The `portmap` *Utility*

E

The `rpcbind` utility replaces the `portmap` utility available in previous releases of SunOS. This appendix is included to help you understand the history of port and network address resolution using the `portmap` utility.

SunOS 4.x RPC-based services use `portmap` as a system registration service. It manages a table of correspondences between ports (logical communications channels) and the services registered at them. It provides a standard way for a client to look up the TCP/IP or UDP/IP port number of an RPC program supported by the server.

System Registration Overview

For client programs to find distributed services on a network, they need a way to look up the network addresses of server programs. Network transport (protocol) services do not provide this function. Their task is to provide process-to-process message transfer across a network (that is, a message is sent to a transport-specific network address). A network address is a logical communications channel—by listening on a specific network address, a process receives messages from the network.

The way a process waits on a network address varies from one operating system to the next, but all provide mechanisms by which a process can synchronize its activity with arriving messages. Messages are not sent across networks to receiving processes, but rather to the network address at which receiving processes pick them up. Network addresses are valuable because they allow message receivers to be specified in a way that is independent of

the conventions of the receiving operating system. TI-RPC, being transport independent, makes no assumptions about the structure of a network address. It uses a universal addresses. This universal address is specified as a null-terminated string of characters. Such universal addresses are translated into local transport addresses by routines specific to each transport provider.

The `rpcbind` protocol defines a network service that provides a standard way for clients to look up the network address of any remote program supported by a server. Because it can be implemented on any transport, it provides a single solution to a general problem that works for all clients, all servers, and all networks.

`portmap` Protocol

The `portmap` program maps RPC program and version numbers to transport-specific port numbers. This program makes dynamic binding of remote programs possible.

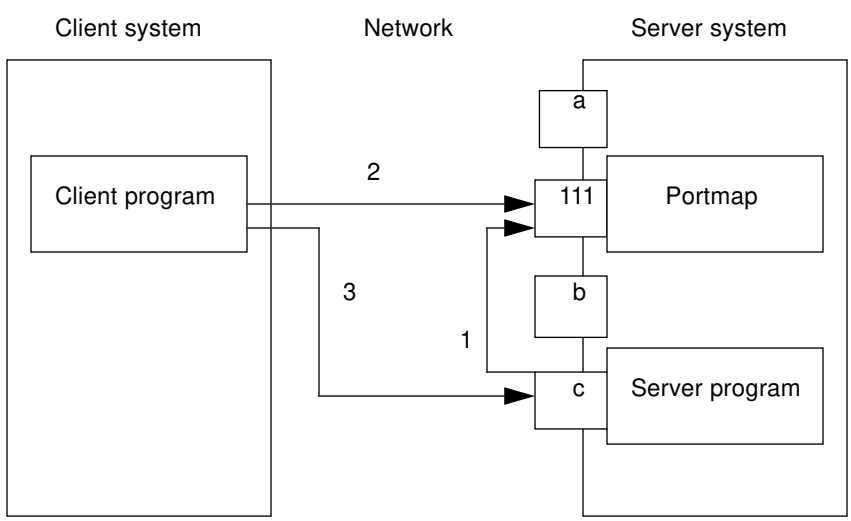

Figure E-1 Typical Portmap Sequence (For TCP/IP Only)

Figure E-1 shows the process:

1. The server registers with `portmap`
2. The client gets the server's port from `portmap`
3. The client calls the server.

The range of reserved port numbers is small and the number of potential remote programs is very large. By running only the port mapper on a well known port, the port numbers of other remote programs can be ascertained by querying the port mapper. In Figure E-1, a, 111, b, and c represent port numbers, where 111 is the assigned portmapper port number.

The port mapper also aids in broadcast RPC. A given RPC program usually has different port number bindings on different machines, so there is no way to directly broadcast to all of these programs. The port mapper, however, does have a fixed port number. So, to broadcast to a given program, the client actually sends its message to the port mapper located at the broadcast address. Each port mapper that receives the broadcast then calls the local service specified by the client. When `portmap` gets the reply from the local service, it returns the reply to the client. The `portmap` protocol specification is shown in Code Example E-1.

Code Example E-1 `portmap` Protocol Specification (in RPC Language)

```
const PMAP_PORT = 111;        /* portmapper port number */
 /*
  * A mapping of (program, version, protocol) to port number
  */
 struct pmap {
    unsigned int prog;
    unsigned int vers;
    unsigned int prot;
    unsigned int port;
 };
 /*
  * Supported values for the "prot" field
  */
 const IPPROTO_TCP = 6; /* protocol number for TCP/IP */
 const IPPROTO_UDP = 17; /* protocol number for UDP/IP */
 /*
  * A list of mappings
  */
 struct pmaplist {
```

```
   pmap map;
   pmaplist *next;
};
/*
 * Arguments to callit
 */
struct call_args {
   unsigned int prog;
   unsigned int vers;
   unsigned int proc;
   opaque args<>;
};
/*
* Results of callit
*/
struct call_result {
   unsigned int port;
   opaque res<>;
};
/*
* Port mapper procedures
*/
program PMAP_PROG {
   version PMAP_VERS {
       void
       PMAPPROC_NULL(void) = 0;
       bool
       PMAPPROC_SET(pmap) = 1;
       bool
       PMAPPROC_UNSET(pmap) = 2;
       unsigned int
       PMAPPROC_GETPORT(pmap) = 3;
       pmaplist
       PMAPPROC_DUMP(void) = 4;
       call_result
       PMAPPROC_CALLIT(call_args) = 5;
   } = 2;
} = 100000;
```

portmap Operation

portmap currently supports two protocols (UDP/IP and TCP/IP). portmap is contacted by talking to it on assigned port number 111 (SUNRPC (5)) on either of these protocols. The following is a description of each of the portmapper procedures.

PMAPPROC_NULL

This procedure does no work. By convention, procedure zero of any protocol takes no parameters and returns no results.

PMAPPROC_SET

When a program first becomes available on a machine, it registers itself with the local port map program. The program passes its program number "prog", version number "vers", transport protocol number "prot", and the port "port" on which it receives service requests. The procedure refuses to establish a mapping if one already exists for the specified port and it is bound. If the mapping exists and the port is not bound, portmap unregisters the port and performs the requested mapping. The procedure returns TRUE if the procedure successfully established the mapping and FALSE otherwise. See also the pmap_set function in the rpc_soc(3N) manpage.

PMAPPROC_UNSET

When a program becomes unavailable, it should unregister itself with the port mapper program on the same machine. The parameters and results have meanings identical to those of PMAPPROC_SET. The protocol and port number fields of the argument are ignored. See also the pmap_unset function in the rpc_soc(3N) manpage.

PMAPPROC_GETPORT

Given a program number "prog", version number "vers", and transport protocol number "prot", this procedure returns the port number on which the program is awaiting call requests. A port value of zeros means the program has not been registered. The "port" field of the argument is ignored. See also the pmap_getport function in the rpc_soc(3N) manpage.

`PMAPPROC_DUMP`

This procedure enumerates all entries in the port mapper's database. The procedure takes no parameters and returns a list of program, version, protocol, and port values. See also the `pmap_getmaps` function in the `rpc_soc(3N)` manpage.

`PMAPPROC_CALLIT`

This procedure allows a caller to call another remote procedure on the same machine without knowing the remote procedure's port number. It is intended for supporting broadcasts to arbitrary remote programs via the well-known port mapper's port. The parameters "prog," "vers," "proc," and the bytes of "args" are the program number, version number, procedure number, and parameters of the remote procedure. See also the `pmap_rmtcall` function in the `rpc_soc(3N)` manpage.

This procedure only sends a response if the procedure was successfully executed and is silent (no response) otherwise. It also returns the remote program's port number, and the bytes of results are the results of the remote procedure.

The port mapper communicates with the remote program using UDP/IP only.

Bibliography

1. Birrell, Andrew D. and Nelson, Bruce Jay; "Implementing Remote Procedure Calls"; *XEROX* CSL-83-7, October 1983.
2. Cheriton, D.; "VMTP: Versatile Message Transaction Protocol," Preliminary Version 0.3; Stanford University, January 1987.
3. Postel, J.; "Transmission Control Protocol - DARPA Internet Program Protocol Specification," *RFC* 793; Information Sciences Institute, September 1981.
4. Postel, J.; "User Datagram Protocol," *RFC* 768; Information Sciences Institute, August 1980.
5. Reynolds, J. & Postel, J.; "Assigned Numbers," *RFC* 923; Information Sciences Institute, October 1984.