Title
RAW QIO INTERFACE TO EUNICE TCP CIRCUITS.

Permalink
https://escholarship.org/uc/item/3hf6519v

Author
Sventek, J.S.

Publication Date
1984-04-01
RAW QIO INTERFACE TO EUNICE TCP CIRCUITS

J.S. Sventek

April 1984

TWO-WEEK LOAN COPY
This is a Library Circulating Copy which may be borrowed for two weeks. For a personal retention copy, call Tech. Info. Division, Ext. 6782.

Prepared for the U.S. Department of Energy under Contract DE-AC03-76SF00098
DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.
Raw QIO Interface to Eunice TCP Circuits

Joseph S. Sventek

Computer Science Research Department
University of California
Lawrence Berkeley Laboratory
Berkeley, California 94720

April 1984

This work was supported by the Applied Mathematical Sciences Research subprogram of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.
Raw QIO Interface
to
Eunice TCP Circuits

Joseph S. Sventek
Computer Science Research Department
Computing Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

Abstract

This document describes the raw QIO interface to TCP circuits provided by the Eunice TCP/IP networking code. Example sections of code (in FORTRAN) are also provided to aid others writing network applications.

Disclaimer

No guarantees of the accuracy of this documentation is implied or expressed, nor is any support for the example programs implied or expressed. Written notification of problems WITH fixes will be accepted and incorporated in future versions of this document.

Acknowledgement

This work was supported by the Applied Mathematical Sciences Research sub-program of the Office of Energy Research, U.S. Department of Energy under contract DE-AC03-76SF00098.
1. Introduction

This document describes the raw QIO interface to TCP circuits provided by the Eunice TCP/IP network software. Before describing the details of the QIO interface, it is necessary to present some introductory information on the general ideas behind the network software.

The fundamental object over which network communication takes place is called a socket. Immediately after creation, a socket merely represents an endpoint for future communication within a particular address family. Since we are concentrating on TCP, the address family will always be the ARPA internet family.

Before cooperating processes can begin to communicate, the socket must be bound to an address within the selected address family.

The actual rendezvous between cooperating processes is accomplished by an active connect by the client, and a passive accept by the server process. Upon completion of these calls in the two processes, a full-duplex, flow-controlled, reliable and sequenced TCP circuit is in place. At this point, the two processes may perform send/recv system calls in the appropriate manner to accomplish their joint mission.
2. Normal Scenarios for Distributed Applications

As you might have discerned from the above discussion, a specific sequence of operations is necessary in order to establish a TCP circuit between two processes. The sequence is different for the client process and the server process. The appropriate sequence for each process is outlined below:

Client

* create a socket
* connect to appropriate TCP port on server machine
* send/receive according to protocol between the cooperating processes
* remove the socket

Server

* create a socket
* bind the socket to the appropriate internet address and port
* listen for connects
* accept connect request from client
* send/receive according to protocol between the cooperating processes
* remove the socket
3. Interface Specifics

This section describes those VMS system calls and machinations necessary to perform each of the operations described above. Before describing each of the specific routine interfaces, a few words on the data structures used are in order.

The main data structure which you will see in the C language client and server programs provided with the network system is `sockaddr_in` which consists of the following parts:

* a 16-bit integer in which the address family is kept.
* a 16-bit integer in which the port number of the server is kept (in network order).
* a 4-byte array in which the internet address of the server's host (in network order) is placed by the client before the connect call.
* an 8-byte dummy array.

This data structure will be called $S$ in the descriptions, and these locations will be referenced as $S.af$, $S.port$ and $S.inetaddr$ respectively.

The only other complex data structure used by the system is that used to receive the address of the client when the server accepts the connect request. That structure consists of one 16-bit integer followed by a 128-byte array. The 16-bit integer will contain the number of bytes in the 128-byte array used to return the address of the accepted client. This data structure will be called $R$ in the descriptions below.

In the descriptions which follow, symbolic constants are used. Please consult section 4.1 for their actual values.

In all cases, successful completion of the request will be indicated with the low bit set, for either the function value of the SYS$GIZORK call, or in the first word of the io status block. Eunice specific errors are returned in the iosb with the high bit enabled, and contain the UNIX error number shifted left 3 bits. To decrypt the error returned in iosb(1), the following algorithm should be used:

```plaintext
err = '7fff'x .and. iosb(1)
err = err / 8
```

Now compare the value in `err` with those listed in `/usr/include/errno.h`. The attached routine "eunice_error" shows one way in which these values can be turned into printable strings.
3.1. Creating a Socket

(1) assign a channel to the device 'INET0:

(2) issue qiw request on socket channel with the following parameters

* function = IO$_SOCKET
* p1 = %val(AF_INET)
* p2 = %val(SOCK_STREAM)
* p3 = p4 = p5 = p6 = 0

3.2. Binding the Socket to an Address

(1) place the value AF_INET into S.af

(2) place the port number into S.port in network order (byte swapped) and load S.inetaddr with the address 0.

(3) issue qiw request on socket channel with the following parameters

* function = IO$_BIND
* p1 = %ref(S)
* p2 = %val(16)
* p3 = p4 = p5 = p6 = 0

3.3. Listen for Connect Requests

(1) issue qiw request on socket channel with the following parameters

* function = IO$_LISTEN
* p1 = %val(backlog)
* p2 = p3 = p4 = p5 = p6 = 0

The 'backlog' parameter passed in p1 indicates how many incoming connect requests the process wishes to be queued up while servicing an accepted connection.

Note that the listen completes immediately, since it simply indicates to the system that your process wishes to process connections to the specified port. The process is actually blocked when it executes the accept request described below.
3.4. **Accept a Connect Request**

(1) issue qiow request on socket channel with the following parameters

* func = IO$\_ACCEPT\_WAIT
* p1 = p2 = p3 = p4 = p5 = p6 = 0

This causes the process to block until an incoming connect request is received.

(2) assign a new channel to 'INET0:'

(3) issue qiow request on new socket channel with the following parameters

* func = IO$\_ACCEPT
* p1 = %ref(R)
* p2 = %val(130)
* p3 = %val(original socket channel)
* p4 = p5 = p6 = 0

Setting p1 and p2 to non-zero values is optional, with the only required parameter being p3.

3.5. **Receiving Packets over the Circuit**

(1) issue qiow request on socket channel used in the accept of connect requests, with the following parameters

* func = IO$\_RECEIVE
* p1 = %ref(buffer to receive next packet)
* p2 = %val(sizeof(buffer))
* p3 = p4 = p5 = p6 = 0

The length of the received message is returned in iosb(2).

**NOTE:** a successful receive with a length of 0 seems to indicate that the partner has disappeared.
3.6. Transmitting Packets over the Circuit

(1) issue qio request on socket channel used in the accept of connect requests, with the following parameters

* func = IO$_\$SEND
* p1 = %ref(buffer with data to send)
* p2 = %val(number of bytes to send)
* p3 = p4 = p5 = p6 = 0

3.7. Initiate a Connect Request

(1) place the value of the port number for the connection into S.port

(2) place the value of the internet address of the server machine into S.inetaddr (see section on address resolution below)

(3) issue qio request on socket channel with the following parameters

* func = IO$_\$CONNECT
* p1 = %ref(S)
* p2 = %val(16)
* p3 = p4 = p5 = p6 = 0
3.8. Address Resolution

If we consider the internet address [first.second.third.fourth] and that the structure $S$ is a 16-byte array, the following must be done prior to issuing the connect request in the client process:

\[
\begin{align*}
S(5) &= \text{first} \\
S(6) &= \text{second} \\
S(7) &= \text{third} \\
S(8) &= \text{fourth}
\end{align*}
\]

Of course, one often has the name of the host, not its internet address. The binding of internet address to hostnames and nicknames is contained in the file etc:hosts. The format of the file is as follows:

(1) Lines beginning with the character '//' are comments.

(2) A '//' character in any other position in a line indicates the start of a comment, and is thus the logical end-of-line.

(3) The information binding internet addresses to names is of the form

\[
111.222.333.444 \text{ official-name[ nickname]*}
\]

Section 4.4.4 contains the FORTRAN source code which will sequentially scan etc:hosts for a particular host name and return the internet address in the appropriate format for inclusion in the $S$ data structure.
4. Sample Programs

The following two sections present the FORTRAN code for a sample client and server.

The server listens for a connection on port 4321. After successfully accepting a connect request, it simply receives buffers from the link until the received byte count goes to 0, indicating that the client has exited. It then waits for another connect request.

The client, when defined as a foreign DCL command, fetches the hostname from the command line for the server connection. It also will take optional values from the command line for repeat count and buffer size. After successfully connecting to the server, it sends <buffer size> buffers <repeat count> times. After closing the connection, the program displays the elapsed time and throughput in bytes/second.

4.1. Include File - INETSYSM.INC

```fortran
integer AF_INET
parameter (AF_INET=2)
integer SOCK_STREAM
parameter (SOCK_STREAM=1)

integer IOS_ACCESS
parameter (IOS_ACCESS='32'x)
integer IOS_READBLK
parameter (IOS_READBLK='31'x)
integer IOS_WRITEBLK
parameter (IOS_WRITEBLK='30'x)

integer IOS_SEND
parameter (IOS_SEND=IOS_WRITEBLK)
integer IOS_RECEIVE
parameter (IOS_RECEIVE=IOS_READBLK)
integer IOS_SOCKET
parameter (IOS_SOCKET=IOS_ACCESS)
integer IOS_BIND
parameter (IOS_BIND=IOS_ACCESS+64)
integer IOS_LISTEN
parameter (IOS_LISTEN=IOS_ACCESS+128)
integer IOS_ACCEPT
parameter (IOS_ACCEPT=IOS_ACCESS+192)
integer IOS_CONNECT
parameter (IOS_CONNECT=IOS_ACCESS+256)
integer IOS_ACCEPT_WAIT
parameter (IOS_ACCEPT_WAIT=IOS_ACCESS+320)
```
4.2. Receiver Process - RECEIVE.FOR

program receive

This code is a TCP receiver using Kashtan's port of the UNIX networking code. It listens on TCP port 4321, accepts a connect request, and receives all data packets until the connection is broken. It then goes back and waits for another connect request.

FORTRAN RECEIVE.FOR
LINK RECEIVE.OBJ
RECEIVE:==SYS$DISK:[THIS.DIRECTORY]RECEIVE
SPAWN/NOWAIT/OUTPUT=RECEIVE.OUT RECEIVE

include 'INETSYM.INC'

integer*4 fd, ad, errlen
integer*2 sys$assign, sys$qiow
character errbuf*256
integer*2 iosb(4), s
integer*2 swap
logical*1 buffer(2048)
logical*4 error

FORTRAN equivalent of sockaddr_in

integer*2 i2buf(8)
logical*1 l1buf(16)
equivalence (i2buf(1), l1buf(1))

assign channel to device and create socket

s = sys$assign('INET:', fd,)
if (error(s, 1, errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
s = sys$qiow(%val(0), %val(fd), %val(IO$ SOCKET), %ref(iosb),
1,, %val(AF_INET), %val(SOCK_STREAM),)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

fill in address family, port # and wild card address.

Bind socket to that address

i2buf(1) = AF_INET
i2buf(2) = swap(4321)
l1buf(5) = 0
l1buf(6) = 0
l1buf(7) = 0
lbuf(8) = 0
s = sys$qiow(%val(0), %val(fd), %val(IO$BIND), %ref(iosb),
1 , , , %ref(lbuf), %val(16), , , ,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

listen on port
s = sys$qiow(%val(0), %val(fd), %val(IO$LISTEN), %ref(iosb),
1 , , %val(1), , , , , , , ,
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

main loop - wait for connect request, accept it and process it
continue

wait for connect request
s = sys$qiow(%val(0), %val(fd), %val(IO$ACCEPT), %ref(iosb),
1 , , , %ref(iosb), , , , , , , ,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

assign new channel to INET8:
s = sys$assign('INET8:', sd, ,)
if (error(s, 1, errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

accept connect request on the new socket
s = sys$qiow(%val(0), %val(sd), %val(IO$ACCEPT), %ref(iosb),
1 , , , %val(fd), , , , , , , ,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif

read on socket until 0 length read - seems to imply
that the partner has exited
continue
s = sys$qiow(%val(0), %val(sd), %val(IO$RECEIVE), %ref(iosb),
1 , , %ref(buffer), %val(2048), , , , ,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
if (iosb(2) .gt. 0) goto 2
call sys$dassign(%val(sd))
goto 1
include 'ERRORX.INC'
include 'SWAB.INC'
include 'ERROR.INC'
include 'EUNICEERR.INC'
4.3. Transmit Process - TRANSMIT.FOR

```
program transmit

This code is a TCP transmitter using Kashtan's port of the UNIX
networking code. It connects to a receiver on port 4321, and
transmits a fixed number of fixed size packets. Upon completion
of the request, the elapsed time and throughput in bytes/second
are displayed.

FORTRAN TRANSMIT.FOR
LINK TRANSMIT.OBJ
TRANSMIT:==SYS\$DISK: [THIS.DIRECTORY] TRANSMIT
TRANSMIT HOST [-REPCNT] [-BUFSIZ]

include 'INETSYM.INC'

integer*4 repcnt, bufsiz, host_len, arg_len, sd, total, err_len
integer*4 start(2), stop(2), result(2), msec, rem, thruput
integer*2 sysassign, sysqow
logical*4 inet_getarg, inet_gethost
logical*1 buffer(2048)
real*4 x
integer*2 iosb(4), s
logical*4 error
integer*2 swab
character arg_buf*40, host*40, errbuf*256

FORTRAN equivalent of sockaddr_in

integer*2 i2buf(8)
logical*1 l1buf(16)

equivalence (i2buf(1), l1buf(1))

fetch command line arguments

repcnt = 1000
bufsz = 512
host_len = 0
do while (inet_getarg(arg_buf, arg_len))
call inet_lower(arg_buf(1:arg_len))
if (arg_buf(1:1).eq. '-') then
  if (arg_buf(2:2).eq. 'b') then
    call ots$cv_t_i_l(arg_buf(3:arg_len), bufsiz)
  elseif (arg_buf(2:2).eq. 'r') then
    call ots$cv_t_i_l(arg_buf(3:arg_len), repcnt)
  else
    type *, arg_buf(1:arg_len), ': invalid argument'
  endif
else
  host_len = arg_len
```

-12-
host = argBuf
endif
enddo
if (host_len .eq. 0) then
  call errorx('usage: transmit [-rrepn] [-bbuf] host')
endif
assign channel to device and create socket
s = sys$assign('INET0:', sd,) if (error(s, 1, errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
s = sys$ioaw(%val(0), %val(sd), %val(IO$ SOCKET), %ref(iosb),
1 , , %val(AF_INET), %val(SOCK_STREAM),,,,,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
fill in destination port and host address. inet_gethost locates
the entry for the specified host in the file ETC:HOSTS,
and returns the internet address in the correct order
i2buf(1) = AF_INET
i2buf(2) = swab(4321)
if (.not. inet_gethost(host(1:host_len), i2buf(5))) then
  call errorx('Unknown host name')
endif
connect to server
s = sys$ioaw(%val(0), %val(sd), %val(IO$ CONNECT), %ref(iosb),
1 , , %ref(i2buf), %val(16),,,,,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
initialize counters and note current system time
send 'repn' buffers of 'bufsz' characters to the server
total = 0
call sys$gettim(%ref(start))
do while (repcnt .gt. 0)
s = sys$ioaw(%val(0), %val(sd), %val(IO$ SEND), %ref(iosb),
1 , %ref(buffer), %val(bufsz),,,,,)
if (error(s, iosb(1), errbuf, errlen)) then
call errorx(errbuf(1:errlen))
endif
total = total + bufsize
repnct = repnct - 1
endo
note system time and output transfer statistics
call sys$gettim(%ref(stop))
call lib$subx(stop, start, result)
call lib$div(10000, result, msec, rem)
    x = float(msec) / 1000.
    thruput = int (float(total) / x)
    type 100, x
    format(1x,f8.3, ' seconds elapsed time')
    type 101, thruput
    format(1x,i8, ' bytes/second throughput')
call sys$dassgn(%val(sd))
call exit
end

include 'LOWER.INC'
include 'GETARG.INC'
include 'GETHOST.INC'
include 'GETWORD.INC'
include 'ERRORX.INC'
include 'SWAB.INC'
include 'ERROR.INC'
include 'EUNICEERR.INC'
4.4. Included Routines

4.4.1. Print error and exit - ERRORX.INC

subroutine errorx(str)
  character*(*) str
  type *, str
  call exit
end

4.4.2. Fold character string to lower case - LOWER.INC

subroutine inet_lower(buf)
  character*(*) buf
  integer n, i, biga, bigz, diff, x

  n = len(buf)
  i = 1
  biga = ichar('A')
  bigz = ichar('Z')
  diff = ichar('a') - biga
  do while (i .le. n)
    x = ichar(buf(i:i))
    if (x .ge. biga .and. x .le. bigz) then
      buf(i:i) = char(x+diff)
    endif
    i = i + 1
  enddo

return
end
4.4.3. Fetch next argument from command string - GETARG.INC

```fortran
logical function inet_getarg(arg_buf, arg_len)

character(*) arg_buf
integer arg_len
logical first
integer force, cmd_len, ind
integer lib$get_foreign, inet_getword
character cmd_buf*256

data first /.true./

if (first) then
    first = .false.
    force = 0
    if (.not. lib$get_foreign(cmd_buf,, cmd_len, force)) then
        cmd_buf = ''
        cmd_len = 1
    elsif (cmd_len .le. 0) then
        cmd_buf = ''
        cmd_len = 1
    endif
    ind = 1
endif
arg_len = inet_getword(cmd_buf(1:cmd_len), ind, arg_buf)
if (arg_len .le. 0) then
    inet_getarg = .false.
else
    inet_getarg = .true.
endif

return
dend
```
4.4.4. Find host in etc:hosts and return inet address - GETHOST.INC

logical function inet_gethost(host, adrrbuf)

character*(*) host
logical*1 adrrbuf(4)
integer*4 lun, hostlen, n, i, adrlen, m, j, k
integer*4 lib$get_lun, inet_getword
character buffer*256, address*40, nicknm*40

integer*4 i4
logical*1 l1

equivalence (l1,i4)

if (.not. lib$get_lun(lun)) then
  inet_gethost = .false.
  return
endif

open (unit=lun, file='ETC:HOSTS.', type='OLD', READONLY, 1 err=10)
hostlen = len(host)
continue
read (lun, 100, end=11) n, buffer
100 format(q, (a))
if (buffer(1:1) .eq. '#') goto 1 have a comment
i = index(buffer(1:n), '#')
if (i .gt. 0) then
  n = i
endif
do 4 i = 1, n
  k = ichar(buffer(i:i))
  if (k .eq. 8) then
    buffer(i:i) = ' ' ! replace tabs by blanks
  endif
4 continue
i = 1
adrlen = inet_getword(buffer(1:n), i, address)
adrlen = adrlen + 1
address(adrlen:adrlen) = '.'
2 continue
m = inet_getword(buffer(1:n), i, nicknm)
if (m .le. 0) goto 1
if (m .ne. hostlen) goto 2
if (nicknm(1:m) .ne. host(1:m)) goto 2
close(unit = lun)
call lib$free_lun(lun)
i = 1
continue
3 j = 1, 4
  k = i + index(address(i:adrlen), '.') - 2
call ots$cvt_ti_i(address(i:k), i4)
adrrbuf(j) = i1
  i = k + 2
3 continue
inet_gethost = .true.

-17-
4.4.5. Swap bytes in short integer - SWAB.INC

integer*2 function swab(short)

integer*2 short, result
logical*1 bytes(2), temp

equivalence (result, bytes(1))

result = short
temp = bytes(1)
bytes(1) = bytes(2)
bytes(2) = temp
swab = result

return
end
4.4.6. Fetch next word from buffer - GETWORD.INC

```fortran
integer*4 function inet_getword(buf, i, out)

character(*) buf, out
integer*4 i, n, j

n = len(buf)
1 continue
   if (i .gt. n) then
      goto 2
   elseif (buf(i:i) .ne. ' ') then
      goto 2
   else
      i = i + 1
   endif
   goto 1
2 continue
   j = 1
3 continue
   if (i .gt. n) then
      goto 4
   elseif (buf(i:i) .eq. ' ') then
      goto 4
   else
      out(j:j) = buf(i:i)
      j = j + 1
      i = i + 1
   endif
   goto 3
4 continue

inet_getword = j - 1

return
end
```
4.4.7. Translate error into printable string - ERROR.INC

    logical function error(first, second, errbuf, errlen)

    integer*2 first, second
    character(*) errbuf
    integer*4 errlen
    integer*2 err

    errlen = 0
    if (first .and. second) then
      error = .false.
      return
    endif
    if (.not. first) then
      err = first
    else
      err = second
    endif
    if ((err .and. '8000'x) .eq. '8000'x) then
      call eunice_error(err, errbuf, errlen)
    else
      call sys$getmsg(%val(err), %ref(errlen), errbuf, %val(15),)
    endif
    error = .true.
    return

end
subroutine eunice_error(error, errbuf, errlen)

integer*2 error
character*(*) errbuf, temp*100
integer*4 i, errlen

i = error .and. '7fff'x
i = i / 8
if (i .le. 0 .or. i .gt. 65) then
  temp = 'EUNKNOWN, Unknown Eunice error'
else
  goto (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65), i
  temp = 1 'EPERM, Not owner'
goto 100
temp = 1 'ENOENT, No such file or directory'
goto 100
temp = 1 'ESRCH, No such process'
goto 100
temp = 1 'EINTR, Interrupted system call'
goto 100
temp = 1 'EIO, I/O error'
goto 100
temp = 1 'ENXIO, No such device or address'
goto 100
temp = 1 'E2BIG, Arg list too long'
goto 100
temp = 1 'ENXIO, Bad file number'
goto 100
temp = 1 'ECHILD, No children'
goto 100
temp = 1 'EAGAIN, No more processes'
goto 100
temp = 1 'ENOMEM, Not enough core'

goto 100
temp =
1  'EACCES, Permission denied'
goto 100
temp =
1  'EFAULT, Bad address'
goto 100
temp =
1  'ENOTBLK, Block device required'
goto 100
temp =
1  'ENOSPC, No space left on device'
goto 100
temp =
1  'EReSYS, Mount device busy'
goto 100
temp =
1  'ESPIPE, Illegal seek'
goto 100
temp =
1  'ERFS, Read-only file system'
goto 100
temp =
1 'EMFILE, Too many links'
goto 100

32 temp =
1 'EPipe, Broken pipe'
goto 100

33 temp =
1 'EDOM, Argument too large'
goto 100

34 temp =
1 'ERANGE, Result too large'
goto 100

35 temp =
1 'EWOULDBLOCK, Operation would block'
goto 100

36 temp =
1 'EINPROGRESS, Operation now in progress'
goto 100

37 temp =
1 'EALREADY, Operation already in progress'
goto 100

38 temp =
1 'ENDTSOCK, Socket operation on non-socket'
goto 100

39 temp =
1 'EDESTADDRREQ, Destination address required'
goto 100

40 temp =
1 'EMSGSIZE, Message too long'
goto 100

41 temp =
1 'EPROTOTYPE, Protocol wrong type for socket'
goto 100

42 temp =
1 'ENOPROTOOPT, Protocol not available'
goto 100

43 temp =
1 'EPNOTSUPP, Protocol not supported'
goto 100

44 temp =
1 'ESOCKTNOSUPPORT, Socket type not supported'
goto 100

45 temp =
1 'EOPNOTSUPP, Operation not supported on socket'
goto 100

46 temp =
1 'EPFNOSUPPORT, Protocol family not supported'
goto 100

47 temp =
1 'EAFNOSUPPORT, Address family not supported by protocol family'
goto 100

48 temp =
1 'EADDRINUSE, Address already in use'
goto 100

49 temp =
1 'EADDRNOTAVAIL, Cannot assign requested address'
goto 100
   temp =
50
   1 'ENETDOWN, Network is down'
goto 100
   temp =
51
   1 'ENETUNREACH, Network is unreachable'
goto 100
   temp =
52
   1 'ENETRESET, Network dropped connection on reset'
goto 100
   temp =
53
   1 'ECONNABORTED, Software caused connection abort'
goto 100
   temp =
54
   1 'ECONNRESET, Connection reset by peer'
goto 100
   temp =
55
   1 'ENETBUFS, No buffer space available'
goto 100
   temp =
56
   1 'EISCONN, Socket is already connected'
goto 100
   temp =
57
   1 'ENOTCONN, Socket is not connected'
goto 100
   temp =
58
   1 'ESHUTDOWN, Cannot send after socket shutdown'
goto 100
   temp =
59
   1 'ETOOMANYREFS, Too many references: cannot splice'
goto 100
   temp =
60
   1 'ETIMEDOUT, Connection timed out'
goto 100
   temp =
61
   1 'ECONNREFUSED, Connection refused'
goto 100
   temp =
62
   1 'ELOOP, Too many levels of symbolic links'
goto 100
   temp =
63
   1 'ENAMETOOLONG, File name too long'
goto 100
   temp =
64
   1 'EHOSTDOWN, Host is down'
goto 100
   temp =
65
   1 'EHOSTUNREACH, No route to host'
goto 100
endif
100 continue
   errbuf = 'Eunice-E-' // temp
   errlen = len(errbuf)
do while (errlen .gt. 0)
if (errbuf(errlen,errlen) .ne. '') then  
goto 101  
endif  
    errlen = errlen - 1  
enddo  
101 continue  
  return  
end
This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.