Remote Tapes in AIPS

Eric W. Greisen

June 30, 1992

1 Introduction

The management of magnetic tape devices in $\mathcal{AIPS}$ has been complicated by the wide diversity of tape formats and manufacturers and the lack of any widespread standards for returned condition codes and other structures indicative of the tape position and condition. $\mathcal{AIPS}$ needs to have considerable control over tapes. In particular, it must not only read and write, but also do the tape operations of rewind, forward-file, backward-file, and write-end-of-file. The operations of forward-record and backward-record were used previously, but now only `MK3TX` uses forward-record. "Error" conditions including end-of-file, beginning-of-medium, and end-of-medium, as well as non-specific true tape errors, must be detected and reported reliably to the higher-level software.

$\mathcal{AIPS}$ support for a diversity of tape devices has always been suspect. The Z routines to drive tapes must be host operating system dependent, but we have assumed that they have to support only one tape device type, or family of device types with identical data structures and condition codes. In the real world, this is not always the case. The IBM computer called `lemur` in Charlottesville has, for example three tape devices supporting 12-inch 1600/6250-bpi reels, Exabytes. and DATs. Fortunately, all seem to use the same driver, or at least the same return codes. Some of our user sites have not been so fortunate, and have had to develop local versions of the Z routines to support different or diverse tape devices. The obvious solution for this dilemma is to pass character strings to the Z routines describing the tape device types, allowing them to branch as needed to support them all. This solution is in the process of being implemented in the `15OCT92` release of $\mathcal{AIPS}$.

The uncertainty of $\mathcal{AIPS}$ support for multiple tape devices was made even more evident with the advent of computer networking. Tape devices are used too infrequently to justify the expense of putting one on each workstation in the local area network. Thus, users wish to read and write tape devices located on one of more "public" computers from their own computer. The "standard" UNIX solution to the "remote tape" problem was tried in $\mathcal{AIPS}$ for a while using `rmt` and other remote procedures. This solution worked well — between certain cpu's and certain tape devices — but failed in other combinations due to the lack of standard condition codes and tape-position data structures. A more $\mathcal{AIPS}$-like solution for remote tapes is described in this memo and was first installed in $\mathcal{AIPS}$ at the end of 1991 (`15APR92` release). The next two sections of this memo deal with the structure and details of processing tape function calls remotely. Section 4 of this memo describes the underlying Berkeley socket implementation used for the actual communication including some tests of the details which can have a drastic affect on the performance.

2 TPMON

The tape monitor program called `TPMON` is at the heart of the new remote tape system. It operates as a dæmon process on the computer which actually provides the tape drive service. Since every computer supports a disk-file-based virtual tape device, at least one copy of `TPMON` runs on every computer under the name `TPMON1`. Additional copies of `TPMON` run on those computers which have real tape devices under the name(s) `TPMON`n, where $n = 2 \ldots N$ supports $\mathcal{AIPS}$ tape drives $1 \ldots (N-1)$ in the obvious order.

TPMON is fairly simple in structure. When it begins, it determines its name and thereby which tape, or pseudo-tape, device it is to support. Then it uses the Z routine ZVTPRO to open a communications channel to await requests from a client. When one comes along, TPMON moves to the subroutine TPMONI which is virtually an infinite loop. It uses subroutine ZVTPRX to read the transmission from the client. Extracting the subroutine name and the in and out word counts from this transmission, TPMONI calls internal subroutine A*xxxxx* to support a request from *AIPS* Z routine Z*xxxxx*. At present, ZMOUNT, ZTAPE, ZTPCLS, ZTPMIO, and ZTPOPN are the only routines which need to be supported in TPMON. Each of the A*xxxxx* subroutines (1) unpacks the buffer from the client, (2) checks the arguments for inconsistencies, (3) calls the corresponding local (real) Z routine using the arguments provided by the client, and (4) packs any return data into the buffer. TPMONI then calls ZVTPRX to return the error code and any words of returned data. Then the read/process/return cycle is repeated *ad infinitum*.

Actually *ad infinitum* usually ends fairly quickly, in a sense, when the client program terminates its connection. ZVTPRX sees this as an end-of-file condition to which it responds by closing the actual tape device (if it was still open) and then calling routine ZVTPGC to terminate the old connection and wait for a new one.

From the client side, the process is also relatively straightforward. For magnetic tapes, much of the real action is put in the verb MOUNT which has two new adverbs REMHOST, the remote host name, and REMTAPE, the *AIPS* tape drive desired on the remote system. Subroutine ZMOUNT begins the mount by translating the logical TAPE*n*, where *n* is the requested tape drive number. If that translation is the string "REMOTE", then a common variable is set to *hostname*, the value of REMHOST converted to lowercase, and the logical name AMTO*n* is set to AIPSMT*n*:*hostname*. Otherwise, the common variable is set to "LOCAL" and AMTO*n* is set to the translation of TAPE*n*. For remote tapes, subroutine ZMOUNR opens up a communication channel, sends the mount request to the desired TPMON on *hostname*, and closes the channel. For local tapes, the subroutine ZMOUN2 does the actual mount, which can be a no-op, a lock-file operation (*e.g.*, Suns), or a complex manipulation of the tape (*e.g.*, Convex).

All of the client-side tape Z routines, ZTPOPN, ZTPMIO, ZTPWAT, ZTAPE, and ZMOUNT have lower-level associated routines whose names end in R for remote tapes and in 2 for local magnetic tapes. The upper level routines perform generic functions and then call the appropriate lower level routine to do the real work. The ZT*xxx*R routines are the exact reverse of the AT*xxxx* routines inside TPMON. Since all remote tape reading and writing are forced to be synchronous, ZTPWAR only returns data lengths and error codes put in the FTAB by ZTPMIR.

Pseudo-tape disk files are never mounted. Instead, ZTPOPN detects the use of pseudo-tape via the logical unit number and then checks the file name to determine whether to use local or remote techniques. File names beginning with some characters followed by a double colon are taken to be remote with those characters taken as the remote host name. The remaining characters in the file name are passed to the remote TPMON to be used as the file name in the usual way. The only difficulty is that the logical name portion of the file name will be interpreted by the remote TPMON and, hence, will almost certainly have to be some widely known logical symbol such as FITS. If you use, for example, HOME, then you will get the area which was "home" to the process which started the remote TPMON, not necessarily your home area or even *AIPS*'s home area.

3 Communication format

The communication between server (TPMON) and client (some *AIPS* task such as AIPS or PRTTP) requires buffers which consist of 5 words of "header" information followed by *n* words of call sequence data, where *n* varies with the call sequence and with the data which are to be read from or written to the tape. All words in the buffer are in FITS-standard format (ASCII characters, 32-bit big-endian integers, IEEE floating, etc.) except the data directly transmitted to/from the tape. That is treated as a byte stream and not translated in any way. This convention allows computers of dissimilar binary formats to serve each other with remote tapes.

The formats are listed in Table 1.

	Word	Usage
Header	1–2	Subroutine name (6 characters used)
	3	Number data words from client to server
	4	Number data words from server to client
	5	Error code (0 means okay)
ZMOUNT	6	1 for mount, else dismount
	7	Tape drive number
	8	Tape density
	9	Remote user's *POPS* number — ignored
	10	Remote user's *AIPS* number
	for input to `TPMON` and returns	
	6	System error code
	7–26	System error message (80 characters)
ZTPOPN	6	Logical unit number
	7	Tape drive number
	8	Operation code (4 characters)
	9–20	Disk file name (pseudo-tape) (48 characters)
	for input to `TPMON` and returns no data	
ZTAPE	6	Operation code (4 characters)
	7	Logical unit number
	8	Operation count
	for input to `TPMON` and returns no data	
ZTPMIO	6	Operation code (4 characters)
	7	Logical unit number
	8	Number of 8-bit bytes
	9	Buffer number to use
	10	Record number (used in pseudo-tape I/O)
	11–n	Data to be written to tape
	for input to `TPMON` and returns	
	6	System error from `FTAB`
	7	Bytes actually transferred
	8	Record number
	9–n	Data read from tape
	and adjusts the header count of words from `TPMON` to client on reads	
ZTPCLS	6	Logical unit number
	for input to `TPMON` and returns no data	

Table 1 : Communication formats

4 Network Z routines — Discussion

The full text of the $APLBERK network routines for doing the remote tape I/O via Berkeley sockets appears in Section 5. The most unusual aspect of socket I/O is that, when one attempts to read or write n bytes of data, one gets only $m \leq n$ bytes. Therefore, ZVTPX2 and ZVTPX3 are obliged to loop until the desired number of bytes has been achieved. Socket I/O is done from the user's buffers to internal (mostly) hidden buffers after which the data are broken into segments and transmitted. The maximum segment size is determined by the two hosts by agreement and cannot exceed 1460 bytes on Ethernet connections. The default sizes of the hidden send and receive buffers is 4096 bytes.

When the *AIPS* remote tape system was first put into use, I naively had the headers written and read separately from the data, although the two are consecutive in the buffers provided to ZVTPX2 and ZVTPX3. Furthermore, I broke the data transmissions up into 1024-byte blocks. This turned out to be rather inefficient. In one sequence of tests, using PRTTP on a particular tape, I found a real time of 109 seconds with a directly connected tape and 182 seconds with TPMON. Changing the data transmissions either to 512-byte blocks or to the number of bytes actually desired (28800) increased the remote real times to around 255 seconds. With 1024-byte segments, but with the headers and data combined in write operations only, the real time fell to 129 seconds. I don't really understand this since this only changed the number of I/O's in one direction from 30 to 29 while changing the number in the other direction from 2 to 1.

In a later sequence of tests, using PRTTP on a different tape, I found a real time of 152 seconds for directly connected tapes and 272 seconds for tapes connected via TPMON (with header and data written separately in 1024-byte maximum blocks). Changing only TPMON to use 1460-byte records (the agreed maximum segment size) and to concatenate header and data on writes reduced the real time to 192 seconds. Making the same changes to PRTTP brought the time down to 150 seconds, virtually the same as the direct connection. Via PRTTP the cpu time was 20 seconds while the direct connection used only 4. It is possible to raise the size of the "hidden" system buffer, an option used by Unix utilities such as rdump. Doing this allowed me to change the reads and writes from ≤ 1460 bytes to the actual number desired with an improvement in the cpu time to about 15 seconds with no change in the real times.

After testing a variety of tape devices and a variety of cpus, Rick Perley wrote in a mail message of 26 June, 1992 "There is almost no measurable difference between writing on a local tape and writing onto a remote one." He found that IPCs were almost 50% slower than IPXs when writing multiple-file FITS Exabyte tapes and that one CoCOMP 8500 Exabyte on host laguna was peculiarly inefficient. *AIPS* tasks PRTTP and FITTP were used to test both reading and writing. I hope that the routines below can now stand the further tests of time.

5 Network Z routines — Listings

For the client side of the communication, the $APLBERK ZVTPO2.C, ZVTPX2.C, and ZVTPC2.C routines do open, I/O, and close, respectively. They are:

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>
   int Z_maxseg, Z_sndbuf, Z_rcvbuf, Z_sndini, Z_rcvini;

zvtpo2_(fcb, plen, pname, ierr)
/*--------------------------------------------------------------------*/
```

```
/*! open connection in client to server of remote, real tape          */
/*# Tape Z2                                                            */
/*   This software is the subject of a User agreement and is           */
/*   confidential in nature. It shall not be sold or otherwise made    */
/*   available or disclosed to third parties.                          */
/*--------------------------------------------------------------------*/
/*  ZVTPO2 opens the connection in the Virtual tape program to the    */
/*  remote computer which provides the actual tape device.            */
/*  Inputs:                                                            */
/*     FCB     I(*)   File descriptor                                  */
/*     PLEN    I      Length of PNAME                                  */
/*     PNAME   H(*)   Name of remote port as AIPSMTn:machine where n  */
/*                    is a number 1-NTapes and machine is the remote  */
/*                    machine name.  No imbedded blanks               */
/*                    ' ' => real tape                                 */
/*  Output:                                                            */
/*     IERR    I      Error: 0 => okay                                 */
/*  Unix Berkely version - bsd sockets.                                */
/*--------------------------------------------------------------------*/
int *fcb, *plen, *ierr;
char pname[24];
/*--------------------------------------------------------------------*/
{
                                      /* Offsets to entries in the  */
                                      /* file control blocks        */
   extern int Z_fcbfd, Z_fcberr, Z_mfcb;
   extern int errno;
   char device[10], lname[132];
   int i, fd, dupfd, optlen;
   char *machine_in;
   struct sockaddr_in client_in;
   struct servent *sp_in;
   struct hostent *hp_in;
   extern char *getenv();
   extern char *index();
/*--------------------------------------------------------------------*/
   *ierr = 0;
   errno = 0;
                                      /* Extract Aips device name   */
                                      /* from pname                 */
   for (i=0; i < *plen && i < 131 && pname[i] !='\0' && pname[i] !=' ';
      i++)
      lname[i] = pname[i];
   lname[i] = '\0';
   for (i=0; i < 10 && lname[i] != ':' && lname[i] != '\0'; i++)
      device[i] = lname[i];
   device[i] = '\0';
   if (pname[i] == ':')
      machine_in = &lname[i+1];
   else {
      fprintf(stderr, "ZVTPO2: NO REMOTE MACHINE IN %s\n", pname);
      *ierr = 2;
      goto exit;
      }
                                      /* open socket  INET domain   */
   if (strncmp(device,"AIPSMT",6)==0) {
```

```
                                   /* malformed names go here    */
   if (!isalnum(*machine_in)) {
      fprintf(stderr, "ZVTPO2: MALFORMED NAME %s\n", machine_in);
      fprintf(stderr, "ZVTPO2: FROM %s\n", lname);
      *ierr = 2;
      goto exit;
      }
                                   /* translate name             */
   if ((sp_in = getservbyname(device,"tcp")) == NULL) {
      fprintf(stderr,"ZVTPO2: tcp/%s not a service\n",device);
      *ierr = 2;
      goto exit;
      }
   if ((hp_in = gethostbyname(machine_in)) == NULL) {
      fprintf(stderr,"ZVTPO2: %s: unknown host\n",machine_in);
      *ierr = 2;
      goto exit;
      }
   bzero((char *) &client_in, sizeof(client_in));
   bcopy(hp_in->h_addr, (char *) &client_in.sin_addr,
      hp_in->h_length);
   client_in.sin_family = hp_in->h_addrtype;
   client_in.sin_port = sp_in->s_port;
   if ((fd = (int) socket (AF_INET, SOCK_STREAM, 0)) < 0) {
      perror("ZVTPO2 socket (INET)");
      *ierr = 1;
      goto exit;
      }
   *(fcb + Z_fcbfd) = fd;
   if ((dupfd = dup(fd)) == -1) {
                                   /* Store 2nd buffer info in   */
                                   /* 1st buffer FCB for error   */
                                   /* processing (kludge).       */
      *(fcb + Z_mfcb + Z_fcbfd) = dupfd;
      for (i = 0; i < Z_mfcb; i++)
         *(fcb + i) = *(fcb + Z_mfcb + i);
      *(fcb + Z_fcberr) = errno;
      close (fd);
      *ierr = 6;
      goto exit;
      }
   else {
      *(fcb + Z_mfcb + Z_fcbfd) = dupfd;
      }

   if (connect(*fcb, (struct sockaddr *) &client_in,
      sizeof(client_in)) < 0) {
      perror("ZVTPO2 connect (INET)");
      *(fcb + Z_fcberr) = errno;
      close (fd);
      close (dupfd);
      *ierr = 1;
      goto exit;
      }
   optlen = sizeof (Z_sndbuf);
   if (getsockopt (fd, SOL_SOCKET, SO_SNDBUF, (char *) &Z_sndbuf,
```

```
         &optlen) < 0) {
         perror("ZVTPO2 SO_SNDBUF get");
         Z_sndbuf = 4096;
         }
/*    else
         fprintf (stderr,
            "Default SO_SNDBUF is %d bytes\n", Z_sndbuf); */
      optlen = sizeof (Z_rcvbuf);
      if (getsockopt (fd, SOL_SOCKET, SO_RCVBUF, (char *) &Z_rcvbuf,
         &optlen) < 0) {
         perror("ZVTPO2 SO_RCVBUF get");
         Z_rcvbuf = 4096;
         }
/*    else
         fprintf (stderr,
            "Default SO_RCVBUF is %d bytes\n", Z_rcvbuf); */
      optlen = sizeof (Z_maxseg);
      if (getsockopt (fd, IPPROTO_TCP, TCP_MAXSEG, (char *) &Z_maxseg,
         &optlen) < 0) {
         perror("ZVTPO2 TCP_MAXSEG get");
         Z_maxseg = 1024;
         }
/*    else
         fprintf (stderr,
            "Default TCP_MAXSEG is %d bytes\n", Z_maxseg); */
      Z_sndini = Z_sndbuf;
      Z_rcvini = Z_rcvbuf;
      }
                                  /* not AIPSMTn                */
   else {
      fprintf(stderr, "ZVTPO2: NOT TAPE DEVICE = %s\n", device);
      *ierr = 2;
      }
                                  /* Put system error # in FTAB */
 exit:
   if (*ierr != 0) *(fcb + Z_fcberr) = errno;
   return;
}


#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>

zvtpx2_ (fcb, bufsw, bufsr, buffer, ierr)
/*-----------------------------------------------------------------*/
/*! writes/reads to/from server for the client (virtual tape) machine */
/*# Tape Z2                                                        */
/*   This software is the subject of a User agreement and is       */
/*   confidential in nature. It shall not be sold or otherwise made */
/*   available or disclosed to third parties.                      */
/*-----------------------------------------------------------------*/
/*  ZVTPX2 transfers data between the clent (virtual tape) machine  */
/*  and the server (actual tape) computer                          */
/*  Inputs:                                                        */
/*     fcb     I(*)   File descriptor                              */
```

```
/*     bufsw   I      Number of data words (each 32 bits) to send     */
/*                    beyond the header.                              */
/*  In/out:                                                           */
/*     bufsr   I      Number of data words (each 32 bits) to read back*/
/*                    beyond the header: really set by TPMON          */
/*     buffer  I(*)   Data buffer: header + data                      */
/*  Output:                                                           */
/*     ierr    I      Error: 0 => okay                                */
/*  UNIX Berkeley version - read/write from socket.                   */
/*  The header format is, in 32-bit words:                            */
/*      1,2    H(2)   Subroutine name                                 */
/*      3      I      bufsw                                           */
/*      4      I      bufsr                                           */
/*      5      I      ierr                                            */
/*  Data begins in the 6th 32-bit word in any format agreed between   */
/*  TPMON and the relevant subroutine.  E.g.,                         */
/*     ZTAPE:  header   = 'ZTAPE    ', 3, 0, 0                        */
/*             data in  = OPcode, LUN, Count                          */
/*             out      = only header word 5 (IERR) used              */
/*     ZTPMIO: header   = 'ZTPMIO  ',  4, 2+n, 0            (read)  */
/*             data in  = 'READ', LUN, Nbytes, Buf#                   */
/*             data out = SysError, BytesXfer, buffer(N)              */
/*     ZTPMIO: header   = 'ZTPMIO  ',  4+n, 2, 0            (write) */
/*             data in  = 'WRIT', LUN, Nbytes, Buf#, buffer(N)        */
/*             data out = SysError, BytesXfer                         */
/*             where N = # local words to Xfer Nbytes of data         */
/*                   n = # 32-bit words to Xfer Nbytes of data        */
/*             header word 5 (IERR) also used on output.              */
/*--------------------------------------------------------------------*/
int *fcb, *bufsw, *bufsr, *ierr;
char buffer[];
/*--------------------------------------------------------------------*/
{
                                        /* Offsets to entries in the  */
                                        /* file control blocks        */
   extern int Z_fcbfd, Z_fcberr, Z_maxseg, Z_sndbuf, Z_rcvbuf,
      Z_sndini, Z_rcvini;
   extern int errno;
   int nbytes, mbytes, lbytes, sfd, ioff, lwords, iwords, jwords,
      kwords, optlen;
   char *iaddr;
/*--------------------------------------------------------------------*/
   errno = 0;
   sfd = *(fcb + Z_fcbfd) ;
   *ierr = 0;
                                        /* write the header + data    */
   ioff = 0;
                                        /* Loop invariant: elements   */
                                        /* 0 to ioff-1 of buffer have */
                                        /* been written to the socket */
                                        /* Transfer data:             */

   nbytes = 4 * (*bufsw) + 20 ;

   if ((Z_sndbuf == Z_sndini) && (nbytes > Z_sndini)) {
      optlen = sizeof (Z_sndini);
```

```
      Z_sndini = 29000;
      if (setsockopt (sfd, SOL_SOCKET, SO_SNDBUF, (char *) &Z_sndbuf,
         optlen) < 0) {
         perror ("ZVTPX2 increase sndbuf size");
         Z_sndbuf = Z_sndini+1;
         }
      else
         fprintf (stderr, "ZVTPX2 increased send buffer size\n");
      }

/* fprintf (stderr,
      "ZVTPX2 intends %d data bytes to TPMON\n", nbytes); */

   for (nbytes = 4 * (*bufsw) + 20 ; nbytes > 0 ; nbytes -= mbytes) {
      lbytes = (nbytes > Z_maxseg) ? Z_maxseg : nbytes ;
      lbytes = nbytes ;
      iaddr = &(buffer[ioff]) ;
      if ((mbytes = write(sfd, iaddr, lbytes)) == -1) {
         perror("ZVTPX2 WRITE DATA ERROR");
         *ierr = 3;
         goto exit;
         }
      ioff += mbytes;
/*   fprintf (stderr,
        "ZVTPX2 sends %d data bytes to TPMON\n", mbytes); */
      }
                                    /* read the header + data     */
   ioff = 0;
                                    /* Loop invariant: elements   */
                                    /* 0 to ioff-1 have been read */
                                    /* from the socket            */

/* fprintf (stderr, "ZVTPX2 intends to read header from TPMON\n"); */

   for (nbytes = 20 ; nbytes > 0; nbytes -= mbytes) {
      iaddr = &(buffer[ioff]) ;
      if ((mbytes = read(sfd, iaddr, nbytes)) <= 0) {
         if (mbytes == 0)           /* End of file indicator      */
            *ierr = 4;
         else
            *ierr = 3;
         perror("ZVTPX2 READ HEADER ERROR");
         goto exit;
         }
      ioff += mbytes;
/*    fprintf (stderr, "ZVTPX2 reads %d bytes from TPMON\n",mbytes); */
      }
                                   /* number data words           */
                                   /* FITS byte order here        */
   iwords = buffer[13] ;
   jwords = buffer[14] ;
   kwords = buffer[15] ;
   if (iwords < 0) iwords = iwords + 256 ;
   if (jwords < 0) jwords = jwords + 256 ;
   if (kwords < 0) kwords = kwords + 256 ;
   lwords = 256 * (256 * iwords + jwords) + kwords ;
```

```
   *bufsr = lwords;

   nbytes = 4 * (lwords) ;

   if ((Z_rcvbuf == Z_rcvini) && (nbytes > Z_rcvini)) {
      optlen = sizeof (Z_rcvini);
      Z_rcvini = 29000;
      if (setsockopt (sfd, SOL_SOCKET, SO_RCVBUF, (char *) &Z_rcvbuf,
         optlen) < 0) {
         perror ("ZVTPX2 increase rcvbuf size");
         Z_rcvbuf = Z_rcvini+1;
         }
      else
         fprintf (stderr, "ZVTPX2 increased receive buffer size\n");
      }

/* fprintf (stderr,
     "ZVTPX2 intends to read %d data bytes from TPMON\n", nbytes); */

   for (nbytes = 4 * lwords ; nbytes > 0; nbytes -= mbytes) {
      lbytes = (nbytes > Z_maxseg) ? Z_maxseg : nbytes ;
      lbytes = nbytes ;
      iaddr = &(buffer[ioff]) ;
      if ((mbytes = read(sfd, iaddr, lbytes)) <= 0) {
         if (mbytes == 0)              /* End of file indicator      */
            *ierr = 4;
         else
            *ierr = 3;
         perror("ZVTPX2 READ DATA ERROR");
         goto exit;
         }
      ioff += mbytes;
/*    fprintf (stderr, "ZVTPX2 reads %d bytes from TPMON\n",mbytes); */
      }

exit:
   if (*ierr != 0) *(fcb + Z_fcberr) = errno;
   return ;
}


zvtpc2_(fcb, ierr)
/*--------------------------------------------------------------------*/
/*! close virtual tape connection to remote, real-tape computer       */
/*# Tape Z2                                                           */
/*   This software is the subject of a User agreement and is          */
/*   confidential in nature. It shall not be sold or otherwise made   */
/*   available or disclosed to third parties.                         */
/*--------------------------------------------------------------------*/
/*  ZVTPC2 closes the connection in the Virtual tape program to the   */
/*  remote computer which provides the actual tape device.            */
/*  In/out:                                                           */
/*     fcb   I(*)   File descriptor                                   */
/*  Output:                                                           */
/*     ierr  I      Error: 0 => okay                                  */
/*  BSD 4.2 version                                                   */
```

```
/*  deassigns via 'close' certain devices i.e. socket               */
/*-----------------------------------------------------------------*/
int *fcb, *ierr;
/*-----------------------------------------------------------------*/
{
                                        /* Offsets to entries in the  */
                                        /* file control blocks        */
   extern int Z_fcbfd, Z_fcberr, Z_mfcb;
   extern int errno;
   int fd;
/*-----------------------------------------------------------------*/
   *ierr = 0;
                                        /* Close first buffer file    */
                                        /* descriptor.                */
   fd = *(fcb + Z_fcbfd);
   if (close (fd) != 0) {
      *(fcb + Z_fcberr) = errno;
      *ierr = 1;
      }
                                        /* Close second buffer file   */
                                        /* descriptor.                */
   fd = *(fcb + Z_mfcb + Z_fcbfd);
   if (close (fd) != 0 && *ierr == 0) {
      *(fcb + Z_mfcb + Z_fcberr) = errno;
      *ierr = 1;
      }

   return;
}
```

From the server (TPMON) side, the corresponding $APLBERK routines ZVTPO3.C, ZVTPX3.C, and ZVTPC3.C routines do open, I/O, and close, respectively, while ZVTPGC.C closes one connection and waits for the next. They are:

```
#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netdb.h>
   struct sockaddr_un fromb, serverb;
   struct sockaddr_in fromb_in, serverb_in;
   struct servent *spb_in;
   int Z_maxseg, Z_sndbuf, Z_rcvbuf, Z_sndini, Z_rcvini;

zvtpo3_(hlen, hsock, fcb, ierr)
/*-----------------------------------------------------------------*/
/*! open connection in server (real-TP) to client (virtual-TP)      */
/*# Tape Z2                                                         */
/*   This software is the subject of a User agreement and is        */
/*   confidential in nature. It shall not be sold or otherwise made  */
```

```
/*   available or disclosed to third parties.                         */
/*--------------------------------------------------------------------*/
/*  ZVTP03 opens the connection in TPMON to the remote machine which  */
/*  is running the AIPS Virtual tape code.                            */
/*  Outputs:                                                          */
/*     fcb   I(*)   File descriptor                                   */
/*     ierr  I      Error: 0 => okay                                  */
/*                     1 => failure                                   */
/*                     2 => invalid device name                       */
/*   SUN (and other Berkeley UNIX ?) version                          */
/*   Create a socket to receive commands/requests from remote         */
/*   computers for TP service.                                        */
/*--------------------------------------------------------------------*/
int *fcb, *ierr, *hlen;
char hsock[8];
/*--------------------------------------------------------------------*/
{
                                       /* Offsets to entries in the  */
                                       /* file control blocks        */
   extern int Z_fcbfd, Z_fcbreq, Z_fcberr, Z_fcbxfr, Z_fcbsiz,
      Z_fcbsp1, Z_fcbsp2, Z_fcbsp3, Z_fcbsp4;
   extern int errno;
   char device[48];
   int i, len, sfd, cfd, optlen;
/*--------------------------------------------------------------------*/
   *ierr = 0;
   errno = 0;

   for (i=0; i < 48 && hsock[i] != ':' && hsock[i] != '\0'; i++)
      device[i] = hsock[i];
   device[i] = '\0';
                                       /* open socket, INET domain   */
   if (strncmp(device,"AIPSMT",6)==0) {
                                       /* translate name             */
      if ((spb_in = getservbyname (device,"tcp")) == NULL) {
         fprintf(stderr,"ZVTP03: tcp/%s not a service\n", device);
         *ierr = 2;
         goto exit;
         }
      serverb_in.sin_port = spb_in->s_port;
      if ((sfd = (int)socket(AF_INET, SOCK_STREAM, 0)) < 0) {
         perror("ZVTP03 socket (INET)");
         *ierr = 1;
         goto exit;
         }
      *(fcb+Z_fcbfd) = sfd;
      if (bind(sfd, (struct sockaddr *) &serverb_in, sizeof(serverb_in)) < 0) {
         perror("ZVTP03 bind (INET)");
         *ierr = 1;
         *(fcb + Z_fcberr) = errno;
         goto cleanup;
         }
      listen(sfd, 5);
      len = sizeof(fromb_in);
      if ((cfd = accept(sfd, (struct sockaddr *) &fromb_in, &len)) < 0) {
         perror("ZVTP03 accept (INET)");
```

```
         *ierr = 1;
         goto cleanup;
         }
      else {
         *(fcb+Z_fcbsp1) = cfd;
         }
      optlen = sizeof (Z_sndbuf);
      if (getsockopt (cfd, SOL_SOCKET, SO_SNDBUF, (char *) &Z_sndbuf,
         &optlen) < 0) {
         perror("ZVTP03 SO_SNDBUF get");
         Z_sndbuf = 4096;
         }
/*    else
         fprintf (stderr,
            "Default SO_SNDBUF is %d bytes\n", Z_sndbuf); */
      optlen = sizeof (Z_rcvbuf);
      if (getsockopt (cfd, SOL_SOCKET, SO_RCVBUF, (char *) &Z_rcvbuf,
         &optlen) < 0) {
         perror("ZVTP03 SO_RCVBUF get");
         Z_rcvbuf = 4096;
         }
/*    else
         fprintf (stderr,
            "Default SO_RCVBUF is %d bytes\n", Z_rcvbuf); */
      optlen = sizeof (Z_maxseg);
      if (getsockopt (cfd, IPPROTO_TCP, TCP_MAXSEG, (char *) &Z_maxseg,
         &optlen) < 0) {
         perror("ZVTP03 TCP_MAXSEG get");
         Z_maxseg = 1024;
         }
/*    else
         fprintf (stderr,
            "Default TCP_MAXSEG is %d bytes\n", Z_maxseg); */
      Z_sndini = Z_sndbuf;
      Z_rcvini = Z_rcvbuf;
      }
                                        /* UNIX domain not in VTP     */
   else {
      fprintf(stderr,"ZVTP03: UNIX domain not supported\n");
      *ierr = 2;
      }
                                        /* system error # in FTAB     */
 exit:
   if (*ierr != 0) *(fcb + Z_fcberr) = errno;
   goto final;

 cleanup:
   if (shutdown(sfd, 0) != 0) {
      *ierr += 2;
      }
   if (close(sfd) != 0) {
      *ierr += 1;
      }

 final:
   return;
```

```
}


#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>

zvtpx3_ (fcb, bufsw, buffer, ierr)
/*--------------------------------------------------------------------*/
/*! reads/writes from/to client (virtual TP) for the server (real TP) */
/*# TP-IO                                                             */
/*   This software is the subject of a User agreement and is          */
/*   confidential in nature. It shall not be sold or otherwise made   */
/*   available or disclosed to third parties.                         */
/*--------------------------------------------------------------------*/
/*  ZVTPX3 reads into TPMON (on the computer with the real tape) from */
/*  the computer running AIPS virtual tape code and returns answers   */
/*  and error codes.                                                  */
/*  Inputs:                                                           */
/*     fcb      I      File descriptor                                */
/*     bufsw    I      Number words to send (incl header); if = 0,    */
/*                     implies this is a read instead.                */
/*  In/out:                                                           */
/*     buffer   I(*)   5-word header plus data buffer: in             */
/*                     FITS-standard 32-bit integer form!!!!!         */
/*  Output:                                                           */
/*     ierr     I      Error: 0 => okay                               */
/*                        3 => IO error                               */
/*                        4 => End of File                            */
/*  BSD 4.2 version.                                                  */
/*--------------------------------------------------------------------*/
int *fcb, *bufsw, *ierr;
char buffer[];
/*--------------------------------------------------------------------*/
{
                                       /* Offsets to entries in the  */
                                       /* file control blocks        */
   extern int Z_fcbfd, Z_fcbreq, Z_fcberr, Z_fcbxfr, Z_fcbsiz,
      Z_fcbsp1, Z_fcbsp2, Z_fcbsp3, Z_fcbsp4;
   extern int Z_bytflp, Z_maxseg, Z_sndbuf, Z_rcvbuf, Z_sndini,
      Z_rcvini;
   int cfd, nbytes, mbytes, lbytes, ioff, lwords, iwords, jwords,
      kwords, optlen;
   char *iaddr;
   extern int errno;
/*--------------------------------------------------------------------*/
   *ierr = errno = 0;
   cfd = *(fcb + Z_fcbsp1);
                                       /* write the header + data    */
   if (*bufsw > 0) {
      ioff = 0;

      nbytes = 4 * (*bufsw) ;

      if ((Z_sndbuf == Z_sndini) && (nbytes > Z_sndini)) {
         optlen = sizeof (Z_sndini);
```

```
         Z_sndini = 29000;
         if (setsockopt (cfd, SOL_SOCKET, SO_SNDBUF,
            (char *) &Z_sndbuf, optlen) < 0) {
            perror ("ZVTPX3 increase sndbuf size");
            Z_sndbuf = Z_sndini+1;
         }
      else
         fprintf (stderr, "ZVTPX3 increased send buffer size\n");
         }

/*    fprintf (stderr,
         "ZVTPX3 to send back %d data bytes\n", nbytes); */

      for (nbytes = 4 * (*bufsw) ; nbytes > 0; nbytes -= mbytes) {
         lbytes = (nbytes > Z_maxseg) ? Z_maxseg : nbytes ;
         lbytes = nbytes ;
         iaddr = &(buffer[ioff]) ;
         if ((mbytes = write(cfd, iaddr, lbytes)) == 0) {
            perror("ZVTPX3 WRITE DATA ERROR");
            *ierr = 3;
            goto exit;
            }
         ioff += mbytes;
/*      fprintf (stderr,
           "ZVTPX3 sends back %d data bytes\n", mbytes); */
         }
      }
                                       /* read new header            */
   else {
      ioff = 0;
                                       /* Loop invariant: elements   */
                                       /* 0 to ioff-1 of buffer have */
                                       /* been read from the socket  */

/*    fprintf (stderr, "ZVTPX3 to read header\n"); */

      for (nbytes = 20; nbytes > 0; nbytes -= mbytes) {
         iaddr = &(buffer[ioff]);
         if ((mbytes = read(cfd, iaddr, nbytes)) <= 0) {
            if (mbytes == 0)
               *ierr = 4;
            else {
               *ierr = 3;
               perror("ZVTPX3 READ HEADER ERROR");
               }
            goto exit;
            }
         ioff += mbytes;
/*      fprintf (stderr, "ZVTPX3 reads %d header bytes\n",mbytes); */
         }
                                       /* read new data              */
      ioff = 20;
                                       /* FITS byte order here       */
      iwords = buffer[9] ;
      jwords = buffer[10] ;
      kwords = buffer[11] ;
```

```
      if (iwords < 0) iwords = iwords + 256 ;
      if (jwords < 0) jwords = jwords + 256 ;
      if (kwords < 0) kwords = kwords + 256 ;
      lwords = 256 * (256 * iwords + jwords) + kwords ;
                                    /* Loop invariant: elements  */
                                    /* 16 to ioff-1 of buffer    */
                                    /* have been read from the   */
                                    /* socket                    */

      nbytes = 4 * lwords;

   if ((Z_rcvbuf == Z_rcvini) && (nbytes > Z_rcvini)) {
      optlen = sizeof (Z_rcvini);
      Z_rcvini = 29000;
      if (setsockopt (cfd, SOL_SOCKET, SO_RCVBUF, (char *) &Z_rcvbuf,
         optlen) < 0) {
         perror ("ZVTPX2 increase rcvbuf size");
         Z_rcvbuf = Z_rcvini+1;
         }
      else
         fprintf (stderr, "ZVTPX2 increased receive buffer size\n");
      }

/*    fprintf (stderr,
         "ZVTPX3 intends to read %d data bytes\n", nbytes); */

      for (nbytes = 4 * lwords; nbytes > 0; nbytes -= mbytes) {
         lbytes = (nbytes > Z_maxseg) ? Z_maxseg : nbytes ;
         lbytes = nbytes ;
         iaddr = &(buffer[ioff]) ;
         if ((mbytes = read(cfd, iaddr, lbytes)) <= 0) {
            if (mbytes == 0)
               *ierr = 4;
            else
               *ierr = 3;
            perror("ZVTPX3 READ DATA ERROR");
            goto exit;
            }
         ioff += mbytes;
/*       fprintf (stderr, "ZVTPX3 reads %d data bytes\n", mbytes); */
         }
      }


exit:
   return;
}


zvtpc3_(fcb, ierr)
/*--------------------------------------------------------------------*/
/*! close connection in real-TP computer to client, virtual-TP comp.  */
/*# Tape Z2                                                           */
/*   This software is the subject of a User agreement and is          */
/*   confidential in nature. It shall not be sold or otherwise made   */
/*   available or disclosed to third parties.                         */
```

```
/*--------------------------------------------------------------------*/
/*  ZVTPC3 closes the connection in TPMON to the remote machine which */
/*  is running the AIPS Virtual tape code.                            */
/*  Inputs:                                                           */
/*     fcb   I(*)   File descriptor                                   */
/*  Output:                                                           */
/*     ierr  I      Error: 0 => okay                                  */
/* Berkeley BSD 4.2 version                                           */
/* deassigns via 'close' certain devices i.e. tape socket             */
/*--------------------------------------------------------------------*/
int *fcb, *ierr;
/*--------------------------------------------------------------------*/
{
                                        /* Offsets to entries in the  */
                                        /* file control blocks        */
   extern int Z_fcbfd, Z_fcbreq, Z_fcberr, Z_fcbxfr, Z_fcbsiz,
      Z_fcbsp1, Z_fcbsp2, Z_fcbsp3, Z_fcbsp4;
   extern int errno;
   int sfd, cfd;
/*--------------------------------------------------------------------*/
   *ierr = 0;

   sfd = *(fcb + Z_fcbfd) ;
   cfd = *(fcb + Z_fcbsp1) ;

   if (shutdown(cfd, 0) != 0) {
      *(fcb + Z_fcberr) = errno;
      *ierr += 8;
      }
   if (close(cfd) != 0) {
      *(fcb + Z_fcberr) = errno;
      *ierr += 4;
      }

   if (shutdown(sfd, 0) != 0) {
      *(fcb + Z_fcberr) = errno;
      *ierr += 2;
      }
   if (close(sfd) != 0) {
      *(fcb + Z_fcberr) = errno;
      *ierr += 1;
      }

   return;
}


#include <stdio.h>
#include <errno.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <netdb.h>
   struct sockaddr_un fromb, serverb;
```

```
   struct sockaddr_in fromb_in, serverb_in;
   struct servent *spb_in;

zvtpgc_(fcb, ierr)
/*--------------------------------------------------------------------*/
/*! close & reopen connection in server (real-tape) to client (Vtape) */
/*# Tape Z2                                                           */
/*   This software is the subject of a User agreement and is          */
/*   confidential in nature. It shall not be sold or otherwise made   */
/*   available or disclosed to third parties.                         */
/*--------------------------------------------------------------------*/
/*  opens the connection in TPMON to the remote machine which is      */
/*  running the AIPS Virtual tape code.  This differs from ZVTPO3     */
/*  in that the socket is already there and an old connection must be */
/*  closed before a new one can be accepted.  Called by ZVTPRX.       */
/*  In/Outputs:                                                       */
/*     fcb   I(*)   File descriptor                                   */
/*     ierr  I      Error: 0 => okay                                  */
/*   SUN (and other Berkeley UNIX ?) version                          */
/*--------------------------------------------------------------------*/
int *fcb, *ierr;
/*--------------------------------------------------------------------*/
{
                                        /* Offsets to entries in the  */
                                        /* file control blocks        */
   extern int Z_fcbfd, Z_fcberr, Z_fcbsp1;
   extern int errno;
   int len, cfd, sfd;
/*   char *machine_in;
   struct sockaddr_un client;
   struct sockaddr_in client_in;
   struct hostent *hp_in;*/
/*--------------------------------------------------------------------*/
   *ierr = 0;
   errno = 0;
   sfd = *(fcb + Z_fcbfd);
   cfd = *(fcb + Z_fcbsp1);
                                        /* shutdown connection        */
   shutdown(cfd, 0);
   close(cfd);
                                        /* accept new connection      */
   len = sizeof(fromb_in);
   if ((cfd = accept(sfd, &fromb_in, &len)) < 0) {
      perror("ZVTPGC accept (INET) error");
      *ierr = 1;
      *(fcb + Z_fcberr) = errno;
      }
   else {
      *(fcb + Z_fcbsp1) = cfd;
      }

   return;
}
```