Lösungen zur Vorlesung
GRUNDLAGEN DER INFORMATIK I
Studiengang Elektrotechnik
SS 03

AG Betriebssysteme FB3
Kirsten Berkenkötter

Lösung Übungszettel 6

1 Aufgabe 1: Parallel-Server

1.1 Client

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>

#define BUFFER 1000

int main(int argc, char **argv)
{
    int sfd;                                    //Portnummer des Servers
    struct sockaddr_in saddr;                   //Socket-Adresse
    struct hostent *hostinfo;                   //Server (Host)
    size_t lensent;                             //Laenge der uebertragenen
                                                //Zeichen
    size_t len;                                 //Laenge der eingegebenen
                                                //Zeichen
    size_t wlen;                                //Laenge der uebertragenen
                                                //Zeichen pro write
    char sendbuffer[BUFFER];                    //Puffer zum Senden
    int port;                                   //Portnummer
    char **endptr;                              //fuer Aufruf von
strtoul
    char escape[2];



    //Abfragen der Argumente
    if (argc != 3)
    {
        perror("Aufruf client <HOST> <PORT>");
```

```
        exit(2);
    }

    //Hostname ist erster Parameter
    //Achtung: argv[0] ist Anzahl der Parameter
    hostinfo = gethostbyname(argv[1]);

    //Host gueltig?
    if(hostinfo == NULL)
    {
        perror("Kann den Host nicht finden!");
        exit(1);
    }

    //Port gueltig?
    port = (int)strtoul(argv[2], endptr, 10);
    if(port == 0)
    {
        perror("Ungueltiger Port");
        exit(1);
    }

    printf("%d\n", port);

    //Socket oeffnen, bei Misserfolg ist der Rueckgabewert < 0, sonst
FD
    //PF_INET = IP Protocol
    //SOCK_STREAM = verbindungsbasierter Byte-Stream
    if ((sfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket-Call()");
        exit(1);
    }

    //Server-Adresse initialisieren
    bzero((char *)&saddr, sizeof(saddr));
    //Socket Adress Familie = IP Protocol
    saddr.sin_family = AF_INET;
    //Adresse und Port setzen
    saddr.sin_port = htons(port);
    memcpy(&saddr.sin_addr.s_addr, hostinfo->h_addr, hostinfo->h_length);

    //Verbindungsaufbau
```

```
    if((connect(sfd, (struct sockaddr *)&saddr, sizeof(saddr))) < 0)
    {
        perror("Connect-Call()");
        exit(1);
    }

    //Escape als String
    escape[0] = 27;
    escape[1] = '\n';

    while(1)
    {
        //etwas einlesen
        scanf("%s", sendbuffer);
        len = strlen(sendbuffer);

        //newline-Zeichen einfuegen
        sendbuffer[len] = '\n';
        //String terminieren
        sendbuffer[len+1] = 0;
        //neue Laenge
        len = len + 1;

        //lensent sind die gesendeten Daten aus dem Puffer,
        //mit 0 initialisieren
        lensent = 0;

        //senden (es können mehrere write-Befehle benötigt werden,
        //um alles zu übertragen)
        while (lensent < len)
        {
            //wlen, wirklich gesendete Zeichen pro write
            if((wlen = write(sfd, &(sendbuffer[lensent]), len - lensent))
< 0)
            {
                perror("Write-Call()");
                exit(1);
            }
            //merken, wieviel vom Puffer gesendet wurde
            lensent = lensent + wlen;
        }

        //bei ESC wird die Verbindung geschlossen
```

```
        if (strncmp(sendbuffer, escape, sizeof(escape)) == 0)
        {
            close(sfd);
            exit(0);
        }
    }
}
```

1.2 Server

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <unistd.h>

#define BUFFER 1000

void handleConnection(int clientsfd, int childnum)
{
    char recvbuffer[BUFFER + 1];                //Puffer zum Empfangen
    size_t lenrecv;                             //Laenge der Daten
    char escape[2];                             //Escape
    char dateiname[20];                         //Dateiname
    FILE *datei;

    //Dateiname erzeugen
    sprintf(dateiname, "datei.%d.txt", childnum);

    //Datei erzeugen
    if((datei = fopen(dateiname, "w")) == NULL)
    {
        perror("Fopen-Call()");
        exit(1);
    }

    //fuer die Ausgabe in neuer Zeile anfangen
    printf("\n");

    //Escape belegen
    escape[0] = 27;
```

```
    escape[1] = '\0';

    while(1)
    {
        //Lesen
        if((lenrecv = read(clientsfd, recvbuffer, BUFFER)) < 0)
        {
            perror("Read-Call()");
            exit(1);
        }

        //wurde etwas gelesen?
        if(lenrecv > 0)
        {
            //Ende markieren
            recvbuffer[lenrecv] = 0;

            if(strncmp(recvbuffer, escape, sizeof(escape)) == 0)
            {
                printf("Server-Child %d exists\n", childnum);
                close(clientsfd);
                fclose(datei);
                free(recvbuffer);
                return;
            }

            //Schreiben
            if(fputs(recvbuffer, datei) == EOF) ;
            printf("%s: %s", dateiname, recvbuffer);
        }
    }
}

int main(int argc, char **argv)
{
    int sfd;                                    //Socket File Descriptor
    int clientsfd;                              //Client File Descriptor
    struct sockaddr_in saddr;                   //Socket-Adresse
    struct sockaddr clientaddr;                 //Client-Adresse
    socklen_t lenclientaddr;                    //Laenge der Client-Adresse
    pid_t childpid;                             //pid des Clients
    int optval;                                 //zum Freigeben des
Ports
```

```
    int port;                                   //Port
    int childnum = 0;                           //Nummer des Clients
    char **endptr;                              //fuer strtoul

    //Stimmen die übergebenen Parameter
    if(argc != 2)
    {
        perror("Aufruf server <PORT>");
        exit(2);
    }

    //Ist der Port gueltig?
    port = (int)strtoul(argv[1], endptr, 10);
    if(port == 0)
    {
        perror("Ungueltiger Port");
        exit(1);
    }

    //Socket erzeugen
    if((sfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket-Call()");
        exit(1);
    }

    //Port wieder freigeben
    optval = 1;
    if (setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &optval, sizeof(optval))
< 0)
    {
        perror("Setsockopt-Call()");
        exit(1);
    }

    //IP-Adresse und Port an Server binden
    //Adresse mit 0 initialisieren
    bzero((char *)&saddr, sizeof(saddr));

    //IP-Protokoll
    saddr.sin_family = AF_INET;
    //alle eingehenden Verbindungswuensche erlauben
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
    //Port setzen
    saddr.sin_port = htons(port);

    //binden
    if(bind(sfd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0)
    {
        perror("Bind-Call()");
        exit(1);
    }

    //auf Verbindungswuensche horchen, vier gleichzeitig moeglich
    if(listen(sfd, 4) < 0)
    {
        perror("Listen-Call()");
        exit(1);
    }

    //Verbindungen annehmen
    while(1)
    {
        //Laenge der Clientadresse
        lenclientaddr = sizeof(clientaddr);

        //Verbindungen annehmen
        if((clientsfd = accept(sfd, &clientaddr, &lenclientaddr)) <
0)
        {
            perror("Accept-Call()");
            exit(1);
        }

        //Wurde ein Client erzeugt?
        if (clientsfd >= 0)
        {
            //Anzahl der Verbindungen hochzaehlen
            childnum ++;

            //Kindprozess fuer Verbindung mit Client erzeugen
            if((childpid = fork()) < 0)
            {
                perror("Fork-Call()");
                exit(1);
            }
```

```
            //Ist es der Kindprozess, dann Verbindung abarbeiten
            //der Vaterprozess (childpid != 0) macht in der while-Schleife
            //weiter
            if(childpid == 0)
            {
                handleConnection(clientsfd, childnum);
            }
        }
    }
}
```