UE3 Aufgabenstellung: Frei wählbare Übung mit Schwerpunkt Socket-Verbindung.

UE3 Hangman:

Diese Übung implementiert das Hangman-Spiel mit einem Server an dem sich mehrere Clients anmelden können. In server.c werden die angemeldeten Clients verwaltet. Jeder neue Client wird auf der Konsole mit der zugewiesenen Port-Nummer angezeigt. Das WordCheck.c File implementiert den Spielablauf (auswerten der Eingabe sowie den Fortschritt des Spieles). Das zu erratende Wort wird vom WordCheck aus einer Liste gewählt. Client.c implementiert die Gameloop auf der User-Seite. Der Client muss sich mittels ./client localhost 3131 am Server anmelden.

WordCheck.c

```
/*-------------------------------------------------------        */
/* WordCheck.c                          090614          */
/* Robert Muehlberger                                    */
/*                                                       */
/* a simple console game                                 */
/* this file was created by Prof. Gerhard Jahn           */
/* basic file for socket lession                         */
/*-------------------------------------------------------        */
#include<errno.h>
#include<syslog.h>
#include<time.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<netinet/in.h>
#define WORDLEN                 30   /* needed for the buffer size */
#define INCOMPLETE              1    /* game is still running */
#define WON                     2    /* player has won! */
#define LOST                    3    /* player has lost! */

//int main(){
//              /* starts the server process locally (no network) */
//              /* 0 = stdin; 1 = stdout */
//   ServerProcess(0,1);
//}

/* ServerProcess plays Hangman with a single player */
/* IN: stream to read input from                    */
/* OUT: stream to stream to write output to         */
void ServerProcess (int in,  int out) {
   int          max_lives=10;                /* number of guesses we offer */
   char  *words []= {                        /* the words to be guessed */
      "home",
      "ubuntu",
      "zombie",
      "daemon"
   }; /* ServerProcess */

   char  part_word [WORDLEN],          /* contains the guessed part */
     *whole_word,                      /* contains the whole word */
     guess_word[WORDLEN],              /* contains the word to be guessed */
     hostname[WORDLEN],                /* hostname (not used...) */
     outbuff[WORDLEN];                 /* output buffer */
   int          lives,                 /* number of initial lives */
    word_len,                          /* length of the word to be guessed */
    game_status = INCOMPLETE,          /* initial status of the game */
    hits,                              /* number of hits */
    i;
   time_t  timer;
   struct tm *t;
   /* initialize  */
```

```
    lives = max_lives;
    /* pick up a random word */
    time (&timer);
    t = (struct tm *) malloc (sizeof (struct tm));
    t = localtime (&timer);
    whole_word = words [
                  (t->tm_sec + rand()) %
                  (sizeof (words) / sizeof (char*))
                ];
    word_len = strlen (whole_word);
    syslog (LOG_USER|LOG_INFO,
          "word server chooses word %s ",whole_word);

    /* initialize empty word */
    for (i=0; i<word_len; i++)
       part_word [i]='-';
    part_word [i]= '\0';

    /* output empty word */
    sprintf (outbuff, "%s  lives:%d \n", part_word, lives);
    write (out, outbuff, strlen (outbuff));

   /* do the game */
    while (game_status == INCOMPLETE) {
       while (read (in, guess_word, WORDLEN) < 0) {
          /* restart if interrupted by signal ??  */
          if (errno != EINTR)
             perror ("reading players guess");
          exit(1);
          printf("re-starting the read\n");
       }

     /* check for hits */
       hits = 0;
       for (i=0; i<word_len; i++) {
          if (guess_word[0] == whole_word [i]) {
             hits=1;
             part_word[i] = whole_word[i];
          } /* if */
       } /* for */

       /* check for end of game */
       if (!hits) {
          lives--;
          if (lives == 0) {
             game_status = LOST;
          } /* game is over */
       } /* lost one life */

       if (strcmp (part_word, whole_word) == 0) {
          /* he did it */
          game_status = WON;
          sprintf (outbuff, "You won!\n");
          write (out, outbuff, strlen(outbuff));
          return;
       } /* if */

       /* game over */
       else if (lives == 0) {
          game_status = LOST;
          strcpy (part_word, whole_word);
       } /* else */
       /* show word */
       sprintf (outbuff,  "%s  lives: %d \n",  part_word,lives);
       write (out, outbuff, strlen (outbuff));
       if (game_status == LOST) {
          sprintf (outbuff, "\nGame over.\n");
          write (out, outbuff, strlen (outbuff));
       } /* he looses */
    } /* game is incomplete */
} /* ServerProcess */
```

Server.c

```
/*-------------------------------------------------------*/
/* server.c                                 090614       */
/* Robert Muehlberger                                    */
/*                                                       */
/* Client connect to server via localhost:3131           */
/*-------------------------------------------------------*/
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define SERVERPORT 3131 // the port users will be connecting to
#define BACKLOG 10 // how many pending connections queue will hold
extern void ServerProcess(int in, int out);
#include <stdio.h>


/* user defined Signal handler! (avoids zombies) */
void waiter() {
    int cpid, stat;
    /* wait for children */
    cpid = wait(&stat);
    /* reestablish signal handler */
    signal(SIGCHLD, waiter);
} // waiter


int main(int argc, char** argv) {
    signal(SIGCHLD, waiter);            /* establish signal handler */
    struct sockaddr_in server;           /* represents the server */
    struct sockaddr_in client;          /* represents the client*/
    int sockfd, fd;                     /* file descriptors */
    uint client_len;                    /* information about the client */
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons (SERVERPORT);
    /* create socket */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        printf("Cannot get socket!\n");
        return 1;
    }

    int yes = 1;

    /* lose the pesky "address already in use" error message */
    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));

    /* bind */
    if ((bind(sockfd, (struct sockaddr *) &server, sizeof(server))) == -1) {
        printf("Error while binding\n");
        return 1;
    }

    /* listen */
    listen(sockfd, BACKLOG);
    printf("Waiting for clients\n");
    int clients = 0;

    /* start game */
    while (1) {
        client_len = sizeof(client);

        /* accept */
        fd = accept(sockfd,(struct sockaddr *) &client,   &client_len);
        printf("new client! Hostname: %s, Port: %i \n",
              (char*)inet_ntoa((client.sin_addr)),
              (client.sin_port));
        clients++;
        printf("number of clients: %i\n", clients);
```

```
        if (fd < 0) {
            printf("unable to accept client!\n");
            return 1;
        }

 /* for each client: create child process */
        if (fork() == 0 ) {

            /* play the game! */
            ServerProcess(fd,fd);

            /* close "game" file descriptor (= fd of the child!) */
            close(fd);
            return 1;
        } else
            /* close "master" file descriptor (= fd of the father) */
            close(fd);
    }

     /* close the socket */
    close(sockfd);
    return 0;
} // main
```

Client.c

```
/*-------------------------------------------------          */
/* client.c                                      090614 */
/* Robert Muehlberger                                   */
/*                                                      */
/* Client connect to server via localhost:3131          */
/*-------------------------------------------------          */
#include <signal.h>
#include <poll.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <stdio.h>

/* POLL */
void comm(int tfd , int nfd) {
    int n, i;
    struct pollfd pfd [2];
    char buf [256];
    pfd [0]. fd = tfd; /* terminal */
    pfd [0]. events = POLLIN ;
    pfd [1]. fd = nfd; /* network */
    pfd [1]. events = POLLIN ;

  for (;;) {
        /* Wait for events to occur. */
        if ( poll(pfd , 2, -1) < 0) {
            printf (" poll failed \n");
            break ;
        }

        /* Check each file descriptor. */
        for (i = 0; i < 2; i++) {
            /* If an error occurred , just return . */
            if (pfd[i]. revents&( POLLERR | POLLHUP | POLLNVAL))
                return ;

            /* If there are data present , read them from */
            /* one file descriptor and write them to the other one. */
            if (pfd[i]. revents& POLLIN ) {
```

```
            n = read(pfd[i].fd , buf , sizeof (buf));
            if (n > 0) {
                write( pfd[1-i].fd , buf , n);
            } else {
                if (n < 0)
                    printf (" read failed \n");
                return ;
            }
        }
    }
  }
} // comm


int main(int argc, char** argv) {
    struct hostent * serverEntry = NULL ;          /* represents host and server */
    struct sockaddr_in client ;                     /* client socket */
    int port, sockfd ;                              /* port and socket */
    /* check parameters */
    if (argc != 3) {
        printf("Wrong number of arguments!\n");
        printf("USAGE: %s host portnumber\n",argv[0]);
        return 1;
    }

    if (atoi(argv[2]) < 0) {
        printf("invalid port\n");
        printf("USAGE: %s host portnumber\n",argv[0]);
        return 1;
    }

    /* error with host name (invalid) */
    if ((serverEntry = gethostbyname( argv [1])) == NULL) {
        printf ("error while gethostbyname");
        exit (1);
    }

    serverEntry = gethostbyname ( argv [1]) ;

    /* get port for the game */
    port = atoi(argv[2]);

    /* open socket */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    /* inet_addr : converts string into proper address for IN_ADDR */
    client . sin_family = AF_INET ;

    /* inet_ntoa : convert internet host address to a dotted */
    client.sin_addr.s_addr = inet_addr ( inet_ntoa (*( struct in_addr *) serverEntry -> h_addr ));
    client . sin_port = htons ( atoi ( argv [2]) );

    /* call connect -- TODO : check for error */
    connect ( sockfd , ( struct sockaddr *)&client , sizeof ( client ));

    /* game loop */
    while (1) {
        char Buff[80];

        /* read from socket (= read from server)*/
        read(sockfd, Buff, sizeof(Buff));

        /* check if game is over */
        if (strcmp("Game over",Buff) == 0 || strcmp("You won!",Buff) == 0)
            return 0;

        /* write server message to screen */
        write(1, Buff, strlen(Buff));

        /* read user input */
        read(0, Buff, sizeof(Buff));

        /* write to socket (= write to server) */
```

```
        write(sockfd, Buff, strlen(Buff));
    } // while
    return 0;
} // main
```

Test: