APPENDIX C: TRANSPORT INTERFACE EXAMPLES

C

The examples presented throughout this guide are shown in entirety in this appendix.

Connection-Mode Client

The following code represents the connection-mode client program described in Chapter 19. This client establishes a transport connection with a server, and then receives data from the server and writes it to its standard output. The connection is released using the orderly release facility of the Transport Interface. This client will communicate with each of the connection-mode servers presented in the guide.

```
#include <stdio.h>
#include <tiuser.h>
#include <fcntl.h>

#define SRV_ADDR 1  /* server's well known address */

main()
{
   int fd;
   int nbytes;
   int flags = 0;
   char buf[1024];
   struct t_call *sndcall;
   extern int t_errno;

   if ((fd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
      t_error("t_open failed");
      exit(1);
   }

   if (t_bind(fd, NULL, NULL) < 0) {
      t_error("t_bind failed");
      exit(2);
   }

   /*
    * By assuming that the address is an integer value,
    * this program may not run over another protocol.
    */
   if ((sndcall = (struct t_call *)t_alloc(fd, T_CALL, T_ADDR)) == NULL) {
      t_error("t_alloc failed");
      exit(3);
   }
```

C

```
    sndcall->addr.len = sizeof(int);
    *(int *)sndcall->addr.buf = SRV_ADDR;

    if (t_connect(fd, sndcall, NULL) < 0) {
        t_error("t_connect failed for fd");
        exit(4);
    }

    while ((nbytes = t_rcv(fd, buf, 1024, &flags)) != -1) {
        if (fwrite(buf, 1, nbytes, stdout) < 0) {
            fprintf(stderr, "fwrite failed\n");
            exit(5);
        }
    }

    if ((t_errno == TLOOK)  &&  (t_look(fd) == T_ORDREL)) {
        if (t_rcvrel(fd) < 0) {
            t_error("t_rcvrel failed");
            exit(6);
        }
        if (t_sndrel(fd) < 0) {
            t_error("t_sndrel failed");
            exit(7);
        }
        exit(0);
    }
    t_error("t_rcv failed");
    exit(8);
}
```

Connection-Mode Server

The following code represents the connection-mode server program described in Chapter 19. This server establishes a transport connection with a client, and then transfers a log file to the client on the other side of the connection. The connection is released using the orderly release facility of the Transport Interface. The connection-mode client presented earlier will communicate with this server.


```
#include <tiuser.h>
#include <stropts.h>
#include <fcntl.h>
#include <stdio.h>
#include <signal.h>

#define DISCONNECT  -1
#define SRV_ADDR  1 /* server's well known address */

int conn_fd;        /* connection established here */
extern int t_errno;

main()
{
  int listen_fd;    /* listening transport endpoint */
  struct t_bind *bind;
  struct t_call *call;

  if ((listen_fd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
     t_error("t_open failed for listen_fd");
     exit(1);
  }

  /*
   * By assuming that the address is an integer value,
   * this program may not run over another protocol.
   */
  if ((bind = (struct t_bind *)t_alloc(listen_fd, T_BIND, T_ALL)) == NULL) {
     t_error("t_alloc of t_bind structure failed");
     exit(2);
  }
  bind->qlen = 1;
  bind->addr.len = sizeof(int);
  *(int *)bind->addr.buf = SRV_ADDR;

  if (t_bind(listen_fd, bind, bind) < 0) {
     t_error("t_bind failed for listen_fd");
     exit(3);
  }
```

C

```
    /*
     * Was the correct address bound?
     */
    if (*(int *)bind->addr.buf != SRV_ADDR) {
        fprintf(stderr, "t_bind bound wrong address\n");
        exit(4);
    }

    if ((call = (struct t_call *)t_alloc(listen_fd, T_CALL, T_ALL)) == NULL) {
        t_error("t_alloc of t_call structure failed");
        exit(5);
    }

    while (1) {
        if (t_listen(listen_fd, call) < 0) {
            t_error("t_listen failed for listen_fd");
            exit(6);
        }

        if ((conn_fd = accept_call(listen_fd, call)) != DISCONNECT)
            run_server(listen_fd);
    }
}


accept_call(listen_fd, call)
int listen_fd;
struct t_call *call;
{
    int resfd;

    if ((resfd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
        t_error("t_open for responding fd failed");
        exit(7);
    }

    if (t_bind(resfd, NULL, NULL) < 0) {
        t_error("t_bind for responding fd failed");
        exit(8);
    }
```

```
   if (t_accept(listen_fd, resfd, call) < 0) {
      if (t_errno == TLOOK) {  /* must be a disconnect */
         if (t_rcvdis(listen_fd, NULL) < 0) {
            t_error("t_rcvdis failed for listen_fd");
            exit(9);
         }
         if (t_close(resfd) < 0) {
            t_error("t_close failed for responding fd");
            exit(10);
         }
         /* go back up and listen for other calls */
         return(DISCONNECT);
      }
      t_error("t_accept failed");
      exit(11);
   }
   return(resfd);
}


connrelease()
{
   /* conn_fd is global because needed here */
   if (t_look(conn_fd) == T_DISCONNECT) {
      fprintf(stderr, "connection aborted\n");
      exit(12);
   }

   /* else orderly release indication - normal exit */
   exit(0);
}


run_server(listen_fd)
int listen_fd;
{
   int nbytes;
   FILE *logfp;       /* file pointer to log file */
   char buf[1024];

   switch (fork()) {

   case -1:
      perror("fork failed");
      exit(20);
```

C

```
  default: /* parent */

     /* close conn_fd and then go up and listen again */
     if (t_close(conn_fd) < 0) {
        t_error("t_close failed for conn_fd");
        exit(21);
     }
     return;

  case 0:      /* child */

     /* close listen_fd and do service */
     if (t_close(listen_fd) < 0) {
        t_error("t_close failed for listen_fd");
        exit(22);
     }
     if ((logfp = fopen("logfile", "r")) == NULL) {
        perror("cannot open logfile");
        exit(23);
     }

     signal(SIGPOLL, connrelease);
     if (ioctl(conn_fd, I_SETSIG, S_INPUT) < 0) {
        perror("ioctl I_SETSIG failed");
        exit(24);
     }
     if (t_look(conn_fd) != 0) {/* was disconnect already there? */
        fprintf(stderr, "t_look returned unexpected event\n");
        exit(25);
     }

     while ((nbytes = fread(buf, 1, 1024, logfp)) > 0)
        if (t_snd(conn_fd, buf, nbytes, 0) < 0) {
           t_error("t_snd failed");
           exit(26);
        }

     if (t_sndrel(conn_fd) < 0) {
        t_error("t_sndrel failed");
        exit(27);
     }
     pause(); /* until orderly release indication arrives */
  }
}
```

Connectionless-Mode Transaction Server

The following code represents the connectionless-mode transaction server program described in Chapter 19. This server waits for incoming datagram queries, and then processes each query and sends a response.

```
#include <stdio.h>
#include <fcntl.h>
#include <tiuser.h>

#define SRV_ADDR 2      /* server's well known address */

main()
{
   int fd;
   int flags;
   struct t_bind *bind;
   struct t_unitdata *ud;
   struct t_uderr *uderr;
   extern int t_errno;


   if ((fd = t_open("/dev/tidg", O_RDWR, NULL)) < 0) {
      t_error("unable to open /dev/provider");
      exit(1);
   }

   if ((bind = (struct t_bind *)t_alloc(fd, T_BIND, T_ADDR)) == NULL) {
      t_error("t_alloc of t_bind structure failed");
      exit(2);
   }
   bind->addr.len = sizeof(int);
   *(int *)bind->addr.buf = SRV_ADDR;
   bind->qlen = 0;

   if (t_bind(fd, bind, bind) < 0) {
      t_error("t_bind failed");
      exit(3);
   }

   /*
    * is the bound address correct?
    */
   if (*(int *)bind->addr.buf != SRV_ADDR) {
      fprintf(stderr, "t_bind bound wrong address\n");
      exit(4);
   }
```

C

```
    if ((ud = (struct t_unitdata *)t_alloc(fd, T_UNITDATA, T_ALL)) == NULL) {
        t_error("t_alloc of t_unitdata structure failed");
        exit(5);
    }
    if ((uderr = (struct t_uderr *)t_alloc(fd, T_UDERROR, T_ALL)) == NULL) {
        t_error("t_alloc of t_uderr structure failed");
        exit(6);
    }

    while (1) {
        if (t_rcvudata(fd, ud, &flags) < 0) {
            if (t_errno == TLOOK) {
                /*
                 * Error on previously sent datagram
                 */
                if (t_rcvuderr(fd, uderr) < 0) {
                    t_error("t_rcvuderr failed");
                    exit(7);
                }
                fprintf(stderr, "bad datagram, error = %d\n",
                    uderr->error);
                continue;
            }
            t_error("t_rcvudata failed");
            exit(8);
        }

        /*
         * Query() processes the request and places the
         * response in ud->udata.buf, setting ud->udata.len
         */
        query(ud);

        if (t_sndudata(fd, ud, 0) < 0) {
            t_error("t_sndudata failed");
            exit(9);
        }
    }
}

query()
{
    /* Merely a stub for simplicity */
}
```

Read/Write Client

The following code represents the connection-mode **read/write** client program described in Chapter 19. This client establishes a transport connection with a server, and then uses **cat**(1) to retrieve the data sent by the server and write it to its standard output. This client will communicate with each of the connection-mode servers presented in the guide.

C

```
#include <stdio.h>
#include <tiuser.h>
#include <fcntl.h>
#include <stropts.h>

#define SRV_ADDR 1  /* server's well known address */

main()
{
    int fd;
    int nbytes;
    int flags = 0;
    char buf[1024];
    struct t_call *sndcall;
    extern int t_errno;

    if ((fd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
        t_error("t_open failed");
        exit(1);
    }

    if (t_bind(fd, NULL, NULL) < 0) {
        t_error("t_bind failed");
        exit(2);
    }

    /*
     * By assuming that the address is an integer value,
     * this program may not run over another protocol.
     */

    if ((sndcall = (struct t_call *)t_alloc(fd, T_CALL, T_ADDR)) == NULL) {
        t_error("t_alloc failed");
        exit(3);
    }

    sndcall->addr.len = sizeof(int);
    *(int *)sndcall->addr.buf = SRV_ADDR;
```

C

```
    if (t_connect(fd, sndcall, NULL) < 0) {
        t_error("t_connect failed for fd");
        exit(4);
    }

    if (ioctl(fd, I_PUSH, "tirdwr") < 0) {
        perror("I_PUSH of tirdwr failed");
        exit(5);
    }

    close(0);
    dup(fd);

    execl("/bin/cat", "/bin/cat", 0);

    perror("execl of /bin/cat failed");
    exit(6);
}
```

Event-Driven Server

The following code represents the connection-mode server program described in Chapter 19. This server manages multiple connect indications in an event-driven manner. Either connection-mode client presented earlier will communicate with this server.

C

```
#include <tiuser.h>
#include <fcntl.h>
#include <stdio.h>
#include <poll.h>
#include <stropts.h>
#include <signal.h>

#define NUM_FDS         1
#define MAX_CONN_IND4
#define SRV_ADDR        1   /* server's well known address */

int conn_fd;                /* server connection here */
struct t_call *calls[NUM_FDS][MAX_CONN_IND];/* holds connect indications */
extern int t_errno;

main()
{
  struct pollfd pollfds[NUM_FDS];
  struct t_bind *bind;
  int i;

  /*
   * Only opening and binding one transport endpoint,
   * but more could be supported
   */
  if ((pollfds[0].fd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
     t_error("t_open failed");
     exit(1);
  }

  if ((bind = (struct t_bind *)t_alloc(pollfds[0].fd, T_BIND, T_ALL)) == NULL) {
     t_error("t_alloc of t_bind structure failed");
     exit(2);
  }
  bind->qlen = MAX_CONN_IND;
  bind->addr.len = sizeof(int);
  *(int *)bind->addr.buf = SRV_ADDR;

  if (t_bind(pollfds[0].fd, bind, bind) < 0) {
     t_error("t_bind failed");
     exit(3);
  }
```

C

```
  /*
   * Was the correct address bound?
   */
  if (*(int *)bind->addr.buf != SRV_ADDR) {
    fprintf(stderr, "t_bind bound wrong address\n");
    exit(4);
  }

  pollfds[0].events = POLLIN;

  while (1) {
    if (poll(pollfds, NUM_FDS, -1) < 0) {
      perror("poll failed");
      exit(5);
    }

    for (i = 0; i < NUM_FDS; i++) {

      switch (pollfds[i].revents) {

      default:
        perror("poll returned error event");
        exit(6);

      case 0:
        continue;

      case POLLIN:
        do_event(i, pollfds[i].fd);
        service_conn_ind(i, pollfds[i].fd);
      }
    }
  }
}


do_event(slot, fd)
{
  struct t_discon *discon;
  int i;

  switch (t_look(fd)) {

  default:
    fprintf(stderr,"t_look returned an unexpected event\n");
    exit(7);

  case T_ERROR:
    fprintf(stderr,"t_look returned T_ERROR event\n");
    exit(8);
```

```
    case -1:
      t_error("t_look failed");
      exit(9);

    case 0:
      /* since POLLIN returned, this should not happen */
      fprintf(stderr,"t_look returned no event\n");
      exit(10);

    case T_LISTEN:
      /*
       * find free element in calls array
       */
      for (i = 0; i < MAX_CONN_IND; i++) {
        if (calls[slot][i] == NULL)
          break;
      }

      if ((calls[slot][i] = (struct t_call *)t_alloc(fd, T_CALL, T_ALL)) == NULL)
      {
        t_error("t_alloc of t_call structure failed");
        exit(11);
      }

      if (t_listen(fd, calls[slot][i]) < 0) {
        t_error("t_listen failed");
        exit(12);
      }
      break;

    case T_DISCONNECT:
      discon = (struct t_discon *)t_alloc(fd, T_DIS, T_ALL);

      if (t_rcvdis(fd, discon) < 0) {
        t_error("t_rcvdis failed");
        exit(13);
      }
      /*
       * find call ind in array and delete it
       */
      for (i = 0; i < MAX_CONN_IND; i++) {
        if (discon->sequence == calls[slot][i]->sequence) {
          t_free(calls[slot][i], T_CALL);
          calls[slot][i] = NULL;
        }
      }
      t_free(discon, T_DIS);
      break;
    }
}
```

C

```
service_conn_ind(slot, fd)
{
  int i;

  for (i = 0; i < MAX_CONN_IND; i++) {
    if (calls[slot][i] == NULL)
      continue;

    if ((conn_fd = t_open("/dev/tivc", O_RDWR, NULL)) < 0) {
      t_error("open failed");
      exit(14);
    }
    if (t_bind(conn_fd, NULL, NULL) < 0) {
      t_error("t_bind failed");
      exit(15);
    }

    if (t_accept(fd, conn_fd, calls[slot][i]) < 0) {
      if (t_errno == TLOOK) {
        t_close(conn_fd);
        return;
      }
      t_error("t_accept failed");
      exit(16);
    }
    t_free(calls[slot][i], T_CALL);
    calls[slot][i] = NULL;

    run_server(fd);
  }
}


connrelease()
{
  /* conn_fd is global because needed here */
  if (t_look(conn_fd) == T_DISCONNECT) {
    fprintf(stderr, "connection aborted\n");
    exit(12);
  }

  /* else orderly release indication - normal exit */
  exit(0);
}
```

```
run_server(listen_fd)
int listen_fd;
{
  int nbytes;
  FILE *logfp;    /* file pointer to log file */
  char buf[1024];

  switch (fork()) {

  case -1:
    perror("fork failed");
    exit(20);

  default:  /* parent */

    /* close conn_fd and then go up and listen again */
    if (t_close(conn_fd) < 0) {
      t_error("t_close failed for conn_fd");
      exit(21);
    }
    return;

  case 0:      /* child */

    /* close listen_fd and do service */
    if (t_close(listen_fd) < 0) {
      t_error("t_close failed for listen_fd");
      exit(22);
    }
    if ((logfp = fopen("logfile", "r")) == NULL) {
      perror("cannot open logfile");
      exit(23);
    }

    signal(SIGPOLL, connrelease);
    if (ioctl(conn_fd, I_SETSIG, S_INPUT) < 0) {
      perror("ioctl I_SETSIG failed");
      exit(24);
    }
    if (t_look(conn_fd) != 0) {/* was disconnect already there? */
      fprintf(stderr, "t_look returned unexpected event\n");
      exit(25);
    }

    while ((nbytes = fread(buf, 1, 1024, logfp)) > 0)
      if (t_snd(conn_fd, buf, nbytes, 0) < 0) {
        t_error("t_snd failed");
        exit(26);
      }
```

```
    if (t_sndrel(conn_fd) < 0) {
      t_error("t_sndrel failed");
      exit(27);
    }
    pause();  /* until orderly release indication arrives */
  }
}
```