English Edition

6. Inter Process Communication (3)

6. IPC
6.4 Remote Procedure Calls

/home/esser/Daten/Dozent/Folien/bs-esser-19-english.odp

6.4 IPC: Remote Procedure Calls (RPC)

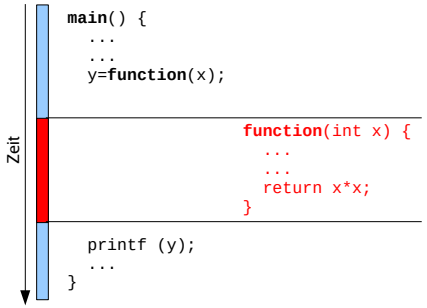
Remote Procedure Calls (1)

- idea: function calls that work across processes or even over the network
- programmer doesn't have to worry about the necessary communikation
  - message passing, I/O etc. are hidden from the programmer's view
- function arguments and results are transmitted between the participating processes via an RPC protocol
- client / server principle

Remote Procedure Calls (2)

Normal function call:

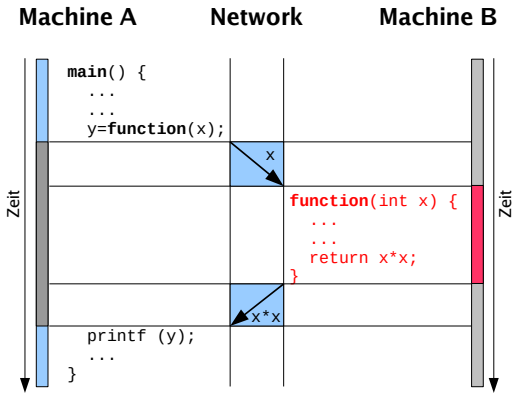
- program executes *main*() function.
- when calling a function, it pushes parameters onto the stack, and the CPU jumps to the function's start address via **call**.
- while the function computes, the *main*() function is suspended (synchronous request).
- after finishing the function, the CPU jump back into the *main*() function which uses the result.



Remote Procedure Calls (3)

Remote function call (conceptual):

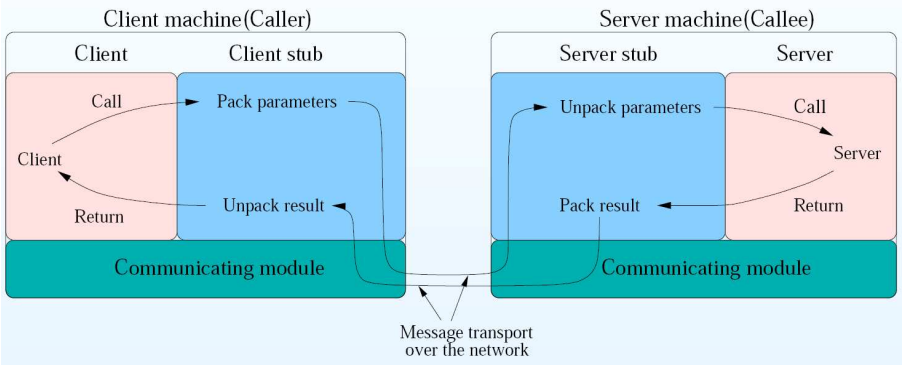
- program executes *main*() function.
- before starting the function, the arguments are sent (via the network) to the process which will execute the function.
- there the result is calculated and sent back.
- while the function executes and during the two send operations the calling process is idle (synchronous request).



Remote Procedure Calls (4)

- Prozess which calls the function contains a **client stub** for this function. Its purpose:
  - forward the function call (via the network) to a different process
  - receive reply and return result (to caller)
- process which executes the function contains a **server stub** for this function. Purpose:
  - receive arguments from a client stub
  - execute function
  - send result back

Remote Procedure Calls (5)



source: http://www-wjp.cs.uni-sb.de/lehre/seminar/ss04/reports/A_Shadrin-RPC-folien.pdf

Remote Procedure Calls (6)

comparison with regular functions:

- no **call by reference** possible
  – calling code and called function have no common address space
- however, client stub can accept calls by reference and then prepare the data for transport
- no **side effects** possible (e.g.: function modifies a global variable)

Remote Procedure Calls (7)

Marshalling / Serialization (1)

- problem: client and server machines use different internal representations for datatypes
  - classical (and most simple) example: Byte order – big-endian / little-endian 0xCC99 = 0xCC 0x99 or 0x99 0xCC
  - how to pack strings, floats, complexer structures (abstract datatypes), objects etc.?

Remote Procedure Calls (8)

Marshalling / Serialization (2)

- system supplies **pack()** and **unpack()**
- it is guaranteed that the execution path
  - client: s = **pack** (object);
send (srv, s);
  - server: recv (cl, s);
object = **unpack** (s);

leads to a situation where „object“ will represent the same object on both machines

Remote Procedure Calls (9)

Marshalling / Serialization (3)

- example for marshalling: Python

```
import pickle
data1 = {'a': [1, 2.0, 3, 4+6j],
         'b': ('string', u'Unicode string'),
         'c': None}
selfref_list = [1, 2, 3]
selfref_list.append(selfref_list)

output = open('data.pkl', 'wb')

# Pickle dictionary using protocol 0.
pickle.dump(data1, output)

# Pickle the list using the highest protocol
# available.
pickle.dump(selfref_list, output, -1)

output.close()
```

```
import pickle

pkl_file = open('data.pkl', 'rb')

data1 = pickle.load(pkl_file)
print data1

data2 = pickle.load(pkl_file)
print data2

pkl_file.close()
```


Remote Procedure Calls (10)

Marshalling / Serialization (4)

```
> hexdump -C data.pkl
00000000  28 64 70 30 0a 53 27 61  27 0a 70 31 0a 28 6c 70  |(dp0.S'a'.p1.(lp|
00000010  32 0a 49 31 0a 61 46 32  2e 30 0a 61 49 33 0a 61  |2.I1.aF2.0.aI3.a|
00000020  63 5f 5f 62 75 69 6c 74  69 6e 5f 5f 0a 63 6f 6d  |c__builtin__.com|
00000030  70 6c 65 78 0a 70 33 0a  28 46 34 2e 30 0a 46 36  |plex.p3.(F4.0.F6|
00000040  2e 30 0a 74 70 34 0a 52  70 35 0a 61 73 53 27 63  |.0.tp4.Rp5.asS'c|
00000050  27 0a 70 36 0a 4e 73 53  27 62 27 0a 70 37 0a 28  |'.p6.NsS'b'.p7.(|
00000060  53 27 73 74 72 69 6e 67  27 0a 70 38 0a 56 55 6e  |S'string'.p8.VUn|
00000070  69 63 6f 64 65 20 73 74  72 69 6e 67 0a 70 39 0a  |icode string.p9.|
00000080  74 70 31 30 0a 73 2e 80  02 5d 71 00 28 4b 01 4b  |tp10.s...]q.(K.K|
00000090  02 4b 03 68 00 65 2e                              |.K.h.e.|
00000097
```

RPC Example (1)

- a server program shall calculate averages (mean values)
- a client calls the server function transparently via RPC
- tasks:
  - create interface definition and translate it with **rpcgen** (this will create client and server stubs, among other files)
  - program client and server

RPC Example (2)

rpcgen creates all necessary files:

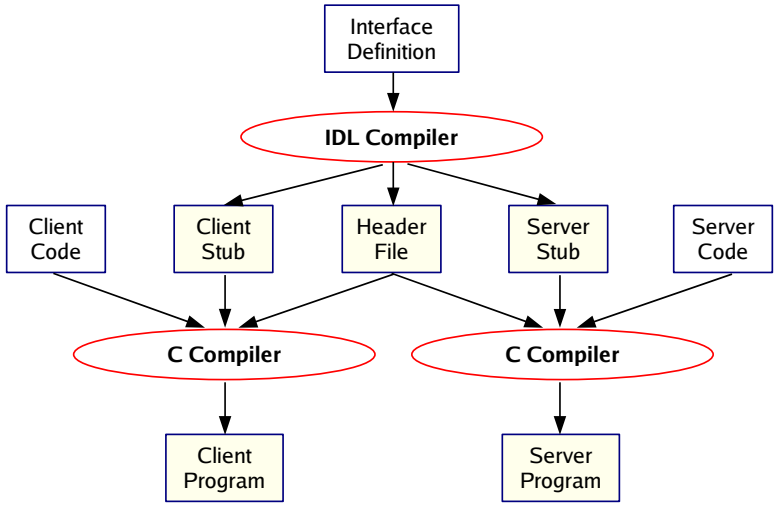
```
$ rpcgen avg.x
```

- *avg_clnt.c* contains client code (the client stub)
- *avg_svc.c* contains server code (the server stub)
- *avg_xdr.c* contains type definitions
- *avg.h* is the header file

```
/* avg.x
 *
 * The average procedure receives an array
 * of real numbers and returns the average
 * of their values. This toy service handles
 * a maximum of 200 numbers.
 */
const MAXAVGSIZE  = 200;

struct input_data {
  double input_data<200>;
};

typedef struct input_data input_data;

program AVERAGEPROG {
    version AVERAGEVERS {
        double AVERAGE(input_data) = 1;
    } = 1;
} = 22855;
```

source for this example (code):
http://www.linuxjournal.com/article/2204

RPC Example (3)



RPC Example (4)

rpcgen-generated header file (*avg.h*):

```
/* Please do not edit this file. It was generated using rpcgen. */
#include <rpc/rpc.h>
#define MAXAVGSIZE 200

struct input_data {
        struct {
                u_int input_data_len;
                double *input_data_val;
        } input_data;
};
typedef struct input_data input_data;

#define AVERAGEPROG 22855
#define AVERAGEVERS 1

extern double * average_1(input_data *, CLIENT *);
extern double * average_1_svc(input_data *, struct svc_req *);
extern int averageprog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

/* the xdr functions */

extern bool_t xdr_input_data (XDR *, input_data*);
extern bool_t xdr_input_data (XDR *, input_data*);
```

RPC Example (5)

rpcgen-generated client stub (*avg_clnt.c*):

```
/* Please do not edit this file. It was generated using rpcgen. */

#include <memory.h> /* for memset */
#include "avg.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

double * average_1(input_data *argp, CLIENT *clnt) {
        static double clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, AVERAGE,
                (xdrproc_t) xdr_input_data, (caddr_t) argp,
                (xdrproc_t) xdr_double, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```


RPC Example (6)

rpcgen-generated server stub (*avg_svc.c*)

```
/*
 * Please do not edit
 * this file. It was gene-
 * rated using rpcgen.
 */

#include "avg.h"
#include <stdio.h>
#include <stdlib.h>
#include <rpc/pmap_clnt.h>
#include <string.h>
#include <memory.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifndef SIG_PF
#define SIG_PF void(*)(int)
#endif
```

```
static void averageprog_1(struct svc_req *rqstp, register SVCXPRT *transp) {
   union {
         input_data average_1_arg;
     } argument;
   char *result;
   xdrproc_t _xdr_argument, _xdr_result;
   char *(*local)(char *, struct svc_req *);

   switch (rqstp->rq_proc) {
      case NULLPROC:
         (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
         return;
      case AVERAGE:
         _xdr_argument = (xdrproc_t) xdr_input_data;
         _xdr_result = (xdrproc_t) xdr_double;
         local = (char *(*)(char *, struct svc_req *)) average_1_svc;
         break;
      default:
         svcerr_noproc (transp);
         return;
   }
   memset ((char *)&argument, 0, sizeof (argument));
   if (!svc_getargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
      svcerr_decode (transp);
      return;
   }
   result = (*local)((char *)&argument, rqstp);
   if (result != NULL && !svc_sendreply(transp, (xdrproc_t) _xdr_result, result)) {
      svcerr_systemerr (transp);
   }
   if (!svc_freeargs (transp, (xdrproc_t) _xdr_argument, (caddr_t) &argument)) {
      fprintf (stderr, "%s", "unable to free arguments"); exit (1);
   }
   return;
}
```


RPC Example (7)

rpcgen-generated server stub (*avg_svc.c*) (continued)

```
int main (int argc, char **argv) {
   register SVCXPRT *transp;

   pmap_unset (AVERAGEPROG, AVERAGEVERS);

   transp = svcudp_create(RPC_ANYSOCK);
   if (transp == NULL) {
      fprintf (stderr, "%s", "cannot create udp service.");
      exit(1);
   }
   if (!svc_register(transp, AVERAGEPROG, AVERAGEVERS, averageprog_1, IPPROTO_UDP)) {
      fprintf (stderr, "%s", "unable to register (AVERAGEPROG, AVERAGEVERS, udp).");
      exit(1);
   }

   transp = svctcp_create(RPC_ANYSOCK, 0, 0);
   if (transp == NULL) {
      fprintf (stderr, "%s", "cannot create tcp service.");
      exit(1);
   }
   if (!svc_register(transp, AVERAGEPROG, AVERAGEVERS, averageprog_1, IPPROTO_TCP)) {
      fprintf (stderr, "%s", "unable to register (AVERAGEPROG, AVERAGEVERS, tcp).");
      exit(1);
   }

   svc_run ();
   fprintf (stderr, "%s", "svc_run returned");
   exit (1);
   /* NOTREACHED */
}
```


RPC Example (8): Client Code

```
#include "avg.h"
#include <stdlib.h>

void averageprog_1( char* host, int argc, char *argv[]) {
   CLIENT *clnt;
   double *result_1, *dp, f;
   char *endptr;
   int i;
   input_data average_1_arg;
   average_1_arg.input_data.input_data_val =
     (double*) malloc(MAXAVGSIZE*sizeof(double));
   dp = average_1_arg.input_data.input_data_val;
   average_1_arg.input_data.input_data_len = argc - 2;
   for (i=1;i<=(argc - 2);i++) {
      f = strtod(argv[i+1],&endptr);
      printf("value   = %e\n",f);
      *dp = f;
      dp++;
   }
   clnt = clnt_create(host, AVERAGEPROG,
                      AVERAGEVERS, "udp");
   if (clnt == NULL) {
      clnt_pcreateerror(host);
      exit(1);
   }
   result_1 = average_1(&average_1_arg, clnt);
   if (result_1 == NULL) {
      clnt_perror(clnt, "call failed:");
   }
   clnt_destroy( clnt );
   printf("average = %e\n",*result_1);
}
```

```
main( int argc, char* argv[] ) {
   char *host;

   if(argc < 3) {
     printf(
      "usage: %s server_host value ...\n",
      argv[0]);
      exit(1);
   }
   if(argc > MAXAVGSIZE + 2) {
      printf("Two many input values\n");
      exit(2);
   }
   host = argv[1];
   averageprog_1( host, argc, argv);
}
```

RPC Example (9): Server Code

```
#include <rpc/rpc.h>
#include "avg.h"
#include <stdio.h>

static double sum_avg;

double * average_1(input_data *input,
        CLIENT *client) {

   double *dp = input->input_data.input_data_val;
   u_int i;
   sum_avg = 0;
   for(i=1;i<=input->input_data.input_data_len;i++) {
      sum_avg = sum_avg + *dp; dp++;
   }
   sum_avg = sum_avg / input->input_data.input_data_len;
   return(&sum_avg);
}

double * average_1_svc(input_data *input,
        struct svc_req *svc) {
  CLIENT *client;
  return(average_1(input,client));
}
```


RPC Example (10)

- compilation:

```
$ rpcgen avg.x
$ gcc avgclient.c avg_clnt.c \
  avg_xdr.c -o avgclient -lnsl
$ gcc avgserver.c avg_svc.c \
  avg_xdr.c -lnsl
```

- start server and check RPC entry:

```
$ ./avgserver &
$ /usr/sbin/rpcinfo -p localhost
   Program Vers Proto   Port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    391002    2   tcp    892  sgi_fam
     22855    1   udp   1025
     22855    1   tcp   1025
```

- start client:

```
$ ./avgclient localhost 49.32 12.8 1.0
value   = 4.932000e+01
value   = 1.280000e+01
value   = 1.000000e+00
average = 2.104000e+01
```


NFS (1)

- NFS: Network File System
- Sun has developped RPC („Sun RPC“) for NFS
- NFS procedures:
  - mkdir, rmdir, readdir
  - create, remove, read, write
  - getattr, setattr
  - link, symlink, readlink
  - statfs

(in */usr/include/linux/nfs2.h*)

```
/*
 * Procedure numbers for NFSv2
 */
#define NFSPROC_NULL            0
#define NFSPROC_GETATTR         1
#define NFSPROC_SETATTR         2
#define NFSPROC_ROOT            3
#define NFSPROC_LOOKUP          4
#define NFSPROC_READLINK        5
#define NFSPROC_READ            6
#define NFSPROC_WRITECACHE      7
#define NFSPROC_WRITE           8
#define NFSPROC_CREATE          9
#define NFSPROC_REMOVE          10
#define NFSPROC_RENAME          11
#define NFSPROC_LINK            12
#define NFSPROC_SYMLINK         13
#define NFSPROC_MKDIR           14
#define NFSPROC_RMDIR           15
#define NFSPROC_READDIR         16
#define NFSPROC_STATFS          17
```


NFS (2)

- watch the transfer of NFS procedure requests and results:

```
tcpdump -s 1024 host 192.168.1.2 | grep nfs > /tmp/tcpdump.log
```

- on NFS client side create directory and file, rename, delete
- NFS client calls NFS procedures mkdir, create, write, read, remove aus via RPC

NFS (3)

```
cd /mnt/server; mkdir tmpdir
```

```
13:01:42.624 sony.9583 > amd64.nfs: 120 getattr fh Unknown/7B0905003C4B04009B540000EB14000002000000
13:01:42.627 amd64.nfs > sony.9583: reply ok 116 getattr DIR 40700 ids 500/100 sz 36144
13:01:42.627 sony.6799 > amd64.nfs: 124 access fh Unknown/7B0905003C4B04009B540000EB14000002000000 001f
13:01:42.630 amd64.nfs > sony.6799: reply ok 124 access c 001f
13:01:42.630 sony.4015 > amd64.nfs: 132 lookup fh Unknown/7B0905003C4B04009B540000EB14000002000000 "tmpdir"
13:01:42.632 amd64.nfs > sony.4015: reply ok 120 lookup ERROR: No such file or directory
13:01:42.632 sony.1231 > amd64.nfs: 160 mkdir fh Unknown/7B0905003C4B04009B540000EB14000002000000 "tmpdir"
13:01:42.636 amd64.nfs > sony.1231: reply ok 256 mkdir fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400
```

```
echo "Betriebssysteme" > /mnt/server/tmpdir/bs.txt
```

```
13:01:51.308 sony.2879 > amd64.nfs: 120 getattr fh Unknown/7B0905003C4B04009B540000EB14000002000000
13:01:51.313 amd64.nfs > sony.2879: reply ok 116 getattr DIR 40700 ids 500/100 sz 36168
13:01:51.313 sony0095 > amd64.nfs:  124 access fh Unknown/7B0905003C4B04009B540000EB14000002000000 001f
13:01:51.315 amd64.nfs > sony.0095: reply ok 124 access c 001f
13:01:51.315 sony.7311 > amd64.nfs: 132 lookup fh Unknown/7B0905003C4B04009B540000EB14000002000000 "tmpdir"
13:01:51.318 amd64.nfs > sony.7311: reply ok 248 lookup fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400
13:01:51.318 sony.4527 > amd64.nfs: 124 access fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 001f
13:01:51.322 amd64.nfs > sony.4527: reply ok 124 access c 001f
13:01:51.322 sony.1743 > amd64.nfs: 132 lookup fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs.txt"
13:01:51.327 amd64.nfs > sony.1743: reply ok 120 lookup ERROR: No such file or directory
13:01:51.327 sony8959 > amd64.nfs:  164 create fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs.txt"
13:01:51.332 amd64.nfs > sony.8959: reply ok 280 create fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500
13:01:51.332 sony.6175 > amd64.nfs: 124 getattr fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400
13:01:51.335 amd64.nfs > sony.6175: reply ok 188 getattr REG 2 ids 2/500 sz 2147483648000
13:01:51.336 sony.3391 > amd64.nfs: 156 write fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500 16 (16) bytes @0
13:01:51.349 amd64.nfs > sony.3391: reply ok 140 write [|nfs]
```


NFS (4)

```
cat /mnt/server/tmpdir/bs.txt
```

```
13:01:56.256 sony.30607 > amd64.nfs:  120 getattr fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400
13:01:56.259 amd64.nfs > sony.30607:  reply ok 116 getattr DIR 40755 ids 500/500 sz 72
13:01:56.259 sony.7823 > amd64.nfs:   132 lookup fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs.txt"
13:01:56.262 amd64.nfs > sony.7823:   reply ok 248 lookup fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500
13:01:56.262 sony.5039 > amd64.nfs:   124 access fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500 002d
13:01:56.264 amd64.nfs > sony.5039:   reply ok 124 access c 000d
```

```
mv /mnt/server/tmpdir/bs.txt /mnt/server/tmpdir/bs-alt.txt
```

```
13:02:03.920 sony.2255 > amd64.nfs:   136 lookup fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs-alt.txt"
13:02:03.922 amd64.nfs > sony.2255:   reply ok 120 lookup ERROR: No such file or directory
13:02:03.922 sony.9471 > amd64.nfs:   120 getattr fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500
13:02:03.925 amd64.nfs > sony.9471:   reply ok 116 getattr REG 100644 ids 500/500 sz 16
13:02:03.925 sony.6687 > amd64.nfs:   188 rename fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs.txt"
                                      -> fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs-alt.txt"
13:02:03.929 amd64.nfs > sony.6687:   reply ok 264 rename
```

```
rm /mnt/server/tmpdir/bs-alt.txt
```

```
13:02:08.376 sony.3903 > amd64.nfs:   120 getattr fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400
13:02:08.378 amd64.nfs > sony.3903:   reply ok 116 getattr DIR 40755 ids 500/500 sz 80
13:02:08.378 sony.1119 > amd64.nfs:   136 lookup fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs-alt.txt"
13:02:08.381 amd64.nfs > sony.1119:   reply ok 248 lookup fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500
13:02:08.381 sony.8335 > amd64.nfs:   124 access fh Unknown/C8C70B00ABC70B00C33F2000ABC70B007B090500 002d
13:02:08.384 amd64.nfs > sony.8335:   reply ok 124 access c 000d
13:02:08.385 sony.5551 > amd64.nfs:   136 remove fh Unknown/ABC70B007B090500C33F20007B0905003C4B0400 "bs-alt.txt"
13:02:08.390 amd64.nfs > sony.5551:   reply ok 124 remove
```


LPC: Local Procedure Call

- variant of RPC where client and server processes run on the same machine
- some simplifications are possible:
  - no need for marshalling (sender and receiver use the same data representations)
  - *call by reference* may be possible if sender and receiver process use shared memory
  - special RPC errors (e.g.: function result „server is unreachable“) cannot occur
- Microsoft LPC: Windows automatically switches to a simpler protocol when it recognizes a local connection