1077

Bell Laboratories

Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title- UNIX DH-11 Driver to Support Both Teletypewriter and DATASPEED® 40 Timesharing Terminals

Date- December 17, 1975

TM- 75-4145-01

Other Keywords- UNIX
DATASPEED 40
202 Data Set

Author(s)	Location and Room	Extension
R. C. White, Jr.	HO 4C-614	6142

Charging Case- 39447-13

Filing Case- 40117

ABSTRACT

A new DH-11 driver for the UNIX operating system has been written in order to support the general CRT form entry and display requirements of the CORDS/RBCS development. The driver was implemented in such a way as to incorporate support for both the traditional full-duplex line discipline used by teletypewriter-like terminals equipped with Bell System 100-series data sets and the previously unsupported half-duplex line discipline used by the Teletype Corporation's asynchronous series of DATASPEED 40 CRT terminals equipped with 202 series data sets. In both cases the full range of UNIX program development tools is supported, and in addition for the DATASPEED 40 terminal, full support with appropriate timing is provided for all form display, entry, and edit features. In the case of the DATASPEED 40, the line discipline implemented uses only standard terminal and data set options without the use of special applique hardware. The particular full or half-duplex discipline assigned to a particular UNIX port is determined by an editable disk file and can be changed while normal operating system tasks are continuing. For the DATASPEED 40 single line, multiline, and character block modes of transmission are supported along with proper safeguards to prevent the 1200-baud data rate from swamping the operating system. Dial-up, private line and null modems are supported by the driver. In the case of 202-series data sets, the 202S and T versions on two-wire facilities are recommended and should be equipped with JY2 reverse channel circuit packs.

Pages Text 9 Other 22 Total 32

No. Figures ____ No. Tables ____ No. Refs. 1

Address Label

BELL TELEPHONE LABORATORIES, INC. CO-UNOS-1

COMPLETE MEMORANDUM TO

ALBERTS,BARBARA A
ARNOLD,S L
BAYER,DOUGLAS L
BIRCHALL,R H
BIREN,MRS IRMA B
BLUM,MRS MARION
BRANDT,RICHARD B
BURROWS,T A
CANADAY,RUDD H
CARDOZA,WAYNE H
CARRAN,J H
CARR,DAVID C
CHAFFEE,H F
CHODROW,MARK H
COOK,THOMAS J
COREY,D A
CRANE,RODERICK P
CUNNINGHAM,STEPHEN J
DE JAGER,D S
DOLOTTA,T A
DOND,PATRICK G
DRUMMOND,R E
DWYER,T J
EDMUNDS,T W
ERRICHIELLO,PHILIP H
FACTOR,R H
FEDER,J
FLANDRENA,R J
FORTNEY,V J
FRANK,H G
GANNON,T F
GATES,G W
GLASSER,ALAN L
GRANT,GEORGE E
GRAVEMAN,R F
GUIDI,PIER V
HAIGHT,R C
HALPIN,THOMAS
HUPKA,MRS FLORENCE
HYMAN,B
IVIE,EVAN L
JACKOWSKI,D J
JOHNSON,STEPHEN C
JOHNSTON,WALTER E JR
KAPLAN,A E
KAUFELD,J C JR
KAUFMAN,LARRY S
KAYEL,R G
KEESE,W H
KETTERER,MRS B A
KEVORKIAN,DOUGLAS E
KNOX,BEVERLY G
LESSEK,PETER V
LIND,R O
LORENC,ANTHONY
LUDERER,GOTTFRIED W R
LUKACS,M E
LYONS,T G

MACHOL,R E JR
MARANZANO,JOSEPH F
MAROWITZ,J E
MASHEY,JOHN R
MC MILLAN,L G
MORGAN,S P
NEY,L

COMPLETE MEMORANDUM TO

NORTON,MRS SUZANNE W
O BRYAN,HENRY M JR
O CONNELL,T F
O NEILL,DENNIS M
OSSANNA,J F JR
PARA,P S
PATEL,C K N
PERDUE,R J
PEREZ,MRS CATHERINE D
PERNESKI,A J
PETERSON,RALPH W
PHILLIPS,S J
PILLA,MICHAEL A
RHODES,S P
RIDDLE,GUY G
ROCHKIND,M J
ROMITO,LETITIA J
ROSLER,LAWRENCE
ROVEGNO,MRS HELEN D
SABSEVITZ,A L
SATZ,L R
SHRUM,EDGAR V
SIX,FREDERICK B
SMITH,D W
SOUTHERN,MS M J
STAMPFEL,JOHN P
STEVENSON,D E
STORM,A R JR
STURMAN,JOEL N
SWANSON,GEORGE K
SWANSON,R H
TAGUE,BERKLEY A
THOMAS,P J
UNDERWOOD,R W
VASSALLO,J C
VIGGIANO,F A
VOGEL,GERALD C
WALKER,MS E A
WANDZILAK,P D
WATKINS,G T
WEBB,FRANCIS J
WEHR,L A
WHITE,RALPH C JR
WILSON,GEOFFREY A
WONSIEWICZ,B C
WOOD,J L
111 NAMES

COVER SHEET ONLY TO

ACKERMAN,A F
AHERN,PETER L
ALLISON,CHARLES E
AMOSS,JOHN J
ARNOLD,GEORGE W
BACCASH,MISS J M
BLAZIER,S O

BONANNI,LORENZO E
BROWN,COLIN W
BURNETTE,W A
CAMPBELL,STEPHEN T
CANNON,T B
CHERRY,MS L L
CHIANG,T C

COVER SHEET ONLY TO

CHRIST,C W JR
CLOUTIER,J E
COHEN,HARVEY
COLE,LOUIS M
COPP,DAVID H
COSTON,WALTER P
CRAGUN,D W
D STEFAN,D J
DAVIS,D R
DAVIS,R D
DONOFRIO,L J
DUDLEY,MRS E M
EDELSON,D
EITELBACH,DAVID L
ESTOCK,R G
FELTON,W A
FOUNTOUKIDIS,A
FRANZ,ANN M
FREEMAN,M I
FREEMAN,R DON
FREIDENREICH,MRS B
GIBSON,M T JR
HAGGERTY,JOSEPH P
HALL,JOE T
HAMILTON,PATRICIA
HANNAM,JUDY R
HEDRICK,MISS ELLEN L
HELD,RICHARD W
HESTER,S D
HOYT,WILLIAM F
IMAGNA,CLYDE P
JACKSON,JAMES H
JAKUBEK,RAYMOND J
JONES,DAVID P
KAMINSKY,MICHAEL L
KEMP,C W
KERNIGHAN,BRIAN W
KLEIN,MISS R L
KNUDSEN,DONALD B
KOLETTIS,N J
LA CAVA,JOSEPH L
LACY,J V
LAMBERT,MRS C A
LANG,A
LEGENHAUSEN,S
LEVINE,P R
LICHINKO,J S
LINDERMAN,J
LIST,WILLIAM H S
LONG,DAVID W
LOYD,D G
LYCKLAMA,HEINZ
MALIK,JOSEPH M
MASHEY,MRS R
MC ILROY,M DOUGLAS
MELCHNER,MELVIN J
MENZEL,MRS F S
MILLER,R E
MILLSPAUGH,J A JR

MILNE,D C
MORGAN,DENNIS J
NOWITZ,D A
OLSEN,RONALD G
PEARLMAN,MS M E
PEKARICH,S P
PETERSON,T G

COVER SHEET ONLY TO

PETSCHENIK,N M
QUINN,MISS M
RITACCO,J E
RIZZO,J F
ROBINSON,HERMAN E
ROH,DOUGLAS B
ROOME,WILLIAM D
ROSENBLUM,M J
SAND,DOUGLAS S
SCHLEGEL,C T
SCRIBNER,NEAL M
SHELTON,JAMES C
SHOMO,R E
SIGGR,DONALD J
SILVERSTEIN,STEVEN M
SMITH,K M
STROHECKER,CARY A
STUKAS,MRS LORETTA I
TARTARONE,STUART A
THOMPSON,BERNARD E
THOMPSON,MRS M
VAN ORMUM,JAMES M
VIZCARRONDO,J
VYSSOTSKY,VICTOR A
WALFORD,ROBERT B
WEYTHMAN,J E
WHIPPLE,JOHN M
WILHOYTE,M L
WINCKELMANN,M A
WINNEIN,MISS IRENE A
110 NAMES

COMPLETE MEMORANDUM TO
N. Amitay
R. M. Baig
A. R. Eggerman
J. Feder
J. A. Giunta
W. Heidel
W. H. Iarley
H. P. Kelly
J. W. Landis
P. Long
J. F. Maranzano
E. R. Nagelberg
K. C. O'Brien
J. W. Pan
M. A. Pilla
T. M. Raleigh
D. M. Ritchie
A. E. Rosenberg
T. P. Tabloski
B. A. Tague

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
UNOS# = UNIX/OPERATING SYSTEM

COVER SHEET TO:

BIREN,MRS I B
MH 2C548

CO-UNOS-1
TOTAL PAGES

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

subject: UNIX DH-11 Driver to Support Both Teletypewriter and DATASPEED® 40 Timesharing Terminals - Charging Case 39447-13, Filing Case 40117

date: December 17, 1975

from: R. C. White, Jr.
TM 75-4145-01

MEMORANDUM FOR FILE

I. INTRODUCTION

Among the several input/output processes required by the RBCS* development is the ability to display, enter, and edit network data, some of which is available in mechanized form and some available only from paper records. The initial RBCS system design[1] proposed that this data, particularly the unmechanized portion, be interfaced to and from the user via the Teletype Corporation DATASPEED 40[2] CRT display using specially written application programs, one each, for the several categories of data required to be displayed for various RBCS users. The UNIX operating system was best suited and recommended as the major software tool for developing and running RBCS, but did not support the newly introduced DATASPEED 40 terminal. This memorandum describes the UNIX driver that was written to provide support for the DATASPEED 40 so that the necessary CRT application programs for interfacing RBCS data to users could be written.

II. FUNCTIONS IMPLEMENTED IN THE DRIVER

Although the driver was destined for use by the RBCS project, it was designed and written from the beginning with the intent that it would be compatible with and support all the normal functions of standard UNIX. It is hoped that the driver with perhaps some modifications will become the standard UNIX DH-11/DM-11[3,4] driver. As a consequence of this intent, the driver fully supports all the standard program development tools of UNIX plus all the protected and unprotected form display and edit features of the DATASPEED 40. The driver was thus fully used during RBCS development and in the actual field operation of the RBCS system. It has been in use a total of about eight months.

* Record Base Coordination System.

The driver was implemented in such a way as to incorporate support for both the traditional UNIX full-duplex line discipline used by teletypewriter-like terminals equipped with 100[5]-series data sets, as well as the previously unsupported half-duplex line discipline used by the Teletype Corporation's asynchronous series of DATASPEED 40 CRT terminals equipped with 202S and T[6,7] series data sets. In the case of the DATASPEED 40, a User Line Control[8] discipline was implemented with support for a buffered terminal and reverse channel. This technique masks from the human operator many of the physical realities of the half-duplex nature of the DATASPEED 40 with a 202 data set and makes the terminal appear full-duplex. Both the user and computer are free to transmit at any time as long as they are not already receiving data from the other end. The operator uses the CONVERSATIONAL mode of the DATASPEED 40 and does not have to press a special SEND key to transmit; one only must type the message which is buffered by the DATASPEED 40 and hit the return key when the line is complete. This is how a regular teletype terminal is used. There is no requirement that the user and computer alternate one-for-one the transmission of messages, neither must the computer poll the user terminal consuming CPU cycles unnecessarily, nor are special BTL hardware appliques[9] required for proper line control. The driver requires only standard DATASPEED 40 and 202 data set options. This is a great advantage for the RBCS project in that all terminal and data set hardware can be provided and supported in a standard way by Bell System Operating Companies. One DATASPEED 40 option required by RBCS but not by the driver is the Remote Send option which is documented and supported by Teletype Corporation but not normally made available to non-Bell System customers. This option allows the host computer to place the DATASPEED 40 in the SEND mode and is used to simplify certain RBCS operator tasks when working with forms. Appendices I and II respectively tabulate the options required by the RBCS project for its DATASPEED 40 terminals[10,11] and 202 data sets.[12]

The provision of the proper full or half-duplex discipline to a particular UNIX port is determined by an editable UNIX disk file and can be changed, provided the proper password is given, while normal operating system tasks continue uninterrupted.

For users of DATASPEED 40 terminals, single line, multiline, and character block modes of transmission are supported along with proper safeguards to prevent the 1200-baud data rate from swamping the operating system when long messages are composed by the user and sent to the computer in one burst. This safeguard is implemented by using the reverse channel carrier to temporarily suspend input transmissions when the CPU is heavily loaded.

Dial-up, private line, and null modems are supported by the driver. In the case of the 202 data set, circuit assurance that a user is active is provided by reception at the computer of reverse channel from the DATASPEED 40 at all times other than when one end is actively transmitting. Loss of reverse channel for more than 10 seconds when no transmitting is taking place will log the user off of the offending port. This feature is necessary in a timesharing environment to handle the case of users hanging up without logging off the system. Reverse channel signaling is also used at DATASPEED 40 terminal positions to prematurely terminate transmissions from the computer in the case of errors in command strings. This is essentially the DELETE feature available at a standard teletype-like UNIX port. Early versions of the driver allowed such terminated transmissions to be resumed by the user, but this feature was eventually removed because it was infrequently used.

III. REVIEW OF HOW CERTAIN FEATURES WERE IMPLEMENTED IN CODE

A. User Port Control Table

The standard UNIX control table that is maintained for each user port was expanded to include space for keeping track of the additional logical states that have to be defined when handling a half-duplex discipline. Moreover, an additional flag bit was defined to indicate if the control table bits apply to a full or half-duplex port. This flag bit, defined as "MOD40", appears repeatedly in the driver and is used to invoke half-duplex sections of code when the bit is set. Changing this bit using the standard UNIX "stty" system call is the way line disciplines are changed to either half or full-duplex without stopping normal operating system tasks. The format of the new UNIX port control table, called struct tty, is given in Appendix III along with the mnemonic define parameters indicating the use of each logical bit.

B. Prevention of Clist Overflow

Since UNIX temporarily stores characters that have just been received, but not passed along to the respective user programs in a 600-character common buffer area called the Clist, a 120-character per second DATASPEED 40 transmitting a multiline message can quickly exhaust the entire buffer area. To prevent this situation from occurring, special code has been put in the DH-11 driver, dh40.c, and in the general tty subroutine package, tty.c, both of which are reproduced in Appendices IV and V respectively. In the driver on page 3 beginning at the arrow labeled Clist Control, a check is made to see if the raw queue character count is greater than or equal to 90 and if it is, the supervisory (reverse channel)

transmit data bit to the 202 data set is cleared so that the DATASPEED 40 hardwired logic will be forced to temporarily stop transmitting. Furthermore, the UNIX operating system will immediately begin passing the already accumulated characters to the application program even though the message is as yet incomplete. This frees space in the common buffer area. Meanwhile, in tty.c, Appendix V on page 4, beginning at the arrow marked Clist Control, the user port's raw queue character count is checked to see when it drops below the low water mark of 30 characters. When this occurs the threat to the common buffer area is over so the supervisory transmit data lead to the 202 modem can be set allowing the hardwired logic in the DATASPEED 40 to resume transmission. In practice, the user rarely sees this juggling because a single character at 1200 baud takes 8 milliseconds to transmit while the 60 or so characters in the clist can be moved via core-to-core transfer to the application program in less time than this.

C. Transmission Interrupt

When the computer is transmitting to the DATASPEED 40 and the user wants to stop the message, this is accomplished by toggling the INTRPT key on the DATASPEED 40 console. This causes a momentary interruption of reverse carrier back to the computer which the driver interprets as a request to entirely abort the current task. The code for recognizing this situation begins on page 5 of the driver and is labeled Transmission Interrupt. If supervisory-received data (reverse channel) goes off and active transmission is taking place, an ASCII delete character is input by the driver on the application program's port. This is the standard UNIX method of aborting the task executing on that port and returning the user of that port to the command level. Early versions of the driver merely suspended transmission and allowed the user to either input his own delete character or resume transmission where it was left off. This feature was removed when most of the time it was found that users wanted to return to the command level.

D. Burst Transmissions

When the DATASPEED 40 is sending data from a form displayed on its screen, only the unprotected, or variable, portion is transmitted with each unprotected field separated from the next by a horizontal tab character. Since this type of message contains none of the standard UNIX line terminating characters, the message builds up in the clist and is not sent to the application program. The 90-character limit is still effective so the concern is primarily with messages that are not even multiples of 90. In this situation the application program

controlling the form has been provided control of the so-called BLOCK bit, which is set and cleared by the stty system call mentioned previously. When set by the application program, all normal UNIX message terminating characters such as newline, return, EOT, ETX, and DC4 are treated as ordinary characters and message termination is recognized only by the loss of primary received carrier. The code that implements this function begins on page 7 of the driver at the arrow labeled Burst Transmissions. If a carrier transition from on to off occurs and the BLOCK bit is set, then the data on the raw queue will be passed to the application program. Assisting in the proper operation of the BLOCK mode of transmission is a section of code in tty.c on page 5 beginning also at the arrow labeled Burst Transmissions. Here the primary purpose is to treat normal message ending characters as ordinary characters only if the BLOCK bit is set.

E. DATASPEED 40 Delays

When forms and field attributes are being written to the DATASPEED 40 terminal special two-character control sequences are employed each of which requires a certain amount of time for execution before the next data character or control sequence can be sent. The code beginning on page 6 of tty.c at the arrow labeled DATASPEED 40 Delays implements the correct amount of delay for the special functions that require it.

There are many other instances in the driver and the helping subroutines in tty.c where additions and modifications have been made to support the asynchronous terminals that are typically found in a RBCS environment. The modem interrupt control section, page 6 of the driver, is particularly critical because of the many modem interrupts generated by a half-duplex discipline using reverse channel. Care had to be taken not to disrupt full-duplex ports because of the special requirements of the DATASPEED 40. At one point in the RBCS system design it was found necessary to support a half-duplex low-speed protocol to a remote IBM 1050 card punch.[13] The DH-11 driver was modified to accomplish this but with a minimum of kernel space code because it was felt the punch would not be in use enough of the time to justify using the limited kernel space available for that terminal device.

IV. CHANGING UNIX TO INCLUDE THE NEW DH-11 DRIVER

In addition to changing tty.c and tty.h, incorporating dh40.c into UNIX required the modification of several other UNIX programs.

A. Init.c

Init.c was modified to accept more than one "getty.s" routine. This allows the line discipline for a particular user port to be changed without halting the operating system.

B. Getty.s

The original getty.s for full-duplex was renamed getty0.s and a new getty1.s was created for half-duplex ports. The different "getty.s" programs are used to initialize the user port's control table at the beginning of a session giving the port the correct flag and speed bits for the required port discipline.

C. Editing of Disk File /etc/ttys

The file /etc/ttys is edited to indicate which disciplining is to be used on each port, i.e., 150 on a line says that port 5 is full-duplex, 161 on a line says port 6 is half-duplex, and 071 says port 7 is half-duplex but is not recognized as a legal timesharing port.

D. Modifying Command Line Prompter

The UNIX command line interpreter, shell.c, was modified to use the "bell" "@" combination as the prompter to the user. The bell will alert the user that the last command has completed and the @ sign will cause the bell character to be ignored in tty.c when the message is passed to the application program. Also, after shell.c has read a command line, it now outputs a new line character to move the DATASPEED 40 cursor to the next line on the screen.

E. Recompiling UNIX Modules that Use tty.h

All UNIX kernel and user programs that "include" or refer to tty.h must be recompiled since the size of the port control table was increased.

F. Installing dh40.c

The new dh40.c must be compiled and used to replace the old DH-11/DM-11 driver in /usr/sys/lib2.

G. Installing tty.c

The new tty.c must be compiled and used to replace the old tty.c in /usr/sys/lib2.

H. UNIX Operating System

A new /unix must be loaded and the system rebooted.

Conversations with M. A. Pilla were most valuable during the development of the driver.

R. C. White, Jr.

HO-4145-RCW-mk

R. C. White, Jr.

Atts.
References
Appendices I through V

REFERENCES

1. P. M. Bauer, G. K. Chang and R. C. White, Jr., "A Proposal for a Minimal Miami Implementation," Memorandum for File, File 40117, May 30, 1974.

2. "DATASPEED® 40", GL: 73-06-100, June 22, 1973.

3. "DH-11 Asynchronous 16-Line Multiplexer Maintenance Manual," DEC-11-HDHAA-A-D, Digital Equipment Corporation, Maynard, Massachusetts.

4. "DM-11-BB Modem Control Option Manual," DEC-11-HDMBA-B-D, Digital Equipment Corporation, Maynard, Massachusetts.

5. "Data Sets 202S and 202T," GL: 74-05-153, May 30, 1974.

6. "Data Set 202S Transmitter-Receiver Description and Operation," BSP 592-028-100, Issue 1, August 1973.

7. "Data Set 202T Transmitter-Receiver Description and Operation," BSP 592-031-100, Issue 1, August 1973.

8. Carl Mee, III, "Techniques for Control of Half-Duplex Asynchronous Modems," TM 75-8233-1, August 28, 1975.

9. P. E. Rosenfeld, "A Hardware Applique for the 202 Data Set to Control Line Turnaround," TM 75-8231-9, June 9, 1975.

10. "DATASPEED 40 Field Installation and Maintenance Practice," BSP 575-100-350, Issue 1, December 1973.

11. "DATASPEED 40 Stations for DATAPHONE® Service," AT&T Publication, 417515, December 1973.

12. R. C. White, Jr., "Application Program to Operate an IBM Model 1050 Remote Card Punch," Memorandum for File, File 40117, to be published.

FILL OUT AND
KEEP WITH SET

APPENDIX I

STATION FEATURES AND OPTION RECORD

☒ Check on variable * as furnished
* (asterisk) Indicates Factory Programming

SET & FEATURES	RELATED FIELD OPTIONS	
BASIC KSR ☒ (Serial EIA)	1 (a) EIA to Printer ☐ 3 (a) 1050 Baud Send Receive ☐ 3 (c) 2400 WPM future use 4 (a) Rev. Chan. Sent & Received * ☒ 28 (a) Disconnect after 45 seconds * ☐	1 (b) SSI to Printer * ☒ 3 (b) 1200 Baud Send Receive * ☒ 4 (b) Rev. Chan. Not Sent-Received ☐ 28 (b) Does not disconnect-disabled ☒
KSR (PTI) ☐	None	
PRINTER ☒	17 (a) 1st Char. — 1st Position * ☒ 17 (c) Last Char. — 80th Position * ☒ 17 (e) 11 Inch Form Length ☐ 18 (a) No Paper Feed Out ☐ 18 (c) Paper FO on RM Loss and ETX * ☒ 19 (a) Symbol on Even V. P. Error * ☒ 19 (c) No Symbol on V. P. Error ☐ 19 (e) 64 Character Set ☐ 20 (a) Single LF * ☒ 21 (a) Lower and Upper Case Print (U. L.) * ☒ 22 (a) Lower Case Print as Error Symbol (Mono) ☐ 23 (a) Prints Extended ASCII Characters ☐	17 (b) 1st Char. — 2nd thru 12th Position ☐ 17 (d) Last Char. — 73rd thru 79th Position ☐ 17 (f) 2-1/2 to 22 Inch Form Length ☐ 18 (b) Paper FO of 16 Lines on RM Loss ☐ 19 (b) Symbol on Odd V. P. Error ☐ 19 (d) 96 Character Set ☒ 19 (f) Printers using Ext. ASCII Char. Set ☐ 20 (b) Double LF ☐ 21 (b) Lower Case Printed as Upper Case (U. L.) ☐ 22 (b) Lower Case Printed as Upper Case (Mono)* ☐ 23 (b) Ext. ASCII Char. not Printed (See 19 a, b, or c.) ☐
FULL EDIT (Controller) ☒	5 (a) Reject NULL * ☒ 5 (c) Reject CR * ☒ 5 (e) Reject DELETE * ☒ 6 (a) Esc. Displayed, No Functions ☐ 7 (a) Vert. Par. Err. Char. Not Displayed ☐ 8 (a) End Page on FF ☐ 8 (c) End Page on ETX * ☒ 8 (e) End Page on EOT * ☒ 8 (g) End Page on GS * ☒ 9 (a) High Light Not Sent ☒ 10 (a) Send CR LF ☐ 10 (c) Send LF ☒ 11 (a) Go Local after Send * ☐ 12 (a) No Form Enter in Local ☐ 13 (a) Send All (P and Unp.) as Displayed ☐ 13 (c) Send Protect as SPACE ☐ 13 (e) Send Protect as DELETE ☐ 13 (g) Send Unp. and HT at End of Field ☒ 27 (a) Home Cursor on Transmit ☐	5 (b) Accept NULL ☐ 5 (d) Accept CR ☐ 5 (f) Accept DELETE ☐ 6 (b) Esc. Not Displayed/Function Enabled * ☒ 7 (b) Vert. Par. Err. Char. Displayed * ☒ 8 (b) Do Not End Page on FF * ☒ 8 (d) Do Not End Page on ETX ☐ 8 (f) Do Not End Page on EOT ☐ 8 (h) Do Not End Page on GS ☐ 9 (b) High Light Sent * ☐ 10 (b) Send CR CR LF * ☐ 11 (b) Go Receive after Send ☒ 12 (b) Form Enter Enabled in Local * ☒ 13 (b) Send All, HT Convert to SPACE * ☐ 13 (d) Send Protect & HT as SPACE ☐ 13 (f) Send Unprotect (Unp.) Only ☐ 13 (h) Send Unp. and HT as SPACE ☐ 27 (b) Send from Cursor Position * ☒
(Controller) FULL EDIT (PTI Sets With EXT. Mode) ☐	11 (c) Replaces 11 (a) or 11 (b) * ☐ 14 (a) Rec. Select on * ☐ 15 (a) Override goes to Rec. ☐ 16 (a) Rec. after TDA * ☐ 16 (c) Home after TDA ☐	14 (b) Rec. Select off ☐ 15 (b) Stays in Send or Loc * ☒ 16 (b) Loc after TDA ☐ 16 (d) Stays at Mess. end * ☐
(Controller) CONVERSATION MODE ☒	2 (a) DC_2 Sent Auto. * ☐	2 (b) DC_2 Sent with Message ☒
RO SET ☐	3 (a) 1050 Baud ☐ 24 (a) Even VP Detection * ☐ 25 (a) Odd Parity Null Sent to Printer ☐ 25 (c) D. E. Lamp Turns On ☐ 26 (a) Discr Call Coded ☐ 26 (c) Discr Call Sent ☐	3 (b) 1200 Baud * ☐ 24 (b) Odd VP Detection ☐ 25 (b) Received Character Sent to Printer * ☐ 25 (d) D. E. Lamp Does Not Turn On * ☐ 26 (b) Discr Call Not Coded * ☐ 26 (d) Discr Call Not Sent * ☐
SET VARIATIONS	Mono-Case ☐ Upper-Lower Case ☒ 24 Line Display ☐ 48 Line Display ☐ 72 Line Display ☒	Pedestal ☐ Paper Winder ☐ Copy Holder ☐

IN CASE OF TROUBLE CALL _____

APPENDIX II

202S and T Options*

1. JY2 reverse channel circuit pack required with:
   a. reverse channel independent of primary channel
   b. no local copy on reverse channel, (ZF).
2. 202S modem circuit pack with:
   a. transmit level, -7 (ZQ)
   b. reverse channel, in (ZC)
   c. soft turnoff, 24 ms (R)
   d. squelch, 156 ms (R)
   e. fast carrier, out (N)
   f. clear to send, 180 ms (G)
   g. out answer, in (B)
   h. local copy on primary channel, out (ZB)
   i. clamp, in (F)
   j. grounding (ZG)
3. 202T modem circuit pack with:
   a. 2-wire private line
   b. same as 202S otherwise

---

* BSP 592-028-200
BSP 592-031-200.

Jul 14 09:29 ttv.h Page 1

```
struct clist {
        int     c_cc;
        int     c_cf;
        int     c_cl;
};

struct tty {
        struct  clist t_rawq;
        struct  clist t_canq;
        struct  clist t_outq;
        int     t_flags;
        int     *t_addr;
        char    t_delct;
        char    t_col;
        char    t_intrup;
        char    t_quit;
        int     t_state;
        char    t_char;
        int     t_speeds;
        int     t_ghaddr;
        char    t_stopct;
        char    t_offct;
        int     t_dev;
};

char partab[];

#define TTIPRI  10
#define TTOPRI  20
#define CERASE  '#'
#define CEOT    004
#define CEIX    003
#define CDC2    022
#define CDC4    024
#define CTAB    011
#define CKILL   '@'

/* limits */
#define TTHIWAT 50
#define TTLOWAT 30
#define TTYHOG  256

/* modes */
#define NODELAY 01
#define XTABS   02
#define LCASE   04
#define ECHO    010
#define CRMOD   020
#define RAW     040
#define ODDP    0100
#define EVENP   0200
#define m2741   0400
#define OCORR   02000
#define ICORR   04000
#define NTDELAY 010000
#define MOD40   0400
```

Changed from char to int to give more logical states

Flag bit for Dataspeed 40

```
#define BLOCK   01000

/* Bits */
#define DONE    0200
#define IENABLE 0100
#define TIMEOUT 01
#define WOPEN   02
#define ISOPEN  04
#define SSTART  010
#define CARR_ON 020
#define BUSY    040
#define REC_ON  0100
#define TRAN_ON 0200
#define SRDBIT  0400
#define STDBIT  01000
#define SUPTD 010
#define ENDBIT  02000
#define DELAY40     04000
#define WAK377  010000
#define WR_ON   020000
#define SRD_ON  040000
#define STD_ON  0100000
#define DELTA   1
#define PAUSE   18
#define ONE     60
#define TWO     120
#define THREE   180
#define TEN     600
#define FIFTEEN 900
```

Flag bit for block transmissions

Additional logical state bits

Timeout delays

60 = 1 minute

```
#
/*
 *      Copyright 1975 Bell Telephone Laboratories Inc
 */

/*
 *      DH-11 driver
 *      for 103 and 202 type modems.
 */

#include "/usr/sys/param.h"
#include "/usr/sys/conf.h"
#include "/usr/sys/user.h"
#include "/usr/sys/tty.h"
#include "/usr/sys/proc.h"

#define DHADDR  0160020
#define NDH11   16

struct  tty dh11[NDH11];
int     ndh11   NDH11;
int rocslock,rocswstd;

#define DMADDR  0170500

#define BITS6   01      /* excluding parity */
#define BITS7   02
#define BITS8   03
#define STOP2   04
#define PENABLE 020
#define OPAR    040     /* beware DEC manuals */
#define HDUPLX  040000
#define ENABLE 030100  /* receive, transmit, and storage */
#define PERROR  010000
#define FRERROR 020000
#define SSPEED  7       /* standard speed: 300 baud */
#define HSPEED  9       /* MOD40 speed: 1200 baud half duplex */
#define SCENABL 040
#define CLSCAN  04000
#define ROSEND  04
#define CLSEND  040
#define CDLEAD  02
#define LENABLE 01
#define CSTRANS 020000
#define SRTRANS 010000
#define CTRANS  040000
#define RGTRANS 0100000
#define SUPTD   010
#define SUPRD   020
#define CARRIER 0100
#define DMBUSY    020
#define CLEAR   0177776

#define PS      0177776

int     dhsar;
```

```
struct dhregs {
        int dhcsr;
        int dhnxch;
        int dhlpr;
        int dhcar;
        int dhbcr;
        int dhbar;
        int dhbreak;
        int dhsilo;
};

struct {
        char    lbyte;
        char    hbyte;
};

dhopen(dev, flag)
{
        register struct tty *tp;
        extern dhstart();

        if (dev.d_minor >= NDH11) {
                u.u_error = ENXIO;
                return;
        }
        tp = &dh11[dev.d_minor];
        tp->t_addr = dhstart;
        DHADDR->dhcsr =| ENABLE;
        tp->t_state =| WOPEN|SSTART;
        if((tp->t_state&ISOPEN) == 0){
                tp->t_quit = 034;               /* FS */
                tp->t_intrup = 0177;            /* DEL */
                if((tp->t_flags & MOD40) != 0){
                        tp->t_speeds= HSPEED|(HSPEED<<8);
                        tp->t_flags= EVENP|NODELAY|CRMOD|MOD40;
                }else{
                        tp->t_speeds = SSPEED | (SSPEED<<8);
                        tp->t_flags = EVENP|ECHO|CRMOD|XTABS;
                }
                dhparam(tp);
        }
        dmopen(dev);
        tp->t_state =& ~WOPEN;
        tp->t_state =| ISOPEN;
        if (u.u_procp->p_ttyp == 0)
                u.u_procp->p_ttyp = tp;
        tp->t_dev = dev;
}

dhclose(dev)
{
        register struct tty *tp;

        tp = &dh11[dev.d_minor];
        tp->t_state=& (CARR_ON|SSTART);
        wflushtty(tp);
```

```
}

dhread(dev)
{
        register struct tty *tp;

        tp = &dh11[dev.d_minor];
        if ((tp->t_state&CARR_ON) != 0)
                ttread(tp);
}

dhwrite(dev)
{
        register struct tty *tp;

        tp = &dh11[dev.d_minor];
        if((tp->t_state & CARR_ON) == 0) return;
        if((tp->t_flags & MOD40) != 0){
                tp->t_state=| WR_ON;
                while((tp->t_state & REC_ON) != 0)
                        sleep(&tp->t_state,TTOPRI);
                dmstat(tp,'s',ROSEND,0);
                while(((tp->t_state & TRAN_ON) == 0) ||
                        ((tp->t_state & SRD_ON) == 0))
                                sleep(&tp->t_state,TTOPRI);
                ttwrite(tp);
                tp->t_state=& ~WR_ON;
        }else ttwrite(tp);
}

dhrint()
{
        register struct tty *tp;
        register int c;

        while ((c = DHADDR->dhnxch) < 0) {      /* char. present */
                tp = &dh11[(c>>8)&017];
                if (tp >= &dh11[NDH11])
                        continue;
                if(((tp->t_state & ISOPEN) == 0) ||
                        (c&PERROR) ||
                        ((tp->t_state & TRAN_ON) != 0)){
                                wakeup(&tp->t_state);
                                continue;
                }
                if (c&FRERROR)          /* break, return null */
                        c = 0;
                if(((tp->t_flags & MOD40) == 0) &&
                        (rbcslock & (01<<((c>>8)&017)))) continue;
                        else ttyinput(c,tp);
                if((tp->t_rawq.c_cc >= 90) &&
                        ((tp->t_flags & MOD40) != 0) &&
                        ((tp->t_state & STOBIT) == 0)){
                                dmstat(tp,'c',SUPTO,0);
                                tp->t_state=| STOBIT;
                                if((tp->t_state & WAK377) == 0){
```

clist control

```
                                        if(putc(0377,&tp->t_rawq) == 0){
                                                tp->t_delct++;
                                                tp->t_state=| WAK377;
                                        }
                                }
                                wakeup(&tp->t_rawq);
                        }
                }
}

dhsgtty(dev, av)
int *av;
{
        register struct tty *tp;
        register r;
        register int *v;

        tp = &dh11[dev.d_minor];
        v = av;
        if (v) {
                *v++ = tp->t_speeds;
                *v++ = 0;
                *v++ = tp->t_flags;
                return(0);
        }
        wflushtty(tp);
        tp->t_speeds = u.u_arg[0];
        tp->t_flags = u.u_arg[2];
        while((tp->t_flags & MOD40) && dmstat(tp,'t',ROSEND,0)) turnrd(tp);
        dhparam(tp);
        return(0);
}

dhparam(atp)
struct tty *atp;
{
        register struct tty *tp;
        register int lpr;

        tp = atp;
        spl5();
        DHADDR->dhcsr.lbyte = (tp - dh11) | ENABLE;
        lpr = (tp->t_speeds.hbyte<<10) | (tp->t_speeds.lbyte<<6);
        if (tp->t_speeds.lbyte == 4)            /* 134.5 baud */
                lpr=| BITS7|HDUPLX; else
                if (tp->t_flags&EVENP)
                        if (tp->t_flags&ODDP)
                                lpr =| BITS8; else
                                lpr =| BITS7|PENABLE; else
                        lpr =| BITS7|OPAR|PENABLE;
        if(tp->t_speeds.lbyte == 3)     /* 110 baud */
                lpr=| STOP2;
        if((tp->t_flags & MOD40) != 0) lpr=| HDUPLX;
        DHADDR->dhlpr = lpr;
        spl0();
}
```

```
dhxint()
{
        extern turnrd();
        register struct tty *tp;
        register ttybit, bar;

        bar = DHADDR->dhbar;
        DHADDR->dhcsr =& ~0101060;
        bar = (bar|dhsar)^bar;
        ttybit = 1;
        for (tp = dh11; tp < &dh11[NDH11]; tp++) {
                if(bar&ttybit) {
                        dhsar =& ~ttybit;
                        tp->t_state =& ~BUSY;
                        if(((tp->t_flags & MOD40) != 0) &&
                                ((tp->t_state & SRD_ON) == 0) &&
                                ((tp->t_state & TRAN_ON) != 0)){
                                        ttyinput(0177,tp);
                                        tp->t_state=& ~(WR_ON|ENDBIT);
                                        timeout(turnrd,tp,DELTA);
                        } else dhstart(tp);
                }
                ttybit =<< 1;
        }
}

dhstart(atp)
struct tty *atp;
{
        extern ttrstrt(),endit();
        register lineno, c;
        register struct tty *tp;
        int sps;
        struct { int int; };

        sps = PS->int;
        spl5();
        tp = atp;
        if (tp->t_state&(TIMEOUT|BUSY) || (tp->t_outq.c_cc==0))
                return;
        if ((c = getc(&tp->t_outq))<=0177) {
                tp->t_char = c;
                lineno = tp-dh11;
                DHADDR->dhcsr.lbyte = lineno | ENABLE;
                DHADDR->dhcar = &tp->t_char;
                DHADDR->dhbcr = -1;
                lineno = 1<<lineno;
                DHADDR->dhbar =| lineno;
                dhsar =| lineno;
                tp->t_state =| BUSY;
        } else {
                timeout(ttrstrt, tp, c&0177);
                tp->t_state =| TIMEOUT;
        }
        if (tp->t_outq.c_cc == 0 || tp->t_outq.c_cc == TTLOWAT){
```

Transmission Interrupt

```
                wakeup(&tp->t_outq);
                if(((tp->t_flags & MOD40) != 0) &&
                        ((tp->t_state & ENDBIT) == 0)){
                                tp->t_state=| ENDBIT;
                                timeout(endit,tp,PAUSE);
                }
        }
        PS->int = sps;
}

/*
 *      Copyright 1975 Bell Telephone Laboratories Inc
 */

/*
 *      DM-BB driver
 */



struct dmregs {
        int     dmcsr;
        int     dmlstat;
};

int dmc,dms;

dmopen(dev)
{
        register struct tty *tp;

        tp = &dh11[dev.d_minor];
        spl5();
        DMADDR->dmcsr=& ~SCENABL;
        DMADDR->dmcsr = dev.d_minor;
        if((tp->t_state & ISOPEN) == 0)
                if((tp->t_flags & MOD40) == 0)
                        DMADDR->dmlstat= LENABLE|CDLEAD|ROSEND;
                        else DMADDR->dmlstat= LENABLE|CDLEAD;
        dmc= DMADDR->dmcsr;
        dms= DMADDR->dmlstat;
        if(dms & CARRIER) tp->t_state=| CARR_ON;
        if(dms & SUPRD) tp->t_state=| (SRD_ON|CARR_ON);
        DMADDR->dmcsr = IENABLE|SCENABL;
        while((tp->t_state & CARR_ON) == 0)
                sleep(&tp->t_state,TTIPRI);
        spl0();
}

dmint()     <-------------------------------------------------------- Modem Interrupt Control
{
        register struct tty *tp;
        register csr,lsr;
        int scratch;
        extern hup();
```

```
        if((DMADDR->dmcsr & DONE) != 0){
                tp= &dh11[DMADDR->dmcsr&017];
                if(tp < &dh11[ndh11]){
                        csr= DMADDR->dmcsr;
                        lsr= DMADDR->dmlstat;
                        if((tp->t_flags & MOD40) == 0)
                                if ((DMADDR->dmlstat&CARRIER)==0) {
                                        tp->t_state =& ~CARR_ON;
                                        if((tp->t_state & WOPEN) == 0){
                                                tp->t_state=& ~ISOPEN;
                                                signal(tp, SIGHUP);
                                                DMADDR->dmlstat =& ~CDLEAD;
                                                flushtty(tp);
                                        }
                                } else {
                                        tp->t_state=| CARR_ON;
                                        wakeup(&tp->t_state);
                                }
                        else{
                                if((csr & CTRANS) != 0){
                                        if((lsr & CARRIER) == 0){
                                                tp->t_state=& ~REC_ON;
                                                DMADDR->dmlstat=& ~SUPTD;
                                                if((tp->t_flags & BLOCK) != 0)
                                                        if((tp->t_state & WAK377) == 0){
                                                                if(putc(0377,&tp->t_rawq) == 0)
                                                                        tp->t_delct++;
                                                                if(putc(0377,&tp->t_rawq) == 0)
                                                                        tp->t_delct++;
                                                        } else if(putc(0377,&tp->t_rawq) == 0)
                                                                        tp->t_delct++;
                                                wakeup(&tp->t_rawq);
                                                wakeup(&tp->t_state);
                                        } else {
                                                tp->t_state=| REC_ON;
                                                scratch= 01<<(DMADDR->dmcsr&017);
                                                if(rbcslock & scratch) rbcswstd=| scratch;
                                                        else DMADDR->dmlstat=| SUPTD;
                                        }
                                }
                                if((csr & CSTRANS) != 0){
                                        if((lsr & CLSEND) != 0) tp->t_state=| TRAN_ON;
                                        wakeup(&tp->t_state);
                                }
                        }
                        if((csr & SRTRANS) != 0)
                                if((lsr & SUPND) == 0){
                                        tp->t_state=& ~SRD_ON;
                                        if((tp->t_state & (REC_ON|TRAN_ON|WOPEN)) == 0)
                                                timeout(hup,tp,TEN);
                                } else {
                                        tp->t_state=| SRD_ON|CARR_ON;
                                        wakeup(&tp->t_state);
                                }
                }
```

Burst Transmission

```
                dmc= DMADDR->dmcsr;
                dms= DMADDR->dmlstat;
                DMADDR->dmcsr= IENABLE|SCENABL;
        }
}

dmstat(atp,arg)
{
        register struct tty *tp;
        register *argp;
        int ret;

        tp= atp;
        argp= &arg;
        spl5();
        DMADDR->dmcsr=& ~SCENABL;
        DMADDR->dmcsr= tp - dh11;
        while(*argp)
                switch(*argp++&0377){
                        case 't':{
                                if((DMADDR->dmlstat & *argp++) != 0) ret= 1;
                                else ret= 0;
                                break;
                        }
                        case 's':{
                                DMADDR->dmlstat=| *argp++;
                                break;
                        }
                        case 'e':{
                                DMADDR->dmlstat= *argp++;
                                break;
                        }
                        case 'c':{
                                DMADDR->dmlstat=& ~*argp++;
                        }
                }
        dmc= DMADDR->dmcsr;
        dms= DMADDR->dmlstat;
        DMADDR->dmcsr= IENABLE|SCENABL;
        spl0();
        return(ret);
}

endit(atp)
{
        extern turnrd();
        register struct tty *tp;

        tp= atp;
        if(((tp->t_state & ENDBIT) == 0) ||
                (tp->t_outq.c_cc > TTLOWAT)){
                        tp->t_state=& ~ENDBIT;
                        return;
        }
        if((tp->t_outq.c_cc != 0) ||
                ((tp->t_state & (WR_ON|BUSY|DELAY40|TIMEOUT)) != 0)){
```

```
                        timeout(endit,tp,PAUSE);
                        return;
        }
        tp->t_state=& ~ENDBIT;
        timeout(turnrd,tp,DELTA);
}

turnrd(atp)
{
        register struct tty *tp;

        tp= atp;
        if(((tp->t_state & (WR_ON|BUSY|DELAY40|ENDBIT|TIMEOUT)) == 0) &&
                (tp->t_outq.c_cc == 0)){
                        dmstat(tp,'c',ROSEND,0);
                        tp->t_state=& ~TRAN_ON;
        }
}

hup(atp)
{
        register struct tty *tp;

        tp= atp;
        if((tp->t_state & (SRD_ON|REC_ON|TRAN_ON|WOPEN)) == 0){
                tp->t_state=& ~(ISOPEN|CARR_ON);
                signal(tp,SIGHUP);
                dmstat(tp,'c',CDLEAD,0);
                flushtty(tp);
        }
}
```

```
#
/*
 *      Copyright 1973 Bell Telephone Laboratories Inc
 *       UNIX SUPPORT GROUP MODIFICATION 2.36
 */

/*
 * general TTY subroutines
 */
#include "/usr/sys/param.h"
#include "/usr/sys/systm.h"
#include "/usr/sys/user.h"
#include "/usr/sys/tty.h"
#include "/usr/sys/proc.h"
#include "/usr/sys/inode.h"
#include "/usr/sys/file.h"
#include "/usr/sys/reg.h"
#include "/usr/sys/conf.h"

struct { int int; };
#define PS      0177776

char    maptab[]
{
        000,000,000,000,004,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,'|',000,'#',000,000,000,'`',
        '{','}',000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        '@',000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,000,000,
        000,000,000,000,000,000,'~',000,
        000,'A','B','C','D','E','F','G',
        'H','I','J','K','L','M','N','O',
        'P','Q','R','S','T','U','V','W',
        'X','Y','Z',000,000,000,000,000,
};

struct cblock {
        struct cblock *c_next;
        char info[6];
};

struct cblock cfree[NCLIST];
struct cblock *ctreelist;

struct {
        int ttrcsr;
        int ttrbuf;
        int tttcsr;
        int tttbuf;
};
```

```
gtty()
{
        sgtty(&u.u_arg[0]);
}

stty()
{

        sgtty(0);
}

sgtty(v)
{
        register struct file *fp;
        register int *up;
        register struct inode *ip;
        int     words;
        int     count;

        up = u.u_arg[0];

        if(v == 0)   {
        u.u_arg[0] = fuword(up++);
        u.u_arg[1] = fuword(up++);
        u.u_arg[2] = fuword(up);
        up -= 2; /*reset pointer*/
        }
        if((fp = getf(u.u_ar0[R0])) == NULL)
        return;

        ip =fp->f_inode;
        if((ip->i_mode&IFMT) != IFCHR) {
                u.u_error = ENOTTY;
                return;
        }
        words = (*cdevsw[ip->i_addr[0].d_major].d_sgtty)(ip->i_addr[0], v);
        if(u.u_error) return;

        if(words <= 0 && (v != 0)) words = 3; /*default to 3
                                                words of results for gtty */
        for(count = 0;count < words; count++)
                suword(up++,u.u_arg[count]);
}

wflushtty(atp)
struct tty *atp;
{
        register struct tty *tp;

        tp = atp;
        spl5();
        while (tp->t_outq.c_cc)
                sleep(&tp->t_outq, TTOPRI);
        flushtty(tp);
        spl0();
```

```
}

cinit()
{
        register int ccp;
        register struct cblock *cp;
        register struct cdevsw *cdp;

        ccp = cfree;
        for (cp=(ccp+07)&~07; cp <= &cfree[NCLIST-1]; cp++) {
                cp->c_next = cfreelist;
                cfreelist = cp;
        }
        ccp = 0;
        for(cdp = cdevsw; cdp->d_open; cdp++)
                ccp++;
        nchrdev = ccp;
}

/*
 * flush all TTY queues
 */
flushtty(atp)
struct tty *atp;
{
        register struct tty *tp;
        register int sps;

        tp = atp;
        while (getc(&tp->t_canq) >= 0);
        while (getc(&tp->t_outq) >= 0);
        wakeup(&tp->t_rawq);
        wakeup(&tp->t_outq);
        sps = PS->int;
        spl5();
        while (getc(&tp->t_rawq) >= 0);
        tp->t_delct = 0;
        PS->int = sps;
}

/*
 * transfer raw list to canonical list,
 * doing code conversion.
 */
canon(atp)
struct tty *atp;
{
        register char *bp;
        char *bp1;
        register struct tty *tp;
        register int c;
        extern dmstat();

        tp = atp;
        spl5();
        while (tp->t_delct==0) {
```

```
                if ((tp->t_state&CARR_ON)==0)
                        return(0);
                sleep(&tp->t_rawq, TTIPRI);
        }
        spl0();
loop:
        bp = &canonb[2];
        while ((c=getc(&tp->t_rawq)) >= 0) {
                if (c==0377) {
                        tp->t_delct--;
                        break;
                }
                if ((tp->t_flags&RAW)==0) {
                        if (bp[-1]!='\\') {
                                if (c==CERASE) {
                                        if (bp > &canonb[2])
                                                bp--;
                                        continue;
                                }
                                if (c==CKILL)
                                        goto loop;
                                if (c==CEOT || c==CETX || c==CDC2)
                                        continue;
                        } else
                        if (maptab[c] && (maptab[c]==c || (tp->t_flags&LCASE))) {
                                if (bp[-2] != '\\')
                                        c = maptab[c];
                                bp--;
                        }
                }
                *bp++ = c;
                if (bp>=canonb+CANBSIZ)
                        break;
        }
        bp1 = bp;
        bp = &canonb[2];
        c = &tp->t_canq;
        while (bp<bp1)
                putc(*bp++, c);
        if(((tp->t_state & STDBIT) != 0) &&
                (tp->t_rawq.c_cc < TTLOWAT)){
                        tp->t_state=& ~STDBIT;
                        spl5();
                        if(tp->t_state & REC_ON)
                                dmstat(tp,'s',SUPTD,0);
                        spl0();
        }
        return(1);
}

/*
 * place a character on raw TTY input queue, putting in delimiters
 * and waking up top half as needed.
 * also echo if required.
 */
ttyinput(ac, atp)
```

Clist control

Sep 19 15:05 tty.c Page 5

```
struct tty *atp;
{
        register int t_flags, c;
        register struct tty *tp;

        tp = atp;
        c = ac;
        t_flags = tp->t_flags;
        if ((c &= 0177) == '\r' && t_flags&CRMOD)
                c = '\n';
        if ((t_flags&RAW)==0 && (c==tp->t_quit || c==tp->t_intrup)) {
                signal(tp, c==tp->t_intrup? SIGINT:SIGQIT);
                flushtty(tp);
                return;
        }
        if (tp->t_rawq.c_cc>=TTYHOG) {
                flushtty(tp);
                return;
        }
        if (t_flags&LCASE && c>='A' && c<='Z')
                c += 'a'-'A';
        tp->t_state&= ~WAK377;
        putc(c, &tp->t_rawq);
        if(((t_flags & BLOCK) == 0) ||
           ((t_flags & MOD40) == 0)){
                if(t_flags&RAW || c=='\n' || c==0377 ||
                   c==CEOT || c==CETX || c==CDC4){
                        if (putc(0377, &tp->t_rawq)==0){
                                tp->t_state|= WAK377;
                                tp->t_delct++;
                        }
                        wakeup(&tp->t_rawq);
                }
        }
        if (t_flags&ECHO) {
                ttyoutput(c, tp);
                ttstart(tp);
        }
}

/*
 * put character on TTY output queue, adding delays,
 * expanding tabs, and handling the CR/NL bit.
 */
ttyoutput(ac, tp)
struct tty *tp;
{
        register int c;
        register struct tty *rtp;
        register char *colp;
        int ctype;

        rtp = tp;
        c = ac&0177;
        if (c==004 && (rtp->t_flags&RAW)==0)
                return;
```

Burst Transmissions

```
        if (c=='\t' && rtp->t_flags&XTABS) {
                do
                        ttvoutput(' ', rtp);
                while (rtp->t_col&07);
                return;
        }
        if (rtp->t_flags&LCASE) {
                colp = "({)}!|^~'`";
                while(*colp++)
                        if(c == *colp++) {
                                ttvoutput('\\', rtp);
                                c = colp[-2];
                                break;
                        }
                if ('a'<=c && c<='z')
                        c += 'A' - 'a';
        }
        if (c=='\n' && rtp->t_flags&CRMOD)
                ttvoutput('\r', rtp);
        if (putc(c, &rtp->t_outq))
                return;
        colp= &rtp->t_col;
        if((tp->t_state & DELAY40) !=0){
                ctype= 1;
                switch (c) {
                /* tab clear */
                case '2':
                        ctype=10;
                        break;
                /* clear */
                case 'J':
                        ctype=14;
                        break;
                /* device reset */
                case 'R':
                        ctype=7;
                        *colp= 0;
                        break;
                /* cursor left */
/*
                case '\b':
                        (*colp)--;
                        break;
*/
                /* cursor right */
                case 'C':
                        (*colp)++;
                        break;
                /* cursor tab */
                case '@':
                        ctype=2;
                        *colp=| 07;
                        (*colp)++;
                        break;
                /* line insert or delete */
/*
```

Dataspeed 40 Delays

```
                case 'L':
                case 'M':
                        ctype=20;
*/
                /* cursor return or home */
                case 'G':
                case 'H':
                        *colp= 0;
                }
                putc(ctype|0200,&rtp->t_outq);
                tp->t_state=& ~DELAY40;
                return;
        }
        if( c==033 ) tp->t_state=| DELAY40;
        ctype = partab[c];
        c = 0;
        switch (ctype&077) {

        /* ordinary */
        case 0:
                (*colp)++;

        /* non-printing */
        case 1:
                break;

        /* backspace */
        case 2:
                if (*colp)
                        (*colp)--;
                break;

        /* newline */
        case 3:
                if (*colp)
                        c = max((*colp>>4) + 3, 6);
                *colp = 0;
                break;

        /* tab */
        case 4:
                if ((rtp->t_flags&NTDELAY)==0)
                        c = 1 - (*colp | ~07);
                *colp =| 07;
                (*colp)++;
                break;

        /* vertical motion */
        case 5:
                c = 0177;
                break;

        /* carriage return */
        case 6:
                c = 15;
                *colp = 0;
```

```
        }
        if (c && (rtp->t_flags&NODELAY)==0)
                putc(c|0200, &rtp->t_outq);
}

ttrstrt(atp)
{
        register struct tty *tp;

        tp = atp;
        tp->t_state &= ~TIMEOUT;
        ttstart(tp);
}

ttstart(atp)
struct tty *atp;
{
        register int *addr, c;
        register struct tty *tp;
        struct { int (*func)(); };

        tp = atp;
        addr = tp->t_addr;
        if (tp->t_state&SSTART) {
                (*addr.func)(tp);
                return;
        }
        if ((addr->tttcsr&DONE)==0 || tp->t_state&TIMEOUT)
                return;
        if ((c=getc(&tp->t_outq)) >= 0) {
                if (c<=0177)
                        addr->tttbuf = c | (partab[c]&0200);
                else {
                        timeout(ttrstrt, tp, c&0177);
                        tp->t_state =| TIMEOUT;
                }
        }
}

ttread(atp)
struct tty *atp;
{
        register struct tty *tp;

        tp = atp;
        if (tp->t_canq.c_cc || canon(tp))
                while (tp->t_canq.c_cc && passc(getc(&tp->t_canq))>=0);
}

ttwrite(atp)
struct tty *atp;
{
        register struct tty *tp;
        register int c;

        tp = atp;
```

```
        while ((c=cpass())>=0) {
                spl5();
                while (tp->t_outq.c_cc > TTHIWAT) {
                        ttstart(tp);
                        sleep(&tp->t_outq, TTOPRI);
                }
                spl0();
                ttyoutput(c, tp);
        }
        ttstart(tp);
}
```