Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

21. Juli 2006

Interprozeßkommunikation
Schlusstest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel erlaubt sind: Das Skript, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Die Benutzung eigener Notebooks ist gestattet.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führe deshalb zunächst die folgenden Schritte durch:

- Setze die Rechte deines Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Lege in Deinem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)

Achte bei der Bearbeitung der Aufgaben auf folgendes:

- Kommentiere die notwendigen Änderungen *knapp* im Sourcecode.
- Benutze bei den Aufgaben jeweils die Programme, die sich aus der vorhergehenden Teilaufgabe ergeben.
- Kopiere nach dem Bearbeiten einer Teilaufgabe deine Programme in das Verzeichnis *~/klausur*. Benenne den Sourcecode sowie das dazugehörige kompilierte Programm dabei nach den Teilaufgaben:
  (server|client)."Aufgabennr".[cpp] also z.B. *server.3.cpp* bzw. die entsprechend kompilierte Version dann *server.3*.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

21. Juli 2006

Interprozeßkommunikation
Schlusstest

Aufgaben

Beim vorliegenden Server/Client-Paar *server.cpp* und *client.cpp* im Verzeichnis */home/IPC/klausur* schickt der Client Benutzereingaben an den Server, der die empfangenen Texte in Großbuchstaben wieder zurücksendet. Dies wiederholt der Client solange, bis er eine Eingabe erhält, die mit einem Punkt beginnt. Erhält der Server eine Eingabe, die mit einem Punkt beginnt, schließt er die Verbindung. Erstelle ausgehend von diesem Programm nach und nach einen rudimentären Schlüsselserver.

1. Ändere den Server so ab, daß er nicht mehr den Port aus der Datei *local.h* benutzt, sondern sich selbstständig einen freien Port aussucht. Dazu ist es ausserdem notwendig, daß der Server den benutzten Port ausgibt und der Client den Port, zu dem er sich verbinden soll, auf der Commandozeile übergeben bekommt. Programmiere auch diese beiden Änderungen.

2. Da die Verbindung zum Server immer nur sehr kurz ist, braucht die Verbindung nicht mit Sohnprozessen abgewickelt zu werden. Entferne also den ”fork-Teil” des Servers so, daß dieser trotzdem noch (hintereinander) mit mehreren Clients kommunizieren kann.

3. Nun soll die Ausgabe des Servers verändert werden. Anstelle der trivialen Wandlung in Großbuchstaben soll er nun zu jeder Eingabe die Ausgabe des Unix-Programms ***sha1sum*** zurückliefern. Achte darauf, dass der Client die Ausgabe des Servers komplett ausgibt und passe ihn gegebenenfalls an. Es darf davon ausgegangen werden, daß der im Client eingegebene String keine ” enthält.

4. Abschließend stellen wir fest, dass Server und Client auf Grund des kurzen Datenaustausches besser über einen verbindungslosen Socket kommunizieren würden. Ändere also Client und Server so ab, daß sie verbindungslose Sockets benutzen. Stelle sicher, daß der Server so (gleichzeitig) mit mehreren Clients kommunizieren kann.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

21. Juli 2006

Interprozeßkommunikation
Lösungsvorschlag - Schlußtest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel erlaubt sind: Das Skript, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Die Benutzung eigener Notebooks ist gestattet.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führe deshalb zunächst die folgenden Schritte durch:

- Setze die Rechte deines Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Lege in Deinem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)

Achte bei der Bearbeitung der Aufgaben auf folgendes:

- Kommentiere die notwendigen Änderungen *knapp* im Sourcecode.
- Benutze bei den Aufgaben jeweils die Programme, die sich aus der vorhergehenden Teilaufgabe ergeben.
- Kopiere nach dem Bearbeiten einer Teilaufgabe deine Programme in das Verzeichnis *~/klausur*. Benenne den Sourcecode sowie das dazugehörige kompilierte Programm dabei nach den Teilaufgaben:
  (server|client)."Aufgabennr".[cpp] also z.B. *server.3.cpp* bzw. die entsprechend kompilierte Version dann *server.3*.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

21. Juli 2006

Interprozeßkommunikation
Lösungsvorschlag - Schlußtest

Aufgaben

Beim vorliegenden Server/Client-Paar *server.cpp* und *client.cpp* im Verzeichnis */home/IPC/klausur* schickt der Client Benutzereingaben an den Server, der die empfangenen Texte in Großbuchstaben wieder zurücksendet. Dies wiederholt der Client solange, bis er eine Eingabe erhält, die mit einem Punkt beginnt. Erhält der Server eine Eingabe, die mit einem Punkt beginnt, schließt er die Verbindung. Erstelle ausgehend von diesem Programm nach und nach einen rudimentären Schlüsselserver.

- Bei den beiden Programmen handelt es sich um die bereits aus den Übungen bekannten Programme *p10.6.cxx* und *p10.7.cxx* von Gray.

1. Ändere den Server so ab, daß er nicht mehr den Port aus der Datei *local.h* benutzt, sondern sich selbstständig einen freien Port aussucht. Dazu ist es ausserdem notwendig, daß der Server den benutzten Port ausgibt und der Client den Port, zu dem er sich verbinden soll, auf der Commandozeile übergeben bekommt. Programmiere auch diese beiden Änderungen.

   - Die Aufgabe ist Teil von Übung 7.4:

Listing 1: server.1.cpp

```
1  ...
2  int
3  main(  ) {
4    //Ein paar zusaetzliche Variablen fuer "info_adr" sind
         erforderlich
5  ...
6    socklen_t        clnt_len, i_len;            // Length of
         client address
7    struct sockaddr_in                    // Internet addr client &
          server
8                     clnt_adr, serv_adr, info_adr;
9  ...
10  // hier waehlt der Server selber den Port (PORT -> 0)
11   serv_adr.sin_port         = htons(0);        // Use our fake
         port
12                                                  // BIND
```

```
13 ...
14   //Infos auslesen (getsockname)
15   i_len=sizeof(info_adr);
16   memset(&info_adr, 0, i_len);
17   if (getsockname(orig_sock, (struct  sockaddr *) &info_adr,
       &i_len) < 0)
18     {
19       perror("getsockname || gethostent error");
20       close(orig_sock);
21       exit(9);
22     }
23   //Port ausgeben
24   printf("\nPort-Nr          : %d\n", ntohs(info_adr.sin_port));
25 ...
26 }
27 ...
```

Listing 2: client.1.cpp

```
1  ...
2  int
3  main( int argc, char *argv[] ) {
4  ...
5  //Abfrage auf 3 Argumente aendern
6  if ( argc != 3 ) {                    // Check cmd line for host
       name
7      cerr << "usage: " << argv[0] << " server" << " #_port" <<
           endl;
8      return 1;
9    }
10 ...
11 //Anstelle von PORT das 2. Argument benutzen
12   serv_adr.sin_port    = htons( atoi(argv[2]) );           // Use
         our fake port
13 ...
14 }
```

2. Da die Verbindung zum Server immer nur sehr kurz ist, braucht die Verbindung nicht mit Sohnprozessen abgewickelt zu werden. Entferne also den "fork-Teil" des Servers so, daß dieser trotzdem noch (hintereinander) mit mehreren Clients kommunizieren kann.

   - Hier sind lediglich im Server 4 Zeilen zu löschen, der Client bleibt unverändert:

Listing 3: server.2.cpp

```
...
  do {
...
    }
    // fork faellt natuerlich weg, es bleibt nur die
       "Beantwortungsschleife"
    while ( (len=read(new_sock, buf, BUFSIZ)) > 0 ){
        for (i=0; i < len; ++i)                    // Change the
           case
          buf[i] = toupper(buf[i]);
        write(new_sock, buf, len);                 // Write back
           to socket
        if ( buf[0] == '.' ) break;                // Are we done
           yet?
      }
    //auch das return muss weg und close(new_sock) reicht einmal
      close(new_sock);
  } while( true );                                 // FOREVER
  return 0;
}
...
```

3. Nun soll die Ausgabe des Servers verändert werden. Anstelle der trivialen Wandlung in Großbuchstaben soll er nun zu jeder Eingabe die Ausgabe des Unix-Programms ***sha1sum*** zurückliefern. Achte darauf, dass der Client die Ausgabe des Servers komplett ausgibt und passe ihn gegebenenfalls an. Es darf davon ausgegangen werden, daß der im Client eingegebene String keine " enthält.

   - Analog zu Übung 8.2 läßt sich im Server die Beantwortung anpassen:

Listing 4: server.3.cpp

```
...
int
main(  ) {
...
//Ausserdem brauchen wir nun ein paar zusaetzliche Variablen
   fuer sha1sum
  struct hostent  *host_info;
  FILE *fin;
  static char sha1buf[BUFSIZ]; //fuer Ergebnis sha1sum

...
    // fork faellt natuerlich weg, es bleibt nur die
       "Beantwortungsschleife"
    while ( (len=read(new_sock, buf, BUFSIZ)) > 0 ){
```

```
13
14          //Pipe mit sha1sum oeffnen ...
15          memset(sha1buf, 0x0, BUFSIZ);
16          sprintf(sha1buf, "echo \"%s\" | sha1sum\n", buf);
17          fin = popen( sha1buf, "r" );
18          memset(sha1buf, 0x0, BUFSIZ);
19          // ... und auslesen
20          read(fileno(fin), sha1buf, BUFSIZ);
21          // cout << "sha1sum " << sha1buf; //debug
22
23          write(new_sock, sha1buf, BUFSIZ);                    // Write
                back to socket
24          if ( buf[0] == '.' ) break;                  // Are we done
                yet?
25      }
26  ...
27  }
28  ...
```

- Im Client muss die Leselänge geändert werden:

Listing 5: client.3.cpp

```
1   ...
2     do {                                        // Process
3       write(fileno(stdout),"> ", 3);
4       if ((len=read(fileno(stdin), buf, BUFSIZ)) > 0) {
5         write(orig_sock, buf, len);
6         //die Leselaenge muss hier veraendert werden
7         if ((len=read(orig_sock, buf, BUFSIZ)) > 0 )
8           write(fileno(stdout), buf, len);
9       }
10  ...
```

4. Abschließend stellen wir fest, dass Server und Client auf Grund des kurzen Datenaustausches besser über einen verbindungslosen Socket kommunizieren würden. Ändere also Client und Server so ab, daß sie verbindungslose Sockets benutzen. Stelle sicher, daß der Server so (gleichzeitig) mit mehreren Clients kommunizieren kann.

   - Hier muss etwas mehr getan werden als bei den Aufgaben zuvor. Leichter wird's wenn man sich an Übung 9.1 und 9.2 orientiert:

Listing 6: server.4.cpp

```
1   #include "local.h"
2   #include <netdb.h>
3   void signal_catcher(int);
```

```
4  int
5  main(  ) {
6    int              sock;           // Original socket in server
7    socklen_t        client_len, server_len;           // Length
         of client address
8    struct sockaddr_in                    // Internet addr client &
         server
9                     client, server;
10   int              len, i;             // Misc counters, etc.
11                                        // Catch when child
                                              terminates
12   struct hostent  *host_info;
13   FILE *fin;
14   static char sha1buf[BUFSIZ]; //fuer Ergebnis sha1sum
15   if (signal(SIGCHLD , signal_catcher) == SIG_ERR) {
16     perror("SIGCHLD");
17     return 1;
18   }
19   if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
20     perror("SERVER socket ");  return 1;
21   }
22   memset(&server, 0, sizeof(server));  // Clear structure
23   server.sin_family      = AF_INET;    // Set address type
24   server.sin_addr.s_addr = htonl(INADDR_ANY);
25   server.sin_port        = htons(0);
26                                        // BIND
27   if (bind(sock, (struct sockaddr *) &server,
28       sizeof(server) ) < 0) {
29     perror("SERVER bind ");  return 2;
30   }
31   server_len = sizeof(server);         // Obtain  address length
32                                        // Find picked port #
33   if (getsockname(sock, (struct sockaddr *) &server,
34       &server_len) < 0) {
35     perror("SERVER getsocketname ");  return 3;
36   }
37   cout << "Server using port " << ntohs(server.sin_port) <<
         endl;
38
39     do {
40     client_len = sizeof(client);                  // set the
           length
41     memset(buf, 0, BUFSIZ);                       // clear the
           buffer
42     if ((len=recvfrom(sock, buf, BUFSIZ, 0,        // get the
           client's msg
43         (struct sockaddr *) &client, &client_len)) < 0){
44       perror("SERVER recvfrom ");
45       close(sock); return 4;
```

```
46      }
47
48           memset(sha1buf, 0x0, BUFSIZ);
49      sprintf(sha1buf, "echo \"%s\" | md5sum\n", buf);
50      fin = popen( sha1buf, "r" );
51      memset(sha1buf, 0x0, BUFSIZ);
52      read(fileno(fin), sha1buf, BUFSIZ);
53      cout << "sha1sum " << sha1buf;
54
55      if ((sendto(sock, sha1buf, strlen(sha1buf) ,0,      /* send
            it to client */
56                      (struct sockaddr *) &client,
57                      sizeof(struct sockaddr_in))) < 0)
58                    {
59                    perror("SERVER sendto ");
60            return 1;
61            }
62
63    } while( true );                                  // FOREVER
64    return 0;
65  }
66  void
67  signal_catcher(int the_sig){
68    signal(the_sig, signal_catcher);                  // reset
69    wait(0);                                          // keep the
        zombies at bay
70  }
```

Listing 7: client.4.cpp

```
1   #include "local.h"
2   int
3   main( int argc, char *argv[] ) {
4     int             sock,           // Original socket in client
5                     len;                  // Misc. counter
6     socklen_t       server_len;
7     struct sockaddr_in
8                     server, client;           // Internet addr of
                          server
9     struct hostent  *host;                  // The host (server) info
10    static char sha1buf[BUFSIZ]; //fuer Ergebnis sha1sum
11  //Abfrage auf 3 Argumente aendern
12  if ( argc != 3 ) {                    // Check cmd line for host
        name
13      cerr << "usage: " << argv[0] << " server" << " #_port" <<
            endl;
14      return 1;
15    }
16    if (!(host=gethostbyname(argv[1]))){
```

```
17      perror("CLIENT gethostname ");  return 2;
18    }                                       // Set server address
          info
19    memset(&server, 0, sizeof(server));  // Clear structure
20    server.sin_family = AF_INET;         // Address type
21    memcpy(&server.sin_addr, host->h_addr, host->h_length);
22    server.sin_port   = htons(atoi(argv[2]));
23                                         // SOCKET
24    if ((sock=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) {
25      perror("CLIENT socket "); return 3;
26    }                                       // Set client address
          info
27    memset(&client, 0, sizeof(client));  // Clear structure
28    client.sin_family      = AF_INET;    // Address type
29    client.sin_addr.s_addr = htonl(INADDR_ANY);
30    client.sin_port        = htons( 0 );
31                                         // BIND
32    if (bind(sock, (struct sockaddr *) &client,
33        sizeof(client)) < 0) {
34      perror("CLIENT bind "); return 4;
35    }
36    do {                                    // Process
37      write(fileno(stdout),"> ", 3);
38      if ((len=read(fileno(stdin), buf, BUFSIZ)) > 0) {
39        server_len=sizeof(server);
40        if (sendto( sock, buf, strlen(buf), 0,            /* send
              msg to server */
41                  (struct sockaddr *) &server, server_len) < 0
                        ){
42          perror("CLIENT sendto ");
43          close(sock); exit(5);
44        }
45      if ((len=recvfrom(sock, sha1buf, BUFSIZ, 0,            /*
              server's message   */
46                        (struct sockaddr *) &server,
                             &server_len)) < 0){
47            perror("CLIENT recvfrom ");
48            close(sock);  exit(6);
49        }
50      write(fileno(stdout),sha1buf,len);                     /*
          show msg to clnt   */
51      }
52    } while( buf[0] != '.' );            // until end of input
53    close(sock);
54    return 0;
55  }
```

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL
VERSITÄT

14. Februar 2007

Interprozeßkommunikation
Schlusstest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel erlaubt sind: Das Skript, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)

Achten Sie bei der Bearbeitung der Aufgaben auf folgendes:

- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".
- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

14. Februar 2007

Aufgabe

Die vorliegenden Fragmente des Client/Server-Paars aus dem Verzeichnis */home/IPC/klausur* sollen zu einem rudimentären Network Time Protokoll (NTP) - Client/Server vervollständigt werden.
Mit Hilfe des NTP-Protokolls können zwei Rechner ihre Zeit synchronisieren. Dabei läuft die Zeitsynchronisation zwischen Client und Server vereinfacht wie folgt ab. Der Client schickt eine Nachricht mit seiner aktuellen Zeit (t1) an der Server. Dieser antwortet, indem er die Ankunftszeit des Pakets vom Client (t2) und den Zeitpunkt seiner Antwort (t3) an den Client zurückschickt. Mit dem Zeitpunkt des Eintreffens seiner Antwort beim Client (t4) ergeben sich so vier Zeitstempel.
Aus diesen Zeitstempeln lassen sich nun zwei Größen bestimmen. Zum einen das "Delay", also die Zeit, die die Nachrichten im Netz unterwegs waren, sowie das "Offset", d.h. die Zeitspanne, worin die Uhren der Rechner differieren:

$$delay = (t4 - t1) - (t3 - t2)$$

$$offset = \frac{(t4 - t3) + (t1 - t2)}{2}$$

Ergänzen Sie nun die Codefragmente zu einem rudimentären Network Time Protokoll (NTP) - Client/Server. Beachten Sie dabei folgende Hinweise:

- Client und Server sollen mittels UDP (verbindungslosen Sockets) miteinander kommunizieren; dabei soll der Server seine Portnummer zufällig wählen und anschließend ausgeben.
- Zeitstempel sollen mit der Funktion gettimeofday(struct timeval *restrict tp, void *restrict tz) erstellt werden, die entsprechenden Aufrufe befinden sich bereits in den Codefragmenten.
- Um die Zeitstempel zu verschicken soll die Struktur ntp_timestamps verwendet werden. Sie besteht lediglich aus vier Zeitstempeln des Typs timeval und wird im File *local.h* definiert.
- Die Berechnungen von delay und offset im Client sind bereits vollständig.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

14. Februar 2007

Interprozeßkommunikation
Lösungsvorschlag - Schlußtest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel erlaubt sind: Das Skript, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)

Achten Sie bei der Bearbeitung der Aufgaben auf folgendes:

- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".
- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

14. Februar 2007

Aufgabe

Die vorliegenden Fragmente des Client/Server-Paars aus dem Verzeichnis */home/IPC/klausur* sollen zu einem rudimentären Network Time Protokoll (NTP) - Client/Server vervollständigt werden.
Mit Hilfe des NTP-Protokolls können zwei Rechner ihre Zeit synchronisieren. Dabei läuft die Zeitsynchronisation zwischen Client und Server vereinfacht wie folgt ab. Der Client schickt eine Nachricht mit seiner aktuellen Zeit (t1) an der Server. Dieser antwortet, indem er die Ankunftszeit des Pakets vom Client (t2) und den Zeitpunkt seiner Antwort (t3) an den Client zurückschickt. Mit dem Zeitpunkt des Eintreffens seiner Antwort beim Client (t4) ergeben sich so vier Zeitstempel.
Aus diesen Zeitstempeln lassen sich nun zwei Größen bestimmen. Zum einen das "Delay", also die Zeit, die die Nachrichten im Netz unterwegs waren, sowie das "Offset", d.h. die Zeitspanne, worin die Uhren der Rechner differieren:

$$delay = (t4 - t1) - (t3 - t2)$$

$$offset = \frac{(t4 - t3) + (t1 - t2)}{2}$$

Ergänzen Sie nun die Codefragmente zu einem rudimentären Network Time Protokoll (NTP) - Client/Server. Beachten Sie dabei folgende Hinweise:

- Client und Server sollen mittels UDP (verbindungslosen Sockets) miteinander kommunizieren; dabei soll der Server seine Portnummer zufällig wählen und anschließend ausgeben.
- Zeitstempel sollen mit der Funktion gettimeofday(struct timeval *restrict tp, void *restrict tz) erstellt werden, die entsprechenden Aufrufe befinden sich bereits in den Codefragmenten.
- Um die Zeitstempel zu verschicken soll die Struktur ntp_timestamps verwendet werden. Sie besteht lediglich aus vier Zeitstempeln des Typs timeval und wird im File *local.h* definiert.
- Die Berechnungen von delay und offset im Client sind bereits vollständig.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

14. Februar 2007

Listing 1: client.cpp

```
1  #include "local.h"
2  int
3  main( int argc, char *argv[] ) {
4    int             sock,
5                    len;
6    socklen_t       server_len;
7    struct sockaddr_in
8                    server, client;
9    struct hostent  *host;
10   ntp_timestamps ts;
11   double delay, offset;
12
13   // Zahl der Argumente abfragen
14   if ( argc != 3 ) {
15     cerr << "usage: " << argv[0] << " server" << " #port" <<
         endl;
16     return 1;
17   }
18
19   // Kommunikation vorbereiten
20
21   if (!(host=gethostbyname(argv[1]))){
22     perror("CLIENT gethostname "); return 2;
23   }                                    // Set server address
       info
24   memset(&server, 0, sizeof(server));  // Clear structure
25   server.sin_family = AF_INET;         // Address type
26   memcpy(&server.sin_addr, host->h_addr, host->h_length);
27   server.sin_port   = htons(atoi(argv[2]));
28
29   // SOCKET
30   if ((sock=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) {
31     perror("CLIENT socket "); return 3;
32   }                                    // Set client address
       info
33   memset(&client, 0, sizeof(client));  // Clear structure
34   client.sin_family      = AF_INET;    // Address type
35   client.sin_addr.s_addr = htonl(INADDR_ANY);
36   client.sin_port        = htons( 0 );
37
38   //BIND
39   if (bind(sock, (struct sockaddr *) &client,
40       sizeof(client)) < 0) {
41     perror("CLIENT bind "); return 4;
42   }
43
44   //Zeitstempel generieren, verschicken und die Antwort
       entgegennehmen
```

```
45
46    gettimeofday(&ts.t1, 0);
47
48    server_len=sizeof(server);
49    if (sendto( sock, &ts, sizeof(ts), 0,              /* send msg
          to server */
50                (struct sockaddr *) &server, server_len) < 0 ){
51          perror("CLIENT sendto ");
52       close(sock); return 5;
53    }
54
55    if ((len=recvfrom(sock, &ts, sizeof(ts), 0,              /*
          server's message   */
56                (struct sockaddr *) &server, &server_len)) < 0){
57          perror("CLIENT recvfrom ");
58          close(sock);  return 6;
59    }
60
61    gettimeofday(&ts.t4, 0);
62
63    // Verbindung schliessen
64    close(sock);
65
66    // debug output
67    //  cout << ts.t1.tv_sec << ':' << ts.t1.tv_usec << endl;
68    //  cout << ts.t2.tv_sec << ':' << ts.t2.tv_usec << endl;
69    //  cout << ts.t3.tv_sec << ':' << ts.t3.tv_usec << endl;
70    //  cout << ts.t4.tv_sec << ':' << ts.t4.tv_usec << endl;
71
72    // Berechnungen
73
74    delay = (double) ((ts.t4.tv_sec - ts.t1.tv_sec) -
          (ts.t3.tv_sec - ts.t2.tv_sec) +
75              (double)((ts.t4.tv_usec - ts.t1.tv_usec) -
                  (ts.t3.tv_usec - ts.t2.tv_usec))/1000000.0);
76    offset = (double) (((ts.t4.tv_sec - ts.t3.tv_sec) -
          (ts.t2.tv_sec - ts.t1.tv_sec) +
77               (double)((ts.t4.tv_usec - ts.t3.tv_usec) -
                   (ts.t2.tv_usec - ts.t1.tv_usec))/1000000.0)/2);
78
79    printf("delay=  %f\n",delay);
80    printf("offset= %f\n",offset);
81    return 0;
82 }
```

Listing 2: server.cpp

```
1  #include "local.h"
2  void signal_catcher(int);
```

```
3  int
4  main(  ) {
5    int              sock;
6    socklen_t        client_len, server_len;
7    struct sockaddr_in
8                     client, server;
9    int              len;
10   struct hostent   *host_info;
11   ntp_timestamps ts;
12
13   // Signal catcher
14
15   if (signal(SIGCHLD , signal_catcher) == SIG_ERR) {
16     perror("SIGCHLD");
17     return 1;
18   }
19
20   // Kommunikation vorbereiten; Portnummer zufaellig waehlen
21
22   if ((sock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
23     perror("SERVER socket "); return 1;
24   }
25   memset(&server, 0, sizeof(server));  // Clear structure
26   server.sin_family      = AF_INET;    // Set address type
27   server.sin_addr.s_addr = htonl(INADDR_ANY);
28   server.sin_port        = htons(0);
29                                       // BIND
30   if (bind(sock, (struct sockaddr *) &server,
31       sizeof(server) ) < 0) {
32     perror("SERVER bind "); return 2;
33   }
34   server_len = sizeof(server);        // Obtain  address length
35                                       // Find picked port #
36   if (getsockname(sock, (struct sockaddr *) &server,
37       &server_len) < 0) {
38     perror("SERVER getsocketname "); return 3;
39   }
40
41   cout << "Server using port " << ntohs(server.sin_port) <<
       endl;
42
43   // Endlosschleife in der die Anfragen entgegengenommen und
       beantwortet werden
44
45     do {
46     client_len = sizeof(client);                  // set the
         length
47
```

```
48      if ((len=recvfrom(sock, &ts, sizeof(ts), 0,              /*
            client's message   */
49                  (struct sockaddr *) &client, &client_len)) < 0){
50              perror("CLIENT recvfrom ");
51              close(sock);  return 4;
52        }
53
54      gettimeofday(&ts.t2, 0);
55      cout << ts.t2.tv_sec << ':' << ts.t2.tv_usec << endl;
56           //kleine Pause
57      srand ( time(NULL) );
58      sleep((rand()%2)+1);
59      gettimeofday(&ts.t3, 0);
60
61      if (sendto( sock, &ts, sizeof(ts), 0,            /* send msg
            to client */
62                  (struct sockaddr *) &client, client_len) < 0 ){
63              perror("SERVER sendto ");
64              close(sock); return 5;
65        }
66    } while( true );                                  // FOREVER
67
68    return 0;
69  }
70
71  void
72  signal_catcher(int the_sig){
73    signal(the_sig, signal_catcher);                  // reset
74    wait(0);                                          // keep the
        zombies at bay
75  }
```

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

31. Juli 2007

Interprozesskommunikation
Schlusstest

Allgemeines

Grundlage des Schlusstests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.

Als Hilfsmittel erlaubt sind: Das Skript, Bücher, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist. Die Benutzung von Suchmaschinen oder des WWW ist nicht erlaubt.

Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - **führen zum Ausschluss von der Prüfung**.

Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur.ss07" an.
  (***mkdir ~/klausur.ss07***)

Online-Zugriff auf das Skript erhalten Sie, indem Sie auf ihrem Arbeitsrechner folgenden Befehl ausführen:
evince http://www.db.informatik.uni-kassel.de/IPC.pdf

Unseren Server erreichen Sie wie gewohnt mit:
ssh [-X] ⟨username⟩@rapunzel.db.informatik.uni-kassel.de

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

31. Juli 2007

Achten Sie bei der Bearbeitung der Aufgaben auf folgendes:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie bei den Aufgaben jeweils die Programme, die sich aus der vorhergehenden Teilaufgabe ergeben.
- Kopieren Sie nach dem Bearbeiten einer Teilaufgabe Ihr Programm in das Verzeichnis *~/klausur.ss07*. **Benennen Sie den Sourcecode und** zugehöriges **kompiliertes Programm dabei nach den Teilaufgaben**: (server|client)."Aufgabennr".[cpp] also z.B. *server.3.cpp* bzw. die entsprechend kompilierte Version dann *server.3*. Alles außerhalb des Verzeichnisses ~/klausur.ss07 wird nicht für die Bewertung berücksichtigt.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Benutzen Sie – sofern vorhanden und sinnvoll – die bereits in der Includedatei und den Quelldateien definierten Variablen.
- **Geben Sie gesendete und empfangene Daten im Client und Server auf der Standardausgabe aus, um die Vorgänge transparent zu halten.**
- Achten Sie darauf, dass Ihre Programme sich auch auf anderen Systemen als rapunzel wie gewünscht verhalten.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL
VERSITÄT

31. Juli 2007

Aufgabe

Mit den vorliegenden Fragmenten des Client/Server-Paars aus dem Verzeichnis */home/IPC/klausur* soll das Prinzip eines FTP Server/Client-Paares stark vereinfacht nachgestellt werden. Aus Gründen der Übersichtlichkeit wandelt der Server jedoch nur alle Zeichenketten, die er erhält in Großbuchtaben um. Im Grundzustand sendet der Client die Eingabe an den Server und beendet sich, wenn das erste Zeichen ein Punkt ist.

1. Ändern Sie Server und Client so ab, dass sich der Client zum Server über einen festen Port verbindet. Benutzen sie als festen Port dabei 10 000 plus die Nummer Ihres Arbeitsrechners, die Sie am Gehäuse finden. Ist z.B. die Nummer Ihres Arbeitsrechners 42, dann soll der Client eine Verbindung zu Port 10 042 des Servers aufbauen.
   Ändern Sie weiterhin den Server so ab, dass er nach der Verbindung des Clients seine Systeminformationen an den Client sendet. Benutzen Sie dazu den Befehl ”uname -a”. Geben Sie die Systeminformationen im Client aus.

2. Ändern Sie den Server derart ab, dass er auch mehrere Clients gleichzeitig bedienen kann.

Nun sollen extra Datenverbindungen implementiert werden. Dabei sollen über die erste Verbindung (vom Client zum Server) die ”Befehle” des Clients und die Bestätigungen des Servers gesendet werden – im weiteren als ”Kommandoverbindung” bezeichnet. Die Daten des Servers sollen über die neu anzulegende Datenverbindung gesendet werden. Der Ablauf soll dabei wie folgt sein:
Der Client soll wie zuvor eine Verbindung zum Server aufbauen (Kommandoverbindung). Nun wartet der Client auf die Eingabe ACT oder PSV, um zu entscheiden, wie die Datenverbindung zum Server aufgebaut werden soll (siehe unten). Nach Eingabe des entsprechenden Kommandos wird die Datenverbindung aufgebaut. Alle weiteren Eingaben im Client werden über die Kommandoverbindung gesendet. Der Server quittiert den Empfang auf der Kommandoverbindung mit einem OK und sendet dann auf der Datenverbindung die in Großbuchstaben gewandelte Eingabe zurück. Hat der Client das OK erhalten, liest er die Datenverbindung und zeigt die Daten an. Die Befehlsstrings sind dabei in den Variablen COM_* in der *local.h* definiert.

Der Aufbau der Verbindung im Einzelnen:

3. Aktive Verbindung:

   - Nach Eingabe des Befehls ACT über die Tastatur öffnet der Client einen neuen Port und sendet dem Server auf der Kommandoverbindung mittels "ACT␣⟨*Portnummer*⟩" die Portnummer (also z.B. "ACT␣54321").
   - Der Server sendet dem Client auf der Kommandoverbindung ein OK und verbindet sich dann zur Datenverbindung.
   - Alle weiteren Eingaben des Clients werden als Befehl über die Kommandoverbindung übertragen.

   Wie oben beschrieben quittiert der Server den Empfang eines Befehls des Clients auf der Kommandoverbindung mit einem OK und sendet dann auf der Datenverbindung den in Großbuchstaben gewandelten Befehl des Clients zurück. Hat der Client das OK erhalten, liest er die Datenverbindung und zeigt die Daten an. Ist das erste Zeichen der Eingabe des Clients ein Punkt, so beendet sich der Client.

4. Passive Verbindung:

   - Nach Eingabe des Befehls PSV sendet der Client dem Server auf der Kommandoverbindung mittels "PSV" den Wunsch nach einer passiven Verbindung.
   - Der Server öffnet einen Port und sendet dem Client auf der Kommandoverbindung mittels OK "OK␣⟨*Portnummer*⟩" die Portnummer (also z.B. "OK␣54321").
   - Der Client verbindet sich zum Server auf dem angegebenen Port.
   - Alle weiteren Eingaben des Clients werden als Befehl über die Kommandoverbindung übertragen.

   Die übrige "Datenübertragung" erfolgt wie bereits gehabt.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

31. Juli 2007

Interprozesskommunikation
Lösungsvorschlag - Schlußtest

Allgemeines

Grundlage des Schlusstests sind die Programme *server.cpp*, *client.cpp* und *local.h* im Verzeichnis */home/IPC/klausur*.

Als Hilfsmittel erlaubt sind: Das Skript, Bücher, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist. Die Benutzung von Suchmaschinen oder des WWW ist nicht erlaubt.

Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - **führen zum Ausschluss von der Prüfung**.

Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur.ss07" an.
  (***mkdir ~/klausur.ss07***)

Online-Zugriff auf das Skript erhalten Sie, indem Sie auf ihrem Arbeitsrechner folgenden Befehl ausführen:
evince http://www.db.informatik.uni-kassel.de/IPC.pdf

Unseren Server erreichen Sie wie gewohnt mit:
ssh [-X] ⟨username⟩@rapunzel.db.informatik.uni-kassel.de

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

31. Juli 2007

Achten Sie bei der Bearbeitung der Aufgaben auf folgendes:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie bei den Aufgaben jeweils die Programme, die sich aus der vorhergehenden Teilaufgabe ergeben.
- Kopieren Sie nach dem Bearbeiten einer Teilaufgabe Ihr Programm in das Verzeichnis *~/klausur.ss07*. **Benennen Sie den Sourcecode und** zugehöriges **kompiliertes Programm dabei nach den Teilaufgaben**: (server|client)."Aufgabennr".[cpp] also z.B. *server.3.cpp* bzw. die entsprechend kompilierte Version dann *server.3*. Alles außerhalb des Verzeichnisses ~/klausur.ss07 wird nicht für die Bewertung berücksichtigt.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Benutzen Sie – sofern vorhanden und sinnvoll – die bereits in der Includedatei und den Quelldateien definierten Variablen.
- **Geben Sie gesendete und empfangene Daten im Client und Server auf der Standardausgabe aus, um die Vorgänge transparent zu halten.**
- Achten Sie darauf, dass Ihre Programme sich auch auf anderen Systemen als rapunzel wie gewünscht verhalten.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL
VERSITÄT

31. Juli 2007

Aufgabe

Mit den vorliegenden Fragmenten des Client/Server-Paars aus dem Verzeichnis */home/IPC/klausur* soll das Prinzip eines FTP Server/Client-Paares stark vereinfacht nachgestellt werden. Aus Gründen der Übersichtlichkeit wandelt der Server jedoch nur alle Zeichenketten, die er erhält in Großbuchtaben um. Im Grundzustand sendet der Client die Eingabe an den Server und beendet sich, wenn das erste Zeichen ein Punkt ist.

1. Ändern Sie Server und Client so ab, dass sich der Client zum Server über einen festen Port verbindet. Benutzen sie als festen Port dabei 10 000 plus die Nummer Ihres Arbeitsrechners, die Sie am Gehäuse finden. Ist z.B. die Nummer Ihres Arbeitsrechners 42, dann soll der Client eine Verbindung zu Port 10 042 des Servers aufbauen.
   Ändern Sie weiterhin den Server so ab, dass er nach der Verbindung des Clients seine Systeminformationen an den Client sendet. Benutzen Sie dazu den Befehl ”uname -a”. Geben Sie die Systeminformationen im Client aus.

2. Ändern Sie den Server derart ab, dass er auch mehrere Clients gleichzeitig bedienen kann.

Nun sollen extra Datenverbindungen implementiert werden. Dabei sollen über die erste Verbindung (vom Client zum Server) die ”Befehle” des Clients und die Bestätigungen des Servers gesendet werden – im weiteren als ”Kommandoverbindung” bezeichnet. Die Daten des Servers sollen über die neu anzulegende Datenverbindung gesendet werden. Der Ablauf soll dabei wie folgt sein:
Der Client soll wie zuvor eine Verbindung zum Server aufbauen (Kommandoverbindung). Nun wartet der Client auf die Eingabe ACT oder PSV, um zu entscheiden, wie die Datenverbindung zum Server aufgebaut werden soll (siehe unten). Nach Eingabe des entsprechenden Kommandos wird die Datenverbindung aufgebaut. Alle weiteren Eingaben im Client werden über die Kommandoverbindung gesendet. Der Server quittiert den Empfang auf der Kommandoverbindung mit einem OK und sendet dann auf der Datenverbindung die in Großbuchstaben gewandelte Eingabe zurück. Hat der Client das OK erhalten, liest er die Datenverbindung und zeigt die Daten an. Die Befehlsstrings sind dabei in den Variablen COM_* in der *local.h* definiert.

Der Aufbau der Verbindung im Einzelnen:

3. Aktive Verbindung:

   - Nach Eingabe des Befehls ACT über die Tastatur öffnet der Client einen neuen Port und sendet dem Server auf der Kommandoverbindung mittels "ACT␣⟨*Portnummer*⟩" die Portnummer (also z.B. "ACT␣54321").
   - Der Server sendet dem Client auf der Kommandoverbindung ein OK und verbindet sich dann zur Datenverbindung.
   - Alle weiteren Eingaben des Clients werden als Befehl über die Kommandoverbindung übertragen.

   Wie oben beschrieben quittiert der Server den Empfang eines Befehls des Clients auf der Kommandoverbindung mit einem OK und sendet dann auf der Datenverbindung den in Großbuchstaben gewandelten Befehl des Clients zurück. Hat der Client das OK erhalten, liest er die Datenverbindung und zeigt die Daten an. Ist das erste Zeichen der Eingabe des Clients ein Punkt, so beendet sich der Client.

4. Passive Verbindung:

   - Nach Eingabe des Befehls PSV sendet der Client dem Server auf der Kommandoverbindung mittels "PSV" den Wunsch nach einer passiven Verbindung.
   - Der Server öffnet einen Port und sendet dem Client auf der Kommandoverbindung mittels OK "OK␣⟨*Portnummer*⟩" die Portnummer (also z.B. "OK␣54321").
   - Der Client verbindet sich zum Server auf dem angegebenen Port.
   - Alle weiteren Eingaben des Clients werden als Befehl über die Kommandoverbindung übertragen.

   Die übrige "Datenübertragung" erfolgt wie bereits gehabt.

Fachbereich Mathematik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

31. Juli 2007

Listing 1: client.cpp Lösungsvorschlag

```
1  #include "local.h"
2  int
3  main( int argc, char *argv[] ) {
4    int             orig_sock,                  // Original socket
         in client
5                    data_sock;                  // Data socket in
                         client
6
7    struct sockaddr_in                          // Server address
8                    serv_adr;
9    socklen_t                    serv_len;
                     // Length of server address
10
11   int serv_data_port;
                 // Data port for server (passive)
12
13   struct hostent  *host;                      // The host
         (server) info
14
15   struct sockaddr_in
                          // Struct for address information
16                   info_adr;
17   socklen_t info_len;
18
19   struct hostent  *host_info;
          // The data port info (active)
20
21   struct sockaddr_in                          // Address of data
         transfer (active)
22                   data_adr;
23
24   int             len;                      // Misc. counter
25
26   static char uname_buf[UNAME_SIZE];          // output from uname
          command
27   static char buf[BUFSIZ];                    // Buffer for server
            messages
28   static char port_buf[BUFSIZ];               // Buffer for server
          port (passive)
29
30   if ( argc != 2 ) {                        // Check cmd line for
         host name
31     cerr << "usage: " << argv[0] << " server" << endl;
32     return 1;
33   }
34   host = gethostbyname(argv[1]);            // Obtain host (server)
         info
35   if (host == (struct hostent *) NULL ) {
```

```
36      perror("gethostbyname ");
37      return 2;
38    }
39    memset(&serv_adr, 0, sizeof( serv_adr));        // Clear
          structure
40    serv_adr.sin_family = AF_INET;                  // Set address
          type
41    memcpy(&serv_adr.sin_addr, host->h_addr, host->h_length);
42    serv_adr.sin_port    = htons( PORT );
43                                                    // SOCKET
44    if ((orig_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
45      perror("generate error");
46      return 3;
47    }                                               // CONNECT
48    if (connect( orig_sock,(struct sockaddr *)&serv_adr,
49                 sizeof(serv_adr)) < 0) {
50      perror("connect error");
51      return 4;
52    }
53    //uname ausgeben
54    if (read(orig_sock, uname_buf, UNAME_SIZE) > 0) {
55        write(fileno(stdout), uname_buf, UNAME_SIZE);
56      }
57
58      while ((len=read(fileno(stdin), buf, BUFSIZ)) > 0) {
59  //
      ***************************************************************
60  // ********************** ACTIVE
      ***********************************
61  //
      ***************************************************************
62                  if ( strncmp(COM_ACTIVE, buf, strlen(buf)-1) == 0
                        ) { // ACTIVE // filter newline
63                       if ((data_sock = socket(PF_INET,
                             SOCK_STREAM, 0)) < 0) {
64                       perror("generate error");
65                       return 5;
66                       }
67
68                       memset( &data_adr, 0, sizeof(data_adr)
                             );       // Clear structure
69                       data_adr.sin_family       = AF_INET;
                                      // Set address type
70                       data_adr.sin_addr.s_addr =
                             htonl(INADDR_ANY);  // Any interface
71                       data_adr.sin_port         = 0;        //
                             Use our fake port
72                                                 // BIND
```

```
73                          if (bind( data_sock, (struct sockaddr
                                *) &data_adr,
74                  sizeof(data_adr)) < 0){
75                          perror("bind error");
76                          close(data_sock);
77                          return 6;
78                          }
79                          if (listen(data_sock, 5) < 0 ) {
                                                // LISTEN
80                          perror("listen error");
81                          close (data_sock);
82                          return 7;
83                          }
84                          //port ausgeben
85
86                          info_len=sizeof(info_adr);
87                          memset(&info_adr, 0, info_len);
88
89                          if (getsockname(data_sock, (struct
                                sockaddr *) &info_adr, &info_len) < 0
90                                  || (host_info=gethostent()) ==
                                     (struct hostent*) NULL) {
91                  perror("getsockname || gethostent error");
92                  close(data_sock);
93                  exit(9);
94                  }
95
96
97          sprintf(buf,"%s %d", COM_ACTIVE, info_adr.sin_port);
98          cout << "Sending: "<<buf<<endl;
99                  write(orig_sock, buf, BUFSIZ);
100
101                 read(orig_sock, buf, BUFSIZ);
102                 cout << "Received: "<<buf<<endl;
103
104                         if ( strncmp(COM_OK,buf,strlen(buf)) !=
                                0 ) continue; // !OK - continue
105
106                         cout <<"Opening port
                                "<<ntohs(info_adr.sin_port)<<"
                                ..."<<endl;
107                         serv_len = sizeof(serv_adr);
                                                // ACCEPT a connect
108                 if ((data_sock = accept( data_sock, (struct
                        sockaddr *) &serv_adr,
109                                 &serv_len)) < 0) {
110                 perror("accept error");
111                 close(orig_sock);
112                 return 8;
```

```
113                         }
114                         break;
115                 }//if
116 //
    ********************************************************************
117 // ********************** PASSIVE
    ***********************************
118 //
    ********************************************************************
119
120         if ( strncmp(COM_PASSIVE,buf,strlen(buf)-1) == 0 ) {
                // PASSIVE // filter newline
121       sprintf(buf,"%s", COM_PASSIVE);
122       cout << "Sending: "<<buf<<endl;
123         write(orig_sock, buf, BUFSIZ);
124
125         read(orig_sock, buf, BUFSIZ);
126         cout << "Receiving: "<<buf<<endl;
127
128         if ( strncmp(COM_OK,buf,2) != 0 ) continue; // !OK -
                continue
129
130                 strcpy(port_buf,strstr(buf," "));
131                 serv_data_port = ntohs(atoi(port_buf));
132                 cout << "Connecting to port
                    "<<serv_data_port<<" ..."<<endl;
133
134                 serv_adr.sin_port    = htons( serv_data_port );
                                // Use the given port
135
136                                         // SOCKET
137                 if ((data_sock = socket(PF_INET, SOCK_STREAM,
                    0)) < 0) {
138                 perror("generate error");
139                 return 9;
140                 }
141                                                                 //
                                                                    CON
142                 if (connect( data_sock,(struct sockaddr
                    *)&serv_adr,
143                 sizeof(serv_adr)) < 0) {
144                 perror("connect error");
145                 return 10;
146                 }
147                 break;
148          }//if PASSIVE
149
150         cout << "Command: unknown."<<endl;
151         memset(&buf, 0, BUFSIZ);
```

```
152         }//while
153
154   do {                                    // Process
155     write(fileno(stdout),"> ", 3);
156     memset(&buf, 0, BUFSIZ);
157     if ((len=read(fileno(stdin), buf, BUFSIZ)) > 0) {
158       write(orig_sock, buf, len);
159       memset(&buf, 0, BUFSIZ);
160       if ((len=read(orig_sock, buf, BUFSIZ)) > 0 )
161         if ( strncmp(COM_OK, buf, strlen(buf)) != 0 )
162                 write(fileno(stdout), buf, len);
163         else {
164         memset(&buf, 0, BUFSIZ);
165         if ((len=read(data_sock, buf, BUFSIZ)) > 0 )
166                         write(fileno(stdout), buf, len);
167         }
168     }
169   } while( buf[0] != '.' );               // until end of input
170   close(orig_sock);
171   return 0;
172 }
```

Listing 2: server.cpp Lösungsvorschlag

```
1  #include "local.h"
2  void signal_catcher(int);
3  int
4  main(  ) {
5    int             serv_orig_sock,          // Original socket
         in server
6                    serv_new_sock,           // New socket from
                         connect
7                    clnt_data_sock;          // Data socket for
                         client
8
9    struct sockaddr_in                        // Server address
10                   serv_adr;
11   socklen_t                      serv_len;
                     // Length of server address
12
13   struct sockaddr_in                       // Client address
14                   clnt_adr;
15   socklen_t       clnt_len;                // Length of client
         address
16
17   int             clnt_data_port;          // Port for client -
          active/passive
18   struct sockaddr_in                       // Client address
         (data)
19                   clnt_data_adr;
20   socklen_t       clnt_data_len;           // Length of client
         address (data)
21
22
23   struct sockaddr_in                       // Struct for
         address information
24                   info_adr;
25   socklen_t info_len;
26
27   struct hostent  *host_info;
          // The host (server) info (passive)
28
29   int             len, i;                  // Misc counters,
         etc.
30
31   static char port_buf[BUFSIZ];            // Buffer for client
          port (active)
32   static char uname_buf[UNAME_SIZE];       // Buffer for output
          of uname command
33   static char buf[BUFSIZ];                 // Buffer for client
          messages
34
```

```
35   FILE *pipefile;                                    // File for pipe I/O
36
37
38 // Catch when child terminates
39   if (signal(SIGCHLD , signal_catcher) == SIG_ERR) {
40     perror("SIGCHLD");
41     return 1;
42   }
43
44 // Initialize Socket
45   if ((serv_orig_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
46     perror("generate error");
47     return 2;
48   }
49   memset( &serv_adr, 0, sizeof(serv_adr) );       // Clear
       structure
50   serv_adr.sin_family      = AF_INET;             // Set address
       type
51   serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);   // Any
       interface
52   serv_adr.sin_port        = htons(PORT);         // Use our fake
        port
53                                                   // BIND
54   if (bind( serv_orig_sock, (struct sockaddr *) &serv_adr,
55             sizeof(serv_adr)) < 0){
56     perror("bind error");
57     close(serv_orig_sock);
58     return 3;
59   }
60   if (listen(serv_orig_sock, 5) < 0 ) {               // LISTEN
61     perror("listen error");
62     close (serv_orig_sock);
63     return 4;
64   }
65                 // aus p10.2.cxx
66                 // uname -a
67         pipefile = popen( "uname -a", "r" );
68         memset(uname_buf, 0x0, UNAME_SIZE);
69         read(fileno(pipefile), uname_buf, UNAME_SIZE);
70         cout << uname_buf << endl;
71
72   do {
73     clnt_len = sizeof(clnt_adr);                  // ACCEPT a
         connect
74     if ((serv_new_sock = accept( serv_orig_sock, (struct
         sockaddr *) &clnt_adr,
75                                  &clnt_len)) < 0) {
76       perror("accept error");
77       close(serv_orig_sock);
```

```
78        return 5;
79      }
80      if ( fork( ) == 0 ) {                                   // Generate
            a CHILD
81        write(serv_new_sock, uname_buf, UNAME_SIZE); // Write
              uname -a to client
82        while ( (len=read(serv_new_sock, buf, BUFSIZ)) > 0 ){
83
84
85  //
      ***********************************************************************
86  // ************************ ACTIVE
      ************************************
87  //
      ***********************************************************************
88
89                          if ( strncmp(COM_ACTIVE, buf,3) == 0 ) {
                                // ACTIVE
90                                  strcpy(port_buf,strstr(buf,"␣
                                      "));
91                                  clnt_data_port =
                                      ntohs(atoi(port_buf));
92                                  cout << "Received:␣"<<buf<<endl;
93
94                                  sprintf(buf,"%s", COM_OK);
95                                  cout << "Sending:␣"<<buf<<endl;
96                      write(serv_new_sock, buf, BUFSIZ);
97
98                                  cout << "Connecting␣to␣port␣
                                      "<<clnt_data_port<<"␣
                                      ..."<<endl;
99
100                                 clnt_adr.sin_port    = htons(
                                      clnt_data_port );
                                      // Use the given port
101
102                                     // SOCKET
103                                 if ((clnt_data_sock =
                                      socket(PF_INET, SOCK_STREAM,
                                      0)) < 0) {
104                                 perror("generate␣error");
105                                 return 6;
106                                 }
107                                                                 //
                                                                      CON
```

```
108                                 if (connect(
                                        clnt_data_sock,(struct
                                        sockaddr *)&clnt_adr,
109                     sizeof(clnt_adr)) < 0) {
110                                 perror("connect error");
111                                 return 7;
112                                 }
113                                 break;
114                             }//active
115
116 //
    ***********************************************************************
117 // ************************ PASSIVE
    *********************************
118 //
    ***********************************************************************
119
120                             if ( strncmp(COM_PASSIVE,buf,3) == 0 ) {
                                    // PASSIVE
121                                 cout << "Received: "<<buf<<endl;
122
123                                 if ((clnt_data_sock =
                                        socket(PF_INET, SOCK_STREAM,
                                        0)) < 0) {
124                                 perror("generate error");
125                             return 8;
126                                 }
127                                 memset( &clnt_data_adr, 0,
                                        sizeof(clnt_data_adr) );
                                        // Clear structure
128                                 clnt_data_adr.sin_family        =
                                        AF_INET;                //
                                        Set address type
129                                 clnt_data_adr.sin_addr.s_addr =
                                        htonl(INADDR_ANY);      //
                                        Any interface
130                                 clnt_data_adr.sin_port          =
                                        0;
                                                        // Use our
                                        fake port
131                                                     // BIND
132                                 if (bind(clnt_data_sock, (struct
                                         sockaddr *) &clnt_data_adr,
133                     sizeof(clnt_data_adr)) < 0){
134                                 perror("bind error");
135                                 close(clnt_data_sock);
136                                 return 9;
137                                 }
```

```
138                                 if (listen(clnt_data_sock, 5) <
                                        0 ) {                  // LISTEN
139                                 perror("listen error");
140                                 close (clnt_data_sock);
141                                 return 10;
142                                 }
143                                 //port ausgeben
144
145                                 info_len=sizeof(info_adr);
146                                 memset(&info_adr, 0, info_len);
147
148                                 if (getsockname(clnt_data_sock,
                                        (struct  sockaddr *)
                                       &info_adr, &info_len) < 0
149                             || (host_info=gethostent()) == (struct
                                    hostent*) NULL) {
150                             perror("getsockname || gethostent
                                    error");
151                             close(clnt_data_sock);
152                             exit(9);
153                             }
154
155                 sprintf(buf,"%s %d", COMLOK, info_adr.sin_port);
156                                 cout << "Sending: "<<buf<<endl;
157                 write(serv_new_sock, buf, BUFSIZ);
158
159                                 cout <<"Opening port
                                        "<<ntohs(info_adr.sin_port)<<"
                                        ..."<<endl;
160                                 clnt_data_len =
                                        sizeof(clnt_data_adr);
                                                    // ACCEPT a
                                        connect
161                             if ((clnt_data_sock = accept(
                                    clnt_data_sock, (struct sockaddr *)
                                   &clnt_data_adr,
162                                    &clnt_data_len)) < 0) {
163                             perror("accept error");
164                             close(clnt_data_sock);
165                             return 11;
166                                 }
167                                 break;
168                             }//passive
169                 }//while
170
171                 memset(&buf,0,BUFSIZ);
172
173         while ( (len=read(serv_new_sock, buf, BUFSIZ)) > 0 ){
174
```

```
175         write(serv_new_sock, COMLOK, strlen(COMLOK));
176
177         for (i=0; i < len; ++i)                    // Change the
                case
178           buf[i] = toupper(buf[i]);
179         cout << buf;
180         write(clnt_data_sock, buf, len);              // Write
                back to socket
181         if ( buf[0] == '.' ) break;                // Are we done
                yet?
182         memset(&buf,0,BUFSIZ);
183               }
184       close(serv_new_sock);                          // In
            CHILD process
185       return 0;
186     } else //if fork()
187       close(serv_new_sock);                          // In
            PARENT process
188   } while( true );                                 // FOREVER
189   return 0;
190 }//main
191
192
193 void
194 signal_catcher(int the_sig){
195   signal(the_sig, signal_catcher);                 // reset
196   wait(0);                                         // keep the
        zombies at bay
197 }
```

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

20. Februar 2008

Interprozeßkommunikation
Schlusstest

Allgemeines

Grundlage des Schlußtests sind die Programme *tee.cpp*, *udp_client.cpp* und *tcp_client.cpp* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel sind erlaubt: Das Skript, Bücher, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)
- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL
VERSITÄT

20. Februar 2008

Aufgabe

Das im Verzeichnis */home/IPC/klausur* liegende ***tee***-Programm liest von stdin und schreibt sowohl auf stdout als auch in eine Datei.

- Erweitern Sie das Programm ***tee*** um die Fähigkeit die Ausgabe zusätzlich noch via TCP ausgeben zu können.
- Vervollständigen Sie den rudimentären Client *tcp_client.cpp* so, dass er die entsprechende Ausgabe empfangen und auf stdout ausgeben kann. Dabei soll er genau eine TCP-Verbindung entgegen nehmen und sich, nachdem das ***tee***-Programm die Verbindung geschlossen hat, selbstständig beenden.
- Erweitern Sie das Programm ***tee*** um die Fähigkeit die Ausgabe zusätzlich noch via UDP ausgeben zu können.
- Vervollständigen Sie den rudimentären Client *udp_client.cpp* so, dass er die entsprechende Ausgabe empfangen und auf stdout ausgeben kann. Dabei soll der Client beliebig viele UDP-Pakete entgegen nehmen und sich mittels STRG-C beenden lassen.

Beachten Sie dabei folgende Hinweise:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie die bereits in den Dateien definierten Variablen.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Verwenden Sie als Compiler ***g++***.
- Sie können die Aufgaben entweder lokal auf dem Poolrechner bearbeiten und dann auf rapunzel kompilieren und testen. Dabei können Sie die Dateien z.B. mit ***scp login@rapunzel:/pfad/datei /lokalerPfad/*** bzw. ***scp /lokalerPfad/lokaleDatei login@rapunzel:/pfad*** kopieren.
  Oder Sie können die Dateien remote auf rapunzel bearbeiten. z.B. login via ***ssh -X login@rapunzel*** und dann einen Editor wie z.B. mped (grafisch) oder mcedit, vi (konsolenbasiert) aufrufen.
  Beachten Sie jedoch, dass in beiden Fällen die Programme auf rapunzel kompilierbar und lauffähig sein müssen.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

20. Februar 2008

Interprozeßkommunikation
Lösungsvorschlag - Schlußtest

Allgemeines

Grundlage des Schlußtests sind die Programme *tee.cpp*, *udp_client.cpp* und *tcp_client.cpp* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel erlaubt sind: Das Skript, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)
- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL
VERSITÄT

20. Februar 2008

Aufgabe

Das im Verzeichnis */home/IPC/klausur* liegende ***tee***-Programm liest von stdin und schreibt sowohl auf stdout als auch in eine Datei.

- Erweitern Sie das Programm ***tee*** um die Fähigkeit die Ausgabe zusätzlich noch via TCP ausgeben zu können.
- Vervollständigen Sie den rudimentären Client *tcp_client.cpp* so, dass er die entsprechende Ausgabe empfangen und auf stdout ausgeben kann. Dabei soll er genau eine TCP-Verbindung entgegen nehmen und sich, nachdem das ***tee***-Programm die Verbindung geschlossen hat, selbstständig beenden.
- Erweitern Sie das Programm ***tee*** um die Fähigkeit die Ausgabe zusätzlich noch via UDP ausgeben zu können.
- Vervollständigen Sie den rudimentären Client *udp_client.cpp* so, dass er die entsprechende Ausgabe empfangen und auf stdout ausgeben kann. Dabei soll der Client beliebig viele UDP-Pakete entgegen nehmen und sich mittels STRG-C beenden lassen.

Beachten Sie dabei folgende Hinweise:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie die bereits in den Dateien definierten Variablen.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Verwenden Sie als Compiler ***g++***.
- Sie können die Aufgaben entweder lokal auf dem Poolrechner bearbeiten und dann auf rapunzel kompilieren und testen. Dabei können Sie die Dateien z.B. mit ***scp login@rapunzel:/pfad/datei /lokalerPfad/*** bzw. ***scp /lokalerPfad/lokaleDatei login@rapunzel:/pfad*** kopieren.
  Oder Sie können die Dateien remote auf rapunzel bearbeiten. z.B. login via ***ssh -X login@rapunzel*** und dann einen Editor wie z.B. mped (grafisch) oder mcedit, vi (konsolenbasiert) aufrufen.
  Beachten Sie jedoch, dass in beiden Fällen die Programme auf rapunzel kompilierbar und lauffähig sein müssen.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

20. Februar 2008

Listing 1: tee.cpp

```
1  /* System includes */
2
3  #include <netdb.h>
4  #include <iostream>
5  #define BUF 1024
6
7
8  void usage()
9  {
10     /* Display a usage message */
11     printf("%s: Split output into two or more streams.\n\n",
           "tee");
12
13     printf("usage: %s [-f file] [-t host port] [-u host port]
          <input\n\n", "tee");
14     printf("Options:\n");
15     printf("    -f      Append output to file.\n");
16     printf("    -t      Send output via tcp to host:port.\n");
17     printf("    -u      Send output via udp to host:port.\n");
18 }
19
20 int main(int argc, char *argv[])
21 {
22     int                 argcounter=1;           /* argv
          counter*/
23         char buffer[BUF]; /* "the buffer" */
24
25         const char *filename=NULL; FILE *fp;  /* filedata */
26
27         const char *tcphost=NULL; int tcpport; /* tcpdata */
28         int tcpsock;
29     struct sockaddr_in tcpaddr;
30     struct hostent  *tcphostent;
31
32         const char *udphost=NULL; int udpport; /* udpdata */
33         int  udpsock;
34         struct sockaddr_in udpaddr;
35     struct hostent *udphostent;
36
37
38
39
40 // ***** PARAMETER *****
41
42         if (argc<3) {usage();return EXIT_FAILURE;}
43     for (int argcounter=1; argcounter<argc; ++argcounter) {
44
45                         // file_parameters
```

```
46                          if (strcmp(argv[argcounter],"-f")==0)
47                                  if (argcounter+1<argc)
48                                                  {filename=argv[++argcounter
49                                          else
50                                                  {usage();return
                                                        EXIT_FAILURE;}
51
52                          // tcp_parameters
53                          if (strcmp(argv[argcounter],"-t")==0)
54                                  if (argcounter+2<argc)
55                                                  {tcphost
                                                        =
                                                        argv[++argcounter];tcpp
56                                          else
57                                                  {usage();return
                                                        EXIT_FAILURE;}
58
59                          // udp_parameters
60                          if (strcmp(argv[argcounter],"-u")==0)
61                                  if (argcounter+2<argc)
62                                                  {udphost
                                                        =
                                                        argv[++argcounter];udpp
63                                          else
64                                                  {usage();return
                                                        EXIT_FAILURE;}
65          }
66
67  // ***** INITIALIZE *****
68
69          if (filename!= NULL) {
70
71                          fp = fopen(filename, "w");
72                          if (fp == NULL)
73                                  {
74                                  fprintf(stderr, "%s: Can't
                                        write: %s\n",argv[0],
                                        filename);
75                                  return EXIT_FAILURE;
76                                  }
77
78
79          } // if filename
80
81          if (tcphost!= NULL) {
82                  tcphostent = gethostbyname(tcphost);        //
```

```
83                    Obtain host (server) info
                      if (tcphostent == (struct hostent *)
                          NULL ) {
84                    perror("gethostbyname ");
85                  return EXIT_FAILURE;
86                    }
87
88                  memset(&tcpaddr, 0, sizeof( tcpaddr));        //
                        Clear structure
89                  tcpaddr.sin_family = AF_INET;
                        // Set address type
90                  memcpy(&tcpaddr.sin_addr, tcphostent->h_addr,
                        tcphostent->h_length);
91                  tcpaddr.sin_port    = htons(tcpport);
92                                      // SOCKET
93                  if ((tcpsock = socket(PF_INET, SOCK_STREAM, 0))
                        < 0) {
94                  perror("generate error");
95                  return EXIT_FAILURE;
96                  }                                          // CONNECT
97
98                  if (connect(tcpsock,(struct sockaddr *)&tcpaddr,
99                sizeof(tcpaddr)) < 0) {
100                 perror("connect error");
101                 return EXIT_FAILURE;
102                 }
103
104         } // if tcphost
105
106         if (udphost != NULL) {
107
108                 if (!(udphostent=gethostbyname(udphost))){
109                             perror("CLIENT gethostname ");
110                             return EXIT_FAILURE;
111                   }
112
113                   if ((udpsock = socket(PF_INET,
                          SOCK_DGRAM, 0)) < 0) {
114                   perror("SERVER socket ");
115                   return EXIT_FAILURE;
116                   }
117                   memset(&udpaddr, 0, sizeof(udpaddr));
                          // Clear structure
118                   udpaddr.sin_family      = AF_INET;
                          // Set address type
119         memcpy(&udpaddr.sin_addr, udphostent->h_addr,
                 udphostent->h_length); // Set host
120                   udpaddr.sin_port        =
                          htons(udpport); // Set port
```

```
121
122             } // if udphost
123
124 // ***** STREAMING-LOOP *****
125
126     /* Split the input to several streams */
127     while(true)
128     {
129                     fgets(buffer, BUF, stdin);
130
131         fputs(buffer, stdout);
132         fflush(stdout);
133
134         if (filename != NULL) {
135                         fputs(buffer, fp);
136             fflush(fp);
137         } // if filename
138
139                     if (tcphost!= NULL) {
140                 write(tcpsock, buffer, strlen(buffer));
141                     } // if tcphost
142
143                     if (udphost != NULL) {
144
145                                 if (sendto(     udpsock, buffer,
                                        strlen(buffer), 0,
146                                                 (struct sockaddr
                                                     *)
                                                    &udpaddr,
                                                    sizeof(udpaddr))
                                                    <0 )
147             {
148                 perror("SERVER sendto ");
149                 return EXIT_FAILURE;
150             }
151
152                     } // if udphost
153
154
155                     memset(&buffer, 0, sizeof(buffer));  // Clear
                            buffer
156                     if (feof(stdin)) break; // EOF
157     }
158
159 // ***** CLEANUP *****
160             if (filename!= NULL) {
161             if (fp == NULL) fclose(fp);
162             } //if filename
163
```

```
164             if (tcphost!= NULL) {
165                     close(tcpsock);
166             } // if tcphost
167
168             if (udphost != NULL) {
169                     close(udpsock);
170             } // if udphost
171
172     fflush(stdout);
173     return EXIT_SUCCESS;
174
175 }
```

Listing 2: tcp_client.cpp

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <arpa/inet.h>
5  #include <sys/un.h>
6
7  #define BUF 1024
8
9  int
10 main(int argc, char *argv[]){
11   int              orig_sock, new_sock, len;
12   socklen_t        client_len;
13   struct sockaddr_in client;
14
15   char buffer[BUF];
16
17   // ***** PARAMETER *****
18
19   if ( argc < 2 ) {                    // We need port #
20     printf("usage: %s port\n",argv[0]);
21     return EXIT_FAILURE;
22   }
23   // ***** INITIALIZE TCP *****
24
25   if ((orig_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
26     perror("generate error");
27     return EXIT_FAILURE;
28   }
29                                        // Set client address
                                             info
30   memset(&client, 0, sizeof(client));  // Clear structure
31   client.sin_family      = AF_INET;    // Address type
32   client.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
33    client.sin_port         = htons(atoi(argv[1]));
34                                          // BIND
35    if (bind( orig_sock, (struct sockaddr *) &client,
36              sizeof(client)) < 0){
37      perror("bind error");
38      close(orig_sock);
39      return EXIT_FAILURE;
40    }
41
42    if (listen(orig_sock, 5) < 0 ) {                // LISTEN
43      perror("listen error");
44      close (orig_sock);
45      return EXIT_FAILURE;
46    }
47
48    client_len = sizeof(client);                  // ACCEPT a
        connect
49    if ((new_sock = accept( orig_sock, (struct sockaddr *)
        &client,
50                            &client_len)) < 0) {
51          perror("accept error");
52          close(orig_sock);
53          return EXIT_FAILURE;
54    }
55
56    // ***** RECEIVE *****
57    memset(buffer,0,BUF);
58    while ( (len=read(new_sock, buffer, BUF)) > 0 ){  //get
        message
59                // ***** OUTPUT *****
60             fputs(buffer, stdout);
61          fflush(stdout);
62          memset(buffer,0,BUF); // clear the buffer
63    } // while
64
65    close(new_sock);
66    close(orig_sock);
67    return EXIT_SUCCESS;
68
69  } //main
```

Listing 3: udp_client.cpp

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <arpa/inet.h>
5  #include <sys/un.h>
```

```
6
7  #define BUF 1024
8
9  int
10 main(int argc, char *argv[]){
11   int              sock, n;
12   socklen_t        server_len;
13   struct sockaddr_in client,server;
14
15   char buffer[BUF];
16
17   // ***** PARAMETER *****
18
19   if ( argc < 2 ) {                    // We need server name &
         port #
20     printf("usage: %s port\n",argv[0]);
21     return EXIT_FAILURE;
22   }
23   // ***** INITIALIZE *****
24   if ((sock=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) {
25     perror("CLIENT socket ");
26     return EXIT_FAILURE;
27   }
28                                        // Set client address
                                             info
29   memset(&client, 0, sizeof(client));  // Clear structure
30   client.sin_family      = AF_INET;    // Address type
31   client.sin_addr.s_addr = htonl(INADDR_ANY);
32   client.sin_port        = htons(atoi(argv[1]));
33                                        // BIND
34   if (bind(sock, (struct sockaddr *) &client,sizeof(client)) <
         0) {
35     perror("CLIENT bind ");
36     return EXIT_FAILURE;
37   }
38
39   server_len=sizeof(server);
40
41   // ***** LOOP *****
42   while( 1 ) {
43                 memset(buffer,0,BUF); // clear the buffer
44
45                 // ***** RECEIVE *****
46       if (n=recvfrom(sock, buffer, BUF, 0,              //get
             message
47                      (struct sockaddr *) &server, &server_len)
                           < 0) {
48                        perror("CLIENT recvfrom ");
49                        close(sock);
```

```
50                          return EXIT_FAILURE;
51          }
52
53          // ***** OUTPUT *****
54          fputs(buffer, stdout);
55          fflush(stdout);
56
57      } //while
58
59      close(sock);
60      return EXIT_SUCCESS;
61
62  } //main
```

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

30. Juli 2009

Interprozeßkommunikation
Schlusstest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp* und *client* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel sind erlaubt: Das Skript, Bücher, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)
- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".

Beachten Sie dabei folgende Hinweise:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie die bereits in den Dateien definierten Variablen.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Verwenden Sie als Compiler ***g++*** oder ***gcc***. Falls Sie lieber den ***gcc*** verwenden möchten, so müssen Sie für das Startprogramm noch mittels "-lstdc++" die entsprechende Library einbinden.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

30. Juli 2009

Aufgabe

Das im Verzeichnis */home/IPC/klausur* liegende ***server***-Programm nimmt Pakete über eine verbindungslose Verbindung entgegen und sendet dessen Inhalt als in Großbuchstaben gewandelten Text zurück.
Das im Verzeichnis */home/IPC/klausur* liegende ***monitor***-Programm nimmt Pakete über eine verbindungslose Verbindung entgegen und gibt dessen Inhalt als in Kleinbuchstaben gewandelten Text auf der Standardausgabe aus.
Bearbeiten Sie nun nacheinander die folgenden Aufgaben. Stellen Sie dabei sicher, dass Sie nach dem Lösen einer Teilaufgabe den Quellcode entsprechend sichern bevor Sie weiter fortfahren, so dass alle Lösungen der Teilaufgaben am Ende noch verfügbar sind.

- Der Server erhält den Port, auf dem er lauschen soll via Argumentenübergabe. Ändern Sie den Server so ab, dass er eine freie, zufällige Portnummer wählt und anschließend ausgibt.
- Ändern Sie nun den Server so ab, dass er seine Antwort nicht nur zum Client zurücksendet, sondern auch noch eine Kopie seiner Antwort zu einem beliebigen Rechner/Port (dem Monitor) sendet, den er auf der Kommandozeile angegeben bekommt. (siehe Abbildung 1)

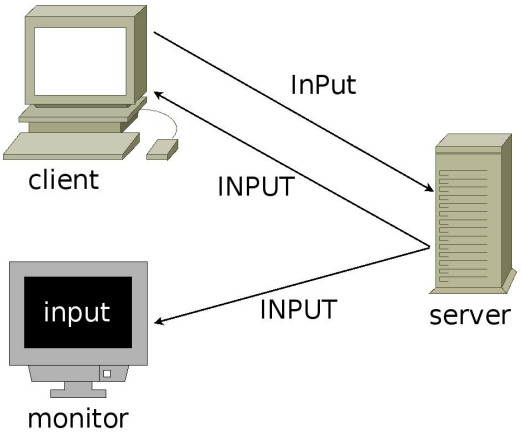

Abbildung 1: Senden zum Monitor

- Nun soll die Ausgabe des Servers verändert werden. Anstelle der trivialen Wandlung in Großbuchstaben soll die *Ausgabe der Kopie zum Monitor* verschlüsselt erfolgen. Benutzen Sie dafür die Ausgabe des Unix-Scripts ***aes_encrypt_mime***.
  Passen Sie dazu auch das ***monitor***-Programm so an, dass der empfangene String durch das Unix-Script ***mime_aes_decrypt*** entschlüsselt und dann im Klartext angezeigt wird.
  Sie finden beide Scripte in ***/usr/local/bin***. Benutzen Sie das File */home/IPC/klausur/key* als Schlüssel. Beispiel: ***aes_encrypt_mime /home/IPC/klausur/key***.
  Sie dürfen ausserdem davon ausgehen, daß der im Client eingegebene String nur aus Buchstaben und Zahlen besteht. (siehe Abbildung 2).

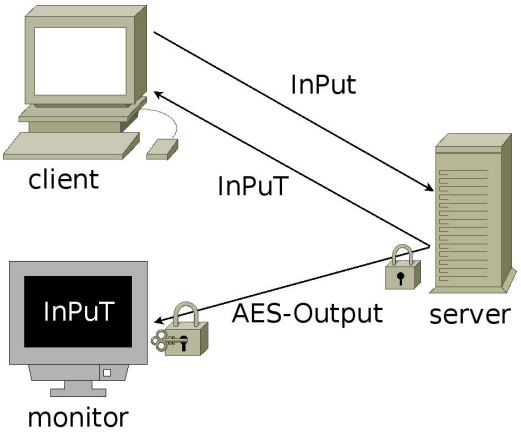

Abbildung 2: Verschlüsselung zum Monitor

- Ändern Sie nun das Programm so ab, dass die *eingehende Verbindung* des Clients durch ein verbindungsorientiertes Netzwerkprotokoll erfolgt.

Um ihren Programmcode zu testen und den Client zu simulieren können Sie auf das Programm ***netcat*** *zurückgreifen.* z.B.:

nc hostname port	Verbindet via TCP zum *port* von *hostname*.
nc -u hostname port	Verbindet via UDP zum *port* von *hostname*.
nc -l -p port	Lauscht via TCP auf *port*.
nc -u -l -p port	Lauscht via UDP auf *port*.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

30. Juli 2009

Interprozeßkommunikation
Lösungsvorschlag - Schlußtest

Allgemeines

Grundlage des Schlußtests sind die Programme *server.cpp* und *client* im Verzeichnis */home/IPC/klausur*.
Als Hilfsmittel sind erlaubt: Das Skript, Bücher, eigene Aufzeichnungen, sämtliche Übungen einschließlich Lösungsvorschlägen und Beispielprogrammen in */home/IPC/* sowie alle Dokumentation, die auf rapunzel in Form von Manpages, Infopages, selbst bearbeiteten Aufgaben, etc. vorhanden ist.
Jegliche Kommunikationsversuche - unabhängig davon, in welcher Form sie stattfinden - führen zum Ausschluss von der Prüfung.
Um die nachfolgenden Aufgaben zu bearbeiten, führen Sie deshalb zunächst die folgenden Schritte durch:

- Setzen Sie die Rechte Ihres Homeverzeichnisses auf 700.
  (***chmod 700 ~***)
- Legen Sie in Ihrem Homeverzeichnis ein Unterverzeichnis "klausur" an.
  (***mkdir ~/klausur***)
- Bearbeiten Sie die Aufgaben im Verzeichnis "~/klausur".

Beachten Sie dabei folgende Hinweise:

- Kommentieren Sie die notwendigen Änderungen *knapp* und *aussagekräftig* im Sourcecode.
- Benutzen Sie die bereits in den Dateien definierten Variablen.
- Beachten Sie die Kommentare im Quelltext, die Ihnen beim Aufbau des Programms helfen sollen.
- Verwenden Sie als Compiler ***g++*** oder ***gcc***. Falls Sie lieber den ***gcc*** verwenden möchten, so müssen Sie für das Startprogramm noch mittels "-lstdc++" die entsprechende Library einbinden.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

30. Juli 2009

Aufgabe

Das im Verzeichnis */home/IPC/klausur* liegende ***server***-Programm nimmt Pakete über eine verbindungslose Verbindung entgegen und sendet dessen Inhalt als in Großbuchstaben gewandelten Text zurück.
Das im Verzeichnis */home/IPC/klausur* liegende ***monitor***-Programm nimmt Pakete über eine verbindungslose Verbindung entgegen und gibt dessen Inhalt als in Kleinbuchstaben gewandelten Text auf der Standardausgabe aus.
Bearbeiten Sie nun nacheinander die folgenden Aufgaben. Stellen Sie dabei sicher, dass Sie nach dem Lösen einer Teilaufgabe den Quellcode entsprechend sichern bevor Sie weiter fortfahren, so dass alle Lösungen der Teilaufgaben am Ende noch verfügbar sind.

- Der Server erhält den Port, auf dem er lauschen soll via Argumentenübergabe. Ändern Sie den Server so ab, dass er eine freie, zufällige Portnummer wählt und anschließend ausgibt.
- Ändern Sie nun den Server so ab, dass er seine Antwort nicht nur zum Client zurücksendet, sondern auch noch eine Kopie seiner Antwort zu einem beliebigen Rechner/Port (dem Monitor) sendet, den er auf der Kommandozeile angegeben bekommt. (siehe Abbildung 1)

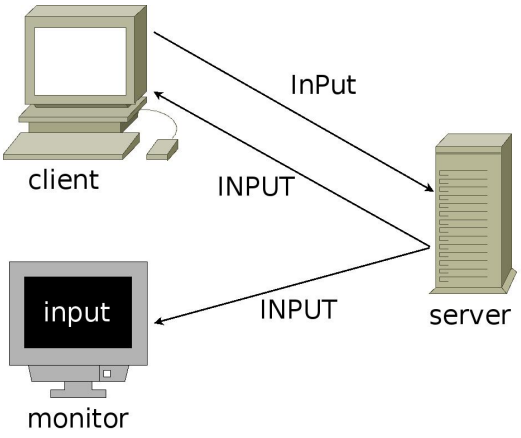

Abbildung 1: Senden zum Monitor

- Nun soll die Ausgabe des Servers verändert werden. Anstelle der trivialen Wandlung in Großbuchstaben soll die *Ausgabe der Kopie zum Monitor* verschlüsselt erfolgen. Benutzen Sie dafür die Ausgabe des Unix-Scripts ***aes_encrypt_mime***.
  Passen Sie dazu auch das ***monitor***-Programm so an, dass der empfangene String durch das Unix-Script ***mime_aes_decrypt*** entschlüsselt und dann im Klartext angezeigt wird.
  Sie finden beide Scripte in ***/usr/local/bin***. Benutzen Sie das File */home/IPC/klausur/key* als Schlüssel. Beispiel: ***aes_encrypt_mime /home/IPC/klausur/key***.
  Sie dürfen ausserdem davon ausgehen, daß der im Client eingegebene String nur aus Buchstaben und Zahlen besteht. (siehe Abbildung 2).

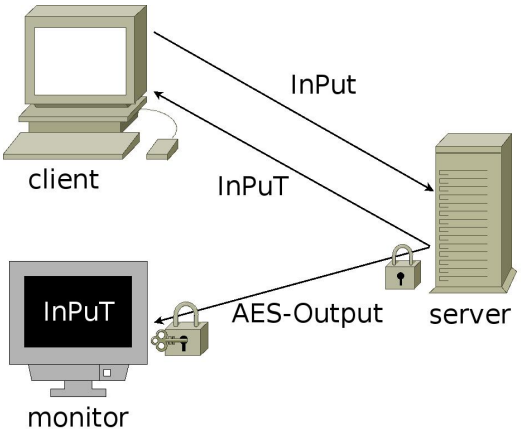

Abbildung 2: Verschlüsselung zum Monitor

- Ändern Sie nun das Programm so ab, dass die *eingehende Verbindung* des Clients durch ein verbindungsorientiertes Netzwerkprotokoll erfolgt.

Um ihren Programmcode zu testen und den Client zu simulieren können Sie auf das Programm ***netcat*** *zurückgreifen.* z.B.:

nc hostname port	Verbindet via TCP zum *port* von *hostname*.
nc -u hostname port	Verbindet via UDP zum *port* von *hostname*.
nc -l -p port	Lauscht via TCP auf *port*.
nc -u -l -p port	Lauscht via UDP auf *port*.

Fachbereich Elektrotechnik / Informatik
FG Datenbanken / Interaktive Systeme
Prof. Dr. Lutz Wegner
Sebastian Pape

UNIKASSEL VERSITÄT

30. Juli 2009

Listing 1: final_monitor.cpp

```
1  #include "local_sock.h"
2
3  int
4  main(int argc, char *argv[]) {
5          void    process_msg(char *, int);
6
7  // Listening
8    struct sockaddr_in inputsockaddr;           // Internet
       addresses
9    socklen_t       input_len;
10   int               inputsock;
11
12 // Client
13   struct sockaddr_in client;
                // Internet addresses
14   socklen_t          client_len;
15
16 // MISC
17   int n;
18   FILE *fin;
19   static char buf[BUFSIZ];                  // Buffer for messages
20   static char aesbuf[BUFSIZ];             // Buffer for
       rot13-messages
21
22 // ******************
23 // **** INITIALIZE ***
24 // ******************
25
26   if ( argc < 2 ) {                       // We need server name &
       port #
27     cerr << "usage: " << argv[0] << " port_#" << endl;
28     return 1;
29   }
30
31 // Listening
32
33   if ((inputsock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
34     perror("SERVER socket "); return 4;
35   }
36   memset(&inputsockaddr, 0, sizeof(inputsockaddr));  // Clear
       structure
37   inputsockaddr.sin_family      = AF_INET;          // Set
       address type
38   inputsockaddr.sin_addr.s_addr = htonl(INADDR_ANY);
39   inputsockaddr.sin_port        = htons(atoi(argv[1]));
40                                                    //
                                                          BIND
41   if (bind(inputsock, (struct sockaddr *) &inputsockaddr,
```

```
42        sizeof(inputsockaddr) ) < 0) {
43      perror("SERVER bind ");  return 5;
44    }
45    input_len = sizeof(inputsockaddr);          // Obtain address
          length
46                                                // Find picked port
                                                      #
47    if (getsockname(inputsock, (struct sockaddr *) &inputsockaddr,
48        &input_len) < 0) {
49      perror("SERVER getsocketname ");  return 6;
50    }
51    cout << "Server using port " << ntohs(inputsockaddr.sin_port)
          << endl;
52
53
54  //
55  // **** LOOP ****
56  //
57
58    while ( 1 ) {                                 // Loop forever
59      client_len = sizeof(client);                // set the
            length
60      memset(buf, 0, BUFSIZ);                     // clear the
            buffer
61      if ((n=recvfrom(inputsock, buf, BUFSIZ, 0,  // get the
            client's msg
62          (struct sockaddr *) &client, &client_len)) < 0){
63       perror("SERVER recvfrom ");
64       close(inputsock); return 7;
65      }
66
67  //     process_msg(buf,strlen(buf));
68
69      /* AES */
70      memset(aesbuf, 0x0, BUFSIZ);
71      sprintf(aesbuf, "echo \"%s\" | mime_aes_decrypt
            /home/IPC/klausur/key ", buf);
72      fin = popen( aesbuf, "r" );
73      memset(aesbuf, 0x0, BUFSIZ);
74      read(fileno(fin), aesbuf, BUFSIZ);
75
76      /* OUTPUT */
77  //     write(fileno(stdout), buf, strlen(buf));          //
        display msg on server
78      write(fileno(stdout), aesbuf, strlen(aesbuf));    //
            display msg on server
79    }
80    return 0;
81  }
```

```
82
83 /*
84    Convert upper case alphabetics to lower case
85 */
86 void
87 process_msg(char *b, int len){
88   for (int i = 0; i < len; ++i)
89     if (isalpha(*(b + i)))
90       *(b + i) = tolower(*(b + i));
91 }
```

Listing 2: final_udp_server.cpp

```
1  #include "local_sock.h"
2
3  int
4  main(int argc, char *argv[]) {
5          void     process_msg(char *, int);
6
7  // Sending
8    struct hostent *outputhost;                // For host
       information
9    struct sockaddr_in outputsockaddr;         // Internet
       addresses
10   socklen_t        output_len;
11   int outputsock;
12
13 // Listening
14   struct sockaddr_in inputsockaddr;           // Internet
       addresses
15   socklen_t        input_len;
16   int                 inputsock;
17
18 // Client
19   struct sockaddr_in client;
               // Internet addresses
20   socklen_t           client_len;
21
22 // MISC
23   int n;
24   FILE *fin;
25   static char buf[BUFSIZ];                  // Buffer for messages
26   static char aesbuf[BUFSIZ];             // Buffer for
       AES-messages
27
28 // ******************
29 // **** INITIALIZE ***
30 // ******************
```

```
31
32  // Sending
33
34    if ( argc < 3 ) {                       // We need monitor name &
          port #
35      cerr << "usage: " << argv[0] << "monitor_name   port_#" <<
            endl;
36      return 1;
37    }                                       // Server information
38    if (!(outputhost=gethostbyname(argv[1]))){
39      perror("CLIENT gethostname ");  return 2;
40    }                                       // Set server address
          info
41    memset(&outputsockaddr, 0, sizeof(outputsockaddr));  // Clear
          structure
42    outputsockaddr.sin_family = AF_INET;          // Address type
43    memcpy(&outputsockaddr.sin_addr, outputhost->h_addr,
          outputhost->h_length);
44    outputsockaddr.sin_port   = htons(atoi(argv[2]));
45                                            // SOCKET
46    if ((outputsock=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) {
47      perror("CLIENT socket "); return 3;
48    }                                       // Set client address
          info
49    output_len=sizeof(outputsockaddr);                    // length
          of address
50
51  // Listening
52
53    if ((inputsock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
54      perror("SERVER socket "); return 4;
55    }
56    memset(&inputsockaddr, 0, sizeof(inputsockaddr));  // Clear
          structure
57    inputsockaddr.sin_family      = AF_INET;          // Set
          address type
58    inputsockaddr.sin_addr.s_addr = htonl(INADDR_ANY);
59    inputsockaddr.sin_port        = htons(0);
60                                                                //
                                                                    BIND
61    if (bind(inputsock, (struct sockaddr *) &inputsockaddr,
62        sizeof(inputsockaddr) ) < 0) {
63      perror("SERVER bind "); return 5;
64    }
65    input_len = sizeof(inputsockaddr);          // Obtain  address
          length
66                                                // Find picked port
                                                      #
67    if (getsockname(inputsock, (struct sockaddr *) &inputsockaddr,
```

```
68        &input_len) < 0) {
69      perror("SERVER getsocketname ");  return 6;
70    }
71    cout << "Server using port " << ntohs(inputsockaddr.sin_port)
         << endl;
72
73
74  //
75  // **** LOOP ****
76  //
77
78    while ( 1 ) {                                   // Loop forever
79      client_len = sizeof(client);                  // set the
            length
80      memset(buf, 0, BUFSIZ);                       // clear the
            buffer
81      if ((n=recvfrom(inputsock, buf, BUFSIZ, 0,    // get the
            client's msg
82          (struct sockaddr *) &client, &client_len)) < 0){
83       perror("SERVER recvfrom ");
84       close(inputsock); return 7;
85      }
86
87  //     process_msg(buf,strlen(buf));
88
89      /* AES */
90      memset(aesbuf, 0x0, BUFSIZ);
91      sprintf(aesbuf, "echo \"%s\" | aes_encrypt_mime
            /home/IPC/klausur/key", buf);
92      fin = popen( aesbuf, "r" );
93      memset(aesbuf, 0x0, BUFSIZ);
94      read(fileno(fin), aesbuf, BUFSIZ);
95      //memcpy(&buf, rot13buf, n);
96      /* OUTPUT */
97
98
99      write(fileno(stdout), buf, strlen(buf));                //
            display msg on server
100     write(fileno(stdout), aesbuf, strlen(aesbuf));
                        // display msg on server
101       if ((sendto(inputsock, buf, strlen(buf) ,0,      // send to
              client
102         (struct sockaddr *) &client, client_len)) <0){
103         perror("SERVER sendto ");
104         close(inputsock); return 8;
105       }
106       if ((sendto(outputsock, aesbuf, strlen(aesbuf) ,0,      //
              send to listener
107         (struct sockaddr *) &outputsockaddr, output_len)) <0){
```

```
108             perror("SERVER sendto ");
109             close(inputsock); return 9;
110           }
111       }
112       return 0;
113   }
114
115   /*
116       Convert lower case alphabetics to upper case
117   */
118   void
119   process_msg(char *b, int len){
120       for (int i = 0; i < len; ++i)
121           if (isalpha(*(b + i)))
122               *(b + i) = toupper(*(b + i));
123   }
```

Listing 3: final_tcp_server.cpp

```
1   #include "local_sock.h"
2
3   int
4   main(int argc, char *argv[]) {
5           void    process_msg(char *, int);
6
7   // Sending
8       struct hostent *outputhost;                 // For host
            information
9       struct sockaddr_in outputsockaddr;          // Internet
            addresses
10      socklen_t       output_len;
11      int outputsock;
12
13  // Listening
14      struct sockaddr_in inputsockaddr;           // Internet
            addresses
15      socklen_t       input_len;
16      int                 inputsock;
17
18  // Client
19      struct sockaddr_in client;
                    // Internet addresses
20      socklen_t           client_len;
21
22  // MISC
23      int n;
```

```
24    int new_sock;                                                    //
          New socket from connect
25    FILE *fin;
26    static char buf[BUFSIZ];                  // Buffer for messages
27    static char aesbuf[BUFSIZ];             // Buffer for
          rot13-messages
28
29  // ******************
30  // **** INITIALIZE ***
31  // ******************
32
33  // Sending
34
35    if ( argc < 3 ) {                       // We need monitor name &
           port #
36      cerr << "usage: " << argv[0] << "monitor_name   port_#" <<
            endl;
37      return 1;
38    }                                       // Server information
39    if (!(outputhost=gethostbyname(argv[1]))){
40      perror("CLIENT gethostname ");  return 2;
41    }                                       // Set server address
          info
42    memset(&outputsockaddr, 0, sizeof(outputsockaddr));  // Clear
          structure
43    outputsockaddr.sin_family = AF_INET;          // Address type
44    memcpy(&outputsockaddr.sin_addr, outputhost->h_addr,
          outputhost->h_length);
45    outputsockaddr.sin_port    = htons(atoi(argv[2]));
46                                            // SOCKET
47    if ((outputsock=socket(PF_INET, SOCK_DGRAM, 0)) < 0 ) {
48      perror("CLIENT socket "); return 3;
49    }                                       // Set client address
          info
50    output_len=sizeof(outputsockaddr);                    // length
           of address
51
52  // Listening
53
54  //  if ((inputsock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
55    if ((inputsock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
56      perror("SERVER socket "); return 4;
57    }
58    memset(&inputsockaddr, 0, sizeof(inputsockaddr));  // Clear
          structure
59    inputsockaddr.sin_family       = AF_INET;           // Set
          address type
60    inputsockaddr.sin_addr.s_addr = htonl(INADDR_ANY);
61    inputsockaddr.sin_port         = htons(0);
```

```
62                                                                    //
                                                                          BIND
63    if (bind(inputsock, (struct sockaddr *) &inputsockaddr,
64        sizeof(inputsockaddr) ) < 0) {
65      perror("SERVER bind ");  return 5;
66    }
67    input_len = sizeof(inputsockaddr);          // Obtain address
          length
68                                                // Find picked port
                                                      #
69    if (getsockname(inputsock, (struct sockaddr *) &inputsockaddr,
70        &input_len) < 0) {
71      perror("SERVER getsocketname ");  return 6;
72    }
73    cout << "Server using port " << ntohs(inputsockaddr.sin_port)
          << endl;
74
75    if (listen(inputsock, 5) < 0 ) {                // TCP-LISTEN
76      perror("listen error");
77      close (inputsock);
78      return 7;
79    }
80
81    do {
82      client_len = sizeof(client);                  // ACCEPT a
            connect
83      if ((new_sock = accept( inputsock, (struct sockaddr *)
            &client,
84                              &client_len)) < 0) {
85        perror("accept error");
86        close(inputsock);
87        return 5;
88      }
89
90
91  //
92  // **** LOOP ****
93  //
94
95      if ( fork( ) == 0 ) {                         // Generate a
            CHILD
96        while ( (n=read(new_sock, buf, BUFSIZ)) > 0 ){
97
98          write(fileno(stdout), buf, n);              // display
              msg on server
99
100                 /* AES */
101             memset(aesbuf, 0x0, BUFSIZ);
102             sprintf(aesbuf, "echo \"%s\" | aes_encrypt_mime
```

```
                    /home/IPC/klausur/key", buf);
103                 fin = popen( aesbuf, "r" );
104                 memset(aesbuf, 0x0, BUFSIZ);
105                 read(fileno(fin), aesbuf, BUFSIZ);
106                 //memcpy(&buf, aesbuf, n);
107                 /* OUTPUT */
108             write(new_sock, buf, n);                // Write back to
                    socket
109
110             if ((sendto(outputsock, aesbuf, strlen(aesbuf) ,0,
                    // send to listener
111               (struct sockaddr *) &outputsockaddr, output_len)) <0){
112               perror("SERVER sendto ");
113               close(inputsock); return 9;
114             }//if sendto
115
116              }//while n>0
117         }//if fork
118       } while ( true );
119     }
120     /*
121        Convert lower case alphabetics to upper case
122     */
123     void
124     process_msg(char *b, int len){
125       for (int i = 0; i < len; ++i)
126         if (isalpha(*(b + i)))
127           *(b + i) = toupper(*(b + i));
128     }
```