**Universiteti i Prishtinës**

**Fakulteti Inxhinierisë Elektrike dhe Kompjuterike**

****

**Dokumentim teknik i projektit**

**Lënda: Sisteme operative**

**Titulli i projektit: Multithreaded Client-Server Application**

**Emri profesorit Emri & mbiemri studentëve / email adresa**

|  |  |  |
| --- | --- | --- |
| Prof. Artan Mazrekaj  Ass. Dalinë Vranovci | 1. Diellza Raçi | [diellza.raci1@student.uni-pr.edu](mailto:diellza.raci1@student.uni-pr.edu) |
| 2. Festina Mjeku | [festina.mjeku@student.uni-pr.edu](mailto:festina.mjeku@student.uni-pr.edu) |
| 3. Rina Halili | [rina.halili@student.uni-pr.edu](mailto:rina.halili@student.uni-pr.edu) |

Prishtinë, 2023

Përmbajtja

[**I.** **Hyrje** 3](#_Toc136481853)

[**II.** **Detyra me kerkesat e dhena** 4](#_Toc136481854)

[**III.** **Realizimi i aplikacionit** 5](#_Toc136481855)

[**1.** **Pjesa e kodit** 5](#_Toc136481856)

[1.1 Kodi i serverit 5](#_Toc136481857)

[1.2 Kodi i klientit 8](#_Toc136481858)

[**2.** **Pamjet ne terminal** 10](#_Toc136481859)

[**3.** **Plotësimi i kërkesave të detyrës** 13](#_Toc136481860)

[Detyrat e serverit: 13](#_Toc136481861)

[Detyrat e klientit: 13](#_Toc136481862)

[**IV.** **Përfundim** 14](#_Toc136481863)

# **Hyrje**

Ky aplikacion është një sistem komunikimi klient-server multithread-ësh i implementuar në gjuhën C duke përdorur message queues.

Duke përdorur message queues për komunikim midis proceseve, serveri dhe klienti shkëmbejnë mesazhe asinkronisht dhe të kanë operacione të pavarura. Serveri me shumë thread-e trajton njëkohësisht klientë të shumtë të lidhur, duke siguruar një komunikim efikas dhe të shpejtë. Klienti mund të ndërveprojë me serverin duke dërguar kërkesa dhe duke marrë përgjigje nga dëgjimi i message queues të tij.

# **Detyra me kerkesat e dhena**

**Project Description: Multithreaded Client-Server Application**

You will design and implement a multithreaded client-server application using message passing between processes. The server will accept connections from multiple clients and maintain a list of connected clients. Each client can send requests to the server, which will process them and send back responses.

**Basic Requirements**

The server has the following tasks:

1. Initialize a message queue for accepting connections from clients.
2. Listen for connections from clients and add them to a list of connected clients.
3. For each connected client, spawn a new thread that listens for messages from that client and processes them.
4. Each message from a client should contain a request type and data payload. The server should process each request and generate a response, which should be sent back to the client using its message queue.
5. When a client disconnects, its message queue should be removed from the list of connected clients and the corresponding thread should be terminated.

The client has the following tasks:

1. Initialize a message queue for communicating with the server.
2. Connect to the server by sending a connection request to its message queue.
3. Send requests to the server by sending messages to its message queue.
4. Wait for responses from the server by listening to its message queue.
5. Gracefully disconnect from the server by sending a disconnect request to its message queue.

# **Realizimi i aplikacionit**

## **Pjesa e kodit**

### Kodi i serverit

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <pthread.h>

#include <string.h>

#include <sys/types.h>

#define SERVER\_KEY 1234

#define CLIENT\_KEY 5678

#define MAX\_CLIENTS 10

typedef struct {

    long mtype;

    char data[256];

    pid\_t pid;

} Message;

typedef struct {

    pthread\_t thread;

    int connected;

    pid\_t client\_pid;

    int client\_queue\_id;  // Shto fushen client\_queue\_id per te ruajtur queue ID te klientit

} ClientInfo;

ClientInfo connected\_clients[MAX\_CLIENTS];

pthread\_mutex\_t clients\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

void\* client\_handler(void\* arg) {

    int client\_queue\_id = \*((int\*)arg);

    Message request, response;

    ssize\_t msg\_size;

    pid\_t client\_pid = getpid();

    ClientInfo client\_info;

    pthread\_mutex\_lock(&clients\_mutex);

    for (int i = 0; i < MAX\_CLIENTS; i++) {

        if (connected\_clients[i].thread == pthread\_self()) {

            client\_info = connected\_clients[i];

            break;

        }

    }

    pthread\_mutex\_unlock(&clients\_mutex);

    while ((msg\_size = msgrcv(client\_queue\_id, &request, sizeof(Message) - sizeof(long), CLIENT\_KEY, 0)) > 0) {

        printf("Mesazhi i pranuar nga klienti (PID %d): %s\n", client\_info.client\_pid, request.data);

        if (strcmp(request.data, "disconnect") == 0) {

            printf("Klienti (PID %d) eshte diskonektuar\n", client\_info.client\_pid);

            break;

        }

        response.mtype = CLIENT\_KEY;

        strcpy(response.data, "Mesazhi i pranuar nga serveri");

        if (msgsnd(client\_queue\_id, &response, sizeof(Message) - sizeof(long), 0) == -1) {

            perror("msgsnd");

            break;

        }

    }

    pthread\_mutex\_lock(&clients\_mutex);

    for (int i = 0; i < MAX\_CLIENTS; i++) {

        if (connected\_clients[i].thread == pthread\_self()) {

            connected\_clients[i].thread = 0;

            connected\_clients[i].connected = 0;

            break;

        }

    }

    pthread\_mutex\_unlock(&clients\_mutex);

    msgctl(client\_queue\_id, IPC\_RMID, NULL);  // Fshij message queue te klientit

    pthread\_detach(pthread\_self());

    return NULL;

}

int main() {

    int server\_queue\_id;

    if ((server\_queue\_id = msgget(SERVER\_KEY, IPC\_CREAT | 0666)) == -1) {

        perror("msgget");

        exit(1);

    }

    printf("Serveri eshte duke punuar\n");

    while (1) {

        Message connect\_msg;

        if (msgrcv(server\_queue\_id, &connect\_msg, sizeof(Message) - sizeof(long), 1, 0) == -1) {

            perror("msgrcv");

            exit(1);

        }

        if (strcmp(connect\_msg.data, "disconnect") == 0) {

            printf("Klienti (PID %d) eshte diskonektuar\n", connect\_msg.pid);

            continue;

        }

        int client\_queue\_id;

        if ((client\_queue\_id = msgget(CLIENT\_KEY, IPC\_CREAT | 0666)) == -1) {

            perror("msgget");

            exit(1);

        }

        pthread\_mutex\_lock(&clients\_mutex);

        int i;

        for (i = 0; i < MAX\_CLIENTS; i++) {

            if (!connected\_clients[i].connected) {

                connected\_clients[i].connected = 1;

                connected\_clients[i].client\_pid = connect\_msg.pid;

                connected\_clients[i].client\_queue\_id = client\_queue\_id;  // Ruaj queue ID te klientit

                break;

            }

        }

        pthread\_mutex\_unlock(&clients\_mutex);

        pthread\_create(&connected\_clients[i].thread, NULL, client\_handler, &client\_queue\_id);

    }

    msgctl(server\_queue\_id, IPC\_RMID, NULL);

    return 0;

}

### Kodi i klientit

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <sys/types.h>

#define SERVER\_KEY 1234

#define CLIENT\_KEY 5678

typedef struct {

    long mtype;

    char data[256];

    pid\_t pid;

} Message;

int main() {

    int server\_queue\_id, client\_queue\_id;

    if ((client\_queue\_id = msgget(CLIENT\_KEY, IPC\_CREAT | 0666)) == -1) {

        perror("msgget");

        exit(1);

    }

    if ((server\_queue\_id = msgget(SERVER\_KEY, 0)) == -1) {

        perror("msgget");

        exit(1);

    }

    Message connect\_msg;

    connect\_msg.mtype = 1;

    connect\_msg.data[0] = '\0';

    connect\_msg.pid = getpid();

    if (msgsnd(server\_queue\_id, &connect\_msg, sizeof(Message) - sizeof(long), 0) == -1) {

        perror("msgsnd");

        exit(1);

    }

    printf("Konektuar me server\n");

    while (1) {

        Message request;

        request.mtype = CLIENT\_KEY;

        request.pid = getpid();

        printf("Shkruaj kerkesen/mesazhin: ");

        fgets(request.data, sizeof(request.data), stdin);

        size\_t len = strlen(request.data);

        if (len > 0 && request.data[len - 1] == '\n') {

            request.data[len - 1] = '\0';

        }

        if (msgsnd(client\_queue\_id, &request, sizeof(Message) - sizeof(long), 0) == -1) {

            perror("msgsnd");

            exit(1);

        }

        Message response;

        if (msgrcv(client\_queue\_id, &response, sizeof(Message) - sizeof(long), 0, 0) == -1) {

            perror("msgrcv");

            exit(1);

        }

        printf("Pergjigja nga serveri: %s\n", response.data);

        if (strcmp(request.data, "disconnect") == 0) {

            printf("Diskonektuar nga serveri\n");

            break;

        }

    }

    Message disconnect\_msg;

    disconnect\_msg.mtype = 1;

    strcpy(disconnect\_msg.data, "disconnect");

    disconnect\_msg.pid = getpid();

    if (msgsnd(server\_queue\_id, &disconnect\_msg, sizeof(Message) - sizeof(long), 0) == -1) {

        perror("msgsnd");

        exit(1);

    }

    msgctl(client\_queue\_id, IPC\_RMID, NULL);

    return 0;

}

## **Pamjet ne terminal**

Në momentin kur fillon ekzekutimi i kodit të serverit, në terminal të serverit shfaqet njoftimi “*Serveri është duke punuar*”.

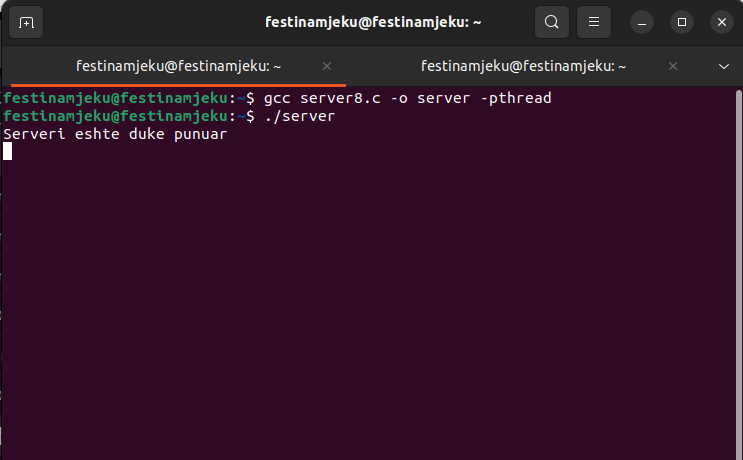


Fig 1. Pamja në terminalin e serverit kur ekzekutohet kodi

Kur të ekzekutohet kodi i klientit dhe ai të lidhet me server, në terminalin e klientit shfaqet një njoftim “*Konektuar me server*”. Pas këtij njoftimi, klientit i ofrohet mundësia për të derguar kërkesë apo mesazhe në server.

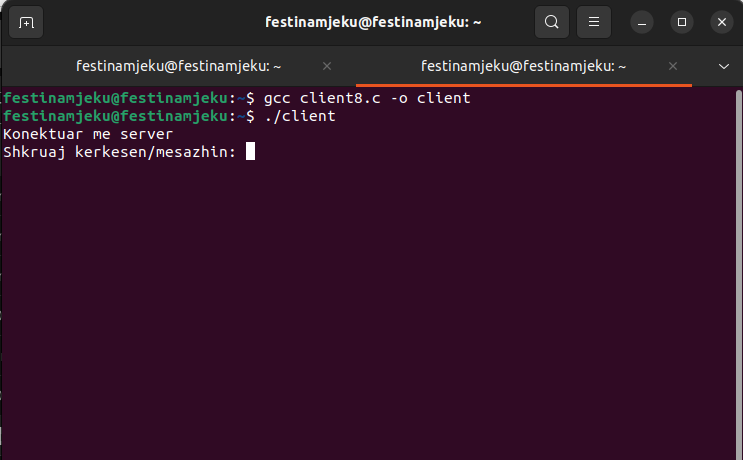


Fig 1. Pamja në terminalin e klientit kur ai lidhet me server

Klienti në këte rast shkruan një mesazh: “*hi*” dhe njoftohet se ky mesazh është pranuar nga serveri. Pas njoftimit, klientit i ofrohet të shkruajë kërkesa/mesazhe të reja.

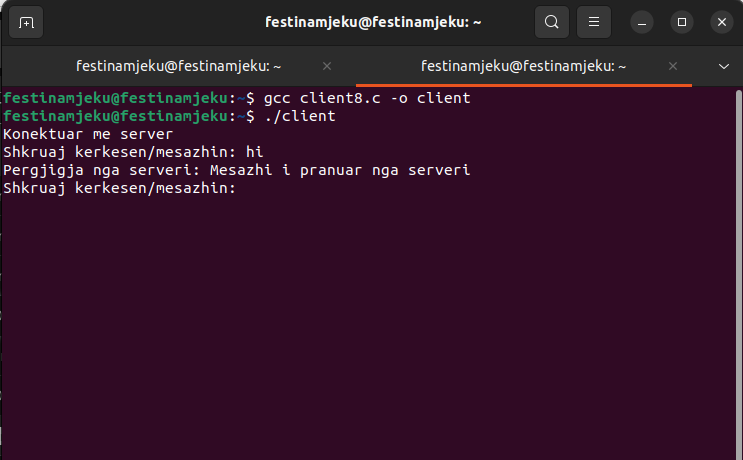


Fig 3. Klienti dërgon mesazhe në server

Mesazhi i pranuar nga klienti shfaqet tek terminali i serverit specifikisht me ID te klientit.

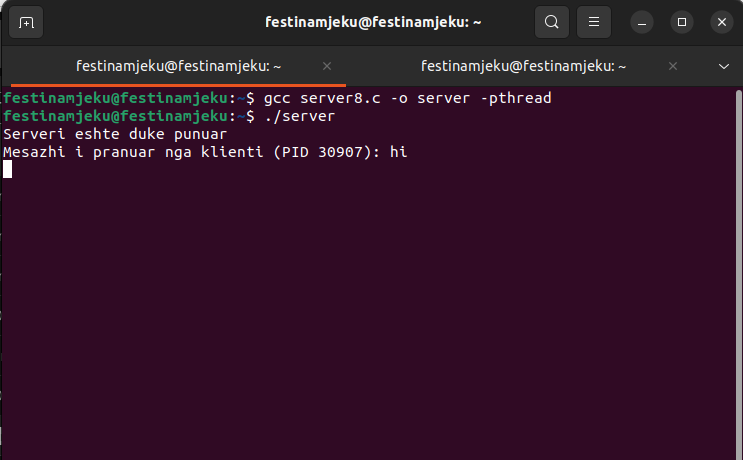


Fig 4. Mesazhi i pranuar nga klienti

Për t’u diskonektuar nga serveri, klienti shtyp kërkesën “*disconnect*”. Kur klienti diskonektohet me sukses, ai merr njoftimin që është fshirë nga lista e klientave.

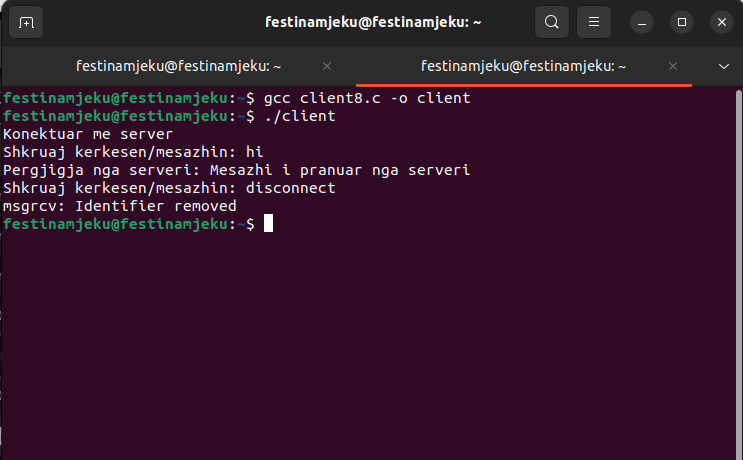


Fig 5. Klienti diskonektohet

Ndërsa tek terminali i serverit, pasi klienti të bëje kërkesën për diskonektim, ajo pranohet dhe shfaqet ID e klientit të diskonektuar.

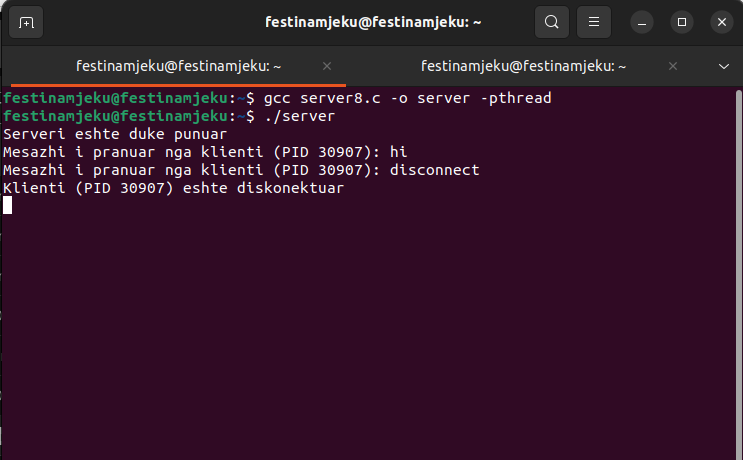


Fig 6. Serveri njofton diskonektimin e klientit

## **Plotësimi i kërkesave të detyrës**

Kjo detyrë është realizuar duke plotësuar të gjitha kërkesat e dhëna.

### Detyrat e serverit:

1. Inicializimi i message queues për pranim të konektimit të klientave: Serveri inicializon një message queue duke përdorur **msgget()** me **IPC\_CREAT flag**, i cili i lejon klientët të konektohen në server.
2. Pritja e konektimeve të klientëve dhe shtimi i tyre në një listë të klientëve të konektuar: Serveri pret mesazhe duke përdorur **msgrcv()** dhe I shton klientët e konektuar në vargun **connected\_clients**.
3. Për çdo klient të konektuar, të krijohet një thread të ri që pret për mesazhe nga ky klient dhe i përpunon ato: Për çdo klient të lidhur, serveri krijon një thread të ri duke përdorur funksionin **pthread\_create()** që menaxhon mesazhet nga ky klient.
4. Serveri duhet të përpunojë çdo kërkesë dhe të gjenerojë një përgjigje, e cila duhet të dërgohet prapa te klienti duke përdorur message queue: Serveri merr mesazhe nga klientët duke përdorur funksionin **msgrcv(**), përpunon kërkesën dhe dërgon përgjigjen prapa në message queue të klientit duke përdorur funksionin **msgsnd()**.
5. Kur një klient diskonektohet, message queue i tij duhet të hiqet nga lista e klientëve të konektuar dhe thread-i përkatës duhet të mbyllet: Kur një klient dërgon një mesazh "*disconnect*", message queue i tij hiqet duke përdorur funksionin **msgctl()** me komandën **IPC\_RMID**, dhe thread-i përkatës mbyllet duke përdorur funksionin **pthread\_detach()**.

### Detyrat e klientit:

1. Inicializimi i message queues për komunikim me serverin: Klienti inicializon një message queue duke përdorur funksionin **msgget()** me **IPC\_CREAT flag** për të mundësuar komunikimin me serverin.
2. Konektohet me serverin duke dërguar një kërkesë konektimi në message queue të tij: Klienti dërgon një mesazh kërkesë lidhjeje në radhën e mesazheve të serverit duke përdorur funksionin **msgsnd().**
3. Dërgon kërkesa te serveri duke dërguar mesazhe në message queue të tij: Klienti dërgon kërkesa te message queue e serverit duke përdorur funksionin **msgsnd().**
4. Pret përgjigje nga serveri duke dëgjuar message queue të tij: Klienti pret përgjigje nga serveri duke përdorur funksionin **msgrcv().**
5. Diskonektohet në mënyrë korrekte nga serveri duke dërguar një kërkesë diskonektimi në message queue të tij: Klienti dërgon një mesazh "*disconnect*" në message queue të serverit duke përdorur funksionin **msgsnd()** për t’u diskonektuar korrektësisht.

# **Përfundim**

Përfundimisht, detyra u zgjidh me sukses duke implementuar një sistem komunikimi klient-server duke përdorur message queues.

Duke përdorur message queues dhe multi-threading, kemi arritur një dizajn të qëndrueshëm dhe të shkallëzueshëm të aplikacionit, duke mundësuar një komunikim të sigurt dhe asinkron midis serverit dhe klientëve. Zgjidhja plotëson kërkesat e specifikuara dhe ofron një bazë të qëndrueshme për përmirësime dhe zgjerime të mëtejshme.