UNITITLED

Writing client and server network applications is challenging. There are many aspects to it. For applications that are clients of services available on a network, the procedures involved in obtaining a service include: determining the name, address and network of the remote system that provides the desired service; establishing a connection to the remote system; requesting a service (e.g. remote command execution); passing authentication (e.g. id and password); data communication; and processing information. For applications that provide a service that is available on a network, the procedures include: registering as a service; receiving service requests; data communications; and processing data. As we can see there is a little overlap in the procedures involved for both the client and server applications — data communications and processing data. These are transport layer and application specific issues respectively. They will not be dealt with here. What we will be dealing with here is how clients get to servers and services, and how servers make themselves available to clients to service their requests.

The basis for this discussion will be the Networking Support and Service Access facilities within UNIX System V Release 4.0. (I can hear the Berkeley wonks groaning already. Gearing-up for the grand debate of which flavor is the ‘‘real’’ UNIX system. Without intending to invite battle I only offer that the core technology in Solaris 2.0 is SVR4. And, as we all know, Solaris is available now!) The emphesis is on integrating our applications with existing system facilities and to avoid re-inventing the wheel or establishing non-standard practices. These system facilities include:

- Network Selection
- Name-to-Address Mapping
- Port Monitors
- NLPS Server
- Loopback Transports

The first two facilities are for client use. They provide a mechanism for finding hosts and services on a network or networks. The third is a server side facility for detecting client connections and routing clients to servers. The fourth is a facility for routing client requests based on a symbolic name (versus a port id or number) of the service. The fifth is a facility for testing our client and server applications on the local system. These are all provided with the Networking Support Utilities (NSU) package. The last facility we will discuss is the Service

Access Facility (SAF). Its purpose is to manage all the port monitors on the system.

Network Selection and Name-to-Address Mapping

Network Selection and Name-to-Address Mapping are facilities that together provide a mechanism for determining the name and address of a particular host and service, on a particular network. Technically these are two separate mechanisms but they are rarely ever used separately. Moreover, for our purposes we will be using them together.

SVR4 supports numerous network providers that coexist peacefully on the system. Network Selection is a mechanism for finding a network provider that provides the correct semantics for a given application. Furthermore, different network providers use different addressing formats as well as certain conventions. For example, Internet addresses are of the (familiar) byte tuple form of A.B.C.D. Whereas other providers may use names, pathnames or even a different byte tuple format, of say B.C.A.D. However, this kind of representation is simply the human readable form of the address. It says nothing about how the machine or ‘‘network readable’’ form of the address is represented. Given these realities, an obvious problem presents itself: How do the human readable addresses of a specific network provider get translated into the machine readable form necessary for communicating on the network? It should also be obvious that it is unreasonable to embed network specific translation information in the application. This is the problem that Name-to-Address mapping solves.

To support multiple network providers, a new convention has been established with respect to hostname/address pairs and service id/port number pairs. Hostname/address pairs are stored on the system in /etc/net/*provider-name*/`hosts`. Service-id/port-id pairs are stored in /etc/net/*provider-name*/`services`. (In some non-converted TCP/IP network packages, `/etc/hosts` and `/etc/services` are still there versus being a symbolic link, for backward compatibility, to the real hosts and services file in `/etc/net/inet`.) The format of these files is specific to the particular provider they support but in most cases it is similar to the TCP/IP format of

address *hostname*

for `/etc/hosts` and

service-id *port-id*

for `/etc/services`.

Also necessary to support multiple providers is a database of provider capabilities and semantics. This information can be found in `/etc/netconfig`. The fields of this file are:

- The name of the transport (e.g., `tcp`, `starlan`).
- The semantics, like connection-oriented or connection-less, of the transport (e.g., `tpi_cots`, `tpi_clts`).
- Flags. The only flag currently defined is `v` for whether the transport is visible for use or not.
- The protocol family of the transport (e.g., `inet`, `osi`, `loopback`).
- The protocol name of the transport (e.g., `tcp`, `udp`).
- The device to use for access to the transport (e.g., `/dev/tcp`, `/dev/starlan`).
- The path of the dynamic shared library to use to do the provider dependent name-to-address mapping (e.g., `/usr/lib/tcpip.so`, `/usr/lib/starlan.so`).

Here is an example of the `netconfig` file that has the basic system loopback providers (more on these later) and the Internet suite of providers installed.

```
#
# The Network Configuration File.
#
# Each entry is of the form:
#
# network_id semantics flags protofamily protoname device nametoaddr_libs
#
ticlts      tpi_clts      v   loopback   -      /dev/ticlts     /usr/lib/straddr.so
ticots      tpi_cots      v   loopback   -      /dev/ticots     /usr/lib/straddr.so
ticotsord   tpi_cots_ord  v   loopback   -      /dev/ticotsord  /usr/lib/straddr.so
tcp         tpi_cots_ord  v   inet       tcp    /dev/tcp        /usr/lib/tcpip.so
udp         tpi_clts      v   inet       udp    /dev/udp        /usr/lib/tcpip.so
rawip       tpi_raw       -   inet       -      /dev/rawip      /usr/lib/tcpip.so
icmp        tpi_raw       -   inet       icmp   /dev/icmp       /usr/lib/tcpip.so
```

Now consider client application that would like to connect to service `fubar` on the system `snafu`. This application would start by using a set of functions called `netconfig` functions, to find a network provider on the local system that is suitable for use. `netconfig` functions parse the system file `/etc/netconfig` similar to the way the `getpwent()` functions parse `/etc/passwd`. Once a suitable network

provider is found, the `netdir` functions would be used to find the address of service `fubar` on the system `snafu`. `netdir` functions do the name-to-address mapping. Given a network provider, a host name and a service name `netdir_getbyname()` will search the appropriate system files and return the transport specific address of the requested service on the named host.

The following code fragment should clarify how to use the Network Selection and Name-to-Address mapping mechanisms.

```
#include <netconfig.h>
#include <netdir.h>

struct nd_addrlist *
FindServiceAddress (char *hostp, char *servicep)
{
     void *  handlep;

     struct netconfig *   ncp;
     struct netbuf *      nbp;
     struct nd_hostserv   ndhs;
     struct nd_addrlist * ndalp = (struct nd_addrlist *) 0;


     handlep = setnetconfig ();

     while (ncp = getnetconfig (handlep))
     {
          /*
          **  'ncp' holds a parsed record from
          **  /etc/netconfig.
          **  Is it a transport we can use?
          */
          if (ncp->nc_semantics != SEMANTICS_WE_NEED)
          {
               /*
               **  Wrong communication semantics.
               */
               continue;
          }
          if (strcmp (ncp->nc_protofmly, PROTO_FAMILY_WE_NEED))
          {
               /*
               **  Wrong protocol-family.
               */
               continue;
```

```
            }
            ndhs.h_host = hostp;
            ndhs.h_serv = servicep;
            if (netdir_getbyname (ncp, &ndhs, &ndalp))
            {
                 /*
                 **  The service on the host we requested
                 **  cannot be contacted on this provider.
                 */
                 ndalp = (struct nd_addrlist *) 0;
                 continue;
            }
            /*
            **  We found a host with the service we want
            **  on the provider we can use.  So return
            **  the address of the service.
            */
            break;
       }
       endnetconfig (handlep);
       return  ndalp;
  }
```

The return value of a list of addresses can now be used by the Transport Layer Interface (TLI) routines to connect and communicate with the server. The name-to-address magic that occurred here was done by `netdir_getbyname`. This function uses the `netconfig` record pointed to by `ncp` to find the correct Dynamic Shared Libary (e.g., `/usr/lib/tcpip.so`) and explicitly link it in. It is the code in the library that searches the correct hosts and services (e.g., `/etc/net/inet/hosts` and `/etc/net/inet/services`) and does the transport provider dependent mapping of host and service, to network address. (More information on Dynamic Shared Libraries can be found in *Pearls from the ABI — Dynamic Shared Libraries,* UNIX Review, May 1991.)

An important note with respect to administration is that it is the responsibility of the network provider software package to provide the dynamic shared library to do the name-to-address mapping and to make appropriate entries in `/etc/netconfig` at the time that the package is installed. The local administrator is only responsible for maintaining the hosts and services files.

Port Monitors

Now that we have covered client side facilities we need to examine how servers plug into

the system facilities in an organised manner and make themselves available to clients. It is obvious that servers need to detect connections and service requests from clients. A server could do this themselves. However, this is strongly discouraged. It is recommended that the server have another process do the connection detection and route the service request to it. This surrogate process is called a ‘‘port monitor’’.

A port monitor is a process that *listens* for connections from clients and routes their service requests to the appropriate server. A port monitor can listen for connections from many clients simultaneously. Furthermore, a port monitor is not limited to connecting clients to only one type of service. One port monitor can facilitate the answering and routing of all the client connections and service requests of an entire transport provider on a system. This alleviates numerous server processes from doing their own connection detection and just hanging-out eating-up system resources waiting for clients to call. Instead one port monitor hangs-out and eats-up very little system resources waiting for clients to call.

Port monitors maintain a database of port ids and service pairs, and other service connection information. A client is connected to a particular service based on the port they came in on. This is similar to the `inetd` daemon which is a port monitor of sorts. However, the analogous SVR4 network port monitor, `listen`, can connect a client to a service in more than one way. A server can be forked and exec’d with its standard input, output and error files bound to the connection (called the transport endpoint) with the client. However, this is not a good mechanism for servers that are standing servers or daemons. That is, they are already running, having been started at system boot time and are servicing local clients as well as remote clients. For these kinds of servers, `listen` can route the client to the server by passing a file-descriptor, that is the transport endpoint, to the server along a named pipe. The `lpNet` daemon, a server to the LP Print Service daemon `lpsched`, is such a server. (Yes, daemons can serve other daemons, this was recorded in some religious documentation long before software was invented.)

An important note with respect to administration is that the initial configuration of a port monitor for a particular network provider is the responsibility of the network provider software package. The local administrator is only responsible for maintaining the registry of available services. For software packages that are servers, it is their responsibility to register their service with existing port monitors during their installation procedure.

The NLPS_server

One problem that can arise with this kind of port-to-service scheme is how do you choose a port that represents your service that no one else in the world is using? The only

guaranteed(?) answer is a central registry (e.g., NIC). However, you can reduce the number of conflicts that a port number oriented scheme provides by using a symbolic port id. There is a server provided with SVR4 whose sole purpose it is to route clients using symbolic names of services to the servers that provide the service. This server is called the `nlps_server` or Network Listener Protocol Service server. It complements the `listen` port monitor in that it registers itself as a service on a particular port; clients connecting on that port are routed to it; a brief conversation occurs where the client specifies the service it wants; and then the `nlps_server` makes the final connection of the client to the server.

Most if not all SVR4 services that are network provider independent, use a symbolic name versus a port number, for registry with the port monitor.

Testing and Loopback Providers

The SVR4 Network Support Utilities (NSU) package (which is required to support networking on a system) provides a set of three loopback providers for testing network applications. (These were listed in the `/etc/netconfig` file above.) These loopback providers don't go anywhere or communicate with any other system and they don't require a real network to function. They are completely local and each of the three has specific semantics it emulates.

`ticlts` emulates a connectionless oriented transport provider.

`ticots` emulates a connection oriented transport provider without ''orderly release indication''.

`ticotsord` emulates a connection oriented transport provider with ''orderly release indication''.

Furthermore, they can be used just like any other ''real'' network provider. They are excellent for testing your applications before you put them on a real network.

The Service Access Facility

As the UNIX system has grown, several methods of accessing system services have evolved. For example, the `login` service was managed through `getty` and `getty` was managed through `init`. Whereas system services available through a network connection were managed by `listen` and it was managed through a boot script. The `getty` and the `listen` processes are both port monitors. The problem was that there wasn't a consistent way to get them started and there wasn't a consistent way to manage the services they provided.

The Service Access Facility (SAF) is model and a mechanism for providing procedurally consistent access to system services. That is, it manages port monitors and through them it manages access to system services. The SAF consists of a controlling process and two tiers of administration. The first level of administration is concerned with port monitor administration and the second level is specific to service administration.

The controlling process for the SAF is `/usr/lib/saf/sac`. It is started through `/etc/init` and has a database in `/etc/saf/_sactab` which lists all the port monitors it manages and whether they are to be started or not. This database is managed using the `/usr/sbin/sacadm` command. This is the first tier of administration.

Also, under `/etc/saf` are directories named after names of the port monitors on the system. Port monitors have arbitrary names relating to the network provider or other system access ports they monitor, like `tcp` or `ttymon`. Therefore, the configuration and service information related to the port monitor for `tcp` and `ttymon` would be located in `/etc/saf/tcp/_pmtab` and `/etc/saf/ttymon/_pmtab` respectively. Port monitors and their respective databases of services are the second tier of administration of the SAF. They are managed using the `/usr/sbin/pmadm` command. Also, since the `nlps_server` is closely coupled with the `listen` port monitor (i.e., they share the same database) it falls under port monitor administration. It is managed by the `/usr/sbin/nlsadmin` command.

As an aside, `ttymon` has replaced `getty` for monitoring `tty` lines. It has the same basic functionality of `getty` but it also provides more flexibility in the services it will start.

In Closing ...

The major point in this discussion has been that there are new facilities in UNIX System V Release 4.0 that make the task of writing network client server applications easier. Although at first it may appear to make the job more complex this is the illusion of learning something new. There is a reason for these facilities: To organise that which was growing randomly and provide a basis for future growth.

Bibliography

[1] *UNIX System V Release 4: System Administrator's Guide,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[2] *UNIX System V Release 4: Network User's and Administrator's Guide,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[3] *UNIX System V Release 4: Programmer's Guide: Networking Interfaces,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[4] *UNIX System V Release 4: Programmer's Reference Manual,* Prentice Hall, Englewood Cliffs, NJ, 1990.