Arquitetura

server.c

O server.c ficou com a mesma arquitetura do arquivo de server multi-thread disponibilizado no moodle. Contudo, eu precisei alterar a lógica contida dentro da thread para fazer o tratamento das funções requisitadas pelo trabalho. Além disso, foi necessário criar dois vetores globais para armazenar tanto os equipamentos quanto os sockets de cada equipamento conectado ao servidor.

```
19  int equip_vector[MAX] = {0}; // global equipment array
20  int csock_vector[MAX] = {0}; // global socket array
21  int count[1] = {0};          // increments a global id
```

Além disso, dentro da thread do servidor ocorre o envio de uma mensagem, de confirmação de adição no array de equipamentos, para o equipamento conectado e um faz broadcast para os outros. Só após esse envio é que é feita a entrada no while para manter a conexão com o servidor. (Obs: o tratamento para verificar se o limite de equipamentos conectados foi extrapolado é feito no else deste if).

```
for (int i = 0; i < MAX; i++)
{
    if (i == index - 1)
    {
        send(csock_vector[index - 1], buf, strlen(buf), 0); // buf had his value changed inside add_equip()
    }                                                       // sends buf just to the equipment that has been just
    else if (csock_vector[i] != 0)                          // added
    {
        memset(aux, 0, BUFSZ);
        sprintf(aux, "Equipment %.2d added", equip_vector[index - 1]);
        send(csock_vector[i], aux, strlen(aux), 0); // sends aux to all equipments connected with server, but the
    }                                               // one which were added
}
```

Dentro do while, é um programa simples que recebe o conteúdo vindo do socket e em seguida faz o tratamento dessa string recebida.

```
70      while (1)
71      {
72          recv(cdata->csock, buf, BUFSZ - 1, 0);
73          strcpy(aux, buf);
74          memset(buf, 0, BUFSZ);
75          strncpy(buf, aux, strlen(aux) - 1); // removes '/0' in the end of buf string
76
77          if (handle_buf(buf, equip_vector, csock_vector, MAX, index) < 0)    // handles each command client sent
78          {
79              pthread_exit(EXIT_SUCCESS);
80          }
81          memset(buf, 0, BUFSZ);
82          memset(aux, 0, BUFSZ);
83      }
```

f_server.c

Este arquivo contém todas as funções de tratamento dos comandos enviados pelos equipamentos. As funções estão todas comentadas, irei colocar o print de cada uma delas e comentar alguns detalhes mais sutis.

add_equip()

```
20  int add_equip(char buf[BUFSZ], int *equip_vector, unsigned int n, int *index, int *count)
```

```
25      for (i = 0; i < n; i++) // run all equip_vector[MAX] values and when the equipment doesn't exist equip_vector[index] = 0
26      {
27          if (equip_vector[i] == 0) // when the first equip_vector[index] = 0 is found, we add the index value into equip_vector[index]
28          {
29              count[0]++;
30              equip_vector[i] = count[0];
31              *index = i + 1;
32              memset(buf, 0, BUFSZ);
33              sprintf(aux, "Equipment %.2d added", equip_vector[i]);
34              puts(aux); // prints it in server terminal
35
36              sprintf(buf, "New ID: %.2d", equip_vector[i]); // sends buf to send(buf, ...)
37              return 1;
38          }
39      }
40      return -1;
41  }
```

A variável count[0] é usada para incrementar os id's dos equipamentos. Isso foi necessário para que quando um equipamento fosse desconectado o seu id não fosse utilizado por nenhum outro equipamento que no futuro conecte.

int handle_buf()

```
int handle_buf(char buf[BUFSZ], int *equip_vector, int *csock_vector, unsigned int n, int index)
```

A única peculiaridade desta função é para o caso de quando o equipamento manda o comando "request information from id", uma vez que ocorre o tratamento da string para conseguir o id requisitado e convertê-lo para integer. Além disso, o for() desta função remove o espaço vazio presente na primeira posição da string.

```
    long int n_index = 0;
    char aux[BUFSZ];
    char *ptr[BUFSZ];
    *ptr = NULL;
    memset(aux, 0, BUFSZ);
    strcpy(aux, strrchr(buf, ' '));
    int count = 0;

    // Traverse the given string. If current character
    // is not space, then place it at index 'count++'
    for (int i = 0; i < strlen(aux); i++)
        if (aux[i] != ' ')
            aux[count++] = aux[i];
    aux[count] = '\0';

    n_index = atoi(aux); // get the id string that had some information requested converted to integer
    // get the string handled and it sends the index of the equipment that the client wants to know about
```

close_connection()

```
void close_connection(char buf[BUFSZ], int *equip_vector, int *csock_vector, unsigned int n, int index)
```

Nenhuma peculiaridade, a lógica é simples. A função envia a mensagem de confirmação para o vetor que requisitou o comando, faz um broadcast avisando aos outros equipamentos qual equipamento foi removido, fecha a conexão e zero os campos dos arrays de equipamento e socket referentes ao equipamento que pediu o “close connection”.

```
        sprintf(aux, "Equipment %.2d removed", equip_vector[index - 1]);
        puts(aux);              // prints the message in server terminal
        sprintf(buf, SUCCESS); // buf content will be sent to equipment using send()

        for (int i = 0; i < MAX; i++)
        {
            if (csock_vector[i] != 0 && i == index - 1)
            {
                send(csock_vector[index - 1], buf, strlen(buf), 0);
            }
            else if (csock_vector[i] != 0 && i != index - 1)
            {
                send(csock_vector[i], aux, strlen(aux), 0);
            }
        }
        close(s);
        // zero those vectors of equipment and csock
        equip_vector[index - 1] = 0; // index is i + 1 in add_equip()/add_cscock(), so equip_vector[i] == equip_vector[index-1]
        csock_vector[index - 1] = 0;
```

request_equipment()

```
void request_equipment(char buf[BUFSZ], int *equip_vector, int *csock_vector, unsigned int n, long int n_index, int index)
```

Roda todo o vetor de equipamentos e envia uma mensagem para o equipamento que mandou o comando e uma outra mensagem para o equipamento que teve a informação requisitada. Caso não entre nesse if(), o comando de equipamento não encontrado é enviado.

```
for (int i = 0; i < n; i++)
{
    if (equip_vector[i] == n_index && equip_vector[i] != 0)
    {
        memset(buf, 0, BUFSZ);
        sprintf(buf, "Value from %.2d: %.2f", equip_vector[i], ((float)rand() / (float)(RAND_MAX)) * 10);
        send(csock_vector[index - 1], buf, strlen(buf), 0); // sends a message to the terminal that requested an information
        memset(aux, 0, BUFSZ);
        sprintf(aux, REQUEST_RESPONSE);
        send(csock_vector[i], aux, strlen(aux), 0); // sends a message to the terminal that had some information requested

        return;
    }
}
```

```
    sprintf(aux, "Equipment %.2ld not found", n_index);
    puts(aux);              // prints into server terminal
    sprintf(buf, TARGET); // sends it to send(buf, ...)
    send(csock_vector[index - 1], buf, strlen(buf), 0);
```

list_equipments()

```
void list_equipments(char buf[BUFSZ], int *equip_vector, int *csock_vector, unsigned int n, int index)
```

Função mais peculiar, precisei criar um método para ordenar o vetor de equipamentos para printar a mensagem com os id's em ordem crescente. Não vi nada disso documentado no TP, mas achei que era necessário. Todos os for()'s estão comentados e indicam claramente o que está sendo feito.

```
for (int i = 0; i < n; i++) // will be used to order the equip vector because it must be printed in order
{
    if (i != index - 1)
    {
        numero[i] = equip_vector[i];
    }
}
```

```
for (int i = 0; i < n; i++) // order the aux vector in crescent order
{
    for (int j = i + 1; j < n; j++)
    {
        if (numero[i] > numero[j])
        {
            aux1 = numero[i];
            numero[i] = numero[j];
            numero[j] = aux1;
        }
    }
}
```

```
for (int i = 0; i < n; i++)
{
    if (numero[i] != 0 && i != index - 1)
    {
        memset(aux, 0, BUFSZ);
        sprintf(aux, "%.2d ", numero[i]); // prints the equipments in crescent order
        strcat(buf, aux);                 // concat all existed equipments excludind the equipment that sent the command
    }
}
```

Além disso, criei uma resposta para um caso de exceção que ocorreu enquanto eu testava meu programa. No caso de só existir um equipamento conectado ao servidor e ele fazer pedir o comando "list equipment", a mensagem de erro abaixo é printada na tela.

```
#define EMPTY "Empty list "
```

```
if (strlen(buf) == 0)
{
    sprintf(buf, EMPTY); // if there is just one equipment connected in the server and this equipment send
}                        // "list equipment" command, it will prints in equipment's terminal this message

memset(aux, 0, BUFSZ);
strncpy(aux, buf, strlen(buf) - 1); // removes the blank space in the end of the string
memset(buf, 0, BUFSZ);
strcpy(buf, aux);
send(csock_vector[index - 1], buf, strlen(buf), 0);
```

server.c

```
while (1)
{
    countr = recv(csock, buf, BUFSZ, 0);
    strcpy(aux, buf);
    strncpy(buf, aux, strlen(aux)-1); //eliminate '/0'
```

equipments.c

Para fazer o broadcast eu criei uma thread dentro do arquivo equipments.c e dentro dela coloquei o recv() em loop infinito. Dessa forma, consegui “liberar” o fgets() e o recv().

```
void *client_thread(void *data)
{
    while (1)
    {
        char buf[BUFSZ];
        memset(buf, 0, BUFSZ);
        char aux[BUFSZ];
        memset(aux, 0, BUFSZ);
        struct client_data *cdata = (struct client_data *)data;
        recv(cdata->csock, buf, BUFSZ, 0);
        if (strcmp(buf, SUCCESS) == 0) // verifies if the server removed the equipment from the array
        {
            puts(buf);
            close(cdata->csock);
            exit(EXIT_SUCCESS);
        }
        if (strlen(buf) != 0) // just prints if has something in buffer
        {
            strcpy(aux, buf);
            strncpy(buf, aux, strlen(aux)-1); //eliminate '/0'
            puts(buf);
        }
    }
}
```

Além disso, dentro da main() e antes da entrada do while() eu coloquei um recv() que terá o papel de receber a mensagem de confirmação do servidor para quando o equipamento for adicionado no “banco de dados”

```
recv(s, buf, BUFSZ, 0);
if (strcmp(buf, LIMIT) == 0)
{
    puts(buf);
    close(s);
    exit(EXIT_SUCCESS);
}

puts(buf);
```

Já dentro do while, primeiro eu chamei a thread e logo em seguida o fgets(). Quando invertia as funções o programa não rodava da forma como precisava.

```
while (1)
{
    struct client_data *cdata = malloc(sizeof(*cdata));
    if (!cdata)
    {
        logexit("malloc");
    }
    cdata->csock = s;

    pthread_t tid;
    pthread_create(&tid, NULL, client_thread, cdata); // calls a thread that will be running recv() until the client asks to close connection
    memset(buf, 0, BUFSZ);
    memset(aux, 0, BUFSZ);
    fgets(buf, BUFSZ - 1, stdin); // get a string from the client
    counts = send(s, buf, strlen(buf), 0); // sends the buffer to the server
    if (counts != strlen(buf))
    {
        logexit("send");
    }
    memset(buf, 0, BUFSZ);
}
```

Contudo, o terminal dos meus equipamentos apresentou uma exceção que eu não consegui entender o porquê ocorre e nem consegui replicar o bug. Isso ocorre quando o mesmo equipamento envia o comando “request information from same_id”, ou seja, quando o equipamento com id = 01 envia o comando “request information from 01”.

```
vitor@vitor-Aspire-A515-52G:~/Documents/UFMG/Redes/Computer-Network-Project2-UFMG$ ./equipment
 127.0.0.1 5152
New ID: 01
Equipment 02 added
list equipments
02
Equipment 03 added
list equipments
02 03
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56requested information
request information from 01
Value from 01: 2.56
requested information
request information from 01
Value from 01: 2.56
```

```
vitor@vitor-Aspire-A515-52G:~/Documents/UFMG/Re
des/Computer-Network-Project2-UFMG$ ./equipment
 127.0.0.1 5152
New ID: 01
Equipment 02 added
Equipment 03 added
request information from 01
Value from 01: 8.40
requested information
request information from 01
Value from 01: 3.94requested information
request information from 01
Value from 01: 7.83
requested information
request information from 01
Value from 01: 7.98
requested information
request information from 01
Value from 01: 9.12
requested information
request information from 01
Value from 01: 1.98
requested information
request information from 01
Value from 01: 3.35
requested information
request information from 01
Value from 01: 7.68
requested information
request information from 01
Value from 01: 2.78
requested information
request information from 01
Value from 01: 5.54
requested information
request information from 01
Value from 01: 4.77
requested information
```