INTERPROZESSSYNCHRONISTATION

Andreas Mieke & Rafael Lazenhofer

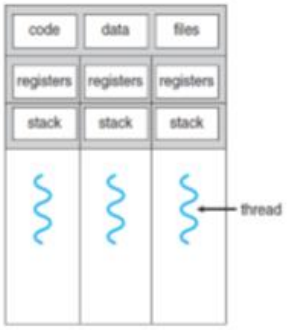
9. JANUAR 2017

HTBL HOLLABRUNN
5BHEL

Inhaltsverzeichnis

1.Aufgabenstellung

Systemprogrammierung Beispiel 10:



- **The unisex bathroom problem**

I wrote this problem[1] when a friend of mine left her position teaching physics at Colby College and took a job at Xerox.

She was working in a cubicle in the basement of a concrete monolith, and the nearest women's bathroom was two floors up. She proposed to the Uberboss that they convert the men's bathroom on her floor to a unisex bathroom, sort of like on Ally McBeal.

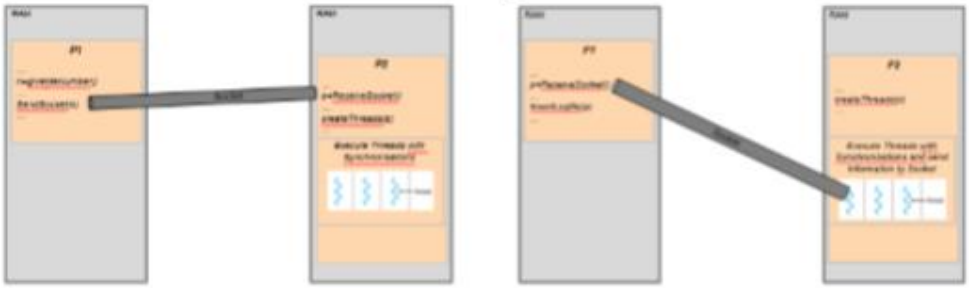
The Uberboss agreed, provided that the following synchronization constraints can be maintained:

- There cannot be men and women in the bathroom at the same time.
- There should never be more than three employees squandering company time in the bathroom.

Of course the solution should avoid deadlock. For now, though, don't worry about starvation. You may assume that the bathroom is equipped with all the semaphores you need.

- **Aufgaben:**
  - **Synchronisation**
    Analysieren Sie das obige Problem und schreiben sie ein entsprechendes C-Programm, welches die Aufgabestellung realisiert und löst. Wählen sie passende Programmkonstrukte um parallele Strukturen, Ein- und Ausgaben und Synchronisationsmechanismen zu realisieren.

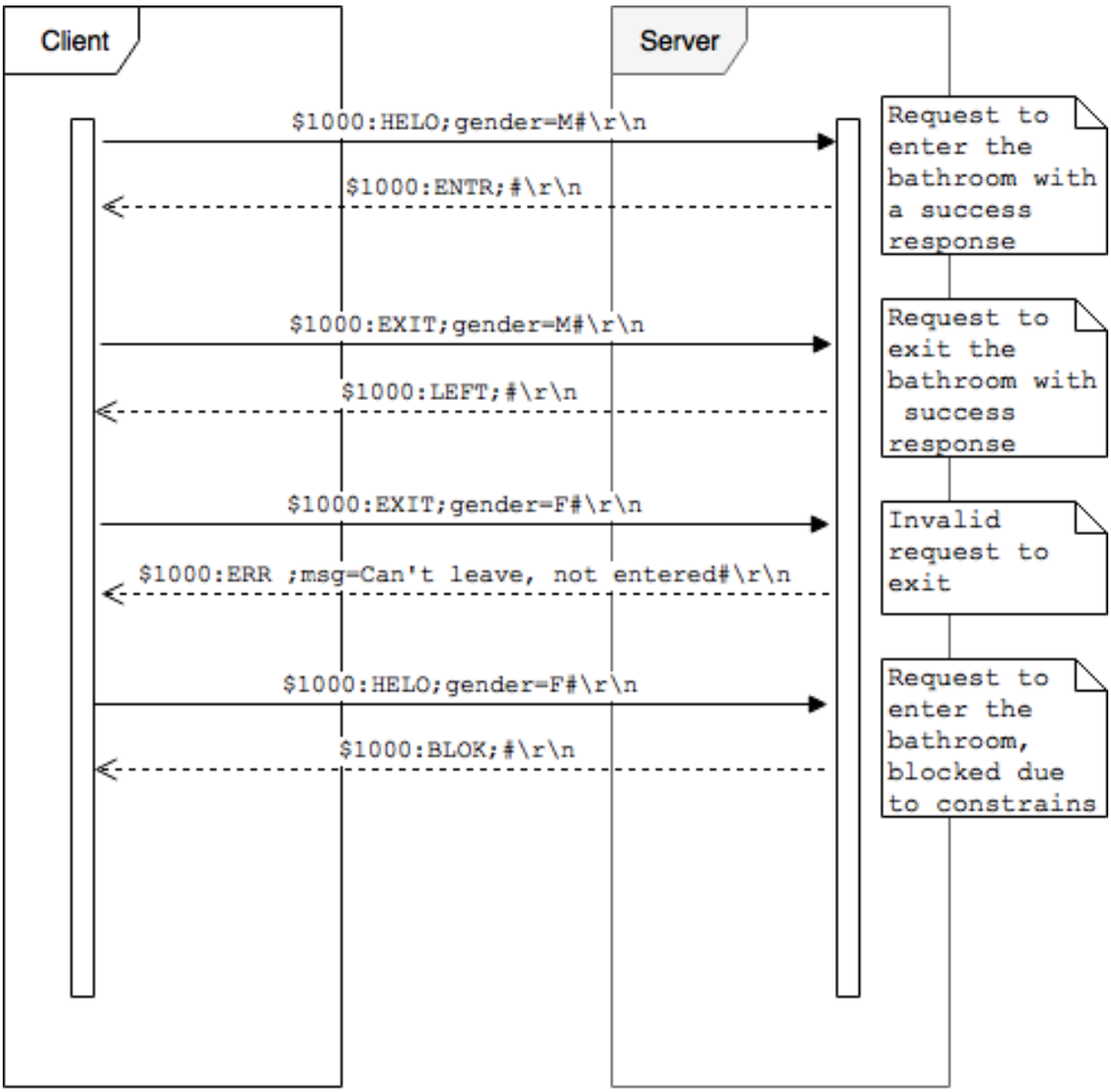
  - **Sockets**
    Erweitern Sie die obige Aufgabenstellung mit einer Socketanwendung. Mit dieser Erweiterung können z.B. Eingaben übergeben werden oder Ausgaben an einen weiteren Serverprozess übertragen werden.



- **Abgabe:**
  Abzugeben sind ein dokumentierter SourceCode und ein Funktionsnachweis in Form von ScreenShots, Unix-Befehlen, Programmausgaben, ...
  Abzugeben ist auch eine prinzipielle Beschreibung des Problems.

2. Realisierung

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



3. Beschreibung des Problems

Das Problem bei dieser Aufgabe liegt daran das ein Mann und eine Frau nicht zur gleichen zeit sich auf der Toilette befinden dürfen, somit muss stehts einer der beiden warten. Darüber hinaus dürfen nicht mehr als 3 Personen des gleichen Geschlechtes sich auf dem Klo befinde, somit muss es falls es mehr als 3 Personen gibt einer warten.

4.Dokomentierten Source Code & Erläuterung

4.2. Client

4.2.1. Erläuterung

Der Client sind die Personen die auf die Toilette müssen. Als Startparameter benötigt der Client die IPv4 Adresse oder den Hostname des Servers und dessen Port. Des weiteren benötigt das Programm auch das Geschlecht der Person.

4.2.3. Source Code

```
/************************
 Name: Client - Toilet
 Autor: Rafael Lazenhofer
 Version: 1.0
 Date: 06.01.2018
 ***********************/
#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","ENTR","BLOK","LEFT"}, 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;gender=M#\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 gender\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("Person: %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 Geschlecht
     1.2 EXIT . . . . rausgehen nach dem man fertig ist
       1.3 BLOK . . . . Man muss warten
       1.4 ENTR . . . . Man darf das Klo betreten
     *********************************************/
     m=value(mesg);
     switch(m)
     {
          case 0:
                 sprintf(buffer, "$%04d:HELO;gender=%s#\r\n",id++,argv[3]);
                 printf("%s: %s",argv[3],buffer);
          break;
          case 1:
                 /*****************************
                 Als Zufallsgenertator wird eine
                        Zahl von 1 bis 10 generiert
                 *****************************/
                 srand(time(NULL));
                 random_variable = rand()%10;
                        sleep(random_variable);
                 sprintf(buffer, "$%04d:EXIT;gender=%s#\r\n",id++,argv[3]);
                 printf("%s: %s",argv[3],buffer);
          break;
          case 2:
          printf("You have to wait until it is space!\n");
              sprintf(buffer, "$%04d:HELO;gender=%s#\r\n",id++,argv[3]);
              sleep(3);
          break;

              case 3:
              //printf("Toilet: %s\n",mesg);
              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("Toilet: %s\n",mesg);
    }while(1==1);

      close(sockfd);
    return 0;
}
```

5.3. Server

5.3.1. Erläuterung

Das Serverprogramm stellt die Toilette dar. Sie kann mehrere Clients mittels Threads ohne Probleme handeln.

5.3.2. Source Code

```
#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


int socket_fd;                          // Socket file descriptor
pthread_mutex_t counters_lock;          // Mutex for the counters


/* The following counter variables may only be accessed when the
 * counters_lock lock is held!
 */
long m_counter = 0;     // Counter variable for male users
long f_counter = 0;     // Counter variable for female users

#define PORT                        8666  // Port to listen on
#define CONCURRENT_BATHROOM_USERS   3     // Number of people of the same
                                          // gender that can be in the
bathroom at the same time
#define BUFFER_SIZE                 2048  // String buffer for receiving

#define BACKLOG         10

/* SIGINT Handler
 */
void handle_int(int dummy)
{
    puts("\rReceived SIGINT, exiting now...");
    close(socket_fd);
    pthread_mutex_destroy(&counters_lock);
    exit(0);
}
```

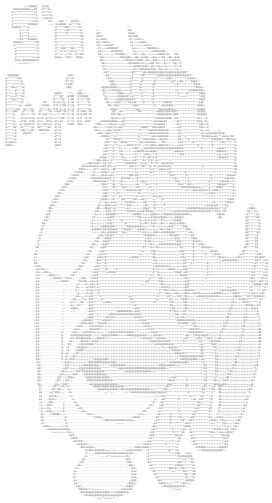
```c
/* This function is used to communicate with a specific client
 * that was accepted in our main loop, in a separate thread for
 * each client.
 */
void *handle_socket(void *l_socket_fd)
{
    int fd = *(int *)l_socket_fd, read_size, msg_id;
    char msg[BUFFER_SIZE], gender[64], *tok;

    // Read data from the client
    while ((read_size = recv(fd, msg, sizeof(msg), 0)) > 0)
    {
        // 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;gender=", 12) == 0)
        {
            tok = strchr(msg, '#');
            *tok = '\0';
            tok = strchr(msg, '=');
            strncpy(gender, tok+1, sizeof(gender));
```

```
// Lock the mutex for the counters
            pthread_mutex_lock(&counters_lock);
            switch (gender[0]) {
                case 'M':
                 if (m_counter < CONCURRENT_BATHROOM_USERS && !f_counter) {
                        m_counter++;
                   printf("%c User entered, now %ld users\n", 'M', m_counter);
                        sprintf(msg, "$%04d:ENTR;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    } else {
                        printf("%c User can't enter now. (%ld M, %ld F)\n",
                        'M', m_counter, f_counter);
                        sprintf(msg, "$%04d:BLOK;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    }
                break;
                case 'F':
                if (f_counter < CONCURRENT_BATHROOM_USERS && !m_counter) {
                        f_counter++;
                        printf("%c User entered, now %ld users\n", 'F',
                        f_counter);
                        sprintf(msg, "$%04d:ENTR;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    } else {
                        printf("%c User can't enter now. (%ld M, %ld F)\n",
                        'F', m_counter, f_counter);
                        sprintf(msg, "$%04d:BLOK;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    }
                break;
                default:
                    sprintf(msg, "$%04d:ERR ;msg=%s#\r\n", msg_id,
                     "Unhandled gender");
                    write(fd, msg, strlen(msg));
                break;
            }

            pthread_mutex_unlock(&counters_lock);
        }
```

```
// EXIT
        else if (strncmp(msg+6, "EXIT;gender=", 12) == 0)
        {
            tok = strchr(msg, '#');
            *tok = '\0';
            tok = strchr(msg, '=');
            strncpy(gender, tok+1, sizeof(gender));

            // Lock the mutex for the counters
            pthread_mutex_lock(&counters_lock);
            switch (gender[0]) {
                case 'M':
                    if (m_counter) {
                        m_counter--;
                        printf("%c User left, now %ld users\n", 'M',
                        m_counter);
                        sprintf(msg, "$%04d:LEFT;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    } else {
                        printf("%c User can't leave, not entered. (%ld M,
                        %ld F)\n", 'M', m_counter, f_counter);
                        sprintf(msg, "$%04d:ERR ;msg=%s#\r\n", msg_id,
                        "Can't leave, not entered");
                        write(fd, msg, strlen(msg));
                    }
                break;
                case 'F':
                    if (f_counter) {
                        f_counter--;
                        printf("%c User left, now %ld users\n", 'F',
                        f_counter);
                        sprintf(msg, "$%04d:LEFT;#\r\n", msg_id);
                        write(fd, msg, strlen(msg));
                    } else {
                        printf("%c User can't leave, not entered. (%ld M,
                        %ld F)\n", 'F', m_counter, f_counter);
                        sprintf(msg, "$%04d:ERR ;msg=%s#\r\n", msg_id,
                        "Can't leave, not entered");
                        write(fd, msg, strlen(msg));
                    }
                break;
                default:
                    sprintf(msg, "$%04d:ERR ;msg=%s#\r\n", msg_id,
                  "Unhandled gender");
                    write(fd, msg, strlen(msg));
                break;
            }

            pthread_mutex_unlock(&counters_lock);
        }
        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;
}

 /* Main */
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(&counters_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...");
    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");
}
```

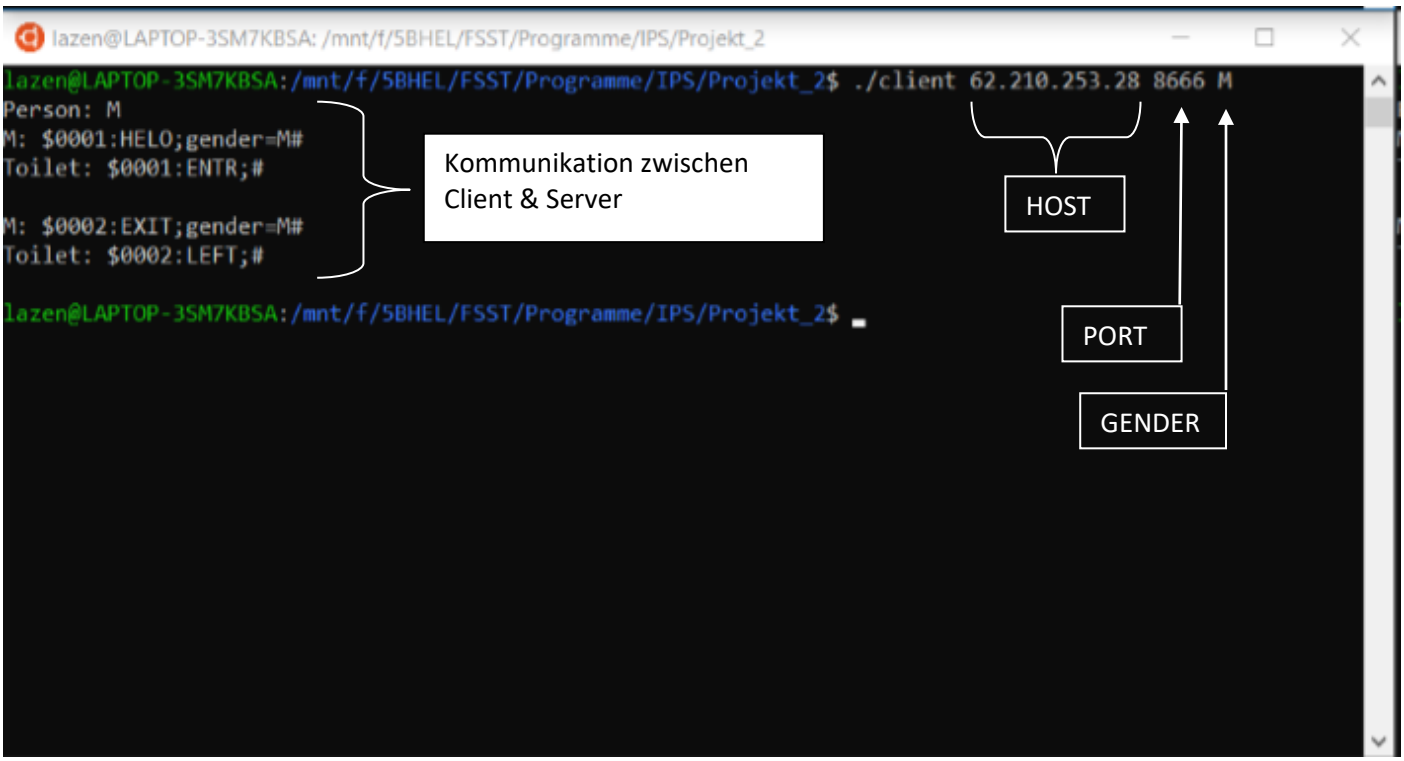


6. Funktionsnachweis

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

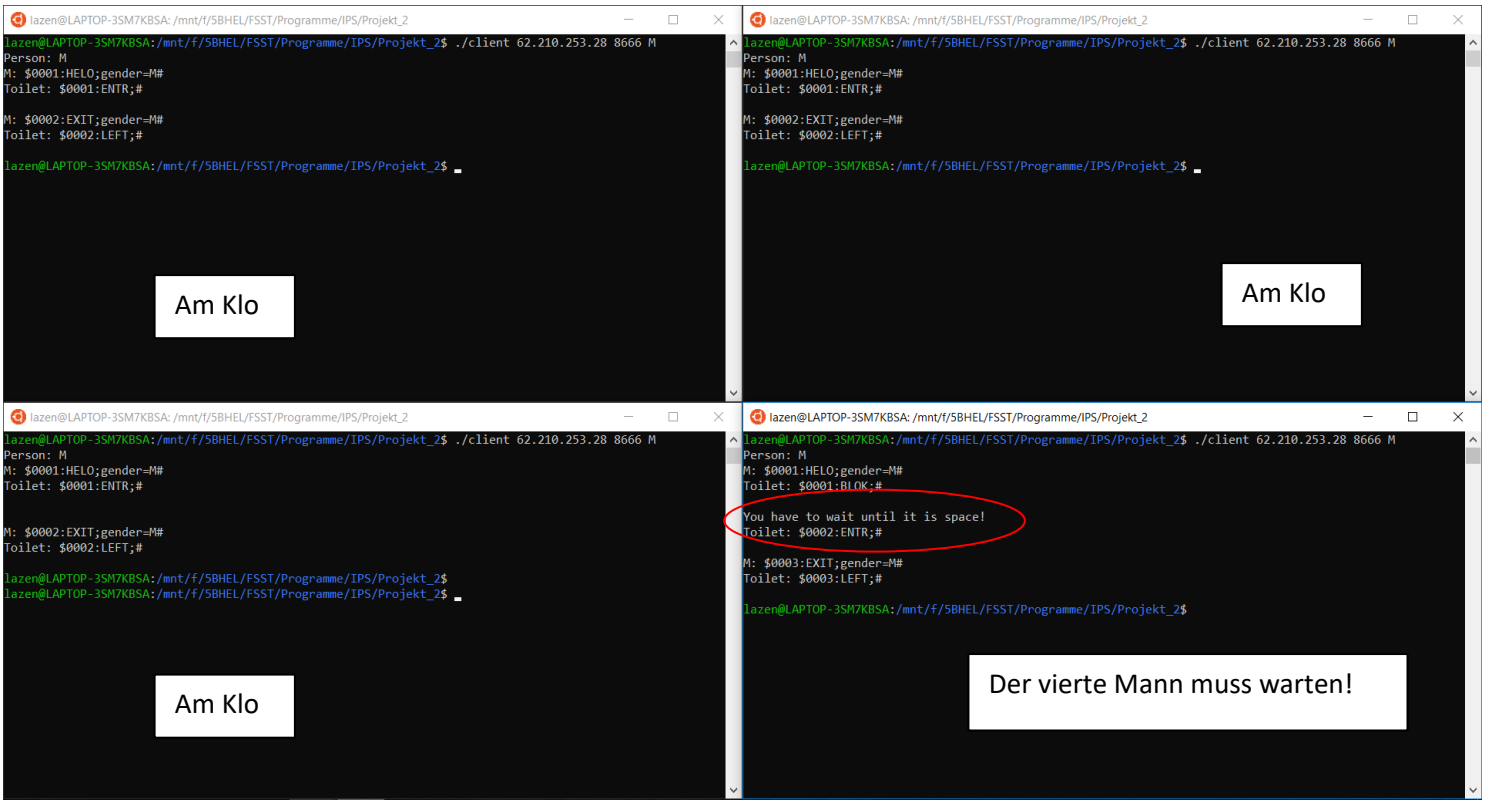
6.1. Tests

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:

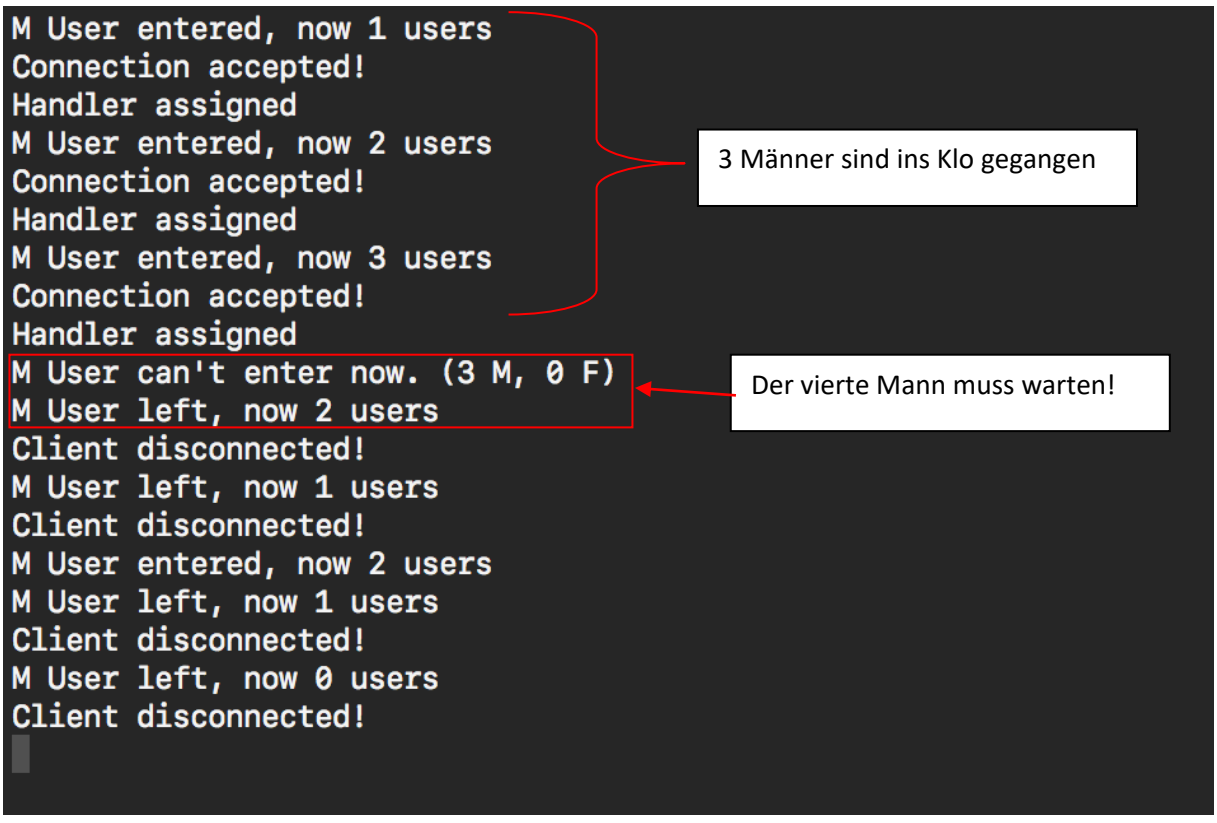


Danach wurde getestet ob wirklich immer nur drei Personen mit dem gleichen Geschlecht auf dem Klo sein können:

CLIENT:

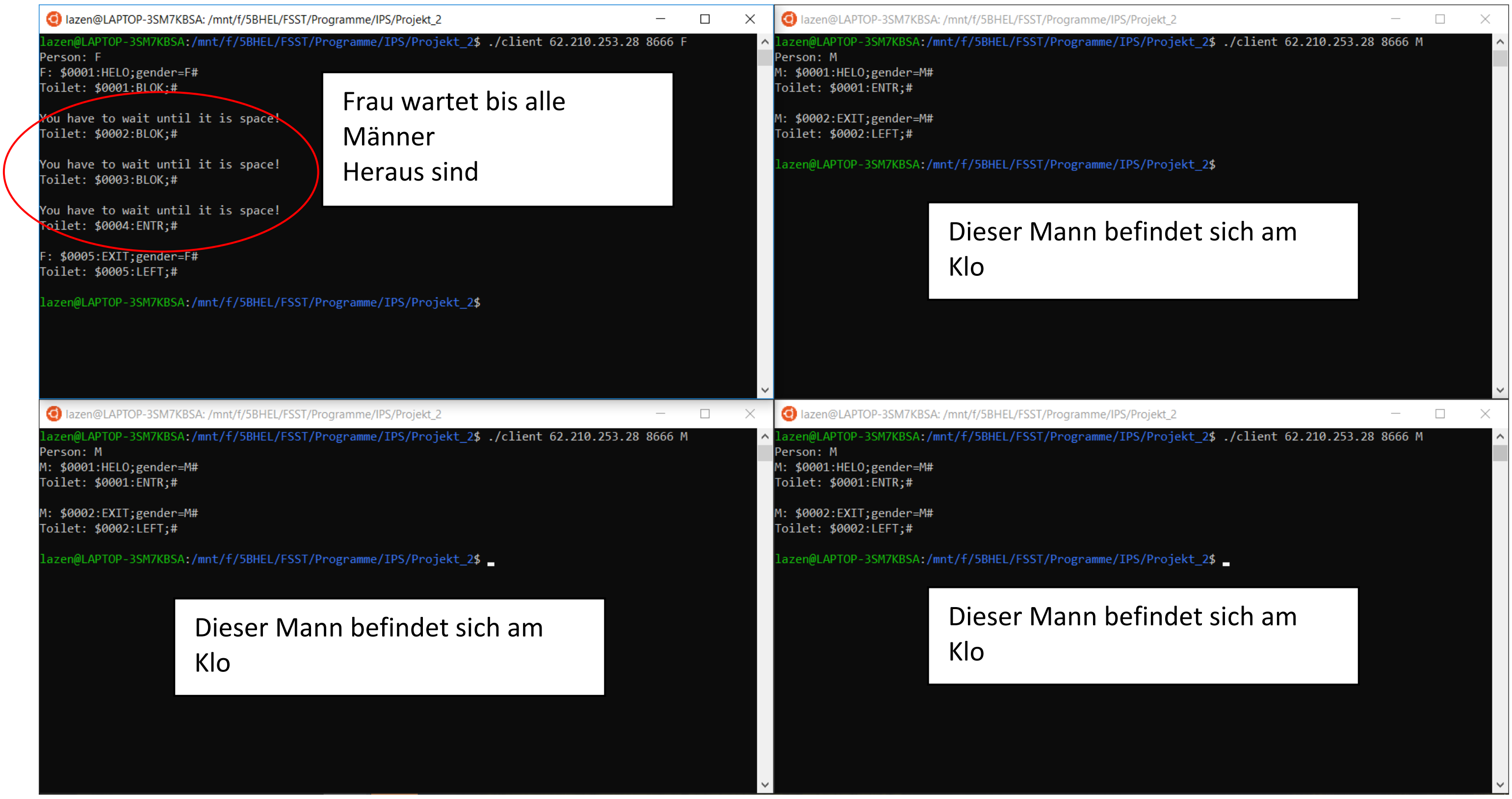


SERVER:



Im Anschluss darauf wurde überprüft ob eine Frau aufs Klo gehen kann wenn bereits Männer sich auf dem Klo befinden:

CLIENT:



SERVER: