UNCS 1979

Bell Laboratories

subject: Using UNIX Capabilities More Effectively.
File: 38957-32

date: Jan. 30, 1979

from: W. J. Mayer
HO 5113
20624 x2306
5113-790130.01MF

ABSTRACT

A new UNIX program is introduced in this memo that allows convenient transfer of data between UNIX and non-UNIX time shared systems. Although standard UNIX software provides communications with other UNIX systems and with some other BTL batch processors, direct data transfers with non-UNIX time shared systems have not previously been possible in a suitably convenient manner.

This new program makes it feasible to complement the capabilities of other systems with the capabilities that are unique to UNIX. UNIX can now be used to prepare source data for another system, run a program with the prepared input and have the output sent back to UNIX for further processing. Or, UNIX can act as the "middle man" to transfer data, with or without editing, between two other systems that may have no other convenient means of communications.

The program is currently being used as a link between EISS (Economic Impact Study System) and the IBM batch processing (through UNIX) and as a link between UNIX and TASP (Toll Alternatives Studies Program) at the AT&T, Piscataway computer facility.

Copy to
(Without att. III)
All Supervision Center 511
W. J. Bebio
J. R. Hackett
T. T. Lee
R. A. Norden
W. M. Rees
T. L. Russell

Bell Laboratories

subject: Using UNIX Capabilities More Effectively.
File: 38857-32

date: Jan. 30, 1979

from: W. J. Mayer
MO 5113
2D624 x2306
5113-790130.01MF

MEMORANDUM FOR FILE

1. Introduction

PWB/UNIX's greatest contribution as a computer tool, so far, stems from its ability to handle text. Whether that text is a source program, a company memorandum or program data, operations such as sorting, formating, editing and searching are far more conveniently done in UNIX then any other system available to us. Even more power comes from the capability of easily forming useful combinations of these and other common operations.

To apply this capability effectively it is necessary to be able to communicate with equal convenience between our UNIX systems and other computer systems that contain the programs and data bases we use in our work. Some communications capability is provided by the UNIX system. Programs such as cu(I), uucp(I) and usend(I)[1] allow data transfers between two UNIX systems. Another program, send(I)[1], provides for RJE (remote job entry)[2] to the IBM and UNIVAC batch processing systems, and complementary facilities[2] provide a means of sending job output back to UNIX. A fairly recent UNIX bulletin shows how to use the RJE facilities to perform numerous data transfer functions between ASP and UNIX (e.g. sending a data set from ASP into UNIX, loading a card deck through ASP into UNIX, etc). The UNIX programs opr(I) and opu(I) provide a simple way to get printed or punched output using the RJE facilities.

The gap that remains is in the ability to communicate between UNIX and non-UNIX time shared systems. For example, EISS (Economic Impact Study System) is a standard economic tool at Bell Labs that is installed on a UNIVAC system at Murray Hill and accessed on a time shared basis. Another program, TASP, is a toll network planning tool that is resident on the AT&T Comptrollers time shared system (IBM 370/VMS) at Piscataway, N.J. Using either of these tools requires tedious preparation of input data, some of which is output from other programs. This job can be done more efficiently on UNIX then on either host computer. These and other situations prescribe the need for a facility to communicate between UNIX and non-UNIX time shared systems.

This memo introduces the program co (Call Out from UNIX) that allows a UNIX user to call any other time shared system through UNIX and interact with it as if directly connected or cause data to be transfered between systems in either direction. A typical application of this capability might be:

1. Run a batch job on ASP and transfer its output to UNIX;
2. Edit it in UNIX into a format acceptable to EISS;
3. Use co to send it as input for an EISS run on UNIVAC and return the output to UNIX;
4. Edit the output on UNIX into a form acceptable for a final report.

Thus, the capabilities of three computer systems are conveniently combined with co and RJE to do a more complete job. Reference 4 outlines another typical application of co to use TASP.

2. Equipment Configuration and Usage

A peripheral referred to as an ACU (Western Electric 801C4 Auto Call Unit) and a line unit (Western Electric 103A3 data set) make it possible for UNIX to dial out to another computer and appear as a calling terminal. Reference 5 describes these in considerable detail. Each Holmdel Computer Center UNIX System is equipped with two of these outgoing ports. They are treated as special files in UNIX, as are all peripherals. The ACU is used only to establish the connection. The data set is then used to send and receive data. It has the same program interface as does an incoming tty port.

A command to execute co with an argument giving the phone number of the system to be called will establish the configuration shown in figure 1. When the synchronization protocol is complete, co will inform the user by sending him the message "connected". He may then communicate directly with the far end as though he had dialed directly from his terminal. By directing commands to each system he can cause data to be transfered between them or cause programs to be executed on either or both. Finally he can give a command to terminate the co session, leaving him connected only to UNIX in the usual manner.

Numerous control sequences are available to do the desired operations outlined above. These are specified in the program description (Attachment I) and the users guide (Attachment II). Attachment III is a copy of the program source listing.

3. Debugging Status

It has not been possible yet to check exhaustively all features of this program. Some bugs can, and probably do exist. However, co has been applied in real situations by three different users working with three different remote systems in addition to UNIX as a remote system. It is believed that this has provided sufficient checks for at least the main modes of co. Any problems should be brought to the attention of the author.

4. Future Features

It is not anticipated that any further development will take place on this program. However, there are some features that should be considered if it is decided at some time to provide a new version of the program. The most significant of these would be the capability to schedule one or more data transfers to take place when ACU lines are available, as background jobs, so that the user does not have to wait on line for transfers to complete.

5. Acknowledgements

Several acknowledgements are in order. First of all, this program was built on to the existing cu program in UNIX and some parts of that code were carried over into co with little or no change. Secondly, the routine used to prepare regular expressions for regcmp(III) was written by W. M. Rees. And finally, W. M. Rees and W. J. Beblo served as friendly users and helped with debugging.

W. J. Mayer

HO-5113-WJM

Atts.
References
Figure 1
Attachments I, II & III

REFERENCES

1. T.A.Dolotta, R.C.Haight, E.M.Piskorik (Editors), PWB/UNIX User's Manual, Edition 1.0, May 1977. See also update packages for editions 1.1 and 1.2.

2. A.L.Sabsevitz, Guide to IBM Remote Job Entry for PWB/UNIX Users, UNIX number 1087.

3. B.R.Davies, Using the RJE Link Between ASP and UNIX, UNIX Information Bulletin number 78-04.

4. W.J.Mayer, Economic Analysis of Metro-STP Alternatives -- Study Tool Implementation, Memo for File, File 39924-29, 5113-781030.01MF.

5. G.B.Foley, A description of Automatic Dialing Using The UNIX Operating System, Memo for File, File 38931, UNIX number 1180, 5133-770801.01MF.

6. J.R.Mashey, PWB/UNIX Shell Tutorial, UNIX Number 1189.

7. S.R.Bourne, An Introduction to the UNIX Shell, UNIX number 1239.

8. B.W.Kernighan, A Tutorial Introduction to the UNIX Text Editor, UNIX number 1000.

9. B.W.Kernighan, D.M.Ritchie, The C Programming Language, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1978. (Page 181)

ISSUE

ENGR. WSM
DRAWN 1-79

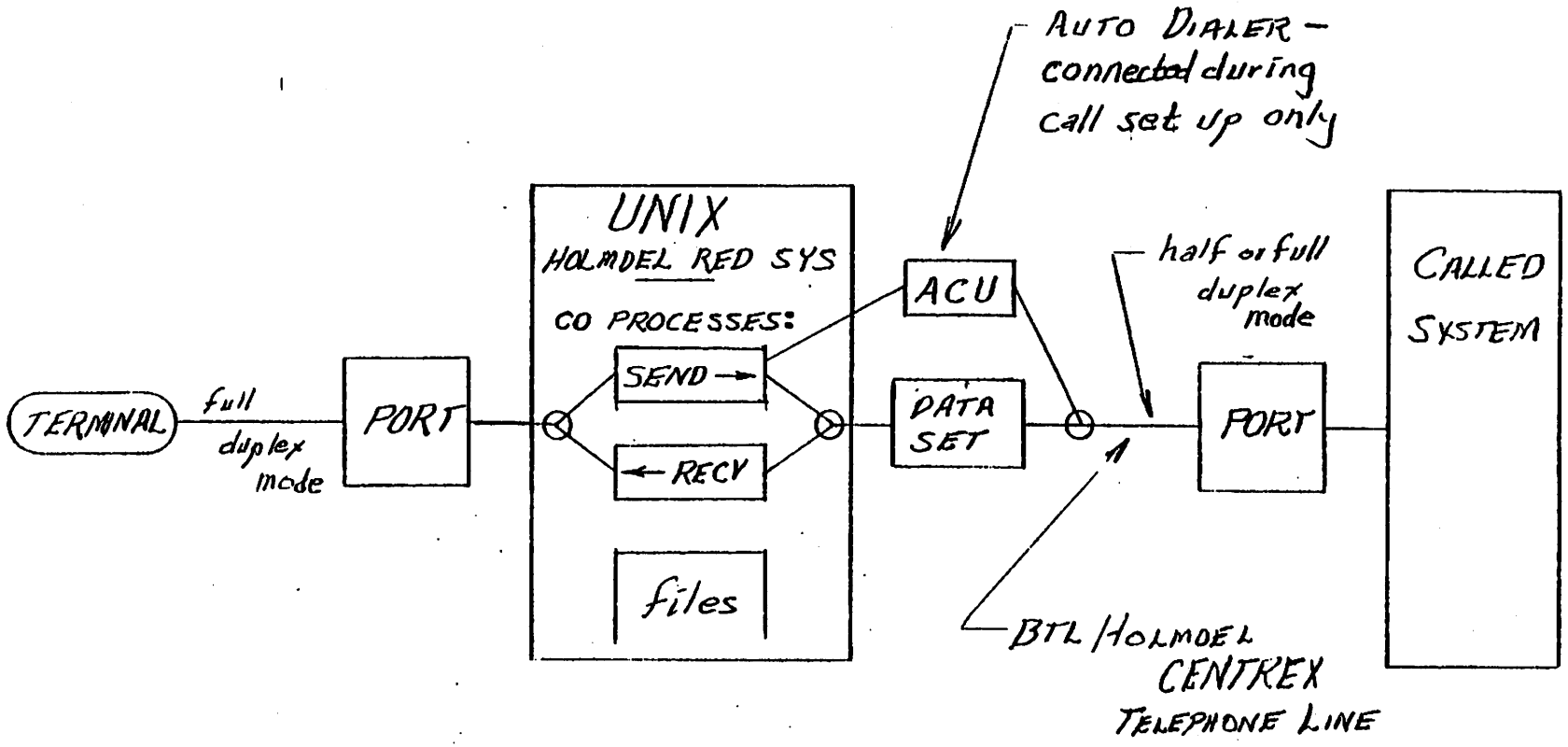
TITLE

FIGURE 1

BELL LABORATORIES

NO. OF SHEETS PER SET

SHEET



EQUIPMENT CONFIGURATION

FOR AN ESTABLISHED CALL

NAME

co - call out from UNIX

SYNOPSIS

co telno [-e | -o] [-s speed] [-l line] [-a acu] [-p "reg-expr"] [-f] [-uc]

DESCRIPTION

Co may be used to call another time shared computer system through UNIX and interact with it while UNIX is in a transparent mode. It may alternately be used to transfer data in either direction between UNIX and the system called. This is an extended version of the UNIX system program cu(I) - Call UNIX. The extensions allow transfer of data with non-UNIX systems that have differing protocols. No cooperating programs are required on the called system.

The command "co 9-5551234" causes UNIX to place a call using its ACU (auto call unit) to the number entered as the first argument. Dashes in the number cause a short delay in dialing and are used to wait for second dial tone. If the call completes successfully, the message "connected" will be returned to the user. He may then send data through his keyboard in the same manner that he would if he had dialed directly from his terminal.

Other arguments in the calling sequence are optional and normally not needed. Parity required by the called system may be specified with -e for even or -o for odd. Normally a zero parity bit is transmitted by UNIX. A few time share systems require even parity. The speed is 300 baud by default but may be set to any other speed for which the UNIX system is equipped. The line and acu are by default /dev/cul0 and /dev/cua0 and need to be specified only if some other units are to be used. The regular expression is used to describe the prompt sequence that will be sent back from the called system (see discussions below for the ~p and ~< filnam commands). If the called system is full duplex and echos tty input, the -f command can be used to prevent double echo. The -uc argument will cause all letters sent to be shifted to upper case.

Once connected, the user may precede typed lines entered with the symbol ~ if he wishes them to be interpreted by co as commands rather than transmitted to the called system. Lines not starting with ~ are sent through to the called system. (Lines starting with ~ ~ are sent through after deleting the first tilda character.) The following are commands that are understood by co:

Transfer Data Into UNIX

~> filnam	Copy data received from the called system to filnam in the users current working directory. (A full path name may be alternatively specified.)
~>: filnam	As above but with no copy to the terminal. (Each line transfered to the file is indicated by forward and backward motion of the print head or cursor.)
~>> filnam	As above, but append to the existing specified file.
~>>: filnam	As above but with no output to terminal.
~>	Stop copy to file.
~>:	Stop copy to terminal.

Transfer Data Out of UNIX

~< filnam — Send the specified file in the users' current working directory (or the specified path name file) to the called system. By default the file is sent at the line rate without any intentional pauses. If the called system can only accept the data a line-at-a-time, the user must first specify a prompt character that will indicate "ready for another line". For example, if the called system sends the symbol > to prompt for another line of input, the user must, before entering this command, enter the command ~p />/ or have entered -p '>' as an argument to co. Each line transfered is signaled to the user by motion of his print head. No copy is made on the terminal.

~< — Stop sending file. Normally this action occurs when an end of file is reached. This command can be used to prematurely stop a transfer. The stop always occurs at the end of a line.

UNIX to UNIX File Transfers

~%take from [to] — Copy the file 'from' on the remote system to the file 'to' on the local system. If the to file is omitted, 'from' is used for both systems.

~%put from [to] — Copy the file 'from' on the local system to the file 'to' on the remote system. If the to file is omitted, 'from' is used for both systems.

Execute Programs Locally

~!pgm args — The indicated program is executed on the local UNIX system with the results sent to the terminal. Nothing is sent to the called system. Any data coming from the called system at the same time will be intermixed in an unpredictable manner.

~$pgm args — Same as above but output from the program is sent to the called system instead of the terminal. Data from the called system is available as standard input to the program.

~! — This command invokes an interactive shell on the local UNIX system. It can be terminated by sending EOT (control-d). It is most useful as a means of doing other work while a long transfer is taking place.

Co Control Parameters

Most of these commands duplicate the calling arguments. They are provided as commands so that a function controlled by them may be modified during a call.

~p /reg-expr/ — A prompt sequence that may be needed by the above ~< filnam command may be specified with this command if it was not entered with the -p argument in the calling sequence or if it changes during a call. The use of a regular expression* allows reasonably general specification of any prompt sequence.

The command ~p, without a regular expression, will cause the

* Regular expression is meant in the usual UNIX sense. See ed(I) or regcmp(I) in the PWB manual for an explanation on how to use them.

currently held regular expression to be displayed.

~e This command forces even parity characters to be sent to the called system.

~o Forces odd parity.

~eo Causes parity bit to be zero always (default).

~f Indicates that the other system is full duplex and echos received data. Co will suppress the echo generated locally so that the tty will not get a double echo.

~h Indicates that the called system is half duplex. Echo will be provided locally. (Default).

~uc All alphas sent will be shifted to upper case.

~ul Alphas will be sent with no case shift. (Default).

Line Connect Control

~. Disconnect from called system.

~telno Disconnect from called system and set up a new call to telno.

The above commands may be entered even when data is being transmitted between systems. But, they must always be at the beginning of a line and may not be entered during execution of ~!. If a line to be sent to the called system begins with the symbol ~, an extra ~ must precede it to escape from the co command interpretation.

FILES

/dev/cua0 /dev/cul0

SEE ALSO

cu(I), dh(IV), dn(IV), tty(IV), ln(I), ttys(V)

Using UNIX Capabilities More Effectively, 5113-7901xx.01MF.

This program was created by taking a copy of cu(I) — Call UNIX — and adding the necessary features to make it communicate with systems other than UNIX. An attempt was made to retain all the features already in cu. The description for cu(I) should be consulted for a description of those features.

BUGS

See cu(I) BUGS.

Attachment II
5113-790130.01MF

USERS GUIDE

co - Call Out from UNIX

1. Introduction

This guide provides general usage information for co. A more complete and concise specification of program usage is contained in 5113-790130.01MF-Att I.

2. Calling Sequence and Program Termination

The telephone lines used by co, those attached to the Holmdel UNIX System outgoing ports, are connected to the BTL-Holmdel Centrex, not the Dimension PBX as might be suspected. Thus, the telephone number used should be prefixed with an 8 for CORNET, a 9 for DDD, etc.

An executable version of the program is currently in the Red System directory

/R2/5113wjm/pgm

so the complete calling sequence would be

/R2/5113wjm/pgm/co 9-5554340

Of course, a Shell procedure [6,7] in ones own login directory should be used to reduce this cumbersome string to a more convenient sequence. Dashes should be used in the telephone number wherever a delay for second dial tone is appropriate. Other arguments to the command are discussed in the following sections.

When a call to the specified number is completed successfully, the message "connected" is written on the users terminal. He may then communicate directly with the called system (e.g. with a login sequence) as though he had dialed directly. The configuration can be logically represented by figure II-1. The output from each node can go to either or, in some cases, both of its two outgoing links. Normally the user directs each of his input lines to one of the other two nodes by preceding those going to co with the symbol ~. Other lines will go to the called

system. The user controls the direction of the output from the other two nodes by commands that are sent to co.

Skipping ahead for a moment, the program may be terminated by entering

~.

which instructs co to disconnect from the called system and exit. If it is desired to terminate before the initial connection is made, the break key may be used.

3. Parity

UNIX will normally ignore the parity of incoming data. For outgoing data the parity bit will be set to zero. Some time shared systems* require their users to send even parity data. This can be done by typing the argument -e (minus e) after the telephone number (leaving a space between arguments).

4. Data Transfers

The user arranges data transfers by first preparing the receiving end to receive and store data and then causing the sending end to send the data to be transfered. The exact sequence depends on what commands are used by the called system. We will assume that there is a command "list filname" and show how it is used to transfer data into UNIX. The command

~> savefile

is sent by the user to instruct co to put all data received from the line into the file named 'savefile'. If a file by this name does not exist in the current working directory, it will be created by co. The command:

list filname

is then sent to cause the called system to list the file. The output will go to the terminal as well as 'savefile'. When the listing is complete, the command:

~>

is sent to stop the process of storing received data in 'savefile'. This completes the transfer.

---

* An example is the system called "MUSIC", used by Long Lines Engineering.

Several variations of the co command allow the transfer to take place with or without listing on the terminal and determine whether the data append to the named file or overwrite it. These variations are shown in Attachment I.

Suppose now that 'savefile' has been modified in UNIX and it is desired to send it back. We will assume that the called system has a command "input" that causes it to collect incoming data in a temporary file. So, the commands:

```
input
~< savefile
```

will prepare it to receive data and cause co to send it. The file will not be listed on the terminal but, each time a line is sent a space-backspace sequence will be sent so that the user can monitor progress of the transfer. When the transfer is completed, that is, when an end of file is reached, a message to this effect is sent to the terminal.

A transfer can be terminated early by the user entering

```
~<
```

This is not a normal operation but is sometimes necessary when one realizes he has made some error and is sending the wrong data. Termination of the transfer always takes place at the end of a line.

When the transfer is complete it is necessary to inform the called system of this and possibly instruct it to save the data in a permanent file. A typical command sequence would be:

```
(blank line)
save filname
```

These two commands could, if desired, be incorporated as the last two lines of the transmitted file.

One other important function may have to be provided for when transfering to the called system. Many systems will accept data only one line at a time. That is, there is a "dead period" between the carriage return and when the prompt is sent for the next line. Anything sent during this period will be lost. With such a system, the user must advise co of the prompt sequence.

By default co will not delay at all between lines. However, if the argument -p '%' is used in the calling sequence co will wait for a % to be returned after sending each line. Any character or sequence of characters may be specified for the prompt. In fact, the UNIX convention for "regular-expressions"* may be

* Reference 8 explains how to use regular expressions.

used for a more general specification of the prompt sequence and the C Language conventions[9] may be used for special characters. Some examples:

```
-p ">:"
-p "\r"
-p "[.]"
```

The second example specified a carriage return to be the prompt. The third example specifies a period. The brackets are necessary because a period is a special character to the regular expression routine.

The prompt sequence can be specified or changed after entering co by using the command

```
~p />:/
```

Note that in this case, the slash is used to delimit the expression rather than the quotes used in the calling sequence arguments.

Data transfers with other UNIX systems can be done using the above methods and with the prompt sequence at its default value (~p //). However, the commands:

```
~%take from [to]
~%put  from [to]
```

which are carried over, with some improvements, from the cu program are intended specifically for UNIX to UNIX transfers and may be the best choice in these cases. Their use is explained in Attachment I.

5. Local Programs

It is possible to execute other programs on the local UNIX system without terminating the co program. This may be done while the called system waits or, in some cases, while a transfer is taking place in the background. Several methods are available to do this. The simplest is to precede any valid UNIX command with "~!". For example,

```
~!ls
```

would cause a list of files in the current working directory to be made at the terminal.

Conflicts with ~! initiated programs and data transfers occur when both are writing on the users terminal. The result is

---

Regex(III) in [1] defines what expressions may be used.

a scrambled looking output. This can be easily avoided by redirecting the ~!program's output to a file or by preventing the file transfer from copying to the terminal. It is often convenient to use redirected I/O together with the ampersand operator. For example:

```
~!(ps >ps.out; echo "BEL")&
```

(where BEL is the control character for the terminal bell) will run ps (process status) as an independent background job and signal the user when it is done. He can then print ps.out to get the needed information. In the meantime, his terminal has not been tied up waiting for the results of a slow running program.

Another procedure is to enter just "~!" on a line by itself. This will cause a conversational shell to be created for the user. He may then enter commands as he normally would. When finished, and EOT (control-d) will terminate the shell and return him to the waiting co program.

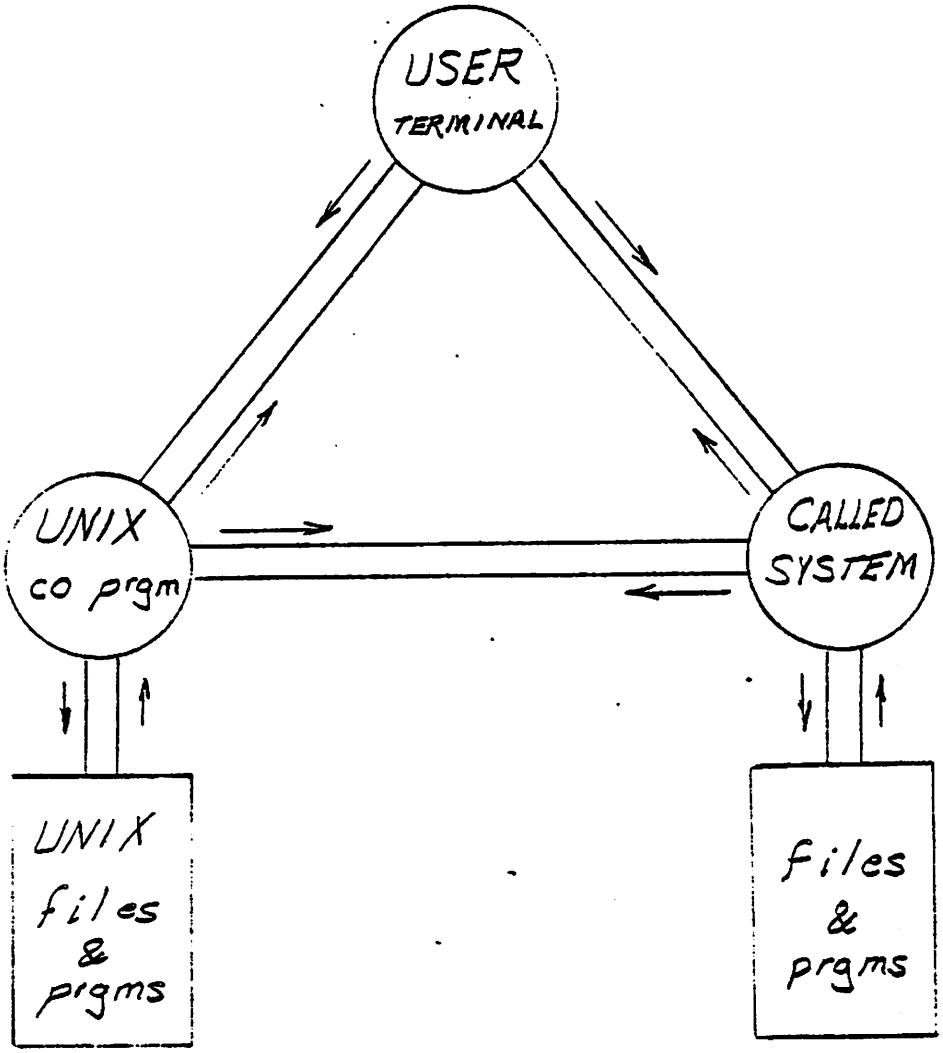
6. Case Shift and Echo

UNIX is usually used with the terminal switches set at full duplex and upper/lower case. Many other systems require the half duplex and upper case settings. If the user tells co what the called system requires, then co can do the mode switching and the user need not be concerned with it as he sends commands alternately to UNIX and the other system.

Co will assume that the other system is half duplex and accepts both upper and lower case. The action can be changed with the command ~uc to send only upper case and ~f to provide for full duplex with echo at the other end (e.g., another UNIX system). The corresponding arguments -uc and -f can be used, instead, when the call is initiated. The commands ~ul and ~h can be used to return to the default settings.

WJM 1/79

LOGICAL CONFIGURATION
OF AN ESTABLISHED CALL



ISSUE	ENGR WJM	TITLE	BELL LABORATORIES
	DRAWN 1-79	FIGURE II-1	SHEET
			NO. OF SHEETS PER SET

PRINTED IN U.S.A. E-1812-A-3 (10-71)