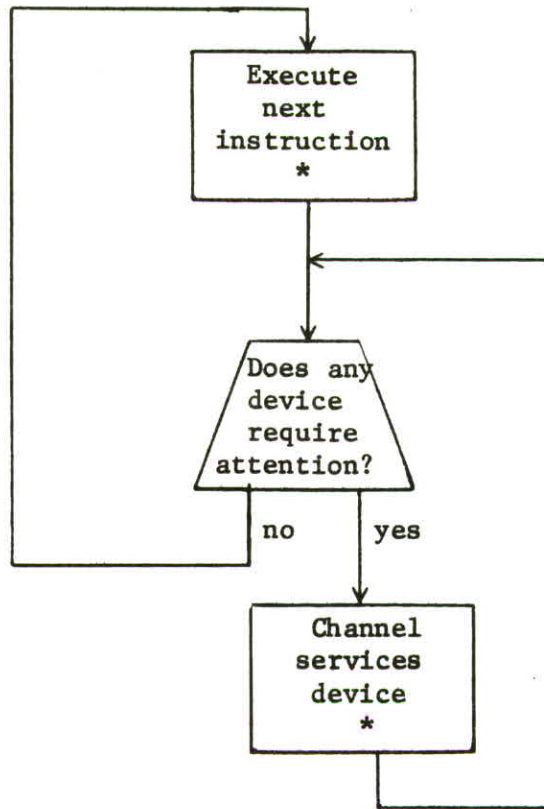channel to the rest of the microprogram.

5

Figure 1. Flowchart of Microprogram

This figure shows the relationship of the channel to the execution of instructions. The asterisks indicate places where changes in the states of processes can occur as the result of the performance of P's and V's.

6

Procedures

Each procedure is stored in a unique segment. Call and return instructions switch a process from the instructions in one procedure to those in another; they also save and restore part of the state word at the time of the call. Arguments and values can be passed in separate segments which are used as push-down stacks, referenced by push and pop instructions.

Sharing procedures is desirable on the Venus machine; only reentrant procedures can be successfully shared. The primary support for reentrant procedures comes from the separate virtual machines. The call, return, push, and pop instructions provide a reentrant procedure interface. In addition, no easy way of storing into procedures is available, and segments provide private and temporary work space whenever this is needed.

LEVELS OF ABSTRACTION

The design principle of levels of abstraction was applied to the microprogram as well as to software. Levels supported by the microprogram and the page fault handler include the virtual memory abstraction, the virtual machine abstraction, and the I/O channel which really provides virtual devices – for example, a card reader which reads an entire card at once. Figure 2 illustrates the resources and methods of appeal associated with these levels.

Abstraction	Resources	Method of Appeal
segments	core page table, directories (maps of segment name and page into disk address)	segment reference or EL1 instruction
virtual devices (microprogrammed channel)	device, device status word table	SIO instruction, channel commands
virtual machines	J queue, processor	P and V instructions

Figure 2. Levels of Abstraction Supported by the Venus Microprogram

SECTION III

EXTENSIONS TO THE VENUS MACHINE

The second main design principle was to let the features of the
Venus machine influence the design of the operating system. Thus, we
expected the system to use segments for data and to be composed of a
combination of reentrant procedures and asynchronous processes. A
feeling for the usefulness of such structures can be gained by con-
sidering how they correspond to the way in which the system most con-
veniently performs its work.

We were interested in designing a system to support multiple
users. Each user is assigned a separate virtual machine and is thus
represented as an independent process. To support the users, the
system must perform certain tasks. Some will most naturally be per-
formed on the user's virtual machine by reentrant system procedures,
which use segments to hold user-related data. Other system tasks are
logically asynchronous with the user (for example, running input/output
devices for the system as a whole); these tasks can be made physically
asynchronous by assigning them to separate processes. Thus, the work
done by the system can be distributed among the processes so that
logically concurrent and asynchronous tasks can be performed by physi-
cally concurrent and asynchronous processes. This leads to clarity
in the design.

To make the features of the Venus machine more convenient to use,
three levels of abstraction were defined in software. The lowest
level supports the convenient use of segments; the second, communica-
tion between processes; and the third increases the number of processes
available by allowing processes to share a virtual machine. These
levels are described below and are illustrated by Figure 3.

DICTIONARIES

Building a system out of segments containing procedures and data
requires cross-referencing between segments. The Venus machine sup-
ports references to segments by internal segment name, which is not
a very convenient item for a programmer to use or remember. In
addition, internal segment names are dynamically assigned (by the
page fault handler); the programmer needs a static name. Therefore,
external segment names were introduced.

Abstraction	Resources	Method Used to Support Abstraction
additional processes	virtual machines	POLLER
communication between processes	segments containing queues	queues accessed by queuing procedures
external segment names	segments containing dictionaries	dictionaries accessed by dictionary procedures

Figure 3. Levels of Abstraction Supporting Extensions to the Venus Machine

10

To support external segment names, a mapping between external and internal names is required. This is supplied by dictionaries, which are stored in segments. An external segment name is actually a pair of symbolic names: the symbolic name attached to the segment and the symbolic name of the dictionary which should be used to perform the mapping. One special dictionary contains the mappings between the symbolic and internal names of all dictionaries. The two-level external name allows related segments to be grouped together (referenced through the same dictionary) and makes it easier for a user to obtain unique names.

Dictionaries are intended to be shared and may be referenced simultaneously. While a dictionary is being changed, it does not contain consistent data. Therefore, each dictionary has an associated semaphore which is used to control sharing. System procedures access dictionaries only by calls to special reentrant dictionary procedures, which perform P's and V's on the associated semaphore where appropriate. Restricting dictionary access to these procedures guarantees that the semaphores will be used correctly (for example, every P will eventually be followed by a V). In addition, it means that the structure of dictionaries can be changed without affecting other parts of the system.

Although dictionaries are primarily intended to support external segment names, they are actually a general mapping facility and are occasionally used as such by the system.

QUEUES

Processes which synchronize with each other may also need to send and receive information. P's and V's permit processes to wait for or signal the occurrence of events but do not contain any information about what the event is. A process may be waiting for several different events; it must know which event occurred in order to take appropriate action. A mechanism was required which allows one process to send information and the same or another process to receive it. Queues were selected for this purpose because they supply a chronological ordering for their elements.

As with dictionaries, only one process at a time may access a queue; therefore, queues are protected by semaphores, and system access is restricted to system-provided queuing procedures, which use semaphores correctly. All queues are held in a single "queue segment"; the location within this segment of the head of a particular queue is obtained from an associated "queue dictionary."

THE POLLER

One consequence of distributing the work of the system over separate processes is that many processes are required. To have all the processes we needed with enough left over for the users, we defined a simple reentrant function called the Poller, which increases the number of processes available by permitting processes to share a virtual machine. The Poller uses a table to determine which processes are sharing its virtual machine. It performs a P on the semaphore controlling all the processes; when it awakens, it polls them until one recognizes the event which occurred.

The Poller is intended primarily for system use. Every virtual machine used by the system (five in all) is controlled by the Poller, which permits the later redistribution of the load if necessary. Selection of which processes share which virtual machine is made on the basis of overall system efficiency.

SECTION IV

RESOURCE MANAGEMENT

An operating system must manage the resources available on a machine in such a manner that deadlock is avoided and access to resources is distributed fairly among the users. Resource management in Venus will be described in some detail, partly because this is an interesting part of the operating system and partly as an example of the use of levels of abstraction and distribution of work among processes. Resource management is primarily provided by independent processes synchronizing and communicating with each other.

If several processes compete freely for resources, deadlock may result. The simplest example of deadlock is:

Process 1 owns Resource A and is waiting for Resource B.

Process 2 owns Resource B and is waiting for Resource A.

Neither Process 1 nor Process 2 can continue. We wanted to avoid deadlock in the Venus Operating System, which required careful design of resource management. Resources to be managed by software include segments and input/output devices; core, the paging disk, and the CPU are managed by the microprogram. Virtual machines must also be managed, but only in the sense that no more can be assigned to users than are available.

MANAGEMENT OF SHARED DATA SEGMENTS

The management of shared data segments is difficult because general solutions which insure that deadlock will not occur tend to prevent users from running even when the situation is perfectly safe. [5,6] In our system, users must control their own sharing. The availability of semaphores provides users with a tool for controlling sharing by means of an algorithm of their own choosing. If the algorithm does not work correctly, only the group of users would be affected.

There still remains the problem of system data structures which are also available to users; for example, the dictionaries and the queues. If such a structure is accessed incorrectly, all users will suffer. Users are expected to access these structures only through the system-provided procedures which use semaphores correctly and insure that no access errors occur.

13

MANAGEMENT OF INPUT/OUTPUT DEVICES

All input/output devices are managed similarly. The system retains ownership of the devices; users are given access for well-defined and <u>limited</u> transactions, which insures that deadlock cannot occur. The definition of a transaction is different for teletypes than for the other devices. This section discusses how devices are managed; the next section explains how levels of abstraction were used to accomplish the management.

Teletypes

Our system is on-line and interactive; and therefore, every user needs a teletype. One solution is to assign each user a teletype which he alone can use. This requires one teletype to be designated the "operator's console" and handled differently, since the system occasionally needs a teletype to announce special conditions and errors. Furthermore, it would not be natural for one user to send a message to another user's teletype or for the system to do so.

The solution chosen defines the teletype on which a user starts running as his "preferred" teletype. He can reference this device symbolically and will ordinarily use it, but he is not constrained to do so and is unable to prevent others from using it. A teletype may be accessed without interruption long enough to write and then read one line. This insures that the standard use of teletypes, which is the user responding to the program (with a command) when told to do so, can occur without interference from other users.

Other Devices

The card reader, the printer, and the two magnetic tapes are all in high demand. Because the disk is small, symbolic data can be maintained in core and disk storage for only a limited time. The card reader and tapes provide the only reasonable method for entering large amounts of symbolic data; only on tape can edited symbolic data be saved and then retrieved later. The printer provides the only reasonable method for listing symbolic data. Users will probably require lengthy access to several of these devices each time they run. All users will benefit if the devices are handled efficiently.

If users can own these devices, there is no guarantee that the devices would be run efficiently. When the system retains control, we can insure that the devices will be run efficiently and kept busy as long as there is work to do. For user convenience, the devices may be accessed without interruption for a fairly long time; for example, long enough to list the assembly of a user's procedure or read the cards in a user's card deck.

LEVELS OF ABSTRACTION

The system performs device management by providing the user with virtual devices which are quite different from real devices. This is accomplished through several levels of abstraction, which are spread over several processes (see Figure 4).

The lowest level, which the microprogram supplies, provides devices which have no real-time constraints but which require core buffers. The first level supplied by software is made up of the Controllers, one for each device. Each Controller is an independent process (although one reentrant procedure may serve as several Controllers). The virtual devices supported by the Controllers have the following characteristics:

    (1)   Only a few types of transfers are possible; for example, exactly one card may be read.

    (2)   The buffers for the transfer are located in segments.

    (3)   A transfer on the device may be requested at any time, regardless of whether the device is currently in use. It will be performed when the device is available.

    (4)   The completion of the transfer is signalled by the performance of a V on a semaphore specified by the transfer request.

Above the Controller level, teletypes are handled quite differently from the other devices.

Teletypes

A single reentrant procedure, the Teletype Requester, handles teletypes. Primarily, it provides an interface between the user and the Controllers; it runs on the user's virtual machine.

Other Devices

The card reader, printer, and tapes are accessed for much larger amounts of data than are accepted by the Controllers. The Drivers support this abstraction. Each Driver handles one device and _s an independent process. The characteristics supported are:

    (1)   The device is accessed a segment at a time. The Driver breaks the segment into buffers acceptable to the Controller.

Level 6. The user.

Level 5. The Requesters provide an interface between the users and the Drivers.

Level 4. The Drivers support virtual devices which are accessed a segment at a time.

Level 2. The Teletype Requester provides an interface between users (and higher level system procedures) and the Teletype Controllers.

Level 1. The Controllers support virtual devices with buffers in segments; requests for transfers are handled one at a time.

Level 0. The microprogrammed channel supports virtual devices with no time constraints and core buffers.
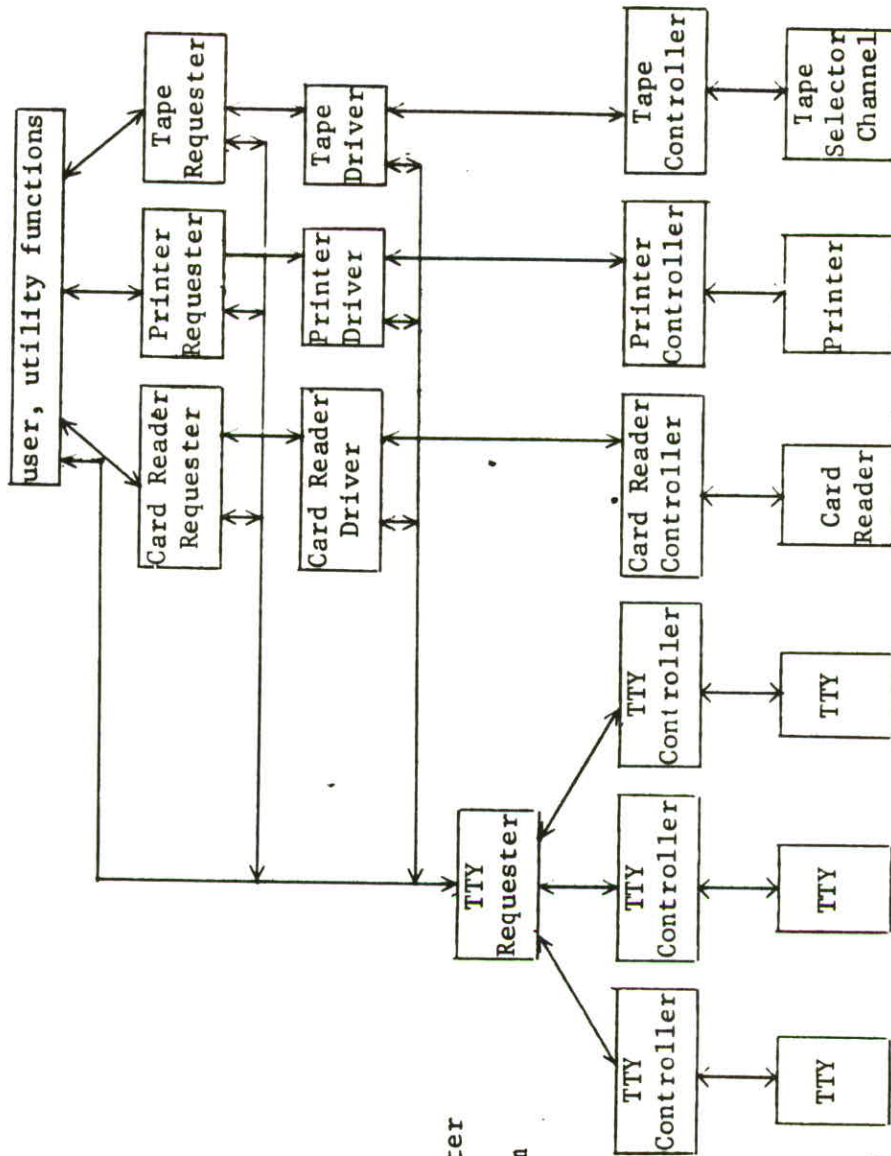
Figure 4. Levels of Abstraction Used in Resource Management

16

(2) The Driver defines the type of synchronization required.
The Card Reader Driver builds segments containing the images
of card decks whenever there are cards in the hopper; user
synchronization is required only to obtain the completed
segments. The Printer Driver requires notification that a
segment should be printed; no synchronization with the com-
pletion of printing is required. The Tape Driver requires
notification to read or write a segment on a tape; the user
must wait for completion.

Above each Driver is a Requester which primarily provides an
interface between the user and the Driver and runs on the user's vir-
tual machine. These Requesters are more elaborate than the Teletype
Requester. For example, the Printer Requester helps the user build
printable segments, while the Tape Requester is an interactive pro-
cedure which reads and writes segments on tape on user command.

SECTION V

CONCLUSIONS

The Venus Operating System has been supporting multiple users for several months. We estimate six man-years were required to design and implement the system on the Venus machine;[3] an additional two to three man-years were spent building the machine. This is a surprisingly brief time to develop such a system. We feel this indicates that the architecture of the Venus machine is suitable for supporting the programming of complex software and, in fact, reduces the difficulties encountered in building such software.

In addition, the use of levels of abstraction proved a valuable tool because it provided a way of thinking about the design with clarity and precision. Errors were uncovered while putting the system together, but the rule about ownership of resources was for the most part faithfully followed, minimizing errors resulting from interactions of parts of the system. Such errors as were discovered resulted from: errors internal to some piece of the system which was not fully checked out and errors traced to breaking the rule about resources. An added benefit of levels of abstraction is the ease with which the system can be modified; its design is clearly stratified so that the effects of proposed modifications are plainly visible.

---

[3]This includes utility functions (Assembler, Editor, and debugging aids) in addition to operating system functions.

APPENDIX

THE USER ENVIRONMENT

A description of the user environment is included to give the
reader an idea of what it is like to run under the operating system.
The user views the system in two ways. First, he must write his pro-
grams; for this he is interested in available data structures and sys-
tem procedures. Then he is interested in running and debugging his
programs.

Programs running under Venus use segments for all storage - pro-
cedures, data, and push-down stacks. Because segments are available
by external name through dictionaries, users can readily share data
and procedures. Users may also synchronize with other users through
queues and semaphores.

The user runs on-line and interactively. He starts up by typing
a command on a teletype and is assigned a virtual machine under the
control of an interactive system procedure called the Loader. The
Loader recognizes many commands; the most important is the "E" com-
mand, which passes control to a specified system, utility, or user
procedure. Important system and utility procedures are the Assembler,
Editor, Tape Requester, and Debugger.

The user submits card decks to be read prior to his run; he may
access all decks through a command to the Loader. The names of the
segments containing the card deck images are entered in a dictionary
associated with the user. The user may read segments from tape; the
names of these segments are also entered into his dictionary, where
they may be assembled or edited on command. Binding of intersegment
references occurs on completion of assembly.

The user executes procedures by giving the "E" command. Execu-
tion can occur with or without debugging; no change to the procedures
being run is required. Debugging is handled by an interactive system
procedure which runs before the execution of every instruction and
can be used to stop execution at a specified "breakpoint." A dialogue
with the user then commences in which the contents of a segment or his
work area may be displayed and modified, a new breakpoint specified,
and control returned to the interrupted procedure or the Loader.

The system also provides interactive interrupt procedures which
run as the result of exceptional conditions; for example, a stack under-
flow or overflow. They make use of a subset of the debugging commands,
permitting the user to discover the reason for the error, restore his
data, and return to the Loader.

19

When the user has finished running, he saves his symbolic data on tape and then informs the Loader, which releases his virtual machine and destroys his symbolic data. His checked-out programs may be entered in the system and become accessible to others through dictionaries.

REFERENCES

1.   E. W. Dijkstra, "The Structure of the 'THE' - Multiprogramming System," Communications of the ACM, 11, 5, May 1968, 341-346.

2.   B. J. Huberman, Principles of Operation of the Venus Microprogram, The MITRE Corporation, MTR 1843 (ESD-TR-70-198), F19(628)-71-C-0002, Bedford, Massachusetts, May 1970.

3.   F. J. Corbato and V. A. Vyssotsky, "Introduction and Overview of the Multics Systems," Proceedings AFIPS 1965 Fall Joint Computer Conference, 27, Part 1, Spartan Books, New York, 185-196.

4.   J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multi-programmed Computations," Communications of the ACM, 9, 3, March 1966, 143-155.

5.   A. N. Habermann, "Prevention of System Deadlocks," Communications of the ACM, 12, 7, July 1969, 373-377, 385.

6.   R. C. Holt, "Comments on Prevention of System Deadlocks," Communications of the ACM, 14, 1, January 1971, 36-38.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*	2a. REPORT SECURITY CLASSIFICATION
The MITRE Corporation P. O. Box 208 Bedford, Massachusetts 01730	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

The Design of the VENUS Operating System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

5. AUTHOR(S) *(First name, middle initial, last name)*

Barbara H. Liskov

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
OCTOBER 1971	27	6

8a. CONTRACT OR GRANT NO. F19(628)-71-C-0002	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 6710	ESD-TR-71-368
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	MTR-2150

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts 01730

13. ABSTRACT

The Venus Operating System is an experimental multiprogramming system which supports five or six concurrent users on a small computer. The system is defined by a combination of microprograms and software. This paper describes the development of the system, with particular emphasis on the principles which guided the design.

DD FORM 1473
1 NOV 65

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
COMPUTER SOFTWARE						
MICROPROGRAMMING						
VENUS						
VENUS MICROPROGRAM						