## NAME

rpc_svc_calls, svc_dg_enablecache, svc_done, svc_exit, svc_fdset, svc_freeargs, svc_getargs, svc_getreq_common, svc_getreq_poll, svc_getreqset, svc_getrpccaller, svc_pollset, svc_run, svc_sendreply – library routines for RPC servers

## MT-LEVEL

See the **NOTES** section of this page.

## DESCRIPTION

These routines are part of the RPC library which allows C language programs to make procedure calls on other machines across the network.

These routines are associated with the server side of the RPC mechanism. Some of them are called by the server side dispatch function, while others (such as **svc_run( )**) are called when the server is initiated.

In the current implementation, the service transport handle **SVCXPRT** contains a single data area for decoding arguments and encoding results. Therefore, this structure cannot be freely shared between threads that call functions that do this. However, when a server is operating in the Automatic or User MT modes, a copy of this structure is passed to the service dispatch procedure in order to enable concurrent request processing. Under these circumstances, some routines which would otherwise be unsafe, become safe. These are marked as such. Also marked are routines that are unsafe for MT applications, and are not to be used by such applications.

### Routines

**#include <rpc/rpc.h>**

**int svc_dg_enablecache(SVCXPRT** **xprt*, **const unsigned long** *cache_size***);**

This function allocates a duplicate request cache for the service endpoint *xprt*, large enough to hold *cache_size* entries. Once enabled, there is no way to disable caching. This routine returns **1** if space necessary for a cache of the given size was successfully allocated, and **0** otherwise.

This function is safe in MT applications.

**int svc_done(SVCXPRT** **xprt***);**

This function frees resources allocated to service a client request directed to the service endpoint *xprt*. This call pertains only to servers executing in the User MT mode. In the User MT mode, service procedures must invoke this call before returning, either after a client request has been serviced, or after an error or abnormal condition that prevents a reply from being sent. After **svc_done**() is invoked, the service endpoint *xprt* should not be referenced by the service procedure. Server multithreading modes and parameters can be set using the **rpc_control**() call.

This function is safe in MT applications. It will have no effect if invoked in modes other than the User MT mode.

**void svc_exit(void);**

This function when called by any of the RPC server procedure or otherwise, destroys all services registered by the server and causes **svc_run( )** to return.

If RPC server activity is to be resumed, services must be reregistered with the RPC library either through one of the **rpc_svc_create**(3N) functions, or using **xprt_register**(3N).

**svc_exit**( ) has global scope and ends all RPC server activity.

**fd_set svc_fdset;**

A global variable reflecting the RPC server's read file descriptor bit mask. This is only of interest if service implementors do not call **svc_run( )**, but rather do their own asynchronous event processing. This variable is read-only, and it may change after calls to **svc_getreqset( )** or any creation routines. Do not pass its address to **select**(3C)! Instead, pass the address of a copy.

MT applications executing in either the Automatic MT mode or the user MT mode should never read this variable. They should use auxiliary threads to do asynchronous event processing.

**bool_t svc_freeargs(const SVCXPRT** **xprt*, **const xdrproc_t** *inproc*, **caddr_t** *in***);**

A function macro that frees any data allocated by the **RPC/XDR** system when it decoded the arguments to a service procedure using **svc_getargs( )**. This routine returns **TRUE** if the results were successfully freed, and **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

**bool_t svc_getargs(const SVCXPRT** **xprt*, **const xdrproc_t** *inproc*, **caddr_t** *in***);**

A function macro that decodes the arguments of an RPC request associated with the RPC service transport handle *xprt*. The parameter *in* is the address where the arguments will be placed; *inproc* is the XDR routine used to decode the arguments. This routine returns **TRUE** if decoding succeeds, and **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

**void svc_getreq_common(const int** *fd***);**

This routine is called to handle a request on the given file descriptor.

**void svc_getreq_poll(struct pollfd** **pfdp*, **const int** *pollretval***);**

This routine is only of interest if a service implementor does not call **svc_run( )**, but instead implements custom asynchronous event processing. It is called when **poll**(2) has determined that an RPC request has arrived on some RPC file descriptors; *pollretval* is the return value from **poll**(2) and *pfdp* is the array of *pollfd* structures on which the **poll**(2) was done. It is assumed to be an array large enough to contain the maximal number of descriptors allowed.

This function macro is unsafe in MT applications.

**void svc_getreqset(fd_set** **rdfds***);**

This routine is only of interest if a service implementor does not call **svc_run( )**, but instead implements custom asynchronous event processing. It is called when **select**(3C) has determined that an RPC request has arrived on some RPC file descriptors; *rdfds* is the resultant read file descriptor bit mask. The routine returns when all file descriptors associated with the value of *rdfds* have been serviced.

This function macro is unsafe in MT applications.

**struct netbuf *svc_getrpccaller(const SVCXPRT** **xprt***);**

The approved way of getting the network address of the caller of a procedure associated with the RPC service transport handle *xprt*.

This function macro is safe in MT applications.

**void svc_run(void);**

This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using **svc_getreq_poll( )** when one arrives. This procedure is usually waiting for the **poll**(2) library call to return.

Applications executing in the Automatic or User MT modes should invoke this function exactly once. It the Automatic MT mode, it will create threads to service client requests. In the User MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.

**bool_t svc_sendreply(const SVCXPRT** **xprt*, **const xdrproc_t** *outproc*,
**const caddr_t** *out***);**

Called by an RPC service's dispatch routine to send the results of a remote procedure call. The parameter *xprt* is the request's associated transport handle; *outproc* is the XDR routine which is used to encode the results; and *out* is the address of the results. This routine returns **TRUE** if it succeeds, **FALSE** otherwise.

This function macro is safe in MT applications utilizing the Automatic or User MT modes.

## SEE ALSO

**rpcgen**(1), **poll**(2), **rpc**(3N), **rpc_control**(3N), **rpc_svc_create**(3N), **rpc_svc_err**(3N), **rpc_svc_reg**(3N), **select**(3C), **xprt_register**(3N)

## NOTES

**svc_dg_enablecache( )** and **svc_getrpccaller( )** are safe in multithreaded applications. **svc_freeargs( )**, **svc_getargs( )**, and **svc_sendreply( )** are safe in MT applications utilizing the Automatic or User MT modes. **svc_getreq_common( )**, **svc_getreqset( )**, and **svc_getreq_poll( )** are unsafe in multithreaded applications and should be called only from the main thread.