4.7 Listen Facility

The *libtsap*(3n) library, supports a facility which permits a process to *listen* for certain connections. This can be useful for implementing an application which requires that a single server process handle multiple clients, or for connection recovery. These routines return the manifest constant `NOTOK` on error, and `OK` on success, they also update the `td` parameter given to the routine. This parameter is a pointer to a `TSAPdisconnect` structure.

Although this facility is described in terms of the *libtsap*(3n) library, it will function at any other higher-layer in the system (e.g., the listen facility can be used for session or application-entities).

A program starts listening for an particular connection by calling the routine `TNetListen`.

```
int     TNetListen (ta, td)
struct TSAPaddr *ta;
struct TSAPdisconnect *td;
```

The parameters to this procedure are:

`ta:` the transport address (0 or more network addresses) to listen on; and,

`td:` a pointer to a `TSAPdisconnect` structure, which is updated only if the call fails.

If the call is successful, then the program is now listening for incoming connections on that network address. Otherwise, the `td` parameter indicates the reason for failure.

A variant of `TNetListen` is the `TNetUnique` routine, which starts the process listening on a set of unique network (sub)addresses.

```
int     TNetUnique (ta, td)
struct TSAPaddr *ta;
struct TSAPdisconnect *td;
```

The parameters to this procedure are:

`ta:` a transport address containing one or more partially filled-in network addresses; and,

`td`: a pointer to a `TSAPdisconnect` structure, which is updated only if the call fails.

If the call is successful, each network address in the `ta` parameter is fully filled-in, the program is now listening for incoming connections on the resulting transport. Otherwise, the `td` parameter indicates the reason for failure.

To check when a new connection is waiting, or when existing connections have activity on them, the routine `TNetAccept` is used.

```
int     TNetAccept (vecp, vec, nfds, rfds, wfds, efds, secs,
                    td)
int    *vecp,
char  **vec;
int     nfds;
fd_set *rfds,
       *wfds,
       *efds;
int     secs;
struct TSAPdisconnect *td;
```

The parameters to this procedure are:

`vecp/vec`: the initialization vector for the new connection.

`nfds/rfds/wfds/efds`: connection-descriptors for use with `xselect`;

`secs`: the maximum number of seconds to wait for activity (a value of `NOTOK` indicates that the call should block indefinitely, whereas a value of `OK` indicates that the call should not block at all, e.g., a polling action); and,

`td`: a pointer to a `TSAPdisconnect` structure, which is updated only if the call fails.

If the call to `TNetAccept` succeeds then the value of `vecp` should be checked. If `vecp` is greater than zero, a new connection has been accepted, and `TInit` should be called, presumably followed by `TConnResponse`.[1] Regardless of the value of `vecp`, the value of `rfds` and `wfds` should be checked to see which

[1] Actually, any service addressable via a transport selector can use this service, e.g., if appropriate, a call to `AcInit`, followed by a call to `AcAssocResponse`, can be made.

connections have activity pending. For these connections, any reads should probably be done with a `secs` argument indicating a polling operation (i.e., a value of `OK`). Otherwise, if the call to `TNetAccept` fails, then the `td` parameter indicates the reason for failure.

> **NOTE:** The `TNetAccept` procedure when first called arranges to clean up dead child processes. If the program will run any subprocesses and check their exit status, the automatic collection of zombie process should be disabled, by first calling `TNetAccept` with a timeout of `OK` and then setting the child signal handler to it's default state.

The routine `TNetAccept` is actually a macro which invokes a routine called `TNetAcceptAux`:

```
int     TNetAcceptAux (vecp, vec, newfd, ta, nfds, rfds, wfds,
                efds, secs, td)
int    *vecp,
char  **vec;
int     newfd;
struct TSAPaddr *ta;
int     nfds;
fd_set *rfds,
       *wfds,
       *efds;
int     secs;
struct TSAPdisconnect *td;
```

The additional parameters to this procedure are:

`newfd:` a pointer to an integer which will be given the value of the connection-descriptor associated with the new connection; and,

`ta:` a pointer to a `TSAPaddr` structure which will be given the value of the transport address receiving the new connection (the called or listening address).

Prior to exiting, the user should call `TNetClose` to stop listening for connections.

```
int     TNetClose (ta, td)
struct TSAPaddr *ta;
struct TSAPdisconnect *td;
```

The parameters to this procedure are:

`ta:` the transport address to stop listening on (use the manifest constant `NULLTA` to stop listening on all addresses); and,

`td:` a pointer to a `TSAPdisconnect` structure, which is updated only if the call fails.

If the call is successful, then the program has stopped listening for incoming connections on that network address. Otherwise, the `td` parameter indicates the reason for failure.

4.8 Queued (non-blocking) Writes Facility

All "read" operations in the ISODE are inherently non-blocking. Historically, "write" operations have not required this capability. However, some applications (e.g., the QUIPU DSA) require non-blocking writes. The routine `TSetQueuesOK` is used to enable or disable this facility:

```
int     TSetQueuesOK (sd, onoff, td)
int     sd;
int     onoff;
struct TSAPdisconnect *td;
```

The parameters to this procedure are:

`sd:` the transport-descriptor;

`onoff:` a flag indicating whether the facility is to be enabled (non-zero value) or disabled (zero value) for the transport-descriptor; and,

`td:` a pointer to a `TSAPdisconnect` structure, which is updated only if the call fails.

If the call is not successful, the `td` parameter indicates the reason for failure. Otherwise, if the `onoff` parameter has a non-zero value, then any "write" operations which ultimately map to this transport-descriptor will not block the process. This is done by maintaining a queue of write operations and periodically retrying them. In order to schedule retries, the routine `TNetAccept` described earlier should be called frequently. In addition, if some failure occurs during the retry (e.g., the transport connection is disconnected), then `TNetAccept` will mark that descriptor as being ready for reading. When the program interrogates the descriptor, the appropriate error code will be returned. Note that in order for this action to occur, any descriptor which has queued writes enabled must be included in the `rfs` parameter supplied when the `TNetAccept` routine is called.

4.9 Error Conventions

All of the routines in this library return the manifest constant `NOTOK` on error, and also update the `td` parameter given to the routine. The `td_reason` element of the `TSAPdisconnect` structure can be given as an parameter to the routine `TErrString` which returns a null-terminated diagnostic string.

```
char   *TErrString (c)
int     c;
```

4.10 Compiling and Loading

Programs using the *libtsap*(3n) library should include `<isode/tsap.h>`. These programs should also be loaded with `-ltsap` and, for reasons explained momentarily, `-licompat`.

4.11 An Example

Let's consider how one might construct a loopback entity that resides on the TSAP. This entity will use a synchronous interface.

First, we must decide at what address the entity will reside. For simplicity's sake, we'll say that the location is `tsap/echo`, as defined in the *isoservices*(5) database.

Next, we actually code the loopback entity. There are two parts to the program: initialization and data transfer; release will occur whenever data transfer fails. In our example, we assume that the routine `error` results in the process being terminated after printing a diagnostic.

In Figure 4.2, the initialization steps for the loopback entity, including the outer *C* wrapper, is shown. The entity does not examine any of its arguments, but could do so after the call to `TInit`. After examining the arguments, it could decide to reject the connection attempt, by using `TDiscRequest`. Instead, it uses `TConnResponse` with the exact negotiated parameters with which it was supplied. Hence, if the initiator wanted to use expedited data transfer, the loopback entity responding to the connection would honor that.

In Figure 4.3 on page 136, the data transfer loop is realized. The loopback entity awaits an event from the service provider, which either indicates that data has arrived or that the connection has been closed. If a disconnection occurred (a T-DISCONNECT.INDICATION event is reported), then the reason is checked. If the event did not occur because the initiator performed a T-DISCONNECT.REQUEST indication, then an error is signaled. Otherwise, the inner-loop is terminated and the process will gracefully terminate. If instead data arrived, it is echoed back to the initiator.

```
#include <stdio.h>
#include <isode/tsap.h>
#include <isode/isoservent.h>


/* ARGSUSED */

main (argc, argv, envp)
int       argc;
char  **argv,                                                    10
     **envp;
{
   int     result,
             sd;
   struct TSAPstart    tss;
   register struct TSAPstart  *ts = &tss;
   struct TSAPdata txs;
   register struct TSAPdata  *tx = &txs;
   struct TSAPdisconnect   tds;
   register struct TSAPdisconnect *td = &tds;                    20

   if (TInit (argc, argv, ts, td) == NOTOK)
         error ("T-CONNECT.INDICATION: %s", TErrString (td -> td_reason));
   sd = ts -> ts_sd;

/* examine argv here, if need be */

   if (TConnResponse (sd, &ts -> ts_called, ts -> ts_expedited,
                     ts -> ts_data, ts -> ts_len, NULLQOS, td) == NOTOK)
         error ("T-CONNECT.RESPONSE: %s", TErrString (td -> td_reason));  30

...
```

Figure 4.2: Initializing the loopback entity

```
...

    for (;;) {
        if (TReadRequest (sd, tx, NOTOK, td) == NOTOK) {
            if (td -> td_reason != DR_NORMAL)
                error ("T-READ.REQUEST: %s", TErrString (td -> td_reason));
            break;
        }

        if (tx -> tx_expedited)                                          10
            result = TExpdRequest (sd, tx -> tx_base, tx -> tx_cc, td);
        else
            result = TDataRequest (sd, tx -> tx_base, tx -> tx_cc, td);
        if (tx -> tx_base)
            free (tx -> tx_base);

        if (result == NOTOK)
            error ("%s:  %s", tx -> tx_expedited ? "T-EXPEDITED-DATA.REQUEST"
                    : "T-DATA.REQUEST", TErrString (td -> td_reason));
                                                                         20
    }

    exit (0);
}
```

Figure 4.3: Data Transfer for the loopback entity

4.12 Compatibility Issues

The *libicompat*(3) library is used as an aid for porting the software from one system to another. This library contains generic service routines, which are in turn composed of the native facilities available on the target host. All of the higher layer `#include` files automatically reference parts of this library as appropriate. Hence, when loading the the portions of the software independently, the loader must be given the `-licompat` flag.

The problem of this approach, of course, is that not all facilities can be precisely emulated. To misquote M.A. Padlipsky[Pad85]:

Sometimes when you try to make an apple look like an orange you get back something that smells like a lemon.

4.13 For Further Reading

The ISO specification for transport services is defined in [ISO86]. The corresponding CCITT recommendation is defined in [CCI84c]. The document describing how these services can be implemented on top of the TCP[Pos81] is [RC87].

4.14 Changes Since the Last Release

A brief summary of the major changes between v 6.0 and v 7.0 are now presented. These are the user-visible changes only; changes of a strictly internal nature are not discussed.

The `na_type` and `na_subnet` fields of the `NSAPaddr` structure are now called `na_stack` and `na_community` respectively. For compatibility, macros are provided. These macros will be removed after this release.