

Interprozesskommunikation

Andreas Mieke & Rafael Lazenhofer



4. Dezember 2017

HTBL HOLLABRUNN

5BHEL

Inhaltsverzeichnis

[1.Aufgabenstellung 2](#_Toc500224025)

[2. Realisierung 3](#_Toc500224026)

[2.Dokomentierten Source Code & Erläuterung 4](#_Toc500224027)

[2.2. Client 4](#_Toc500224028)

[2.2.1. Erläuterung 4](#_Toc500224029)

[2.2.3. Source Code 4](#_Toc500224030)

[2.3. Server 8](#_Toc500224031)

[2.3.1. Erläuterung 8](#_Toc500224032)

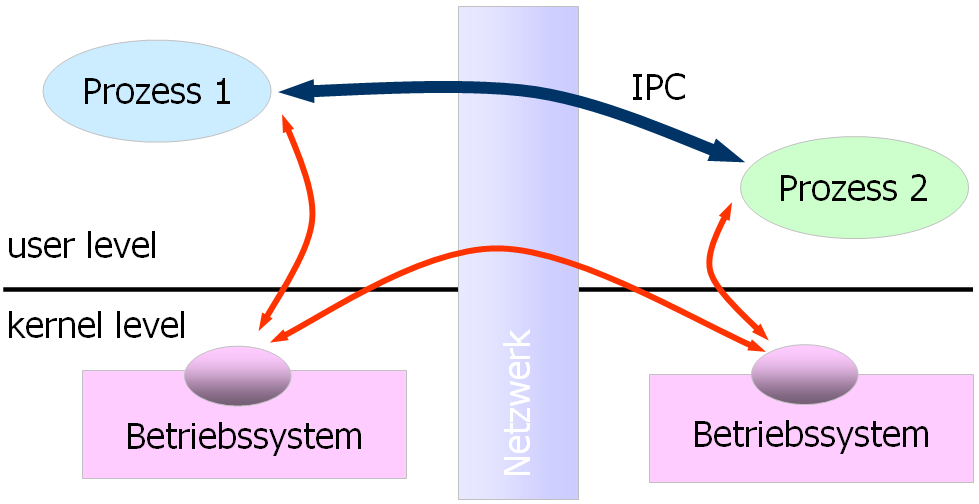
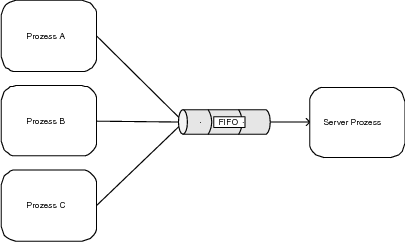
[2.3.2. Source Code 8](#_Toc500224033)

[4. Funktionsnachweis 13](#_Toc500224034)

[4.1. Tests 13](#_Toc500224035)

# 1.Aufgabenstellung

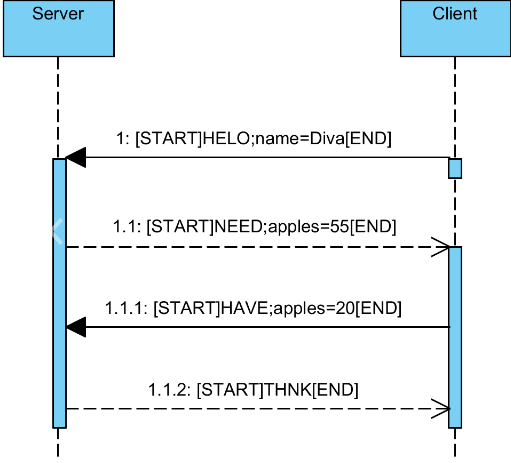
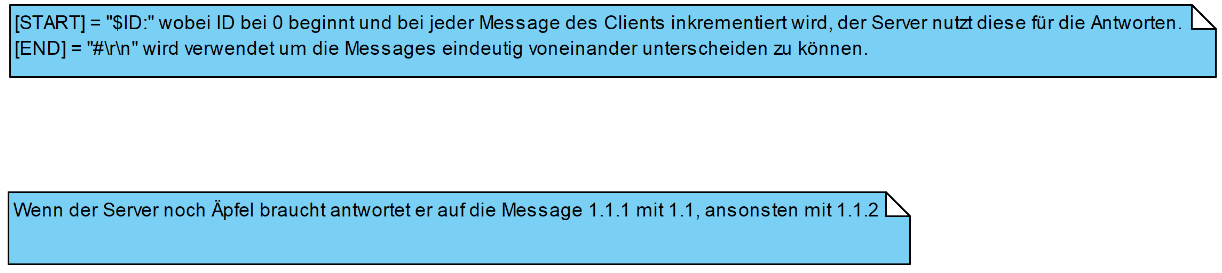
**Systemprogrammierung Beispiel 15:**

* **Angabe:**Es ist Herbst und Bauer Sepp möchte sein Fallobst zu Most verarbeiten. Einige Nachbarskinder und seine eigenen Kinder sammeln Äpfel und bringen sie zu ihm in die Scheune. Sobald die Kinder 100kg Äpfel gesammelt haben, kann Bauer Sepp mit dem Mosten beginnen.  
  Technischer Hintergrund:  
  In diesem Beispiel stellt der Bauer Sepp und seine Mostmaschine einen Server dar (Apfelverwertung). Die Kinder entsprechen Clients, die einen Dienst in Anspruch nehmen (Abnahme der Äpfel). Server und Clients sind als selbständige und unabhängige Programme implementiert. Die Abgabe der Äpfel repräsentiert die Kommunikation von Client zu Server, die Antwort von Bauer Sepp entspricht der Kommunikation vom Server zum Client. Der Server (Bauer Sepp) terminiert, wenn er von den Kindern mehr als 100kg Äpfel erhalten hat. Pro Aufruf bringen die Kinder zwischen 5kg und 10kg Äpfel (Zufallsgenerator). Die Ausgabe des Servers sieht wie folgt aus (der Name des Clients wird ausgegeben):  
    
  Bauer Sepp: heute ist Mosttag ...sammelt bitte Aepfel  
    
  Bauer Sepp: habe 6 kg Aepfel von diva erhalten  
  Bauer Sepp: jetzt sind es insgesamt 6 kg Aepfel  
    
  Bauer Sepp: habe 8 kg Aepfel von diva erhalten  
  Bauer Sepp: jetzt sind es insgesamt 14 kg Aepfel  
    
  ....  
    
  Pro Aufruf gibt der "Kinderprozess" folgende Meldung aus (Aufforderung mehr Äpfel zu sammeln):  
  Kinder: Vater Sepp benoetigt noch 94 kg Aepfel bis er mosten kann
* **Abgabe:**  
  Abzugeben sind ein dokumentierter SourceCode und ein Funktionsnachweis in Form von ScreenShots, Unix-Befehlen, Programmausgaben, …

# 2. Realisierung

Die Aufgabenstellung aus Punkt 1. wurde wie im Folgenden Diagramm dargestellt realisiert:



# 2.Dokomentierten Source Code & Erläuterung

## 2.2. Client

### 2.2.1. Erläuterung

Der Client sind die Kinder die dem Bauer Sepp bzw. dem Server die Äpfel bringen. Als Startparameter benötigt der Client die IPv4 Adresse oder den Hostname des Servers und dessen Port. Des weiteren benötigt das Programm auch den Namen des Kindes bzw. des Clients.

### 2.2.3. Source Code

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: Client - Child

Autor: Rafael Lazenhofer

Version: 1.0

Date: 01.12.2017

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <string.h>**

**#include <sys/types.h>**

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

**#include <netinet/in.h>**

**#include <netdb.h>**

**#include <time.h>**

//Unterprogramm zum Error handling

**void** error**(const char** **\***msg**)**

**{**

perror**(**msg**);**

exit**(0);**

**}**

// Messagetyp auswahl

**int** value**(char** **\***str**) {**

**char** **\***words**[]={**"HELO"**,**"NEED"**,**"THNK"**},** str2**[10];**

**int** i**,**j**=0;**

bzero**(**str2**,10);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Aussortieren für der unwichtigen

Zeichen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**for(**i**=6;**i**<10;**i**++)**

**{**

**if((**strcmp**(**str**,**";"**))==0)**

**break;**

str2**[**j**]=**str**[**i**];**

j**++;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Wert in Messagetyp in Integer

umwandeln für switch im Hauptprogramm

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**for** **(**i **=** **0;** i **<** **sizeof** words**/sizeof** words**[0];** i**++) {**

**if** **((**strcmp**(**str2**,** words**[**i**]) ==** **0)) {**

**return** i**;**

**}**

**}**

**}**

**int** main**(int** argc**,** **char** **\***argv**[])**

**{**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Varibalen definierten

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** sockfd**,** portno**,** n**,** random\_variable**,** id**=1,** m**;**

**struct** sockaddr\_in serv\_addr**;**

**struct** hostent **\***server**;**

**char** buffer**[256];**

**char** mesg**[256]=**"$0001:HELO;name=Diva#**\r\n**"**;**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Abfrage ob alle Argumente beim Start

mit geben wurden

wenn diese nicht mit gelfiefert

werden wird das Programm abgebrochen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(**argc **<** **4) {**

fprintf**(**stderr**,**"usage %s hostname port child-name**\n**"**,** argv**[0]);**

exit**(0);**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

atoi() macht aus einem String eine Int Zahl

(Hier wird der String in dem die Portnr. enthalten ist

umgewandelt in eine Int Zahl)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

portno **=** atoi**(**argv**[2]);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

socket()

AF\_INET => IPv4

SOCK\_STEAM => Unterstützt eine zuverlässig Byte-Stream-Kommunikation

0 => Für das nötige Protokoll

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sockfd **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** **0);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Überprüfung ob Socket zugänglich

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(**sockfd **<** **0)**

error**(**"ERROR opening socket"**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Überprüfen ob Host zugänglich ist

z.B.: 127.0.0.1 //localhost

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

server **=** gethostbyname**(**argv**[1]);**

**if** **(**server **==** NULL**) {**

fprintf**(**stderr**,**"ERROR, no such host**\n**"**);**

exit**(0);**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bzero()

Setzt alle Elemente auf 0 von serv\_addr

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bzero**((char** **\*) &**serv\_addr**,** **sizeof(**serv\_addr**));**

serv\_addr**.**sin\_family **=** AF\_INET**;** //IPv4

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bcopy()

eine Anzahl von Zeichen in einer

andere Zeichenfolge kopieren

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bcopy**((char** **\*)**server**->**h\_addr**,**

**(char** **\*)&**serv\_addr**.**sin\_addr**.**s\_addr**,**

server**->**h\_length**);**

serv\_addr**.**sin\_port **=** htons**(**portno**);**//Port setzen

/\*Zum Server verbinden\*/

**if** **(**connect**(**sockfd**,(struct** sockaddr **\*) &**serv\_addr**,sizeof(**serv\_addr**))<0)**

error**(**"ERROR connecting"**);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Beginn der Kommunikation

zwischen Client & Server

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf**(**"CHILD: %s**\n**"**,**argv**[3]);**

**do**

**{**

bzero**(**buffer**,256);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Message auswahl

Aufbau:

[Befehl-NR][START][NACHRICHT][END]

[START] . . . . ID wird jedes mal inkrementiert

1.1 HALO . . . . senden vom Namen z.B. Diva

1.1.1 HAVE . . . wie viele Aepfel das Kind hat

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

m**=**value**(**mesg**);**

**switch(**m**)**

**{**

**case** **0:**

sprintf**(**buffer**,** "$%04d:HELO;name=%s#**\r\n**"**,**id**++,**argv**[3]);**

printf**(**"%s: %s"**,**argv**[3],**buffer**);**

**break;**

**case** **1:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Als Zufallsgenertator wird eine

Zeitabgfrage verwendet

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

srand**(**time**(**NULL**));**

random\_variable **=** rand**()%20;**

sprintf**(**buffer**,** "$%04d:HAVE;apples=%d#**\r\n**"**,**id**++,**random\_variable**);**

printf**(**"%s: %s"**,**argv**[3],**buffer**);**

**break;**

**case** **2:**

printf**(**"Farmer Sepp has enough apples!!!**\n**"**);**

exit**(0);**

**break;**

**default:**

puts**(**"Unkown Message!"**);**

exit**(0);**

**break;**

**}**

// Schreiben

n **=** write**(**sockfd**,**buffer**,**strlen**(**buffer**));**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ERROR Handling falls beim Befehl

write ()

etwas fehlschlägt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(**n **<** **0)**

error**(**"ERROR writing to socket"**);**

// Lesen

bzero**(**mesg**,256);**

n **=** read**(**sockfd**,**mesg**,255);**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ERROR Handling falls beim Befehl

read()

etwas fehlschlägt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**if** **(**n **<** **0)**

error**(**"ERROR reading from socket"**);**

printf**(**"Sepp: %s**\n**"**,**mesg**);**

**}while(1==1);**

close**(**sockfd**);**

**return** **0;**

**}**

## 2.3. Server

### 2.3.1. Erläuterung

Das Serverprogramm stellt den Bauer Sepp dar. Er kann mehrere Clients mittels Threads ohne Probleme handeln.

### 2.3.2. Source Code

/\*

Written by Andreas Mieke , L, do you know

2017, Public Domain Gods of death love apples?

I : O : O ? Kira

I ,+ZD= O8

++Z8Z7ODO$88N D

+I7ZZOZODON88N,N I

:ZD, 7? D : =:OZZZZZD88DDDNDN

87 N8 8D ?OZZZZZO88DDNDNON :

MMNNDDDD NDDN7 I?ZOOZZZOO8DDNNNNN O D ~8I

+MNNNNNDNND8 Z$ZNDOZZOODNNNNNNNN ~ DND NN

DDDMMNNNNNNNNNNNNNNM7O8D+,8OONNDDNNNNN? NNNNNNNMNN~ =8

ZN: 8NNNNNNMNNNNNNNNOD8Z,,,=8D,~8DNNNNNNNNNNMNNNMNNND8

, :8NMN DDDDNNNNNNNNN8D~,:,,$ ,,?DDNNNNNMMMNNNNNNDDNNNNN=

$NO O8DDNNDNNMNNN8Z?I$D?,,:,,,+,~8DNND8DNNMNNNNNDDNMM8~ NNN+N?

= 8ODNN=DNNNNNNNI=~=$IIIOO+,,,??$D8IIOZ$777NNNNMMMMN8, MD, :8

DMN 8N8NNDNNN8,,::~8D,O$$$Z$OI?ZZ=$$77I7DNNNNNNNMMNM8 NN 8

NN ON:D8D8DNNDNZO7I?~+,,,,, , IZZII77 IINMNNNNNNDNNN8M= =M

NI D N8DONNDNDDNDD,,:Z8+7,7,I O?= = ?8NNNNNNNDMMM8N,ND= $

N D D NZNN8NNONND8,,,,~:888OO8DZO: ~+$8NNNNDNNNDNNNNN M8

N N O 8NNNNNNONNDDD8$,,,O 8DDDDDO$?+++$DNNNNNN8NNNNDNN N N8

Z+NDNDNNNDNZD88OZOOII$,I88D8$7IZ??ONNNNNNDNONNNNNNN N N D

7 ~ N7DODNNNN D NNZ$7$D88Z O8Z7NODNNNNNNNNNNNNNNNNNO , Z

D ~ $ZN8DNMN Z MNZ$$7NNNNDDDNNNNNNNNNMDNNNNNMMMMMMND8 +

, $ZZ8MMZ N , NNO$$$$DNNNNNNNMMNNZONNNNNMMMMMMMMNZD $~

I $OONM= NNNNNNNMMNNNNNNMMMNN MMMNOZO

$$NN88 DNNNNNNNNNNNNNNNNN NNNNN$

$$$ZNN OONNNNNNNNNNNNN8$ NNNOZ7

777$Z8+ ZNNNNNNNNNNNNNZ NN$$$$

$$$$NNNN $NNNNNNNNNNNNN: ZNNN$77

$$ZNNN ZNNNNNNNNNNNNN8 NNNNN$: 8

7ZNNNND IIZZNNNNNNNNNNN$ZNO +MNNNZ$$$

8Z$$$N8 DO$$$$8NNNNNNOZZDNN$ZNZ NNN$N$Z

,O$$ONO DZ+NNNN$NNNNNNNNNN8$NZO =NNNZD$~

Z$$$Z$NN $O+~ZNNNNNNNNNNNNN$ONDOD~ NNNDNZ

=N$8NNN $:+8DDZND8ZZDNNNNNNOODNNO INONNNDZ

$ Z$NNN DO$?:ON88D+=7+88D8DN88NNZMND NNNNNND D

7?$$$N DDD8DDNDN$8$7$7O8ODDDND,NNNNNZ NNNNN$

$$$N+ NDNDD:NNDDDDD8$NNDNNNON$8NNNNNND NNNNN$

$$$NO DDDNNDNNNNDDDDDDNDDN$MDDDNNNNNNNND ?NNNDO

+$DDN ZNNNNNNINNNDNDDDNNNDDDN:DDNNNNNNNZO$ NND$,

NONN N DNNNDNNNDDNDDDNNNDDDNNN~NNNNNN=$$? NNN$

$$NN D~~7NNNNNNNDDNDDNNNNNDDNNDDNNZNNNN88ZO? NN$$

$NN O NN8DNMNNNDDNDNNNNNNDDNNNDNNNNN$888ZOO NNZ$

$$ND = DNO8NNNNNNNNNDNNNNNNDDNNNDNNNNNNNZDNZN$ NNN$

$$$N NZODNNNNN=8N~DDNN8N DD 7NDNNNNNNNNDNND8? ,NNDZ

7ZNNO DZ8NNNNN ON ZD8 NN NNNNNNNNDNNNNNOD $NNN8

OD8N NZNNNNNN: N N NN DD?NNNN88NNNDNOZ DNN$7

,ODN $DNNNNN + N D: N NNNNDOZZNN+, NNN$

?ZO:O $$$DN8N + D + DNNN8ZZ$$ NN$$

~Z7ZDODD$$DNNN NNNNNZ$8 ,NN$$

~88NDNN$$$NNNN ~NNNNN$$ 7DDZ$Z

DNZ$ZNNNNN7 NNNN$$ DNNNN$

O$$$$NNNNN ~NNN$$Z NNDNND

$$$$$$$ON $NNNZ$+ DNNNNNN$

$$$$NNN$NN NNNNN7 NNZ$NNNN=

7$$$DNNNNNN NNNNNNDN$$NNNDN$

$$$$NZNNNNNN NNZ$$NN NNNZN$

$$D$$~ $NZDNN~ DNNNN$N ZNN$NDO

O$OI8D $$$NNN NNNNNNN NDI887

==7+$O $$$DNNN NNNNNNN N7O~ 7

8I8,8, D$NNNN NNNN$$ N8DZ:

,8NN?ON8,$DNNN+ NNNO$$ O8ODND

ODNNNNNM$NNNN NNNN$ NZ NO

NN N 8$ONN NNNN$ NZ8NZ

N NNNN NNN$$ D8

ON$N8 NN8$~

$$NN NN$O

$$NN :NZN$

ZDNNN INN$

Z$NNN ONN8

$NNN NNN$

IZNNO NN$Z

~$ZNNI NNN$Z

+NNNNN NNN8Z

Z8$ZN D8D$O

$N8N MDN$

$NNZ8 ~ZNNO

=NNNN :NNN$

$NNNN NONZ

ZNNNN NDNN

$$NNN ONND

$NNN NND$

$NN 8NN$

? =NZ

\*/

/\*

Here begins the funny configuration stuff! Actually you just have to set the

amount of apples Ryuk ? ugh, I mean Sepp ? will get, the port he will listen

to and other stuff once I come up with it.

\*/

**#define APPLES\_WANTED 100** // How many apples does Ryuk want?

**#define PORT 8666** // On which port should he listen?

**#define NUM\_THREADS 10** // How many clients at one time?

**#define RYUK\_NAME** "Ryuk" // If someone wants to call me Sepp,

// change this line -.-

**#define BUFFER\_SIZE 2048** // String buffer for receiving

/\*

Thank you, human, for the configuration. Now the real magic begins!

\*/

**#define BACKLOG 10**

**#include <stdio.h>**

**#include <string.h>** // strlen

**#include <stdlib.h>**

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

**#include <arpa/inet.h>** // inet\_addr

**#include <unistd.h>** // write

**#include <signal.h>** // signal

**#include <pthread.h>** // Threading

**#include <assert.h>** // assert

**long** apples\_left **=** APPLES\_WANTED**;** // Holds the number of apples to collect

**int** socket\_fd**;** // Socket file descriptor

pthread\_mutex\_t apple\_lock**;** // Mutex for the apples\_left counter

**void** **\***handle\_socket**(void** **\***l\_socket\_fd**);**

**void** handle\_int**(int** dummy**);**

**int** main**(int** argc**,** **char const** **\***argv**[])**

**{**

**int** new\_socket\_fd**,** c**, \***new\_sock**;**

**struct** sockaddr\_in server**,** client**;**

**char** **\***msg**;**

// SIGINT handler registration for graceful termination

signal**(**SIGINT**,** handle\_int**);**

// Init the mutex for the apple\_left counter

**if** **(**pthread\_mutex\_init**(&**apple\_lock**,** NULL**))**

**{**

perror**(**"Could not create mutex"**);**

**return** **1;**

**}**

// Create the server socket

socket\_fd **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** **0);**

**if** **(**socket\_fd **== -1)**

**{**

perror**(**"Could not create socket"**);**

**return** **1;**

**}**

// Address and port

server**.**sin\_family **=** AF\_INET**;**

server**.**sin\_addr**.**s\_addr **=** INADDR\_ANY**;**

server**.**sin\_port **=** htons**(**PORT**);**

// Bind the socket

**if** **(**bind**(**socket\_fd**, (struct** sockaddr **\*)&**server**,** **sizeof(**server**)) <** **0)**

**{**

perror**(**"Could not bind to socket"**);**

**return** **1;**

**}**

puts**(**"Bound to socket!"**);**

// Start listening on the bound socket

listen**(**socket\_fd**,** BACKLOG**);**

puts**(**"Waiting for connections..."**);**

puts**(**RYUK\_NAME ": Please bring me some apples!"**);**

c **=** **sizeof(struct** sockaddr\_in**);**

// Wait for new connections and accept them

**while** **((**new\_socket\_fd **=** accept**(** socket\_fd**,**

**(struct** sockaddr **\*)&**client**,**

**(**socklen\_t **\*)&**c**)**

**))**

**{**

puts**(**"Connection accepted!"**);**

pthread\_t apple\_thread**;**

// Allocate memory for the thread argument

new\_sock **=** malloc**(sizeof(int));**

// Set the argument for the thread to the socket file descriptor

// of the newly accepted connection, so the thread can communicate

// with the client that was just accepted

**\***new\_sock **=** new\_socket\_fd**;**

// Create the thread for the accepted client. This thread will handle all

// further communication with this client.

**if** **(**pthread\_create**( &**apple\_thread**,**

NULL**,**

handle\_socket**,**

**(void** **\*)**new\_sock**)**

**<** **0)**

**{**

perror**(**"Could not create socket thread"**);**

**return** **1;**

**}**

puts**(**"Handler assigned"**);**

**}**

**if** **(**new\_socket\_fd **<** **0)**

**{**

perror**(**"Could not accept connection"**);**

**return** **1;**

**}**

// There is no valid condition that should bring us here, so

// assert that we never reach this point of the program, so if we

// ever do, we can easier debug it.

**assert(!**"End of main should never happen"**);**

**}**

// This function is used to communicate with a specific client

// that was accepted in our main loop, in a separate thread for

// each client.

**void** **\***handle\_socket**(void** **\***l\_socket\_fd**)**

**{**

**int** fd **= \*(int** **\*)**l\_socket\_fd**,** read\_size**,** msg\_id**;**

**char** msg**[**BUFFER\_SIZE**],** name**[64], \***tok**;**

// Read data from the client

**while** **((**read\_size **=** recv**(**fd**,** msg**,** **sizeof(**msg**),** **0)) >** **0)**

**{**

// Parse the message we got from the client

// Look for the valid start sequence of the clients message

tok **=** strchr**(**msg**,** '#'**);**

**if** **(**tok **==** NULL **||**

msg**[0] !=** '$' **||**

msg**[5] !=** ':' **||**

strncmp**(**tok**,** "#**\r\n**"**,** **3) !=** **0)**

**{**

// If there is no valid start sequence, print an error

// and disconnect the client.

puts**(**"Wrong message format, disconnecting client!"**);**

close**(**fd**);**

**break;**

**}**

// The first 4 bytes after the start sequence ($) hold the id of

// the message that was sent.

**char** id**[5] = {**msg**[1],** msg**[2],** msg**[3],** msg**[4],** **0};**

msg\_id **=** atoi**(**id**);**

// Next check the type of the message and the argument:

// HELO

**if** **(**strncmp**(**msg**+6,** "HELO;name="**,** **10) ==** **0)**

**{**

tok **=** strchr**(**msg**,** '#'**);**

**\***tok **=** '\0'**;**

tok **=** strchr**(**msg**,** '='**);**

strncpy**(**name**,** tok**+1,** **sizeof(**name**));**

**}**

// HAVE

**else if** **(**strncmp**(**msg**+6,** "HAVE;apples="**,** **12) ==** **0)**

**{**

tok **=** strchr**(**msg**,** '#'**);**

**\***tok **=** '\0'**;**

tok **=** strchr**(**msg**,** '='**);**

**int** got **=** atoi**(**tok**+1);**

// Lock the mutex for the apple\_left counter

pthread\_mutex\_lock**(&**apple\_lock**);**

// Substract apples we got from the apple\_left counter

// This needs to be done while the lock is hold, to ensure

// that no other thread decrements the counter at the same

// time, causing a data race.

apples\_left **-=** got**;**

pthread\_mutex\_unlock**(&**apple\_lock**);**

printf**(**RYUK\_NAME ": Got %dkg apples from %s, now I have %ldkg.**\n**"**,**

got**,** name**,** APPLES\_WANTED **-** apples\_left**);**

**}**

// Check if we still need more apples

**if** **(**apples\_left **>** **0)**

**{**

// If yes, ask for them

sprintf**(**msg**,** "$%04d:NEED;apples=%ld#**\r\n**"**,** msg\_id**,** apples\_left**);**

write**(**fd**,** msg**,** strlen**(**msg**));**

**}**

**else**

**{**

// If not, we are done. Thank and disconnect the client.

sprintf**(**msg**,** "$%04d:THNK#**\r\n**"**,** msg\_id**);**

write**(**fd**,** msg**,** strlen**(**msg**));**

close**(**fd**);**

puts**(**"Enough apples, disconnecting..."**);**

**break;**

**}**

memset**(**msg**,** **sizeof(**msg**),** **0);**

**}**

**if** **(**read\_size **==** **0)**

**{**

puts**(**"Client disconnected!"**);**

**}**

**if** **(**read\_size **<** **0)**

**{**

perror**(**"Receive failed"**);**

close**(**fd**);**

**}**

// Cleanup the socket

free**(**l\_socket\_fd**);**

**return** **0;**

**}**

// SIGINT handler

**void** handle\_int**(int** dummy**)**

**{**

puts**(**"**\r**Received SIGINT, exiting now..."**);**

close**(**socket\_fd**);**

pthread\_mutex\_destroy**(&**apple\_lock**);**

exit**(0);**

**}**

/\*

LLLLLLLLL8 L? LLLLLLLN

LLLLLLLLLLLLLLLL: :ZLLLL, LLLLLL

LLLLLLLLLLLLLLLLLLLLLLL= ,LLLLLL,

LLLLLLLLLLLLLLLLLLLLLLL: LLLLLLL

LL +LLLLLLLLL,LLLLL LLLLLLL

LLLLL LLLLLLL

Risking your life NLLLLL LLLLLLL

and doing something LLLLLLL LLLLLLL

that could rob you OLLLLLLL ,LLLLLLL

of your life are LLLLLLLL LLLLLLL+

exact opposites. LLLLLLLL LLLLLLL8

? L LLLLLLLL LLLLLLLL

LLLLLLLL LLLLLLLL

LLLLLLLL LLLLLLLL

LLLLLLLL LLLLLLLL

,=LLLLLLLLLLLLL~ LLLLLLLL

:LLLLLLLLLLLLLLLL8 LLLLLLLL

LLLLLLLLLLLLLLLLLL LLLLLLLL

ZLLLLLLLLLLL LLLLLLLL

DLLLLLLLLL LLLLLLLN

LLLLLLLLL LLLLLLL

7LLLLLLLL LLLLLLL

LLLLLLLLLLLLLLI LLLLLL

7LLLLLLLLLLLLL ,LLLLL7

~LLLLLLLLD LLLLL

LLLLLLL ZLLLL

LLLLL +LLL,,

LLLLL LLL

LLLL ,LLL

LLLL LLL ILLLLLLLLLLL: ,

LLL LLLLLLLLLLLLLLLLLLLLLLL, ,

:LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL= LL

DLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLZ

DLLLLLLLLLD$?~,,=IZLLLLLLLLLLLLLLLLLL,

,?LLLL7 ?LLLLLLLLLLZ

7NLL LLLLLL

8LL LL=

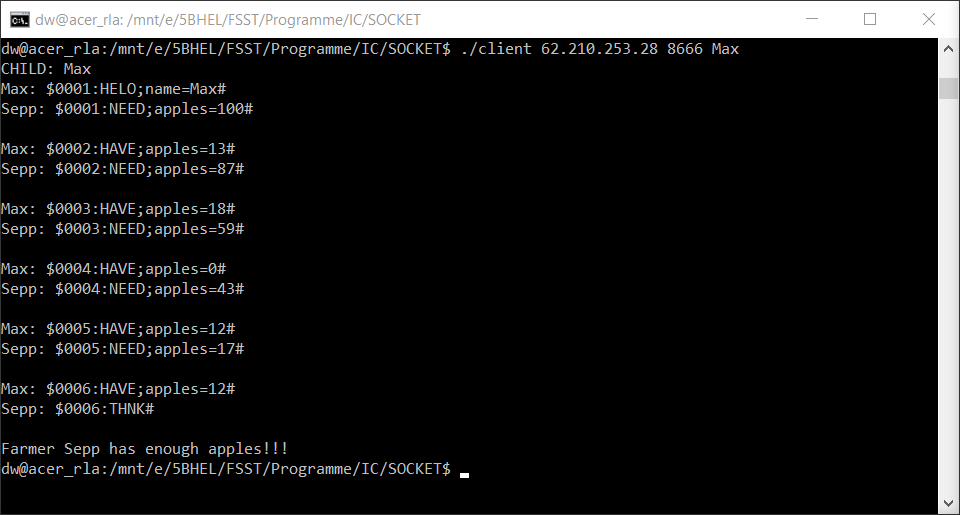
\*/

# 4. Funktionsnachweis

Das Programm wurde auf seine Funktionen geprüft. Alle Ergebnisse werden in den unter punkten beschrieben und mit Screenshots bewiesen.

## 4.1. Tests

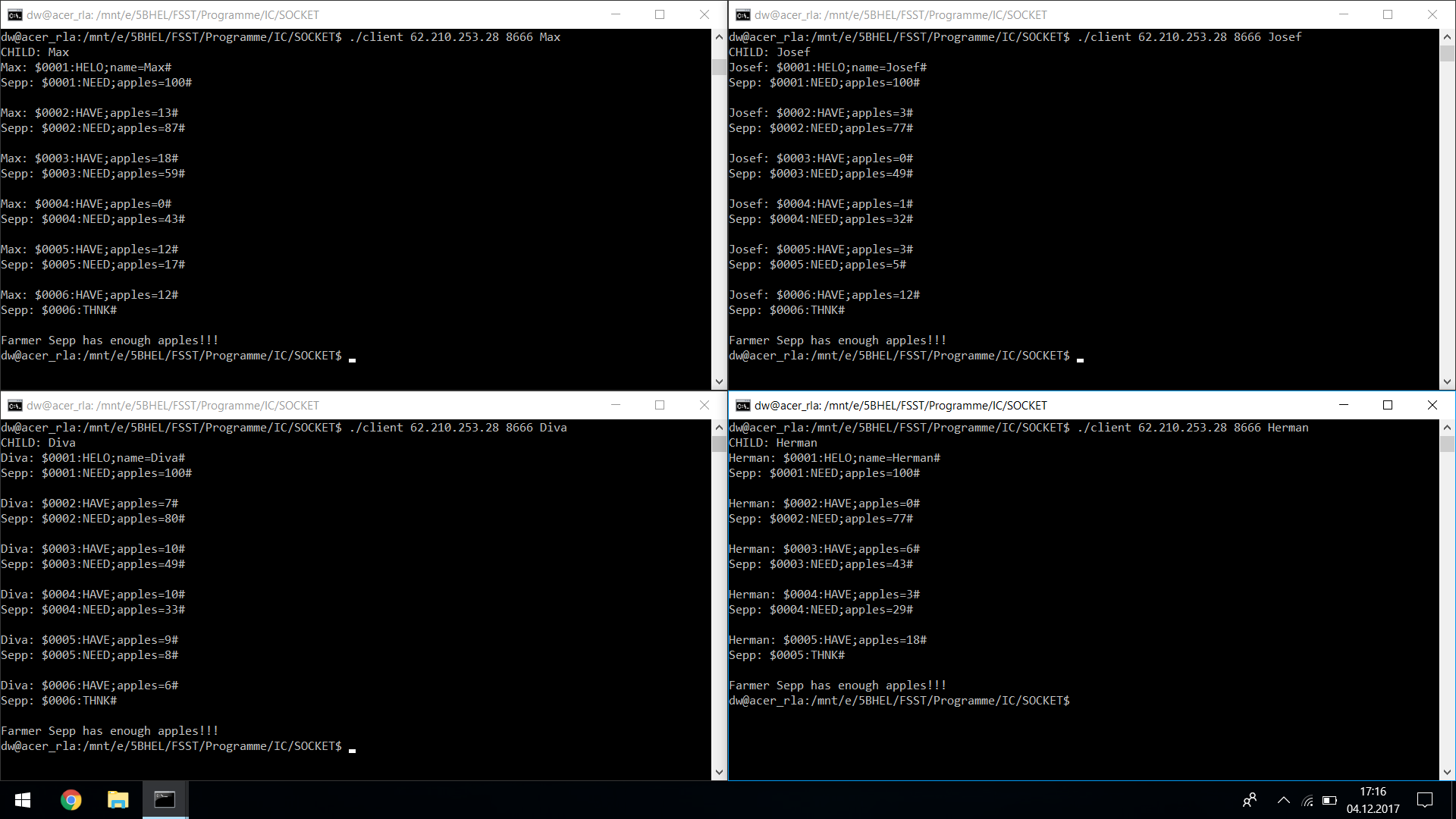
Zu Beginn wurde getestet ob ein Client sich zum Server verbinden kann. Des weiteren ob er Daten vom Server empfangen und zum Server senden kann. Diesen Test hat das Programm bestanden wie im unteren Screenshot zu sehen:



Anfangsparameter für Client

Übertragung der Daten bzw. Äpfel

Danach wurde getestet ob der Server gleichzeitigt mehrere Daten von verschiedenen Clients empfangen und senden kann. Diesen Test hat das Serverprogramm einwandfrei bestanden:

Senden von mehrern Clients:

Herman

Josef

Max

Diva

Server Daten:

Aufforderung für Äpfel

Empfangene Äpfel von Clients