```
**********************************************************
   15073 Thu Dec 19 23:35:27 2013
new/usr/src/man/man1/rpcgen.1
4329 rpcgen(1): Four output files are generated by default
**********************************************************
'\" te
.\" Copyright (C) 2009, Sun Microsystems, Inc. All Rights Reserved
.\" Copyright 1989 AT&T
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" fields enclosed by brackets "[]" replaced with your own identifying informat
.TH RPCGEN 1 "Dec 16, 2013"
.TH RPCGEN 1 "Aug 24, 2009"
.SH NAME
rpcgen \- an RPC protocol compiler
.SH SYNOPSIS
.LP
.nf
\fBrpcgen\fR \fIinfile\fR
.fi

.LP
.nf
\fBrpcgen\fR [\fB-a\fR] [\fB-A\fR] [\fB-b\fR] [\fB-C\fR] [\fB-D\fR \fIname\fR [=
     [\fB-I\fR [\fB-K\fR \fIseconds\fR]] [\fB-L\fR] [\fB-M\fR] [\fB-N\fR] [\fB-
     [\fB-Y\fR \fIpathname\fR] \fIinfile\fR
.fi

.LP
.nf
\fBrpcgen\fR [\fB-c\fR | \fB-h\fR | \fB-l\fR | \fB-m\fR | \fB-t\fR | \fB-Sc\fR |
     [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
.fi

.LP
.nf
\fBrpcgen\fR [\fB-s\fR \fInettype\fR] [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
.fi

.LP
.nf
\fBrpcgen\fR [\fB-n\fR \fInetid\fR] [\fB-o\fR \fIoutfile\fR] [\fIinfile\fR]
.fi

.SH DESCRIPTION
.sp
.LP
The \fBrpcgen\fR utility is a tool that generates C code to implement an
\fBRPC\fR protocol. The input to \fBrpcgen\fR is a language similar to C known
as \fBRPC\fR Language (Remote Procedure Call Language).
.sp
.LP
The \fBrpcgen\fR utility is normally used as in the first synopsis where it
takes an input file and generates four output files. If the \fIinfile\fR is
takes an input file and generates three output files. If the \fIinfile\fR is
named \fBproto.x\fR, then \fBrpcgen\fR generates a header in \fBproto.h\fR,
\fBXDR\fR routines in \fBproto_xdr.c\fR, server-side stubs in
\fBproto_svc.c\fR, and client-side stubs in \fBproto_clnt.c\fR. With the
\fB-T\fR option, it also generates the \fBRPC\fR dispatch table in
\fBproto_tbl.i\fR.
.sp
.LP
\fBrpcgen\fR can also generate sample client and server files that can be
customized to suit a particular application. The \fB-Sc\fR, \fB-Ss\fR, and
\fB-Sm\fR options generate sample client, server and makefile, respectively.
The \fB-a\fR option generates all files, including sample files. If the infile
```



```
is \fBproto.x\fR, then the client side sample file is written to
\fBproto_client.c\fR, the server side sample file to \fBproto_server.c\fR and
the sample makefile to \fBmakefile.proto\fR.
.sp
.LP
The server created can be started both by the port monitors (for example,
\fBinetd\fR or \fBlisten\fR) or by itself. When it is started by a port
monitor, it creates servers only for the transport for which the file
descriptor \fB0\fR was passed. The name of the transport must be specified by
setting up the environment variable \fBPM_TRANSPORT\fR. When the server
generated by \fBrpcgen\fR is executed, it creates server handles for all the
transports specified in the \fBNETPATH\fR environment variable, or if it is
unset, it creates server handles for all the visible transports from the
\fB/etc/netconfig\fR file. Note: the transports are chosen at run time and not
at compile time. When the server is self-started, it backgrounds itself by
default. A special define symbol \fBRPC_SVC_FG\fR can be used to run the server
process in foreground.
.sp
.LP
The second synopsis provides special features which allow for the creation of
more sophisticated \fBRPC\fR servers. These features include support for
user-provided \fB#defines\fR and \fBRPC\fR dispatch tables. The entries in the
\fBRPC\fR dispatch table contain:
.RS +4
.TP
.ie t \(bu
.el o
pointers to the service routine corresponding to that procedure
.RE
.RS +4
.TP
.ie t \(bu
.el o
a pointer to the input and output arguments
.RE
.RS +4
.TP
.ie t \(bu
.el o
the size of these routines
.RE
.sp
.LP
A server can use the dispatch table to check authorization and then to execute
the service routine. A client library can use the dispatch table to deal with
the details of storage management and \fBXDR\fR data conversion.
.sp
.LP
The other three synopses shown above are used when one does not want to
generate all the output files, but only a particular one. See the EXAMPLES
section below for examples of \fBrpcgen\fR usage. When \fBrpcgen\fR is executed
with the \fB-s\fR option, it creates servers for that particular class of
transports. When executed with the \fB-n\fR option, it creates a server for the
transport specified by \fInetid\fR. If \fIinfile\fR is not specified,
\fBrpcgen\fR accepts the standard input.
.sp
.LP
All the options mentioned in the second synopsis can be used with the other
three synopses, but the changes are made only to the specified output file.
.sp
.LP
The C preprocessor \fBcc\fR \fB-E\fR is run on the input file before it is
actually interpreted by \fBrpcgen\fR. For each type of output file,
\fBrpcgen\fR defines a special preprocessor symbol for use by the \fBrpcgen\fR
programmer:
.sp
```

```
126 .ne 2
127 .na
128 \fB\fBRPC_HDR\fR\fR
129 .ad
130 .RS 12n
131 defined when compiling into headers
132 .RE

134 .sp
135 .ne 2
136 .na
137 \fB\fBRPC_XDR\fR\fR
138 .ad
139 .RS 12n
140 defined when compiling into \fBXDR\fR routines
141 .RE

143 .sp
144 .ne 2
145 .na
146 \fB\fBRPC_SVC\fR\fR
147 .ad
148 .RS 12n
149 defined when compiling into server-side stubs
150 .RE

152 .sp
153 .ne 2
154 .na
155 \fB\fBRPC_CLNT\fR\fR
156 .ad
157 .RS 12n
158 defined when compiling into client-side stubs
159 .RE

161 .sp
162 .ne 2
163 .na
164 \fB\fBRPC_TBL\fR\fR
165 .ad
166 .RS 12n
167 defined when compiling into \fBRPC\fR dispatch tables
168 .RE

170 .sp
171 .LP
172 Any line beginning with ''\fB%\fR'' is passed directly into the output file,
173 uninterpreted by \fBrpcgen\fR, except that the leading ''\fB%\fR" is stripped
174 off. To specify the path name of the C preprocessor, use the \fB-Y\fR flag.
175 .sp
176 .LP
177 For every data type referred to in \fIinfile\fR, \fBrpcgen\fR assumes that
178 there exists a routine with the string \fBxdr_\fR prepended to the name of the
179 data type. If this routine does not exist in the \fBRPC\fR/\fBXDR\fR library,
180 it must be provided. Providing an undefined data type allows customization of
181 \fBXDR\fR routines.
182 .SS "Server Error Reporting"
183 .sp
184 .LP
185 By default, errors detected by \fBproto_svc.c\fR is reported to standard error
186 and/or the system log.
187 .sp
188 .LP
189 This behavior can be overridden by compiling the file with a definition of
190 \fBRPC_MSGOUT\fR, for example, \fB-DRPC_MSGOUT=mymsgfunc\fR. The function
191 specified is called to report errors. It must conform to the following
```



```
192 \fBprintf\fR-like signature:
193 .sp
194 .in +2
195 .nf
196 extern void RPC_MSGOUT(const char *fmt, ...);
197 .fi
198 .in -2
199 .sp

201 .SH OPTIONS
202 .sp
203 .LP
204 The following options are supported:
205 .sp
206 .ne 2
207 .na
208 \fB\fB-a\fR\fR
209 .ad
210 .RS 18n
211 Generates all files, including sample files.
212 .RE

214 .sp
215 .ne 2
216 .na
217 \fB\fB-A\fR\fR
218 .ad
219 .RS 18n
220 Enables the Automatic \fBMT\fR mode in the server main program. In this mode,
221 the \fBRPC\fR library automatically creates threads to service client requests.
222 This option generates multithread-safe stubs by implicitly turning on the
223 \fB-M\fR option. Server multithreading modes and parameters can be set using
224 the \fBrpc_control\fR(3NSL) call. \fBrpcgen\fR generated code does not change
225 the default values for the Automatic \fBMT\fR mode.
226 .RE

228 .sp
229 .ne 2
230 .na
231 \fB\fB-b\fR\fR
232 .ad
233 .RS 18n
234 Backward compatibility mode. Generates transport-specific \fBRPC\fR code for
235 older versions of the operating system.
236 .RE

238 .sp
239 .ne 2
240 .na
241 \fB\fB-c\fR\fR
242 .ad
243 .RS 18n
244 Compiles into \fBXDR\fR routines.
245 .RE

247 .sp
248 .ne 2
249 .na
250 \fB\fB-C\fR\fR
251 .ad
252 .RS 18n
253 Generates header and stub files which can be used with ANSI C compilers.
254 Headers generated with this flag can also be used with C++ programs.
255 .RE

257 .sp
```

```
.ne 2
.na
\fB\fB-D\fR\fIname\fR\fB[=\fR\fIvalue\fR\fB]\fR\fR
.ad
.RS 18n
Defines a symbol \fIname\fR. Equivalent to the \fB#define\fR directive in the
source. If no \fIvalue\fR is given, \fIvalue\fR is defined as \fB1\fR. This
option can be specified more than once.
.RE

.sp
.ne 2
.na
\fB\fB-h\fR\fR
.ad
.RS 18n
Compiles into \fBC\fR data-definitions (a header). The \fB-T\fR option can be
used in conjunction to produce a header which supports \fBRPC\fR dispatch
tables.
.RE

.sp
.ne 2
.na
\fB\fB-i\fR \fIsize\fR\fR
.ad
.RS 18n
Size at which to start generating inline code. This option is useful for
optimization. The default \fIsize\fR is 5.
.RE

.sp
.ne 2
.na
\fB\fB-I\fR\fR
.ad
.RS 18n
Compiles support for \fBinetd\fR(1M) in the server side stubs. Such servers can
be self-started or can be started by \fBinetd\fR. When the server is
self-started, it backgrounds itself by default. A special define symbol
\fBRPC_SVC_FG\fR can be used to run the server process in foreground, or the
user can simply compile without the \fB-I\fR option.
.sp
If there are no pending client requests, the \fBinetd\fR servers exit after 120
seconds (default). The default can be changed with the \fB-K\fR option. All of
the error messages for \fBinetd\fR servers are always logged with
\fBsyslog\fR(3C).
.sp
\fBNote:\fR This option is supported for backward compatibility only. It should
always be used in conjunction with the \fB-b\fR option which generates backward
compatibility code. By default (that is, when \fB-b\fR is not specified),
\fBrpcgen\fR generates servers that can be invoked through portmonitors.
.RE

.sp
.ne 2
.na
\fB\fB-K\fR \fIseconds\fR\fR
.ad
.RS 18n
By default, services created using \fBrpcgen\fR and invoked through port
monitors wait 120 seconds after servicing a request before exiting. That
interval can be changed using the \fB-K\fR flag. To create a server that exits
immediately upon servicing a request, use \fB-K\fR \fB0\fR. To create a server
that never exits, the appropriate argument is \fB-K\fR \fB\(mi1\fR\&.
.sp
```



```
When monitoring for a server, some portmonitors, like \fBlisten\fR(1M),
\fBalways\fR spawn a new process in response to a service request. If it is
known that a server are used with such a monitor, the server should exit
immediately on completion. For such servers, \fBrpcgen\fR should be used with
\fB-K\fR \fB0\fR.
.RE

.sp
.ne 2
.na
\fB\fB-l\fR\fR
.ad
.RS 18n
Compiles into client-side stubs.
.RE

.sp
.ne 2
.na
\fB\fB-L\fR\fR
.ad
.RS 18n
When the servers are started in foreground, uses \fBsyslog\fR(3C) to log the
server errors instead of printing them on the standard error.
.RE

.sp
.ne 2
.na
\fB\fB-m\fR\fR
.ad
.RS 18n
Compiles into server-side stubs, but do not generate a "main" routine. This
option is useful for doing callback-routines and for users who need to write
their own "main" routine to do initialization.
.RE

.sp
.ne 2
.na
\fB\fB-M\fR\fR
.ad
.RS 18n
Generates multithread-safe stubs for passing arguments and results between
\fBrpcgen\fR-generated code and user written code. This option is useful for
users who want to use threads in their code.
.RE

.sp
.ne 2
.na
\fB\fB-N\fR\fR
.ad
.RS 18n
This option allows procedures to have multiple arguments. It also uses the
style of parameter passing that closely resembles C. So, when passing an
argument to a remote procedure, you do not have to pass a pointer to the
argument, but can pass the argument itself. This behavior is different from the
old style of \fBrpcgen\fR-generated code. To maintain backward compatibility,
this option is not the default.
.RE

.sp
.ne 2
.na
\fB\fB-n\fR \fInetid\fR\fR
```

```
390 .ad
391 .RS 18n
392 Compiles into server-side stubs for the transport specified by \fInetid\fR.
393 There should be an entry for \fInetid\fR in the \fBnetconfig\fR database. This
394 option can be specified more than once, so as to compile a server that serves
395 multiple transports.
396 .RE

398 .sp
399 .ne 2
400 .na
401 \fB\fB-o\fR \fIoutfile\fR\fR
402 .ad
403 .RS 18n
404 Specifies the name of the output file. If none is specified, standard output is
405 used (\fB-c\fR, \fB-h\fR, \fB-l\fR, \fB-m\fR, \fB-n\fR, \fB-s\fR, \fB-Sc\fR,
406 \fB-Sm\fR, \fB-Ss\fR, and \fB-t\fR modes only).
407 .RE

409 .sp
410 .ne 2
411 .na
412 \fB\fB-s\fR \fInettype\fR\fR
413 .ad
414 .RS 18n
415 Compiles into server-side stubs for all the transports belonging to the class
416 \fInettype\fR. The supported classes are \fBnetpath\fR, \fBvisible\fR,
417 \fBcircuit_n\fR, \fBcircuit_v\fR, \fBdatagram_n\fR, \fBdatagram_v\fR,
418 \fBtcp\fR, and \fBudp\fR (see \fBrpc\fR(3NSL) for the meanings associated with
419 these classes). This option can be specified more than once. \fBNote:\fR The
420 transports are chosen at run time and not at compile time.
421 .RE

423 .sp
424 .ne 2
425 .na
426 \fB\fB-Sc\fR\fR
427 .ad
428 .RS 18n
429 Generates sample client code that uses remote procedure calls.
430 .RE

432 .sp
433 .ne 2
434 .na
435 \fB\fB-Sm\fR\fR
436 .ad
437 .RS 18n
438 Generates a sample Makefile which can be used for compiling the application.
439 .RE

441 .sp
442 .ne 2
443 .na
444 \fB\fB-Ss\fR\fR
445 .ad
446 .RS 18n
447 Generates sample server code that uses remote procedure calls.
448 .RE

450 .sp
451 .ne 2
452 .na
453 \fB\fB-t\fR\fR
454 .ad
455 .RS 18n
```



```
456 Compiles into \fBRPC\fR dispatch table.
457 .RE

459 .sp
460 .ne 2
461 .na
462 \fB\fB-T\fR\fR
463 .ad
464 .RS 18n
465 Generates the code to support \fBRPC\fR dispatch tables.
466 .sp
467 The options \fB-c\fR, \fB-h\fR, \fB-l\fR, \fB-m\fR, \fB-s\fR, \fB-Sc\fR,
468 \fB-Sm\fR, \fB-Ss\fR, and \fB-t\fR are used exclusively to generate a
469 particular type of file, while the options \fB-D\fR and \fB-T\fR are global and
470 can be used with the other options.
471 .RE

473 .sp
474 .ne 2
475 .na
476 \fB\fB-v\fR\fR
477 .ad
478 .RS 18n
479 Displays the version number.
480 .RE

482 .sp
483 .ne 2
484 .na
485 \fB\fB-Y\fR \fIpathname\fR\fR
486 .ad
487 .RS 18n
488 Gives the name of the directory where \fBrpcgen\fR starts looking for the C
489 preprocessor.
490 .RE

492 .SH OPERANDS
493 .sp
494 .LP
495 The following operand is supported:
496 .sp
497 .ne 2
498 .na
499 \fB\fIinfile\fR\fR
500 .ad
501 .RS 10n
502 input file
503 .RE

505 .SH EXAMPLES
506 .LP
507 \fBExample 1 \fRGenerating the output files and dispatch table
508 .sp
509 .LP
510 The following entry

512 .sp
513 .in +2
514 .nf
515 example% \fBrpcgen -T prot.x\fR
516 .fi
517 .in -2
518 .sp

520 .sp
521 .LP
```

```
522 generates all the five files: \fBprot.h\fR, \fBprot_clnt.c\fR,
523 \fBprot_svc.c\fR, \fBprot_xdr.c\fR, and \fBprot_tbl.i\fR.

525 .LP
526 \fBExample 2 \fRSending headers to standard output
527 .sp
528 .LP
529 The following example sends the C data-definitions (header) to the standard
530 output:

532 .sp
533 .in +2
534 .nf
535 example% \fBrpcgen -h prot.x\fR
536 .fi
537 .in -2
538 .sp

540 .LP
541 \fBExample 3 \fRSending a test version
542 .sp
543 .LP
544 To send the test version of the \fB-DTEST\fR, server side stubs for all the
545 transport belonging to the class \fBdatagram_n\fR to standard output, use:

547 .sp
548 .in +2
549 .nf
550 example% \fBrpcgen -s datagram_n -DTEST prot.x\fR
551 .fi
552 .in -2
553 .sp

555 .LP
556 \fBExample 4 \fRCreating server side stubs
557 .sp
558 .LP
559 To create the server side stubs for the transport indicated by \fInetid\fR
560 \fBtcp\fR, use:

562 .sp
563 .in +2
564 .nf
565 example% \fBrpcgen -n tcp -o prot_svc.c prot.x\fR
566 .fi
567 .in -2
568 .sp

570 .SH EXIT STATUS
571 .sp
572 .ne 2
573 .na
574 \fB\fB0\fR\fR
575 .ad
576 .RS 6n
577 Successful operation.
578 .RE

580 .sp
581 .ne 2
582 .na
583 \fB\fB>0\fR\fR
584 .ad
585 .RS 6n
586 An error occurred.
587 .RE
```



```
589 .SH SEE ALSO
590 .sp
591 .LP
592 \fBinetd\fR(1M), \fBlisten\fR(1M), \fBrpc\fR(3NSL), \fBrpc_control\fR(3NSL),
593 \fBrpc_svc_calls\fR(3NSL), \fBsyslog\fR(3C), \fBnetconfig\fR(4),
594 \fBattributes\fR(5)
595 .sp
596 .LP
597 The \fBrpcgen\fR chapter in the \fIONC+ Developer\&'s Guide\fR manual.
```

```
**********************************************************
    6960 Thu Dec 19 23:35:27 2013
new/usr/src/man/man3c/rwlock.3c
4327 rwlock(3c): Formatting issues and typos
**********************************************************
   1 '\" te
   2 .\"  Copyright (c) 1998 Sun Microsystems, Inc.  All Rights Reserved
   3 .\" The contents of this file are subject to the terms of the Common Development
   4 .\" You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE or http:
   5 .\" When distributing Covered Code, include this CDDL HEADER in each file and in
   6 .TH RWLOCK 3C "Dec 19, 2013"
   6 .TH RWLOCK 3C "May 14, 1998"
   7 .SH NAME
   8 rwlock, rwlock_init, rwlock_destroy, rw_rdlock, rw_wrlock, rw_tryrdlock,
   9 rw_trywrlock, rw_unlock \- multiple readers, single writer locks
  10 .SH SYNOPSIS
  11 .LP
  12 .nf
  13 cc -mt [ \fIflag\fR... ] \fIfile\fR...[ \fIlibrary\fR... ]

  15 #include <synch.h>

  17 \fBint\fR \fBrwlock_init\fR(\fBrwlock_t *\fR\fIrwlp\fR, \fBint\fR \fItype\fR, \f
  17 \fBint\fR \fBrwlock_init\fR(\fBrwlock_t *\fR\fIrwlp\fR, \fBint\fR \fItype\fR, \f
  18 .fi

  20 .LP
  21 .nf
  22 \fBint\fR \fBrwlock_destroy\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  23 .fi

  25 .LP
  26 .nf
  27 \fBint\fR \fBrw_rdlock\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  28 .fi

  30 .LP
  31 .nf
  32 \fBint\fR \fBrw_wrlock\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  33 .fi

  35 .LP
  36 .nf
  37 \fBint\fR \fBrw_unlock\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  38 .fi

  40 .LP
  41 .nf
  42 \fBint\fR \fBrw_tryrdlock\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  43 .fi

  45 .LP
  46 .nf
  47 \fBint\fR \fBrw_trywrlock\fR(\fBrwlock_t *\fR\fIrwlp\fR);
  48 .fi

  50 .SH DESCRIPTION
  51 .sp
  52 .LP
  53 Many threads can have simultaneous read-only access to data, while only one
  54 thread can have write access at any given time. Multiple read access with
  55 single write access is controlled by locks, which are generally used to protect
  56 data that is frequently searched.
  57 .sp
  58 .LP
  59 Readers/writer locks can synchronize threads in this process and other
```



```
  60 processes if they are allocated in writable memory  and shared among
  61 cooperating processes (see \fBmmap\fR(2)), and are initialized for this
  62 purpose.
  63 .sp
  64 .LP
  65 Additionally, readers/writer locks must be initialized prior to use.
  66 The readers/writer lock pointed to by \fIrwlp\fR is
  66 \fBrwlock_init()\fR The readers/writer lock pointed to by \fIrwlp\fR is
  67 initialized by  \fBrwlock_init()\fR. A readers/writer lock is capable of having
  68 several types of behavior, which is specified by \fItype\fR. \fIarg\fR is
  68 several types of behavior, which is specified by \fBtype\fR. \fIarg\fR is
  69 currently not used, although a future type may define  new behavior parameters
  70 by way of  \fIarg\fR.
  71 .sp
  72 .LP
  73 The \fItype\fR argument can be one of the following:
  74 .sp
  75 .ne 2
  76 .na
  77 \fB\fBUSYNC_PROCESS\fR \fR
  78 .ad
  79 .RS 18n
  80 The readers/writer lock can synchronize threads in this process and other
  81 processes. The readers/writer lock should be initialized by only one process.
  82 \fIarg\fR is ignored. A readers/writer lock initialized with this type, must be
  83 allocated in memory shared between processses, i.e. either in Sys V shared
  84 memory (see \fBshmop\fR(2)) or in memory mapped to a file (see \fBmmap\fR(2)).
  85 It is illegal to initialize the object this way and to not allocate it in such
  86 shared memory.
  87 .RE

  89 .sp
  90 .ne 2
  91 .na
  92 \fB\fBUSYNC_THREAD\fR \fR
  93 .ad
  94 .RS 18n
  95 The readers/writer lock can synchronize  threads in this process, only.
  96 \fIarg\fR is ignored.
  97 .RE

  99 .sp
 100 .LP
 101 Additionally, readers/writer locks can be initialized by allocation  in zeroed
 102 memory. A \fItype\fR of \fBUSYNC_THREAD\fR is assumed in this case. Multiple
 102 memory. A \fBtype\fR of \fBUSYNC_THREAD\fR is assumed in this case. Multiple
 103 threads must not simultaneously initialize the same  readers/writer lock. And a
 104 readers/writer lock must not be re-initialized while in use by other threads.
 105 .sp
 106 .LP
 107 The following are default readers/writer lock initialization (intra-process):
 108 .sp
 109 .in +2
 110 .nf
 111 rwlock_t rwlp;
 112 rwlock_init(&rwlp, NULL, NULL);

 114 .fi
 115 .in -2

 117 .sp
 118 .LP
 119 or
 120 .sp
 121 .in +2
 122 .nf
```

```
rwlock_init(&rwlp, USYNC_THREAD, NULL);
.fi
.in -2

.sp
.LP
or
.sp
.in +2
.nf
rwlock_t  rwlp  =  DEFAULTRWLOCK;
.fi
.in -2

.sp
.LP
The following is a customized readers/writer lock  initialization
(inter-process):
.sp
.in +2
.nf
rwlock_init(&rwlp, USYNC_PROCESS, NULL);
.fi
.in -2

.sp
.LP
Any state associated with the readers/writer lock pointed to by \fIrwlp\fR are
destroyed by  \fBrwlock_destroy()\fR and the readers/writer lock storage space
is not released.
.sp
.LP
\fBrw_rdlock()\fR gets a read lock on the readers/writer lock pointed to by
\fIrwlp\fR. If the readers/writer lock is currently locked for writing, the
calling thread blocks until the write lock is freed. Multiple threads may
simultaneously hold a read lock on a  readers/writer lock.
.sp
.LP
\fBrw_tryrdlock()\fR tries to get a read lock on the readers/writer lock pointed
\fIrw_tryrdlock()\fR trys to get a read lock on the readers/writer lock pointed
to by \fIrwlp\fR. If the readers/writer lock is locked for writing, it returns
an error; otherwise, the read lock is acquired.
.sp
.LP
\fBrw_wrlock()\fR gets a write lock on the readers/writer lock pointed to by
\fIrwlp\fR. If the readers/writer lock is currently locked for reading or
writing, the calling thread blocks until all the read and write locks are
freed. At any given time, only one thread may have a write lock on a
readers/writer lock.
.sp
.LP
\fBrw_trywrlock()\fR tries to get a write lock on the readers/writer lock
\fIrw_trywrlock()\fR trys to get a write lock on the readers/writer lock
pointed to by \fIrwlp\fR. If the readers/writer lock is currently locked for
reading or writing, it returns an error.
.sp
.LP
\fBrw_unlock()\fR unlocks a readers/writer lock pointed to by  \fIrwlp\fR, if
the readers/writer lock is locked and the calling thread holds the lock for
either reading or writing. One of the other threads that is waiting for the
readers/writer  lock to be freed will be unblocked, provided there are other
readers/writer  lock to be freed will be unblocked, provided there is other
waiting threads. If the calling thread does not hold the lock for either
reading or writing, no error status is returned, and the program’s behavior is
unknown.
.SH RETURN VALUES
```



```
.sp
.LP
If successful, these functions return \fB0\fR. Otherwise, a non-zero value is
returned to indicate the error.
.SH ERRORS
.sp
.LP
The \fBrwlock_init()\fR function will fail if:
.sp
.ne 2
.na
\fB\fBEINVAL\fR \fR
.ad
.RS 11n
\fItype\fR is invalid.
\fBtype\fR is invalid.
.RE

.sp
.LP
The  \fBrw_tryrdlock()\fR or \fBrw_trywrlock()\fR functions will fail if:
.sp
.ne 2
.na
\fB\fBEBUSY\fR \fR
.ad
.RS 10n
The readers/writer lock pointed to by \fIrwlp\fR was already locked.
The reader or writer lock pointed to by \fIrwlp\fR was already locked.
.RE

.sp
.LP
These functions may fail if:
.sp
.ne 2
.na
\fB\fBEFAULT\fR \fR
.ad
.RS 11n
\fIrwlp\fR or \fIarg\fR points to an illegal address.
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5) for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
MT-Level        MT-Safe
.TE

.SH SEE ALSO
.sp
.LP
\fBmmap\fR(2), \fBattributes\fR(5)
.SH NOTES
.sp
.LP
```

```
250 These interfaces also available by way of:
251 .sp
252 .LP
253 \fB#include\fR \fB<thread.h>\fR
254 .sp
255 .LP
256 If multiple threads are waiting for a readers/writer lock, the acquisition
257 order is random by default. However, some implementations may bias acquisition
258 order to avoid depriving writers. The current implementation favors writers
259 over readers.
```

```
**********************************************************
   20636 Thu Dec 19 23:35:28 2013
new/usr/src/man/man3nsl/rpc_clnt_create.3nsl
4340 rpc_clnt_create(3nsl): rpc_createerr is a variable, not a function
**********************************************************
'\" te
.\"  Copyright 1989 AT&T
.\" Copyright (C) 2009, Sun Microsystems, Inc. All Rights Reserved
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH RPC_CLNT_CREATE 3NSL "Dec 16, 2013"
.TH RPC_CLNT_CREATE 3NSL "Jul 23, 2009"
.SH NAME
rpc_clnt_create, clnt_control, clnt_create, clnt_create_timed,
clnt_create_vers, clnt_create_vers_timed, clnt_destroy, clnt_dg_create,
clnt_pcreateerror, clnt_raw_create, clnt_spcreateerror, clnt_tli_create,
clnt_tp_create, clnt_tp_create_timed, clnt_vc_create, rpc_createerr,
clnt_door_create \- library routines for dealing with creation and manipulation
of CLIENT handles
.SH SYNOPSIS
.LP
.nf
#include <rpc/rpc.h>

\fBbool_t\fR \fBclnt_control\fR(\fBCLIENT *\fR\fIclnt\fR, \fBconst uint_t\fR \fI
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create\fR(\fBconst char *\fR\fIhost\fR, \fBconst rpcprog_t
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst char *\fR\fInettype\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_timed\fR(\fBconst char *\fR\fIhost\fR, \fBconst rpc
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst\fR \fInettype\fR,
     \fBconst struct timeval *\fR\fItimetout\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_vers\fR (\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBrpcvers_t *\fR\fIvers_outp\fR,
     \fBconst rpcvers_t\fR \fIvers_low\fR, \fBconst rpcvers_t\fR \fIvers_high\fR
     \fBconst char *\fR\fInettype\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_create_vers_timed\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBrpcvers_t *\fR\fIvers_outp\fR,
     \fBconst rpcvers_t\fR \fIvers_low\fR, \fBconst rpcvers_t\fR \fIvers_high\fR
     \fBchar *\fR\fInettype\fR, \fBconst struct timeval *\fR\fItimeout\fR);
.fi

.LP
.nf
\fBvoid\fR \fBclnt_destroy\fR(\fBCLIENT *\fR\fIclnt\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_dg_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netbuf *\fR\fIsvcaddr\fR, \fBconst rpcprog_t\fR \fIprognum\
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR,
     \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBvoid\fR \fBclnt_pcreateerror\fR(\fBconst char *\fR\fIs\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_raw_create\fR(\fBconst rpcprog_t\fR \fIprognum\fR,
     \fBconst rpcvers_t\fR \fIversnum\fR);
.fi

.LP
.nf
\fBchar *\fR\fBclnt_spcreateerror\fR(\fBconst char *\fR\fIs\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tli_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBconst struct netbuf *\fR\fI
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst uint_t\fR \fIsendsz\fR, \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tp_create\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR);
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_tp_create_timed\fR(\fBconst char *\fR\fIhost\fR,
     \fBconst rpcprog_t\fR \fIprognum\fR, \fBconst rpcvers_t\fR \fIversnum\fR,
     \fBconst struct netconfig *\fR\fInetconf\fR, \fBconst struct timeval *\fR\f
.fi

.LP
.nf
\fBCLIENT *\fR\fBclnt_vc_create\fR(\fBconst int\fR \fIfildes\fR,
     \fBconst struct netbuf *\fR\fIsvcaddr\fR, \fBconst rpcprog_t\fR \fIprognum\
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR,
     \fBconst uint_t\fR \fIrecsz\fR);
.fi

.LP
.nf
\fBstruct rpc_createerr\fR \fBrpc_createerr\fR;
\fBstruct rpc_createerr\fR \fBrpc_createerr\fR
.fi

.LP
.nf
\fBvoid\fR \fBclnt_door_create\fR(\fBconst rpcprog_t\fR \fIprognum\fR,
     \fBconst rpcvers_t\fR \fIversnum\fR, \fBconst uint_t\fR \fIsendsz\fR);
.fi

.SH DESCRIPTION
.sp
.LP
\fBRPC\fR library routines allow  \fBC\fR language programs to make procedure
```

calls on other machines across the network. First a \fBCLIENT\fR handle is created and then the client calls a procedure to send a request to the server. On receipt of the request, the server calls a dispatch routine to perform the requested service, and then sends a reply.
.sp
.LP
These routines are MT-Safe. In the case of multithreaded applications, the \fB-mt\fR option must be specified on the command line at compilation time. **When the \fB-mt\fR option is specified, \fBrpc_createerr\fR becomes a macro that enables each thread to have its own \fBrpc_createerr\fR. See** *When the \fB-mt\fR option is specified, \fBrpc_createerr()\fR becomes a macro that enables each thread to have its own \fBrpc_createerr()\fR. See* \fBthreads\fR(5).
.SS "Routines"
.sp
.LP
See \fBrpc\fR(3NSL) for the definition of the \fBCLIENT\fR data structure.
.sp
.ne 2
.na
\fB\fBclnt_control()\fR\fR
.ad
.sp .6
.RS 4n
A function macro to change or retrieve various information about a client object. \fIreq\fR indicates the type of operation, and \fIinfo\fR is a pointer to the information. For both connectionless and connection-oriented transports, the supported values of \fIreq\fR and their argument types and what they do are:
.sp
.in +2
.nf
CLSET_TIMEOUT struct timeval * set total timeout
CLGET_TIMEOUT   struct timeval *   get total timeout
.fi
.in -2

If the timeout is set using \fBclnt_control()\fR, the timeout argument passed by \fBclnt_call()\fR is ignored in all subsequent calls. If the timeout value is set to \fB0\fR, \fBclnt_control()\fR immediately returns \fBRPC_TIMEDOUT\fR. Set the timeout parameter to \fB0\fR for batching calls.
.sp
.in +2
.nf
CLGET_SERVER_ADDR struct netbuf * get server's address
CLGET_SVC_ADDR struct netbuf * get server's address
CLGET_FD   int *   get associated file descriptor
CLSET_FD_CLOSE void   close the file descriptor when
                destroying the client handle
                (see \fBclnt_destroy()\fR)
CLSET_FD_NCLOSE void   do not close the file
                descriptor when destroying the client handle
CLGET_VERS   rpcvers_t get the RPC program's version
                number associated with the
                client handle
CLSET_VERS   rpcvers_t set the RPC program's version
                number associated with the
                client handle. This assumes
                that the RPC server for this
                new version is still listening
                at the address of the previous
                version.
CLGET_XID   uint32_t   get the XID of the previous
                remote procedure call
CLSET_XID   uint32_t   set the XID of the next
                remote procedure call

CLGET_PROG   rpcprog_t   get program number
CLSET_PROG   rpcprog_t   set program number
.fi
.in -2

The following operations are valid for connection-oriented transports only:
.sp
.in +2
.nf
CLSET_IO_MODE rpciomode_t*   set the IO mode used
                to send one-way requests. The argument for this operation
                can be either:
   - RPC_CL_BLOCKING   all sending operations block
                until the underlying transport protocol has
                accepted requests. If you specify this argument
                you cannot use flush and getting and setting buffer
                size is meaningless.
   - RPC_CL_NONBLOCKING   sending operations do not
                block and return as soon as requests enter the buffer.
                You can now use non-blocking I/O. The requests in the
                buffer are pending. The requests are sent to
                the server as soon as a two-way request is sent
                or a flush is done. You are responsible for flushing
                the buffer. When you choose RPC_CL_NONBLOCKING argument
                you have a choice of flush modes as specified by
                CLSET_FLUSH_MODE.
CLGET_IO_MODE rpciomode_t*   get the current IO mode
CLSET_FLUSH_MODE rpcflushmode_t*   set the flush mode.
                The flush mode can only be used in non-blocking I/O mode.
                The argument can be either of the following:
        - RPC_CL_BESTEFFORT_FLUSH: All flushes send requests
                in the buffer until the transport end-point blocks.
                If the transport connection is congested, the call
                returns directly.
        - RPC_CL_BLOCKING_FLUSH: Flush blocks until the
                underlying transport protocol accepts all pending
                requests into the queue.
CLGET_FLUSH_MODE rpcflushmode_t*   get the current flush mode.
CLFLUSH rpcflushmode_t   flush the pending requests.
                This command can only be used in non-blocking I/O mode.
                The flush policy depends on which of the following
                parameters is specified:
        - RPC_CL_DEFAULT_FLUSH, or NULL: The flush is done
           according to the current flush mode policy
                (see CLSET_FLUSH_MODE option).
        - RPC_CL_BESTEFFORT_FLUSH: The flush tries
                to send pending requests without blocking; the call
                returns directly. If the transport connection is
                congested, this call could return without the request
                being sent.
        - RPC_CL_BLOCKING_FLUSH: The flush sends all pending
                requests. This call will block until all the requests
                have been accepted by the transport layer.
CLSET_CONNMAXREC_SIZE int*   set the buffer size.
                It is not possible to dynamically
                resize the buffer if it contains data.
                The default size of the buffer is 16 kilobytes.
CLGET_CONNMAXREC_SIZE int*   get the current size of the
                buffer
CLGET_CURRENT_REC_SIZE int*   get the size of
                the pending requests stored in the buffer. Use of this
                command is only recommended when you are in non-blocking
                I/O mode. The current size of the buffer is always zero
                when the handle is in blocking mode as the buffer is not
                used in this mode.
.fi

```
.in -2

The following operations are valid for connectionless transports only:
.sp
.in +2
.nf
CLSET_RETRY_TIMEOUT  struct timeval *    set the retry timeout
CLGET_RETRY_TIMEOUT  struct timeval *    get the retry timeout
.fi
.in -2

The retry timeout is the time that \fBRPC\fR waits for the server to reply
before retransmitting the request.
.sp
\fBclnt_control()\fR returns \fBTRUE\fR on success and \fBFALSE\fR on failure.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine for program \fIprognum\fR and version
\fIversnum\fR. \fIhost\fR identifies the name of the remote host where the
server is located. \fInettype\fR indicates the class of transport protocol to
use. The transports are tried in left to right order in \fBNETPATH\fR variable
or in top to bottom order in the netconfig database.
.sp
\fBclnt_create()\fR tries all the transports of the \fInettype\fR class
available from the \fBNETPATH\fR environment variable and the netconfig
database, and chooses the first successful one. A default timeout is set and
can be modified using \fBclnt_control()\fR. This routine returns \fINULL\fR if
it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason
for failure.
.sp
Note that \fBclnt_create()\fR returns a valid client handle even if the
particular version number supplied to \fBclnt_create()\fR is not registered
with the \fBrpcbind\fR service. This mismatch will be discovered by a
\fBclnt_call\fR later (see \fBrpc_clnt_calls\fR(3NSL)).
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_timed()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine which is similar to \fBclnt_create()\fR but
which also has the additional parameter \fItimeout\fR that specifies the
maximum amount of time allowed for each transport class tried. In all other
respects, the \fBclnt_create_timed()\fR call behaves exactly like the
\fBclnt_create()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_vers()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine which is similar to \fBclnt_create()\fR but
which also checks for the version availability. \fIhost\fR identifies the name
```



```
of the remote host where the server is located. \fInettype\fR indicates the
class transport protocols to be used. If the routine is successful it returns a
client handle created for the highest version between \fIvers_low\fR and
\fIvers_high\fR that is supported by the server. \fIvers_outp\fR is set to this
value. That is, after a successful return \fIvers_low\fR <= \fI*vers_outp\fR <=
\fIvers_high\fR. If no version between \fIvers_low\fR and \fIvers_high\fR is
supported by the server then the routine fails and returns \fBNULL.\fR A
default timeout is set and can be modified using \fBclnt_control()\fR. This
routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine
can be used to print the reason for failure.
.sp
Note: \fBclnt_create()\fR returns a valid client handle even if the particular
version number supplied to \fBclnt_create()\fR is not registered with the
\fBrpcbind\fR service. This mismatch will be discovered by a \fBclnt_call\fR
later (see \fBrpc_clnt_calls\fR(3NSL)). However, \fBclnt_create_vers()\fR does
this for you and returns a valid handle only if a version within the range
supplied is supported by the server.
.RE

.sp
.ne 2
.na
\fB\fBclnt_create_vers_timed()\fR\fR
.ad
.sp .6
.RS 4n
Generic client creation routine similar to \fBclnt_create_vers()\fR but with
the additional parameter  \fItimeout\fR, which specifies the maximum amount of
time allowed for each transport class tried.   In all other respects, the
\fBclnt_create_vers_timed()\fR call behaves exactly like the
\fBclnt_create_vers()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_destroy()\fR\fR
.ad
.sp .6
.RS 4n
A function macro that destroys the client's \fBRPC\fR handle. Destruction
usually involves deallocation of private data structures, including \fIclnt\fR
itself. Use of \fIclnt\fR is undefined after calling \fBclnt_destroy()\fR. If
the \fBRPC\fR library opened the associated file descriptor, or
\fBCLSET_FD_CLOSE\fR was set using \fBclnt_control()\fR, the file descriptor
will be closed.
.sp
The caller should call \fBauth_destroy(\fR\fIclnt\fR->\fBcl_auth)\fR (before
calling \fBclnt_destroy()\fR) to destroy the associated \fBAUTH\fR structure
(see \fBrpc_clnt_auth\fR(3NSL)).
.RE

.sp
.ne 2
.na
\fB\fBclnt_dg_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client for the remote program \fIprognum\fR
and version \fIversnum\fR; the client uses a connectionless transport. The
remote program is located at address \fIsvcaddr\fR. The parameter \fIfildes\fR
is an open and bound file descriptor. This routine will resend the call message
in intervals of 15 seconds until a response is received or until the call times
out. The total time for the call to time out is specified by \fBclnt_call()\fR
(see \fBclnt_call()\fR in \fBrpc_clnt_calls\fR(3NSL)). The retry time out and
```

the total time out periods can be changed using \fBclnt_control()\fR. The user may set the size of the send and receive buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. This routine returns \fINULL\fR if it fails.
.RE

.sp
.ne 2
.na
\fB\fBclnt_pcreateerror()\fR\fR
.ad
.sp .6
.RS 4n
Print a message to standard error indicating why a client \fBRPC\fR handle could not be created. The message is prepended with the string \fIs\fR and a colon, and appended with a newline.
.RE

.sp
.ne 2
.na
\fB\fBclnt_raw_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client handle for the remote program \fIprognum\fR and version \fIversnum\fR. The transport used to pass messages to the service is a buffer within the process's address space, so the corresponding \fBRPC\fR server should live in the same address space; (see \fBsvc_raw_create()\fR in \fBrpc_svc_create\fR(3NSL)). This allows simulation of \fBRPC\fR and measurement of \fBRPC\fR overheads, such as round trip times, without any kernel or networking interference. This routine returns \fINULL\fR if it fails. \fBclnt_raw_create()\fR should be called after \fBsvc_raw_create()\fR.
.RE

.sp
.ne 2
.na
\fB\fBclnt_spcreateerror()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_pcreateerror()\fR, except that it returns a string instead of printing to the standard error. A newline is not appended to the message in this case.
.sp
Warning: returns a pointer to a buffer that is overwritten on each call. In multithread applications, this buffer is implemented as thread-specific data.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tli_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client handle for the remote program \fIprognum\fR and version \fIversnum\fR. The remote program is located at address \fIsvcaddr\fR. If \fIsvcaddr\fR is \fINULL\fR and it is connection-oriented, it is assumed that the file descriptor is connected. For connectionless transports, if \fIsvcaddr\fR is \fINULL\fR, \fBRPC_UNKNOWNADDR\fR error is set. \fIfildes\fR is a file descriptor which may be open, bound and connected. If it is \fBRPC_ANYFD\fR, it opens a file descriptor on the transport specified by \fInetconf\fR. If \fIfildes\fR is

\fBRPC_ANYFD\fR and \fInetconf\fR is \fINULL\fR, a \fBRPC_UNKNOWNPROTO\fR error is set. If \fIfildes\fR is unbound, then it will attempt to bind the descriptor. The user may specify the size of the buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. Depending upon the type of the transport (connection-oriented or connectionless), \fBclnt_tli_create()\fR calls appropriate client creation routines. This routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason for failure. The remote \fBrpcbind\fR service (see \fBrpcbind\fR(1M)) is not consulted for the address of the remote service.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tp_create()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_create()\fR except \fBclnt_tp_create()\fR tries only one transport specified through \fInetconf\fR.
.sp
\fBclnt_tp_create()\fR creates a client handle for the program \fIprognum\fR, the version \fIversnum\fR, and for the transport specified by \fInetconf\fR. Default options are set, which can be changed using \fBclnt_control()\fR calls. The remote \fBrpcbind\fR service on the host \fIhost\fR is consulted for the address of the remote service. This routine returns \fINULL\fR if it fails. The \fBclnt_pcreateerror()\fR routine can be used to print the reason for failure.
.RE

.sp
.ne 2
.na
\fB\fBclnt_tp_create_timed()\fR\fR
.ad
.sp .6
.RS 4n
Like \fBclnt_tp_create()\fR except \fBclnt_tp_create_timed()\fR has the extra parameter \fItimeout\fR which specifies the maximum time allowed for the creation attempt to succeed. In all other respects, the \fBclnt_tp_create_timed()\fR call behaves exactly like the \fBclnt_tp_create()\fR call.
.RE

.sp
.ne 2
.na
\fB\fBclnt_vc_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an \fBRPC\fR client for the remote program \fIprognum\fR and version \fIversnum\fR; the client uses a connection-oriented transport. The remote program is located at address \fIsvcaddr\fR. The parameter \fIfildes\fR is an open and bound file descriptor. The user may specify the size of the send and receive buffers with the parameters \fIsendsz\fR and \fIrecvsz\fR; values of \fB0\fR choose suitable defaults. This routine returns \fINULL\fR if it fails.
.sp
The address \fIsvcaddr\fR should not be \fINULL\fR and should point to the actual address of the remote program. \fBclnt_vc_create()\fR does not consult the remote \fBrpcbind\fR service for this information.
.RE

.sp
.ne 2

```
.na
\fB\fBrpc_createerr\fR\fR
\fB\fIrpc_createerr()\fR\fR
.ad
.sp .6
.RS 4n
A global variable whose value is set by any \fBRPC\fR client handle creation
routine that fails.  It is used by the routine \fBclnt_pcreateerror()\fR to
print the reason for the failure.
.sp
In multithreaded applications, \fBrpc_createerr\fR becomes a macro which
enables each thread to have its own \fBrpc_createerr\fR.
.RE

.sp
.ne 2
.na
\fB\fBclnt_door_create()\fR\fR
.ad
.sp .6
.RS 4n
This routine creates an RPC client handle over doors for the given program
\fIprognum\fR and version \fIversnum\fR. Doors is a transport mechanism that
facilitates fast data transfer between processes on the same machine. The user
may set the size of the send buffer with the parameter \fIsendsz\fR. If
\fIsendsz\fR is 0, the corresponding default buffer size is 16 Kbyte. The
\fBclnt_door_create()\fR routine returns \fINULL\fR if it fails and sets a
value for \fBrpc_createerr\fR.
.RE

.SH ATTRIBUTES
.sp
.LP
See \fBattributes\fR(5)  for descriptions of the following attributes:
.sp

.sp
.TS
box;
c | c
l | l .
ATTRIBUTE TYPE  ATTRIBUTE VALUE
_
Architecture    All
_
Interface Stability     Committed
_
MT-Level        MT-Safe
.TE

.SH SEE ALSO
.sp
.LP
\fBrpcbind\fR(1M), \fBrpc\fR(3NSL), \fBrpc_clnt_auth\fR(3NSL),
\fBrpc_clnt_calls\fR(3NSL), \fBrpc_svc_create\fR(3NSL),
\fBsvc_raw_create\fR(3NSL), \fBthreads\fR(5), \fBattributes\fR(5)
```