SU-326-P.39-33

A VIRTUAL INPUT/OUTPUT SYSTEM FOR THE STANFORD EMMY V-ACCESS

by

Jerry Huck

Technical Note No. 144

September 1978
Revised May 1979

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305

The work described herein was supported in part by the Department of Energy under Contract EY-76-S-03-0326-PA 39.

SU-326-P.39-33

Computer Systems Laboratory
Departments of Electrical Engineering and Computer Sciences
Stanford University
Stanford, CA 94306

Technical Note No. 144

September 1978
Revised May 1979

A VIRTUAL INPUT/OUTPUT SYSTEM FOR THE STANFORD EMMY V-ACCESS

by

Jerry Huck

ABSTRACT

This report describes the system of virtual I/O support for the Stanford Emmy. A PDP11/05 controls a pool of peripherals, and communicates with the Emmy processor through a pair of mailboxes. By issuing appropriate requests practically any peripheral can be emulated. Performance for single byte requests exceed 1000 bytes/sec.

The work described herein was supported in part by the Department of Energy under contract EY-76-S-03-0326-PA 39.

Table of Contents

Table of Contents (cont.)

1 Introduction

The emulation or interpretation of input/output operations for current architectures is extremely difficult due to the ad hoc nature of their design. Many devices are timing dependent or have maintenance modes unknown to users (and in some cases undocumented). Since it was not our intention to provide a connector to which the image machine's I/O device could be connected; a system of simulated devices was developed to replace the real hardware.

For this purpose the "match"[4] interface was designed to connect the (unique) Emmy processor bus[1-2] to the Unibus. The Unibus is the center of a system of devices that will provide virtual I/O support to the Emmy processor. The control for the I/O system is a PDP 11/05 processor executing Bell Laboratories' Mini-Unix operation system[3]. Mini-Unix is a scaled down version of Unix capable of running without memory management. A 67Mbyte disk drive, tape drive, and five asynchronous serial communication lines are connected to the Unibus.

A virtual device is a linear space of bytes and allows three operations on it: read, write, and seek. For example, a teletype keyboard is modeled as a read only device. Whereas a disk requires mapping the drive, head, cylinder, and sector into a linear space and allows read, write, and seek operations. Virtual devices are mapped into the Mini-Unix system as files. The Mini-Unix file system makes no distinction between ordinary files (maintained by the system on disk) and special files that are the physical devices connected to Mini-Unix. This lack of distinction between files allows complete transparency of the actual device being accessed. Therefore a virtual disk could be "assigned" to an ordinary file, say /tmp/dsk1, and go through the normal file structure; or be "assigned" to a special file, say /dev/si3, which is one of the system scratch disks.

The communication between the Emmy system and the Mini-Unix system is controlled through a pair of mailboxes. These mailboxes are loaded with requests and the receiving system is signaled that the mail is present. One mailbox is for Emmy to Unix requests and the other is for Unix to Emmy requests or acknowledgments. Executing on the Mini-Unix system is the _v-access_ program which processes the incoming and outgoing mailbox requests. This program is divided into three main sections. The first, _pmail_, processes the incoming mail and generates appropriate outgoing mail. The second, _pcontrol_, handles the configuration and status of the active virtual devices, along with load and dump control of the Emmy system. The third section is a simple vaccess debug facility for initial checkout of emulator I/O.

2 Mailbox protocols

The mailboxes are 16 byte blocks located at the low part of the operating system in unused interrupt vector locations. The mailbox is divided into various fields each 2 bytes long. The structure of the mailbox (from C) is shown in figure 1.

The Emmy to Unix mailbox is loaded by the Emmy processor and signals the Mini-Unix system by setting the flag member to a non-zero value. The Emmy processor detects the receipt of the mail when the

```
struct mbox
        {
        int command;
        int device;
        int bufadr[2];
        int count;
        int flag;
        int unassigned[2];
        } E_TO_UMBOX, U_TO_EMBOX;
```

Figure 1: Mailbox structure

mail processing procedure (pmail) clears the flag.

The Unix to Emmy mailbox is loaded by "pmail", and the Emmy processor is signaled by an interrupt (vector = 50 hex). The pmail procedure detects the receipt of the mail when the Emmy processor clears the flag member. The mailbox protocols are not symmetric because the Mini-Unix system does not allow fast enough latency between fielding an interrupt and signaling a user program (in this case the "pmail" procedure). Also there is no overlap of Mini-Unix system requests made by "pmail" and consequently a test loop gives the smallest latency.

The following four procedures implement the above protocols for putmail and getmail operations in each system.

1.) putmail for Emmy processor.

```
putmail( contents );
    begin
    while E_TO_UMBOX.flag is not equal to 0 do
            pause a short time;
    move contents to E_TO_UMBOX;
    E_TO_UMBOX.flag := 1;
    end;
```

2.) putmail for Mini-Unix system.

```
putmail( contents );
    begin
    while U_TO_EMBOX.flag is not equal to 0 do
            pause a short time;
    move contents to U_TO_EMBOX;
    U_TO_EMBOX.flag := 1;
    send interrupt to Emmy;
    end;
```

3.) getmail operation for Mini-Unix system

```
/* procedure that examines the Emmy_to_Unix mailbox and returns a 1
   if the mailbox contains new mail along with that mail, and a 0 if
   empty.
 */
   getmail( tmpbox )
       begin
       if E_TO_UMBOX.flag is equal to 0 then return( 0 );
       /* mail exists */
       move contents of E_TO_UMBOX to tmpbox;
       E_TO_UMBOX.flag := 0;
       return( 1 );
       end;
```

4.) getmail operation for Emmy processor.
note: This procedure would be called directly by the interrupt.

```
getmail()
    begin
    process the Unix to Emmy request;
    U_TO_EMBOX.flag := 0;
    end;
```

The E_TO_UMBOX is located at unibus addresses 110-126 (octal) and Emmy bus byte addresses F80048-F80056 (hex). The U_TO_EMBOX is located at unibus addresses 250-266 (octal) and Emmy bus byte addresses F800A8-F800B6 (hex).

3 Files and Virtual Devices

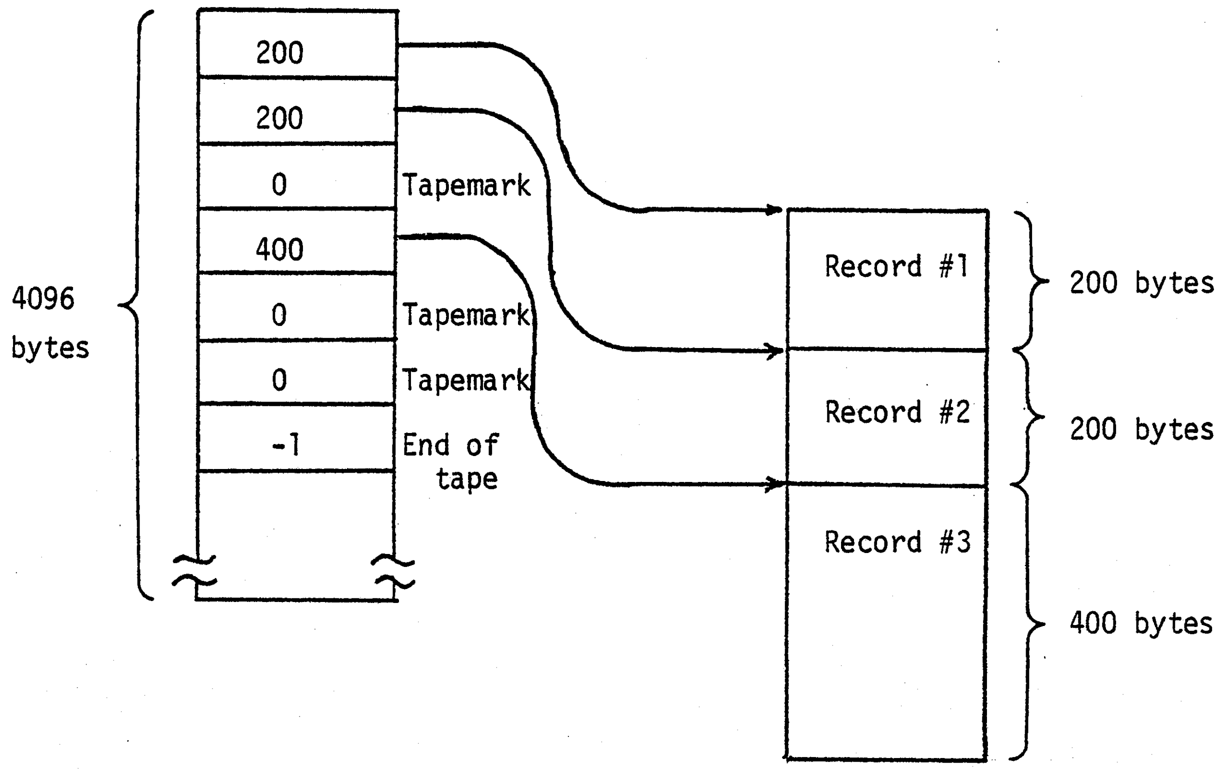
V-access provides a very simple interface to the Emmy processor. I/O requirements are neatly packaged into mailbox messages and acknowledgments clearly signal the completion of the request. Since the transfers occur across a system bus, the error rates are negligible and no error detection system exists. Constructing a disk emulation is easily mapped into the V-access interface. Blocks are mapped 1 to 1 into some counterpart file in Mini-Unix. For each emulated disk action a similar action is requested of V-access. For example a 5 block disk read is emulated by seeking first to the start of the transfer and requesting 5 blocks. This procedure becomes more difficult when more unusual features are required. To illustrate this consider the program, say of a 360, that reformats the sectoring information dynamically. The emulation of that function would require some scheme to include sector information for various tracks and necessitate multiple requests to determine the location of a given disk address. Another aspect of I/O devices that a simple read, write, and seek operations do not adequately handle is magnetic tape functions. Magtapes store implicit length information into each record. So searching for the 10th record is not a simple seek to 10 times the record size. For these reasons and more (mostly efficiency), virtual devices are supported in three different fashions.

The first is the already mentioned block reads, write, and seeks. Second is variable record "tape" access. When a device is declared to be a tape device, variable length records can be read or written. Also tapemarks can be written and sensed for allowing easy emulation of simple tape drives.

The layout of a file to be used as a tape device differs from the simple linear vector. Tape files are organized with a 4K byte record length list followed by the actual records. Positive non-zero entries indicate the size of the record. Zero entries signal tapemarks, while a negative entry marks the end of file. The length list approach was motivated by the need to minimize the number of system calls to Mini-Unix and speed the operation of spacing to the next tapemark. This allows only 2048 records per tape device. While this may seem small, it will exceed the Mini-Unix file size limit when the average record length is greater than 512 bytes. Figure 3.1 illustrates the data structure.

The third device type was motivated mostly for efficiency. Character devices expect byte (or word) at a time transfers. By anticipating this behavior data can be prefetched and buffered. This does require certain restrictions. It is necessary to request either input or output& Also seeking is not allowed. The obvious examples of devices suited to this device class are paper tape readers and punches.

One device that exists in its own class is the terminal keyboard. Unique to this device is the lack of a end of file. Also reads to the keyboard may never be fulfilled or may already have been fulfilled. The special treatment of the keyboard device is explained when relevant.



Note: This file would appear in Mini-Unix with a 4896 byte length.

Figure 3.1: Internal data structure of tape files.

4 Mailbox Requests and the pmail procedure

All virtual devices are assigned a 16 bit (high bit=0) device number identifying the device to the mail processing procedure, "pmail". This number is arbitrarily chosen by the emulator or interpreter. A request is identified by the associated request number, which the "pmail" procedure uses to index a branch table.

The Emmy processor can view the mailbox as a call to an asynchronous processor that will notify the initiator when completed. The "pmail" procedure processes the requests and generates an acknowledgment upon completion. The acknowledgment indicates possible errors or returns data. The first two fields of the mailbox always hold the request number and a unique device number. Field 3 is double length and varies in function dependent on the request. Typically it contains either a 24 bit address or immediate data. It is desigated as the "bufadr" field. The fourth field, designated as the "count" field, likewise varies in function. It is mostly used to hold and return transfer counts. The last field is used to prevent overwriting previous mail and is controlled by the previously discussed protocols. Designated the "flag" field, it always is set by one processor and cleared by the other, preventing any deadlock problems.

There is an incompatibility between the addressing conventions of the Emmy system and the Unibus. In the Emmy system the left most byte of a word or halfword is designated "byte zero", whereas the Unibus designates the right byte "byte zero". The "pmail" procedure transfers data two different ways. Block devices transfer data as 16 bit quantities to/from main memory and 32 bit quantities to/from control store. Figure 4.1 illustrates the mapping of the addresses between Unix, main memory and control store for the various combinations of address and byte counts. It is sometimes necessary for the user to swap bytes in order for data to appear correctly. Data for tape devices is swapped before transfer to the Emmy system and after transfer out of the Emmy system. Tape devices are further restricted by requiring that transfers begin on halfword boundries and request an even number of bytes.

Transfer Between Emmy and Unix

Unix File

byte 0	a
byte 1	b
byte 2	c
byte 3	d
byte 4	e
byte 5	f
byte 6	g
byte 7	h

word 0	b	a
word 1	d	c
word 2	f	e
word 3	h	g

Emmy Control Store

fulword	0	d	c	b	a
fulword	1	h	g	f	e

Emmy Main Memory

Case 1: Even Address (0), Even Count (8)

byte 0	b
byte 1	a
byte 2	d
byte 3	c
byte 4	f
byte 5	e
byte 6	h
byte 7	g

halfword 0	b	a
halfword 1	d	c
halfword 2	f	e
halfword 3	h	g

Emmy Main Memory

Case 3: Even Address (0), Odd Count (7)

byte 0	b
byte 1	a
byte 2	d
byte 3	c
byte 4	f
byte 5	e
byte 6	XX
byte 7	g

halfword 0	b	a
halfword 1	d	c
halfword 2	f	e
halfword 3	XX	g

Emmy Main Memory

Case 2: Odd Address (1), Even Count (8)

byte 0	X
byte 1	b
byte 2	a
byte 3	d
byte 4	c
byte 5	f
byte 6	e
byte 7	h
byte 8	g
byte 9	X

halfword 0	X	b
halfword 1	a	d
halfword 2	c	f
halfword 3	e	h

Emmy Main Memory

Case 4: Odd Address (1), Odd Count (7)

byte 0	X
byte 1	b
byte 2	a
byte 3	d
byte 4	c
byte 5	f
byte 6	e
byte 7	XX
byte 8	g
byte 9	X

halfword 0	X	b
halfword 1	a	d
halfword 2	c	f
halfword 3	e	XX

X - byte is preserved during read operation
XX - byte is destroyed during read operation

Figure 4.1: Example Unix/Emmy transfers to illustrate address mapping of data.

Virtual devices must be accessed consistantly. Tape devices may only be manipulated with the tape requests. Likewise character devices may only be accessed via IMMEDIATE requests.

4.1 Emmy to Unix Requests

Emmy to Unix requests initiate data transfers and file control for the virtual devices. The valid requests with the mailbox field arguments are shown in Table 1.

Name	Number	Bufadr field	Count field
READ	0	Emmybus byte address	byte count
WRITE	1	Emmybus byte address	byte count
SEEK	2	32 bit unsigned offset	bias
READ_IMMEDIATE	3	unused	byte count
WRITE_IMMEDIATE	4	4 byte immediate data	byte count
ABORT	5	unused	unused
RESET_ALL_DEVICES	6	unused	unused
NULL	7-10	unused	unused
TAPEREAD	11	Emmybus byte address	byte count
TAPEWRITE	12	Emmybus byte address	byte count
SPTM	13	unused	direction
WRTM	14	unused	unused
REWIND	15	unused	unused
REWRITE	16	unused	unused
EXIT	17	unused	unused

Table 4.1: Emmy to Unix requests

4.1.1 READ Request

The READ request initiates a transfer of the number of bytes specified in the "count" field from the file indicated by the given device number. The transfer starts at the specified Emmybus byte address. The "count" field must have the high bit cleared. This allows up to 32,766 bytes per request. The "pmail" procedure breaks the request into multiple 8192 byte transfers. If the request size is greater than 8192 then the "count" and "address" fields must be even. Control store request are further restricted where "count" and "address" fields must be a multiple of 4.

4.1.2 WRITE Request

The WRITE request initiates a transfer of the number of bytes specified in the "count" field. The data, starting at the given Emmybus byte address, is transfered to the file indicted by the device number. The specified count must have the high bit cleared. The "pmail" procedure breaks the request into multiple 8192 byte transfers. If the request size is greater than 8192 then the "count" and "address" fields must be even. Control store request are further restricted where "count" and "address" fields must be a multiple of 4. The "write request" is acknowledged after all of the data has been transfered out of the Emmy system. This gives a reasonable amount of overlap, since the acknowledgment will precede the last system write to Mini-Unix.

4.1.3 SEEK Request

The SEEK request updates the current file pointer by the amount in the "bufadr" field. The bias located in the "count" field indicates the type of update desired. The "file pointer" is a pointer to the location in the virtual device's linear space where the next transfer will take place. This pointer is updated after each transfer. The bias is interpreted as indicated by Table 2.

Bias	Meaning	File pointer (fp)
0	absolute	fp := bufadr field
negative	negative offset	fp := fp - bufadr field
positive	positive offset	fp := fp + bufadr field

Table 2: Bias Interpretation for the SEEK Request.

The bias is interpreted a 16 bit two's complement quantity. The SEEK request explicitly moves the "file pointer" in a file. READ and WRITE requests update the "file pointer" implicitly by the size of the transfer. If the Mini-Unix file which was referenced does not allow seeks (e.g. the CRT) then an error message is printed on the user's terminal.

4.1.4 READ IMMEDIATE Request

The READ_IMMEDIATE request transfers 1 to 4 bytes of data in the request acknowledgment. This request is used for devices that require small transfers to avoid the overhead of buffering the data. If the Mini-Unix device specified is a "tty", then the request is recorded and only acknowledged when the data becomes available. Only requests of 1 byte at a time are allowed from the user's keyboard. If the device number is unassigned the request is ignored. An appropriate error message is printed on the user's terminal. If an end of file is reached on the device, then the request is ignored. An error message indicating the device is written to the user's terminal.

4.1.5 WRITE_IMMEDIATE Request

The WRITE_IMMEDIATE request requests that the number of bytes (maximum of four) indicated by the "count" field be transfered from the "bufadr" field to the specified virtual device. As was true for the READ_IMMEDIATE request this request is primarily used for small transfers to save buffering overhead and complexity. The ordering of the "bufadr" field is:

	Unibus loc.	Emmybus loc.
bufadr[0] byte2 byte1	0114	X'F8004C'
bufadr[1] byte4 byte3	0116	X'F8004E'

If the device number is unassigned the request is ignored, and an error message is written to the user's terminal.

4.1.6 ABORT Request

The ABORT request terminates all previous requests for the specified virtual device. It only assures that, after the ABORT is acknowledged, further acknowledgments will be for requests made after the ABORT. Between the sending and the acknowledgment of the ABORT request, previous requests could finish and be acknowledged.

4.1.7 RESET ALL DEVICES Request

The RESET_ALL_DEVICES request is the only request that doesn't specify a particular device. Instead it clears all requests in progress and guaranties their termination when the RESET_ALL_DEVICES is acknowledged. This request is used mostly to emulate bus wide clears and reset instructions.

4.1.8 TAPEREAD Request

The TAPEREAD request initiates a transfer of the next record from the file indicated by the given device number. The transfer starts at the specified Emmybus byte address (must be on 2 byte boundry). If the next record is smaller or equal to the specified "count" field, then that number of bytes is transferred. All byte counts must be even and have the high bit cleared. When the next record is larger, only the number of bytes specified will be transferred. The rest of the record is ignored. If a tapemark or end of tape is encountered no data is transferred.

4.1.9 TAPEWRITE Request

The TAPEWRITE request initiates a transfer of the specified number of bytes. The data, starting at the given Emmybus byte address, is transferred to the file indicated by the device number. All addresses and counts must be even. A write operation creates a new "end of tape" at the end of the record whether data existed past the current record or not.

4.1.10 SPTM Request

The SPTM request positions the file pointer (see section 4.1.3) to the next tape mark. The direction is indicated by the "count" field. When the "count" field is a -1, then the file pointer is spaced backwards and positioned to read the first encountered tape mark (or possibly the beginning of tape). Any other value in the "count" field positions the file pointer foward in the file following the next tape mark. Notice that forward spacing positions _after_ the tapemark and backward spacing _before_ the tapemark. Forward spacing past the end of file is ignored and the file pointer is positioned after the last record.

4.1.11 WRTM Request

The WRTM request writes a null length record which will subsequently be interpreted as a tape mark. The "count" and "bufadr" fields are unused.

4.1.12 REWIND Request

THe REWIND request positions the file pointer to the beginning of the tape. The "count" and "bufadr" fields are unused. This is equivalent but not interchangeable (file types are different) with an absolute SEEK request to zero.

4.1.13 REWRITE Request

The REWRITE request executes the PASCAL REWRITE function to reset the file pointer to zero and truncate the file to zero length.

4.1.14 EXIT Request

The EXIT request deassigns all active devices and returns to the envoker of V-access (probably the SHELL). There is no acknowledgment for this request except for the possible DEVICE_NOT_READY requests due to deassignments. The "device" field is used to return a status byte to the envoker of V-access.

4.2 Unix to Emmy Requests

Unix to Emmy requests are typically acknowledgments to earlier requests, or control requests to the emulator. The valid requests with the mailbox field arguments are shown in Table 3.

The processing of Emmy to Unix requests begins with verification of the virtual device number. If it is unassigned, an error message is written to the user's terminal. All acknowledgments return the originally used virtual device number to identify the source of the acknowledgment.

Request Name	Number	Bufadr	Count
READ_FINISH	1	unused	number of bytes read
WRITE_FINISH	1	unused	number of bytes written
SEEK_FINISH	2	unused	error flag
READ_IMMEDIATE_FINISH	3	immediate data	number of bytes read
WRITE_IMMEDIATE_FINISH	4	unused	number of bytes written
ABORT_FINISH	5	unused	error flag
RESET	6	unused	unused
BREAK	7	unused	unused
DEVICE_READY	8	unused	unused
RESET_ALL_FINISH	9	unused	unused
DEVICE_NOT_READY	10	unused	unused
TAPEREAD_FINISH	11	unused	number of byes read
TAPEWRITE_FINISH	12	unused	number or bytes written
SPTM_FINISH	13	unused	error flag
WRTM_FINISH	14	unused	error flag
REWIND_FINISH	15	unused	error flag

Table 3: Unix to Emmy Requests

4.2.1 READ FINISH Request

The READ_FINISH request is generated by the "pmail" procedure to acknowledge the completion of a previous READ request. The "count" field indicates the actual number of bytes transferred. If the original request specified an unassigned virtual device number the "count" field is set to zero, indicating an error. The "count" field is also set to zero when reading a device initially assigned to be only accessed with READ_IMMEDIATE requests (discussed in section 4). In any event, a READ request is always acknowledged with a READ_FINISH. The "device" field will be unchanged.

4.2.2 WRITE FINISH Request

The WRITE_FINISH request is generated by the "pmail" procedure to acknowledge the completion of a previous WRITE request. The "count" field indicates the actual number of bytes transferred. If the original request specified an unassigned virtual device number the "count" field is set to zero, indicating an error. The "count" field is also set to zero when transferring to a device initially assigned to be only accessed with WRITE_IMMEDIATE requests (discussed in section 4). In any event a WRITE request is always acknowledged with a WRITE_FINISH. The "device" field will be unchanged.

4.2.3 SEEK FINISH Request

The SEEK_FINISH request is generated by the "pmail" procedure to acknowledge the completion of a previous SEEK request. If the original request specified an unassigned, "character", or "tape" device (described in section 4) then the "count" field is set to zero, indicating an error. Otherwise the SEEK is assumed successful and the "count" field is set to 1. If the location exceeds the bounds of the file, then this error is not reported and subsequent requests are unpredictable.

4.2.4 READ IMMEDIATE FINISH Request

The READ_IMMEDIATE_FINISH request both acknowledges the READ_IMMEDIATE request and includes the immediate data in the "bufadr" field. The format of the immediate data is the same as used in the

WRITE_IMMEDIATE request (section 4.1.5). For the device that is assigned to the user's keyboard this request acts differently. In essence the keyboard always assumes a character is desired and "pmail" sends a READ_IMMEDIATE_FINISH request whenever anything is typed. So a READ_IMMEDIATE request has no effect when the device is assigned to the user's keyboard. This is done because the typical terminal used in computer systems (teletypes and CRTs) will send a typed character whether or not is was requested.

4.2.5 WRITE IMMEDIATE FINISH Request

The WRITE_IMMEDIATE_FINISH request acknowledges the WRITE_IMMEDIATE Emmy to Unix request. "pmail" assumes the write operation will be successful and immediately sends this acknowledgment once the device number is found to be valid.

4.2.6 ABORT FINISH Request

The ABORT_FINISH request acknowledges the receipt of the ABORT request. Any other Emmy to Unix request in progress will be terminated for the specified device. If the specified device number was invalid the "count" field is set to zero to indicate the error, otherwise it is set to one.

4.2.7 RESET and BREAK Requests

There are two special requests which are initiated by the user. Their functions are not specified by the "pmail" procedure and are left to the needs of the emulation.

It is intended that they be used as machine level interrupts either at a console or system reset level. One typical emulation used the break request to switch to console mode and the reset to force reinitializing itself.

The RESET and BREAK device numbers are included in the "device" field to allow device number parsing by the emulation or interpretation. The RESET request is generated by typing the arbitrarily assigned "reset" character on the user terminal. Likewise the "break" request is generated by typing the "break" character on the terminal. The "break" and "reset" devices are initially unassigned and set to zero.

4.2.8 DEVICE READY and DEVICE NOT READY Requests

The DEVICE_READY request is generated by the procedure that controls the assignment of virtual device numbers and the unix file names. When the device number is assigned to a unix file, the DEVICE_READY request is sent with the indicated device number to be used or discarded as the emulation sees fit.

The DEVICE_NOT_READY request signals the disassociation of a device number and a unix file. For example, the nova emulation uses these two requests to set and clear the device ready bits of its virtual disk platters.

4.2.9 RESET ALL FINISH Request

The RESET_ALL_FINISH request acknowledges the receipt of the RESET_ALL_DEVICES request. The "device" field is set to the "reset" device if one is assigned, otherwise the most recent value is used.

4.2.10 TAPEREAD FINISH Request

This request signals the completion of a previous TAPEREAD request. The "count" field indicates the actual number of bytes transferred. If the "count" field is zero, then a tape mark was encountered. When a -1 is returned an end of file was sensed. The "device" field indicates which device is being acknowledged. Unassigned device numbers and non-tape devices in the initial TAPEREAD request will be acknowledged with a return in the "count" field of zero, and an error message is written on the user's terminal.

4.2.11 TAPEWRITE FINISH Request

The TAPEWRITE_FINISH request signals the completion of a previous TAPE_WRITE request. The "count" field indicates the actual number of bytes transferred. "pmail" generally assumes the write operation will be successful once the device number has been validated.

4.2.12 SPTM FINISH Request

The SPTM_FINISH request acknowledges a previous SPTM request. The spacing is assumed successful once the device number is validated. The "count" field is set to +1 when valid and zero on any error.

4.2.13 WRTM FINISH Request

The WRTM_FINISH request acknowledges a previous WRTM request. The request is assumed successful once the device number is validated. The "count" field is set to +1 when valid and zero on any error.

4.2.14 REWIND FINISH Request

The REWIND_FINISH request acknowledges a previous REWIND request. The rewind is assumed successful once the device number is validated. The "count" field is set to +1 when valid and zero on any error.

4.2.15 REWRITE FINISH Request

THe REWRITE_FINISH request acknowledges the completion of a REWRITE request. Once the device number is verified the "count" field is set to one indicating success, otherwise an error is signalled by the "count" set to zero.

5 Device assignment and the pcontrol procedure.

The association of unix file names with the virtual device numbers is handled by the "pcontrol" procedure. This association is a strict one to one mapping. Typing the special sequence "control b?" to the "pmail" procedure enters "pcontrol", where "?" is any character other than a "control b". The second character now becomes the first character of the command. If this second character had been a "control b", then "pmail" would interprete the two "control b"'s as a single "control b" and generate a READ_IMMEDIATE_FINISH request (if the KEY-

BOARD is assigned).

The "pcontrol" procedure reads an entire line of text and executes the appropriate command handler. While "pcontrol" is executing no mail processing is done. All keywords used by "pcontrol" can be abbreviated to the shortest distinguishable string. The optional arguments are parsed in the same manner allowing abbreviations. Numerical arguments are interpreted as decimal numbers, if preceded by a zero, they are interpreted as octal numbers. The following sections discuss in detail each command.

5.1 Assign Command

The "assign" command associates the named unix file to the given virtual device number. The Mini-Unix file can be either a ordinary or special file. The command syntax is:

```
assign <device num> [to] {<file name>¦RESET¦BREAK¦KEYBOARD¦
        CRT¦NONE} [  [read¦write][tape] ¦ cread ¦ cwrite ]
```

The square brackets indicate optional arguments. The {...} requires that one argument be chosen. If RESET or BREAK is used instead of a Mini-Unix file name the RESET or BREAK device will be assigned to the given number.

The user's terminal is a somewhat unique device since it acts both as a display and keyboard device to the "pmail" procedure and as the controller for the virtual access system. So (rather than use the normal unix name /dev/tty?) two special names, KEYBOARD and CRT, are reserved for the user terminal's keyboard and display screen respectively. Different virtual device numbers must be used for the KEYBOARD and the CRT since the device number must uniquely define the Mini-Unix file it is assigned to.

If the device was already assigned "pcontrol" prompts the user to ask if the device number should be reassigned. Reassignment causes the current assignment to be terminated and the new file name assigned. If the first letter of the reply is "y", then the number is reassigned, otherwise the command is ignored. When the "NONE" option is used the device number is deassigned. Any further mail requests to that number will result in an error.

The file name can be qualified to indicate the type of access allowed. The qualifier "READ" and "WRITE" only permit read or write mail requests to the file respectively. This is only enforced at the Unix system level. This has the result that WRITE requests will be acknowledged without an error for devices assigned as "read" devices. It remains the responsibility of the user to recognize this problem. In order to reduce the number of costly system calls, a special qualifier "cread" or "cwrite" is used to notify the "pmail" procedure that this device will only be accessed via READ_IMMEDIATE or WRITE_IMMEDIATE mail requests. "pmail" will then use local buffers to speed the transfers. The default access is standard read or write access. Typically papertape readers and punches will be assigned with the "cread" and "cwrite" qualifiers. Whereas a disk uses the default access.

The last qualifier, "tape", signals that the file is to be a tape device. If it doesn't exist a null length tape file is created.

Many errors are reported by the "assign" command. Usually the entire command is ignored with no side effects. One exception is that an already assigned device number could be deassigned even when the command line is in error.

5.2 Reassign Command

The "reassign" command is syntactically the same as the assign command. Functionally the "reassign" command expects the specified device number to already be assigned and associates the number with the given file name. If the device number is unassigned, then "pcontrol" prompts the user if the device number should be newly assigned. A "y" as the first letter of the response assigns the device number otherwise the command is ignored. The command syntax is:

```
reassign <device num> [to] {<file name>|RESET|BREAK|KEYBOARD|
        CRT|NONE} [  [read|write][tape] | cread | cwrite ]
```

5.3 ! Command

The "!" command submits the text string that follows the "!" to the Mini-Unix shell program for execution.

5.4 Debug Command

The "debug" command invoces a preliminary debug routine to assist the of emulator I/O systems. This procedure scans the input keyboard and Emmy to Unix mailbox just as the "pmail" procedure does. But when mail is received the contents of the mailbox are only displayed on the terminal and the Emmy to Unix flag is cleared. The display is self-explanatory and contains all of the mailbox values.

When a key is struck, "debug" collects a line of data, in octal, to be sent as a Unix to Emmy request. The input syntax is:

```
<request> <device> <highadr> <lowadr> <count>
```

All five values must be typed. "debug" then reports the success or failure in trying to send the request. An unsuccessfull response is due to the Unix to Emmy mailbox flag still being set. This indicates the previous mail request was not read.

"debug" mode is exited by typing a request with the request number set to octal 40.

5.5 Break Command

The "break" command changes the break character to the specified character. The space, newline, erase, and kill characters are illegal and ignored. The character "N" is special and signifies no character. If "N" is used the break function will be turned off. The syntax is:

```
break [is] <break character|NONE>
```

5.6 Reset Command

The "reset" command changes the reset character to the specified character. The space, newline, erase, and kill characters are illegal and ignored. The character "N" is special and signifies no character. If "N" is used the reset function will be turned off. The syntax is:

```
reset [is] <reset character|NONE>
```

5.7 Status Command

The "status" command displays all the current information on the assigned device numbers. The device number is displayed in octal followed by the Mini-Unix file name it is assigned to. The next column indicates with a "t" if tracing is turned on. Following the trace flag is the character, mode, and tape indicator columns. A character device is signalled by a "c". The mode by "r","w", or "*" to indicate read, write, or update mode respectively. The tape indicator is set to "t" when true. The last field is the current location (in octal) of the file pointer for the device.

5.8 ^b Command

The "control b" command causes the "pcontrol" procedure to terminate and return to the "pmail" procedure.

5.9 Trace Command

The "trace" command forces all WRITE and WRITE_IMMEDIATE commands to the specified device number to be additionally displayed on the user's terminal. The optional "OFF" argument disables a previous trace command. The syntax is:

```
trace <device number> [OFF]
```

The device number specified must have been previously assigned to a Mini-Unix file (excluding the CRT).

5.10 Cd Command

The "cd" command acts equivalent to the unix "chdir" command. The current directory is changed to the text string that follows. "/etc/glob" is not used to interpret special characters, consequently the name must be spelled out exactly. The syntax is:

```
cd <directory name>
```

5.11 Kick Command

The "kick" command forces a one byte READ_IMMEDIATE Emmy to Unix command to be processed for the specified device number. This is most useful to restart a device, after a reassignment, when the emulator's "device start" came prior to the reassignment. The syntax is:

```
kick <device number>
```

5.12 Clear Command

The "clear" command clears the two mailbox flags. This is used to reset a hung mailbox.

5.13 Commands Command

The "commands" command redirects the input of command lines to the specified file name. The commands are echoed on the display and executed until an end of file is reached. Control then returns to the user's terminal. The syntax is:

```
commands [from] <file name>
```

5.14 : Command

The ":" command is a comment and causes no action. The text that appears after the ":" is ignored.

5.15 Stty Command

In order to understand the implications of the possible settings of the user tty it is necessary for the reader to study in the Unix manual the sections STTY(I) and STTY(II). Initially the "pmail" procedure sets the terminal modes to:

```
raw, anyp, -tabs, nl0, tab0, cr0, nl, -echo, -lcase.
```

This will very closely send characters and print characters as typed. The expansion of tabs is the single exception.

When "pcontrol" is entered, the terminal modes are restored to the values saved during initialization. The user may change and display the current modes used by "pmail" with the "stty" command. The syntax is the same as used by the Unix "stty" command. The effects for most settings are straight forward. "cooked" mode is a noteable exception. When a character is typed in cooked mode, vaccess requests an entire line from the Mini-Unix system. After the line is typed (with possible erase and kill processing), "pmail" processes the characters by either sending a burst of READ_IMMEDIATE_FINISH requests to the Emmy system; or ignores them if the KEYBOARD was unassigned. From the time the first character is typed untill the newline, vaccess does not process any incoming mail. Once the line has been disposed of, and before another character is typed, regular processing continues.

Using the KEYBOARD for block reads has some unpleasant side effects. Once the "pmail" procedure receives the READ request all "control b" processing is suspended. After the request was handled (and perhaps only partially so) "pmail" may pick up more characters that precede another READ request and throw them away or generate READ_IMMEDIATES that will be unexpected. All in all the terminal KEYBOARD should use the READ_IMMEDIATE requests to maintain data and program integrety. Of course the CRT is not effected by the above discussion and is equally enamored to WRITE or WRITE_IMMEDIATE requests for any setting of the terminal modes.

5.16 Exit Command

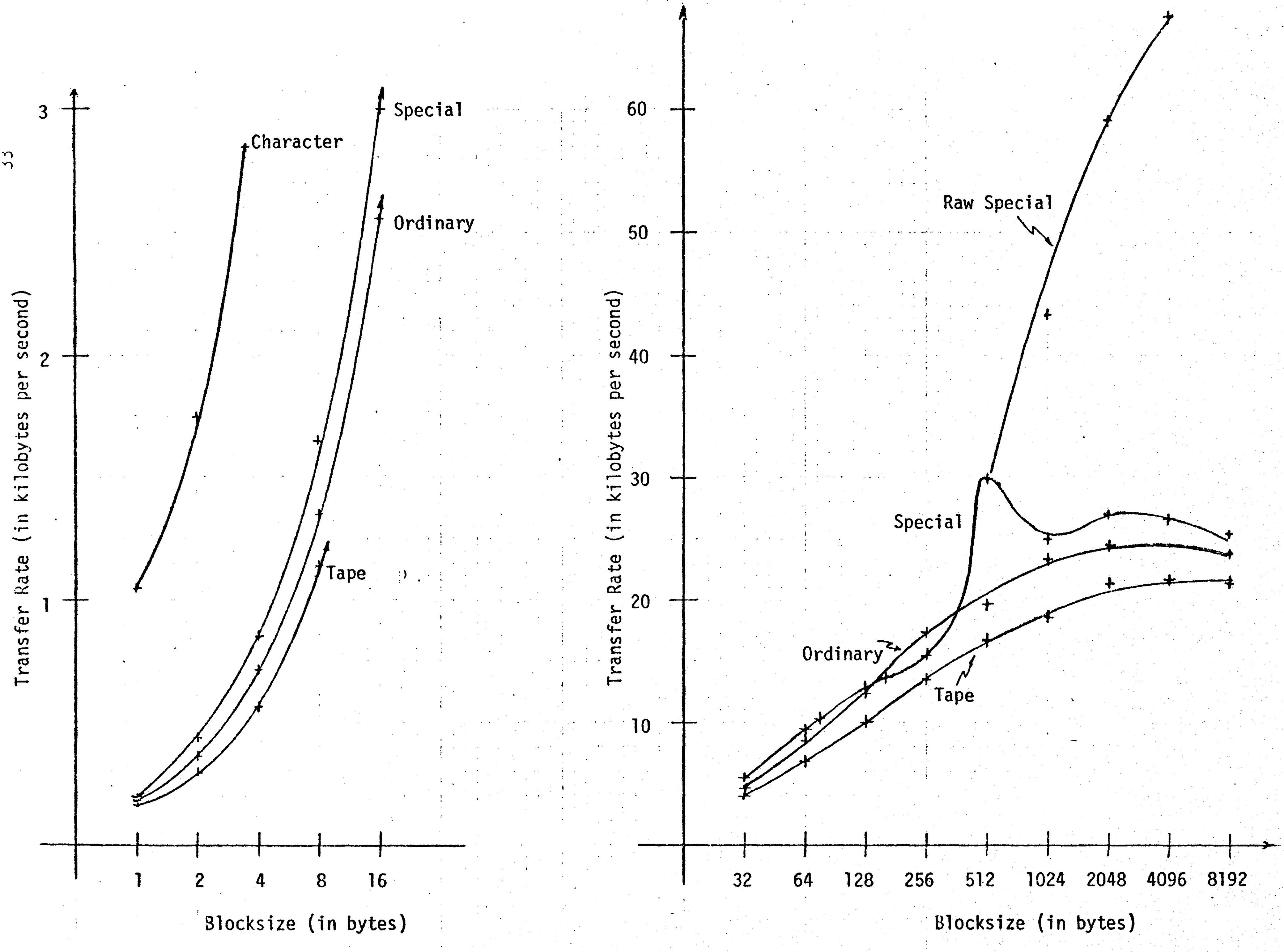
The "exit" command closes all files and terminates the vaccess program. The program interrupt (rubout character) will prompt the user if vaccess should exit. A yes response will exit in the same manner as the "exit" command.

6 Performance

The performance of an emulated I/O device is a function of many different factors, the most important of which is the "blocksize". Mailbox requests require a fixed amount of time to initially parse the request type and device assignment. Once a device is identified the transfer is started. The transfer time between the operating system and the Vaccess program is directly proportional to the blocksize. So, for small blocksizes the initial overhead dominates the total interaction time, while for the larger blocksizes the transfer rate is constant. Because of the buffering scheme used by the UNIX operating system, blocksizes above 512 bytes result in near constant transfer rates. There is one notable exception which involves the use of the "raw" device interface. Raw devices bypass the normal buffer mechanism and use direct disk transfers to the Vaccess program. For raw devices, the transfer rate is proportional to the blocksize. This provides much higher throughput when the blocksize exceeds 1024 bytes.

UNIX makes no distinction, logically, between the physical devices connected to it and its file system, but the transfer rates can vary dramatically. Ordinary files are organized into a file system and are maintained by the operating system. Ordinary files are the easiest to manipulate but are limited to about one megabyte in size (this is a Mini-UNIX restriction). Special files are the actual devices available on the system. Our current system divides the disk into two 16 megabyte scratch regions and two system regions used for swapping and the file system. Raw special files are special files that bypass the UNIX buffering scheme. They are restricted to complete block transfers (multiple of 512), but are ideal for disk emulations because of the considerable speed advantage.

Figure 6.1 shows the expected transfer rates for the various file types and device structures. Tape devices refer to ordinary assigned with the "tape" qualifier. The curve labelled "character" refers to the expected transfer rates of ordinary files assigned with the "cread" or "cwrite" qualifier. The ordinary, special, and raw special labelled curves refer to the rates when assigned as block devices. The curves were generated using blocksizes that are powers of two and the actual curve between these points may be lower because of the internal blocking used by the UNIX file system. Data for blocksizes corresponding to card images and print images (80 and 132 bytes respectively) fit very closely to the curve.



TRANSFER RATES FOR EMULATED I/O DEVICES

FIGURE 6.1

Appendix A: Mailbox Structure and Location

```
                                Unibus loc.  Emmy bus loc.

struct mbox
   {
   int command;             0110         X'F80048'
   int device;              0112         X'F8004A'
   int bufadr[2];           0114         X'F8004C'
   int count;               0120         X'F80050'
   int flag;                0122         X'F80052'
   int unassigned[2];       0124         X'F80054'
   } E_TO_UMBOX;

struct mbox
   {
   int command;             0250         X'F800A8'
   int device;              0252         X'F800AA'
   int bufadr[2];           0254         X'F800AC'
   int count;               0260         X'F800B0'
   int flag;                0262         X'F800B2'
   int unassigned[2];       0264         X'F800B4'
   } U_TO_EMBOX;
```

Appendix B: Emmy to Unix Request summary

Request		Mailbox Fields			
Name	Number	Device	Bufadr	Count	Pmail action
READ	0	Device number	Address	Byte count	Transfer 'count' bytes
WRITE	1	Device number	Address	Byte count	Transfer 'count' bytes
SEEK	2	Device number	32 bit offset	bias =0, absolute >0, positive <0, negative	Move file pointer by bias
READ_IMMEDIATE	3	Device number	unused	Byte count	Request data in acknowledgment
WRITE_IMMEDIATE	4	Device number	Immediate data	Byte count	Transfer 'count' bytes
ABORT	5	Device number	unused	unused	Terminate all active requests
RESET_ALL_DEVICES	6	unused	unused	unused	Terminate all active requests for all devices
NULL	7-10	unused	unused	unused	
TAPEREAD	11	Device number	Address	Byte count	Transfer "count" bytes
TAPEWRITE	12	Device number	Address	Byte count	Transfer "count" bytes
SPTM	13	Device number	unused	Direction = -1, Backward ≠ -1, Forward	Space to indicated Tapemark
WRTM	14	Device number	unused	unused	Write a tapemark
REWIND	15	Device number	unused	unused	Position at BOT
REWRITE	16	Device number	unused	unused	Position at start of file and truncate to zero length
EXIT	17	Return Code	unused	unused	Terminate the V-access program

Appendix C: Unix to Emmy Request Summary

Request		Mailbox Fields			
Name	Number	Device	Bufadr	Count	Pmail action
READ_FINISH	0	Device number	unused	Actual count	READ
WRITE_FINISH	1	Device number	unused	Actual count	WRITE
SEEK_FINISH	2	Device number	unused	Error flag *	SEEK
READ_IMMEDIATE_FINISH	3	Device number	Immediate data	Actual count	READ_IMMEDIATE or typed character
WRITE_IMMEDIATE_FINISH	4	Device number	unused	Original count	WRITE_IMMEDIATE
ABORT_FINISH	5	Device number	unused	Error flag *	ABORT
RESET	6	Reset device	unused	unused	'reset' character typed on keyboard
BREAK	7	Break device	unused	unused	'Break' character typed on keyboard
DEVICE_READY	8	Device number	unused	unused	assignment by 'pcontrol'
RESET_ALL_FINISH	9	Break device	unused	unused	RESET_ALL_DEVICES
DEVICE_NOT_READY	10	Device number	unused	unused	deassignment by 'pcontrol'
TAPEREAD_FINISH	11	Device number	unused	Actual count	TAPEREAD
TAPEWRITE_FINISH	12	Device number	unused	Actual count	TAPEWRITE
SPTM_FINISH	13	Device number	unused	Error flag *	SPTM
WRTM_FINISH	14	Device number	unused	Error flag *	WRTM
REWIND_FINISH	15	Device number	unused	Error flag *	REWIND
REWRITE_FINISH	16	Device number	unused	Error flag *	REWRITE

* =1, success
=0, error

Appendix D: Pcontrol Command Syntax Summary

```
assign <device num> [to] {<file name>¦RESET¦BREAK¦KEYBOARD¦
                      CRT¦NONE} [  [read¦write][tape] ¦ cread ¦ cwrite ]

break [is] <break character¦NONE>

cd <directory name>

clear

commands [from] <file name>

debug

exit

kick <device number>

reassign <device num> [to] {<file name>¦RESET¦BREAK¦KEYBOARD¦
                      CRT¦NONE} [  [read¦write][tape] ¦ cread ¦ cwrite ]

reset [is] <reset character¦NONE>

status

stty  arg1,...

trace <device number> [OFF]

!<shell command>

^b

: <command comment>
```

References

[1] Neuhauser, C., "Emmy System Processor -- Principles of Operation", Technical Note No. 114, Computer Systems Laboratory, Stanford University, Stanford CA 94305.

[2] Neuhauser, C., "Emmy System Peripherals -- Principles of Operation", Technical Note No. 77 (out of print), Computer Systems Laboratory, Stanford University, Stanford CA 94305

[3] Ritchie, D. M., Thompson, K., "The UNIX Time-sharing System.", Communictions of the ACM, Vol. 17, No. 7, July 1974, pp 365-375

[4] Shih, M., "Emmy/Unibus Interface Design Specification", Technical Note No. 109, Computer Systems Laboratory, Stanford University, Stanford CA 94305.