```
                 UNIX-Schnittstelle
                 ==================


      5. Systemrufe für globale Prozesskommunikation
      ===============================================


    - BSD spezifische Prozesskommunikation:
         Sockets
    - Remote Procedure Calls
```



```
         5.1 Kommunikation mit Sockets
         =============================

Sockets - Werkzeuge für die lokale und globale Kommunikation
          von Prozessen
          Vorgestellt: 1982  in BSD 4.1c  für  VAX

Für die Kommunikation werden Protokolle benutzt:

   UNIX-Protokoll  für die lokale Kommunikation
   Internet Protokolle TCP/IP und UDP/IP für globale/lokale
   Kommunikation

IP-Adressen:
     global:   Internetadresse   141.20.20.50
               Klassen: A, B, C, D
     lokal:    Portadresse+Protokoll (TCP - Transmission Control Protocol
                                      UDP - User Datagram Protocol
                                      IP  - Internet Protocoll)
Kommunikationspartner:

     Server
     Client

mit festen Aufgaben bei der Kommunikation
```

```
Verbindungsorientiertes Protokoll TCP/IP
----------------------------------------

          Server                 Client

          socket()               socket()
             |                      |
             V                      |
           bind()                   |
             |                      |
             V                      |
           listen()                 |
             |                      |
             V                      V
           accept() <----------- connect()
             |                      |
             V        Daten         V
            read() <------------- write()
           write() -------------> read()
             |        Daten         |
             V                      V
           close()                close()
```



```
Verbindungsloses Protokoll   UDP/IP
-----------------------------------

          Server                 Client

          socket()               socket()
             |                      |
             V                      V
           bind()                 bind()
             |                      |
             V       Daten          V
         recfrom() <------------ sendto()
          sendto() ------------> recfrom()
             |       Daten          |
             V                      V
           close()                close()
```

```
Socketadressen
--------------

allgemeine Adresstruktur:
  struct sockaddr {
         u_short   sa_family;  /* address family: AF_INET, AF_UNIX,
                                                  AF_NS, AF_IMPLINK */
         char      sa_data[14]; /*Adresse
  };

Hostadresse:
  struct in_addr {
        u_long s_addr;   /* 32-bit host-id, network byte ordered */
  };

Adresstruktur für Internetadressen:
   struct sockaddr_in {
      short   sin_family;       /* AF_INET */
      u_short sin_port;         /* 16-bit port number */
      struct  in_addr sin_addr; /* 32-bit host-id, network ordered */
      char    sin_zero[8];      /* unused */
   };

Headerfiles:
      #include <sys/types.h>
      #include <sys/socket.h>
```



```
int socket(int family, int type, int protocol);

Erzeugen eines Kommunikationsendpunktes (socket). Es wird die
Adressfamilie  family,  der Protokolltyp  type  und das Protokoll
protocol  für den Socket festgelegt.

   family:  AF_UNIX    - interne UNIX-Adressen (Filenamen)
            AF_INET    - Internetadressen
            AF_NS      - Xerox NS Adressen
            AF_IMPLINK - IMP Adressen (ARPANET)

   type:    SOCK_STREAM    - stream socket (verbindungsorientiert)
            SOCK_DGRAM     - datagram socket (verbindungslos)
            SOCK_RAW       - raw socket (Raw-Devices)

   protocol: IPPROTO_UDP   - UDP
             IPPROTO_TCP   - TCP
             IPPROTO_ICMP  - ICMP
             IPPROTO_RAW   - raw
                0          - das zu  type  passende Protokoll wird
                             ausgewählt

Gültige Kombinationen von type  und protocol:

                    AF_UNIX     AF_INET
       SOCK_STREAM    ja          TCP
       SOCK_DGRAM     ja          UDP
       SOCK_RAW                 IP/ICMP
```

socket() gibt einen Socket-ID (sockfd) zurück, der eine Datenstruktur repräsentiert. Damit diese Datenstuktur für eine Kommunikation genutzt werden kann, mussen folgende Informationen in ihr eingetragen sein:

```
   Protokoll:       von socket()
   lokale Adresse:  von bind()
   lokaler Port:    von bind()
   remote Adresse:  von accept(), connect()
   remote Port:     von accept(), connect()
```

Rückkehrwert:

```
  >=0 - Socket-ID
   <0 - Fehler
        EAFNOSUPPORT    - Adresse nicht von Kern unterstützt
        EPROTONOSUPPORT - Sockets unterstützt nicht die angegbene
                          Adressfamilie
        EMFILE          - kein Deskritor frei
        ENOBUFS         - keine Puffer frei
        ENOMEM          - kein Kernspeicher frei
        EPERM           - ein nicht SU versucht einen RAW-socket zu
                          eröffnen.
```



```
#include <sys/types.h>
#include <sys/socket.h>

int socketpair(int family, int type, int protocol, int sockvec[]);
```

Erzeugen zweier miteinander verbundener Kommunikationsendpunkte (Sockets) sockvec[0] und sockvec[1] ähnlich einer Pipe, allerdings sind diese Sockets bidirektional im Gegensatz zur Pipe, die uni- direktional ist. family, type und protocol haben die gleiche Bedeutung wie bei socket().

Rückkehrwert:

```
     = 0    -  OK
     < 0    -  Fehler
               wie bei socket()
               EFAULT   -  sockvec[] hat unzulässige Adresse
```

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd, struct sockaddr *myaddr, int addrlen);
```

bind() dient zum Binden der Adresse (Hostadresse, Port) in *myaddr an einen ungebundenen Socket sockfd. Server müssen bind() verwenden. Dies ist notwendig damit ein Client mit Hilfe dieser Adresse einen Server erreichen kann. Clienten müssen bind() bei der Benutzung verbindungsloser Protokolle verwenden. addrlen gibt die Länge von *myaddr an.

```
Rückkehrwert:
      =0  - Ok
      <0  - Fehler
            EBADF         - sockfd ist falsch
            ENOTSOCK      - sockfd ist ein File und kein Socket
            EADDRNOTAVAIL - falsche Hostadresse
            EADDRINUSE    - Port wird bereits benutzt
            EINVAL        - Socket sockfd ist bereits gebunden
            EACCES        - kein Zugriff auf diesen Port (<1024)
            EFAULT        - falsche Adresse von  myaddr
```



```
#include <sys/socket.h>

int listen(int sockfd, int backlog);
```

Mit listen() teilt ein verbindungsorientiert arbeitender Server dem Kern mit, wieviele ausstehende Verbindungsanforderungen (backlog) für den Socket sockfd in einer Queue gespeichert werden sollen. Maximum wird durch SOMAXCONN (5..8) festgelegt.

```
Rückkehrwert:

       =0  - Ok
       <0  - Fehler
             EBADF       - sockfd ist falsch
             ENOTSOCK    - sockfd ist ein File und kein Socket
             EOPNOTSUPP  - Sockettyp untestützt listen nicht.
```

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int sockfd, struct sockaddr *peer, int *addrlen);

accept()  nimmt eine neue Verbindungsanforderung für den gebundenen
Socket  sockfd  an und erzeugt für diese Verbindung einen neuen
Socket. sockfd  ist ein Socket, der zuvor mit  bind()  und
listen()  gebunden wurde.  peer  zeigt auf ein Feld mit der Länge
*addrlen  auf dem die Adresse des Clienten nach Ausführung des
Rufes  accept()  abgelegt wird.  *addrlen  enthält nach  accept()
die tatsächliche Länge der Adresse des Partners.
Rückkehrwert:
    >=0  -  Socket-ID der neuen Verbindung
     <0  -  Fehler
            EINVAL,EOPNOTSUPP - Für sockfd accept nicht zugelassen
            EBADF          - sockfd ist falsch
            ENOMEM         - kein Kernspeicher frei
            ENOTSOCK       - sockfd ist ein File und kein Socket
            EFAULT         - falsche Adresse von  peer
            EMFILE         - kein Deskritor frei
            EWOULDBLOCK    - es liegt keine Verbindungsanforderung vor
                              (Socket ist nichtblockierend)
```



```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *servaddr, int addrlen);

connect()  stellt eine Verbindung zwischen zwei Sockets her. Beide
Sockets müssen das gleiche Protokoll und die gleiche Adressfamilie
benutzen.  sockfd  spezifiziert den lokalen Socket und  *servaddr
spezifiziert die Adresse des remote Socket. addrlen  gibt die Länge
der Adresse an. Durch  connect()  wird beim Clienten ein freier Port
gebunden, wenn dies durch bind() noch nicht erfolgte.
SOCK_DGRAM-Socket:   connect() möglich, send() und recv() benutzbar
                     kein verbindungsorientiertes Protokoll!!!
SOCK_STREAM-Socket: verbindungsorientiertes Protokoll wird benutzt,
                     read(),  write()  möglich
Rückkehrwert(connect()):
      =0 - connect() erfolgreich
      <0 - Fehler
            EBADF          - sockfd ist falsch
            ENOTSOCK       - sockfd ist ein File und kein Socket
            EADDRNOTAVAIL - falsche Hostadresse
            EAFNOSUPPORT  - Adresse nicht von Kern unterstützt
            EISCONN        - Socket schon verbunden
            ETIMEOUT       - Timeout aufgetreten
            ECONNREFUSED  - connect() zurückgewiesen
            EADDRINUSE    - Adresse benutzt
            EFAULT         - falsche Adresse von servaddr
            EWOULDBLOCK    - es liegt kein accept() beim Server vor
                              (Socket ist nichtblockierend)
```

Beispiele:

```
     TCP-Sockets
      Talk  mit Warten auf den Partner
         inet.h
         s_sock.c
         c_sock.c    mit IP-Adresse
         c_sock0.c   mit Hostnamen
```



```
#include <sys/types.h>
#include <sys/socket.h>

int send(int sockfd, char *buff, int nbytes, int flags);
int recv(int sockfd, char *buff, int nbytes, int flags);
int sendto(int sockfd, char *buff, int nbytes, int flags,
           struct sockaddr *to, int addrlen);
int recfrom(int sockfd, char *buff, int nbytes, int flags,
            struct sockaddr *from, int *addrlen);
int sendmsg(int sockfd, struct msghdr *buff, int flags)
int recvmsg(int sockfd, struct msghdr *buff, int flags)
```

send(), sendto() und sendmsg() werden zum Senden von Daten mittels verbindungsloser oder verbindungsorientierter Protokolle benutzt.

recv(), recfrom() und recvmsg() werden zum Empfangen von Daten mittels verbindungsloser oder verbindungsorientierter Protokolle benutzt.

Wird send() bzw. recv() für verbindungslose Protokolle benutzt, so ist vorher ein connect() notwendig.

```
sockfd   - für die Kommunikation benutzter Socket
to, from - dienen der Adresspezifikation für verbindungslose
           Protokolle
addrlen  - Länge der Adresse
```

```
flags    -  spezielle zusätzliche Möglichkeiten beim Senden und Empfangen:
              MSG_OOB  -  senden und empfangen von out-of-band Daten
              MSG_PEEK -  besichtigen der Daten ohne entfernen aus
                          dem Datenstrom (recv, recvfrom)
              MSG_DONTROUTE - Routingtabelle nicht benutzen (nur sendmsg)
nbytes   -  Länge des Datenpuffers
buff     -  Adresse des Datenpuffers

Bei recvmsg() und sendmsg() hat der Datenpuffer folgende Struktur:
     struct msghdr {
        caddr_t msg_name;              /* optional address */
        int     msg_namelen;           /* size of address */
        struct  iovec *msg_iov;        /* address io-vector */
        u_int   msg_iovlen;            /* # elements in msg_iov */
        caddr_t msg_control;           /* address of control-data-buffer */
        int     msg_controllen;        /* length of control-data */
        int     msg_flags;             /* flags for send/recv */
     };

Rückkehrwert:
      >=0  - Anzahl der übertragenen Bytes
       <0  - Fehler
             EBADF      -  sockfd  ist falsch
             ENOTSOCK   -  sockfd verweist auf ein File
            EWOULDBLOCK - es liegt kein accept() beim Server vor
                          (Socket ist nichtblockierend)
             EINTR      - Signal aufgetreten vor Ende des Calls
             EFAULT     - buff-Parameter falsch
             EMSGSIZE   - zu grosse Länge (nbytes)
```



```
#include <sys/types.h>
#include <sys/socket.h>

getsockopt(int sockfd, int level, int option_name,
           char *option_value, int *option_len)
setsockopt(int sockfd, int level, int option_name,
           char *option_value, int option_len)
```

Lesen (getsockopt()) bzw. setzen (setsockopt()) von Optionen für den Socket sockfd. level gibt das Protokoll an. option_name spezifiziert die gewünscht Option. Folgende Optionen sind möglich:

```
    SO_DEBUG        - Debug-Flag (int)
    SO_ACCEPTION    - Listen enable (int)
    SO_BROADCAST    - Broadcast supported (int)
    SO_KEEPALIVE    - Connection aktiv (int) (SIGPIPE)
    SO_DONTROUTE    - Benutzung der Standardrouten  (int)
    SO_USELOOPBACK  - Sender erhält Kopie der gesendeten Daten (int)
    SO_LINGER       - warten auf Übertragungsende bei close() (int)
    SO_OBINLINE     - out-of-band Data (int)
    SO_SNDBUF       - Puffergrösse send (int)
    SO_RCVBUF       - Puffergrösse recv (int)
    SO_SNDTIMEO     - Sende-Time-Out (struct timeval)
    SO_RCVTIMEO     - Empfangs-Time-Out (struct timeval)

  nur für getsockopt()
    SO_ERROR        - Errorstatus (int)
    SO_TYPE         - Sockettype
```

option_value spezifiziert einen Puffer für den Wert der entsprechenden Option und option_len die Länge des Puffers.

```
Rückkehrwert:

   >=0  - Ok
    <0  - Fehler

             EBADF       -  sockfd  ist falsch
             ENOTSOCK    -  sockfd verweist auf ein File
             EFAULT      -  option_value- oder option_len-Parameter
                            falsch
             ENOPROTOPT -  Option unbekannt
```



```
#include <sys/socket.h>

int getsockname(int sockfd, struct sockaddr *address, int *adr_len)
int getpeername(int sockfd, struct sockaddr *address, int *adr_len)

Holen der zu einem Socket gehörenden Adressen:
     getsockname()  -   lokale Adresse
     getpeername()  -   remote Adresse
sockfd  -  Socket für den die Adresse bestimmt werden soll
*address - Puffer für die  lokale/remote Adresse
*addrlen - Länge des Adressfeldes/der Adresse

Rückkehrwert:
     >=0 - Ok
      <0 - Fehler
           EBADF     - sockfd nicht zulässig
           ENOTSOCK  - kein Socket, sondern File
           ENOBUFS   - keine Puffer vorhanden
           EFAULT    - address oder addrlen  falsch
```

```
Bibliotheks-Hilfsroutinen für Netzwerkdienste
---------------------------------------------

Byteorder-Routinen:
-------------------
     #include <sys/types.h>
     #include <netinet/in.h>
     #include <inttypes.h>

     uint32_t htonl(unint32_t hostlong);

              Konvertieren einer long-Integer von der Host-Darstellung
              in die Netzdarstellung.

     uint16_t htons(uint16_t hostshort);

              Konvertieren einunsigned-short-Integer von der Host-
              Darstellung in die Netzdarstellung.

     uint32_t ntohl(uint32_t netlong);

              Konvertieren einer long-Integer von der Netz-Darstellung
              in die Host-Darstellung

     uint16_t ntohs(uint16_t netshort);

              Konvertieren einer unsigned-short-Integer von der Netz-
              Darstellung in die Host-Darstellung
```



```
Routinen für die Adressumrechnung
---------------------------------

     #include <netdb.h>

     struct hostent *gethostbyname(const char *name);

     struct hostent  *gethostbyname_r(const  char  *name,  struct
     hostent *result, char *buffer, intbuflen, int *h_errnop);

              Bestimmen der Host-Informationen mittels Hostnamen.
              Adressen in Netz-Darstellung.

     struct hostent *gethostbyaddr(const char *addr, int len, int
     type);

     struct  hostent  *gethostbyaddr_r(const  char   *addr,   int
     length,  int type, struct hostent *result, char *buffer, int
     buflen, int *h_errnop);

              Bestimmen der Host-Informationen mittels Host-Adresse.
              Adresse in Netz-Darstellung.

     struct hostent {
         char    *h_name;         /* canonical name of host */
         char    **h_aliases;     /* alias list */
         int     h_addrtype;      /* host address type */
         int     h_length;        /* length of address */
         char    **h_addr_list;   /* list of addresses */
     };
```

Routinen für Adressmanipulationen

```
-------------------------------------

     #include <sys/types.h>
     #include <sys/socket.h>
     #include <netinet/in.h>
     #include <arpa/inet.h>

     unsigned long inet_addr(const char *cp);
                a.b.c.d  ---> Hostadresse long int

     unsigned long inet_network(const char *cp);
                a.b.c    ---> Netzwerkadresse long int

     struct in_addr inet_makeaddr(const int net, const int lna);
                Netzwerkadresse+lokale Hostadresse in in_addr-Struktur

     int inet_lnaof(const struct in_addr in);
                in_addr-Struktur --> lokale Hostadresse

     int inet_netof(const struct in_addr in);
                in_addr-Struktur --> Netzwerkadresse

     char *inet_ntoa(const struct in_addr in);
                in_addr-Struktur --> a.b.c.d
```


Beispiele:

UDP-Sockets - Talk mit Warten auf den Partner

```
          su_sock.c
          cu_sock.c
```

Talk ohne Warten auf den Partner (Prozesse)

```
         s_sock1.c
         c_sock1.c
         s_sock1r.c   mit Client-Adresse bestimmen
```

Talk ohne Warten auf den Partner (Threads)

```
         c_sock_t.c
         s_sock1_t.c
         s_sock2_t.c
```

Hilfsprogramme

```
        addr1 - IP-Adresse -> Hostname
                addr1 141.20.20.50
        addr2 - Hostname -> IP-Adresse
                addr2 star
        msb   - Most significant byte
                msb 141.20.21.50
```

```
Messungen, Zuverlässigkeit

TCP:
      tcptd.c - Server Paketecho
                auf star, nbellus, garak: tcptd
      tcpt.c  - Test Paketecho
                tcpt hostname Paketlaenge Paketanzahl
                auf Notebook, amsel:  time tcpt star 512 100

UDP:
      udptd.c - Server Paketecho
                auf star, nbellus, garak:  udptd
      udpt.c  - Test Paketecho
                udpt hostname Paketlaenge Paketanzahl
                auf Notebook, amsel:  time udpt star 512 100
```



```
           5.2 Remote Procedure Call
           =========================

Modell für Remote Procedure Call  (RPC)

     +--------+                 +--------+
     |Client- |                 |Server- |
     |routine |                 |routine |
     +--------+                 +--------+
    1  |   ^  10              6  |   ^  5
       V   |                     v   |
     +--------+                 +--------+
     |Client- |                 |Server- |
     | stub   |                 | stub   |
     +--------+                 +--------+
    2  |   ^   9              7  |   ^  4
       V   |                     v   |
     +--------+       8         +--------+
     | Netz-  |<----------------| Netz-  |
     |routine |---------------->|routine |
     +--------+       3         +--------+

 1 - lokaler Prozeduraufruf          6 - Übergabe des Prozedurergebnis
 2 - Netzaufruf                      7 - Netzaufruf
 3 - Nachrichtenübertragung          8 - Nachrichtenübertragung
 4 - Server wartet                   9 - Clientstub wartet auf Ergebnis
 5 - Aufruf der Serverprozedur      10 - Rückkehr zur Clientfunktion
```

1 - Der Client ruft eine lokale Prozedur im Client-Stub auf (die gewünschte Prozedure). Der Client-Stub verpackt die Parameter.

2 - Der Client-Stub übergibt die Netznachricht an den Kernel (socket) zum Transport.

3 - Der Kernel überträgt die Netznachricht an das entfernte System.

4 - Der Server-Stub wartet auf das Eintreffen von Netznachrichten. Diese werden ausgepackt.

5 - Der Server-Stub ruft die eigentliche Prozedure auf.

6 - Die Prozedur wird ausgeführt und gibt die Ergebnisse an den Server-Stub.

7 - Der Server-Stub packt die Ergebnisse ein und übergibt die Netznachricht an den Kernel.

8 - Der Kernel überträgt die Netznachricht an das rufende System.

9 - Der Client-Stub wartet auf das Ergebnis und packt es aus.

10- Der Client-Stub übergibt das Ergebnis an den Clienten.

SUN-RPC

SUN-RPC ermöglicht die Benutzung von mehreren Remote-Prozeduren pro Programm. Die Prozeduren müssen vorher definiert werden. Das Programm muß sich beim Portmapper registrieren.

Parameterübergabe

Die Parameterübergabe ist nur für value-Parameter erlaubt. Hierbei transformiert der Client-Stub die Parameter in Netzformat. call-by-reference Parameter sind nicht zulässig, da der Server-Stub keine Informationen darüber hat, was sich hinter der Adresse verbirgt. Es ist jeweils nur ein Parameter und nur ein Ergebnis erlaubt. Sollen komplexere Werte ausgetauscht werden, müssen Strukturen benutzt werden.

Bindung

Mittels des PortMapper-Daemons (rpcbind, portmap) kann ein Client Verbindung zu einem Pogramm auf einem entfernten System knüpfen. Dazu ist es notwendig, daß auf dem Server der Server-Stub sich beim Portmapper registriert hat. Der Client erhält vom Portmapper dann den aktuellen Port für das gewünschte Programm. Die Identifzierung erfolgt über Programmnummer und Versionsnummer.

```
  entferntes System (Server)

  ..........................................................
  . +-------------+        1            +-----------+      .
  . | Portmapper  |<--------------------| Server-   |      .
  . |   Daemon    |                     | programm  |      .
  . +-------------+                     +-----------+      .
  .......^..................................^..^............
         |                                  |  |
         |2                                3|  |4
         |       ...........................|  |
         |       .       +-----------+     .|  |
         +-------------->| Client-   |<-----+  |
                 .       | program   |<--------+
                 .       +-----------+     .
                 ...........................

  lokales System
```

1 - Beim Start des Servers. Der Server erzeugt einen Socket.
    mittels der Funktion svc_register registriert der Server das
    Programm (Nummer) und die Version unter dem Port beim Portmapper
2 - Der Client fragt den Portmapper mittels Programmnummer und
    Programmversion nach dem Port für das Programm.
3 - Der Client sendet den Prozedurcall 1
4 - Der Client sendet den Prozedurcall 2

Transportprotokoll

UDP (maximale Paketlänge 8192 Bytes), TCP (keine Grenze)

Ausnahmebehandlung

Lokale Ausnahme in einer Prozedur sind immer klar erkennbar (Division durch Null, Speicherschutzverletzungen, ...). Bei RPC kommen zusätzlich Fehler hinzu: Netzwerkfehler, Lastprobleme, Absturz des Clienten, Absturz des Servers.
UDP: wenn keine Antwort kommt wird der Call wiederholt. Feste Anzahl von Wiederholungen, dann Rückgabe eines Fehlerkodes.
TCP: Absicherung durch TCP. Rückgabe eines Fehlerkodes.

Semantik der Aufrufe

Bei einer lokalen Prozedur ist klar, wie oft sie abgearbeitet wurde, nachdem sie aufgerufen wurde - genau ein Mal.
Bei einer Remote-Prozedure ist das nicht eindeutig. Folgend Situationen sind möglich:

```
   nicht aufgerufen - der Call ist beim Server nicht angekommen
   genau einmal     - alles ok
   mehrmals         - bei der Datenübertragung ist etwas schief
                      gegangen.
```

Bei SUN-RPC wird jedem Call eine eindeutige zufällige Durchführungs-ID zugeordnet. Diese wird bei Rückkehr auf Gleichheit geprüft. Dadurch ist gesichtert, daß das Ergebnis zu dem Call gehört. Duplikate werden vom Server- bzw. Client-Stub aussortiert.

Datenrepräsentation

Besonders problematisch ist die Datenübertragung zwischen verschieden Architekturen. Deshalb muß bei der Datenübertragung klar sein, was für Daten übertragen werden und wie sie dargestellt worden sind. Dafür gibt es XDR (eXternal Data Representation). Die entsprechenden Konvertierungsroutinen werden durch das Programm rpcgen automatisch erzeugt.

Sicherheit

SUN-RPC unterstützt folgende Authentifzierungsmöglichkeiten:

```
  Null-Authentifikation - keine Authentifizierung

  UNIX-Authentifikation - bei jedem RPC-Call werden folgende Information
                          mitgeliefert: Zeitstempel, Host, UID, GID

  DES-Authentifikation  - DES-Verfahren.
```


Wichtige Library-Calls für SUN-RPC

```
CLIENT *clnt_create(char *host, unsigned long prog,
                           unsigned long vers, char *proto);
```

Erzeugen einer Verbindung zu einem Programm auf einem Server.

```
  host  -  Servername
  prog  -  Programmnummer
  vers  -  Programmversion
  proto -  Protokoll ("tcp","udp")
```

--

```
clnt_destroy(CLIENT *clnt);
```

Löschen einer Verbindung

```
void clnt_pcreateerror(char *s);

Verbindungsfehler ausgeben, CLIENT-Handle nicht erzeugt


------------------------------------------------------

void clnt_perrno(enum clnt_stat stat);

Standardfehlertext ausgeben (callrpc())


------------------------------------------------------

clnt_perror(CLIENT *clnt, char *s);

Fehler nach Prozeduraufruf(clnt_call())
```



```
Erzeugen eines RPC-Programms (Schema für Beispiel)

  Server-Prozedur                            Server-Programm
  +------------+                                +---------+
  |dated.c     |---------------+---------->gcc---->|dated    |
  +------------+               ^            ^   +---------+
                               |            |
                               |          +---+
                               |          | R |
                        +------------+    | P |
  RPC-Spezifikation     |date_svc.c  |    | C |
  +-------+             +------------+    | - |
  |date.x|--->rpcgen--> |date.h      |    | L |
  +-------+             +------------+    | I |
                        |date_clnt.c|     | B |
                        +------------+    +---+
                               |            |
                               |            |
  +------------+               V            V   +---------+
  | rdate.c    |---------------+---------->gcc---->| rdate   |
  +------------+                                +---------+
  Client-main-Funktion                       Client-Programm
```

Spezifikation für rpcgen, daraus wird date_svc.c date.h und date_clnt.c

date.x

```
/* date.x - Specification von "remote date" and "time service"  */

/*
 * Definition von zwei Prozeduren:
 *      bin_date_1() Rueckgabe time and date als long integer.
 *      str_date_1() Rueckgabe von time und date in lesbarer Form
 */

program DATE_PROG {
    version DATE_VERS {
        long    bin_date(void) = 1;      /* Prozedurenummer  = 1, bin_date_1 *
        string  STR_DATE(long) = 2;      /* Prozedurenummer = 2,  str_date_1 *
    } = 1;                               /* Versionsnumber = 1 */
} = 1234567;                             /* Programmnummer = 1234567 */
```


Server-Programm, nur die Prozeduren sind zu definieren, alles
Andere kommt von rpcgen

dated.c

```
/* dated.c - Remote Prozedure; aufgerufen durch Server-Stub */
#include  <time.h>
#include  "date.h"  /* von rpcgen, enthaelt #include <rpc/rpc.h> */

/* Rueckgabe der Zeit in Sekunden */
long * bin_date_1()
{
  static long  timeval;  /* muss static sein */

  timeval = time((long *) 0);
  return(&timeval);
}

/* Datum und Uhrzeit menschlich lesbar */
char ** str_date_1(long *bintime)
{
  static char  *ptr;    /* muss static sein*/

  ptr = ctime(bintime);    /* uebersetzen in local time */
  return(&ptr);            /* Rueckgabe der Adresse des Zeigers */
}
```

```
Client-Programm:  rdate.c

/* rdate.c - Client Program fuer "remote date service" */

#include  <stdio.h>
#include  "date.h"  /* erzeugt durch rpcgen */

int main(int argc, char *argv[])
{
  CLIENT    *cl;      /* RPC handle */
  char    *server;
  long    *lresult;   /* Rueckkehrwert von  bin_date_1() */
  char    **sresult;  /* Rueckkehrwert von  str_date_1() */

  if (argc != 2) {
    fprintf(stderr, "usage: %s hostname\n", argv[0]);
    exit(1);
  }
  server = argv[1];
  /*  Erzeuge Client "handle" */
  if ( (cl = clnt_create(server, DATE_PROG, DATE_VERS, "udp")) == NULL) {
    /*  Verbindung zum Server konnte nicht hergestellt werden */
    clnt_pcreateerror(server);
    exit(2);
  }
```



```
  /* Aufruf von "BIN_DATE" */
  if ( (lresult = bin_date_1(NULL, cl)) == NULL) {
    clnt_perror(cl, server);
    exit(3);
  }
  printf("time on host %s = %ld\n", server, *lresult);
  /*  Aufruf von  "STR_DATE"  */
  if ( (sresult = str_date_1(lresult, cl)) == NULL) {
    clnt_perror(cl, server);
    exit(4);
  }
  printf("time on host %s = %s", server, *sresult);
  clnt_destroy(cl);    /* close Client handle */
  exit(0);
}
```

Aktionen zur Bildung:

```
rpcgen -k date.x
gcc -c -o date_proc.o date_proc.c
gcc -c -o date_svc.o date_svc.c
gcc -o date_svc date_proc.o date_svc.o
gcc -c -o rdate.o rdate.c
gcc -c -o date_clnt.o date_clnt.c
gcc -o rdate rdate.o date_clnt.o
```