



INTERPROZESSKOMMUNIKATION

Andreas Mieke & Rafael Lazenhofer



4. DEZEMBER 2017

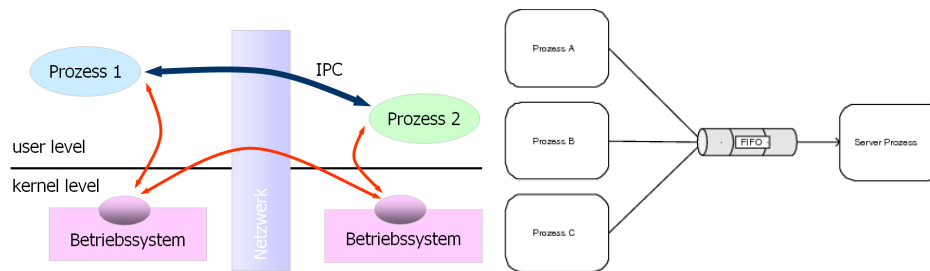
HTBL HOLLABRUNN
5BHEL

Inhaltsverzeichnis

1. Aufgabenstellung	2
2. Realisierung	3
2. Dokumentierten Source Code & Erläuterung	4
2.2. Client.....	4
2.2.1. Erläuterung.....	4
2.2.3. Source Code.....	4
2.3. Server.....	8
2.3.1. Erläuterung.....	8
2.3.2. Source Code.....	8
4. Funktionsnachweis	13
4.1. Tests.....	13

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):

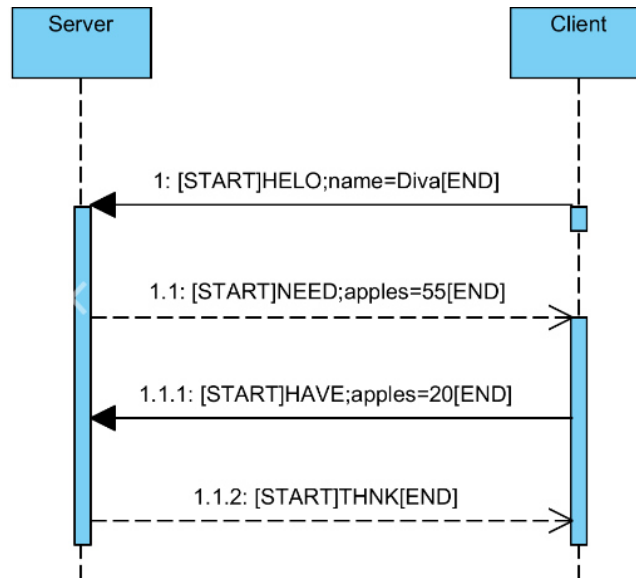
Kinder: Vater Sepp benoetigt noch 94 kg Aepfel bis er mosten kann

- 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:



[START] = "\$ID:" wobei ID bei 0 beginnt und bei jeder Message des Clients inkrementiert wird, der Server nutzt diese für die Antworten.
[END] = "#\r\n" wird verwendet um die Messages eindeutig voneinander unterscheiden zu können.

Wenn der Server noch Äpfel braucht antwortet er auf die Message 1.1.1 mit 1.1, ansonsten mit 1.1.2

2. Dokumentierten 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[])
{
    /*****
    Variablen 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 geliefert
    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_STREAM => 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, "$%04d:HELO;name=%s#\r\n", id++, argv[3]);
            printf("%s: %s", argv[3], buffer);
            break;
        case 1:
            /*****
                Als Zufallsgeneratator wird eine
                Zeitabfrage verwendet
                *****/
            srand(time(NULL));
            random_variable = rand() % 20;
            sprintf(buffer,
"$%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
    2017, Public Domain
    ,
    L, do you know
    Gods of death love apples?
    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
    101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
    201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
    301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400
    401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500
    501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600
    601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700
    701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800
    801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900
    901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000
*/

/*
    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);
    // Subtract 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("\rReceived SIGINT, exiting now...");
    close(socket_fd);
    pthread_mutex_destroy(&apple_lock);
    exit(0);
}
```

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:



```
dw@acer_rla: /mnt/e/5BHEL/FSST/Programme/IC/SOCKET
dw@acer_rla:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$ ./client 62.210.253.28 8666 Max
CHILD: Max
Max: $0001:HELO;name=Max#
Sepp: $0001:NEED;apples=100#

Max: $0002:HAVE;apples=13#
Sepp: $0002:NEED;apples=87#

Max: $0003:HAVE;apples=18#
Sepp: $0003:NEED;apples=59#

Max: $0004:HAVE;apples=0#
Sepp: $0004:NEED;apples=43#

Max: $0005:HAVE;apples=12#
Sepp: $0005:NEED;apples=17#

Max: $0006:HAVE;apples=12#
Sepp: $0006:THNK#

Farmer Sepp has enough apples!!!
dw@acer_rla:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$
```

Übertragung der Daten bzw. Äpfel

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

Senden von mehrern Clients:

```

dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$ ./client 62.210.253.28 8666 Max
CHILD: Max
Max: $0001:HELO;name=Max#
Sepp: $0001:NEED;apples=100#

Max: $0002:HAVE;apples=13#
Sepp: $0002:NEED;apples=87#

Max: $0003:HAVE;apples=18#
Sepp: $0003:NEED;apples=59#

Max: $0004:HAVE;apples=0#
Sepp: $0004:NEED;apples=43#

Max: $0005:HAVE;apples=12#
Sepp: $0005:NEED;apples=17#

Max: $0006:HAVE;apples=12#
Sepp: $0006:THINK#

Farmer Sepp has enough apples!!!
dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$

dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$ ./client 62.210.253.28 8666 Josef
CHILD: Josef
Josef: $0001:HELO;name=Josef#
Sepp: $0001:NEED;apples=100#

Josef: $0002:HAVE;apples=3#
Sepp: $0002:NEED;apples=77#

Josef: $0003:HAVE;apples=0#
Sepp: $0003:NEED;apples=49#

Josef: $0004:HAVE;apples=1#
Sepp: $0004:NEED;apples=32#

Josef: $0005:HAVE;apples=3#
Sepp: $0005:NEED;apples=5#

Josef: $0006:HAVE;apples=12#
Sepp: $0006:THINK#

Farmer Sepp has enough apples!!!
dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$

dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$ ./client 62.210.253.28 8666 Diva
CHILD: Diva
Diva: $0001:HELO;name=Diva#
Sepp: $0001:NEED;apples=100#

Diva: $0002:HAVE;apples=7#
Sepp: $0002:NEED;apples=80#

Diva: $0003:HAVE;apples=10#
Sepp: $0003:NEED;apples=49#

Diva: $0004:HAVE;apples=10#
Sepp: $0004:NEED;apples=33#

Diva: $0005:HAVE;apples=9#
Sepp: $0005:NEED;apples=8#

Diva: $0006:HAVE;apples=6#
Sepp: $0006:THINK#

Farmer Sepp has enough apples!!!
dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$

dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$ ./client 62.210.253.28 8666 Herman
CHILD: Herman
Herman: $0001:HELO;name=Herman#
Sepp: $0001:NEED;apples=100#

Herman: $0002:HAVE;apples=0#
Sepp: $0002:NEED;apples=77#

Herman: $0003:HAVE;apples=6#
Sepp: $0003:NEED;apples=43#

Herman: $0004:HAVE;apples=3#
Sepp: $0004:NEED;apples=29#

Herman: $0005:HAVE;apples=18#
Sepp: $0005:THINK#

Farmer Sepp has enough apples!!!
dw@acer_r1a:/mnt/e/5BHEL/FSST/Programme/IC/SOCKET$
  
```

Server Daten:

```

Bound to socket!
Waiting for connections...
Ryuk: Please bring me some apples!
Connection accepted!
Handler assigned
Connection accepted!
Handler assigned
Connection accepted!
Handler assigned
Ryuk: Got 1kg apples from Diva, now I have 1kg.
Ryuk: Got 4kg apples from Josef, now I have 5kg.
Ryuk: Got 9kg apples from Herman, now I have 14kg.
Ryuk: Got 0kg apples from Diva, now I have 14kg.
Ryuk: Got 18kg apples from Josef, now I have 32kg.
Ryuk: Got 1kg apples from Herman, now I have 33kg.
Ryuk: Got 10kg apples from Diva, now I have 43kg.
Ryuk: Got 8kg apples from Josef, now I have 51kg.
Ryuk: Got 7kg apples from Herman, now I have 58kg.
Ryuk: Got 18kg apples from Diva, now I have 76kg.
Connection accepted!
Handler assigned
Ryuk: Got 1kg apples from Josef, now I have 77kg.
Ryuk: Got 14kg apples from Herman, now I have 91kg.
Ryuk: Got 14kg apples from Max, now I have 105kg.
Enough apples, disconnecting...
Ryuk: Got 18kg apples from Diva, now I have 123kg.
Enough apples, disconnecting...
Ryuk: Got 14kg apples from Josef, now I have 137kg.
Enough apples, disconnecting...
Ryuk: Got 16kg apples from Herman, now I have 153kg.
Enough apples, disconnecting...
  
```

Aufforderung für Äpfel

Empfangene Äpfel von Clients