EDOC039

EUROGAM PROJECT

NSF DATA ACQUISITION SYSTEM

Network Tape System - C library

Edition 2.0
October 1990

NSF Software Systems Group
UK Science and Engineering Research Council
Daresbury Laboratory

Introduction

This document describes the C interface library for the Network Tape System (NTS) used by a Tape Server forming part of the NSF data acquisition system.

The library implements the NTS specification described in the document *Network Tape System* (EDOC020.)

It is intended that different versions of the library could exist, utilising different communications media. So far there is only one implementation. This implementation uses a modified version of the Sun RPC library. This library is described in the document *Remote Procedure Call — C library* (EDOC038.)

In order to use the library, programs should include the header file *nettape.h* (which itself includes *rpcreturn.h*) and should be linked with the libraries *libnettape.a* and *librpclib.a* to find the correct RPC calls.

Summary

The library provides pure or "synchronous RPC"; when a remote procedure is called the calling routine waits until a reply or error is received. It also provides "asynchronous RPC"; when a remote procedure is called the calling routine returns immediately and replies are read with a subsequent call. By default the library operates with synchronous RPC.

The library consists of a number of C procedure calls, each one corresponding to a procedure of the Network Tape Server, as described in the *Network Tape System* document. Details of the call parameters that are not described in this document are described in that one. In addition there are some miscellaneous routines for setting up and message printing and two composite routines that use other underlying NTS calls.

The library expects the NTS server to use the UDP port specified in EDOC020. It does not call the server's portmapper to discover the port.

Description

Synchronous RPC

This is the normal form of RPC provided by the standard Sun library. Each time a call is made to a remote procedure control is not returned to the calling program until the remote server has replied (or an error has occurred.)

Asynchronous RPC

This is a modified form of RPC. Each time a call is made to a remote procedure control is immediately returned to the calling program. This program can subsequently call a reply routine when required. Asynchronous RPC allows calls to be batched by sending several RPC requests without waiting for a reply. There is a limit to the number of calls that can be batched; at present this limit is 16.

The reply routine operates in two modes, blocking or non-blocking. In the first mode the routine waits until a reply or error is seen. In the second mode the routine returns immediately even if there is no reply yet.

Input and Returns

In all cases input strings are expected to be null-terminated according to the normal C Language convention. Returned strings are also null-terminated.

Each NTS call returns an integer. The numerical values are defined in *rpcreturn.h*. Other values are procedure dependent and are described in the *Network Tape System* document. The values are shown in the table below.

SUCCESS	The procedure was successful.
RPC_ERROR	The RPC library detected an error.
RPC_NOREPLY	No reply was received from the remote procedure.
NO_CAPABILITY	The routine was unable to determine what capability to use.
NULL_PARAMETER	Some data required by the library routine was not supplied.
UNKNOWN_PROC	Data was returned from an unknown remote procedure.
RPC_ERRORS	RPC statuses are mapped to NTS errors by adding this value to the RPC status value.

It should be noted that two RPC statuses are defined that are not part of the normal Sun implementation. They are both concerned with asynchronous RPC and are shown in the table below.

RPC_NOREPLYYET (RPC status 18)	There has been no reply received from any of the outstanding RPC requests.
RPC_NOCALLS (RPC status 19) Asynchronous RPC request Asynchronous RPC reply	 No RPC requests slots available. The maximum number of requests that can be outstanding is 16. No RPC requests outstanding.

Network Tape System Calls

For all these procedures the value returned by the procedure refers only to the call (not its action or results) when the procedure is used for asynchronous RPC. In all procedures the values returned in parameters are undefined after the procedure has been used for asynchronous RPC. It is sensible in this case to use a NULL pointer for such parameters.

int NTSnullproc ()

This routine does no NTS work. It calls the server's null procedure. This is a procedure that is made available by all RPC servers to allow server testing and timing.

int NTSallocdevice (identifier, req_name, alloc_name, capability)

int identifier;
char *req_dev_name, **alloc_dev_name;
caddr_t *capability;

This routine allocates the device specified by the string *req_name* for exclusive use of the client to the supplied *identifier*.

The actual device name allocated is returned in the string pointed to by *alloc_name*. If *alloc_name* is NULL the returned allocated device name cannot be returned to the caller but will not otherwise cause an error. If the string pointed to by *alloc_name* is NULL, the routine will return a pointer to its own (otherwise) private string. This string may be freed or overwritten on subsequent calls to **NTSallocdevice()** or **NTSdeallocdevice()**.

Capability is a pointer to a buffer for the server supplied opaque access token for subsequent procedure requests. If *capability* is NULL the token cannot be returned. It will be stored within the library and the buffer will be freed or overwritten on subsequent calls to **NTSallocdevice()** or **NTSdeallocdevice()**. If the buffer pointed to by *capability* is NULL the routine will return a pointer to a buffer it has generated itself. This buffer will be freed on subsequent calls to **NTSdeallocdevice()**.

The server generated capability must be supplied for all subsequent NTS calls.

int NTSmountvolume (identifier, capability, volume_name)

int identifier;
caddr_t capability;
char *volume_name;

This routine requests that the volume specified by the string *volume_name* be loaded onto the device currently allocated to the client's *identifier*.

int NTSopenfile (identifier, capability, mode, file_type, rec_length, blk_length, filename)

int identifier;
caddr_t capability;
int mode, file_type, rec_length, blk_length;
char *filename;

The device specified by *identifier* is prepared for data transfer. The string *filename* specifies the file to be accessed.

int NTSreaddata (identifier, capability, blk_rec, offset, count, mode, read_buffer)

int identifier;
caddr_t capability;
int blk_rec, offset, *count, *mode;
char *read_buffer;

This routine transfers data from the device specified by *identifier* to the buffer *read_buffer*. The number of bytes to be transferred should be specified in the integer pointed to by *count*. The number of bytes actually transferred is returned in the same variable. The position in the file block/record at which to start reading should be specified in *offset*. The maximum number of bytes that will be transferred is 1024. On return from the call the integer pointed to by *mode* will contain the mode returned from the remote procedure. If *count* is NULL or the integer pointed to by *count* is 0 a default count of 1024 bytes is requested from the device. If any of *count*, *mode* or *read_buffer* is NULL, the relevant data cannot be returned in that variable but no error is reported.

int NTSwritedata (identifier, capability, blk_rec, offset, mode, data_length, data)

int identifier;

caddr_t capability;

int blk_rec, offset, mode, data_length;

char *data;

This routine transfers data to the device specified by identifier from the buffer *data*. *Data_length* bytes will be transferred, up to a maximum of 1024. The data will be written to the tape starting at position *offset* in block/record *blk_rec*.

int NTSdevicecommand (identifier, capability, magic_number, option, parameter)

int identifier;

caddr_t capability;

int magic_number, option, parameter;

This routine requests a device specific command be issued to the device allocated to *identifier*. The command is specified in *option*. *Magic_number* should be different for each call to this procedure.

int NTSenqdevstatus (identifier, capability, information)

int identifier;

caddr_t capability;

char *information;

This routine returns information about the state of the device specified by *identifier.* The information is a null-terminated string returned in the user supplied buffer *information.* If *information* is NULL nothing can be returned but no error will be reported. The user supplied buffer must be large enough to hold the returned information.

int NTSclosefile (identifier, capability)

int identifier;

caddr_t capability;

This routine requests that the file specified by *identifier* is closed.

NTSdismountvolume (identifier, capability)

int identifier;

caddr_t capability;

This routine reverses the action of **NTSmountvolume**().

int NTSdeallocdevice (identifier, capability)

int identifier;

caddr_t capability;

This routine deallocates the device currently associated with *identifier*. If the buffer for the capability associated with this *identifier* was allocated by **NTSallocdevice()** it will be freed. If the memory used for the string (*alloc_dev_name*) returned from **NTSallocdevice()** was allocated by that routine it will be freed.

int NTSresetclient ()

This routine deallocates all devices for this client. This implies that it also closes all files and dismounts all volumes.

int NTSenqstrstatus (identifier, capability, proc_status, strm_status, information)

int identifier;

caddr_t capability;

int *proc_status, *strm_status;

char *information;

This routine returns information about the stream associated with *identifier*. The status of the last procedure called is returned in the integer pointed to by *proc_status*. The current status of the stream is returned in the integer pointed to by *strm_status*. Extra text information may be reported in the string *information*. If any of the parameters *proc_status*, *strm_status* or *information* are NULL no data can be returned in the corresponding variable, but this does not cause an error. This means it is possible (for example,) to enquire about the current stream status without the necessity to allocate memory for any returned text message if that is not required.

int NTSrpcreply(which, param1, param2, read_buffer)

int *which;

caddr_t param1, param2;

char *read_buffer;

This routine is the general purpose reply from asynchronous RPC. The number of the procedure from which the reply was received is returned in *which*. The contents of the other parameters are reply dependent and map onto the synchronous call parameters.

If the reply is from the **NTSclaim()** procedure *param1* is treated as being of type (char **) and contains the name of the allocated device. If the reply is from the NTSclaim() procedure *param2* is treated as being of type (char **) and contains the name of the capability assigned by the server.

If the reply is from the **NTSreaddata()** procedure *param1* is treated as being of type (int *) and contains the number of bytes read from the server. If the reply is from the **NTSreaddata()** procedure *param2* is treated as being of type (int *) and contains the mode.

If the reply is from the **NTSstrmstatus()** procedure *param1* is treated as being of type (int *) and contains the last procedure status. If the reply is from the **NTSstrmstatus()** procedure *param2* is treated as being of type (int *) and contains the current stream status.

If the reply is from the **NTSreaddata()** procedure *read_buffer* is contains the data read from the server. If the reply is from the **NTSdevstatus()** or **NTSstrmstatus()** procedures *read_buffer* contains the extra text normally returned in the *information* call parameter.

void NTSflushasync ()

This routine is used in asynchronous operation to flush all pending RPC replies. This is done by reading the RPC port until all outstanding replies have been received or have timed out. The replies are discarded.

Composite Network Tape System Calls

These two routines work in synchronous RPC. They set synchronous operation before working and then reset to the previous type of operation before returning. A chunk is a block or a record, depending on how the file was opened.

int NTSreadchunk (identifier, capability, blk_rec, offset, count, mode, read_buffer)

int identifier;

caddr_t capability;

int blk_rec, offset, *count, *mode;

char *read_buffer;

This routine transfers data from the device specified by *identifier* to the buffer *read_buffer*. The number of bytes to be transferred should be specified in the integer pointed to by *count*. The number of bytes actually transferred is returned in the same variable. The position in the file block/record at which to start reading should be specified in *offset*. The routine makes consecutive calls to **NTSreaddata()** until the requested number of bytes has been transferred, end of chunk or end of file is reached or there is an error. On return from the call the integer pointed to by *mode* will contain the mode returned from the remote procedure. If *count* is NULL or the requested number of bytes is 0 reading will continue to the end of the chunk or file. If any of *count*, *mode* or *read_buffer* is NULL, the relevant data cannot be returned in that variable but no error is reported.

int NTSwritechunk (identifier, capability, blk_rec, offset, data_length, data)

int identifier;

caddr_t capability;

int blk_rec, offset, mode, data_length;

char *data;

This routine transfers data to the device specified by *identifier* from the buffer *data*. *Data_length* bytes will be transferred. The data will be written starting at offset *offset* in the chunk *blk_rec*, where a chunk is a block or record depending on how the file was opened. The data will be written as one chunk. The routine works by making consecutive calls to **NTSwritedata()**.

Miscellaneous Routines

char *NTSinitialise (hostname)

char *hostname;

This routine initialises the C library to use the tape service provided by the system identified by *hostname*. The routine returns a static copy of *hostname*.

int NTSasynchronous (set, blocking)

int set, blocking;

This routine switches the library into synchronous or asynchronous operation.

If the least significant 2 bits of *set* have the value 1 or 3 the library will subsequently operate asynchronously. Values 0 and 2 mean that the library will subsequently operate synchronously. If *set* has a value greater than 1 the second least significant bit will be used to override the *blocking* parameter.

If *blocking* has a value of 1 subsequent calls to **NTSrpcreply()** will block until a reply or an error has been received. Value 0 means that **NTSrpcreply()** returns immediately. Other values have undefined effects. *Blocking* is ignored if *set* has a value greater than 1.

The routine returns the previous value of *set* in its least significant bit and the previous value of *blocking* in the next bit.

Although the meaning of *set* is rather overloaded it can be used to change the operation and blocking action of the library and then to reset them to their previous states by recalling the routine with the previously returned value in *set*.

char *NTSversion()

This routine returns the current library version string.

char *NTSperror (string, errnum)

char *string;

int errnum;

This routine produces a short error message on the standard error describing the error referred to by *errnum*. If *string* is not a NULL pointer and does not point to a null string, the string it points to is printed, followed by a colon, followed by a space, followed by the message. If *string* is a NULL pointer or points to a null string, just the message is printed, followed by a NEWLINE. The message consists of the words "NTS server error " followed by *errnum*, followed by a NEWLINE, followed by a description of the error. *Errnum* is expected to be an error that was previously returned by one of the NTS calls described above. If *errnum* refers to an RPC error a description of that error is printed on the console.

char *NTSpstate (string, state)

char *string;

int state;

This routine produces a short message on the standard error describing the stream state referred to by *state*. If *string* is not a NULL pointer and does not point to a null string, the string it points to is printed, followed by a colon, followed by a space, followed by the message. If string is a NULL pointer or points to a null string, just the message is printed, followed by a NEWLINE. The message consists of the words "NTS stream state" followed by *state* followed by a description of the state.