Food Waste Reducer

Olaru Marius-Alexandru[1][310910401RSL221156]

Alexandru Ioan Cuza University, Faculty of Computer Science, 22, Carol I Boulevard, Iaşi RO – 700505

Abstract. Worldwide food waste severely affects resources and the environment. About 1/3 of global food production, equivalent to 1.3 billion tons, is wasted annually. This problem generates significant greenhouse gas emissions and economic costs in the billions of dollars. Food waste contributes to hunger and depletes natural resources such as water and soil. Solutions include improving the supply chain, promoting responsible consumption and adopting innovative technologies to address this global problem. The Food Waste Reducer project contributes to preventing this phenomenon and improving the environment, organizations such as the World Food Program estimate that, in some cases, these initiatives can reduce food waste by up to 20-50% . In conclusion, such an application could help save a significant percentage of food that would traditionally have been wasted, with substantial social and economic benefits.

Keywords: Food · Waste · Economic benefits.

1 Applied technologies

The application is developed based on the client/server architecture in the C/C++ language. The connection between the client and the server is based on the transport protocol connection oriented, without loss of information, TCP - Transmission Control Protocol. The characteristics of this protocol are safety and security. As the application aims at the transfer between two clients, through the server, of less public information, this implies that the risk of information loss increases significantly, but also the risk of capturing this information by external users. One of the information sent by the Restaurant/Shop will be its location for taking over the donation and, of course, we would not want to put the charity organizations or people in need at risk and reach a wrong destination, where various incidents may occur.

That's why the TCP protocol will ensure us, by transferring information correctly, that we provide all the necessary information without endangering anyone.

2 Application structure

The application is built from a server and two clients, one of the clients will be assigned to charities/the needy, and the other to restaurants/stores, both sending and receiving various services, while the server will monitor and distribute the information of each client.

Server Features The server is assigned the role of central coordinator for managing donation and request information.

Its responsibilities include storing and updating information about food available in stores and restaurants, managing donations and assigning them to charities or people in need of food.

Client Features Clients will be assigned the role of connecting to the server.

Responsibilities of the restaurant/store client consist of registering and updating the available food supply, sending donation requests to the server and managing the donation process. During this time the charity/persons in need client has the responsibilities of viewing available food offers, sending food requests to the server and managing the receipt of donations.

The Interaction Process

Shops and Restaurants:

1. Connects to the server to report available food.
2. Receives requests for donations from charities or people in need.
3. Confirms and manages the donation process.

Charities or Persons in Need:

1. Connects to the server to view available food offers.
2. Sends out donation requests for the desired food.
3. Receives confirmation and donation details from stores or restaurants.

Server:

1. Gets up-to-date information from stores and restaurants about food deals.
2. Processes requests for donations from charities or individuals in need.
3. Manages donation allocation and confirmation.

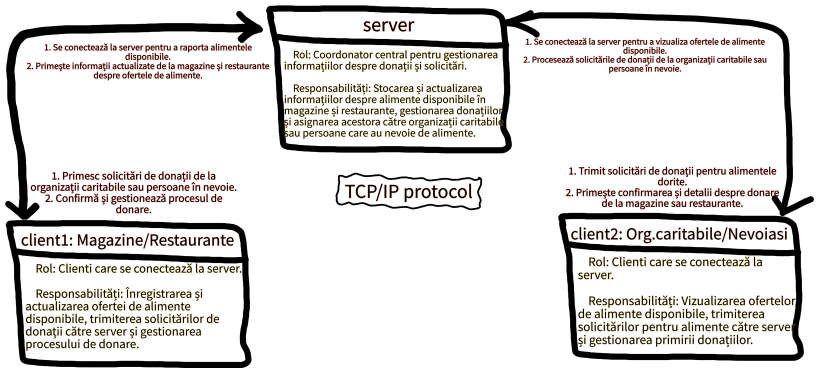

server
Rol: Coordonator central pentru gestionarea informaţiilor despre donaţii şi solicitări.
Responsabilităţi: Stocarea şi actualizarea informaţiilor despre alimente disponibile în magazine şi restaurante, gestionarea donaţiilor şi asignarea acestora către organizaţii caritabile sau persoane care au nevoie de alimente.
1. Se conectează la server pentru a raporta alimentele disponibile.
2. Primeşte informaţii actualizate de la magazine şi restaurante despre ofertele de alimente.
1. Se conectează la server pentru a vizualiza ofertele de alimente disponibile.
2. Procesează solicitările de donaţii de la organizaţii caritabile sau persoane în nevoie.
TCP/IP protocol
1. Primesc solicitări de donaţii de la organizaţii caritabile sau persoane în nevoie.
2. Confirmă şi gestionează procesul de donare.
1. Trimit solicitări de donaţii pentru alimentele dorite.
2. Primeşte confirmarea şi detalii despre donare de la magazine sau restaurante.
client1: Magazine/Restaurante
Rol: Clienti care se conectează la server.
Responsabilităţi: Înregistrarea şi actualizarea ofertei de alimente disponibile, trimiterea solicitărilor de donaţii către server şi gestionarea procesului de donare.
client2: Org.caritabile/Nevoiasi
Rol: Clienti care se conectează la server.
Responsabilităţi: Vizualizarea ofertelor de alimente disponibile, trimiterea solicitărilor pentru alimente către server şi gestionarea primirii donaţiilor.

Fig. 1: Here we have a graphic representation of what was said above.

3 Implementation Aspects

As expected, we have at our disposal two C/C++ codes, each representing the server and the client, respectively. The client.c code implements the TCP/IP communication protocol with the server and communication with it, thus obtaining the information initially requested by the client. Time in which the server.c code implements the concurrent TCP server and the communication protocol, recognizing the orders of each client, but also the type of client (charitable organization/person in need or restaurant/shop)

To be more specific, I will present in more detail essential and representative sections of code for switching between roles, i.e. client and server.

Client Implementation As I mentioned above, one of the essential roles of the client is the implementation of the communication protocol.

This involves receiving the login details from the application user, more precisely the IP address of the server, the port and the type of client it belongs to. Through the received details, the client code will create the client-server connection using the related structure of the server through the lines of code below. [fig 2]

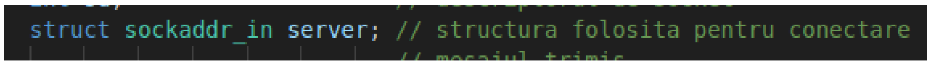
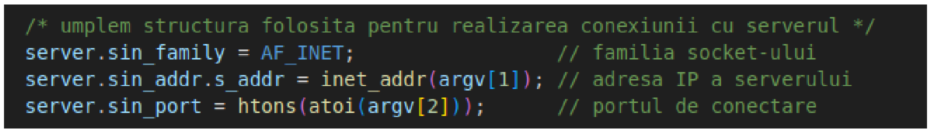

```
struct sockaddr_in server; // structura folosita pentru conectare
```

Fig. 2: Here we have the lines of code that represent the population of the server structure.

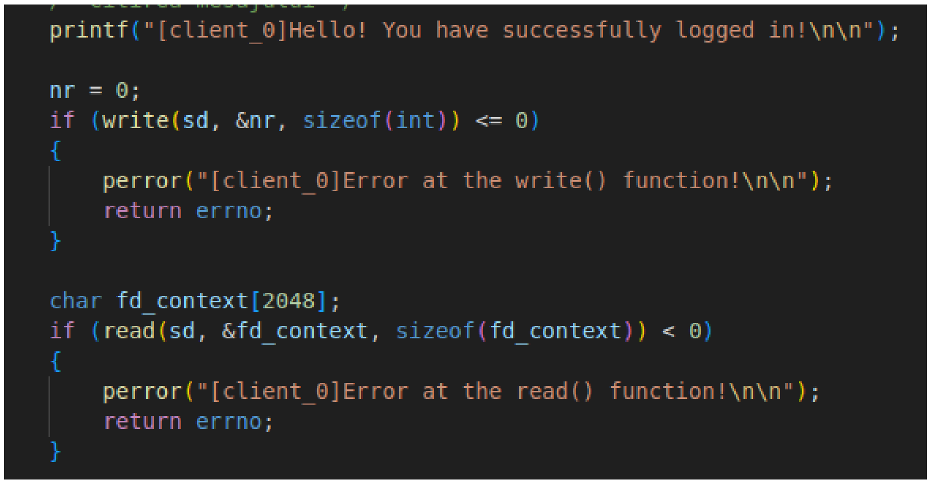
After populating the server structure, the client type is checked, so the following commands and messages will differ depending on each type. [fig 3]



```
/* umplem structura folosita pentru realizarea conexiunii cu serverul */
server.sin_family = AF_INET;                // familia socket-ului
server.sin_addr.s_addr = inet_addr(argv[1]); // adresa IP a serverului
server.sin_port = htons(atoi(argv[2]));      // portul de conectare
```

Fig. 3: Here we have the declaration of the server structure.

Furthermore, after receiving the various information necessary to connect to the server, the actual connection will be made. [fig4]



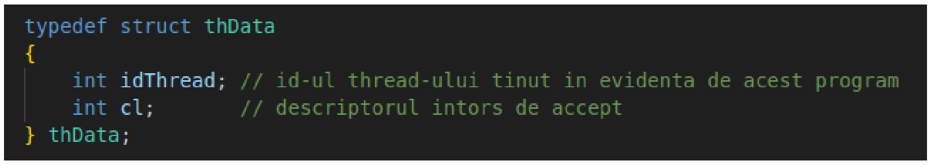
```
    printf("[client_0]Hello! You have successfully logged in!\n\n");

    nr = 0;
    if (write(sd, &nr, sizeof(int)) <= 0)
    {
        perror("[client_0]Error at the write() function!\n\n");
        return errno;
    }

    char fd_context[2048];
    if (read(sd, &fd_context, sizeof(fd_context)) < 0)
    {
        perror("[client_0]Error at the read() function!\n\n");
        return errno;
    }
```

Fig. 4: Here we have the lines of code that connect the current client to the server.

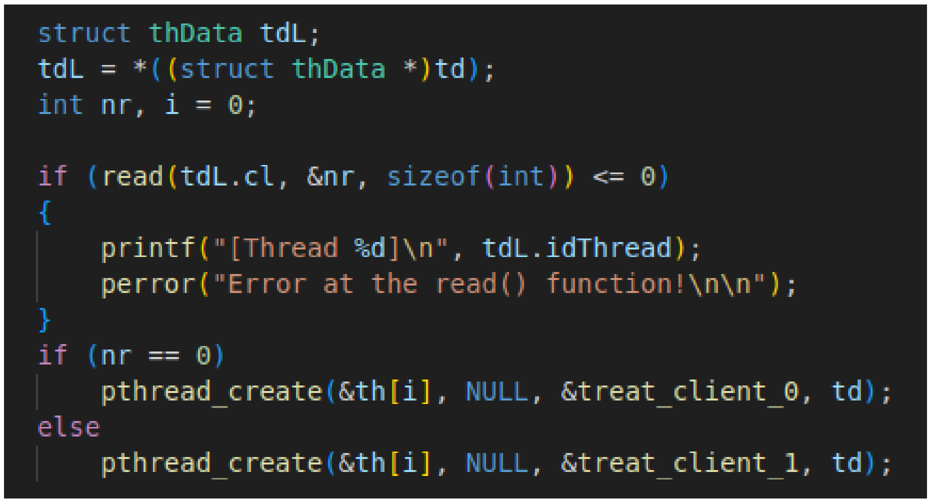
Server Implementation Also, the server code comes with its own specific implementations. Thus, fulfilling one of its tasks, more precisely the implementation of concurrency, the server.c code calls on the concept of threads. [fig 5]



```c
typedef struct thData
{
    int idThread; // id-ul thread-ului tinut in evidenta de acest program
    int cl;       // descriptorul intors de accept
} thData;
```

Fig. 5: Here we have the structure of threads.

Basically what this code does is to create a thread for each connected client, thus, through these threads, the client-server connection will be made, as a result the required tasks are completed. Thus, in order for each thread to execute its task, the main Thread is blocked at the call to accept() and by each when a client is accepted it is created a thread that will serve him. [fig 6]

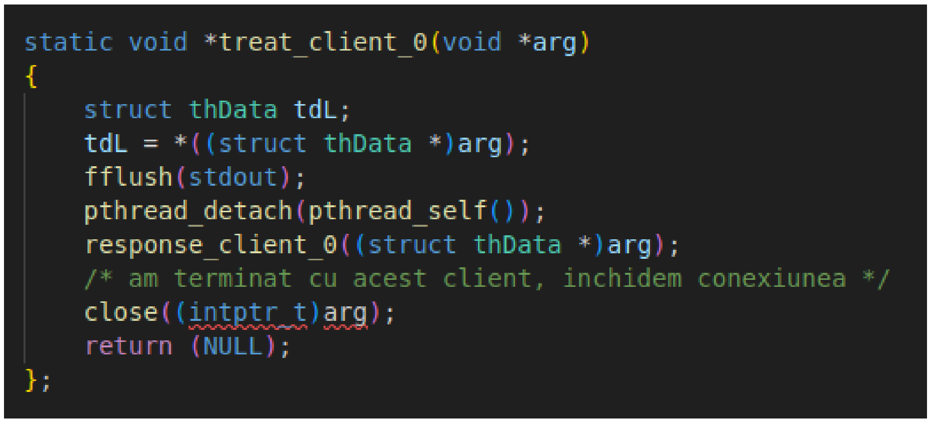


```c
struct thData tdL;
tdL = *((struct thData *)td);
int nr, i = 0;

if (read(tdL.cl, &nr, sizeof(int)) <= 0)
{
    printf("[Thread %d]\n", tdL.idThread);
    perror("Error at the read() function!\n\n");
}
if (nr == 0)
    pthread_create(&th[i], NULL, &treat_client_0, td);
else
    pthread_create(&th[i], NULL, &treat_client_1, td);
```

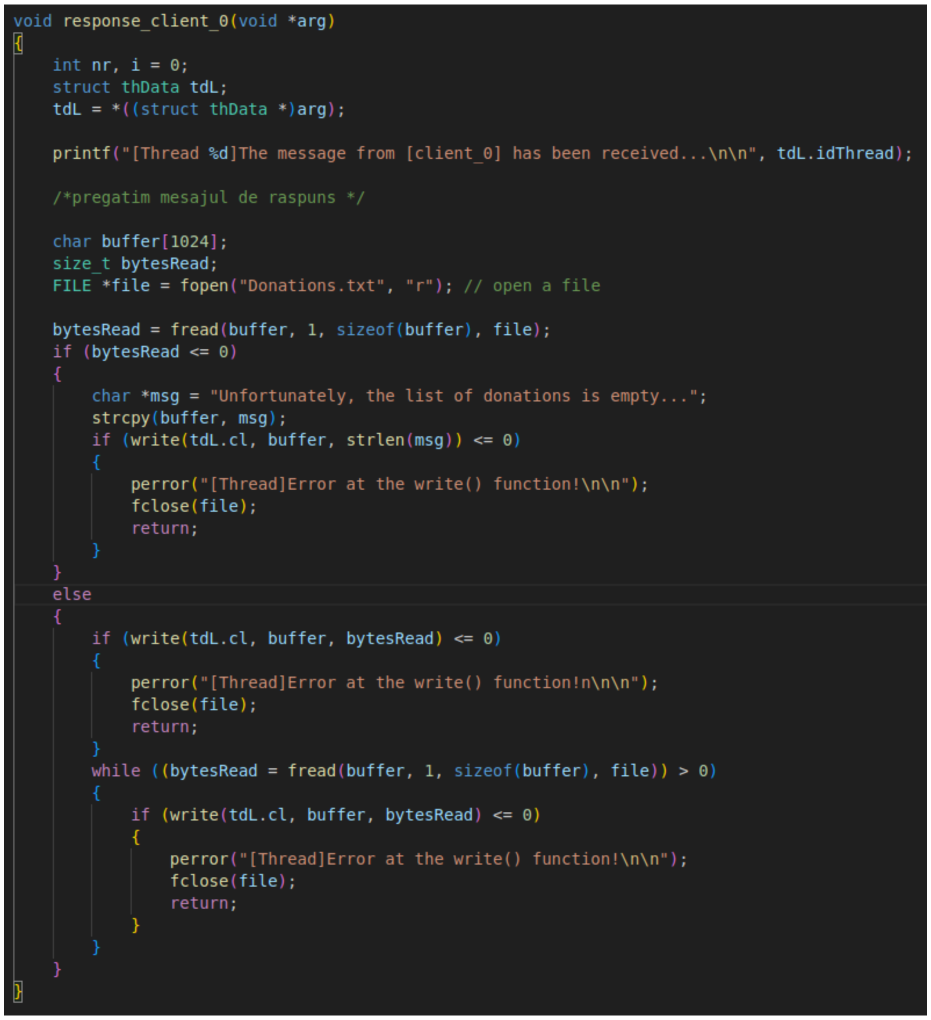
Fig. 6: Here we have the lines of code that associate each thread with its specific function.

Depending on the type of client, functions specific to each client will be called, then these threads will be completed. [fig 7 - 8]



```c
static void *treat_client_0(void *arg)
{
    struct thData tdL;
    tdL = *((struct thData *)arg);
    fflush(stdout);
    pthread_detach(pthread_self());
    response_client_0((struct thData *)arg);
    /* am terminat cu acest client, inchidem conexiunea */
    close((intptr_t)arg);
    return (NULL);
};
```

Fig. 7: Here we have the functions specific to the client of a charitable organization/person in need.



```c
void response_client_0(void *arg)
{
    int nr, i = 0;
    struct thData tdL;
    tdL = *((struct thData *)arg);

    printf("[Thread %d]The message from [client_0] has been received...\n\n", tdL.idThread);

    /*pregatim mesajul de raspuns */

    char buffer[1024];
    size_t bytesRead;
    FILE *file = fopen("Donations.txt", "r"); // open a file

    bytesRead = fread(buffer, 1, sizeof(buffer), file);
    if (bytesRead <= 0)
    {
        char *msg = "Unfortunately, the list of donations is empty...";
        strcpy(buffer, msg);
        if (write(tdL.cl, buffer, strlen(msg)) <= 0)
        {
            perror("[Thread]Error at the write() function!\n\n");
            fclose(file);
            return;
        }
    }
    else
    {
        if (write(tdL.cl, buffer, bytesRead) <= 0)
        {
            perror("[Thread]Error at the write() function!n\n\n");
            fclose(file);
            return;
        }
        while ((bytesRead = fread(buffer, 1, sizeof(buffer), file)) > 0)
        {
            if (write(tdL.cl, buffer, bytesRead) <= 0)
            {
                perror("[Thread]Error at the write() function!\n\n");
                fclose(file);
                return;
            }
        }
    }
}
```

Fig. 8: Here we have the functions specific to the client of a charitable organization/person in need.

4 Concluzions

There will always be room for improvement, as a result an improvement of the Food Waste Reducer project can be brought to the interface level. Thus, the customer who accesses the application must first authenticate as a restaurant/store or as a charity organization/person in need through an easy-to-use menu, this detail will help assign the chosen role to each user, as well as role-specific options . Likewise, another idea for improvement is based on the details offered and transported between customers. That is, the accuracy of the location of the shops/restaurants, the expiration date of the donated products, the net weight of the donated package.

References

1. Computer Networks Homepage, https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php.